Artificial Intelligence, Neural Networks, and You

Lovingly by: Justin Serowik

What is Artificial Intelligence?

- No real clear definition because we have trouble even defining intelligence
- How do you tell if something is artificially intelligent?
  - Artificial is easy - Not us (Humans)
  - Turing test?
  - Does it exist yet?

A.I. History

- Ideas of computers come from antiquity
- Alan Turing published “Computing Machinery and Intelligence” in 1950
- The Logic Theorist. Designed by Newell and Simon in 1955 may be considered the first AI program
  - First program deliberately engineered to mimic the problem solving skills of a human being
- A.I. Winters
  - First: 1974 - early 1980s - Couldn’t deliver on the hype
  - Second: 1987 - 1990s (maybe 2000s) - Collapse of the Lisp machine market and low interest
- Modern Times
  - More intense graphical computation leads to better Graphical Processing Units (GPUs)
  - Old Neural Network research reemerges, and finds that GPUs are efficient in computing them
  - Rise of the Internet and a **LOT** of data, which can be used for training

Artificial Neural Networks

- In 1958 Frank Rosenblatt created the perceptron, an algorithm for pattern recognition based on a two-layer computer learning network
- Computationally Intensive to set up a useful network
  - Unless you have massive parallel processing capabilities, e.g. GPU
- Canadians and American mathematicians and computer scientists expanded on his work, even without interest
- With the advent of distributed computing, neural networks were manageable

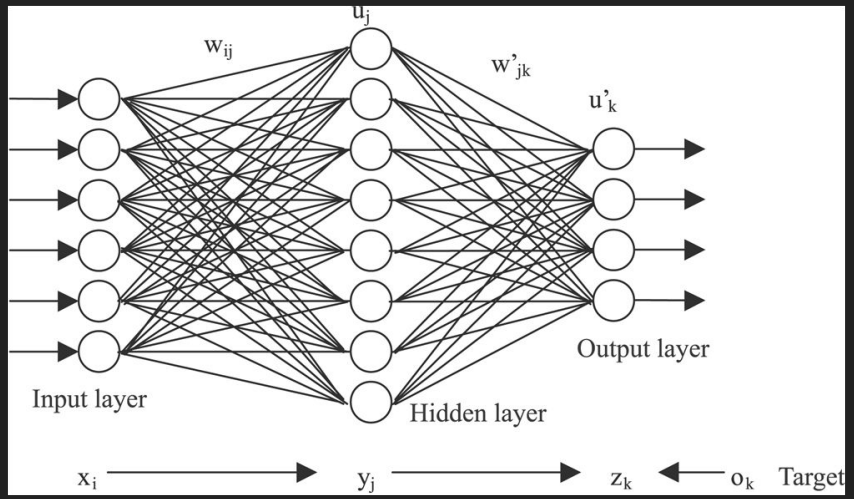
Why are they important?

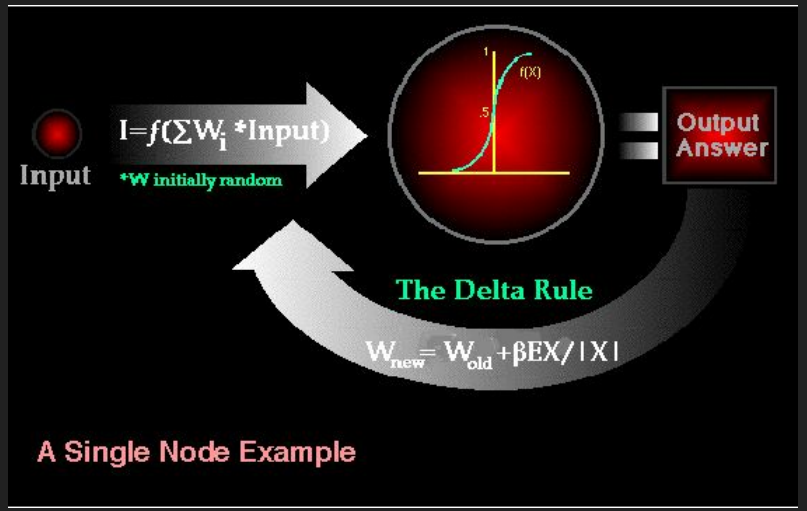
- Artificial Neural Networks are modeled after neurons in your brain
  - They function very similarly in that they must be “trained”
- Excellent at pattern recognition (similar to how humans are)
- Very flexible in the applications they can be used for
- Responsible for the majority of A.I. or “smart” features in your devices and computers today, along with a long list of previously “human” tasks
  - Facebook, Google, Siri
  - Solving Big Data problems such as income inequality, healthcare, and even education

How do Neural Networks work?

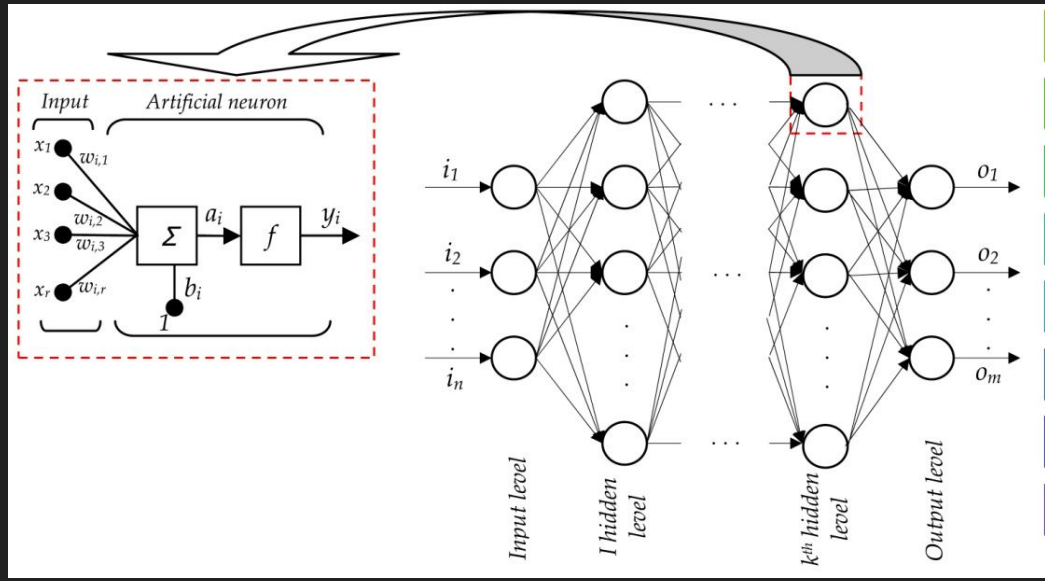
- Made of **layers** of **neurons** (nodes or units) connected to each other
- Node has a **weight** on the connection between layers
- Using the weight, calculate the result to be passed into the next layer/output

$$\sum weight_i \cdot input_i = weight1 \cdot input1 + weight2 \cdot input2 + weight3 \cdot input3$$





Some Enlightening Graphics



1. Assign Random weights to all the linkages to start the algorithm
2. Using the inputs and the (Input ->Hidden node) linkages find the activation rate of Hidden Nodes
3. Using the activation rate of Hidden nodes and linkages to Output, find the activation rate of Output Nodes
4. Find the error rate at the output node and recalibrate all the linkages between Hidden Nodes and Output Nodes
5. Using the Weights and error found at Output node, cascade down the error to Hidden Nodes
6. Recalibrate the weights between hidden node and the input nodes
7. Repeat the process till the convergence criterion is met
8. Using the final linkage weights score the activation rate of the output nodes

Let’s Make One

- Open a new Python file/ instance/ Jupyter, and import numpy

```
from numpy import exp, array, random, dot
```

- Create a new class and seed new random number generator

```
class NeuralNetwork():
    def __init__(self):
        # Seed the random number generator, so it generates the same numbers
        # every time the program runs.
        random.seed(1)

        # We model a single neuron, with 3 input connections and 1 output connection.
        # We assign random weights to a 3 x 1 matrix, with values in the range -1 to 1
        # and mean 0.
        self.synaptic_weights = 2 * random.random((3, 1)) - 1
```

The Sigmoid Function

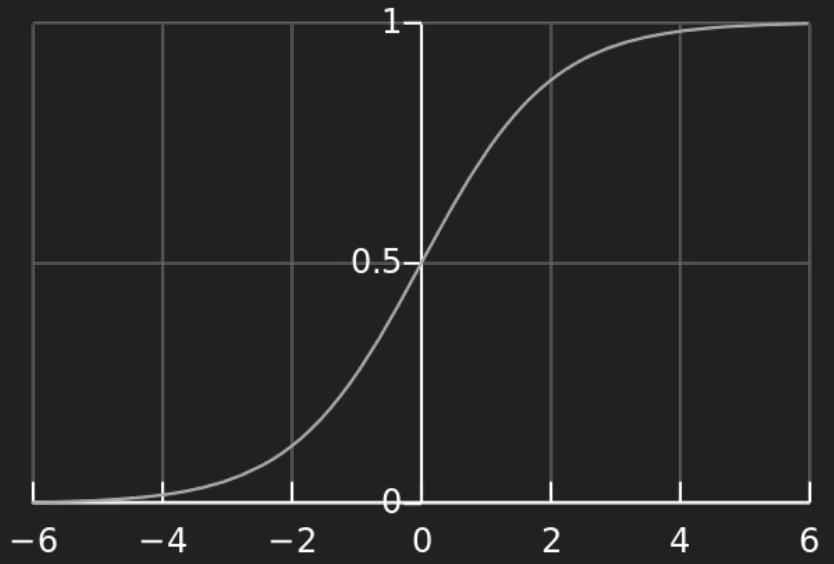
- Need to normalize our neuron output (so the result is between 1 and 0)

$$\sum weight_i \cdot input_i = weight1 \cdot input1 + weight2 \cdot input2 + weight3 \cdot input3$$

Sigmoid function = $\frac{1}{1 + e^{-x}}$

$$\text{Output of neuron} = \frac{1}{1 + e^{-(\sum weight_i input_i)}}$$

```
18      def __sigmoid(self, x):
19          return 1 / (1 + exp(-x))
20
21      def __sigmoid_derivative(self, x):
22          return x * (1 - x)
```



Training the Network

- We now “train” our network with our values (what values we’ll be using are covered in a later slide)

```
    # We train the neural network through a process of trial and error.
    # Adjusting the synaptic weights each time.
    def train(self, training_set_inputs, training_set_outputs, number_of_training_iterations):
        for iteration in xrange(number_of_training_iterations):
            # Pass the training set through our neural network (a single neuron).
            output = self.think(training_set_inputs)

            # Calculate the error (The difference between the desired output
            # and the predicted output).
            error = training_set_outputs - output

            # Multiply the error by the input and again by the gradient of the Sigmoid curve.
            # This means less confident weights are adjusted more.
            # This means inputs, which are zero, do not cause changes to the weights.
            adjustment = dot(training_set_inputs.T, error * self.__sigmoid_derivative(output))

            # Adjust the weights.
            self.synaptic_weights += adjustment
```

The Network thinks

- Define the think function which we will pass values into if we want a prediction

```
44      def think(self, inputs):
45          # Pass inputs through our neural network (our single neuron).
46          return self.__sigmoid(dot(inputs, self.synaptic_weights))
```

- Initialize a Neural Network

```
49  if __name__ == "__main__":
50
51      #Intialize a single neuron neural network.
52      neural_network = NeuralNetwork()
53
54      print "Random starting synaptic weights: "
55      print neural_network.synaptic_weights
```

Our Training

- Our first example we only care about the first two columns (A AND B) to determine our output

```
60      # The training set. We have 4 examples, each consisting of 3 input values
61      # and 1 output value. Here we have a pattern of (A AND B) for [A, B, C|
62      training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1], [1, 0, 0]])
63      training_set_outputs = array([[0, 1, 0, 0, 0]]).T
```

- Input the training data into network

```
68      # Train the neural network using a training set.
69      # Do it 10,000 times and make small adjustments each time.
70      neural_network.train(training_set_inputs, training_set_outputs, 10000)
```

A	B	C	Out
0	0	1	0
1	1	1	1
1	0	1	0
0	1	1	0
1	0	0	0

Results

- We test the network by letting it consider a brand new case

```
#Our test values
test_array = [1, 1, 0]

# Test the neural network with a new situation. Should be clode to 1
print "Considering new situation: "
print test_array
print "Result: "
print neural_network.think(array(test_array))
```

- The neural network outputs a value based on its “confidence” in the right answer, so in this case either a value closer to 1 or closer to 0
  - So here the correct answer is 1, and the NN outputs 0.99281112

```
Considering new situation:
[1, 1, 0]
Result:
[ 0.99281112]
```

A	B	C	Out
0	0	1	0
1	1	1	1
1	0	1	0
0	1	1	0
1	0	0	0
1	1	0	1

What Now?

- Since we verified that our Neural Network is fairly accurate, we don’t need to change anything and can rely on the trained NN
- You can adjust training the training cycles to improve accuracy, but also increase the amount of time it takes to compute an output.
  - After 100,000 cycles:

```
Considering new situation:
[1, 1, 0]
Result:
[ 0.99775321]
```

- For more complex problems, you need to add more layers to the network
  - We only used a 1 layer network for this demo
- Depending on how you connect these layers, you can achieve different types of neural networks and adjust them for your needs

Future of A.I.

- Stevens offers a Data Engineering Concentration in the ECE Department
  - Future Graduate certificate in Deep Learning
- Great Python Libraries for Neural Networks (Many more available than listed)
  - Theano: Can utilize GPUs using built in libraries
  - Keras: Sits on top of Theano and simplifies the syntax and functionality
- A Third A.I. Winter is unlikely as long as expectations are realistic
  - Plenty of great research happening in improving the field
  - “Smart” and Image Processing technologies will continue to need neural networks
  - Up to all of you to keep research and innovation alive in Artificial Intelligence

Citations

- http://www.extremetech.com/wp-content/uploads/2015/07/NeuralNetwork.png
- https://en.wikipedia.org/wiki/Artificial_intelligence
- https://medium.com/technology-invention-and-more/how-to-build-a-simple-neural-network-in-9-lines-of-python-code-cc8f23647ca1
- https://www.analyticsvidhya.com/blog/2014/10/ann-work-simplified/
- https://www.analyticsvidhya.com/blog/2016/03/introduction-deep-learning-fundamentals-neural-networks/
- http://deeplearning.net/software/theano/
- https://keras.io/