Introduction to Artificial Neural Networks (ANNs)

Author, Kirill Goltsman

A Data Science Foundation White Paper

September 2017

---

www.datascience.foundation

Data Science Foundation
Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670

Introduction to Artificial Neural Networks (ANNs)

White Paper

5 September 2017

Introduction

Machine Learning (ML) is a subfield of computer science that stands behind the rapid development of Artificial Intelligence (AI) over the past decade. Machine Learning studies algorithms that allow machines recognizing patterns, construct prediction models, or generate images or videos through learning. ML algorithms can be implemented using a wide variety of methods like clustering, linear regression, decision trees, and more.

In this paper, we are going to discuss the design of Artificial Neural Networks (ANN) – a ML architecture that gathered a powerful momentum in the recent years as one of the most efficient and fast learning methods to solve complex computer vision, speech recognition, NLP (Natural Language Processing), image, audio, and video generation problems. Thanks to their efficient multilayer design that models the biological structure of human brain, ANNs have firmly established themselves as the state-of-the-art technology that drives AI revolution. In what follows, we are going to describe the architecture of a simple ANN and offer you a useful intuition of how it may be used to solve complex nonlinear problems in an efficient way.

What is an Artificial Neural Network?

An Artificial Neural Network is an ML (Machine Learning) algorithm inspired by biological

Data Science Foundation
Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670

computational models of brain and biological neural networks. In a nutshell, an Artificial Neural Network (ANN) is a computational representation of the human neural network that regulates human intelligence, reasoning and memory. However, why should we necessary emulate a human brain system to develop efficient ML algorithms?

The main rationale behind using ANNs (ANN) is that neural networks are efficient in complex computations and hierarchical representation of knowledge. Neurons connected by axons and dendrites into complex neural networks can pass and exchange information, store intermediary computation results, produce abstractions, and pide the learning process into multiple steps. Computation model of such system can thus produce very efficient learning processes similar to the biological ones.

A perceptron algorithm invented in 1957 by Franc Rosenblatt in 1957 was the first attempt to create a computational model of a biological neural network. However, complex neural networks with multiple layers, nodes, and neurons became possible only recently and thanks to the dramatic increase of computing power (Moore's Law), more efficient GPUs (Graphics Processing Units), and proliferation of Big Data used for training ML models. In the 2000s-2010s these developments gave rise to Deep Learning (DL), – a modern approach to the design of ANNs based on a deep cascade of multiple layers that extract features from data and do transformations and hierarchical representations of knowledge.

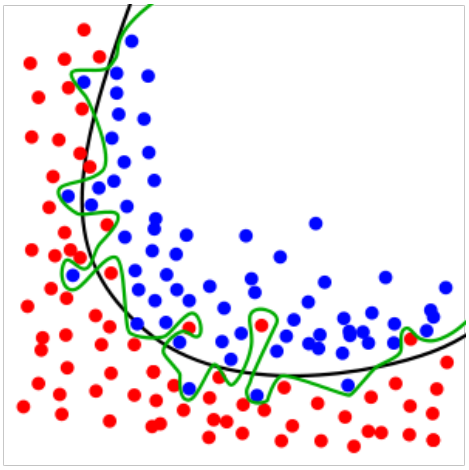


Image #1 Overfitting problem

Thanks to their ability to simulate complex nonlinear processes and create hierarchical, and abstract representations of data, ANNs stand behind recent breakthroughs in image recognition and computer vision, NLP (Natural Language Processing), generative models and various other ML applications that seek to retrieve complex patterns from data. Neural networks are especially useful for studying nonlinear hypotheses with many features (e.g

n=100). Constructing an accurate hypothesis for such a large feature space would require using multiple high-order polynomials which would inevitably lead to overfitting – a scenario in which the model describes the random noise in data rather than underlying relationships and patterns. The problem of overfitting is especially tangible in image recognition problems where each pixel may represent a feature. For example, when working with 50 X 50 pixel images, we may have 25000 features which would make manual construction of the hypothesis almost impossible.

A Simple Neural Network with a Single Neuron

The simplest possible neural network consists of a single "neuron" (see the diagram below). Using a biological analogy, this 'neuron' is a computational unit that takes inputs via (dendrites) as electrical inputs (let's say "spikes") and transmits them via axons to the next layer or the network's output.

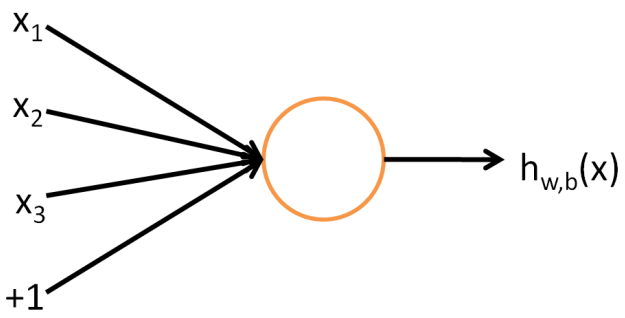


Image #2 A neural network with a single neuron

In a simple neural network depicted above, dendrites are input features (x_1, x_2 ...) and the outputs (axons) represent the results of our hypothesis ($h_{w,b}(x)$). Besides input features, the input layer of a neural network normally has a 'bias unit' which is equal to 1. A bias unit is needed to use a constant term in the hypothesis function.

In Machine Learning terms, the network depicted above has one input layer, one hidden layer (that consists of a single neuron) and one output layer. A learning process of this network is implemented in the following way. The input layer takes input features (e.g pixels) for each training sample and feeds them to the activation function that computes the hypothesis in the hidden layer.

An activation function is normally a logistic regression used for classification, however, other alternatives are also possible. In the case described above, our single neuron corresponds exactly to the input-output mapping that was defined by logistic regression.

$$y = \varsigma(x) = \frac{1}{1+e^{-x}}$$

Image #3 Logistic Regression

As in the case with simple binary classification, our logistic regression has parameters. They are often called “weights” in the ANN (Artificial Neural Network) models.

Multi-Layered Neural Network

To understand how neural networks work, we need to formalize the model and describe it in a real-world scenario. In the image below we can see a multilayer network that consists of three layers and has several neurons. Here, as in a single-neuron network, we have one input layer with three inputs (x_1,x_2,x_3) with an added bias unit (+1). The second layer of the network is a hidden layer consisting of three units/neurons represented by the activation functions. We call it a *hidden* layer because we don’t observe the values computed in it. Actually, a neural network can contain multiple hidden layers that pass complex functions and computations from the “surface” layers to the “bottom” of the neural network. The design of a neural network with many hidden layers is frequently used in Deep Learning (DL) – a popular approach in the ML research that gained a powerful momentum in recent years.

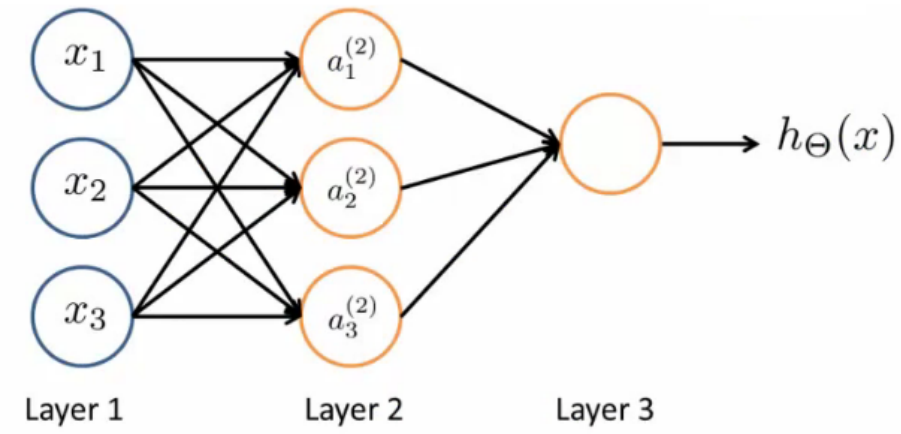


Image #4 Multilayer Perceptron

The hidden layer (Layer 2) above has three neurons (a_1^2, a_2^2, a_3^2). In abstract terms, each unit/neuron of a hidden layer aij is an activation of unit/neuron in in the layer j. In our case, a unit a_1^2 ctivates the first neuron of the second layer (hidden layer). By activation, we mean a value which is computed by the activation function (e.g logistic regression) in this layer and outputted by that node to the next layer.

Finally, Layer 3 is an output layer that gets results from the hidden layer and applies them to its own activation function. This layer computes the final value of our hypothesis. Afterwards, the cycle continues until the neural network comes up with the model and weights that best predict the values of the training data.

So far, we haven't defined how the 'weights' work in the activation functions. For that reason, let's define $Q^{(j)}$ as a matrix of parameters/weights that controls the function mapping from layer j to layer j + 1. For example, Q^1 will control the mapping from the input layer to the hidden layer, whereas Q^2 will control the mapping from the hidden layer to the output layer. The dimensionality of Q matrix will be defined by the following rule. If our network has s_j units in the layer j and s_{j+1} units in the layer j+1, then Q^j will have a dimension of s_{j+1} X (s_j + 1). The + 1 dimension comes from the necessary addition in Q^j of a bias unit x_0 and $Q_0^{(j)}$. In other words, our output nodes will not include the bias unit while the input nodes will.

To illustrate how the dimensionality of the Q matrix works, let's assume that we have two layers with 101 and 21 units in each. Then, using our rule Q^j would be a 21 X 102 matrix with 21 rows and 102 columns.

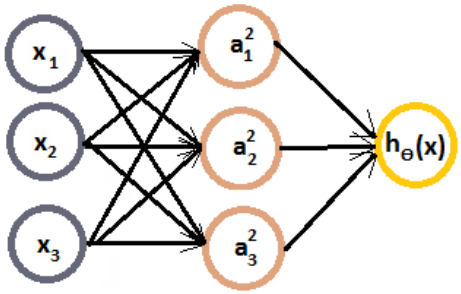


Image #5 A Neural Network Model

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Let's put it all together. In the image above, we see our neural network with three layers again. What we need to do, is to calculate activation functions based on the input values, and our main hypothesis function based on the set of calculations from the previous layer (the hidden layer). In this case, our neural network works as a cascade of calculations where each subsequent layer supplies values to the activation functions of the next one.

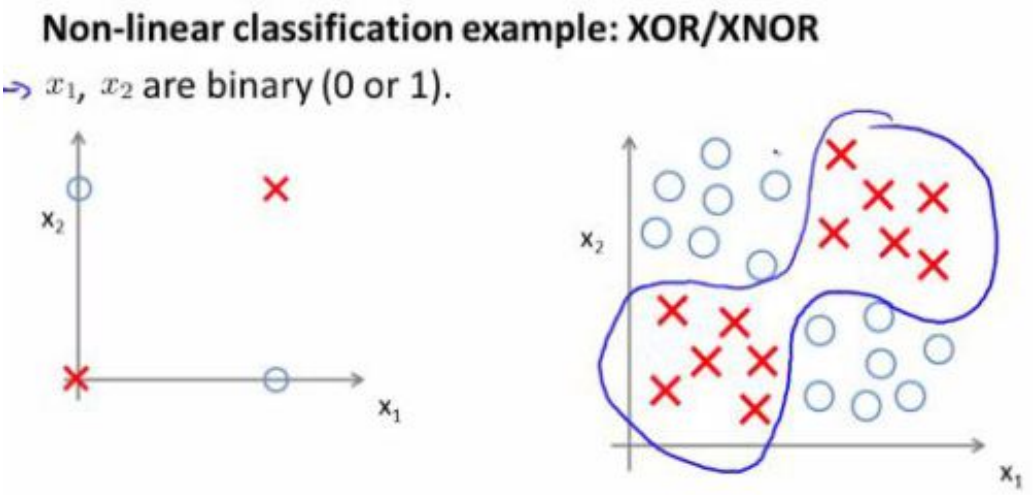
To calculate activations, we first have to define the dimensionality of our Q matrices. In this example, we have 3 input and 3 hidden units, so Q^1 mapping from input to hidden layer is of dimension 3 X 4 because the bias unit is included. The activation layer of each hidden neuron (e.g a_1^2) is equal to our sigmoid function applied to the linear combination of inputs with weights retrieved from the weight matrix Q^j. In the diagram above, you can see that each activation unit is computed by the function g which is our logistic regression function. In its turn, Q^2 refers to the matrix of weights that maps from the hidden layer to the output layer. These weights may be randomly assigned to the matrix before the neural network runs or be a product of previous computations. In our case, Q^2 is a 1 X 4 dimensional matrix (i.e a row vector). To calculate the output results we apply our hypothesis function (sigmoid function) to the results calculated by the activation functions in the hidden layer. If we had several hidden layers, then the results of the previous activation functions would be passed to the next hidden layer and then to the output layer.

This sequential mechanism makes neural networks very powerful in computation on nonlinear hypotheses and complex functions. Instead of trying to fit inputs to polynomial functions designed manually, we can create a neural network with numerous activation functions that exchange intermediary results and update weights. These automatic setup allows creating

nonlinear models that are more accurate in prediction and classification of our data.

Neural Networks in Action

The power of neural networks to compute complex nonlinear functions may be illustrated using the following binary classification example taken from Coursera Machine Learning course by Professor Andrew N_{gi}.



Consider the case when x_1 and x_2 can take two binary values (0,1). To put this binary classification problem in Boolean terms, our task is to compute y = x_1 XOR x_2 , which is the same as computing x_1 XNOR x_2. The latter is a logic gate that may be interpreted as NOT (x_1 XOR x_2). This is the same as saying that the function is true if both x_1 and x_2 are equal 0 or 1.

To make our network calculate XNOR, we first have to describe simple logical functions to be used as intermediary activations in the hidden layer. The first function we want to compute is a logical AND function: **y = x_1 AND x_2**.

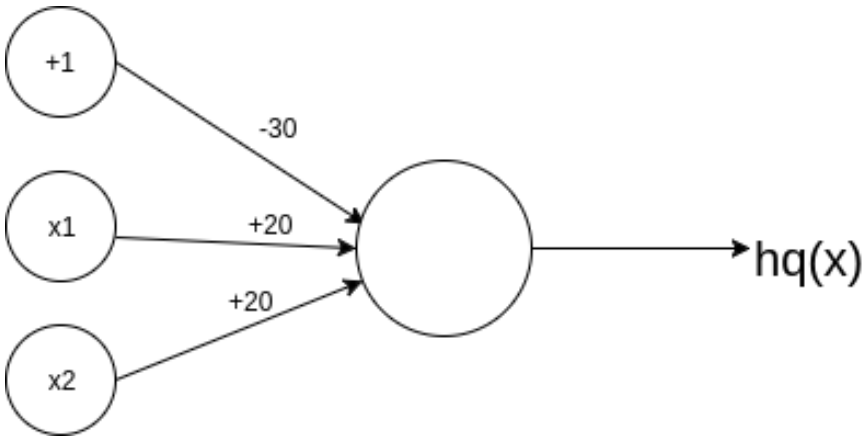


Image #6 Logical AND function

As in the first example above, our AND function is a simple single-neuron network with inputs x_1 and x_2 and a bias unit (+1). The first thing we need to do is to assign weights to the activation function and then compute it based on the input values specified in the truth table below. These input values are all possible binary values that x1 and x2 can take. By fitting 0s and 1s into the function (i.e logistic regression) we can compute our hypothesis.

$h_q(x) = g(-30 + 20x_1 + 20x_2)$.

To understand how the values of the third column of the truth table are found, remember that sigmoid function is 0 at ≈ -4.6 and 1 at ≈ 4.6. As a result, we have:

x_1	x_2	$h_q(x)$
0	0	g(-30) ≈ 0
0	1	g(-10) ≈ 0
1	0	g(-10) ≈ 0
1	1	g(10) ≈ 1

As we can see now, the rightmost column is a definition of a logical AND function that is true only if both x_1 and x_2 are true.

The second function we need for our neural network to work is a logical OR function. In the logical OR, y is true (1) if either x_1 OR x_2 or both of them are 1 (true).

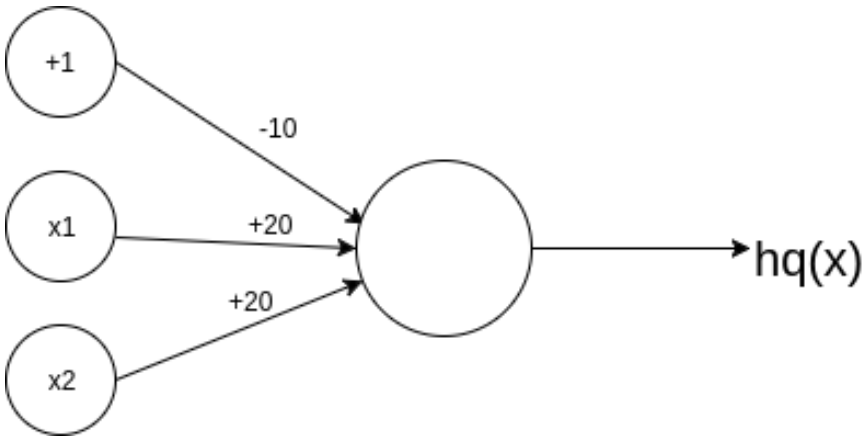


Image #7 Logical OR function

As in the previous case with the logical AND, we assign weights that will fit the definition of the logical OR function. Putting these weights into our logistic function $g(-10 + 20x_1 + 20x_2)$ we get the following truth table:

x_1	x_2	$h_q(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(-10) \approx 1$
1	1	$g(10) \approx 1$

As you see, our function is false (0) only if both x_1 and x_2 are false. In all other cases, it is true. This corresponds to the logical OR function.

The last function we need to compute before running a network for finding x_1 XNOR x_2 is (NOT x_1) and (NOT x_2). In essence, this function consists of two logical negations (NOT).

A single negation NOT x_1 may be presented in the following diagram. In essence, it says that y is true only if x_1 is false. Therefore, the logical NOT has only one input unit (x_1).

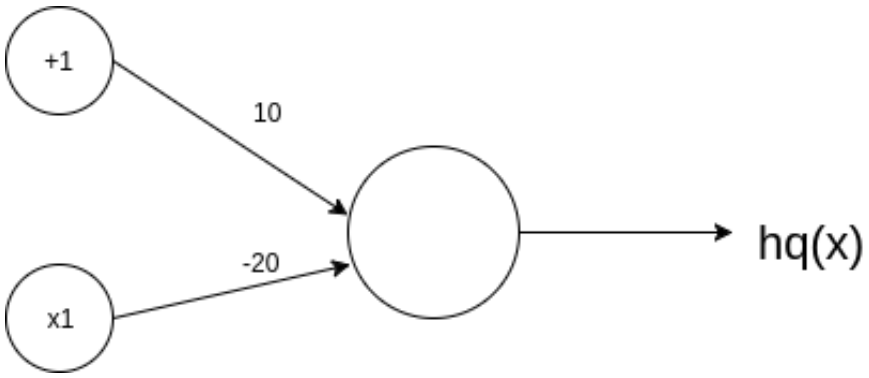


Image #8 Logical NOT

After putting inputs with weights into g = 10 – 20x_1, we end up with the following truth table.

x_1	$h_q(x)$
0	g(10) ≈ 1
1	g(-10) ≈ 0

The output values of this table confirm our hypothesis that NOT function outputs true only if x_1 is false. Now, we can find out values of the logical (NOT x_1) AND (NOT x_2) function.

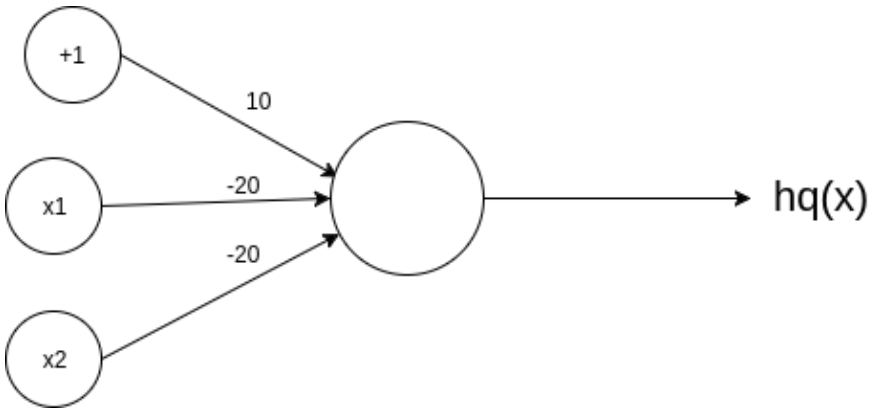


Image #9 Logical (NOT x_1) AND (NOT x_2)

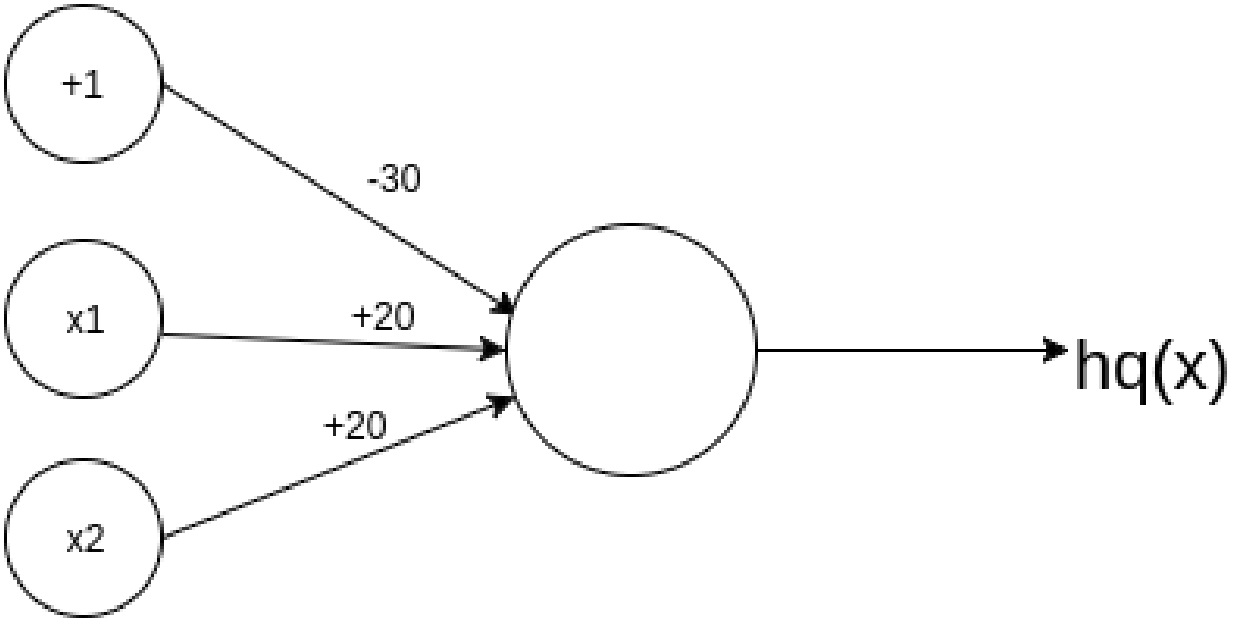
Putting binary values of x_1 and x_2 in the function g(10 - 20x_1 -20x_2) we end up with the following truth table.

x_1	x_2	$h_q(x)$
0	0	g(10) ≈ 1
0	1	g(-10) ≈ 0

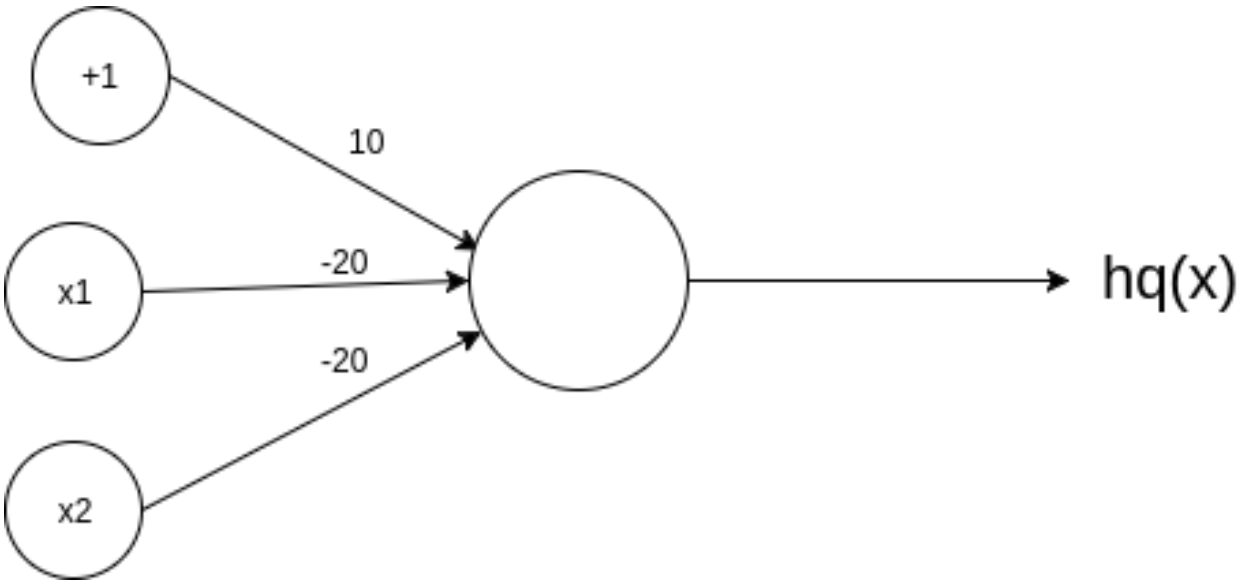
1 0 g(-10) ≈ 0
1 1 g(-30) ≈ 0

This table demonstrates that the logical (NOT x_1) AND (NOT x_2) function is true only if both x_1 and x_2 are false.

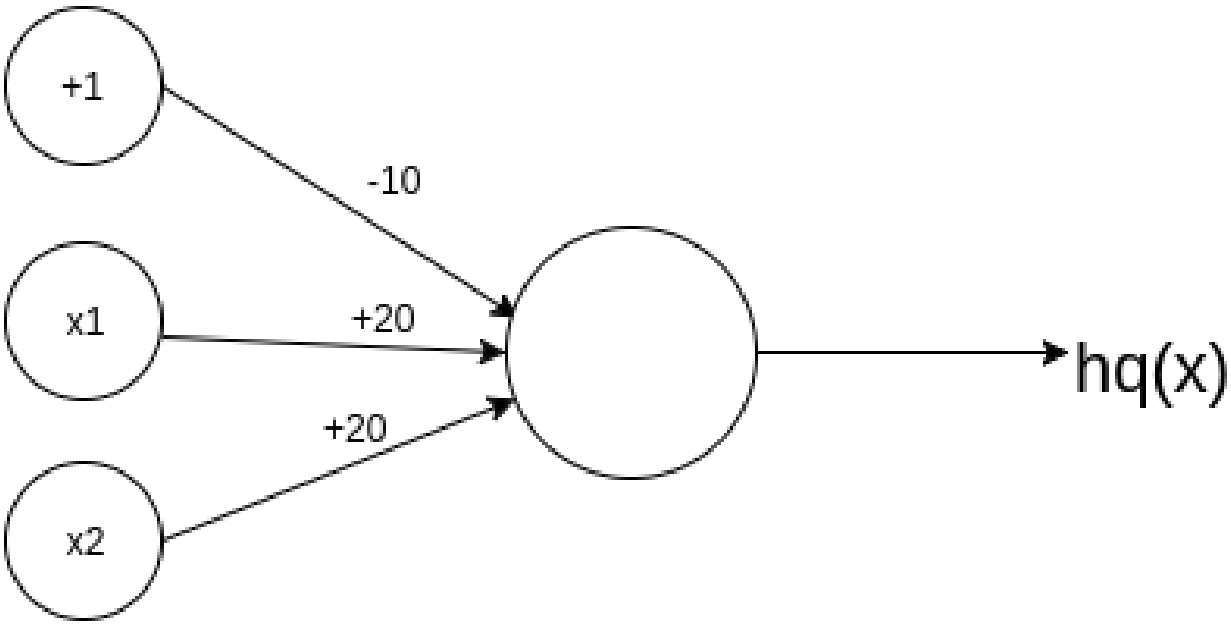
These three simple functions (logical AND, logical OR, and double negation AND function) may be now used as the activation functions in our three-layer neural network to compute another nonlinear function defined in the beginning: x_1 XNOR x_2. To do this, we need to put these three simple functions together into a single network.



Logical AND



Logical (NOT x_1) AND (NOT x_2)



Logical OR

This network uses three logical functions calculated above as the activation functions.

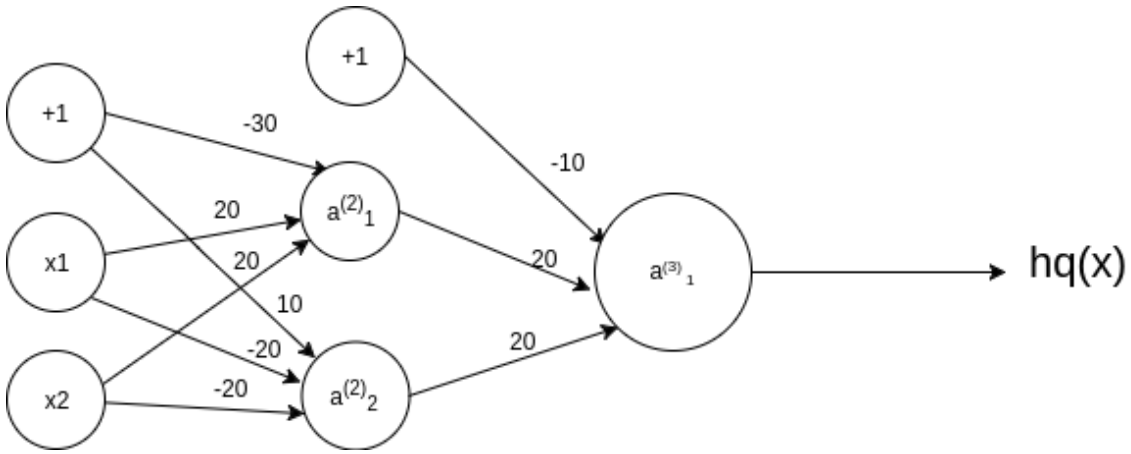


Image #10 A Neural Network to Compute XNOR Function

As you see, the first layer of this network consists of two inputs (x_1 and x_2) plus a bias unit +1. The first unit of the hidden layer is a Logical AND activation function that takes weights specified above (-30, 20, 20). The second unit $a^{(2)}_2$ is represented by the (NOT x_1) AND (NOT x_2) function that takes parameters 10, -20, -20. Doing our usual calculations, we get the values 0,0,0,1 for $a^{(2)}_1$ and the values 1,0,0,0 for the second unit in the hidden layer.

Now, the final step is using the second set of parameters from the logical OR function that sits in the output layer. What we do here, is simply take the values produced by the two units in the hidden layer (logical AND and (NOT x_1) AND (NOT x_2)) and apply them to the OR function with its parameters. The results of this computation make up our hypothesis function (1,0,0,1), which is our desired XNOR function.

x_1	x_2	$a^{(2)}_1$	$a^{(2)}_2$	$h_q(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

That’s it! Hopefully, as this example illustrates, neural networks are powerful in computing complex nonlinear hypotheses by using a cascade of functions. In fact, neural networks can use output values of a certain function as the inputs of other functions. Leveraging this functionality, complex multi-layered networks that can extract complex features and patterns from images, videos, and other data can be designed.

Conclusion

Artificial Neural Networks (ANNs) are the main drivers of the contemporary AI revolution. Inspired by the biological structure of human brain, ANNs are powerful in modeling functions and hypotheses which would be hard to derive intuitively or logically. Instead of inventing your own function with high-order polynomials, which may lead to overfitting, one can design an efficient ANN architecture that can automatically fit complex nonlinear hypotheses to data. This advantage of the ANNs has been leveraged in the algorithmic feature extraction in computer vision and image recognition. For example, instead of manually specifying a finite list of image features to choose from, we can design a Convolutional Neural Network (CNN) that uses the same principle as the animal's visual cortex to extract features. As a human eye, layers of the CNN respond to stimuli only in a restricted region of the visual field. This allows the network to recognize low-level features such as points, edges, or corners and gradually merge them into high-level geometric figures and objects. This example illustrates how good ANNs are in the automatic derivation of hypotheses and models from complex data that includes numerous associations and relationships.

About the Data Science Foundation

The Data Science Foundation is a professional body representing the interests of the Data Science Industry. Its membership consists of suppliers who offer a range of big data analytical and technical services and companies and individuals with an interest in the commercial advantages that can be gained from big data. The organisation aims to raise the profile of this developing industry, to educate people about the benefits of knowledge based decision making and to encourage firms to start using big data techniques.

Contact Data Science Foundation

Email:admin@datascience.foundation
Telephone: 0161 926 3641
Atlantic Business Centre
Atlantic Street
Altrincham
WA14 5NQ
web: www.datascience.foundation