2020 / 2021

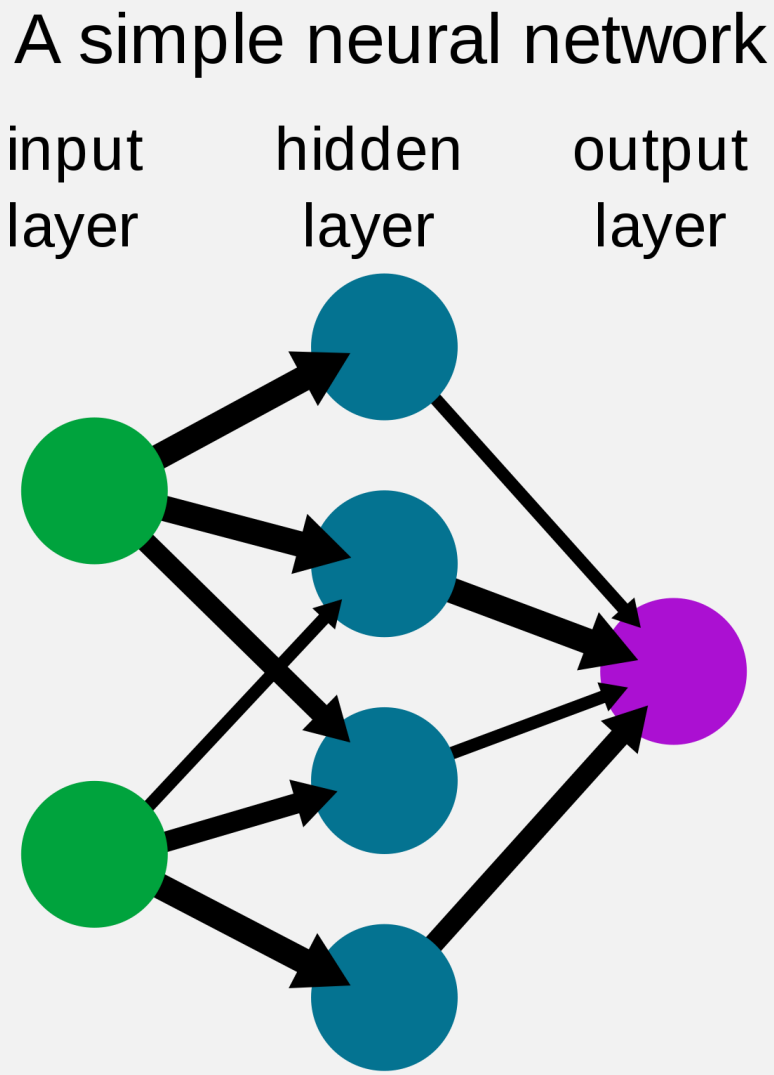
(Neural networks) Domain of AI

Prepared by :

Abdedaiem Amin

What is Neural networks?

Neural networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data. They are used in a variety of applications in financial services, from forecasting and marketing research to fraud detection and risk assessment.



HISTORICAL SIDE OF NN:

Warren McCulloch and Walter Pitts (1943) opened the subject by creating a computational model for neural networks. In the late 1940s, D. O. Hebb created a learning hypothesis based on the mechanism of neural plasticity that became known as Hebbian learning. Farley and Wesley A. Clark (1954) first used computational machines, then called "calculators", to simulate a Hebbian network. Rosenblatt (1958)

created the perceptron. The first functional networks with many layers were published by Ivakhnenko and Lapa in 1965, as the Group Method of Data Handling. The basics of continuous backpropagation were derived in the context of control theory by Kelley in 1960 and by Bryson in 1961, using principles of dynamic programming...

Components of ANNs:

Neurons:

ANNs are composed of artificial neurons which are conceptually derived from biological neurons. Each artificial neuron has inputs and produces a single output which can be sent to multiple other neurons. The inputs can be the feature values of a sample of external data, such as images or documents, or they can be the outputs of other neurons. The outputs of the final output neurons of the neural net accomplish the task, such as recognizing an object in an image.

To find the output of the neuron, first we take the weighted sum of all the inputs, weighted by the weights of the connections from the inputs to the neuron. We add a bias term to this sum. This weighted sum is sometimes called the activation. This weighted sum is then passed through a (usually nonlinear) activation function to produce the output. The initial inputs are external data, such as images and documents. The ultimate outputs accomplish the task, such as recognizing an object in an image.

Connections and weights:

The network consists of connections, each connection providing the output of one neuron as an input to another neuron. Each connection is assigned a weight that represents its relative importance. A given neuron can have multiple input and output connections.

Propagation function:

The propagation function computes the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum. A bias term can be added to the result of the propagation.

MIT report says about NN:

{

In the past 10 years, the best-performing artificial-intelligence systems — such as the speech recognizers on smartphones or Google's latest automatic translator — have resulted from a technique called "deep learning."

Deep learning is in fact a new name for an approach to artificial intelligence called neural networks, which have been going in and out of fashion for more than 70 years. Neural networks were first proposed in 1944 by Warren McCullough and Walter Pitts, two University of Chicago researchers who moved to MIT in 1952 as founding members of what's sometimes called the first cognitive science department.

Neural nets were a major area of research in both neuroscience and computer science until 1969, when, according to computer science lore, they were killed off by the MIT mathematicians Marvin Minsky and Seymour Papert, who a year later would become co-directors of the new MIT Artificial Intelligence Laboratory.

}

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.

Example for face recognition made by Abdedaiem amin:

https://github.com/amincoding/simple_face_recognition_by_amin

HOW TO BUILD A NN FROM SCRATCH:

Creating a Neural Network class in Python is easy.

```
class
NeuralNetwork:
                  def __init__(self, x, y):
                      self.input      = x
                      self.weights1   = np.random.rand(self.input.shape[1],4)
                      self.weights2   = np.random.rand(4,1)
                      self.y          = y
                      self.output     = np.zeros(y.shape)
```

Training the Neural Network

The output $\hat{y}$ of a simple 2-layer Neural Network is:

$$\hat{y} = \sigma(W_2\sigma(W_1 x + b_1) + b_2)$$

You might notice that in the equation above, the weights W and the biases b are the only variables that affects the output ŷ.

Naturally, the right values for the weights and biases determines the strength of the predictions. The process of fine-tuning the weights and biases from the input data is known as training the Neural Network.

Each iteration of the training process consists of the following steps:

- Calculating the predicted output ŷ, known as feedforward
- Updating the weights and biases, known as backpropagation

The sequential graph below illustrates the process

Let's add a feedforward function in our python code to do exactly that. Note that for simplicity, we have assumed the biases to be 0.

```
class
NeuralNetwork:
                def __init__(self, x, y):
                    self.input      = x
                    self.weights1   = np.random.rand(self.input.shape[1],4)
                    self.weights2   = np.random.rand(4,1)
                    self.y          = y
                    self.output     = np.zeros(self.y.shape)

                def feedforward(self):
                    self.layer1 = sigmoid(np.dot(self.input, self.weights1))
                    self.output = sigmoid(np.dot(self.layer1, self.weights2))
```

However, we still need a way to evaluate the "goodness" of our predictions (i.e. how far off are our predictions)? The Loss Function allows us to do exactly that.

Now that we have that, let's add the backpropagation function into our python code.

```
class
NeuralNetwork:
                def __init__(self, x, y):
                    self.input      = x
                    self.weights1   = np.random.rand(self.input.shape[1],4)
                    self.weights2   = np.random.rand(4,1)
                    self.y          = y
                    self.output     = np.zeros(self.y.shape)

                def feedforward(self):
                    self.layer1 = sigmoid(np.dot(self.input, self.weights1))
                    self.output = sigmoid(np.dot(self.layer1, self.weights2))

                def backprop(self):
                    # application of the chain rule to find derivative of the loss
            function with respect to weights2 and weights1
                    d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) *
            sigmoid_derivative(self.output)))
                    d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) *
            sigmoid_derivative(self.output), self.weights2.T) *
            sigmoid_derivative(self.layer1)))

                    # update the weights with the derivative (slope) of the loss function
                    self.weights1 += d_weights1
                    self.weights2 += d_weights2
```

And now u have a NN ready to rock and to be fit with any type and kind of data to be trained for

THE END .

List OF SOURCES:

https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

http://neuralnetworksanddeeplearning.com

https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

https://en.wikipedia.org/wiki/Artificial_neural_network

https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=Neural%20networks%20are%20a%20series,fraud%20detection%20and%20risk%20assessment.