Lec.4. Introduction to Deep Learning

Machine Learning II

Aidos Sarsembayev, IITU, Almaty, 2019

Outline

1. Introduction to Deep Learning
   - The idea behind NN
   - Perceptron
   - Debug and tune deep learning models on conventional prediction problems
   - Lay the foundation for progressing towards modern applications
2. Forward propagation
3. Activation Functions
4. Deeper networks
5. Representation learning

Introduction to Deep Learning.
The idea behind NN

Artificial Neural Networks are the computational models inspired by the human brain.

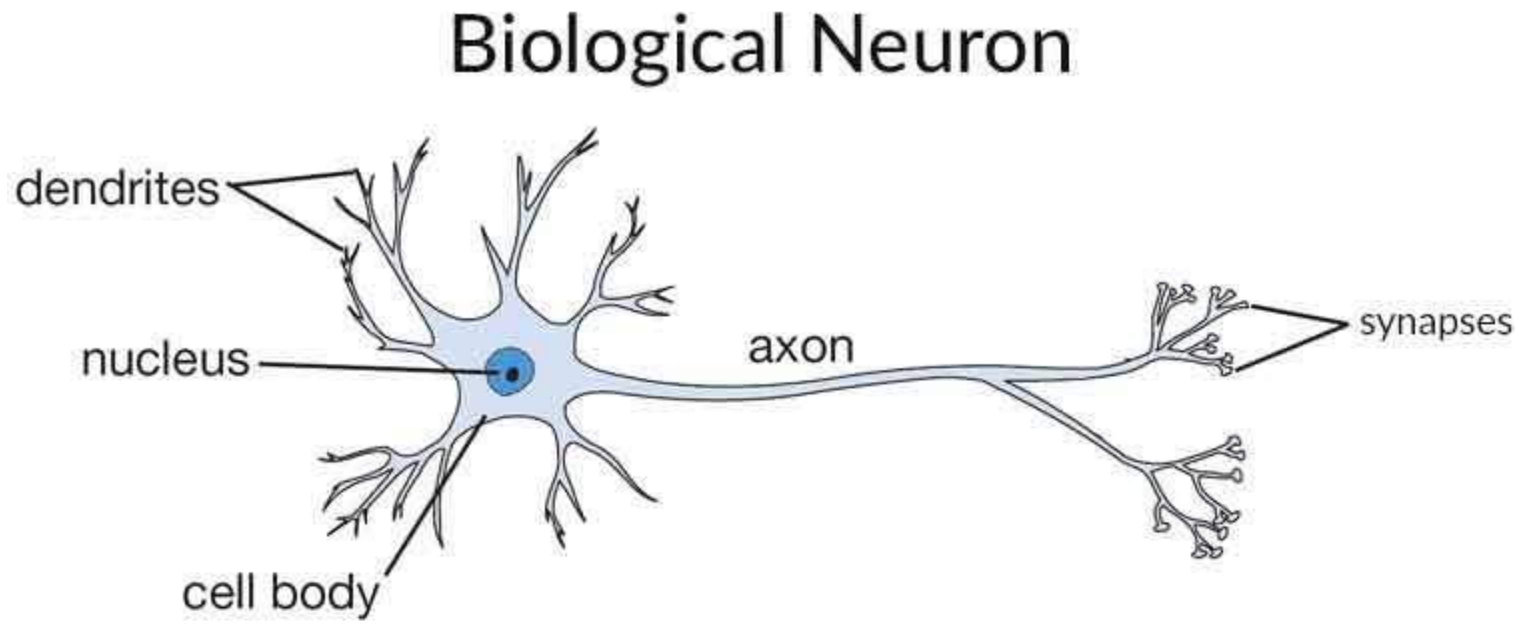
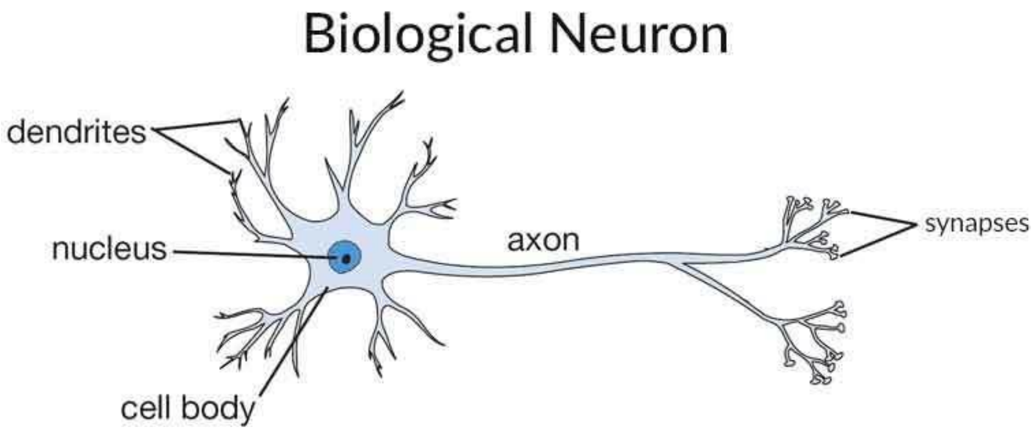


Image Source – cs231n.github.io
https://www.xenonstack.com/blog/artificial-neural-networks-applications-algorithms/

Introduction to Deep Learning. The idea behind NN

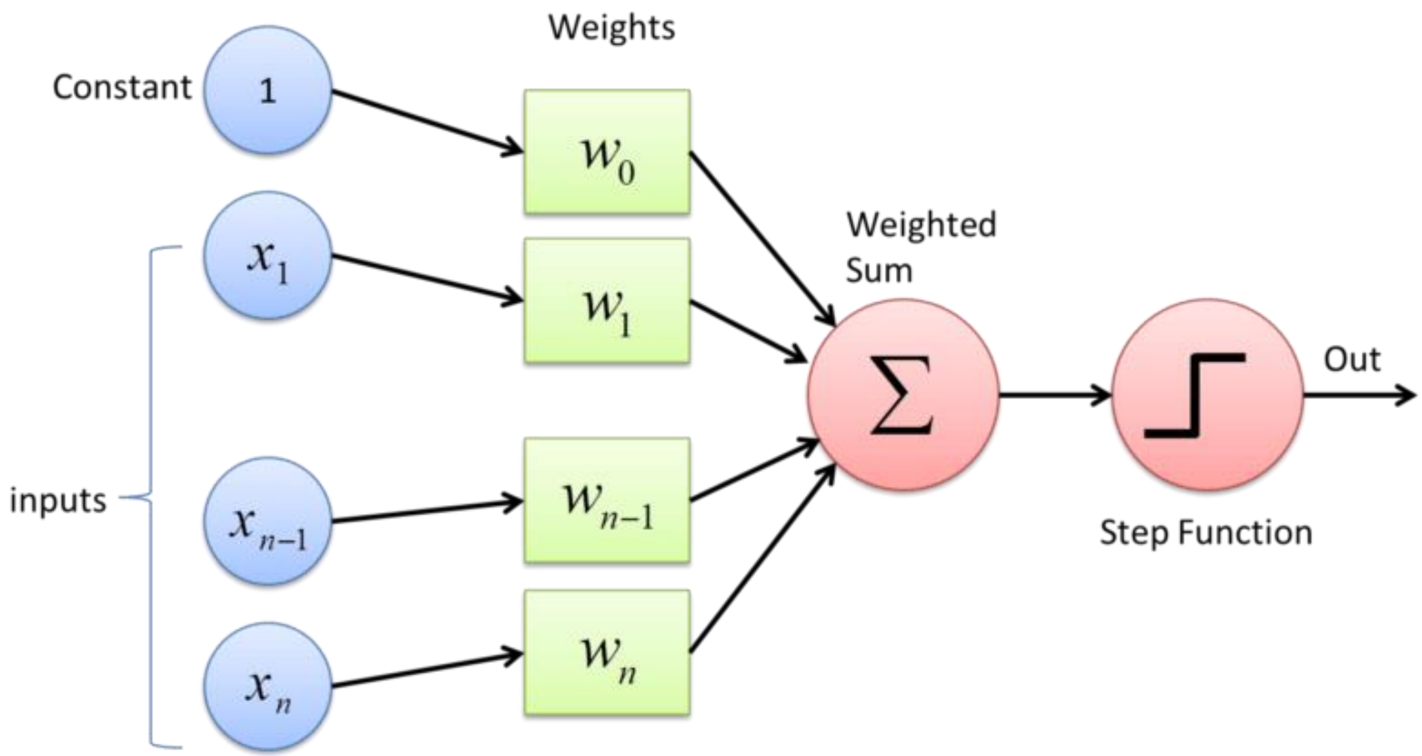


- **Function of Dendrite**
  - It receives signals from other neurons.
- **Soma (cell body)**
  - It sums all the incoming signals to generate input.
- **Axon Structure**
  - When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons.
- **Synapses Working**
  - The point of interconnection of one neuron with other neurons. The amount of signal transmitted depend upon the strength (synaptic weights) of the connections.

Image Source – cs231n.github.io
https://www.xenonstack.com/blog/artificial-neural-networks-applications-algorithms/

Introduction to Deep Learning. Perceptron

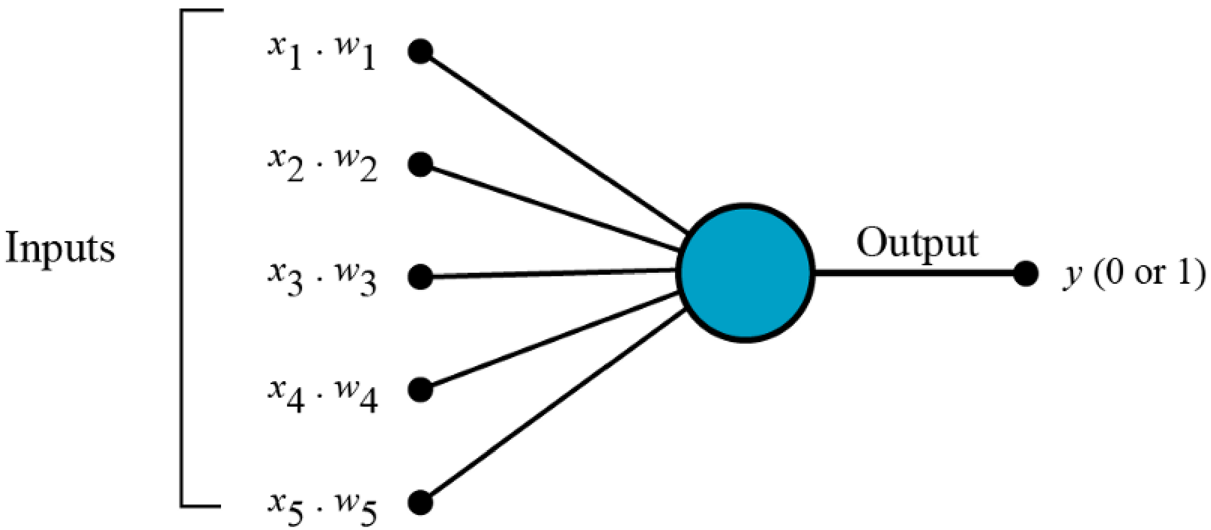
Initially it all started from perceptron



https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Introduction to Deep Learning . Perceptron

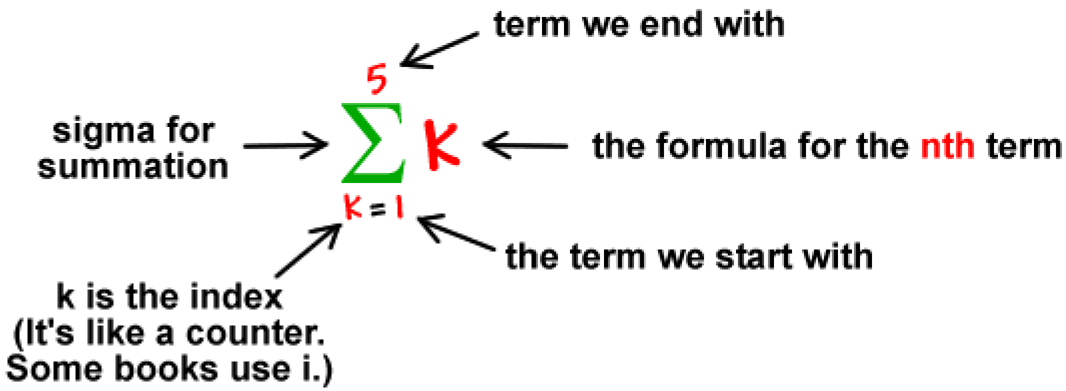
a. All the inputs ***x*** are multiplied with their weights ***w***. Let's call it ***k.***



https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Introduction to Deep Learning . Perceptron

- b. ***Add*** all the multiplied values and call them ***Weighted Sum.***



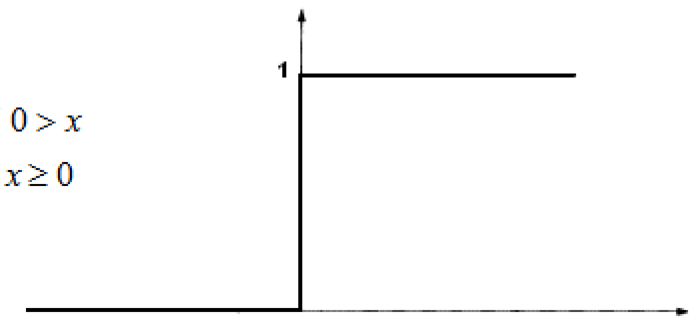
https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Introduction to Deep Learning . Perceptron

- c. ***Apply*** that weighted sum to the correct [***Activation Function***.]()

Unit step (threshold)

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Introduction to Deep Learning . Perceptron

- **Why do we need Weights and Bias?**
  - *Weights* shows the strength of the particular node.
  - *A bias* value allows you to shift the activation function curve up or down.
- **Why do we need Activation Function?**
  - In short, *the activation functions are used to map the input between the required values like (0, 1) or (-1, 1)*.
- **Where we use Perceptron?**
  - Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.

https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

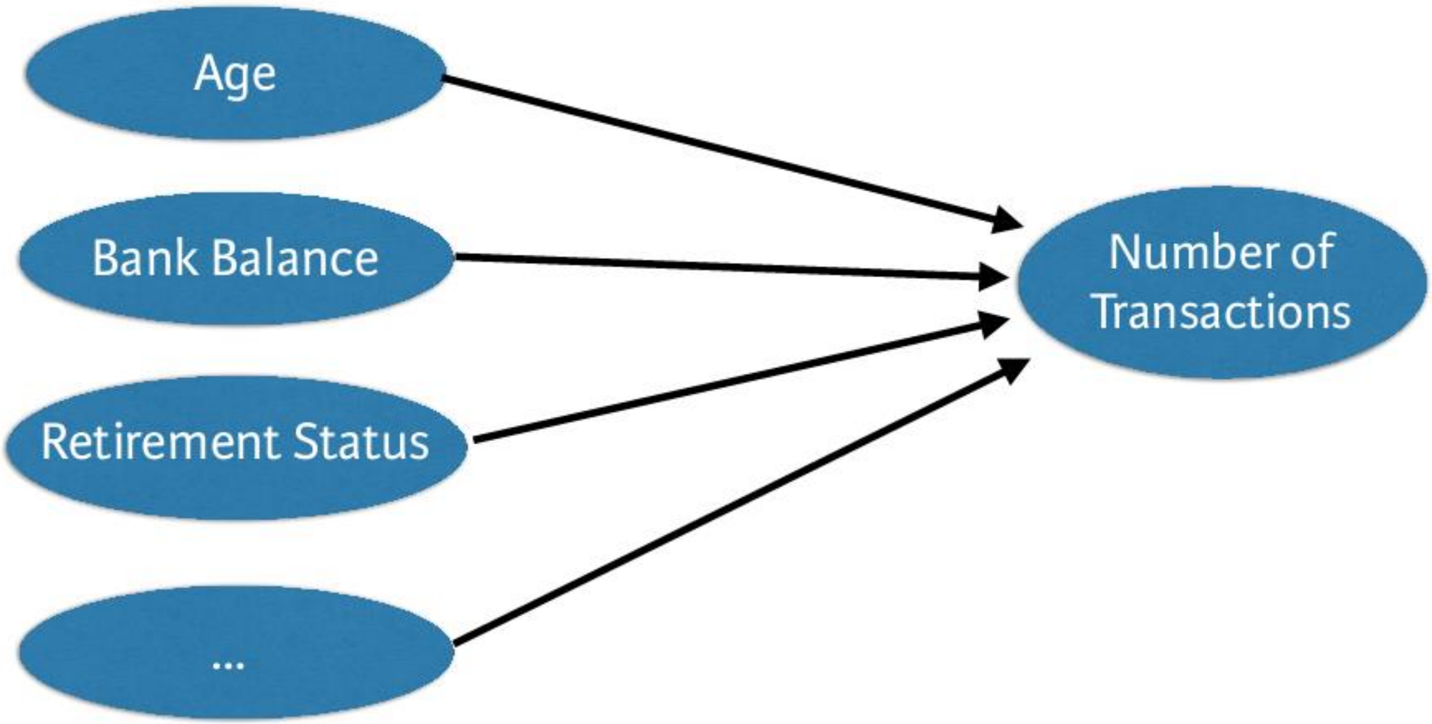
Introduction to Deep Learning

Imagine you work for a bank

● You need to predict how many transactions each customer

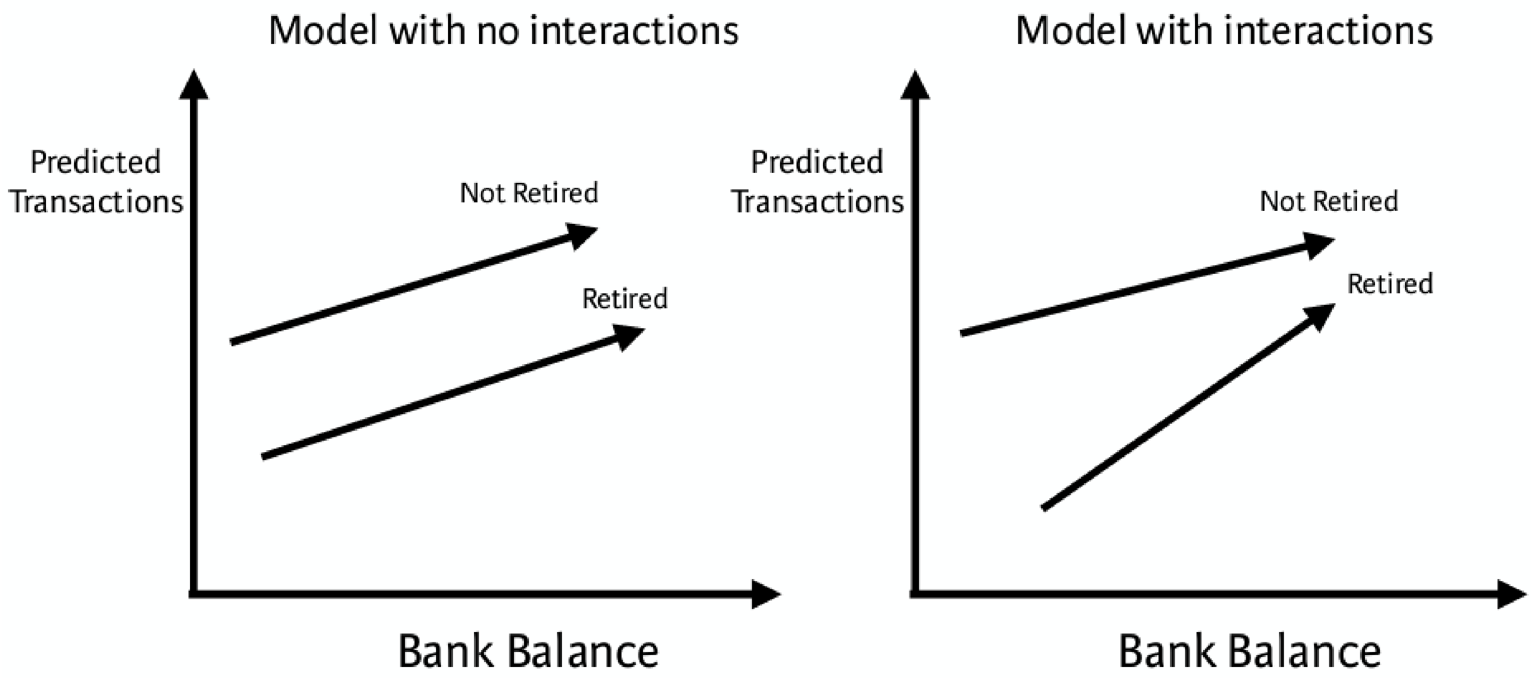
Introduction to Deep Learning

Example as seen by linear regression



Introduction to Deep Learning

Example as seen by linear regression



Introduction to Deep Learning

Interactions:

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

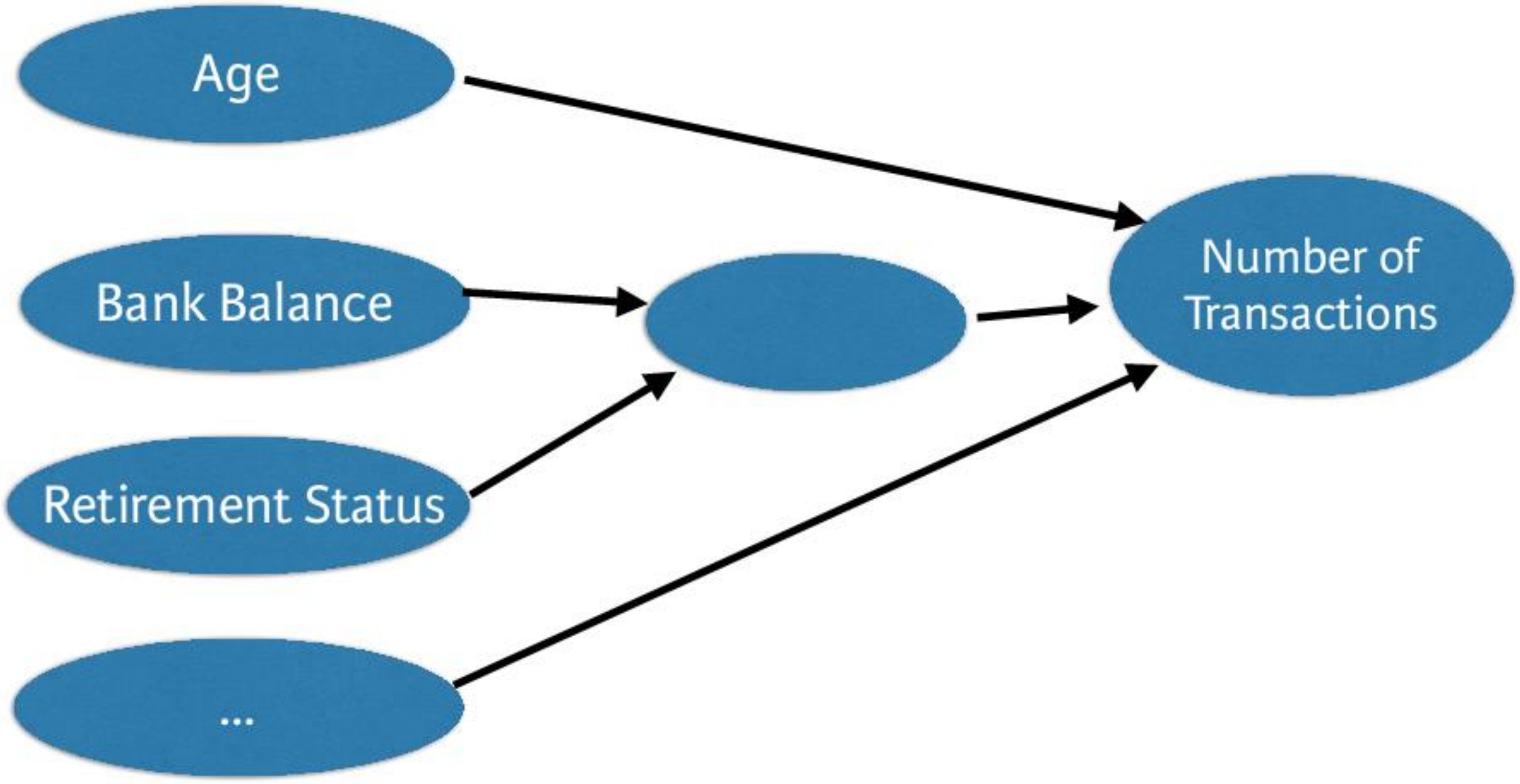
Introduction to Deep Learning

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))
```

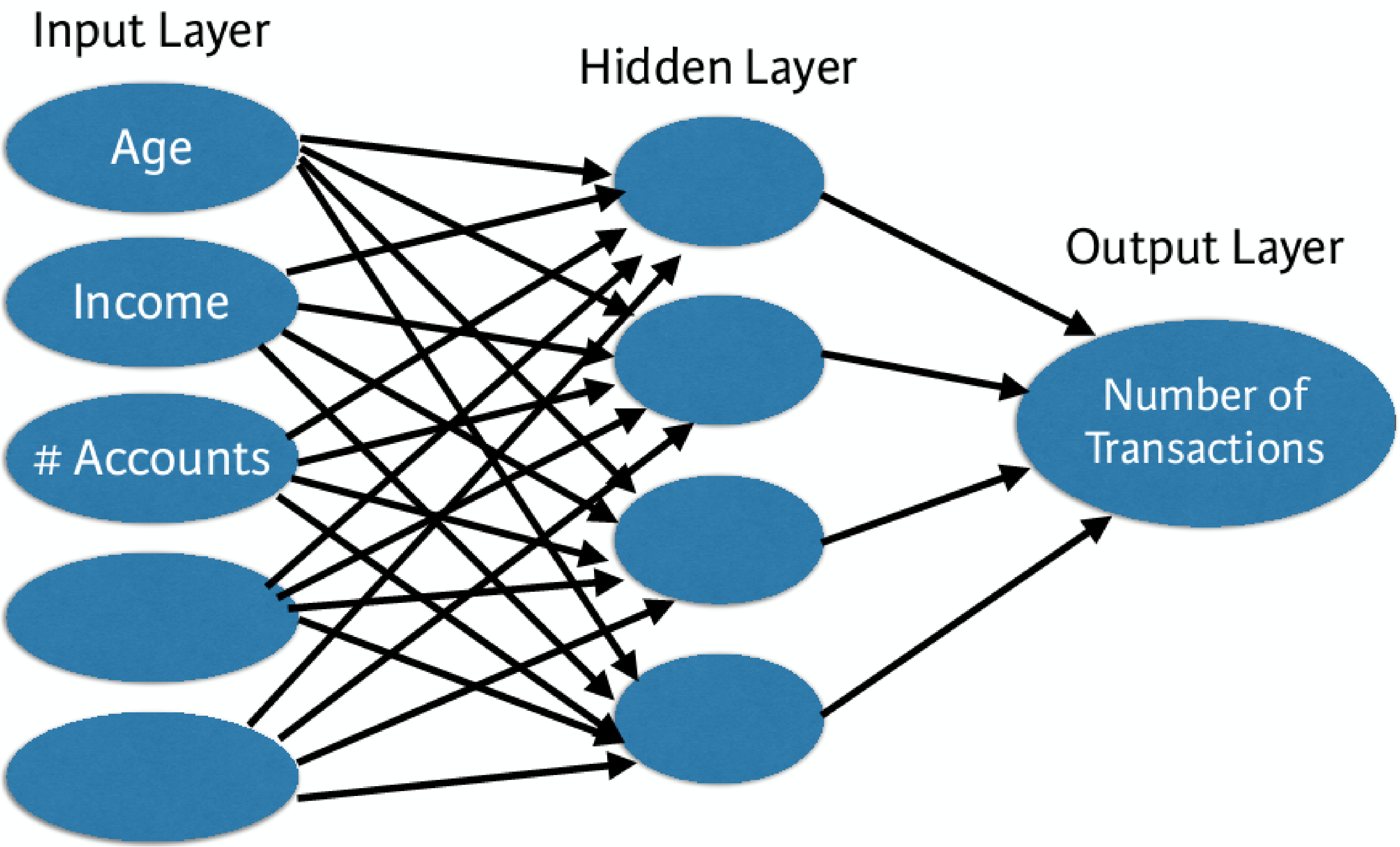
Introduction to Deep Learning

Deep learning models capture interactions



Introduction to Deep Learning

Interactions in neural network

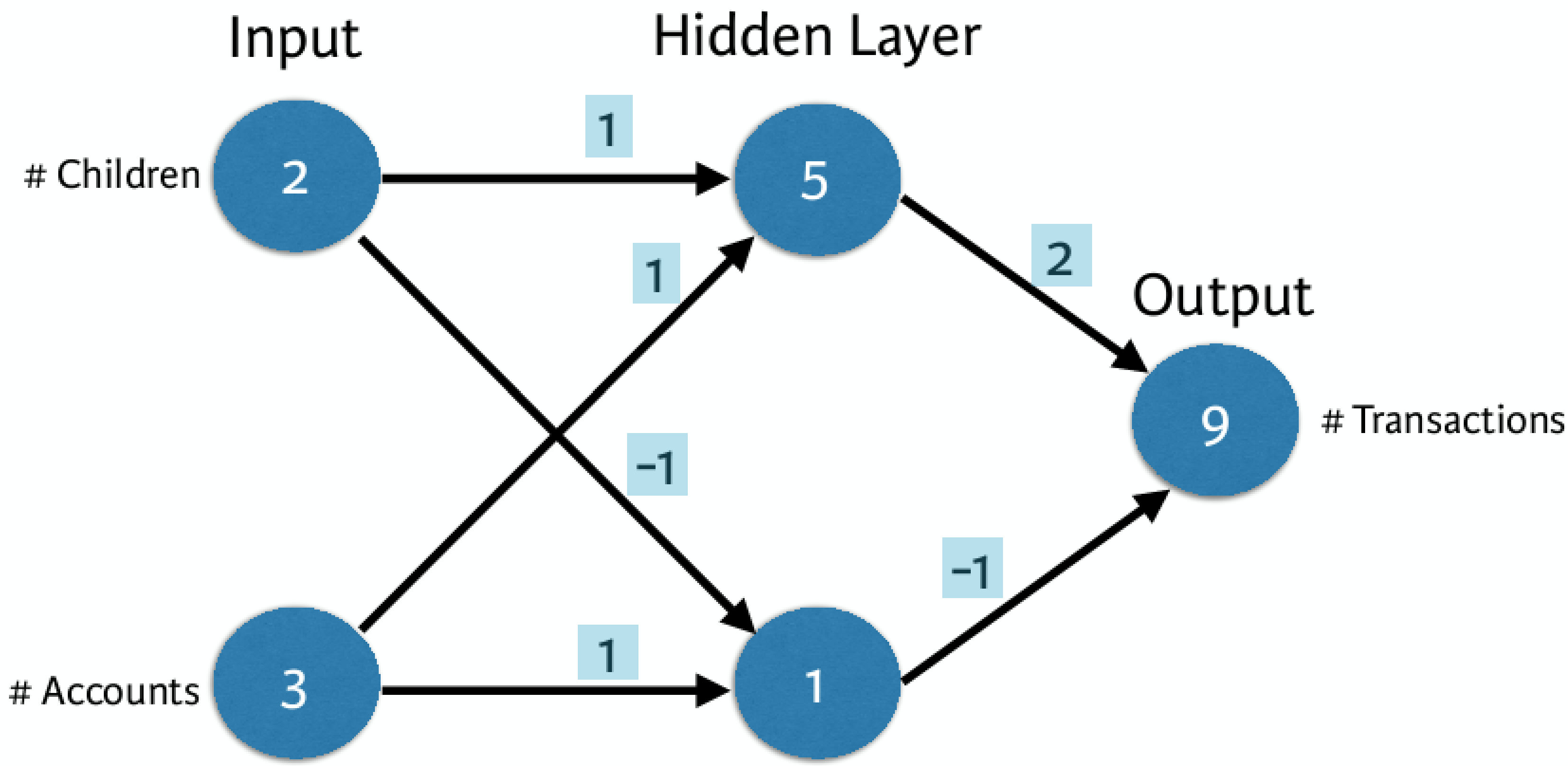


Forward propagation

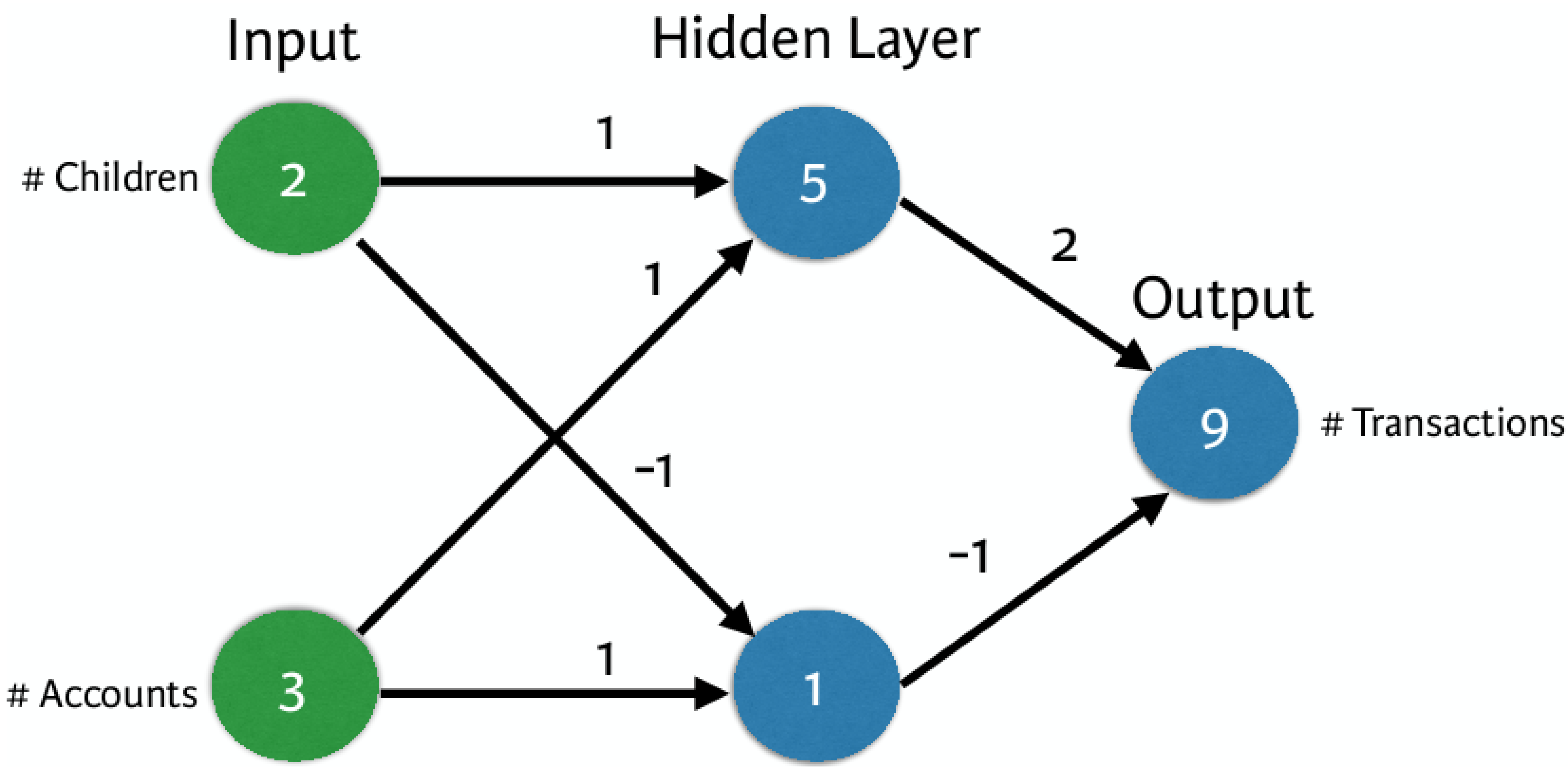
Bank transactions example

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

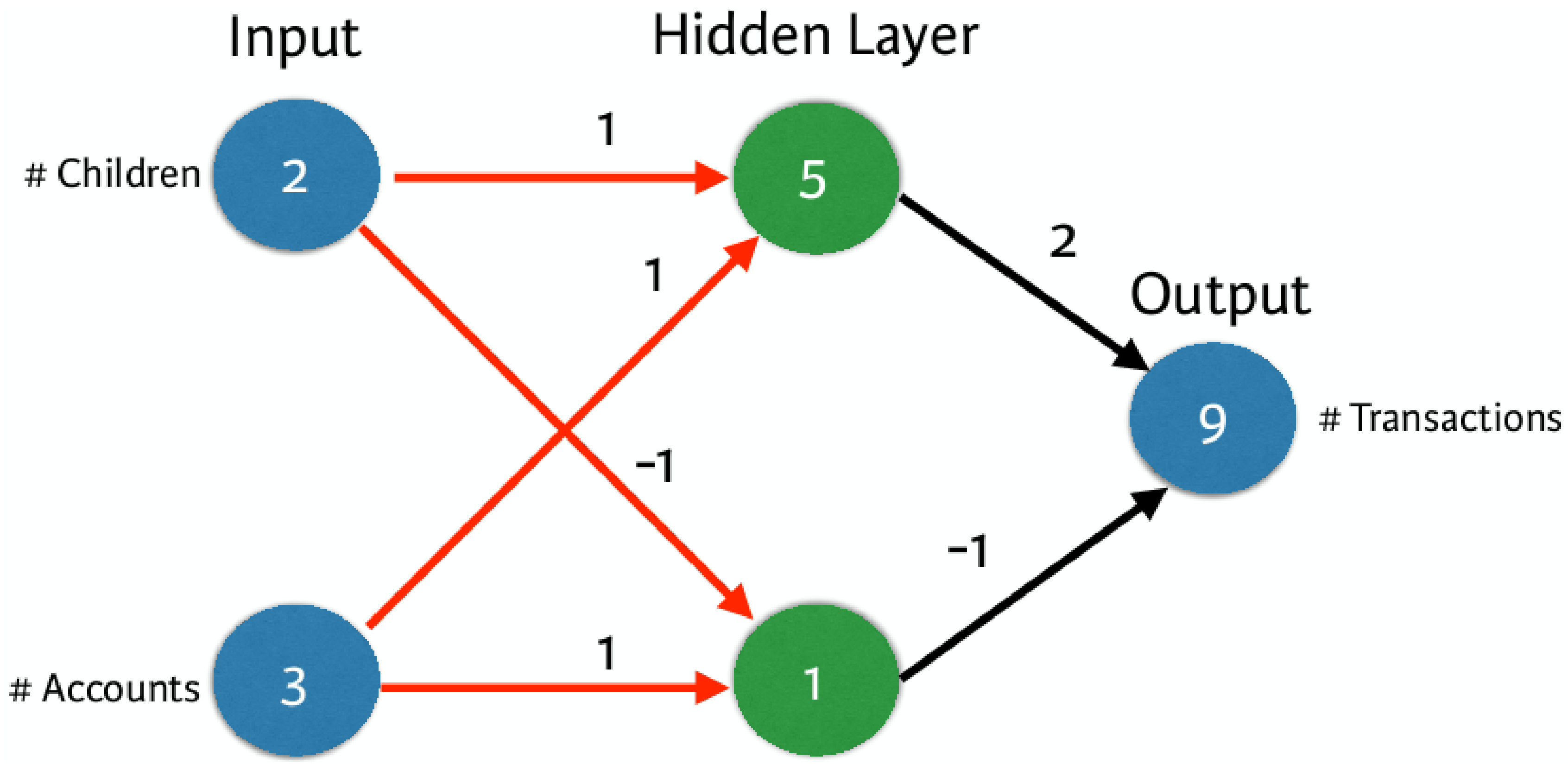
Forward propagation



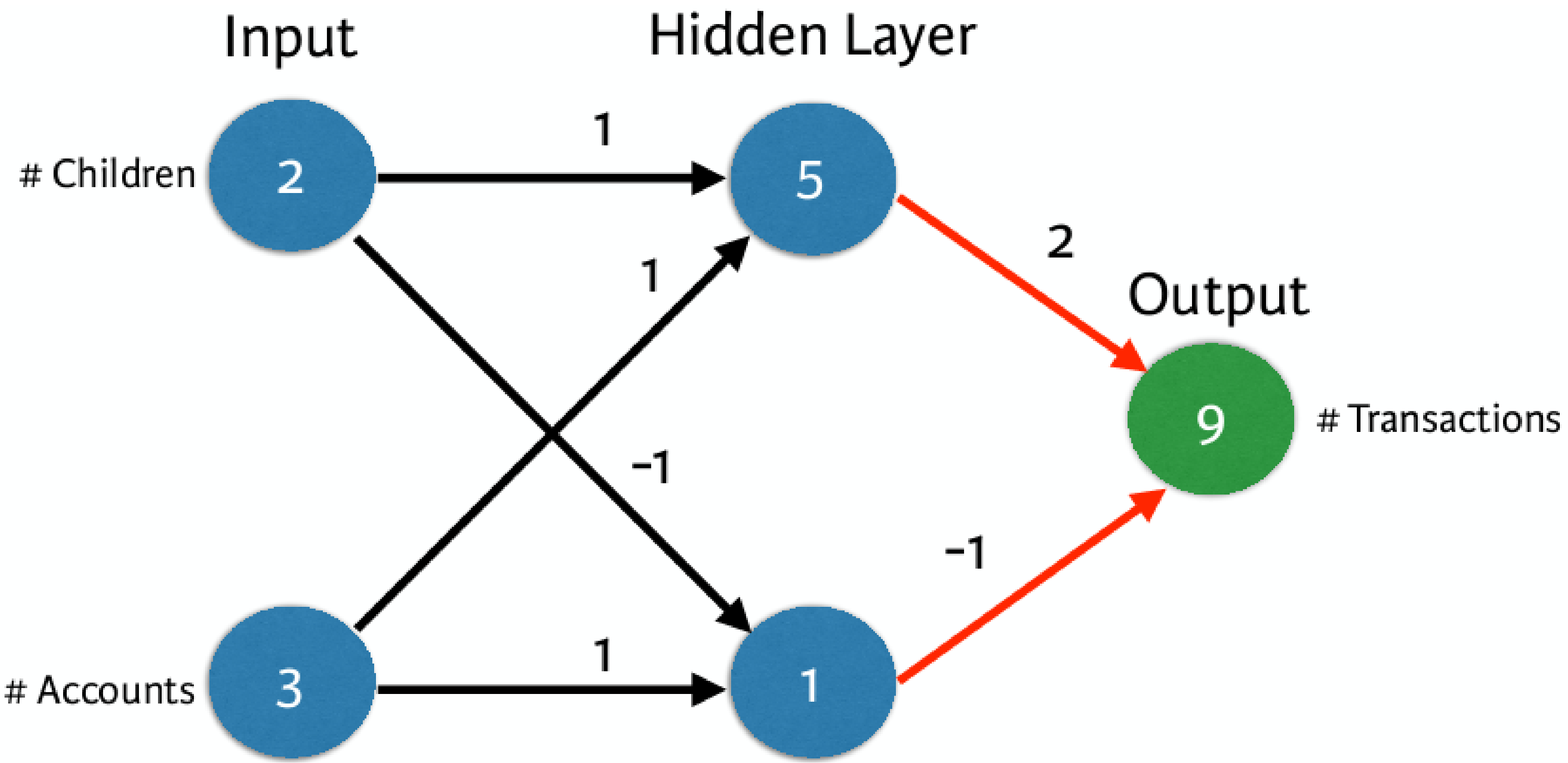
Forward propagation



Forward propagation



Forward propagation

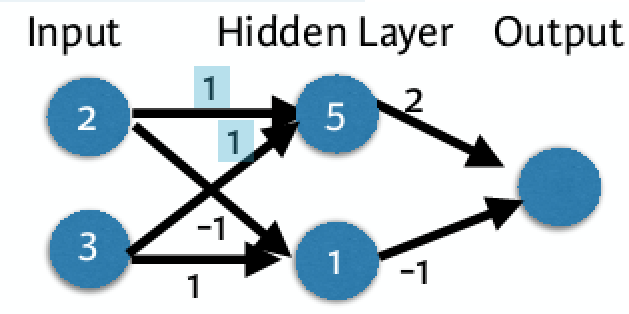


Forward propagation

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

Forward propagation

```
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()
```

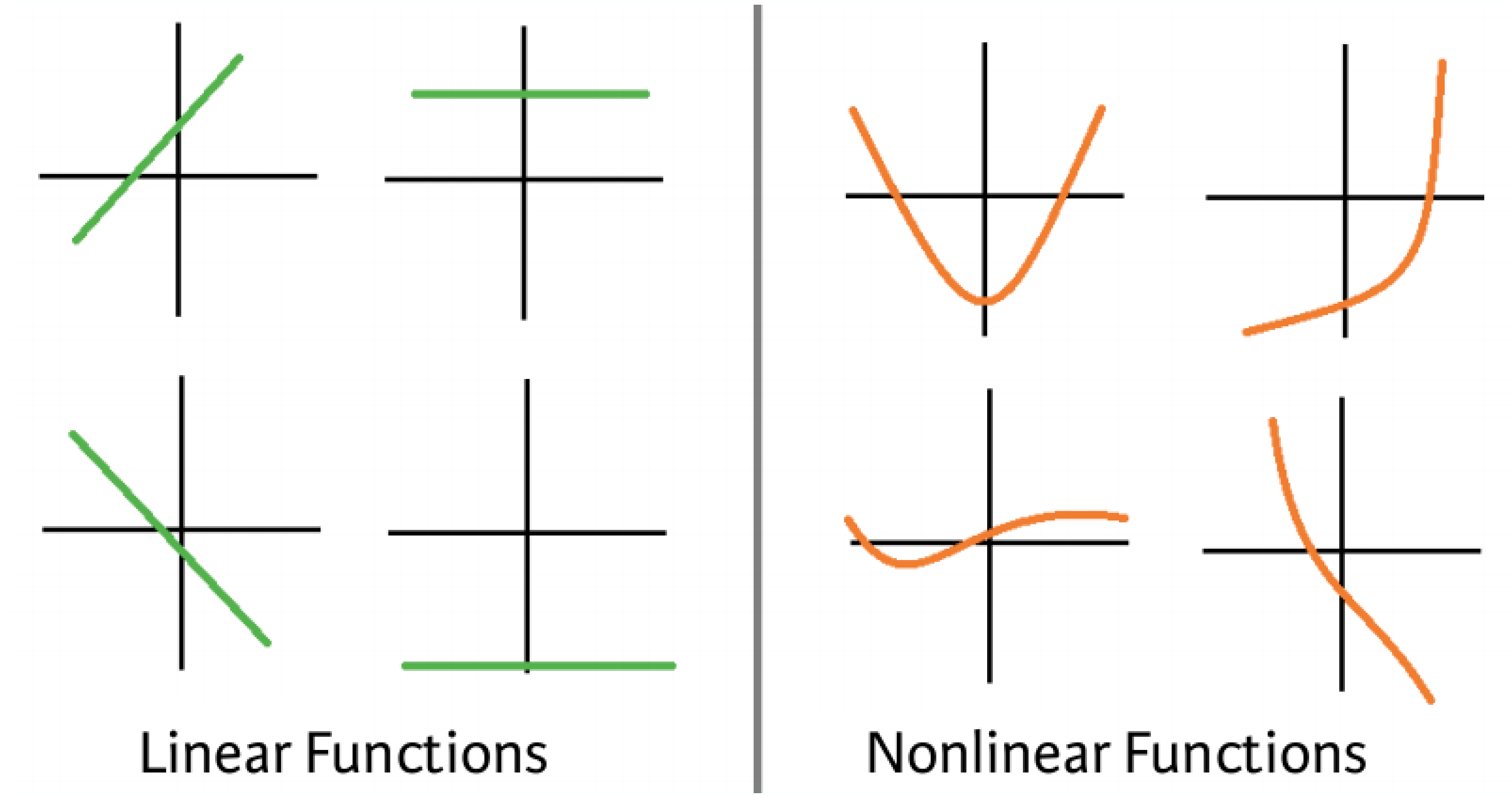



```
In [6]: hidden_layer_values = np.array([node_0_value, node_1_value])

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

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

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

Activation functions

Activation functions

Linear vs Nonlinear Functions

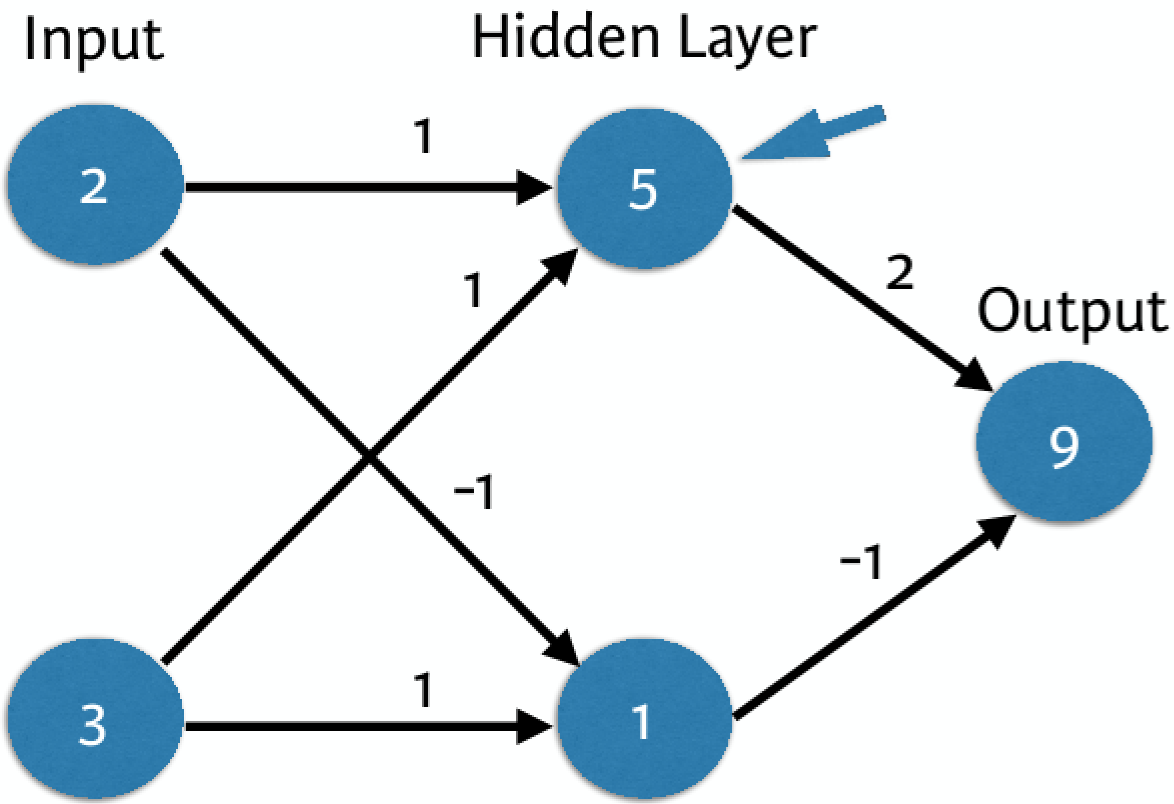


Activation functions

Applied to node inputs to produce node output

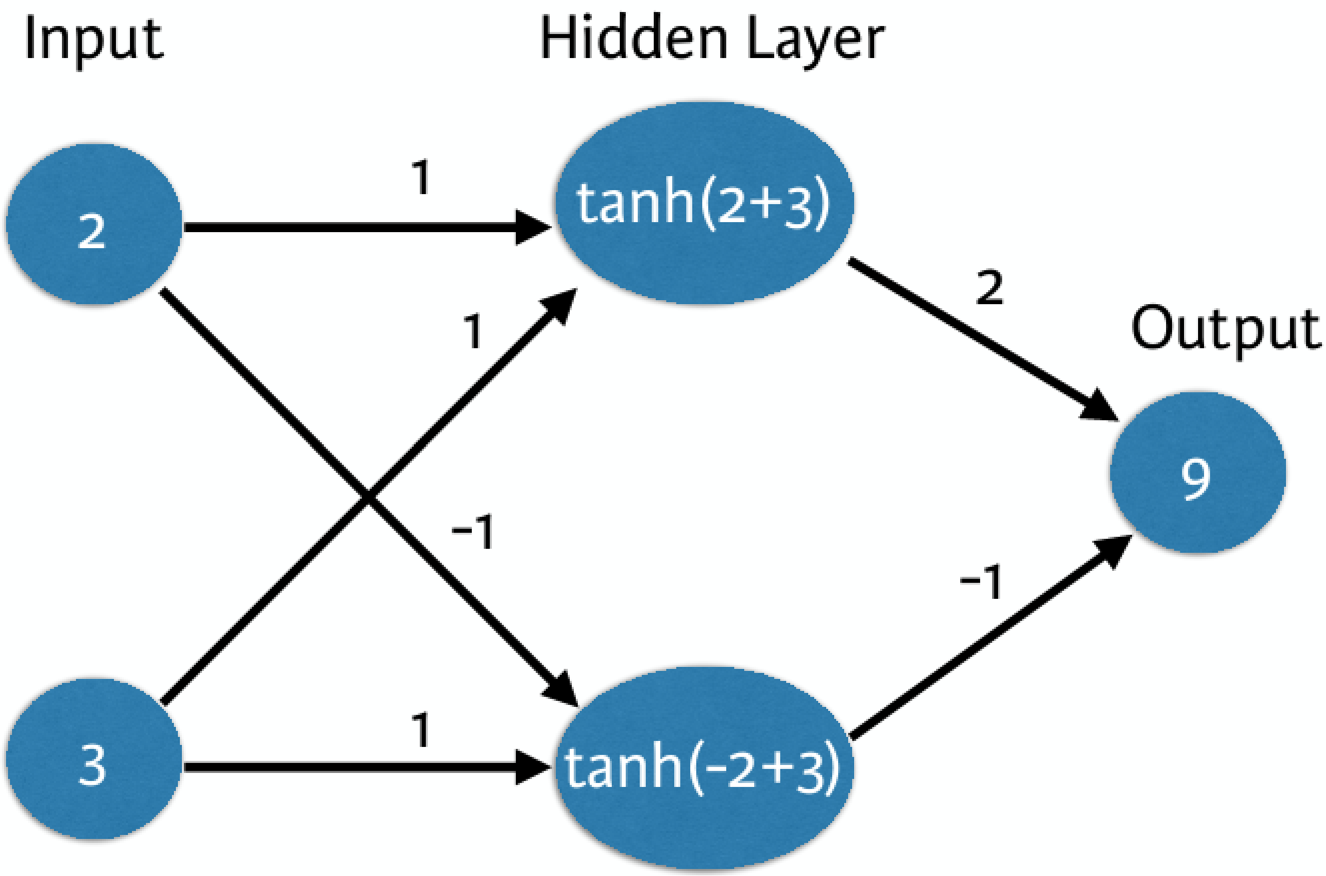
Improving our neural network w/ activation functions

Applied to node inputs to produce node output



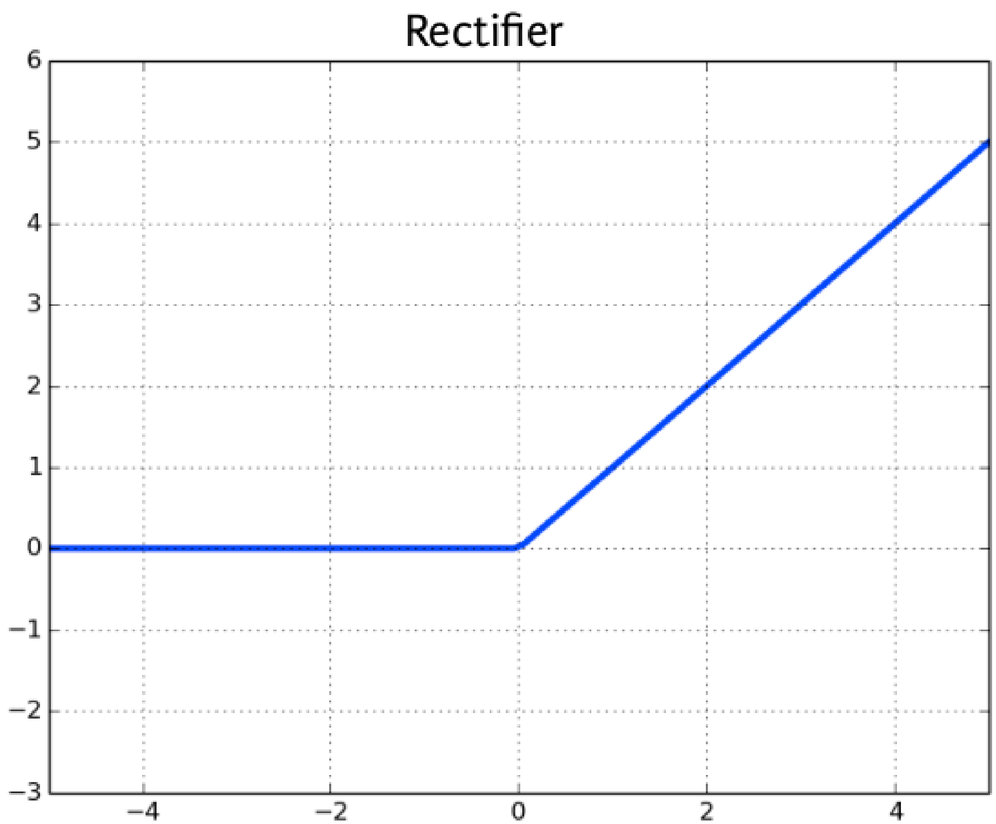
Improving our neural network w/ activation functions

Applied to node inputs to produce node output



ReLU (Rectified Linear Activation)

Applied to node inputs to produce node output

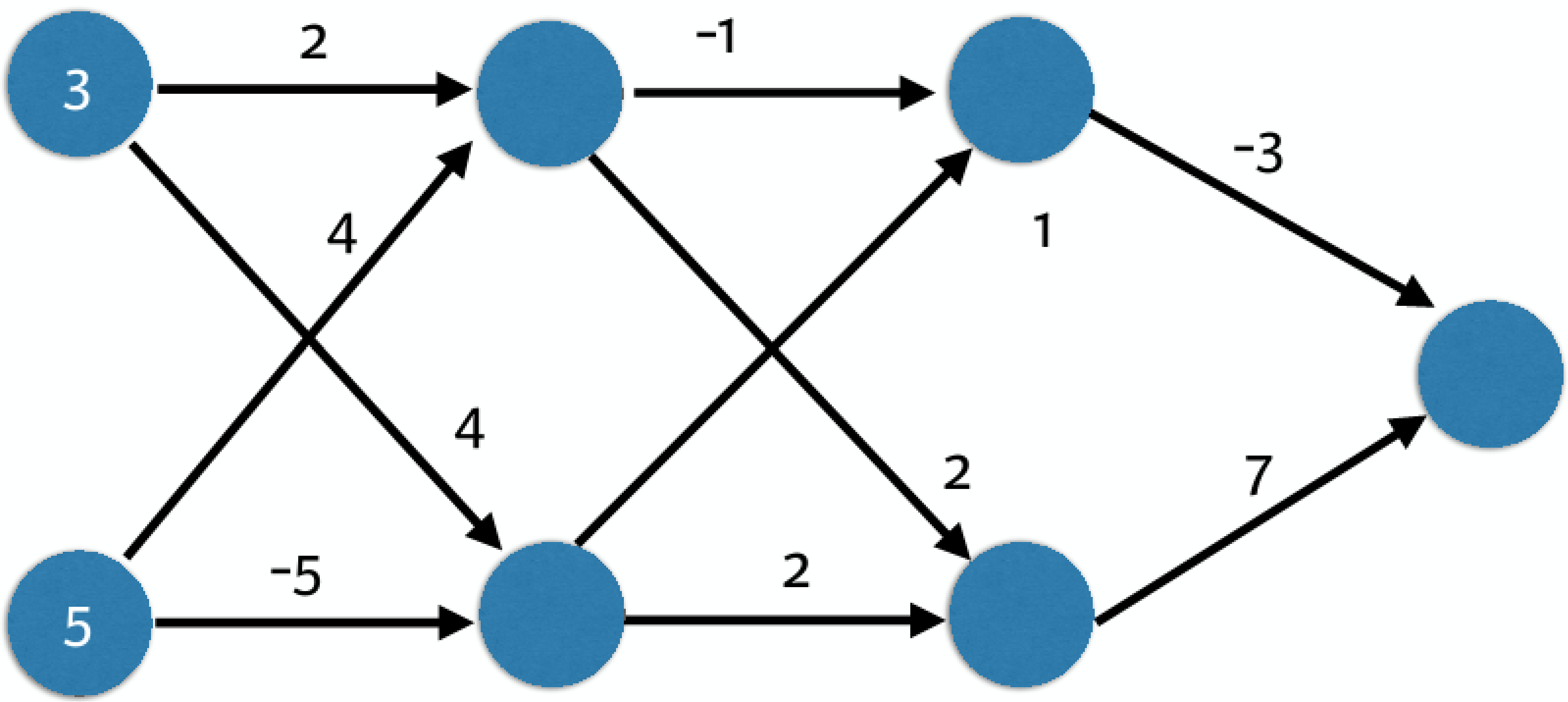


$$RELU(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; if \; x >= 0 \end{cases}$$

Activation functions

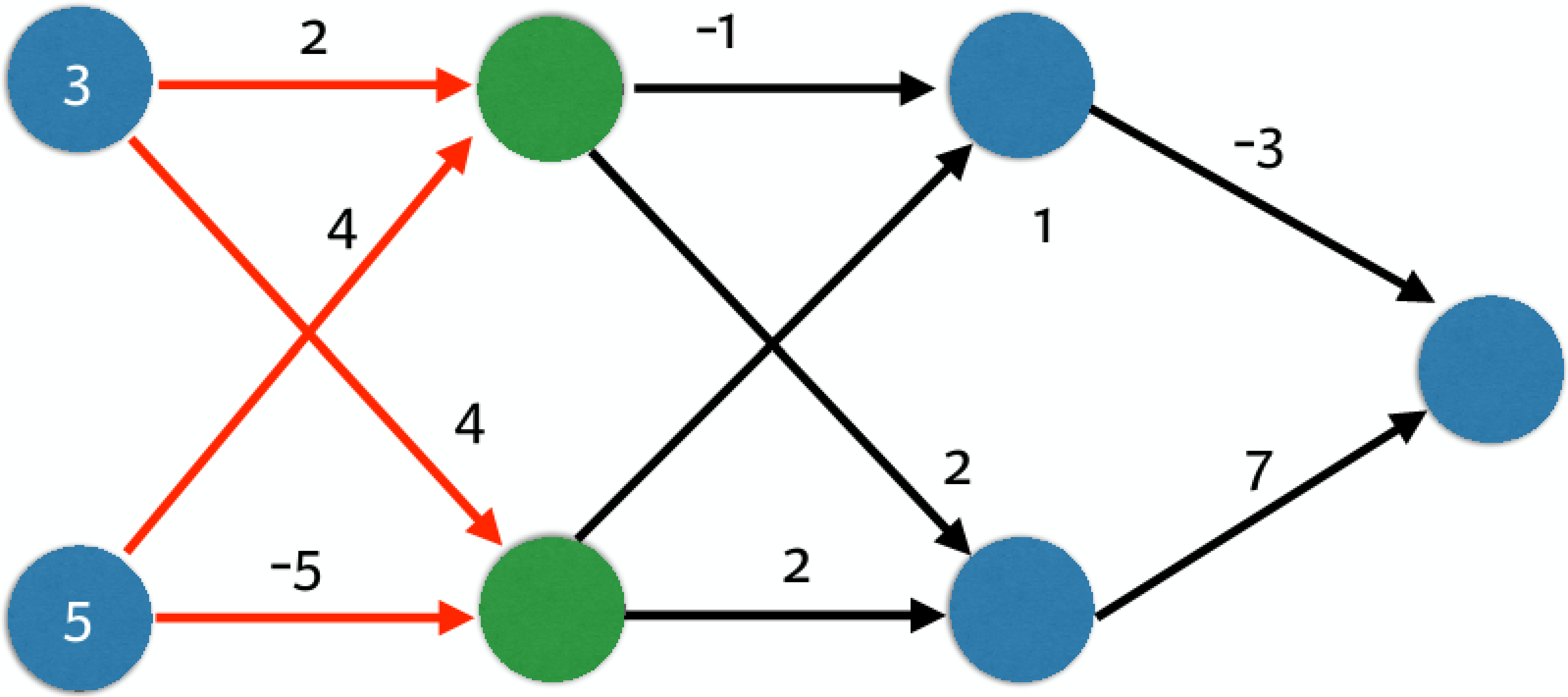
```
In [1]: import numpy as np

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)
1.2382242525694254
```

Deeper networks



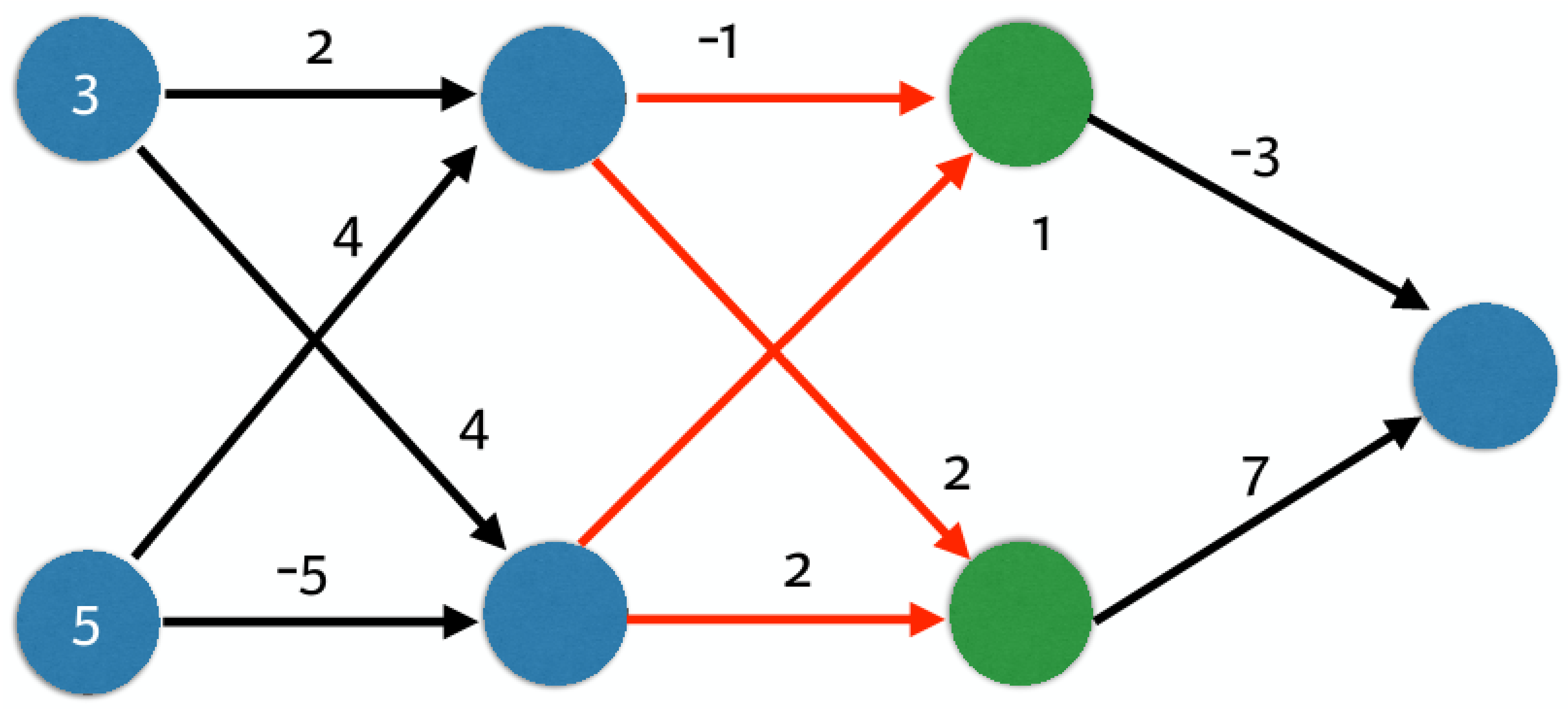
Deeper networks

Multiple hidden layers. Calculate with ReLU Activation Function



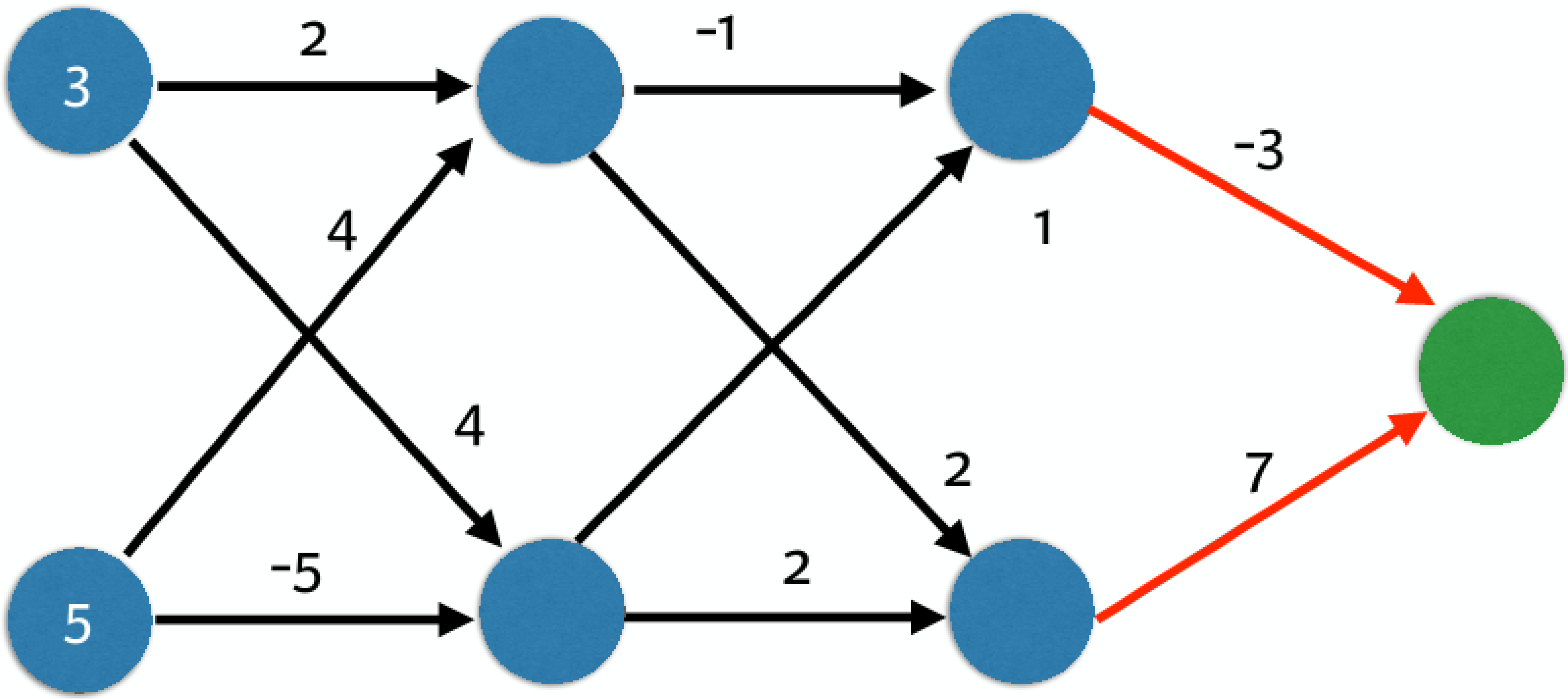
Deeper networks

Multiple hidden layers. Calculate with ReLU Activation Function



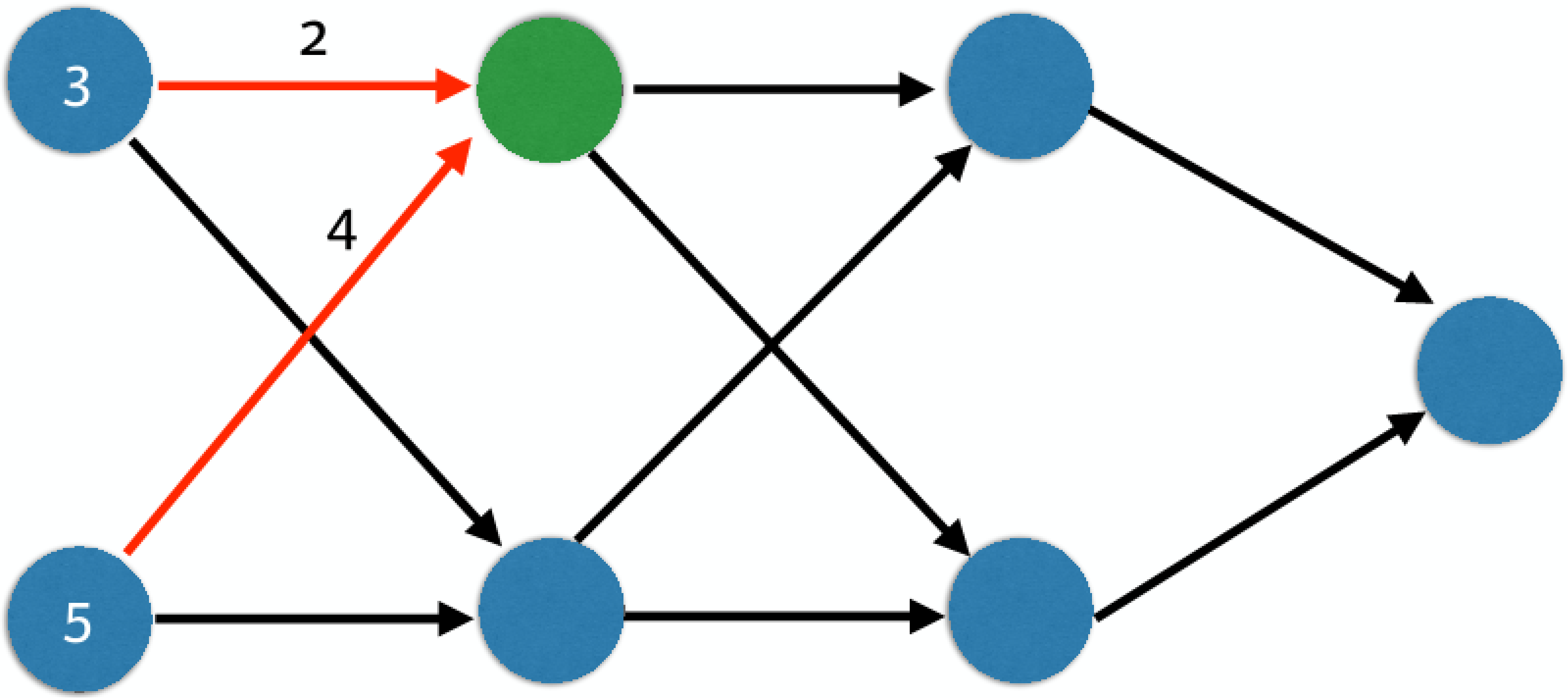
Deeper networks

Multiple hidden layers. Calculate with ReLU Activation Function



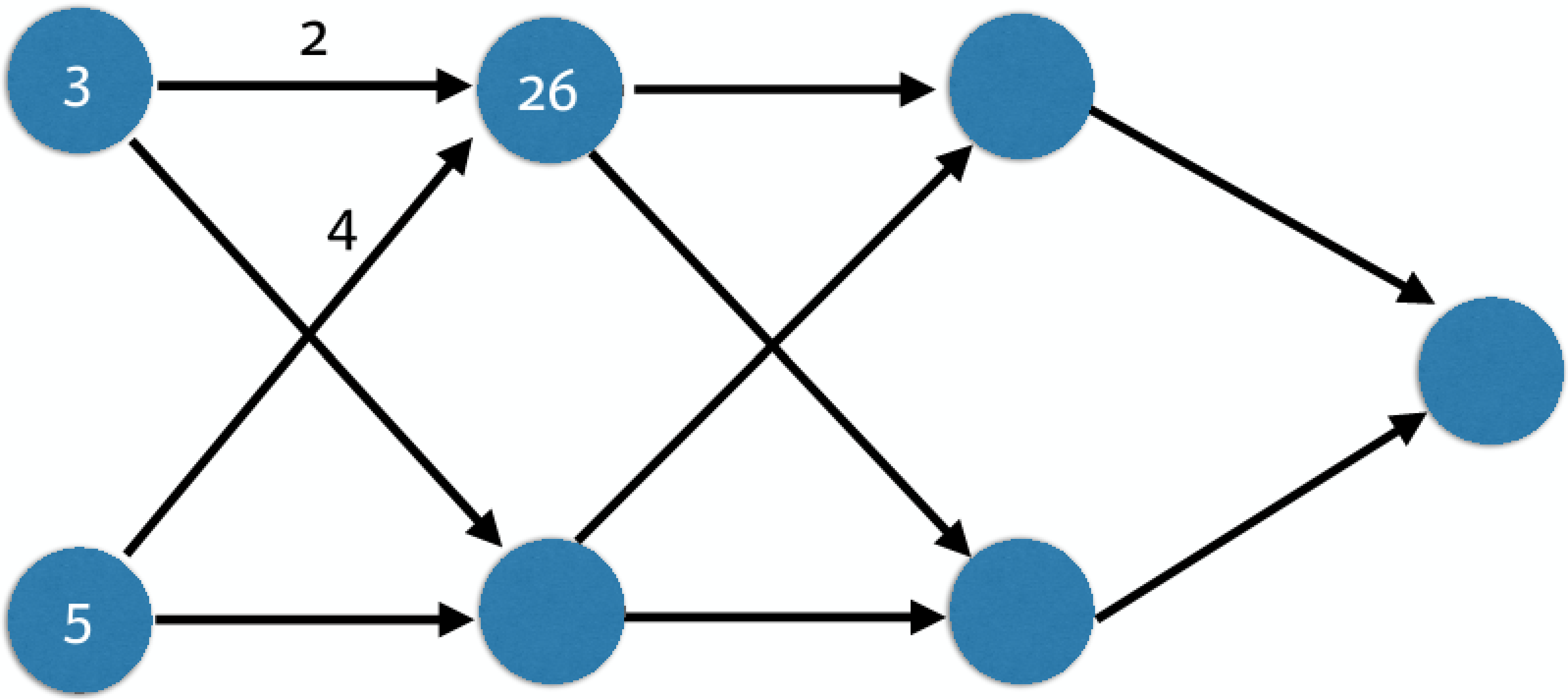
Deeper networks

Multiple hidden layers. Calculate with ReLU Activation Function



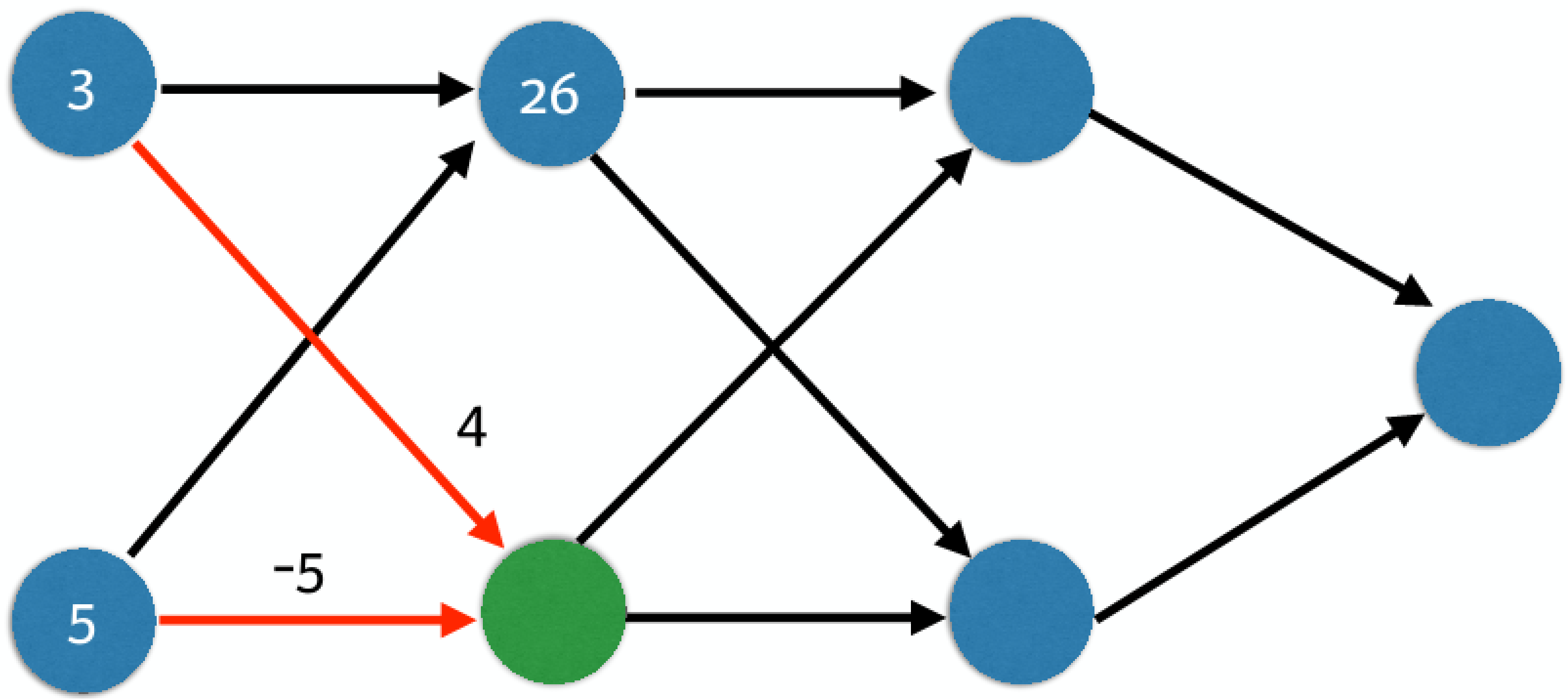
Deeper networks

Multiple hidden layers. Calculate with ReLU Activation Function



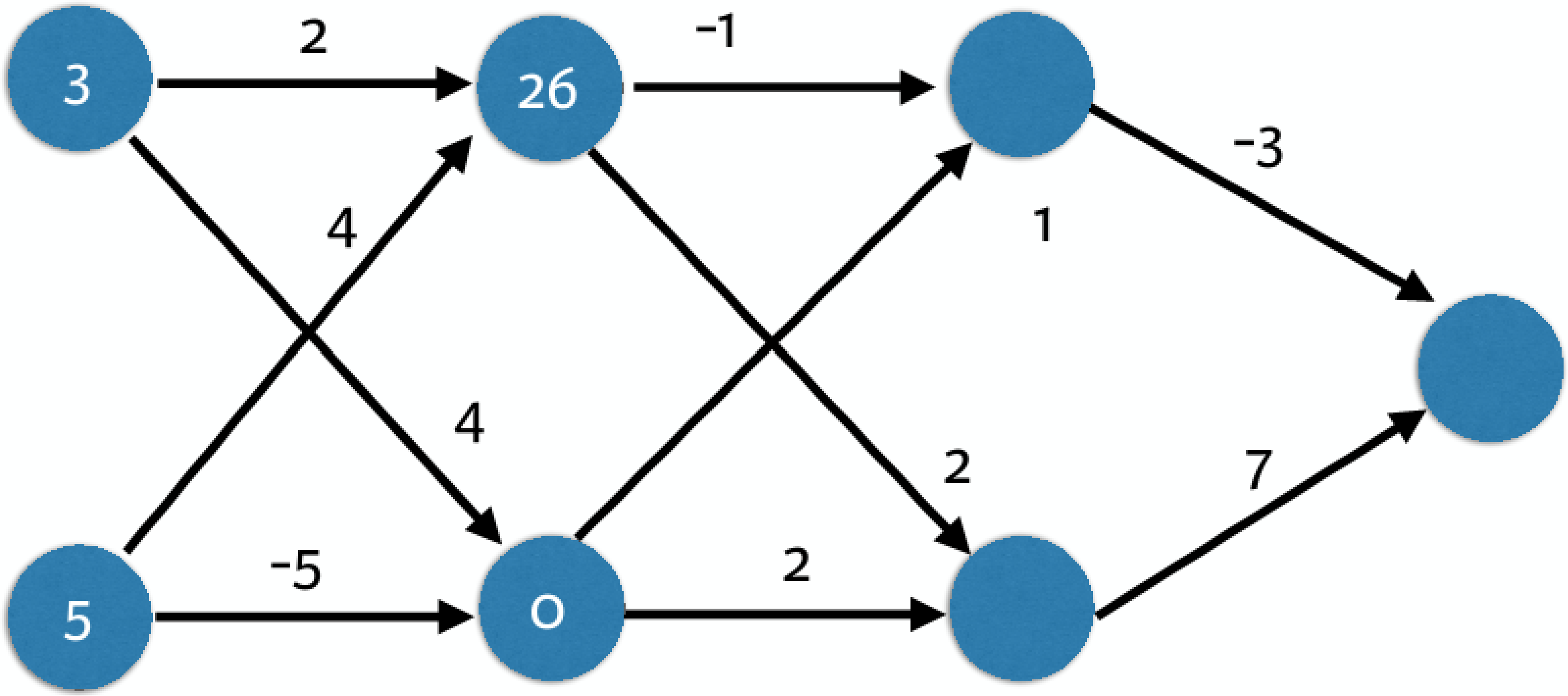
Deeper networks

Multiple hidden layers. Calculate with ReLU Activation Function



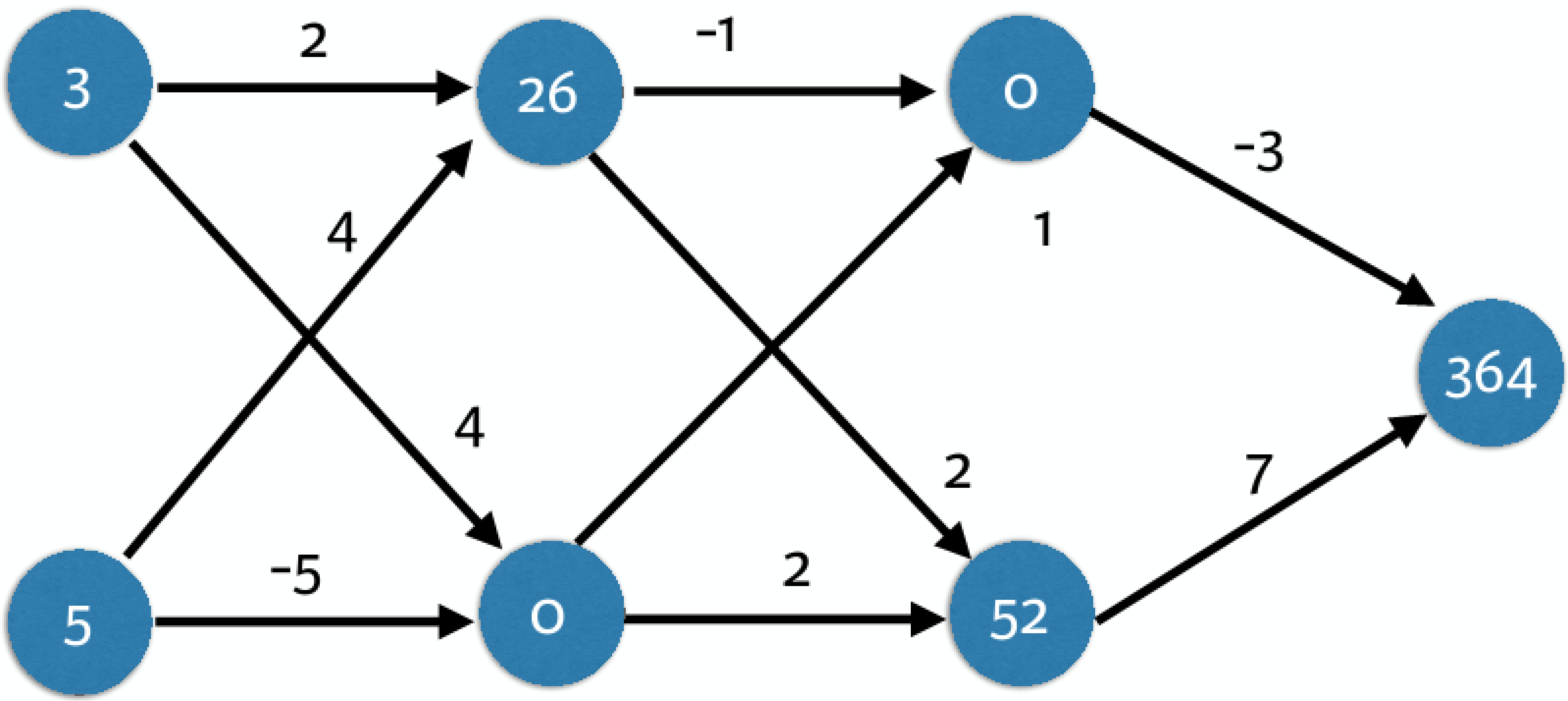
Deeper networks

Multiple hidden layers. Calculate with ReLU Activation Function



Deeper networks

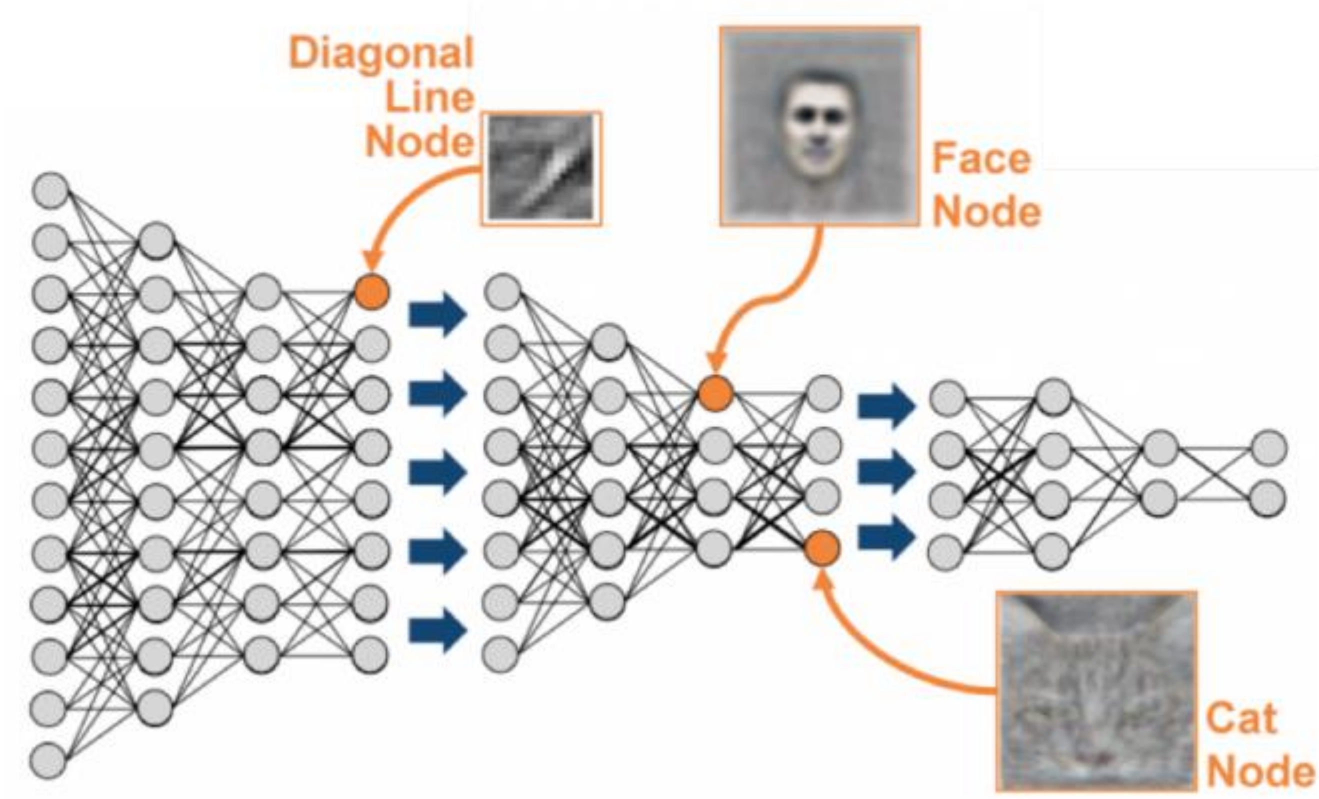
Multiple hidden layers. Calculate with ReLU Activation Function



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

To be continued,

Thanks!