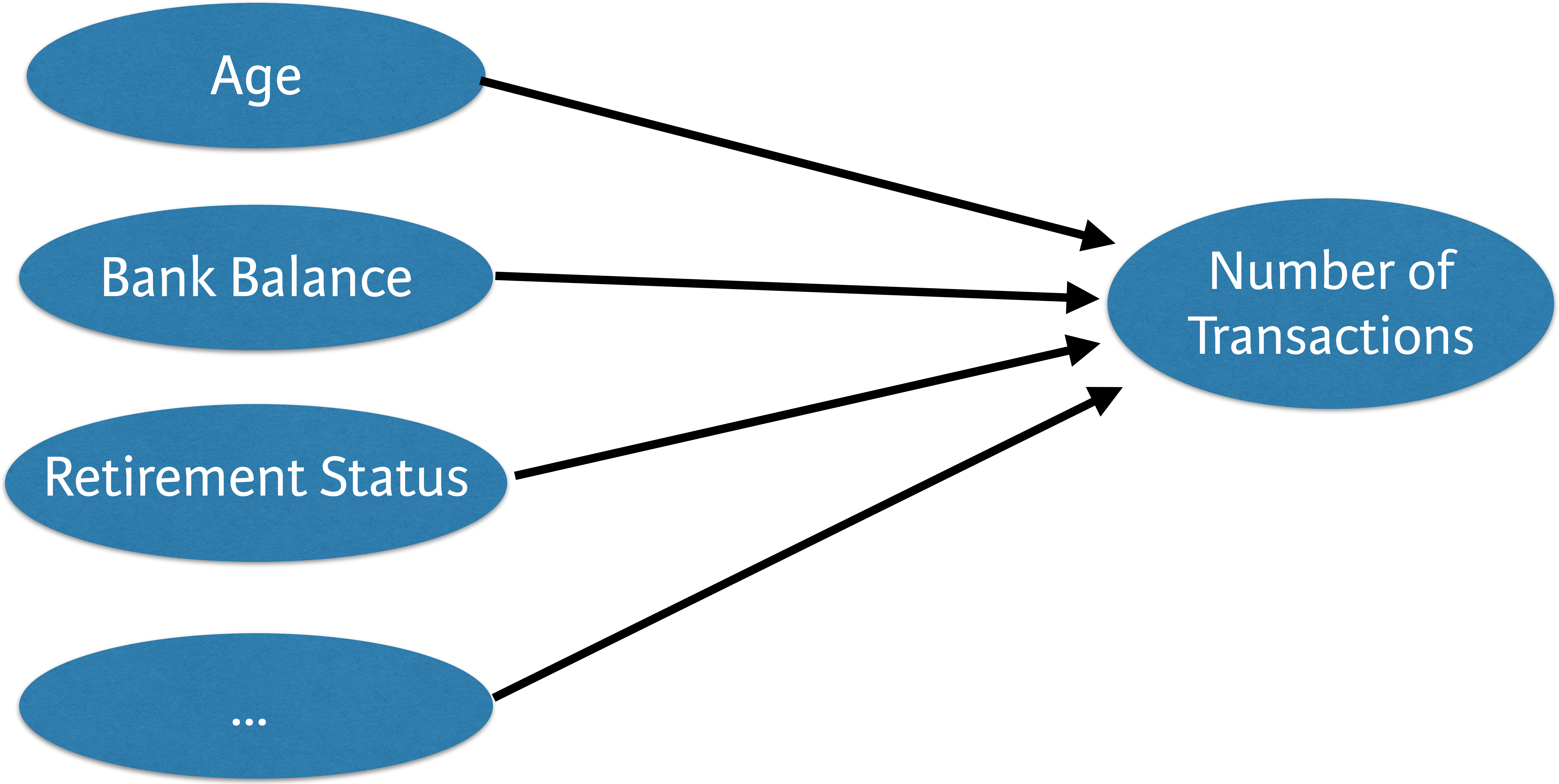
Introduction to deep learning

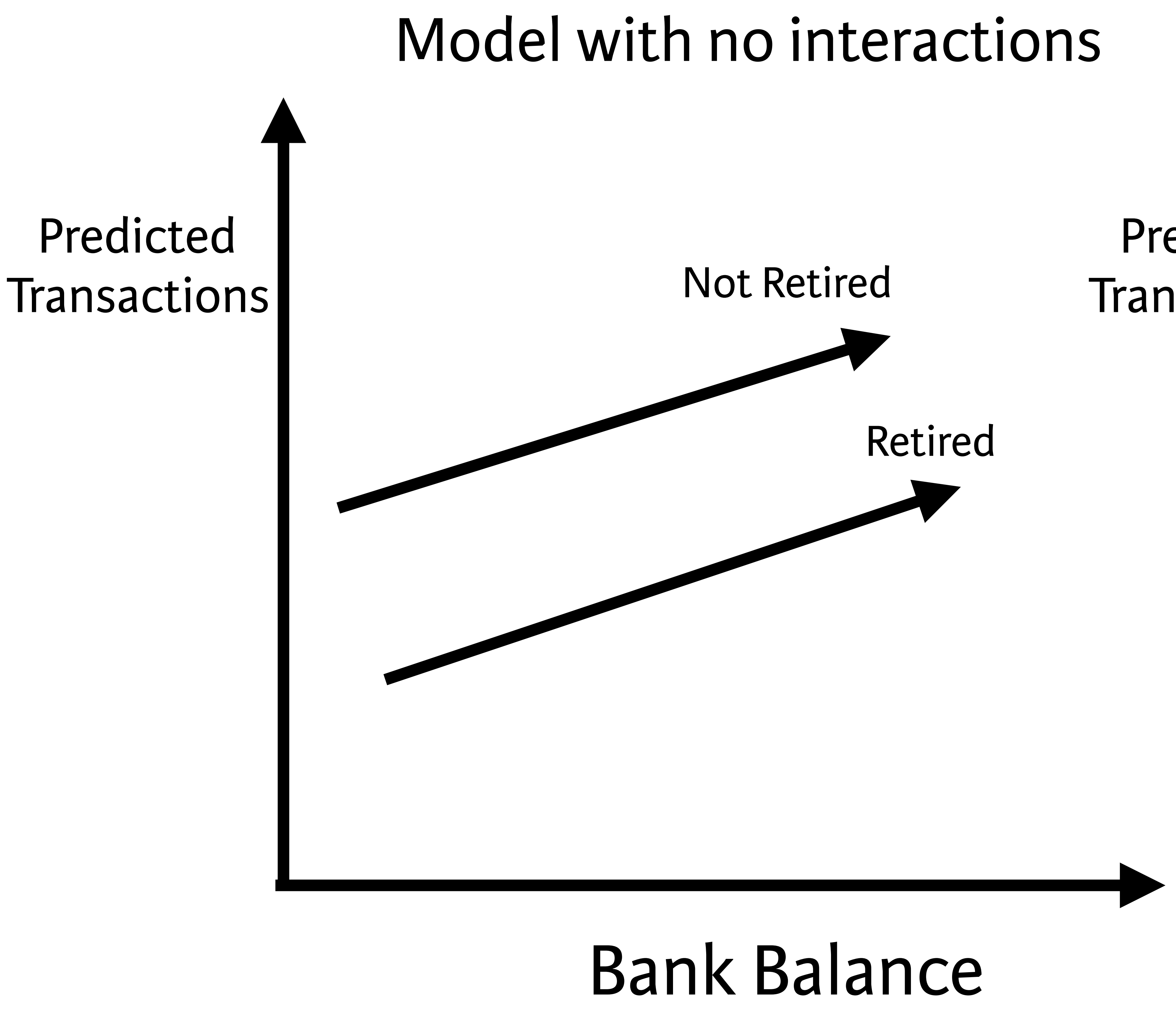
Imagine you work for a bank

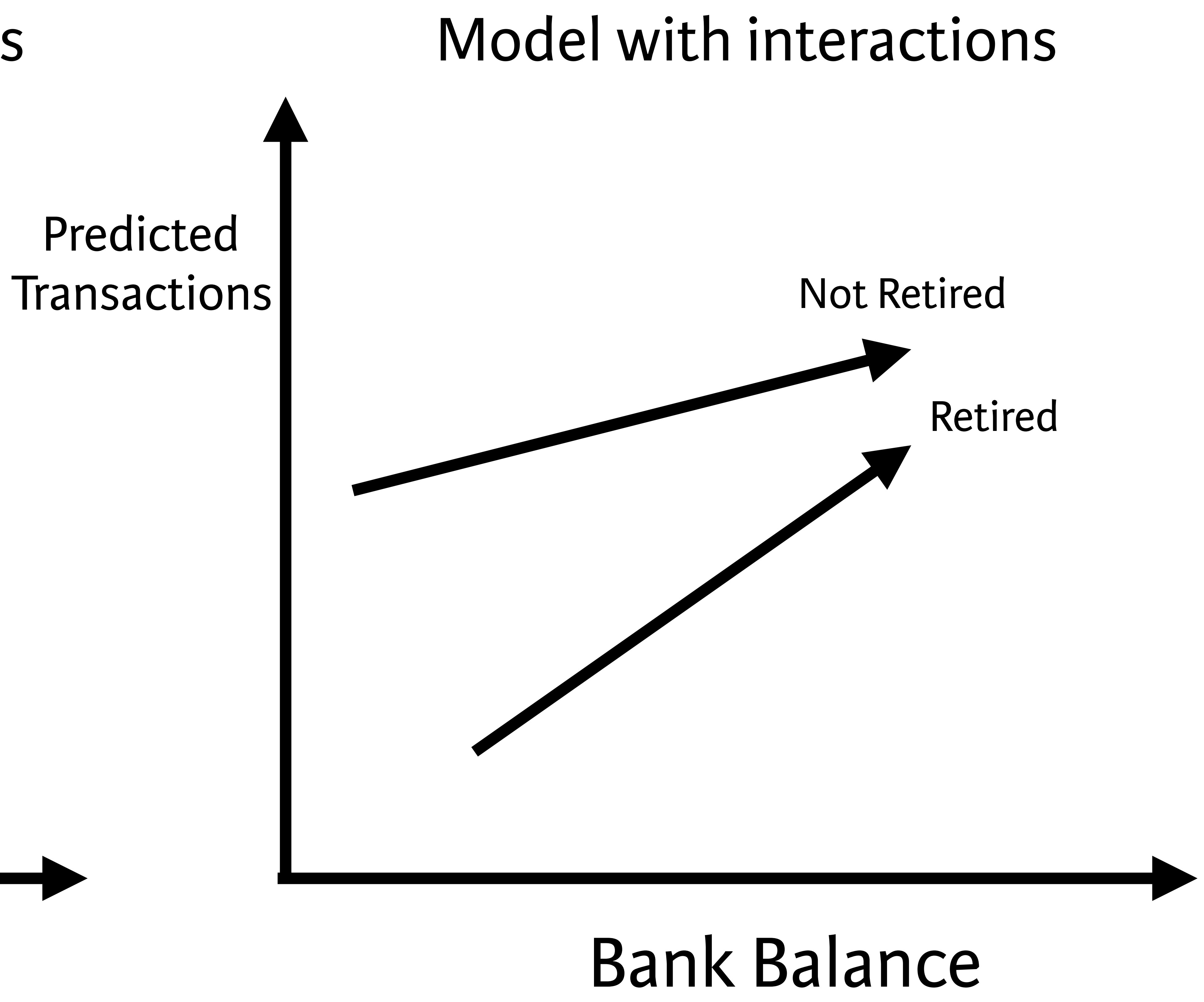
- You need to predict how many transactions each customer will make next year

Example as seen by linear regression



Example as seen by linear regression





Interactions

- Neural networks account for interactions really well
- Deep learning uses especially powerful neural networks
  - Text
  - Images
  - Videos
  - Audio
  - Source code

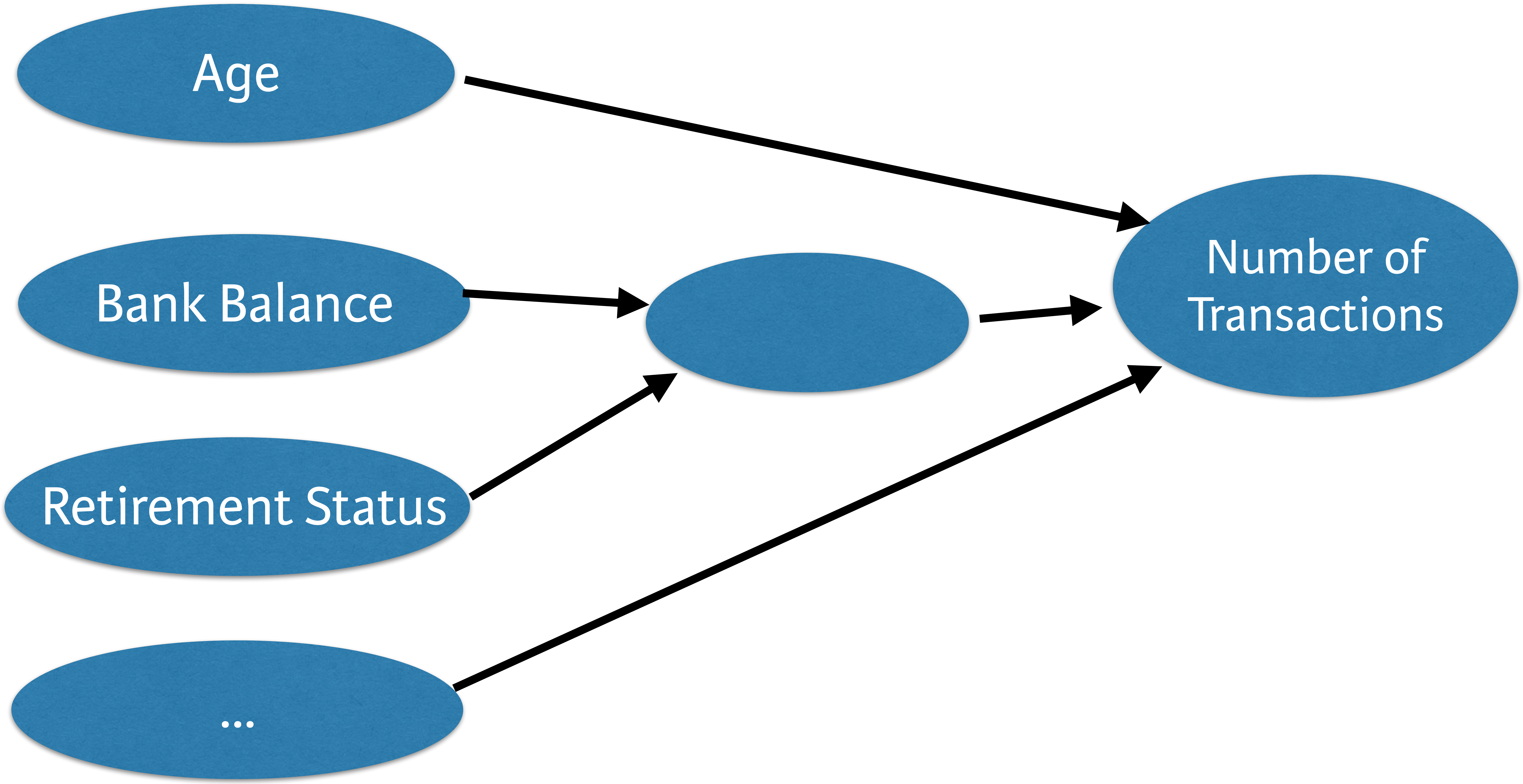
Course structure

- First two chapters focus on conceptual knowledge
  - Debug and tune deep learning models on conventional prediction problems
  - Lay the foundation for progressing towards modern applications
- This will pay off in the third and fourth chapters

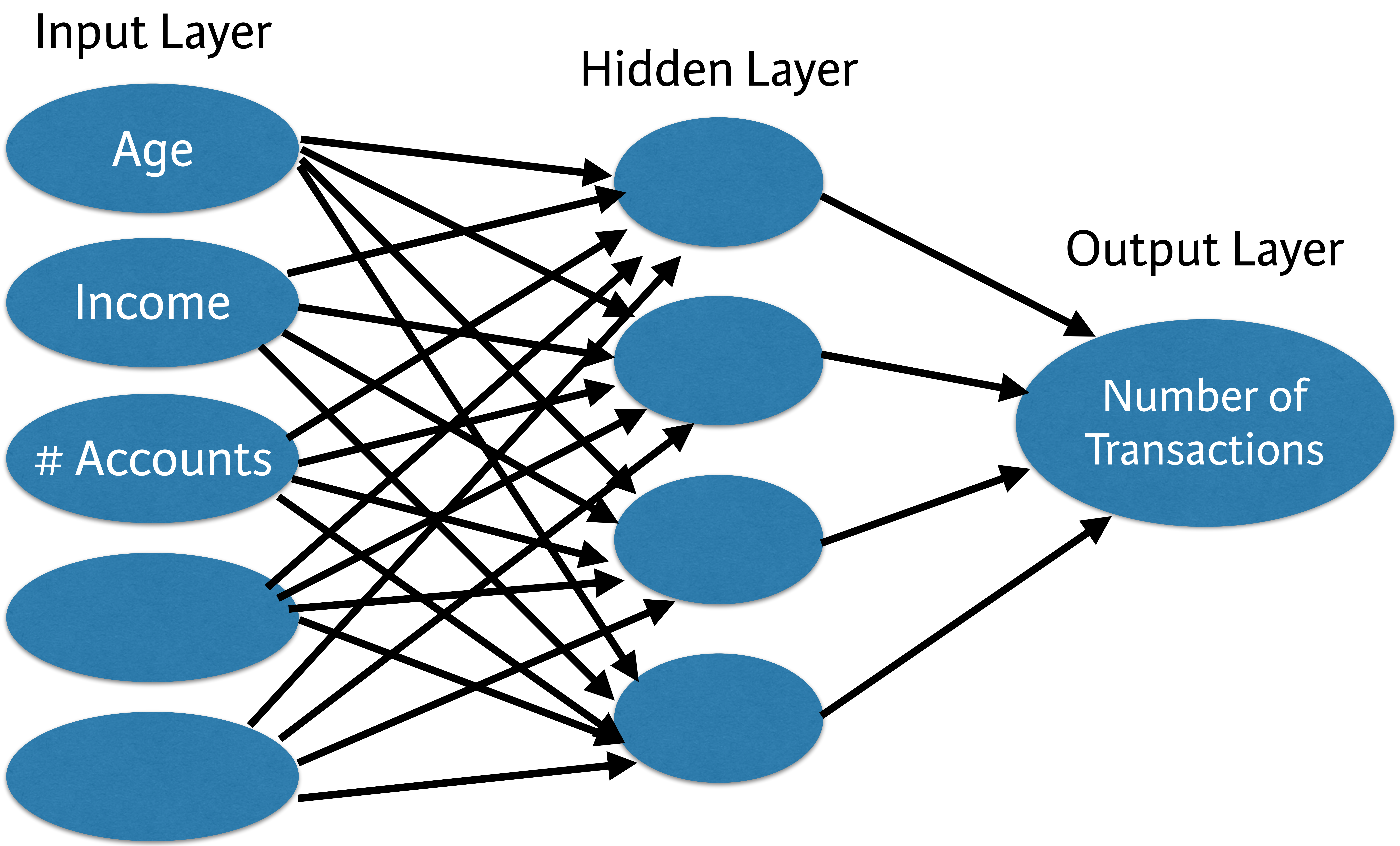
Build deep learning models with keras

```
In [1]: import numpy as np

In [2]: from keras.layers import Dense

In [3]: from keras.models import Sequential

In [4]: predictors = np.loadtxt('predictors_data.csv', delimiter=',')

In [5]: n_cols = predictors.shape[1]

In [6]: model = Sequential()

In [7]: model.add(Dense(100, activation='relu', input_shape = (n_cols,)))

In [8]: model.add(Dense(100, activation='relu'))

In [9]: model.add(Dense(1))
```

Deep learning models capture interactions



Interactions in neural network



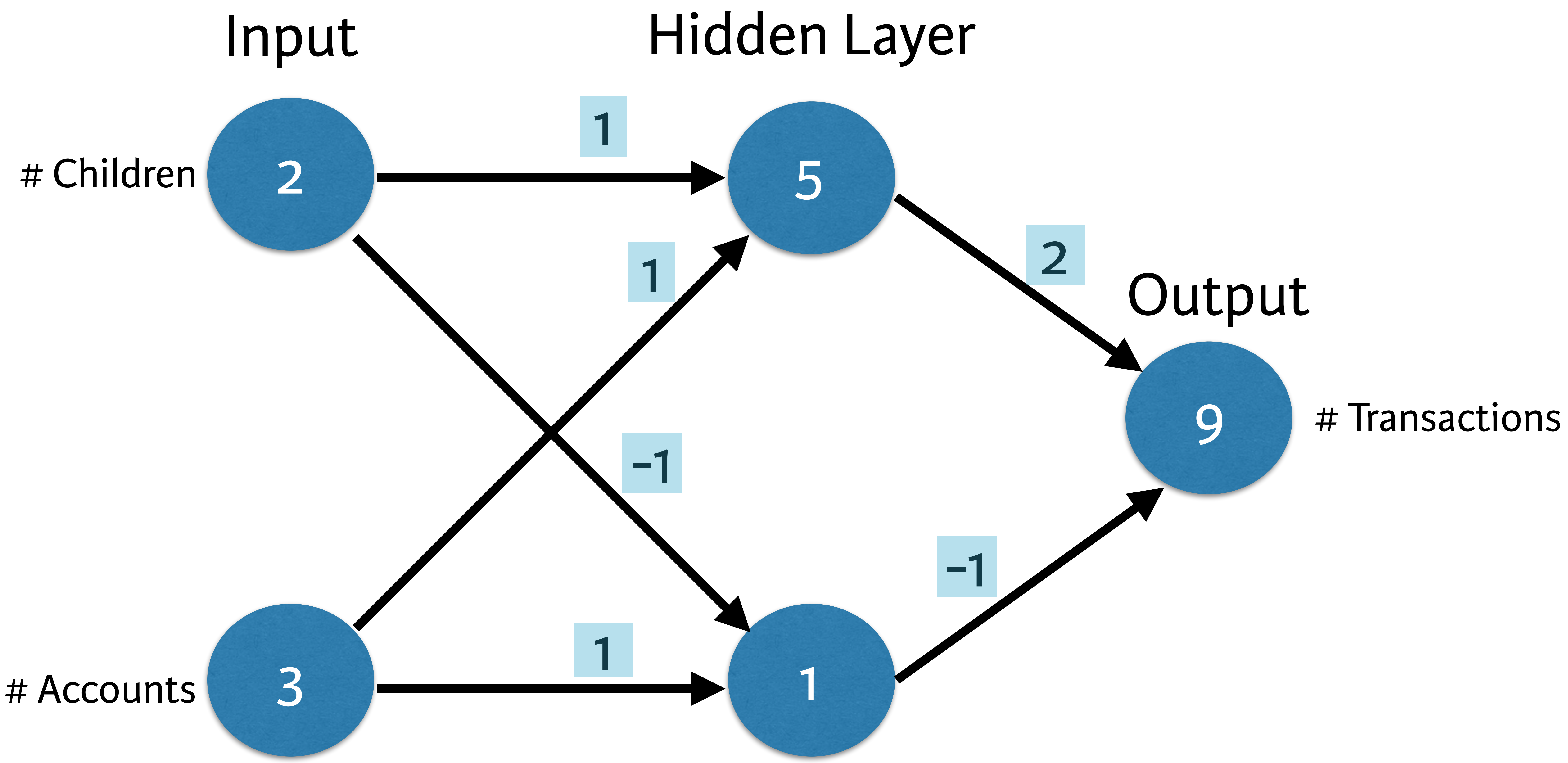
Let’s practice!

Forward propagation

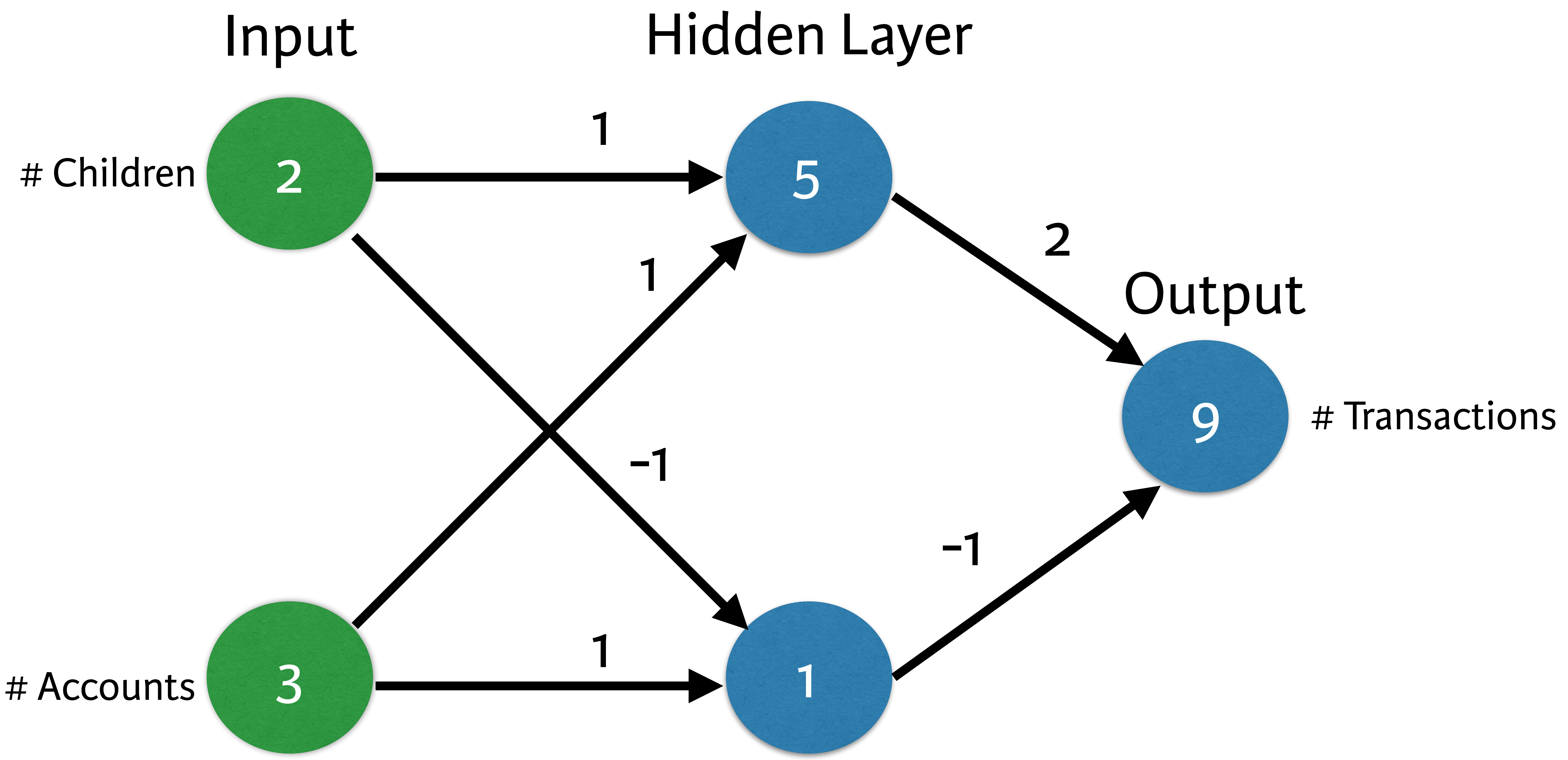
Bank transactions example

- Make predictions based on:
  - Number of children
  - Number of existing accounts

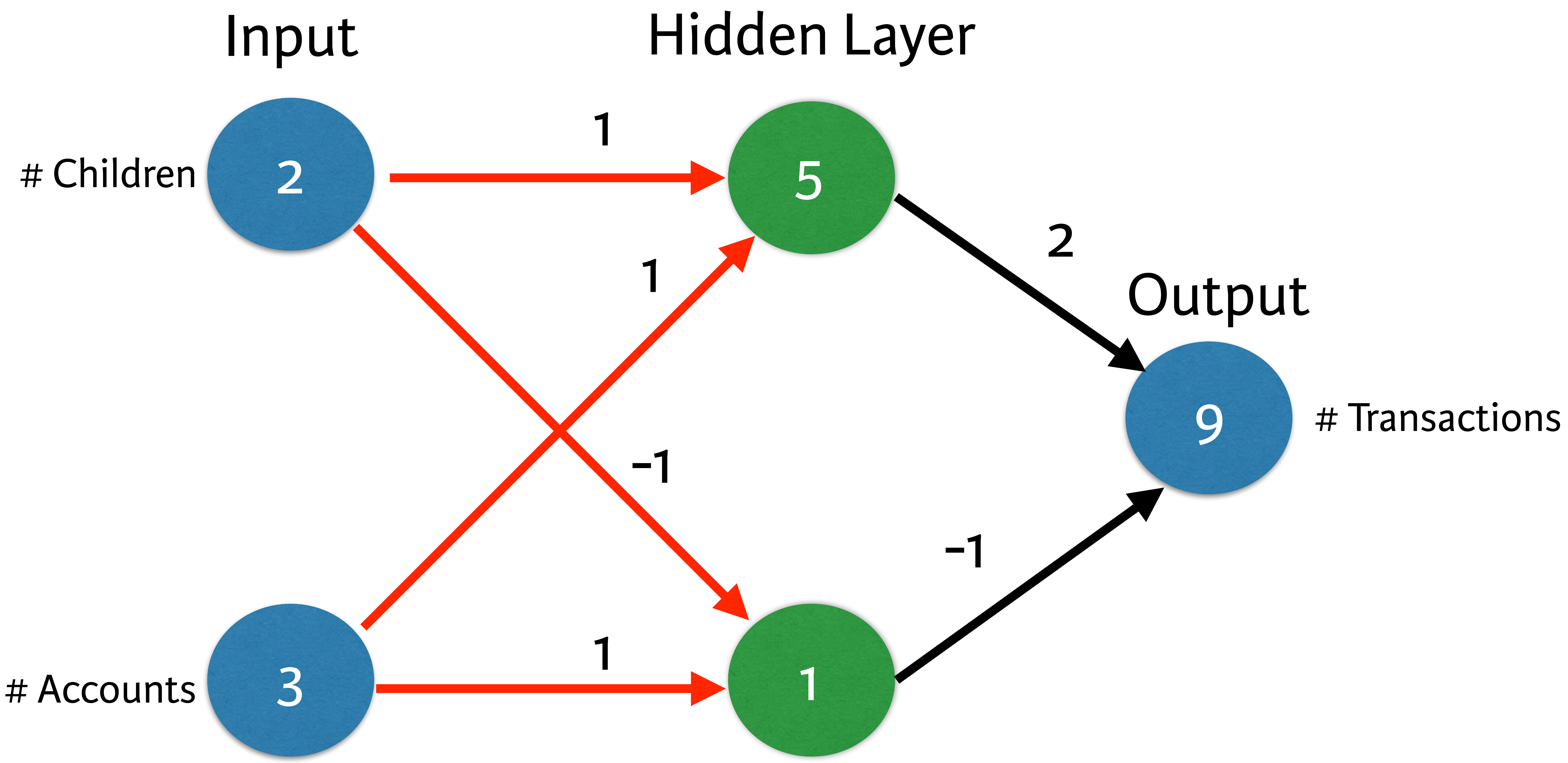
Forward propagation



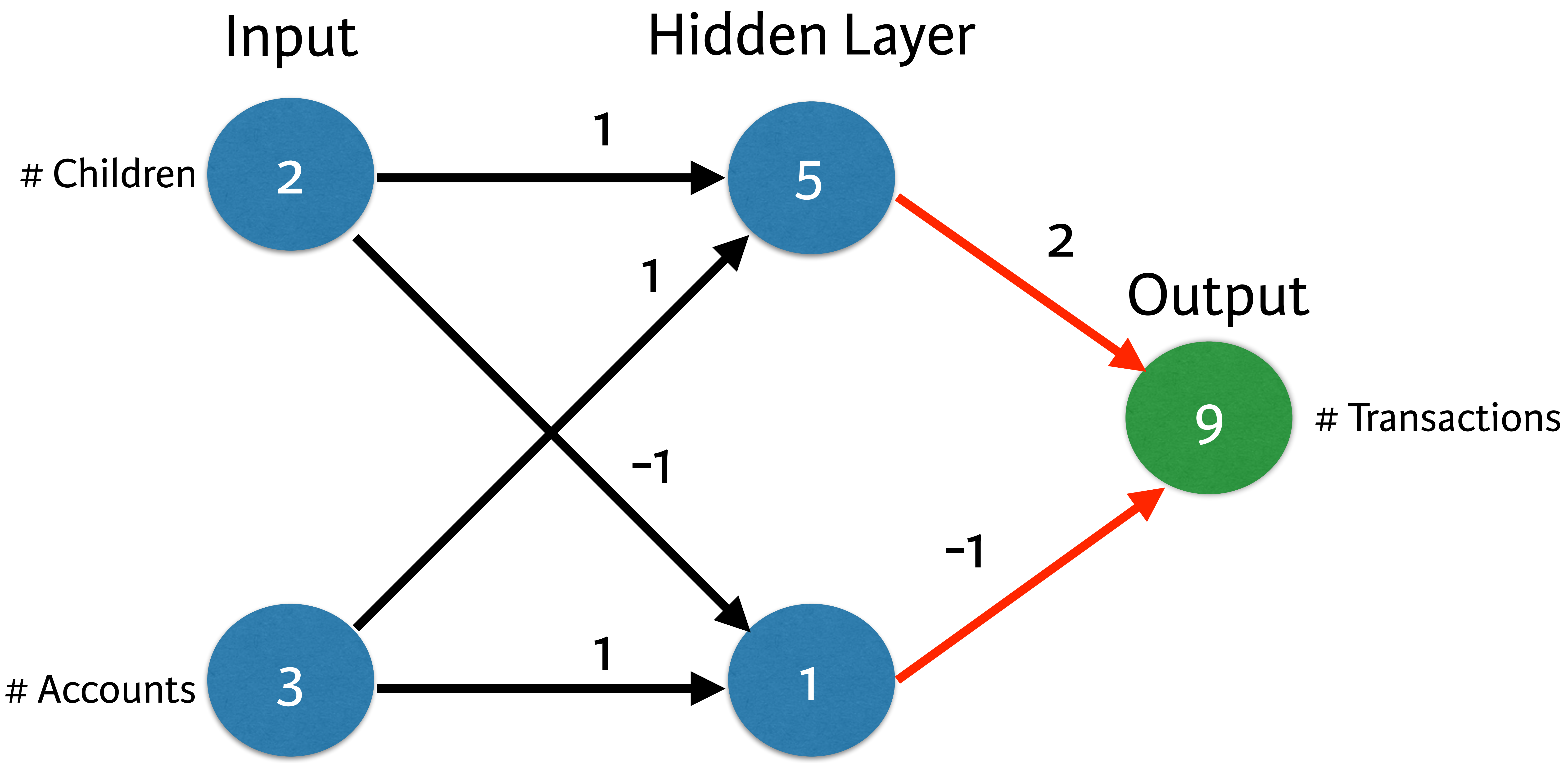
Forward propagation



Forward propagation



Forward propagation

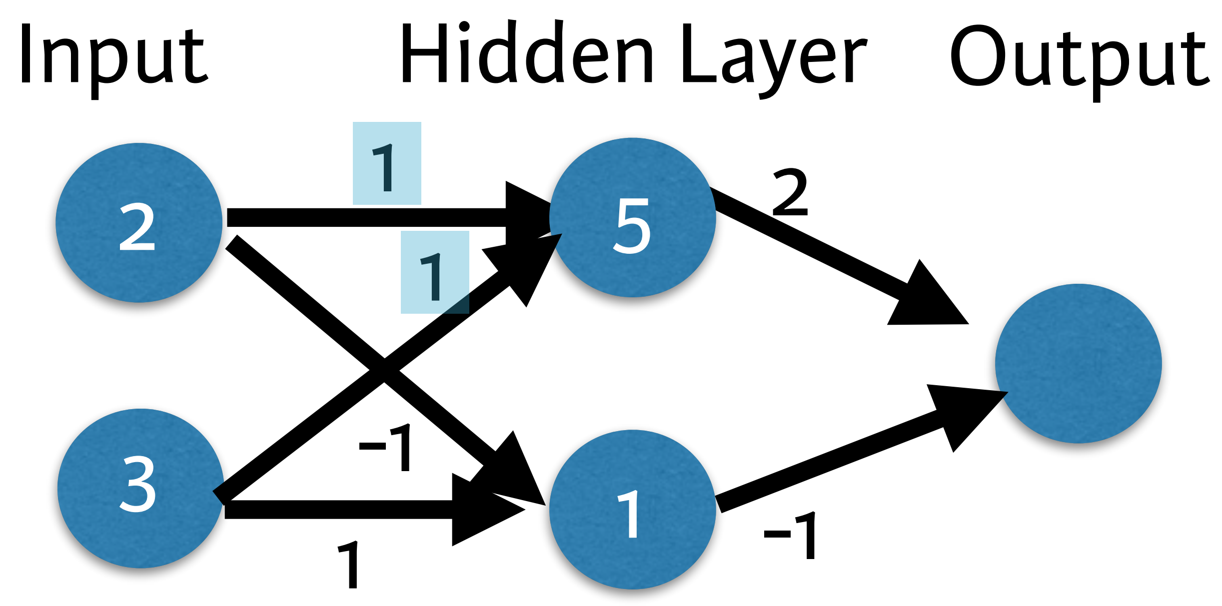


Forward propagation

- Multiply - add process
- Dot product
- Forward propagation for one data point at a time
- Output is the prediction for that data point

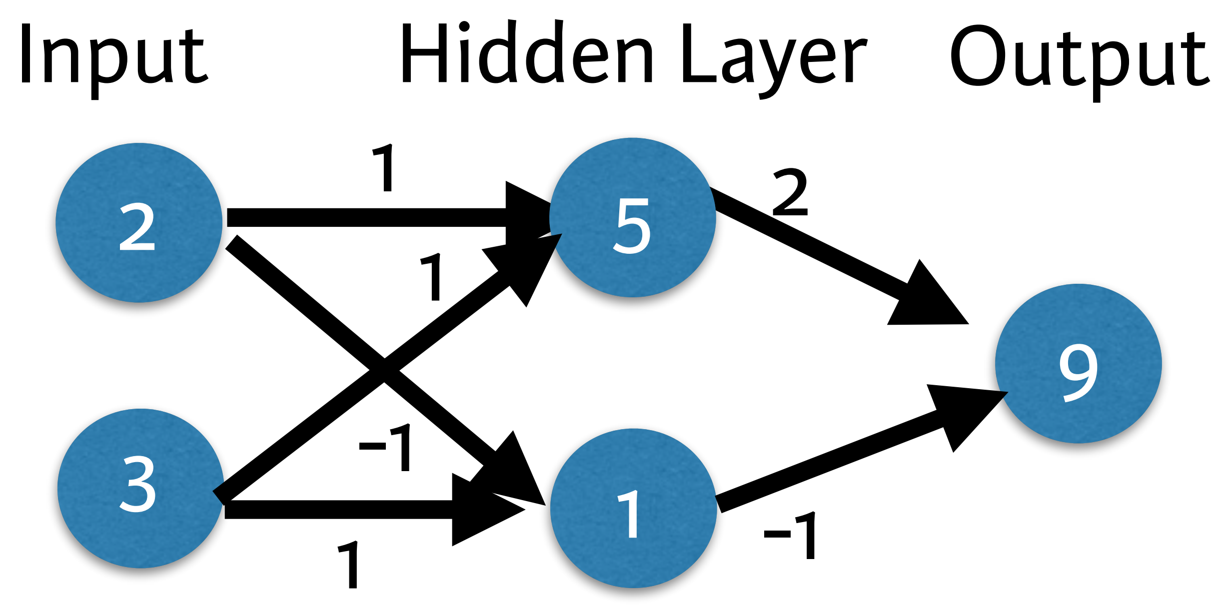
Forward propagation code

```
In [1]: import numpy as np

In [2]: input_data = np.array([2, 3])

In [3]: weights = {'node_0': np.array([1, 1]),
   ...:            'node_1': np.array([-1, 1]),
   ...:            'output': np.array([2, -1])}

In [4]: node_0_value = (input_data * weights['node_0']).sum()

In [5]: node_1_value = (input_data * weights['node_1']).sum()
```



Forward propagation code

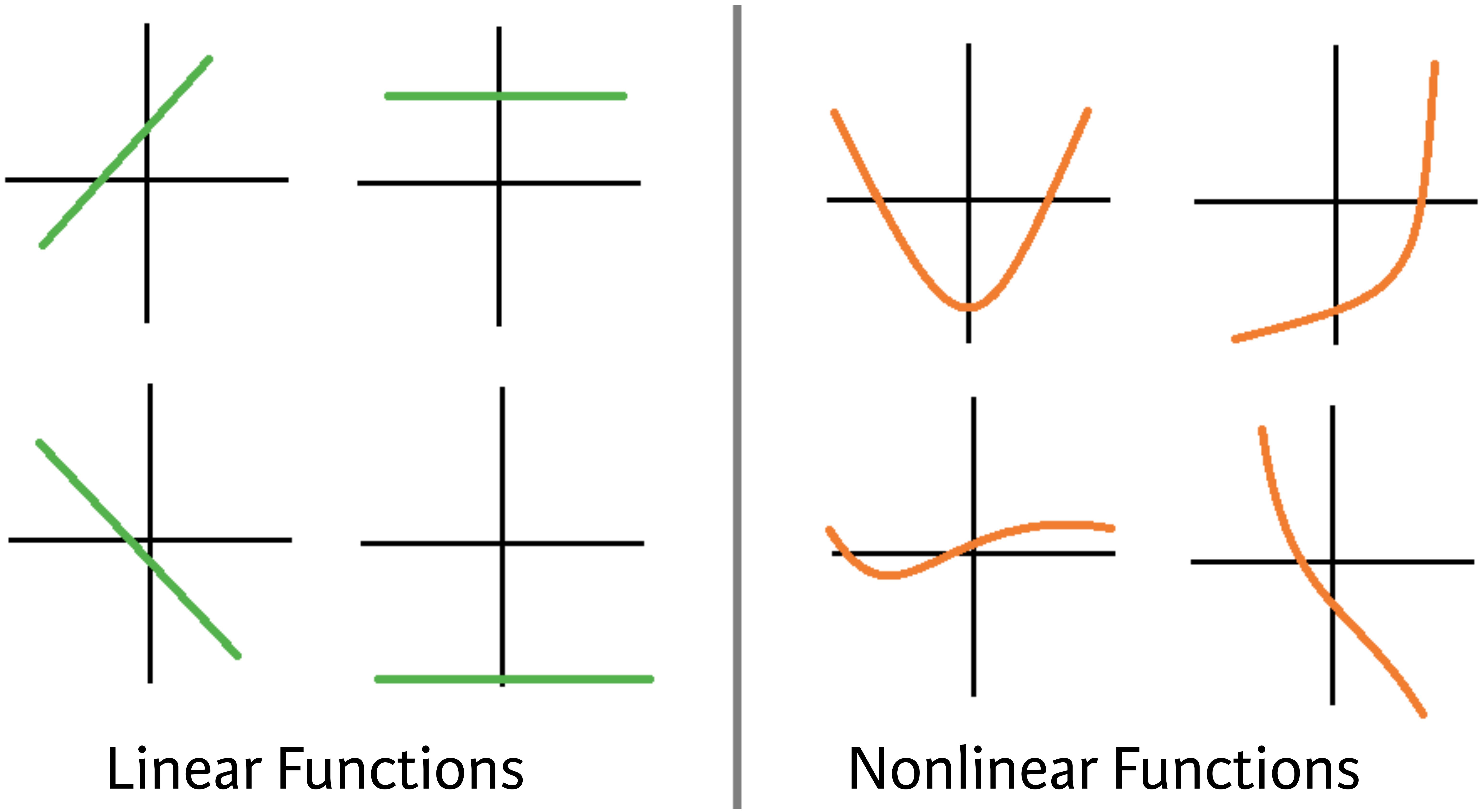
```
In [6]: hidden_layer_values = np.array([node_0_value, node_1_value])

In [7]: print(hidden_layer_values)
[5, 1]

In [8]: output = (hidden_layer_values * weights['output']).sum()

In [9]: print(output)
9
```



Let’s practice!

Activation functions

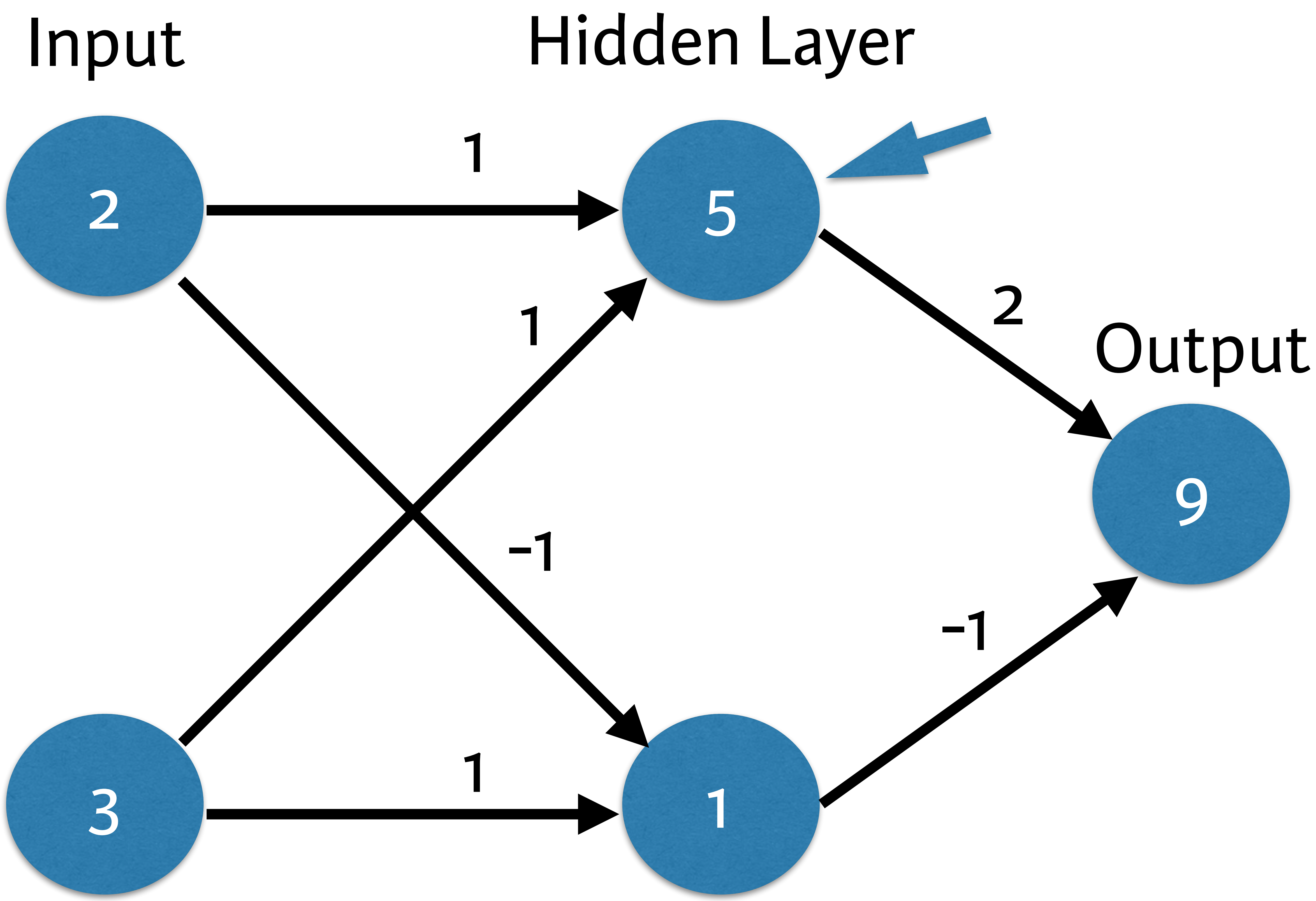
Linear vs Nonlinear Functions



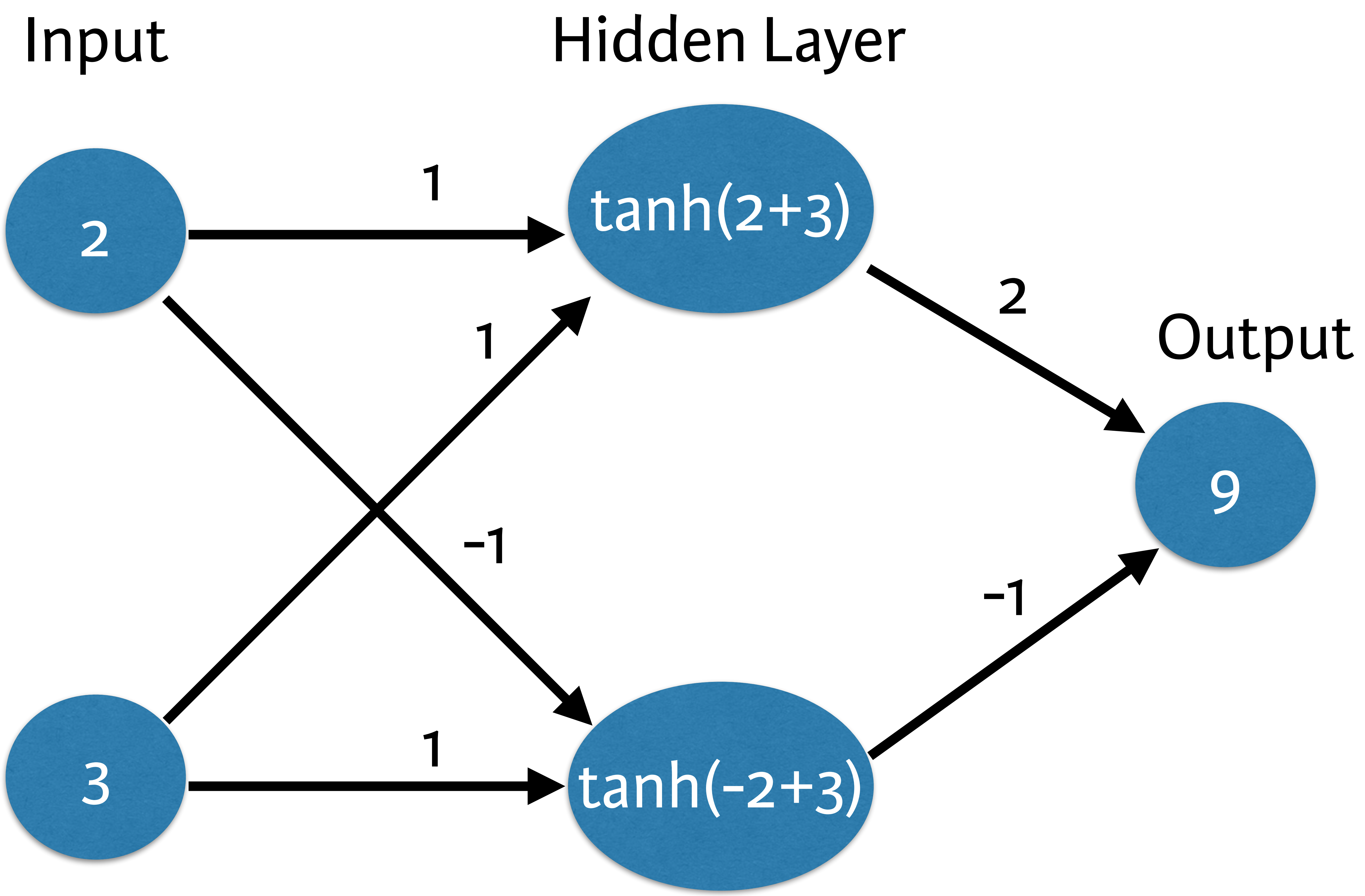
Activation functions

- Applied to node inputs to produce node output

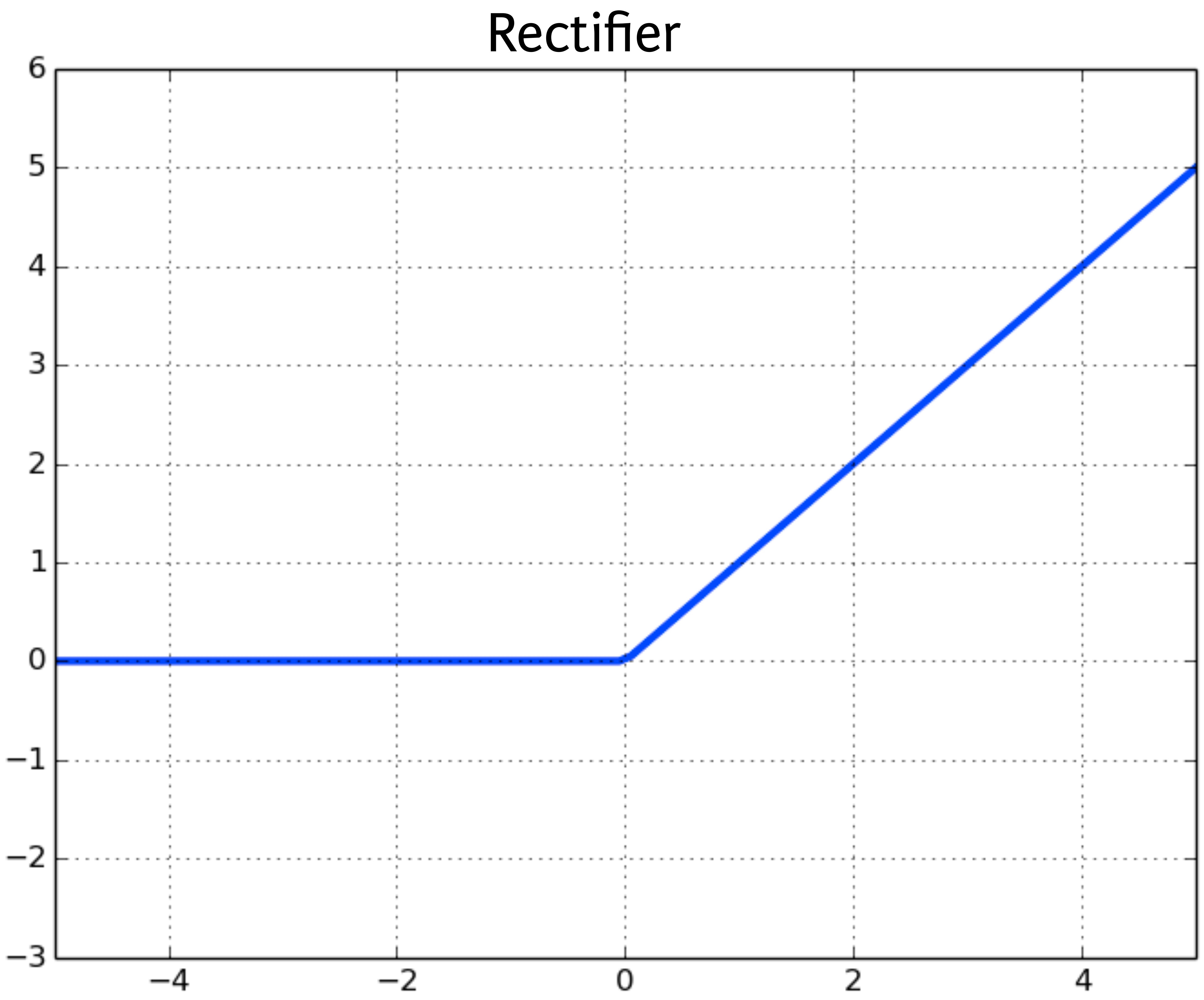
Improving our neural network



Activation functions



ReLU (Rectified Linear Activation)



$$RELU(x) = \begin{cases} 0 \ if \ x < 0 \\ x \ if \ x >= 0 \end{cases}$$

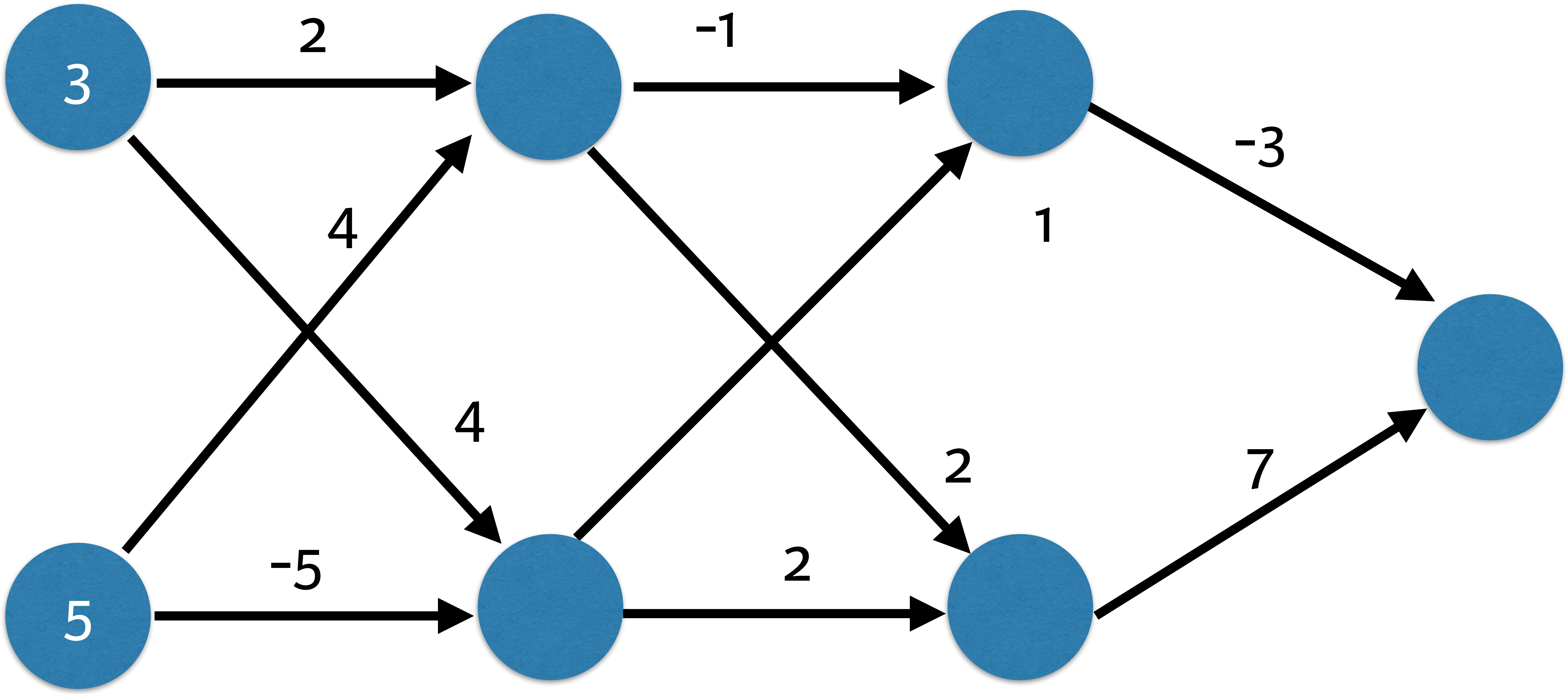
Activation functions

```
In [1]: import numpy as np

In [2]: input_data = np.array([-1, 2])

In [3]: weights = {'node_0': np.array([3, 3]),
   ...:            'node_1': np.array([1, 5]),
   ...:            'output': np.array([2, -1])}

In [4]: node_0_input = (input_data * weights['node_0']).sum()

In [5]: node_0_output = np.tanh(node_0_input)

In [6]: node_1_input = (input_data * weights['node_1']).sum()

In [7]: node_1_output = np.tanh(node_1_input)

In [8]: hidden_layer_outputs = np.array([node_0_output, node_1_output])

In [9]: output = (hidden_layer_output * weights['output']).sum()

In [10]: print(output)
1.23822425257694254
```

Let's practice!

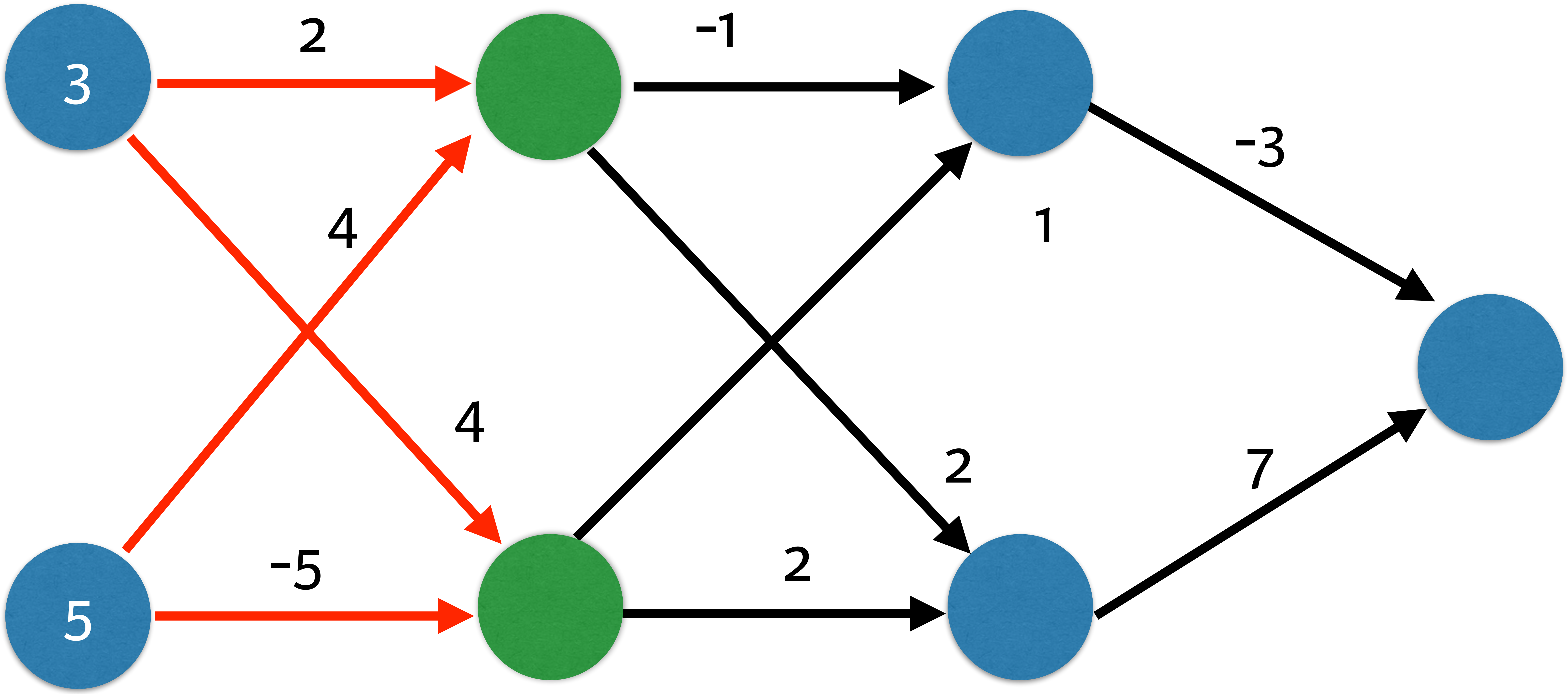
Deeper networks

Multiple hidden layers



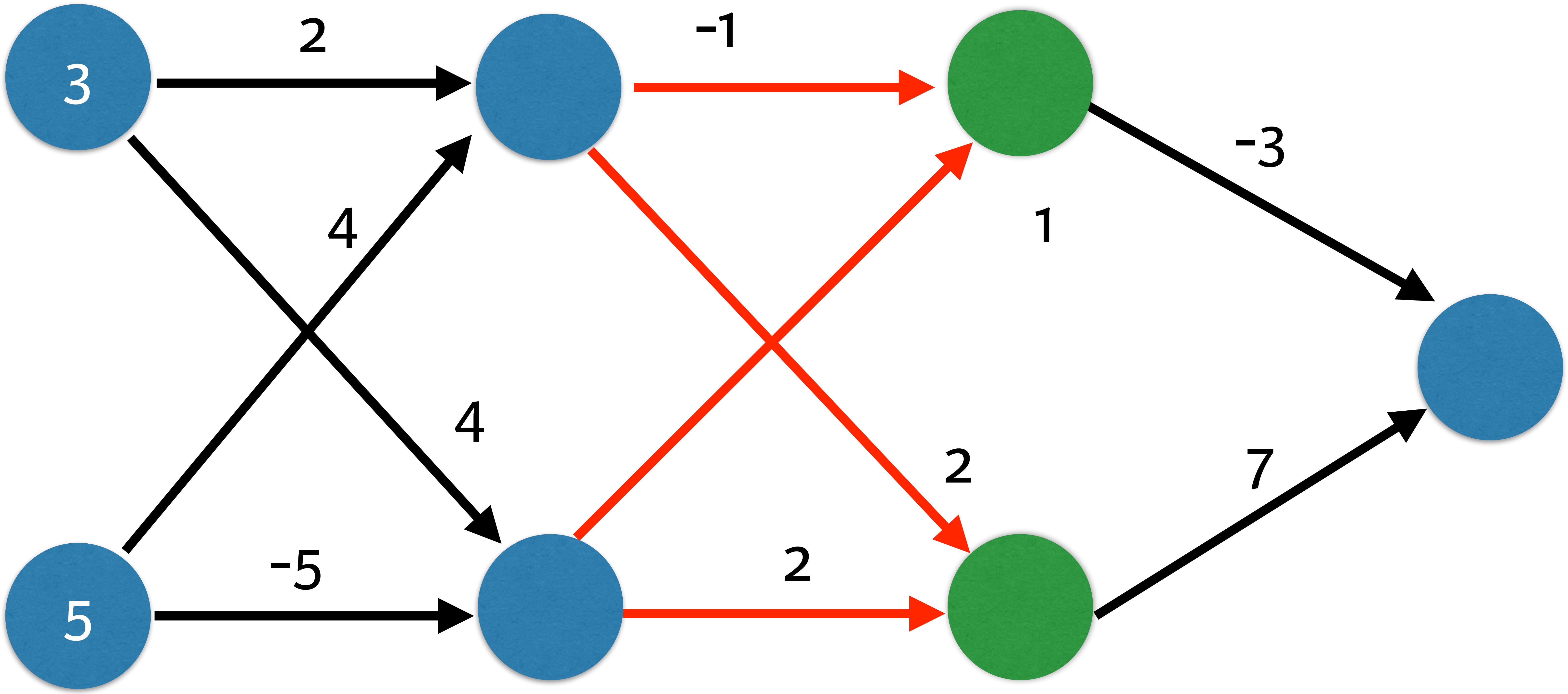
Calculate with ReLU Activation Function

Multiple hidden layers



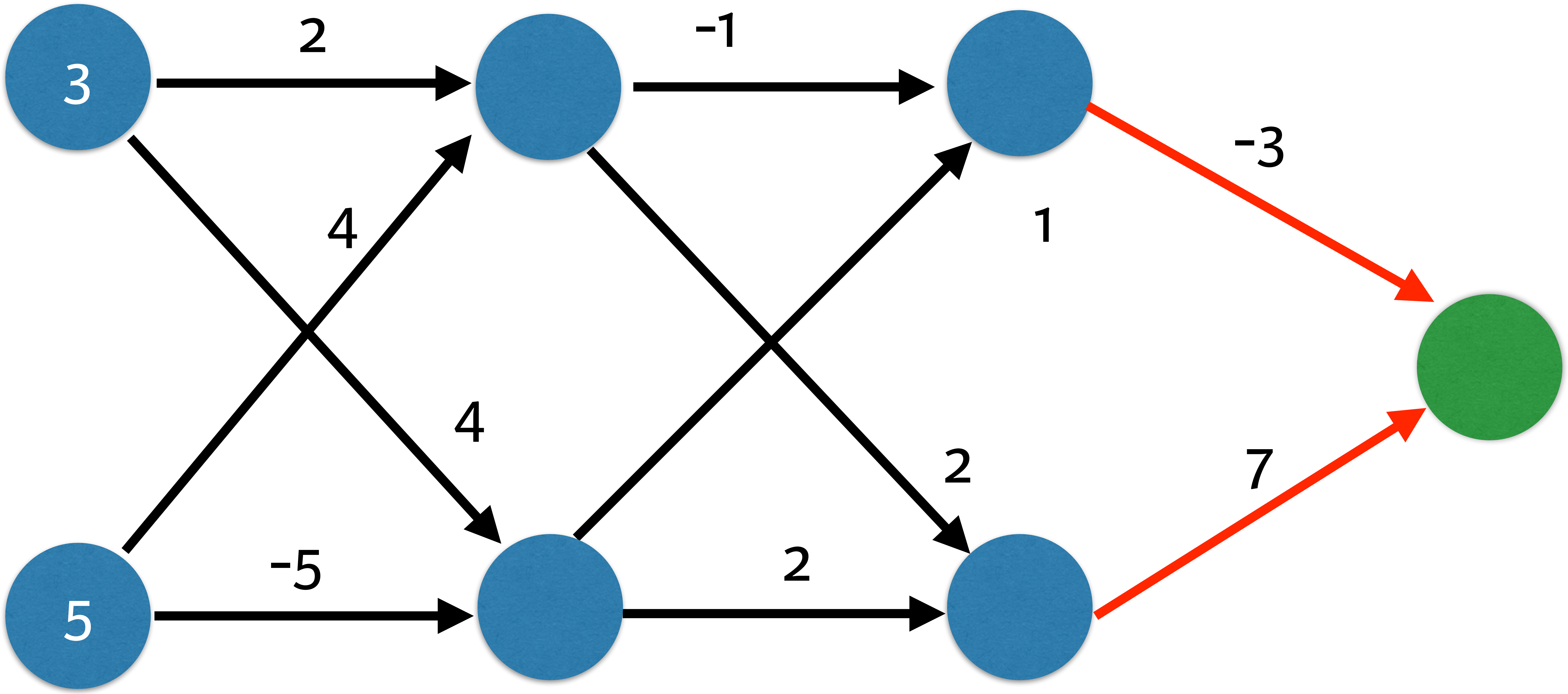
Calculate with ReLU Activation Function

Multiple hidden layers



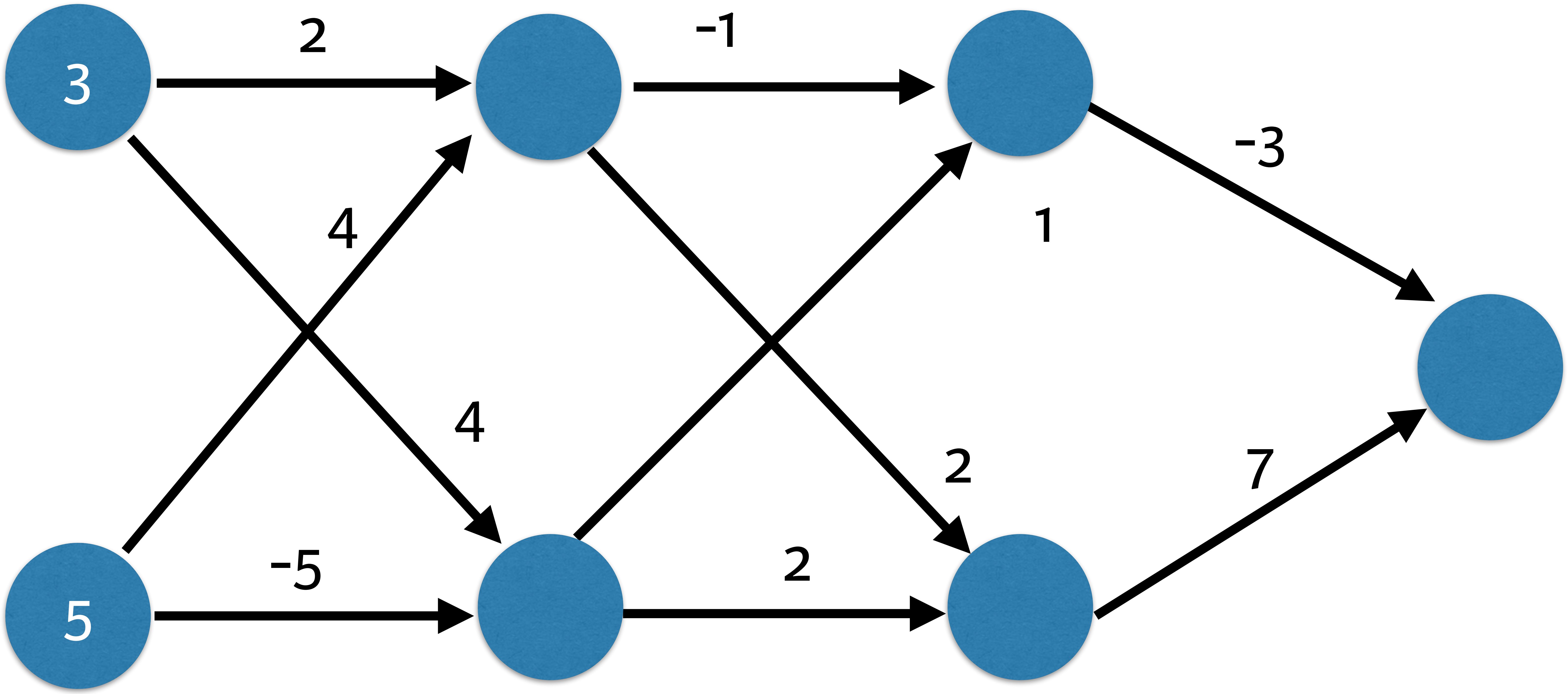
Calculate with ReLU Activation Function

Multiple hidden layers



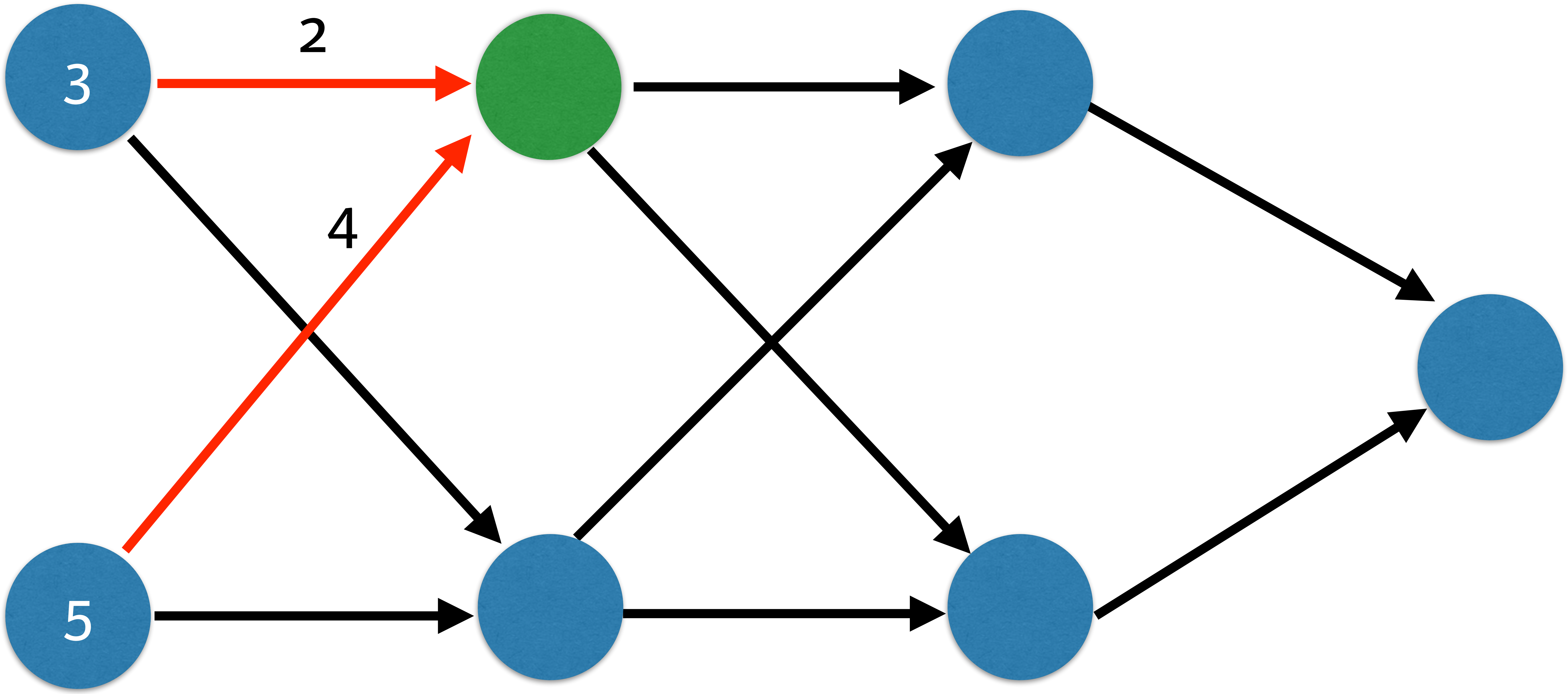
Calculate with ReLU Activation Function

Multiple hidden layers



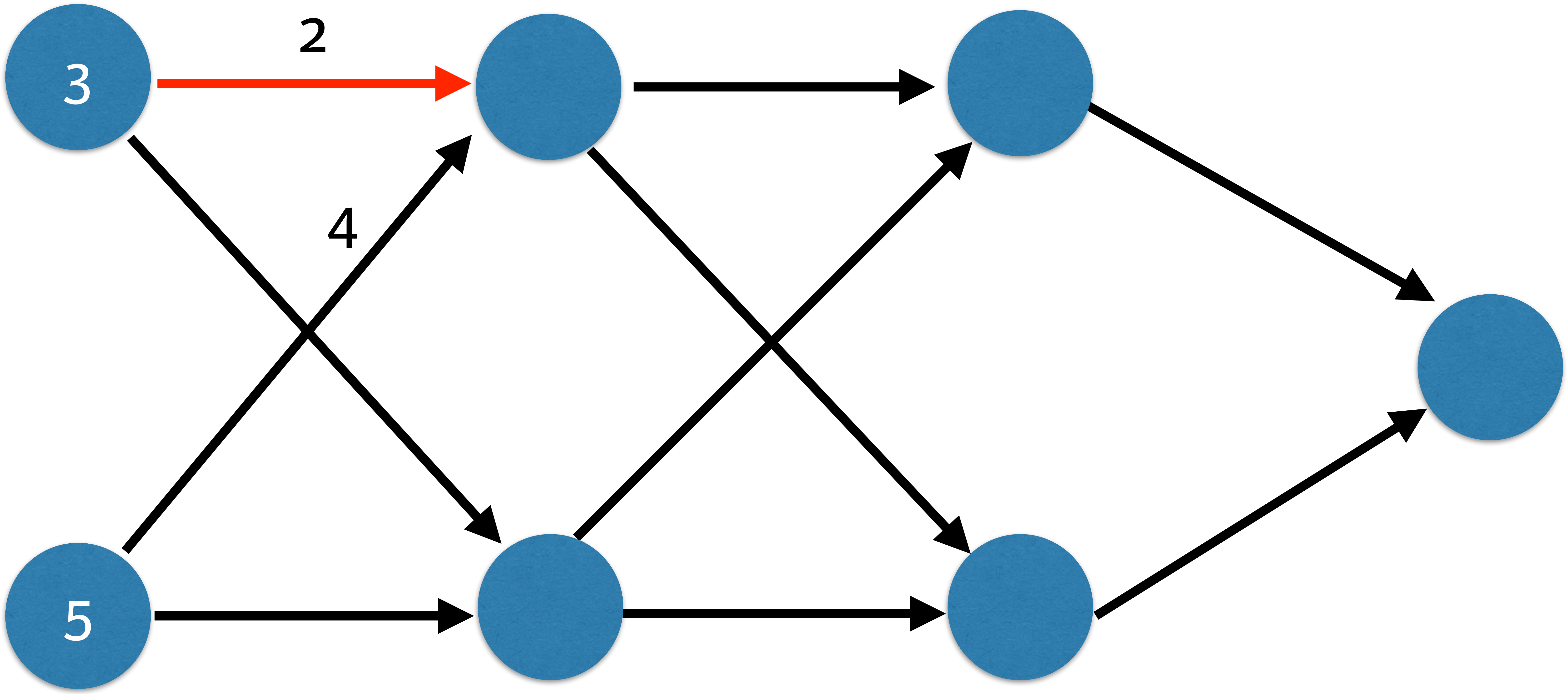
Calculate with ReLU Activation Function

Multiple hidden layers



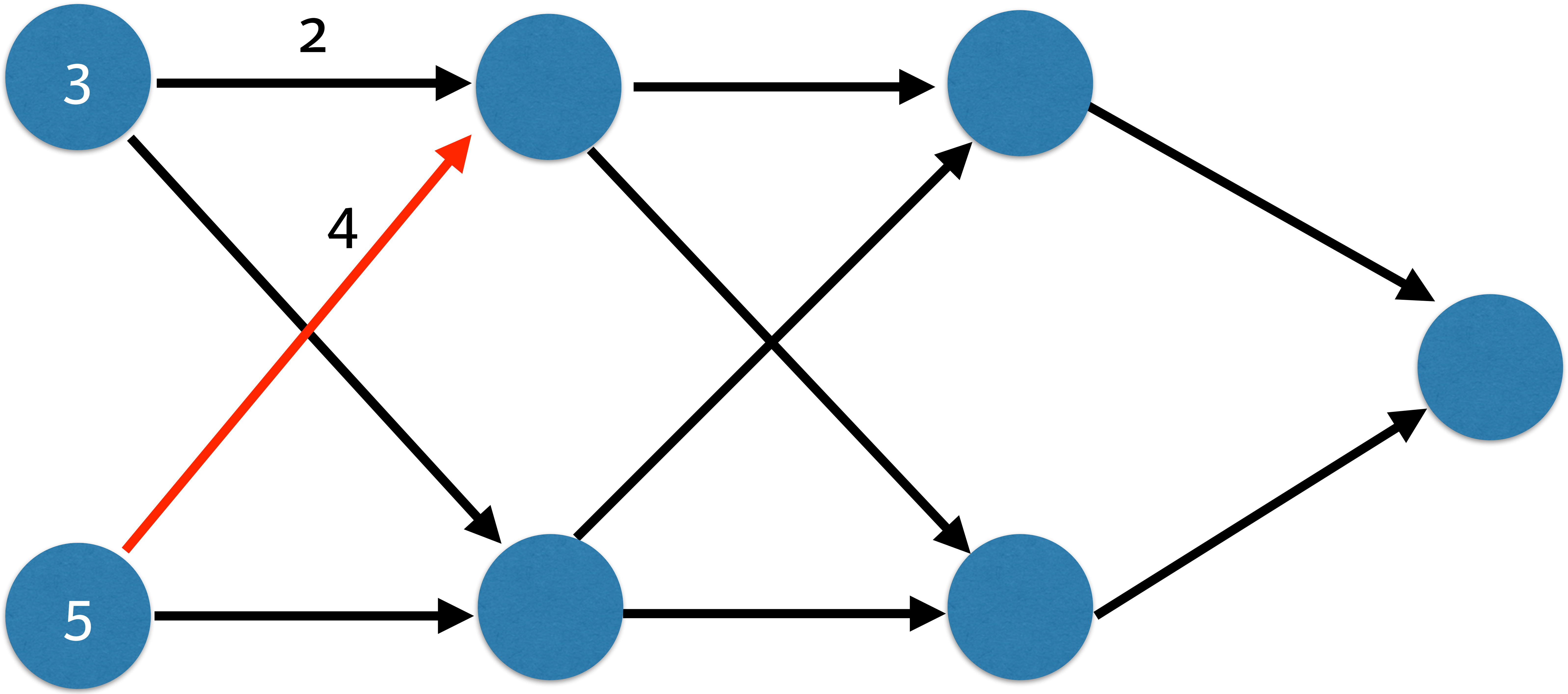
Calculate with ReLU Activation Function

Multiple hidden layers



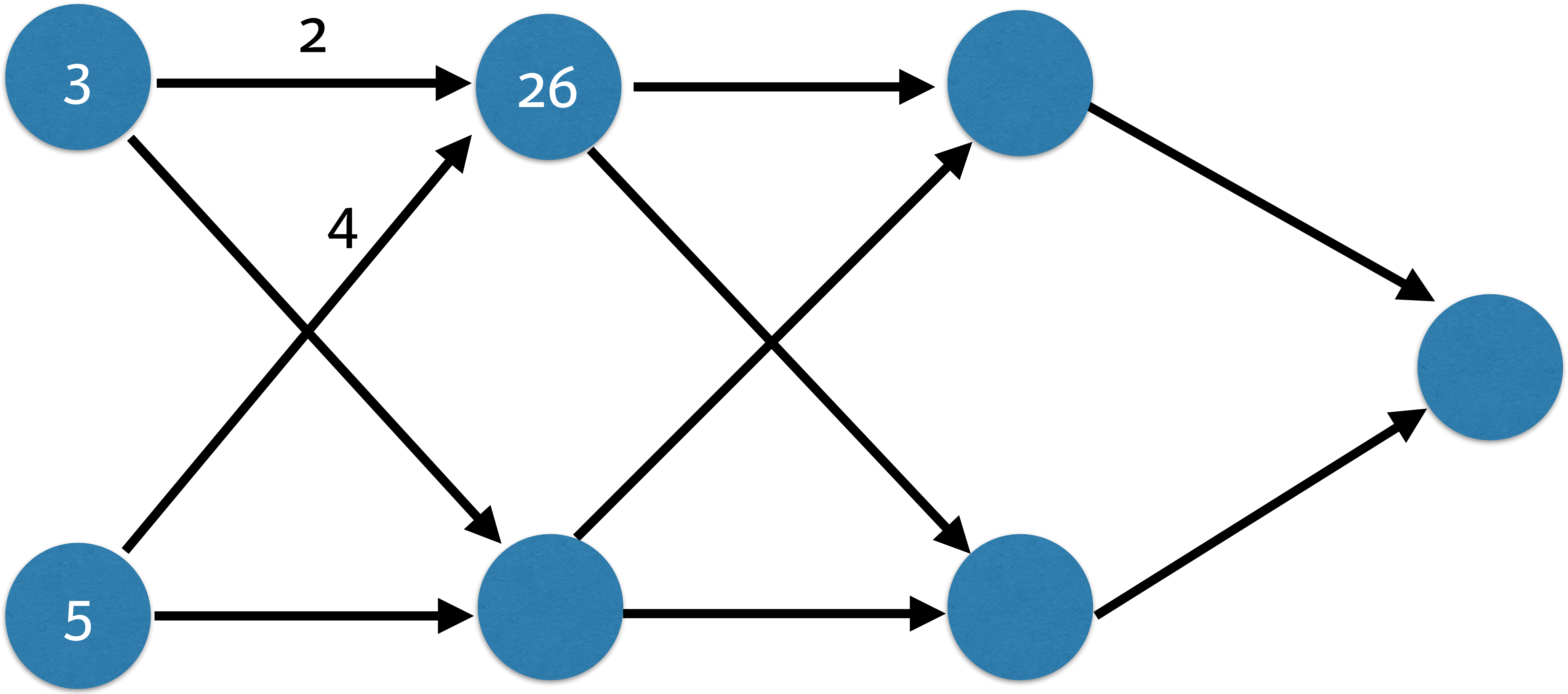
Calculate with ReLU Activation Function

Multiple hidden layers



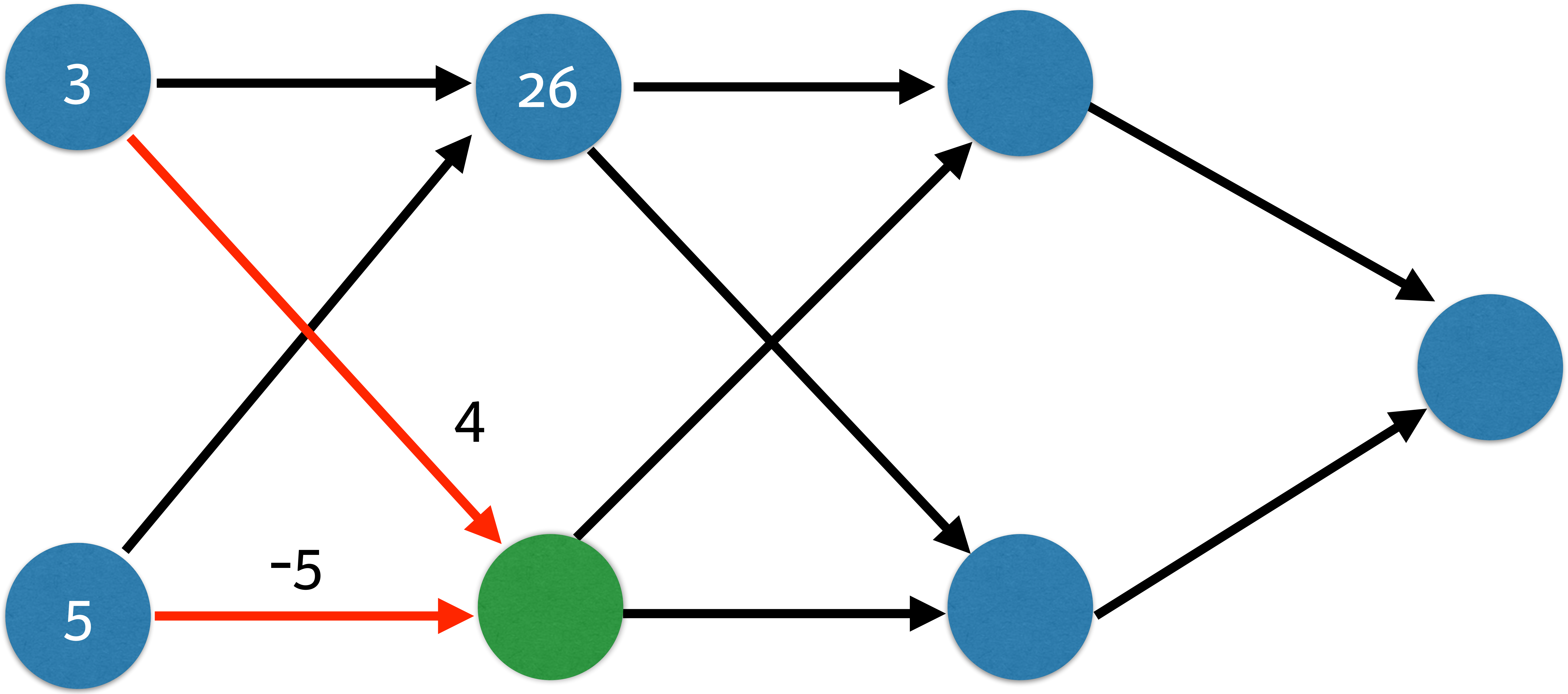
Calculate with ReLU Activation Function

Multiple hidden layers



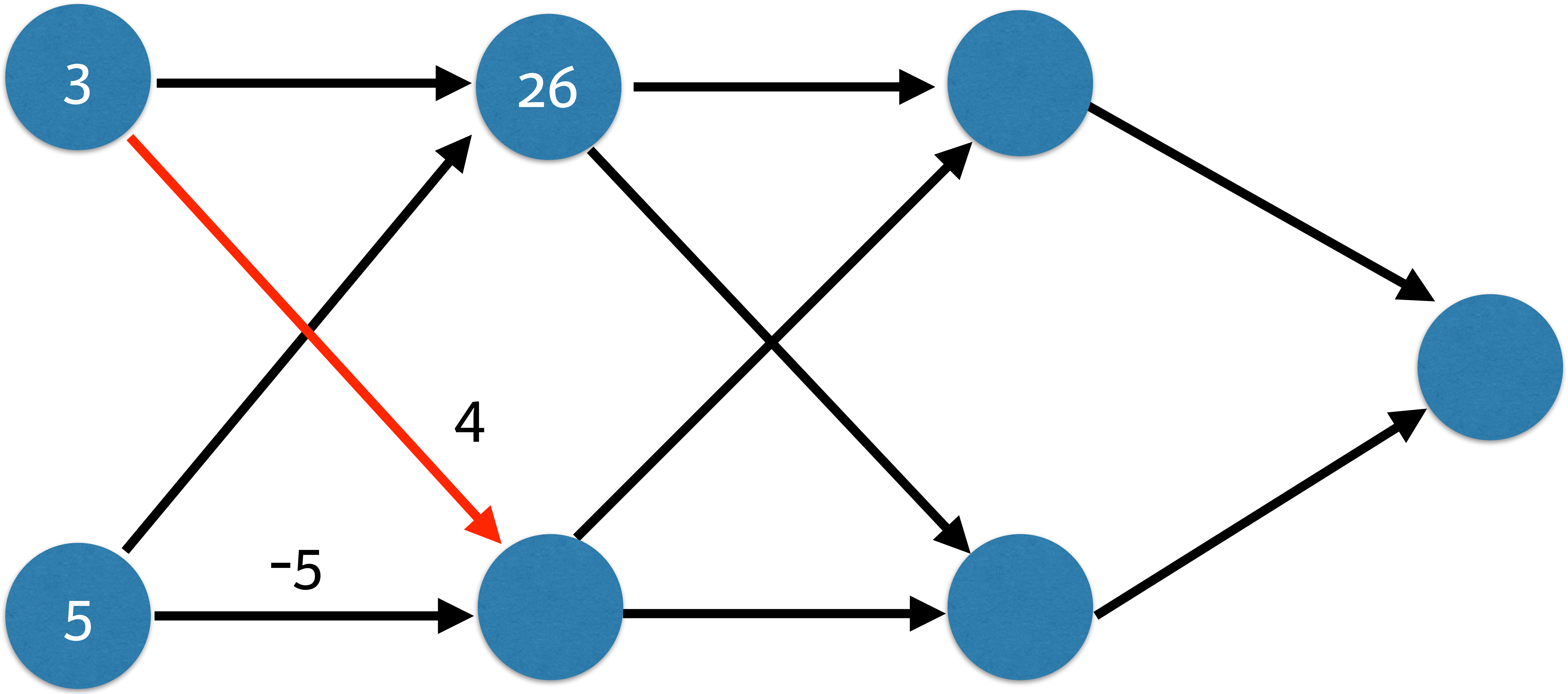
Calculate with ReLU Activation Function

Multiple hidden layers



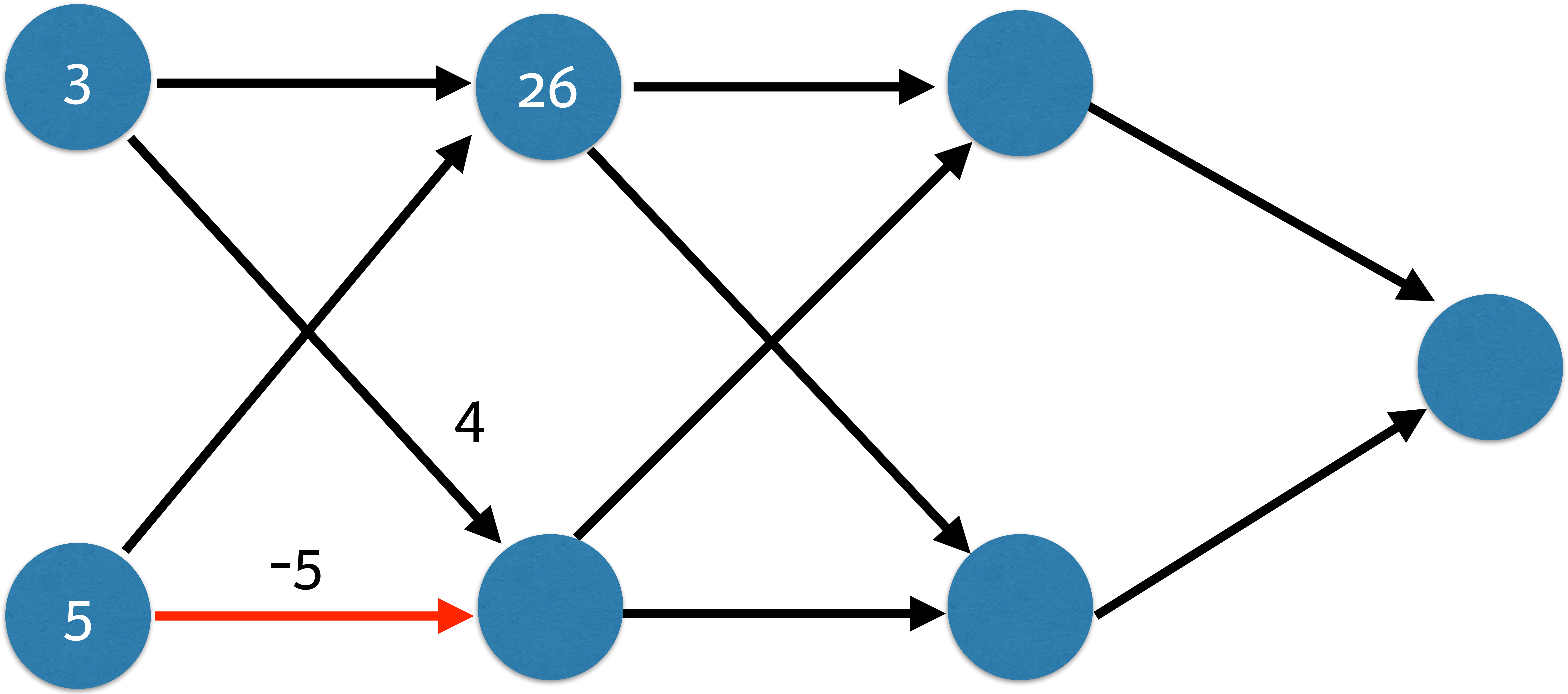
Calculate with ReLU Activation Function

Multiple hidden layers



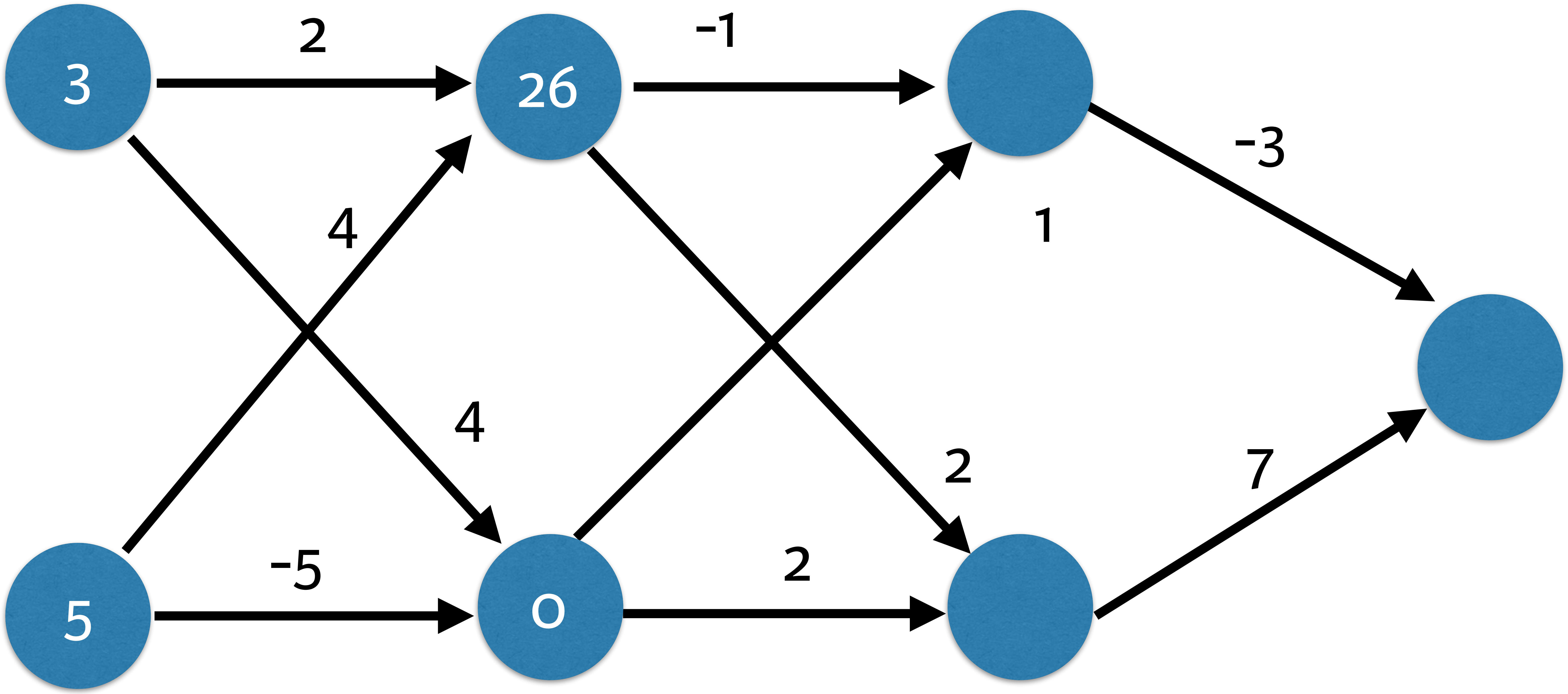
Calculate with ReLU Activation Function

Multiple hidden layers



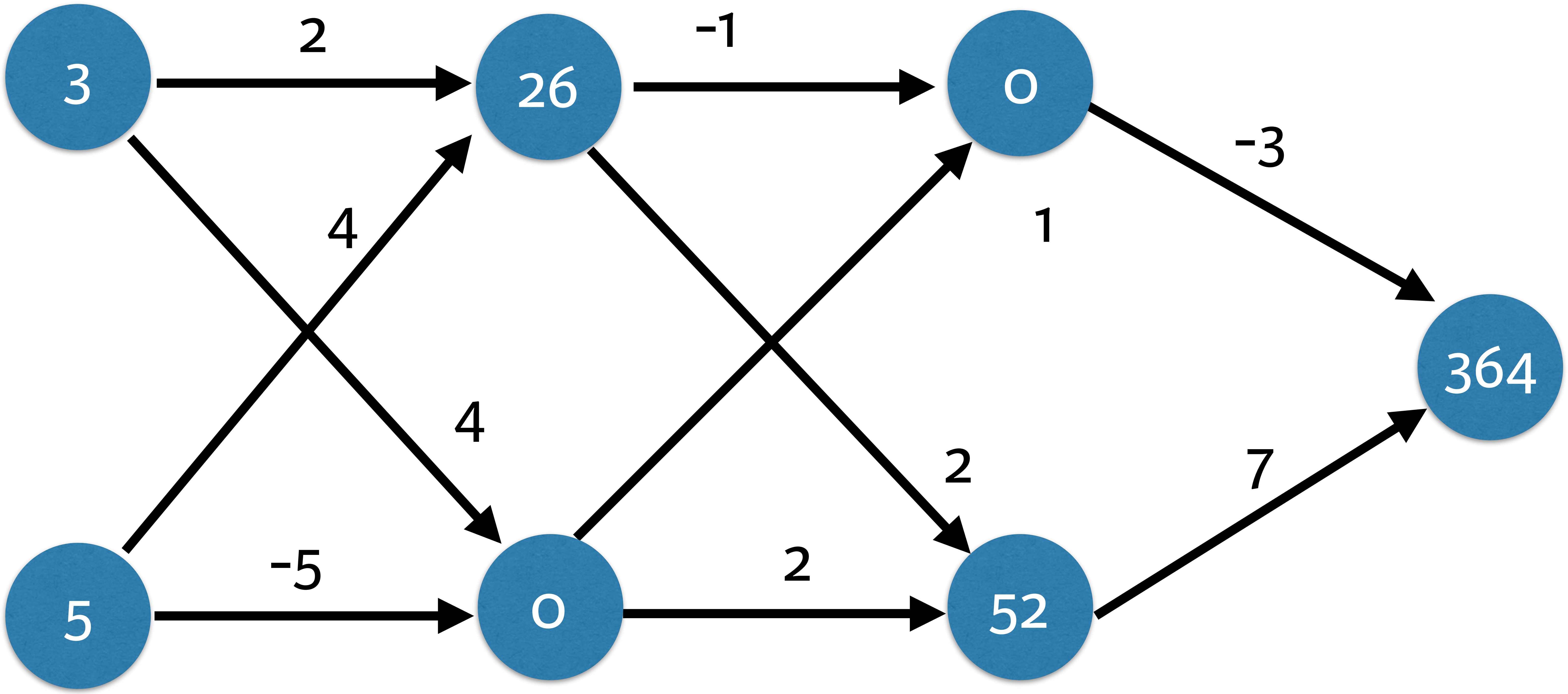
Calculate with ReLU Activation Function

Multiple hidden layers



Calculate with ReLU Activation Function

Multiple hidden layers

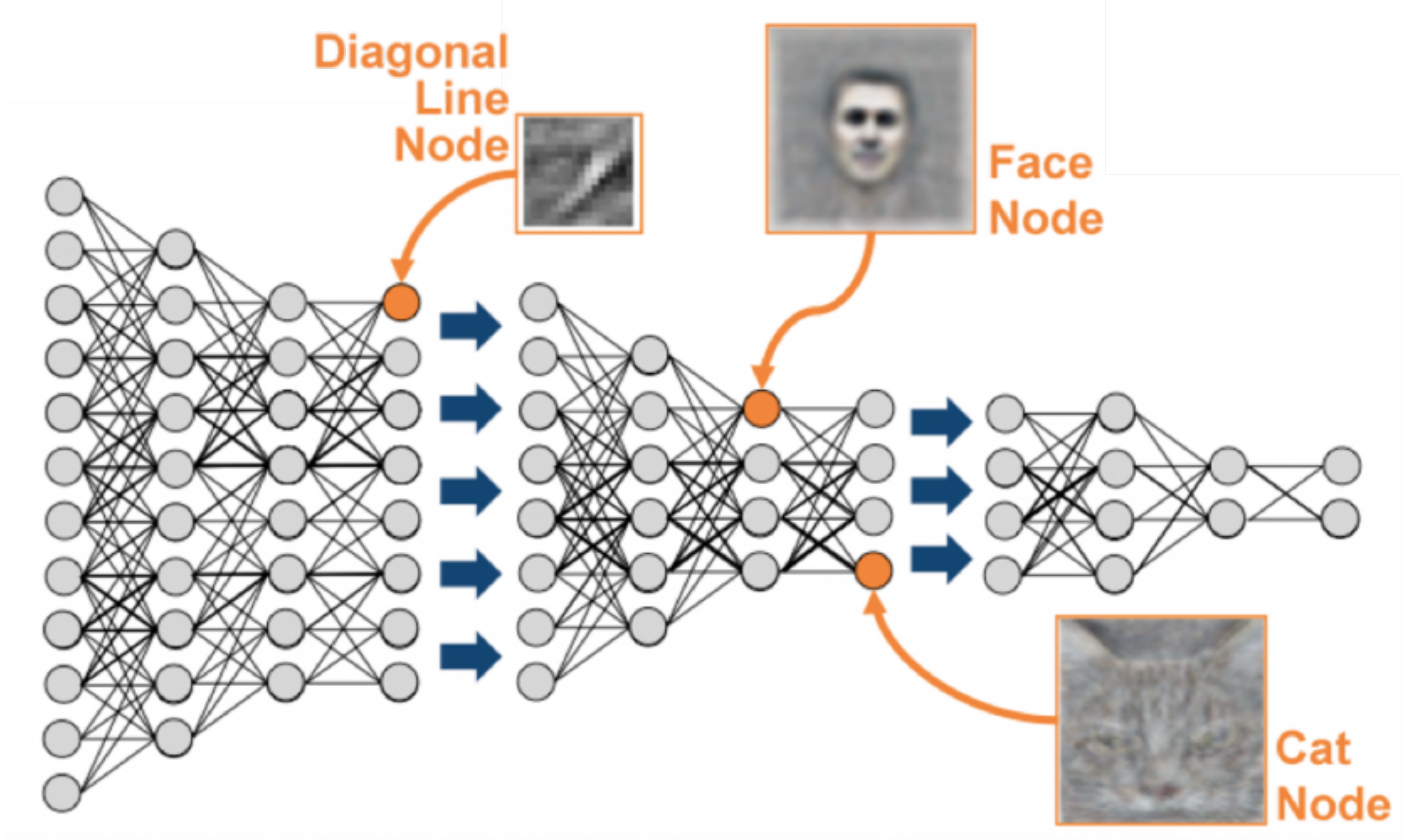


Calculate with ReLU Activation Function

Representation learning

- Deep networks internally build representations of patterns in the data
- Partially replace the need for feature engineering
- Subsequent layers build increasingly sophisticated representations of raw data

Representation learning



Deep learning

- Modeler doesn’t need to specify the interactions
- When you train the model, the neural network gets weights that find the relevant patterns to make better predictions

Let's practice!