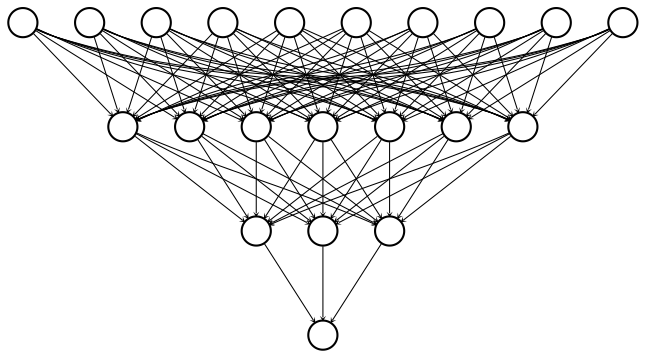
**Using the dense layer operation**

We've now seen how to define dense layers in tensorflow using linear algebra. In this exercise, we'll skip the linear algebra and let keras work out the details. This will allow us to construct the network below, which has 2 hidden layers and 10 features, using less code than we needed for the network with 1 hidden layer and 3 features.



To construct this network, we'll need to define three dense layers, each of which takes the previous layer as an input, multiplies it by weights, and applies an activation function. Note that input data has been defined and is available as a 100x10 tensor: borrower\_features. Additionally, the keras.layersmodule is available.

**Instructions**

**100 XP**

* Set dense1 to be a dense layer with 7 output nodes and a sigmoid activation function.
* Define dense2 to be dense layer with 3 output nodes and a sigmoid activation function.
* Define predictions to be a dense layer with 1 output node and a sigmoid activation function.
* Print the shapes of dense1, dense2, and predictions in that order using the .shape method. Why does each of these tensors have 100 rows?

# Define the first dense layer

dense1 = keras.layers.Dense(7, activation='sigmoid')(borrower\_features)

# Define a dense layer with 3 output nodes

dense2 = keras.layers.Dense(3, activation='sigmoid')(dense1)

# Define a dense layer with 1 output node

predictions = keras.layers.Dense(1, activation='sigmoid')(dense2)

# Print the shapes of dense1, dense2, and predictions

print('\n shape of dense1: ', dense1.shape)

print('\n shape of dense2: ', dense2.shape)

print('\n shape of predictions: ', predictions.shape)

Great work! With just 8 lines of code, you were able to define 2 dense hidden layers and an output layer. This is the advantage of using high-level operations in tensorflow. Note that each layer has 100 rows because the input data contains 100 examples.