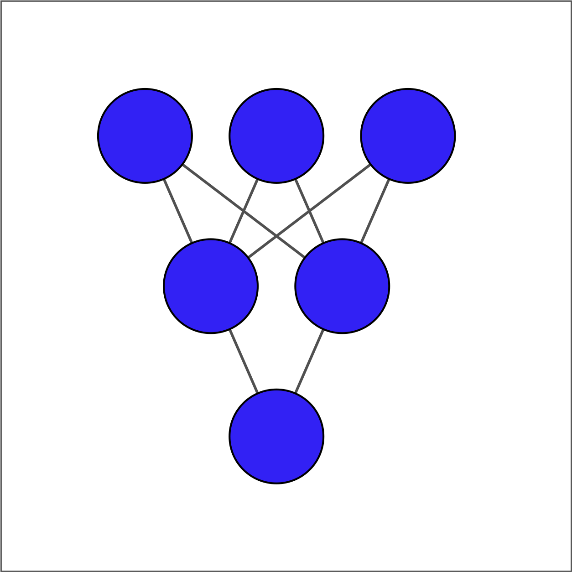
**The linear algebra of dense layers**

There are two ways to define a dense layer in tensorflow. The first involves the use of low-level, linear algebraic operations. The second makes use of high-level keras operations. In this exercise, we will use the first method to construct the network shown in the image below.



The input layer contains 3 features -- education, marital status, and age -- which are available as borrower\_features. The hidden layer contains 2 nodes and the output layer contains a single node.

For each layer, you will take the previous layer as an input, initialize a set of weights, compute the product of the inputs and weights, and then apply an activation function. Note that Variable(), ones(), matmul(), and keras() have been imported from tensorflow.

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Initialize weights1 as a variable using a 3x2 tensor of ones.
* Compute the product of borrower\_features by weights1 using matrix multiplication.
* Use a sigmoid activation function to transform product1 + bias1.

# Initialize bias1

bias1 = Variable(1.0)

# Initialize weights1 as 3x2 variable of ones

weights1 = Variable(ones((3, 2)))

# Perform matrix multiplication of borrower\_features and weights1

product1 = matmul(borrower\_features, weights1)

# Apply sigmoid activation function to product1 + bias1

dense1 = keras.activations.sigmoid(product1 + bias1)

# Print shape of dense1

print("\n dense1's output shape: {}".format(dense1.shape))

* Initialize weights2 as a variable using a 2x1 tensor of ones.
* Compute the product of dense1 by weights2 using matrix multiplication.
* Use a sigmoid activation function to transform product2 + bias2.

# From previous step

bias1 = Variable(1.0)

weights1 = Variable(ones((3, 2)))

product1 = matmul(borrower\_features, weights1)

dense1 = keras.activations.sigmoid(product1 + bias1)

# Initialize bias2 and weights2

bias2 = Variable(1.0)

weights2 = Variable(ones((2, 1)))

# Perform matrix multiplication of dense1 and weights2

product2 = matmul(dense1, weights2)

# Apply activation to product2 + bias2 and print the prediction

prediction = keras.activations.sigmoid(product2 + bias2)

print('\n prediction: {}'.format(prediction.numpy()[0,0]))

print('\n actual: 1')

Excellent work! Our model produces predicted values in the interval between 0 and 1. For the example we considered, the actual value was 1 and the predicted value was a probability between 0 and 1. This, of course, is not meaningful, since we have not yet trained our model's parameters.