Deep Learning

by DeepLearning.AI

Lucas Paiolla Forastiere

August 5, 2021

Contents

Chapter 1

Introduction

The term deep learning refers to training *neural networks*, sometimes very big neural networks. But what are neural networks?

So let's suppose we want to predict housing prices based on the size of the house. And let's say we'll use Logistic Regression to do that. But as we know, house prices can't be negative, so we simply say the value of the house is 0 if the Logistic Regression would predict something negative.

That's indeed the simplest neural network we can have, we have a single input `size` and a single output `price` and in the middle we have a single neuron: the logistic regression.

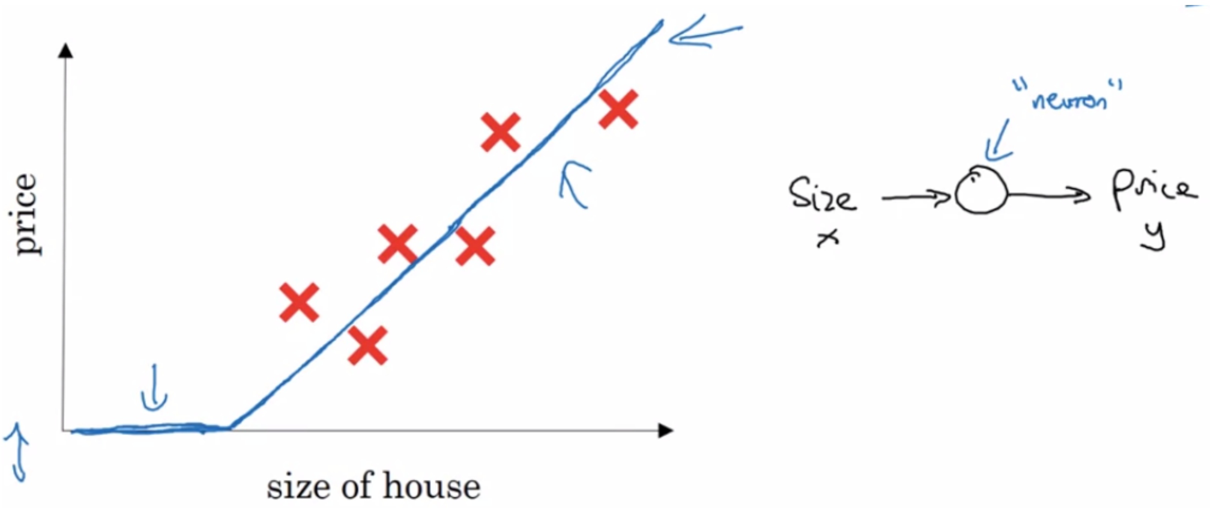

Figure 1.1: Here we see the graph of the problem we described.

That function which is zero and than linear is called *ReLU* and it's used a lot in neural networks. It stands for *Rectified Linear Unit.*

So to get a bigger neural network, we stack these neurons. Instead of predicting using only the size of house, we could use the number of bedrooms, zip code and wealth. We could use the size and number of bedrooms to predict the family size; use the zip code to predict the walkability; and use the zip code and wealth to predict the school quality. And then, we could use the family size, walkability and school quality to predict the price. See in the picture:

However, in general what we have is something a little more complex than that. We would have something like figure 1.3. Here we see that the internal nodes (which are called **hidden nodes** or **hidden neurons** or **hidden units**) receive the output of all the

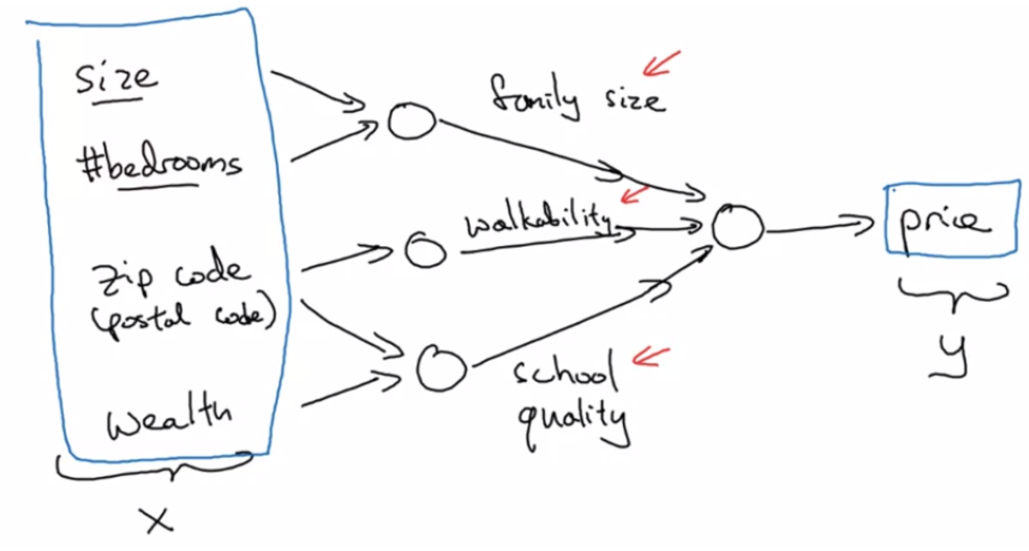

Figure 1.2: Now we have a more complex neural network, which is the stack of many ReLUs.

previous nodes to make it predictions. These hidden nodes don't really have a meaning like the example we gave. We don't try to predict family size or walkability or whatever, we simply let the neural network decide what that neuron will output in order to predict the final output `price` in the better way it can.

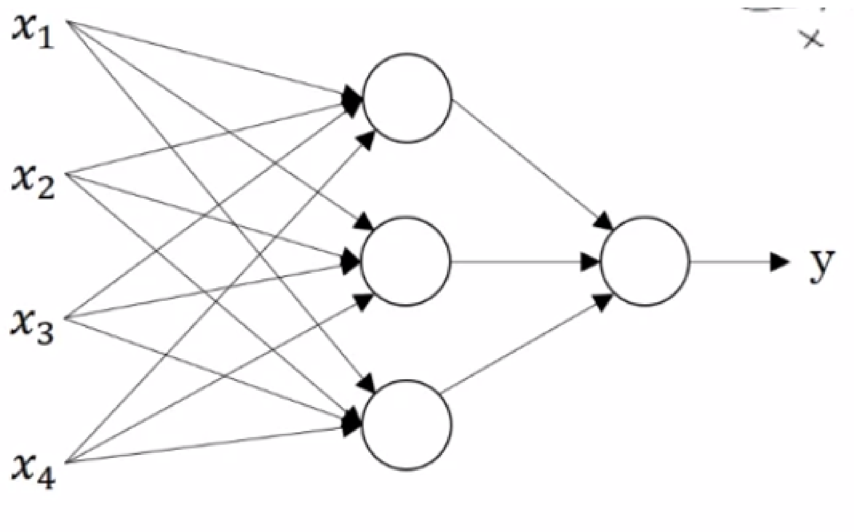

Figure 1.3: The generic form of a neural network.

We can use neural networks in many applications, here we're going to foucos in **supervised learning**, which are problems that you have a set of variables called input (represented by x) and an output (y) related to that input. In order to solve these kind of problems, there are many kinds of neural networks. The one we saw is the most common one, but there are others, like convolutional nn or recurrent nn.

Another thing that's important to decide what kind of nn we'll use is knowing if the data we're leading with is *structured* or *unstructured*.

Structured data is data in the form of a table. We have a very clear set of input variable X and a set of output variables y. Each line of our table represents one instance

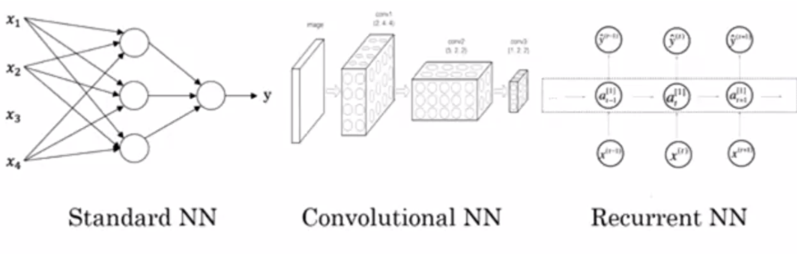

Figure 1.4: Examples of neural networks.

of data with many inputs and one or more outputs related to those inputs.

Unstructured data is all the other kinds of data: audio, video, texts, images, etc.

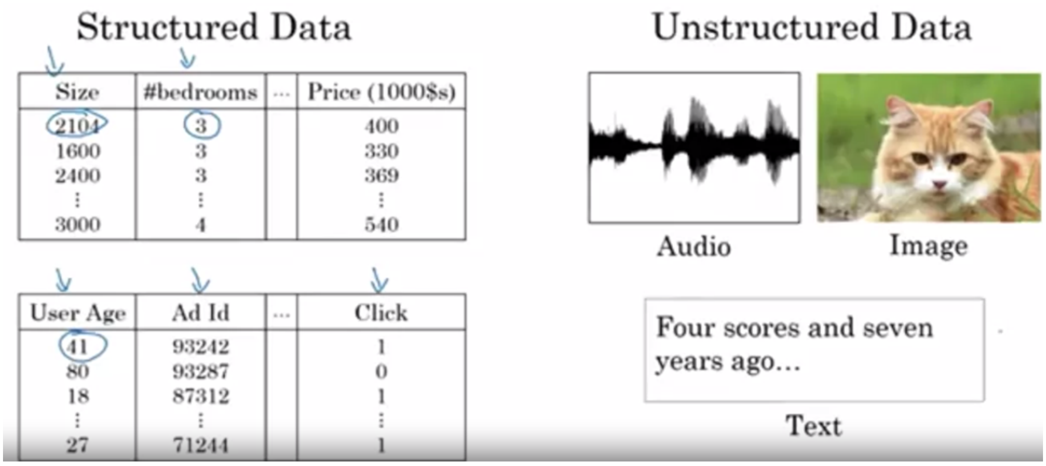

Figure 1.5: The two kinds of data.

It turns out that machine learning algorithms performed better on structured data over the years and more recently neural networks are performing better also on unstructured data.

Why is Deep Learning taking off? This is one of the questions we must ask ourselves when begining to learn deep learning. Let's see the graph of the performance of the machine learning algorithms versus the amount of data that we provide to then. We see that traditional learning algorithms have a plato where they can't improve anymore, which neural networks can lead with that data as we make than bigger and bigger.

We also see in the graph that when we don't have a large amount of data, NNs all algorithms perform pretty much the same.

So in order to answer our question, we have to understand the evolution of three things: *data*, *computation* and *algorithms*.

Through the years, the amount of data available was inscreased a lot, so NNs can take advantage from that. Also the computation power was inscreased with the use of GPUs to make a large amount of computations. And finally new algorithms have been developed to make NNs faster. That's the main reason why deep learning is taking off.

1.1 Notation

Before continuing, we need to define the notation we're going to use.

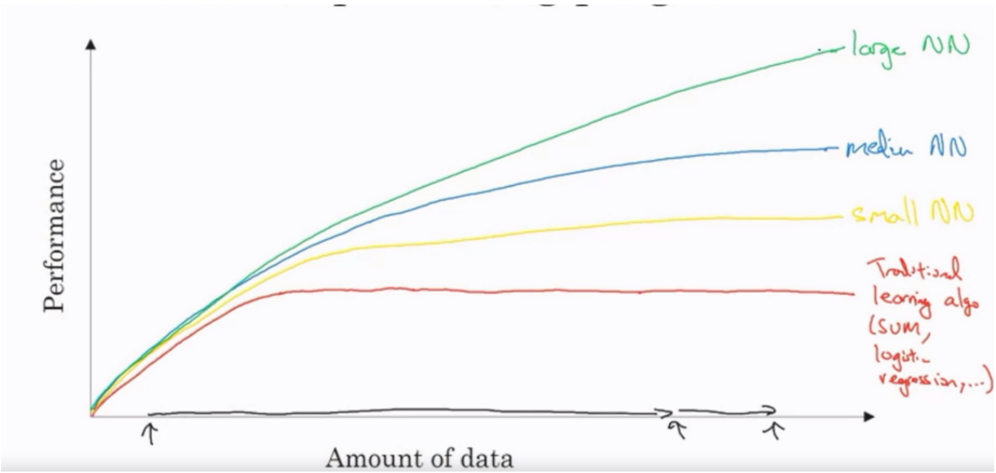

Figure 1.6: The performance of machine learning algorithms in respect to the data we provide to them.

- (x, y) will denote a single training input;
- m or m_{train} denotes the number of training examples;
- $x^{(i)}$ denotes the i-th training input;
- $y^{(i)}$ denotes the i-th traning output;
- n_x or n denotes the number of dimensions x has (or the number of features);
- m_{test} denotes the number of testing examples;
- X is the matrix of all traning examples. It's defined as:

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

  X is an $m \times n$ matrix;

- Y is the matrix of all outputs. It's defined as:

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

  Y is a $1 \times m$ matrix.

Observation. In other courses we might see X defined as the transpouse of the matrix we've just defined. But it turns out that when using this definiting, it's much easier to implement algorithms, so remember te definition we're going to use through out the course.

The same thing for Y. We see that here Y is the transpouse of that it's tends to be in other courses.

1.2 Logistic Regression as a Neural Network

To end this introduction, we'll see the basics of neural network programming using the simplest NN we can: a logistic regression.

So let's recall what's logistic regression and why it's useful. Logistic Regression is used in binary classification, the kind of problem where we have an input and want to predict between 0 or 1. An example could be an image and we want to say it what's a cat (1) or not (0).

Basicly we want an algorhthm to estimate the probability of $y = 1$ given x. In math we write:

$$\hat{y} = P(y = 1 \,|\, x), \qquad x \in \mathbb{R}^n$$

Logistic Regression estimates this quantity using the formula:

$$\hat{y} = \sigma(w^T x + b),$$

where w and b are parameters to be discovered and σ is the **sigmoid function**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

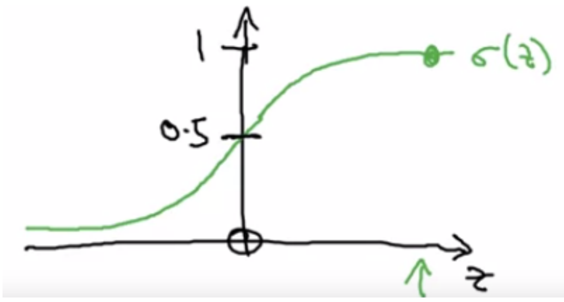

Figure 1.7: A sigmoid graph.

It's also common to create a new input $x_0 = 1$ and use the x vector as $x \in \mathbb{R}^{n+1}$ and use the formula $\hat{y} = \sigma(\theta^T x)$, where

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \qquad \theta_0 = b \qquad w = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

To find the parameters b and w, we need to define a **cost function**, which is a function that says how badly our algorithm is performing. This is a function that we want to minimize and when we minimize, we find the best values of b and w.

The **cost** function is a function of all training examples, while a **loss function** or **error function** is a function of a single traning example that measures how well our algorithm is performing.

For logistic regression, we use the loss function:

$$\mathcal{L}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Notice that this is the same as:

$$\mathcal{L}(\hat{y}, y) = \begin{cases} -\log(\hat{y} - 1), & \text{if } y = 0 \\ -\log \hat{y}, & \text{if } y = 1 \end{cases}$$

That give us the cost function:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

1.2.1 Gradient Descent

We know have:

- A way of predicting the classes 0 or 1 using the sigmoid function;
- A way of measuring the error of our predictions.

What we need now is a way of chaning our parameters b and w in order to minimize the error. That's what the **gradient descent** algorithm does.

Let's first see a general graph of the cost function. In general, it looks like figure 1.8

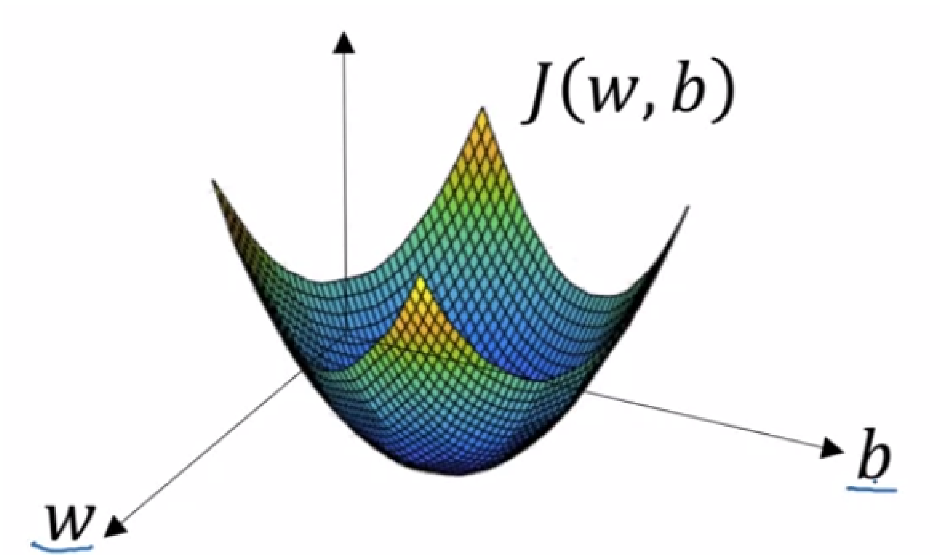

Figure 1.8: A generic graph of the cost function.

We see that J is what we call a **convex function**, which means that it is a function that was only one **local minimum** (or local maxima). This property is very important if we want to apply the gradient descent algorithm.

In Gradient Descent, we initialize b and w randomly and take steps into the direction that leads us to the lowest possible value of J. In order to do that, we calculate the **gradient** (the derivatives in each direction) of the function J and take a step in the opposite direction of the gradient.

Proposition 1.1

The gradient gives us the direction of the maximum increase of the a function.

Algorithm 1.1: Gradient Descent

Repeat {

$$w := w - \alpha \frac{\partial w}{\partial J(w,b)}$$

$$b := b - \alpha \frac{\partial b}{\partial J(w,b)}$$

}

In the algorithm, α is what we call the **learning rate**. It's how large we should step in the direction of the maximum decrease. If we take big steps, we can go much faster to the global minumum, but we might not be so accurated. On the other hand, we be take small steps, we can find the global minum accuratedly, but achieve it much slower.

Gradient Descent is a general optimization algorithm and can be applied to any convex function. So now we need to understand how to use it with logistic regression.

After calculating the derivatives, we'll have:

Algorithm 1.2: Gradient Descent for Logistic Regression

Repeat {

$J = 0;\ dw = 0;\ db = 0$

For $i = 1 \cdots m$:

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J += -\left[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})\right]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw += x^{(i)} dz^{(i)}$

$db += dz^{(i)}$

$J /= m$

$dw /= m;\ db /= m$

$w := w - \alpha dw$

$b := b - \alpha db$

}

This version of the algorithm uses a for loop to compute J, dw and db. But when implementing the code into Python or other language, we always try to **vectorize** the code to make it faster.

Algorithm 1.3: Gradient Descent for Logistic Regression Vectorized

Repeat {

$Z = w^T X + b$

$A = \sigma(Z)$

$dZ = A - Y$

$$dw = \frac{1}{m} X dZ^T$$

$$db = \frac{1}{m} \sum dZ$$

$w := w - \alpha dw$

$b := b - \alpha db$

```
}
```