Neurons (With and Without Activation)

	Neurons without activation (Linear Model)	Neurons with activation (Logistic Model)
Functionality	Implements a linear function: `f_w,b(x(i)) = w.x(i) + b`	Implements a sigmoid function: `f_w,b(x(i)) = sigmoid(w.x(i) + b)`
Output Range	Can output any real number: `(-∞, +∞)`	Outputs a value between 0 and 1: `(0, 1)`
Example	Given `X_train[0] = 1`, weights `w = 200` and bias `b = 100`. The output `f_w,b(X_train[0]) = 200*1 + 100 = 300`	Given `X_train[0] = 1`, weights `w = 2` and bias `b = -4.5`. The output `f_w,b(X_train[0]) = sigmoid(2*1 - 4.5) = 0.01`
Common Use Cases	Used when the output is expected to be continuous. Example: regression problems, predicting house prices	Used when the output is expected to be within a certain range. Example: binary classification problems, predicting if an email is spam (1) or not spam (0)
Training	Trained using methods like least squares or gradient descent	Trained using methods like logistic regression or cross-entropy loss
Advantages	Simple to understand, interpret, and compute	Can model non-linear relationships, better for classifying inputs

Th primary difference lies in the activation function, which introduces non-linearity, allowing for more complex models to be constructed.

Creating a Model without Activation (Linear Model)

1. **Import the necessary libraries**:

   ```
   import numpy as np import tensorflow as tf
   ```

2. **Define the model**: Use the `tf.keras.layers.Dense` class to define a linear model. The `units` parameter sets the number of output neurons (1 in this case) and `activation` is set to 'linear'.

   ```
   linear_layer = tf.keras.layers.Dense(units=1, activation='linear')
   ```

3. **Initialize the model's weights and bias**: Feed the model some input data to allow it to initialize its weights and biases. Note that your input data must be two-dimensional.

   ```
   linear_layer.initialize_weights(X_train[0].reshape(1,1))
   ```

4. **Set the model's weights and bias**: After initialization, you can set your own values for the weights and bias.

```
set_w = np.array([[200]]) set_b = np.array([100]) linear_layer.set_weights([set_w, set_b])
```

5. **Use the model for prediction**:

```
prediction = linear_layer.predict(X_test.reshape(1,1))
```

Creating a Model with Activation (Logistic Model)

1. **Import the necessary libraries**:

```
import numpy as np import tensorflow as tf from tensorflow.keras.models import Sequential
```

2. **Define the model**: Use the `tf.keras.layers.Dense` class to define a logistic model. This time, set `activation` to 'sigmoid'.

```
model = Sequential([tf.keras.layers.Dense(1, input_dim=1, activation='sigmoid', name='L1')])
```

3. **Initialize the model's weights and bias**: Feed the model some input data to allow it to initialize its weights and biases. Note that your input data must be two-dimensional.

```
model.initialize_weights(X_train[0].reshape(1,1))
```

4. **Set the model's weights and bias**: After initialization, you can set your own values for the weights and bias.

```
set_w = np.array([[2]]) set_b = np.array([-4.5])
model.get_layer('L1').set_weights([set_w, set_b])
```

5. **Use the model for prediction**:

```
prediction = model.predict(X_test.reshape(1,1))
```

In both cases, `X_train` and `X_test` are your training and testing datasets, respectively. These steps give you the basic idea of how to create a model with and without activation functions.