grégory châtel

What do deep neural networks understand of fractals ?

A few months ago, I started the fast.ai deeplearning MOOC (which is awesome by the way). During this course, I learned how to use, tune and create neural networks but I still lack *intuition* on network architectures. I decided to run a little experiment to clarify my ideas and I think it could be interesting to some people.

All the code used to produce this post is available on Github.

A neural network is often seen as a black box that approximates a function based on a few example computations. The more complex the neural network, the more complicated the functions it can approximate. That seems easy enough to grasp but what troubled me is the lack of quantifier: how complex does a neural network need to be to approximate a given function ?

Intuitively, a neural network simply computes a big function that contains linear combinations of variables interwoven with functions that add non-linearities. The internal parameters of this network are then tuned to get a network that fit our use case. The three main parameters for a simple neural network are:

> the number of layers which corresponds to the number of linear combinations that will be computed,

the number of nodes in each of the layer which corresponds to the number of inputs of each of these linear combinations

the activation function used to add non-linearity into the network.

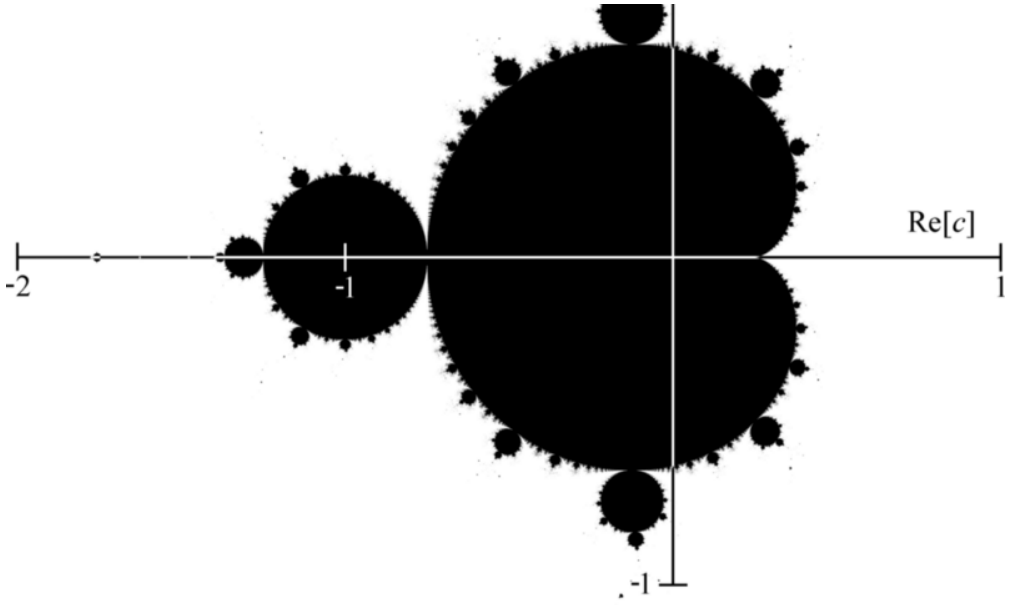
Sure, the universal approximation theorem essentially tells us that this kind of neural network can approximate basically anything with a single layer between the inputs and the outputs given enough nodes. That's not really useful in practice as this does not give us any idea on the number of nodes necessary (among other reasons).

To see a bit clearer in this world, I chose to take a very complicated function and ask various neural networks what they understand of it. The function that I chose to approximate comes from the world of fractal geometry and I will now give an outline of its computation.

What interest us is whether this sequence diverges (the norm of each term goes to infinity) or not for a complex number c when n goes to infinity:

$$z_n = \begin{cases} c & \text{if n} = 0 \\ z_{n-1}^2 + c & \text{otherwise} \end{cases}$$

This function might seem strange but it is in fact well known, not as itself but as what it allows to draw: the Mandelbrot set.



The mandelbrot set (image from wikipedia.org)

In this image, each pixel corresponds to a complex number: the *x* and *y* coordinates gives us respectively the real and imaginary part of the complex number. To get the color of the pixel *(x, y)*, we first find the associated complex number *c* and use it to compute the *z* sequence presented above. If this sequence diverges, the pixel is white and if not, the pixel is black.

I will not get into the details on how to test if a sequence diverges as there is a ton of excellent tutorial on this on the Internet. Here, we are just going to assume that we know how to do it.

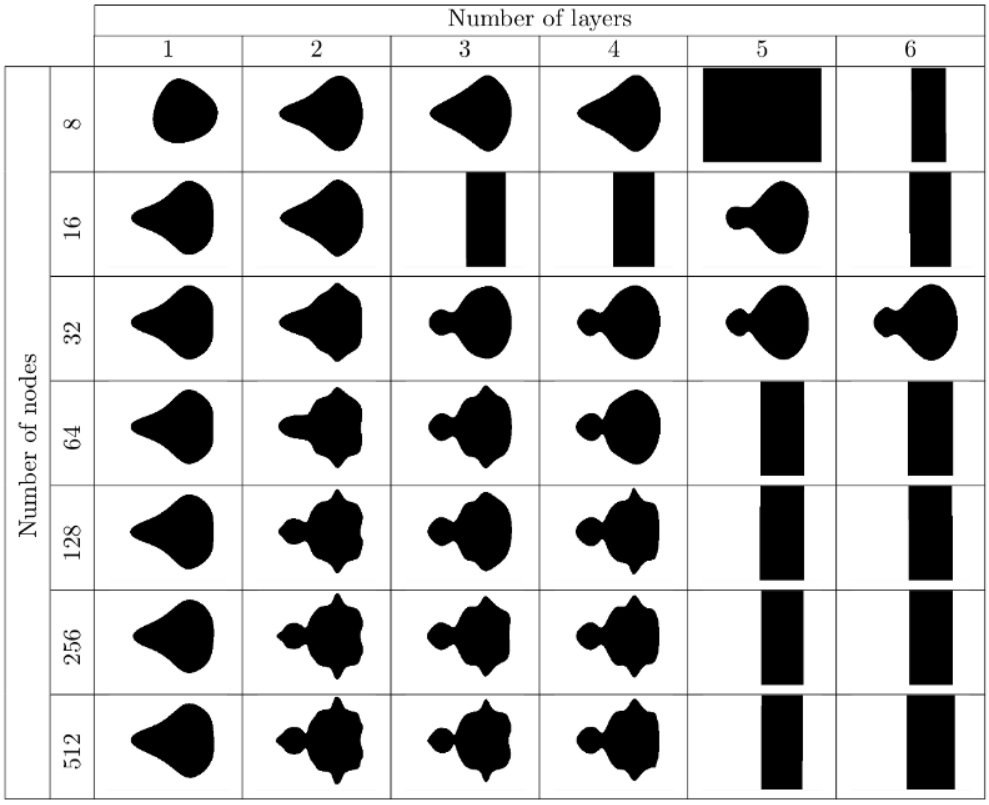
The function that we are trying to approximate is the following one:

$$f(c) = \begin{cases} 1 & \text{if the sequence } z_n \text{ diverges} \\ 0 & \text{otherwise} \end{cases}$$

To build the data set we will use to train the network we take a lot of complex numbers and compute their images by the function *f*. The goal of the network will be to approximate *f* as well as possible (*i.e.* the color of the pixel in the image).

I have created three grids, one for each the following activation functions (the non-linear bits of the network): sigmoid, hyperbolic tangent and rectified linear unit.

First, the results for the networks with the sigmoid activativations:

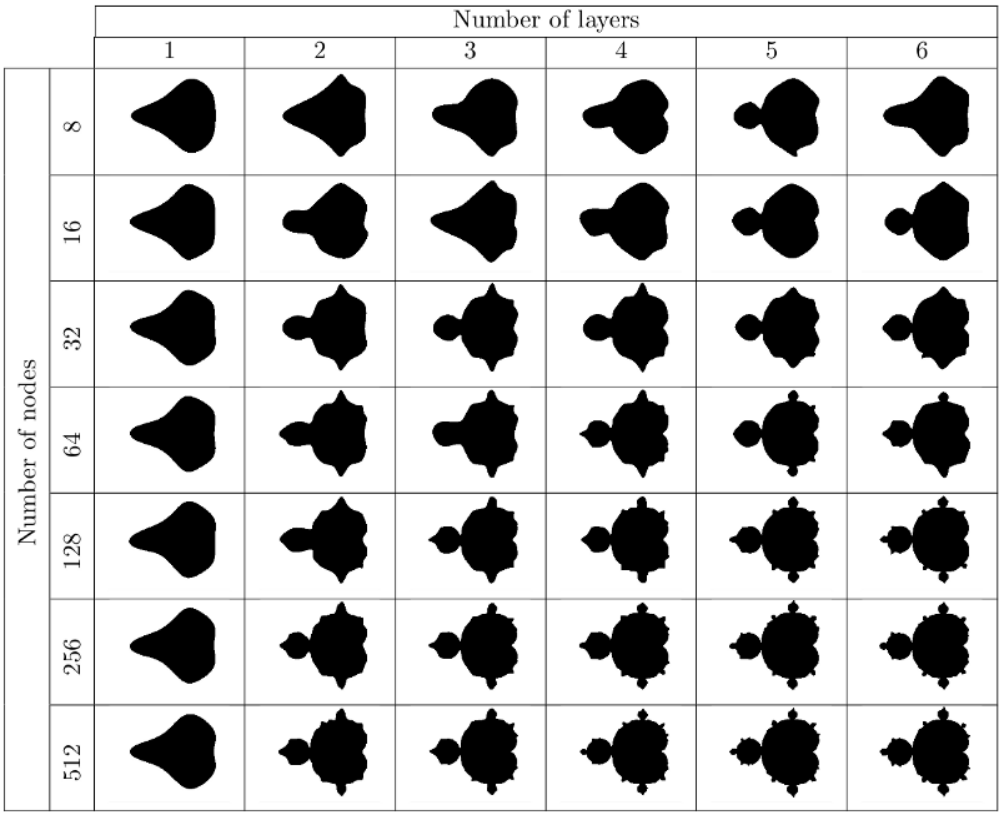


Mandelbrot set approximation using various neural networks with sigmoid activations

These result are pretty bad, to say the least. We can see that as the number of layers and number of nodes in each layer

increases, the network gets confused and do not manage to leave the local optimum that it found. This could be due to the fact that the sigmoid activations tend to produce unstable gradients.

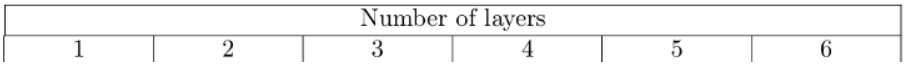
Now the results for the hyperbolic tangent:

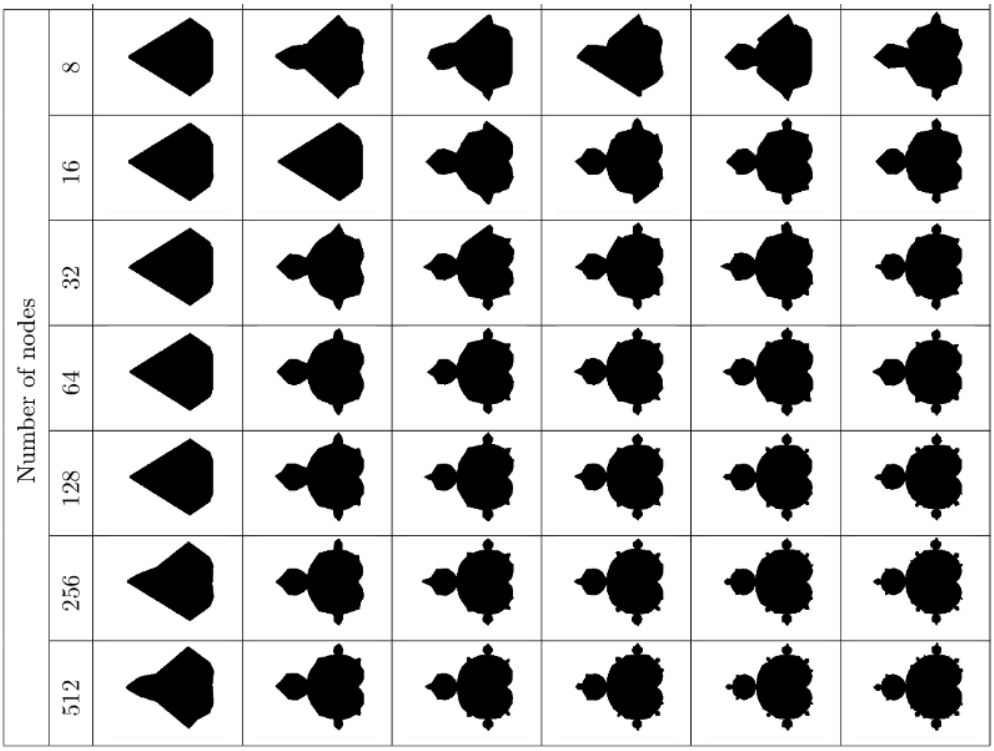


Mandelbrot set approximation using various neural networks with hyperbolic tangent activations

Already, we can see that the images produced are much better. In particular, we can also note that the depth (the number of layer) of the network seems to have a much larger impact than its width (the number of node in each layer).

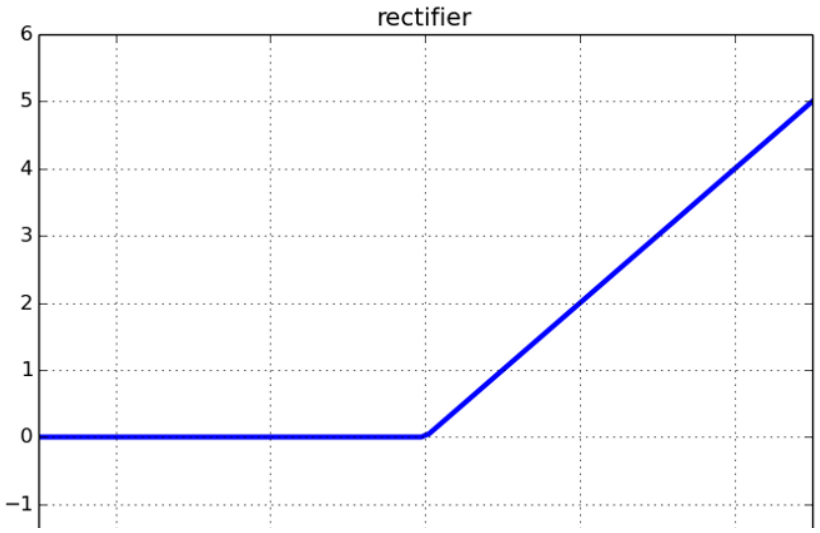
And finally the results for the rectified linear unit:

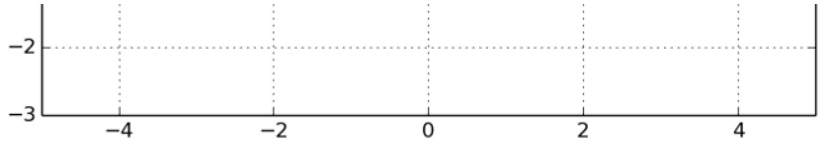




Mandelbrot set approximation using various neural networks with rectified linear unit activations

These images are also really good approximations of the Mandelbrot and we can note something really interesting. For smaller networks, the images produced are much more angular than with the hyperbolic tangent. I suppose that this phenomenon is due to the angle in the activation function:





rectified linear unit graph (from datascience.stackexchange.com)

As a conclusion, we can see that neural networks are capable of producing reasonable approximation of the Mandelbrot set. We can also note that the depth of a network increase its computation power much faster than the number of node in each of the layers.

To finish, we also validate that sigmoid activation, which were used at the beginning of the neural networks developments, are not used anymore for a good reason.

If you have any question, think that I missed something important or find an obvious mistake, do not hesitate to leave a comment.

Machine Learning Neural Networks Deep Learning Fractals

Mathematics