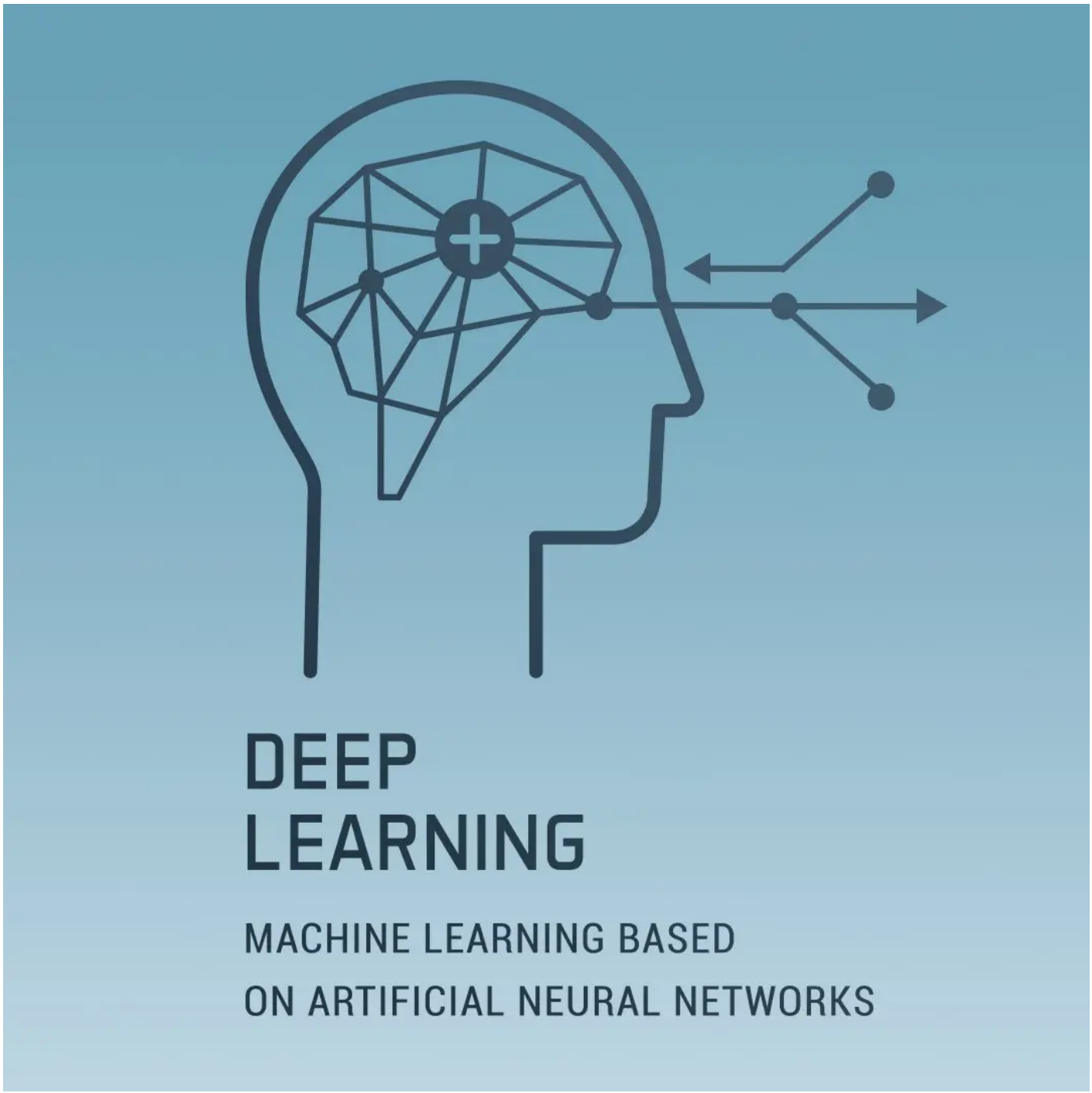
What is deep learning: case study in EDICOM (Part I)

In this article we ask ourselves if it is possible to program an algorithm that automatically qualifies the complexity of a new task. We will review the basics of artificial intelligence, machine learning, and deep learning.

March 11, 2021



Artículo elaborado por:

Jose Blas Vilata

Director Técnico y socio fundador de EDICOM

Introduction

At EDICOM (https://careers.edicomgroup.com/), each client request generates a technical task in our management system where different data are gathered, such as the description of the work to be carried out, the requesting customer, traceability dates, the number of working days to be billed to the customer for such work, etc. Then, an expert manager manually rates the complexity of this task in an order from one to five. On the basis of this qualification, a technical project manager with the appropriate skills will be assigned for its execution. This qualification will also serve to better adjust the capacity of technicians based on the number of projects they manage according to their complexity.

Case study of a business problem solved by deep learning!

In this article, we ask ourselves whether it is possible to program an algorithm that automatically rates the complexity of a new task, thus simulating what is currently done by a human expert, based on the knowledge of all the technical task data currently stored in our management systems for decades and duly rated by a human.

Although it would be possible to approach the issue from a traditional algorithmic point of view, we see quite clearly that the definition of this problem fits well within artificial intelligence algorithms, precisely in the machine learning branch and, specifically, in its most modern version: deep learning.

In this first article, we will review the basic concepts of artificial intelligence, machine learning and deep learning. In the following article, we will focus on the implementation of a deep learning system to solve this problem.

Artificial intelligence, machine learning and deep learning

In recent years, the interest and application of artificial intelligence (AI) has undergone strong growth, becoming a discipline applied in practically all fields of academic and industrial research.

This is exactly why it has grown as a field!

This is largely due to the fact that in the 1980s, a branch of AI emerged, **automatic learning or machine learning** (ML), which applies mathematical algorithms that enable machines to learn. Machine learning is bound to play a key role in the future of civilization.

Machine learning algorithms learn from the data entered and then use this knowledge to draw conclusions from new data, either discrete data by classifying into categories or predictions of continuous numerical results.

Now in the 21st century, in 2011, a branch of machine learning designated **deep learning (DL)** emerged. Deep learning is distinguished from traditional machine learning in that the latter works with **regression algorithms or with decision trees, while deep learning, in addition, uses artificial neural networks** that function much like the biological neural connections in our brains. In practical terms, for problems with many variables, it is much easier to program a deep learning system than it is to develop a regression formula for a traditional machine learning algorithm. In this sense, deep learning has made a decisive contribution to the rise of artificial intelligence.

Why Deep Learning is interesting!

Machine learning, in general, uses different types of algorithm to find patterns in data. These algorithms can be classified in three groups:

- **Supervised machine learning: The general purpose of these algorithms is to predict an outcome from the input we provide at any given time. To do so,** the program is "trained" with a predefined set of "example or training data" for which we know the outcome to be predicted, and uses these data to create structures (models) to represent them. Once the models have been generated, they can be used to predict results by taking as input new data not present in the training sample. It is important to bear in mind that machine learning systems "learn" patterns that are present in the data they are trained on, so they can only recognise scenarios similar to what they have seen before. By using systems trained on past data to predict future outcomes, it is assumed that future behaviour will be similar, which is not always the case.
- **Unsupervised machine learning:** The program receives a large amount of data and must find patterns and correlations in them. In other words, it helps us find relationships between the data that were not known *a priori*, i.e., this type of algorithms help us to know our data better. They address the following types of problems: familiarity, principal component analysis, clustering, prototyping, extraction and feature relationships.
- **Reinforced learning**: The aim is to build a model with an agent that improves its performance based on the reward obtained from the environment with each interaction that takes place. Reward is a measure of how successful an action has been in achieving a given objective. The agent uses this reward to adjust their future behaviour to obtain the maximum reward.

Connects to my knowledge of the different types of machine learning.

In this article, we will focus exclusively on supervised machine learning.

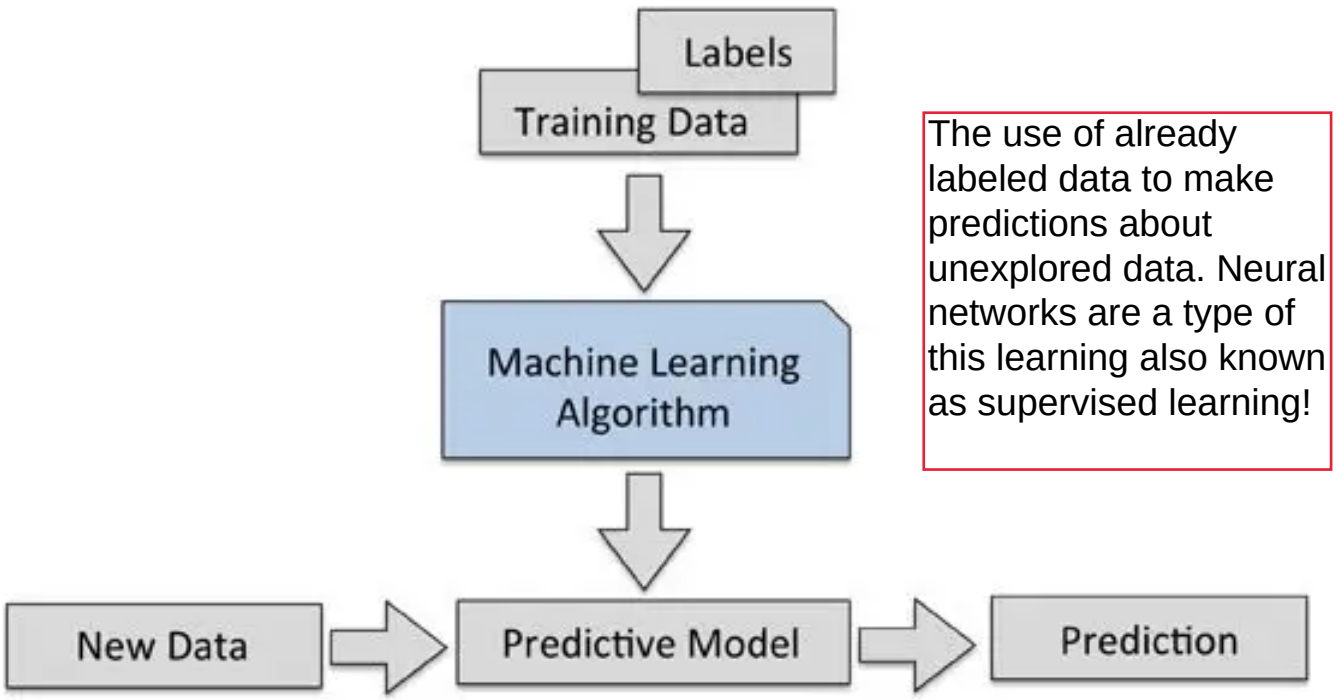
Supervised machine learning

The general procedure is as follows:

1. We have an algorithm that has a handful of labelled examples, say, 10 images of dogs with label 1 (“Dog”) and 10 images of other things with the label 0 (“Not dog”).
2. The algorithm “learns” to identify images of dogs and, when given a new image, it expects to produce the correct label (1 if it is an image of a dog and 0 otherwise).

This configuration can be applied to many real-life problems where the input data might be symptoms and their labels diseases; or where the data might be images of handwritten characters and their labels the actual characters they represent, etc.

In this article, we will try to solve the real EDICOM problem of automatically rating the complexity of a technical task in an order from one to five, based on the previous expert human decision stored in the company’s management system.



There are two main applications of supervised learning: classification and regression.

Classification is a subcategory of supervised learning in which the aim is to predict categorical classes (discrete, unordered values, group membership, etc.). The typical example is spam detection, which is a binary classification (an e-mail is either spam – value "1" – or it is not – value "0"-). There is also multi-class classification, such as handwritten character recognition (where the classes range from 0 to 9) or the real problem proposed by EDICOM that we are trying to solve in this article, as the output will be a discrete number from 1 to 5.

Regression is used in systems where the value to be predicted falls somewhere on a continuous spectrum. These systems help us with the questions "How much?" or "How many?" In this type of learning, we have a number of predictive (explicative) variables and a continuous response (result) variable, and we will try to find a relationship between these variables that will provide us with a continuous result.

Connects to my knowledge of Regression vs. Classification.

Deep learning– neural networks

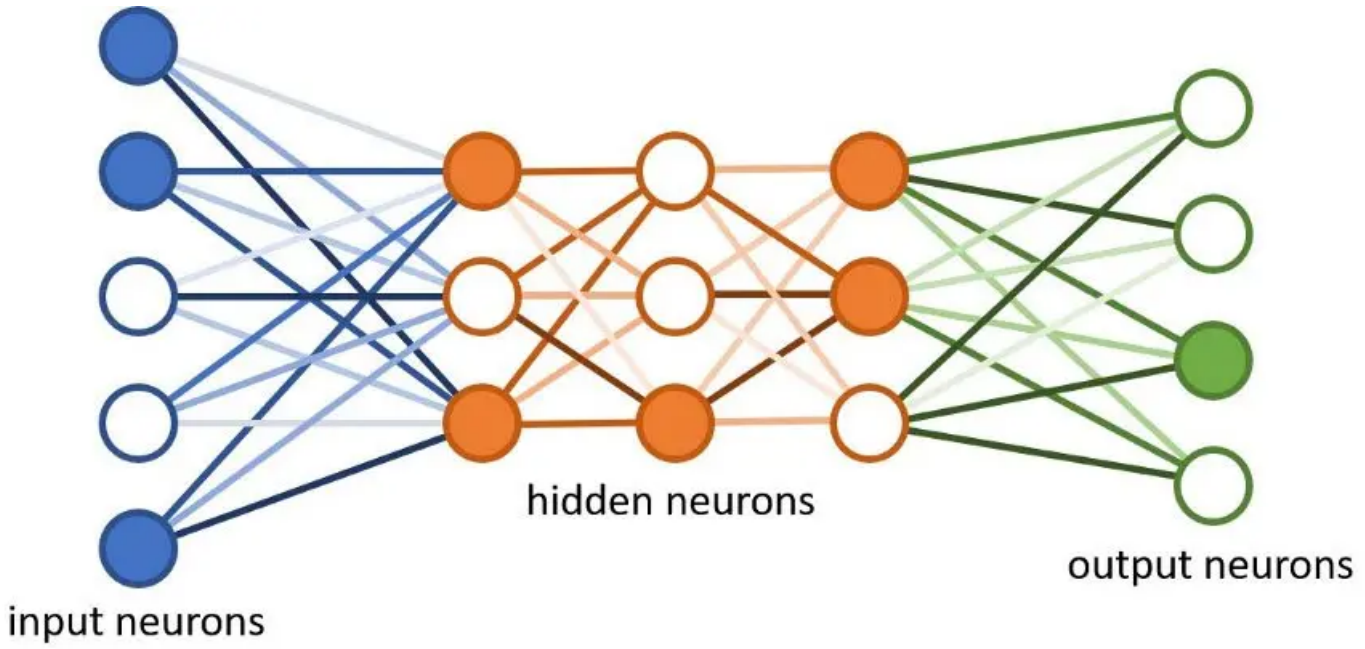
Deep learning is a subfield of machine learning that is characterized by a large number of calculations. This depth of computation, through artificial neural networks, is what has enabled deep learning models to unravel the kinds of complex, hierarchical patterns found in the most challenging real-world datasets.

Neural networks, thanks to their power and scalability, have become the defining model for deep learning. Neural networks are composed of neurons, where each neuron individually performs only one simple computation. The power of a neural network comes, instead, from the complexity of the connections that can be formed by these networks.

How neural networks actually work!

Neural networks are organised in layers composed of neurons or units. Each neuron in a layer is connected to neurons in the previous layer by weights and produces a result that is passed on to all

neurons in the next layer.



The first layer of the neural network takes raw data as input, in our case, data from a technical task, processes it, extracts information and transfers it to the next layer as output. This process is repeated in the following layers. Each layer processes the information provided by the previous layer, and so on until the data reaches the final layer, which is where the prediction is obtained. This movement of calculations through the network is designated *forward propagation*.

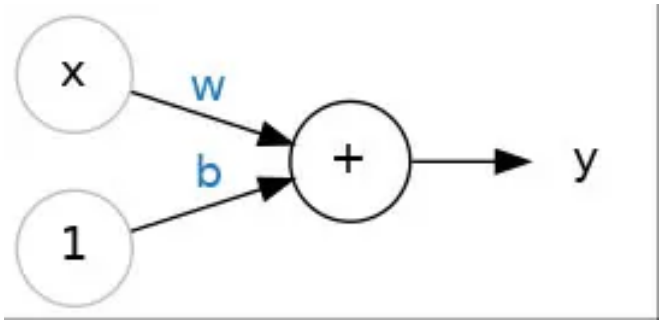
Forward propagation leads to calculations among each layer to create the outcome.

If we are in production mode, the prediction obtained will be the result to be returned; if on the other hand we are still in model training mode, this prediction is compared with the known result and thus identifies errors in the predictions, assigns them weights and biases, and pushes them back to previous layers to train or refine the model. This process is known as *backpropagation or backprop (BP)*. Together, forward and backward propagation allow the network to make predictions about the identity or class of the object while learning from inconsistencies in results. The result is a system that learns as it operates and becomes more efficient and accurate over time as it processes large amounts of data.

Back propagation leads to the minimization of loss and inconsistency.

Linear unit (neuron)

The fundamental component of a neural network is the individual neuron. As a diagram, a neuron (or *unit*) with an input looks like this:



The neuron would be the circle with the "+". The input is *x,* which would correspond to an attribute of our real data. Its connection with the neuron has a weight that is *w*. Whenever a value flows through a connection, it multiplies the value *x* by the weight *w* of the connection. For input *x*, what reaches the neuron is *w * x*. A neural network "learns" by modifying its weights based on iterations. Random weights are assigned in the first iteration.

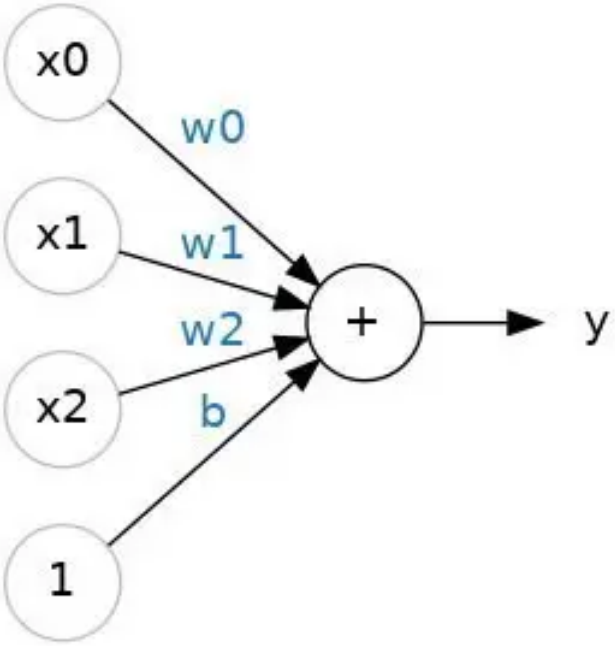
So bias basically represents how much should the input affect the total output!

The *b* is a special type of weighting that we call *bias*. The bias has no input data associated with it; instead, we put a 1 in the diagram so that the value reaching the neuron is simply *b* (as *1 * b = b*). Biasing allows the neuron to modify the output independently of its inputs.

The *y* is the value ultimately produced by the neuron. To obtain the output, the neuron adds up all the values it receives through its connections. The activation or output formula of this neuron is *y = w * x + b*.

Linear regression formula for a single neuron!

If we have several inputs (attributes) x_0, x_1, x_2 for a neuron:



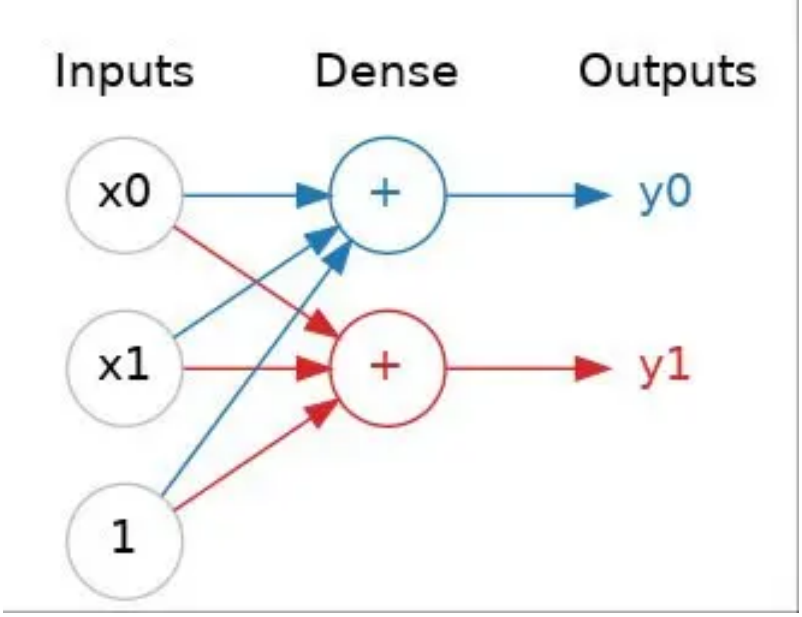
The formula for this neuron would be $y = w_0x_0 + w_1x_1 + w_2x_2 + b$. Graphically, a linear unit with one input will be represented as a line ($y= wx +b$); with two inputs, it will fit in a plane, and a unit with more inputs will fit in a hyperplane.

A linear unit calculates a linear function. We will now look at how to combine and modify these individual units to model more complex relationships: **deep neural networks**.

This was just simple linear regression which will be taken to the next level which has layers and actually learns complex non-linear data.

Layers

Neural networks tend to organize their neurons (*units*) in layers. When we gather linear units that have a common set of inputs, we obtain a *dense layer*.



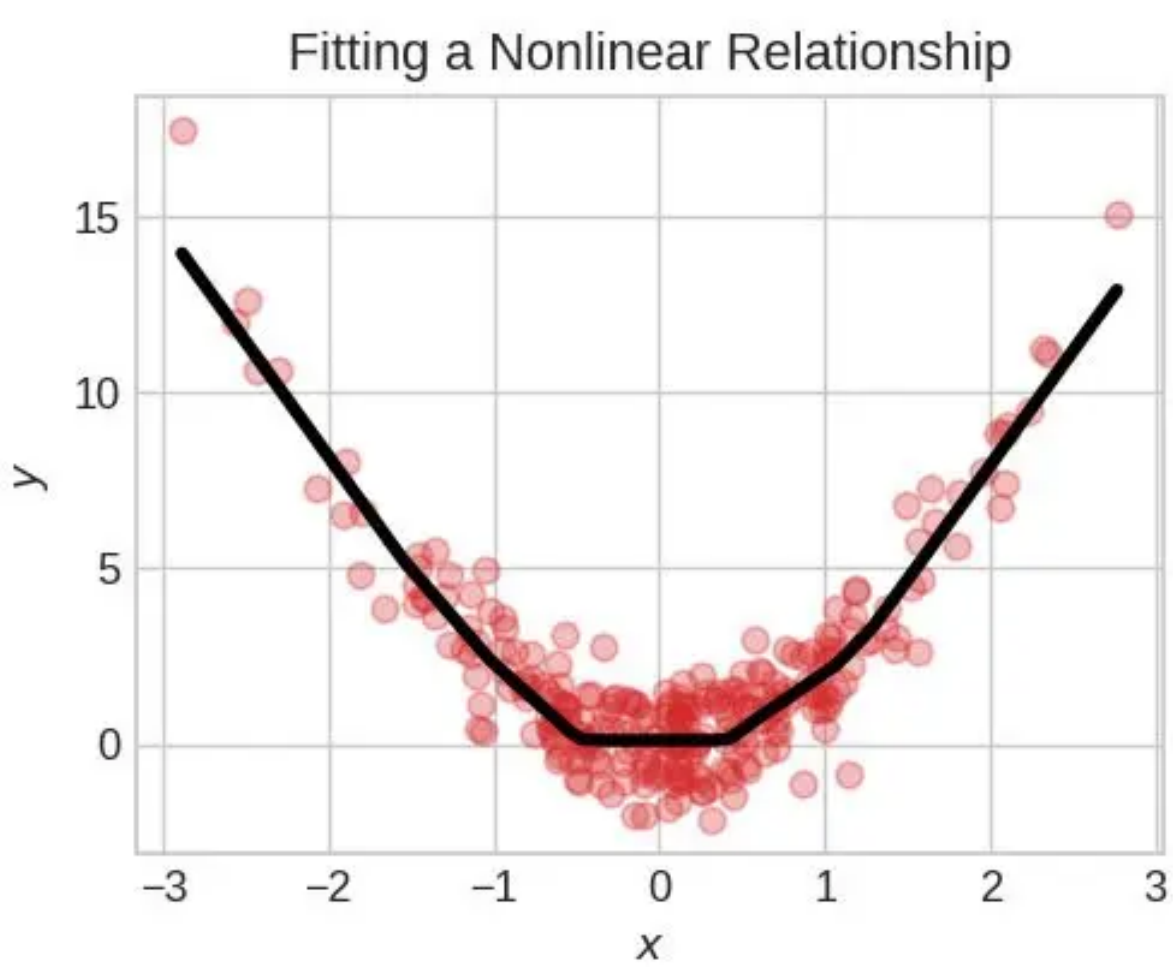
A dense layer of two linear units that receive two inputs and one bias.

Each layer of a neural network performs some relatively simple transformation. Through a stack of layers, a neural network can transform its inputs into increasingly complex shapes. In a well-trained neural network, each layer is a transformation that brings us one step closer to a solution.

The activation function

However, it turns out that two dense layers by themselves are no better than a single dense layer. Dense layers by themselves can never take us out of the world of lines and planes. However, not all problems can be solved with linear approximations. What we need is something non-linear. What we need are "activation functions".

Activation functions that contain non-linear algorithms are really the ones that help create an actual neural network that can process complex data!



Without activation functions, neural networks can only learn linear relationships. To adjust the ratios in the form of curves, we will need to use activation functions.

Transforms the linear data model into a non-linear one with the use of curves

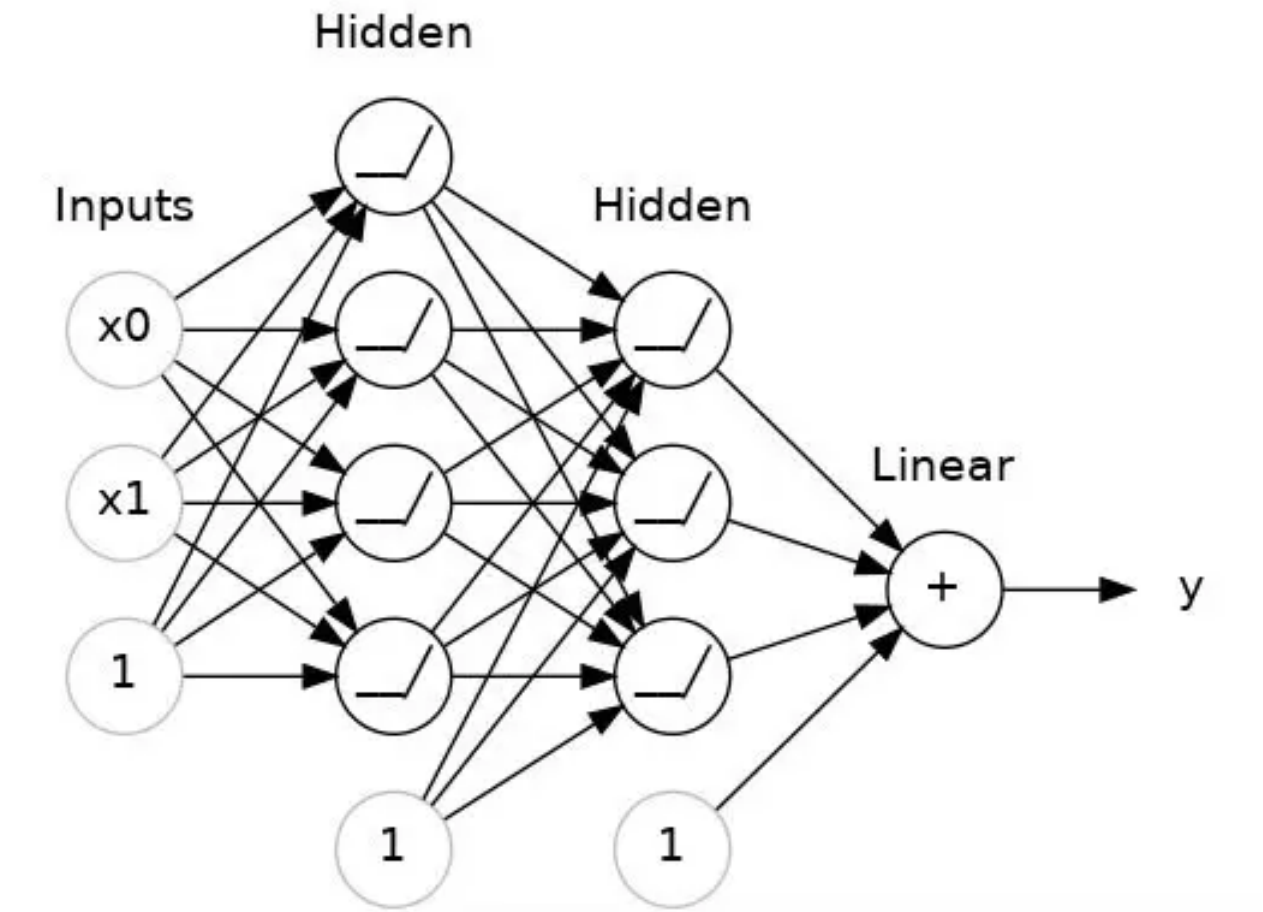
An activation function is simply the one that we apply to each of the outputs of a layer. The most common is the *rectifying function*: *max(0,x)*. The rectifier function has a graph that is a line with the negative part "rectified" to zero. Applying this function to the outputs of a neuron will add curves to the data, moving away from simple lines.

ReLU is the most common activation function where the negative data becomes zero.

When we connect the rectifier to a linear unit, we obtain a *rectified linear unit or ReLU*. (For this reason, the rectifier function is commonly called the "ReLU function"). Applying a ReLU activation to a linear unit means that the output becomes *max (0, w * x + b)*.

Dense layer stacking – hidden layers

Now that we have some nonlinearity, let's see how we can stack layers to achieve complex data transformations:



A stack of dense layers creates a "fully connected" network.

The use of layers and activation function really depends on the problem and the way you are trying to solve the problem. Interesting!

The layers before the output layer are sometimes called hidden layers, as we never see their outputs directly.

Now, in the diagram we can see that the final layer (output) is a linear unit (i.e. with no activation function). This means that this network is appropriate for a regression task, where we are trying to predict some arbitrary numerical value. Other tasks (such as sorting) may require an activation function on the output.

How a neural network learns

When first created, all network weights are set randomly; the network does not "know" anything yet.

As with all machine learning tasks, we start with a training data set. Each example in the training data consists of some features or attributes (the inputs) together with an expected target (the output),

which is another attribute. Training the network means adjusting its weights in such a way that it can transform the features into the target.

In addition to the training data, we need another two things:

- A "loss function" that measures how good the network's predictions are.
- An "optimizer" that can tell the network how to change its weights.

Loss function

The loss function measures the disparity between the actual value of the target and the value predicted by the model.

A common loss function for regression problems, where the task is to predict some numerical value, is the mean absolute error or MAE. For each prediction *y_pred*, MAE gauges the disparity of the true target *y_true* by an absolute difference *abs (y_true – y_pred)*.

The total MAE loss in a data set is the average of all these absolute differences.

I have learned about these algorithms in my crash course.

In addition to MAE, other widely used loss functions for regression problems include ***root mean square error*** (MSE) or Huber's loss.

During training, the model will use the loss function as a guide to find the correct values of its weights (a lower loss is better for the model than a higher loss). In other words, the loss function tells the network its objective.

The optimizer – Stochastic gradient descent (SGD)

The optimizer is an algorithm which adjusts the weights in *backpropagation* to minimize loss.

The actual mathematical algorithm that minimizes loss!

Virtually all optimization algorithms used in deep learning belong to a family of algorithms called ***stochastic gradient descent (SGD)***. They are iterative algorithms that train a network in steps.

A training or iteration step would be like this:

1. A subset of training data (*minibatch* or *batch*) is extracted and passed on through the network to obtain a result (make predictions).
2. The loss between predictions and actual values is measured.
3. Finally, the weights are adjusted in a direction that reduces loss, *backpropagation.*
4. This process is then repeated until the loss is as small as desired or until it no longer decreases.

Back propagation is all about reducing loss!

As passing the whole training set to one iteration may be too heavy, it is usually divided into several batches called ***minibatch*** or ***batch***. Each and every one of the *batches* obtained from the division of the training set is subjected to an iteration. A complete round of iterations over all the batches in the training set is called an **epoch**. The number of epochs for which you train is the number of times the network will see each training example.

Each time SGD sees a new minibatch, it will change weights (*w* and *b*) to their correct values in that batch.

Connects to the information I learned from my courses about supervised learning!

Learning rate and batch size

A lower learning rate means that the network needs to see more minibatches before the weights converge to their best values.

Learning rate and minibatch size are the two parameters that have the greatest effect on how SGD training performs. Their interaction is often subtle and the right choice for these parameters is not always obvious.

Fortunately, for most jobs, it will not be necessary to perform an extensive hyperparameter search to achieve satisfactory results. *Adam* is an SGD algorithm that has an adaptive learning rate that makes it suitable for most problems without any parameter tuning (it is “self-tuning”, in a sense). Adam is a great general purpose optimizer.

Bibliography

- Machine learning with Python and Scikit-learn by Joaquín Amat Rodrigo, available under an Attribution 4.0 International (CC BY 4.0) at https://www.cienciadedatos.net/documentos/py06_machine_learning_python_scikitlearn.html (https://www.cienciadedatos.net/documentos/py06_machine_learning_python_scikitlearn.html)
- A Deep Learning Tutorial: From Perceptrons to Deep Networks by Ivan Vasilev, available at https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks (https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks)
- Intro to Deep Learning by Ryan Holbrook of Kaggle, available at https://www.kaggle.com/learn/intro-to-deep-learning (https://www.kaggle.com/learn/intro-to-deep-learning)

News categories