Introduction to neural ODE

Della Bona Sarah, Dumez Erika

6 février 2021

1 What are neural networks ?

Neural networks are popular types of machine learning models.**[insert formal definition of neural networks]**. Neural networks consist of a series of layers which are just matrix operations, and each layer introduces a little bit of error that compounds through the network as we propagate forward. So we can think of neural networks as a giant composite function of functions within functions, for as many layers as there are.

2 Residual neural network theory

Residual networks have the best accuracy. **Regarder à la def !** The only change to a regular neural network is that we not only feed the output of the previous layer to the next, but also the input of that layer **(en fait l'exo qu'on a fait c'était un residual network)**. It creates skip connections, so that the network can decide how deep it needs to be. As a formula, the $k+1$th layer has the formula :

$$x_{k+1} = x_k + F(x_k)$$

where F is the function of the kth layer and its activation. This simple formula is a special case of the formula :

$$x_{k+1} = x_k + hF(x_k),$$

which is the formula for the Euler method for solving ordinary differential equations (ODEs) when $h = 1$.

3 Ordinary Differential Equations

3.1 A refresher on ODE

Définition 1. Let $f : \Omega \subseteq \mathbb{R}x\mathbb{R}^N \to \mathbb{R}^N$.

A *first order ODE* takes of the form :

$$\partial_t u(t) = f(t, u(t)) \quad \textbf{(1)}$$

— A *solution* for (1) is a function $u : I \to \mathbb{R}^N$ where I is an interval of $\mathbb{R}$ such that :
— u is derivable on I,

— $\forall t \in I, f(t, u(t)) \in \Omega$,
— $\forall t \in I, \partial_t u(t) = f(t, u(t))$
— An *initial condition* (IC) is a condition of the type :

$$u(t_0) = u_0$$

where $(t_0, u_0) \in \Omega$ is fixed.

A *Cauchy problem* is an ODE with IC

$$\begin{cases} \partial_t u(t) &= f(t, u(t)) \\ u(t_0) &= u_0 \end{cases}$$

Définition 2. A *k-order ODE* is of the form :

$$\partial_t^k v(t) = g(t, v(t), ..., \partial^{k-1} v(t))$$

where

$$\begin{aligned} v &: I \to \mathbb{R}^N \\ g &: \Theta \subseteq \mathbb{R} \times \mathbb{R}^N \times ... \times \mathbb{R}^N \to \mathbb{R}^N \end{aligned}$$

It is not always possible to explicitly find a solution to a Cauchy problem, but we can compute a finite number of points $u_i \in \mathbb{R}^N$ which are close to the real solution.

More precisely, let $T \in \mathbb{R}$
$\{0\}$ such that the solution u exists on $[t_0, t_0 + T]$ and let $n \in \mathbb{N}^{\geqslant 2}$. We are then looking for $(u_i)_{i=0}^n$ s.t.

$$u_i \approx u(t_i) \quad \text{where } t_0 < ... < t_n \in [t_0, t_0 + T]$$

Let $h_i := t_{i+1} - t_i$, it is called the *step*. To compute those u_i, we use *1-step methods* such as Eular's method.

3.2 Eular's method

Eular's method is similar to a Taylor development, the idea is to compute $u(t_{i+1})$ using the following formula : [1]

$$u(t_{i+1}) \approx u(t_i) + h.\partial u(t_i)$$

where

$$\partial u(t_i) = f(t_i, u(t_i)).$$

4 Neural ODE

Useful for time series data better than recurrent neural network and their variants. Neural networks are function approximators. They are a series of matrix operations that we apply to matrices. So we have the input matrices, we apply these operations to it and we get an output, this output is the prediction.

1. We consider that $\forall i \ \{0, ..., n\}, h_i = h$.

We group these networks into what are called discrete layers. Layers are just a block of operations. Input times weight, add a bias, activate, repeat.

Here, let's consider a neural network as a continuous function with no grouping, no blocks. Instead of having single layers, we have the entire network be one continuous block of computation. This means that we do not need to specify the number of layers beforehand. We normally have to decide how many layers we want in our neural network, and then we can build the network. With this idea, instead of specifying the number of layers, we need to specify the desired accuracy of the function, and it will learn how to train itself within that margin of error.

Why does this matter ? It has more accurate results for time series predictions (i.e. continuous-time models like finance : how do stock prices change overtime, healthcare : how the patient's biometric data change over time). It also has a faster testing time than recurrent networks (but slower training time) so it's perfect for low power edge computing. (trade-off between precision and speed) Another point is that we can use entirely different optimization solvers (instead of gradient descent for example), here differential equation solvers, for which there's a hundred plus years of theory behind that has not been used yet for neural networks. Lastly, there's a constant memory cost, instead of linearly increasing the cost for each layers in a regular neural network.

The thing about neural network is that with these discrete layers, they expect the intervals for these time series data sets to be fixed (like everyday, every hour, every second). But that's not how the real world operates. Neural networks are bad at predicting output for time series data that is irregular (like in hospitals, doctors don't test a patient's biometric data at fixes time intervals).

Regarder le fonctionnement de la descente de gradient dans le cas où il y a plusieurs hidden layers !