TensorFlow implementation

Inference in code

```
x = np.array([[200.0, 17.0]])
model = Sequential ([
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=1, activation='sigmoid'),
])
```

Data in Tensorflow

Matrix Dimensions

- A matrix is defined as **rows × columns**.
    - Example: [[1, 2, 3], [4, 5, 6]] → shape **(2, 3)**

NumPy Conventions

- `np.array([200, 17])` → shape **(2,)** → 1D vector (no rows/columns)
- `np.array([[200, 17]])` → shape **(1, 2)** → Row vector (1 row, 2 columns)
- `np.array([[200], [17]])` → shape **(2, 1)** → Column vector (2 rows, 1 column)

TensorFlow Conventions

- TensorFlow prefers **2D matrices** for all inputs.
- Input to a model = **(number of examples, number of features)**
- Example:

```
x = tf.constant([[200, 17]])
# shape: (1, 2)
```

- Even a single example must be a 2D tensor: $[[200, 17]]$

> 💡 **Why TensorFlow Uses 2D**
>
> - Optimized for batching and GPU/TPU execution.
> - 2D ensures consistent shape handling across datasets.
> - More computationally efficient than 1D arrays.

Row vs Column Vectors

- Row vector:$[[200, 17]]$ → shape $(1, 2)$
- Column vector: $[[200], [17]]$ → shape $(2, 1)$

Tensor vs NumPy Array

- Conversion:

```
tensor.numpy()  # TensorFlow → NumPy
tf.constant(np_array)  # NumPy → TensorFlow
```

- **Tensor (TensorFlow)**: n-D array optimized for ML computations.
- **NumPy Array**: general-purpose n-D array used in scientific computing.

Building a Neural Network

Architecture

Sequential function in TensorFlow, which allows for easier construction of neural networks by stringing together layers.

- Emphasizes that instead of manually passing data through layers, TensorFlow can automate this process.

```
layer_1 = Dense(units = 3, activation='sigmoid')
layer_2 = Dense(units = 1, activation='sigmoid')
model = Sequential([layer_1, layer_2)
```

Input:

```
x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
```

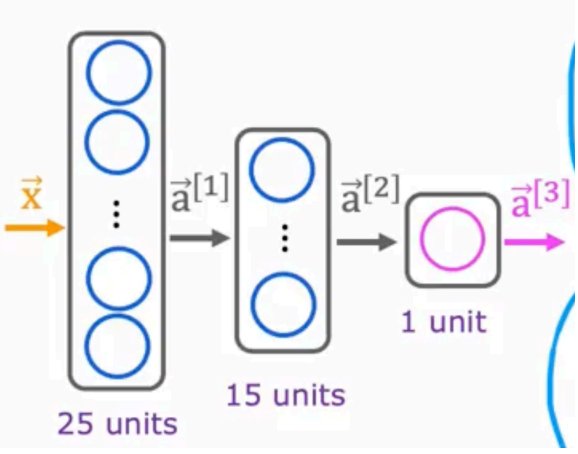
Target:

```
y = np.array([1, 0, 0, 1])
```

```
model.compile(...)
model.fit(x, y)    # → take this neural network that are created by sequential ly string together layers one and two, and to train it on the data, X and Y

model.predict(x_new)
```

Example: Digit Classification model




```
layer_1 = Dense(units=25, activation='sigmoid')
layer_2 = Dense(units=15, activation='sigmoid')
layer_3 = Dense(units=1, activation='sigmoid')
```

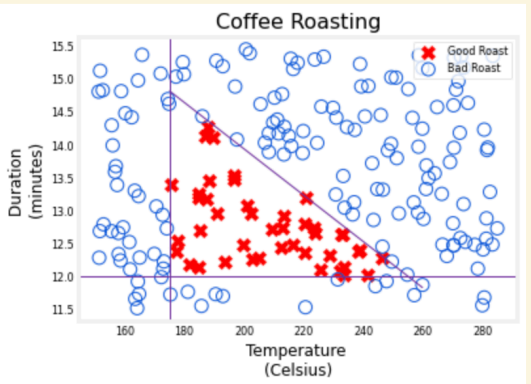
```
model = Sequential ([layer_1, layer_2, layer_3])

model.compile(...)
x = np.array([[0,..., 245,..., 17],
                        [0,..., 200,..., 184]])
y = np.array([1, 0])
model.fit(x, y)
model_predict(x_new)
```

Coding: Simple Neural Network

Coffee Roasting

- The problem illustrates how temperature and duration affect coffee quality, with a dataset indicating good and bad coffee.
- Good coffee is produced within a specific range of temperature and duration, while extremes lead to undercooked or burnt beans.
- Visualize the dataset of the problem.

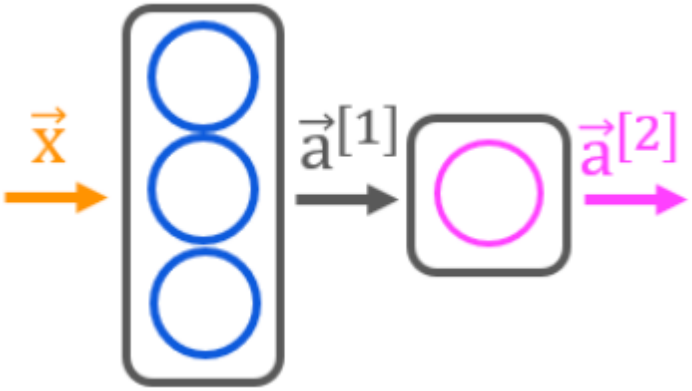


Normalize data

This is the same procedure you used in Course 1 where features in the data are each normalized to have a similar range.

```
norm_l = tf.keras.layers.Normalization(axis=-1)
norm_l.adapt(X)  # learns mean, variance
Xn = norm_l(X)
```

Model




```python
tf.random.set_seed(1234)  # applied to achieve consistent results
model = Sequential(
    [
        tf.keras.Input(shape=(2,)),
        Dense(3, activation='sigmoid', name = 'layer1'),
        Dense(1, activation='sigmoid', name = 'layer2')
    ]
)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 3)                 9

 layer2 (Dense)              (None, 1)                 4

=================================================================
Total params: 13
Trainable params: 13
Non-trainable params: 0
_________________________________________________________________
```

With `model.summary()` we can see that:

Explain:

Layer	Output Shape	Param
Layer 1	$(\text{None}, 3)$	$2 \text{ input} * 3 \text{ neurons} + 3 \text{ biases} = 9$
Layer 2	$(\text{None}, 1)$	$3 \text{ input} * 1 \text{ neurons} + 1 \text{ biases} = 4$

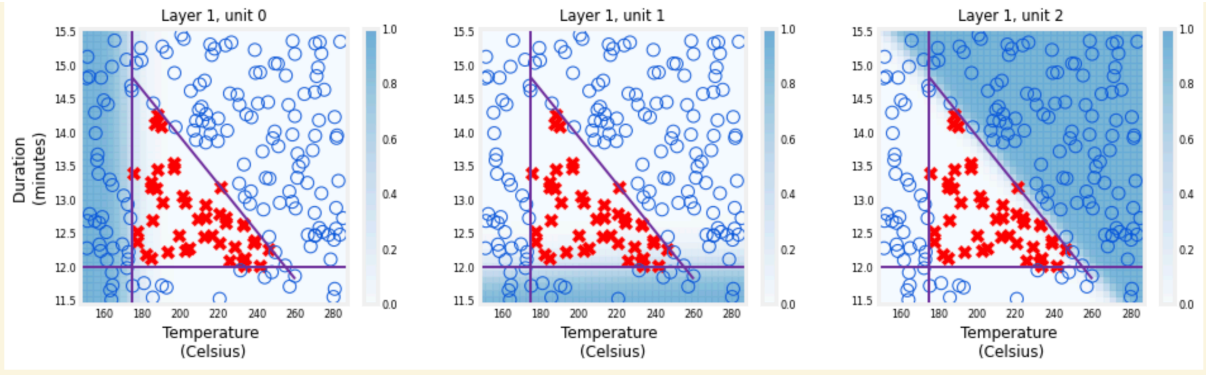
Training the model

```
model.compile(
    loss = tf.keras.losses.BinaryCrossentropy(),
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.01),
)

model.fit(
    Xt,Yt,
    epochs=10,
)
```

Visualization

Plot the output of each node for all values of the inputs `(duration,temp)`. Each unit is a logistic function whose output can range from zero to one. The shading in the graph represents the output value. It is worth noting that the network learned these functions on its own through the process of gradient descent.



In the shading plot for the neural network units, a value near 1 indicates **dark shading** because it represents a high output from the unit in the first layer of the neural network. Here's why:

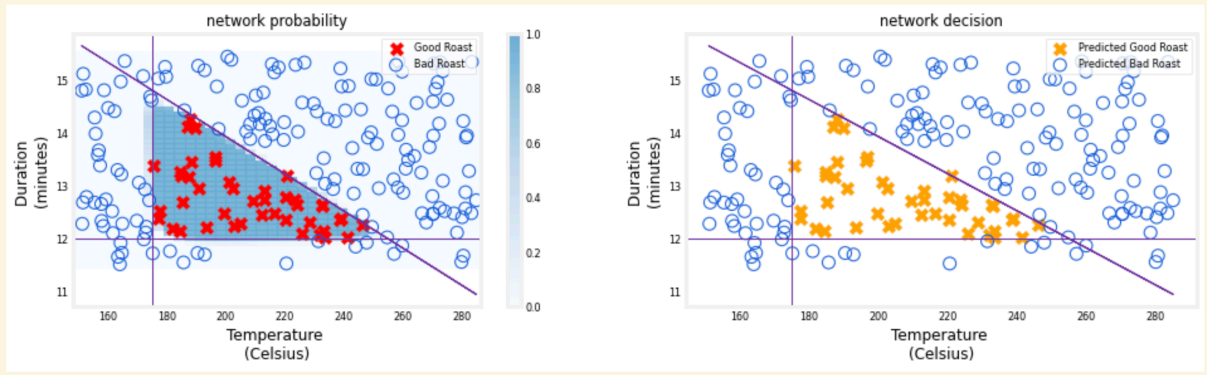
- **Shading Representation**: The shading in the plot shows the output value of each unit, where **darker shading** corresponds to higher output values (close to 1) and **lighter shading** corresponds to lower output values (close to 0).
- **Unit Activation**: In this context, the neural network is analyzing coffee roasting conditions (like temperature and duration). A value near 1 means the unit is **strongly activated**, indicating it's highly confident that the input conditions lead to a "bad roast."
- **Visualization Choice**: Using dark shading for values near 1 is a common technique in data visualization (like heatmaps) to **highlight areas of strong activity or importance**. It makes it easier to spot regions where the unit detects problematic roasting conditions.

The final graph

shows the whole network in action.

- The left graph is the raw output of the final layer represented by the blue shading. This is overlaid on the training data represented by the X's and O's.

- The right graph is the output of the network after a decision threshold. The X's and O's here correspond to decisions made by the network.



- **Units 0, 1, and 2 in the first layer**: Their outputs are high when specific bad roast conditions are detected (low temperature, short duration, or bad combinations). These are not probabilities of a bad roast per se, but rather signals of features that lead to a bad roast.
- **Layer 2 (output layer)**: It takes these signals and computes the probability of a good roast. When the first layer units output high values (indicating bad conditions), the output layer produces a low probability of a good roast, effectively classifying the roast as bad. Conversely, when the first layer outputs are low (no bad conditions), the output layer produces a high probability of a good roast.