Neural ODEs: Ordinary Differential Equations meets Machine Learning

Amit Kumar Jaiswal
UCL

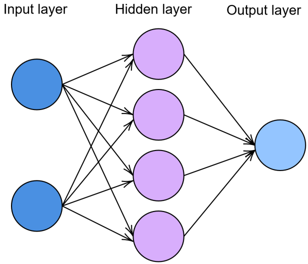
October 4, 2022

What are Neural ODEs?

- Neural ODEs[1] are deep neural network models using ordinary differential equations
- Unlike classical neural networks, the hidden layer in an Neural ODEs is defined as a black box that uses an ODE solver
- They have multiple advantages: constant memory cost, better results in continuous time series data and they are sometime more natural to use.

[1] Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. Advances in neural information processing systems, 31.

Neural Network - Forward and Backward Pass



The simplest
example of a neural network layer is

$$h = \sigma(Wx + b)$$

where σ is an activation function, W is a weight matrix and b a bias vector. The goal is to minimise the training error for every input of the training set. It requires derivatives computation of the loss with respect to the parameters.

Backpropagation [1]

Let θ be the parameters of the network. It needs θ^* which minimise the loss function in order to have the in-sample error as small as possible.

Compute the partial derivatives of the loss function with respect to the parameters, $\frac{\partial \mathcal{L}}{\partial \theta}$, and find θ^* such that these derivatives are 0.

Gradient Descent [2]

It works as follows: at each step of the process, it take a step in the opposite direction of the gradient of the function at the current point.

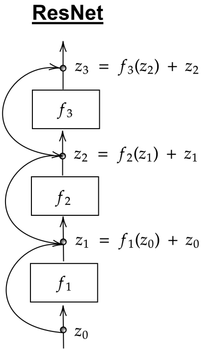
More formally, if a function $g : \mathbb{R}^m \to \mathbb{R}$, $m > 1$, differentiable and a point $x_0 \in \mathbb{R}^m$, we have that if

$$x_{n+1} = x_n - \gamma_n \nabla g(x_n), n \geq 0$$

for $\gamma_n \in \mathbb{R}^+$ small enough, then $g(x_n) \geq g(x_{n+1})$.

Residual neural network

A *residual neural network* [3], also called ResNet, is a neural network which has more connections. Indeed, a layer receives as input the outputs of the previous layer and its inputs.



In these networks, the output of the $(k+1)$th layer is given by

$$z_{k+1} = z_k + f_k(z_k)$$

, where f_k is the function of the kth layer and its activation.

First Order Ordinary Differential Equations

An *ordinary differential equation* (ODE) [4] is an equation that describes the changes of a function through time.

Definition

Let $\Omega \subseteq \mathbb{R} \times \mathbb{R}^N$ an open set. Let $f : \Omega \to \mathbb{R}^N$.
A *first order ODE* takes the form

$$\frac{\partial u}{\partial t}(t) = f(t, u(t))$$

A *solution* for this ODE is a function $u : I \to \mathbb{R}^N$, where I is an interval, such that

- u is differentiable on I,
- $\forall t \in I, (t, u(t)) \in \Omega$,
- $\forall t \in I, \frac{\partial u}{\partial t}(t) = f(t, u(t))$

ResNets and Euler

If we look back at the formula in the ResNet, we can see that this is a special case of the formula for Euler method

$$z_{k+1} = z_k + hf_k(z_k),$$

when $h = 1$.

Explicit and implicit layers

There is two different ways to define a layer : *explicitly* or *implicitly* [5]. When we define a layer explicitly, we specify the exact sequence of operations to do from the input to the output layer.

We can also define them implicitly: specifying the condition, we want the layer's output to satisfy.

Definition

An *explicit layer* is defined by a function $f : \mathcal{X} \rightarrow \mathcal{Y}$. For an *implicit layer*, we give a condition that a function $g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^n$ should satisfy. For example we can search for a y such that $g(x, y) = 0$.

Neural ODE

In a residual neural network, the output for an input x is a composition of functions. We want to extract all these individual layers to only have one "shared" layer.

Definition

A *neural ODE network* (or ODE-Net) [5, 6, 7] takes a simple layer as a building block. This "base layer" is going to specify the dynamics of an ODE.

ODE-Net enable us to replace layers of neural networks with a continuous-depth model.

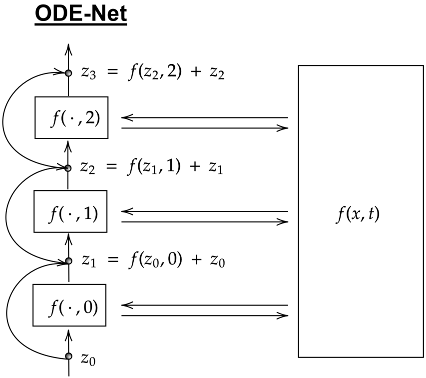
Comparison with ResNets

Let us return to ResNets to give intuition behind this definition. We know that any output of the k^{th} layer of a residual network can be computed with the function

$$F(z_t, t; \theta) = f(z_t, t; \theta) + z_t$$

where $t = k - 1$ and θ represents the parameters of the layers.

Thus, in the ResNet, the output for the input $z_0 = x$ is a composition of the functions $F(z_t, t; \theta)$.



We can then view the variables z_t as a function z of t. For example, $z_1 := z(1) = f(x, 0) + x$.

With that, we can write $F(z_t, t; \theta) = F(z(t), t; \theta)$.

We can see that in ResNets, the outputs of each layer are the solutions of an ODE using Euler's method. The ODE from which it is a solution is

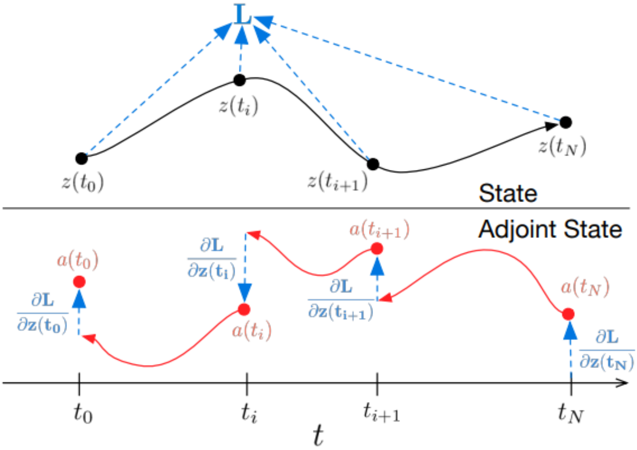
$$\frac{\partial z}{\partial t}(t) = f(z(t), t; \theta).$$

However, to find the solution to this Cauchy problem, we need the initial value of z, which is $z(t_0) := z_0 = x$. We obtain the following Cauchy problem:

$$\begin{cases} \frac{\partial z}{\partial t}(t) = f(z(t), t; \theta) \\ z(t_0) = x \end{cases} \tag{1}$$

Forward and Backward pass [7]

The output $z(t_N)$ of an ODE-Net with the input $z(t_0)$ is defined by the Cauchy problem (1), which depends on the parameters $z(t_0), t_0, \theta$.



Adjoint method

Let $\mathcal{L}$ be a loss function. To minimise this loss function $\mathcal{L}$, we need gradients with respect to the parameters $z(t_0), t_0, t_N, \theta$. To achieve that, we can determine how the gradient of the loss depends on the hidden state $z(t)$ for each t, which is

$$a(t) = \frac{\partial \mathcal{L}}{\partial z(t)} \tag{2}$$

This quantity is called the **adjoint**. We would like to determine its dynamics, so we need to compute its derivative with respect to t.

Adjoint method

With a continuous hidden state, we can write the transformation after an ε change in time as :

$$z(t+\varepsilon) = \int_{t}^{t+\varepsilon} f(z(t), t, \theta)dt + z(t). \tag{3}$$

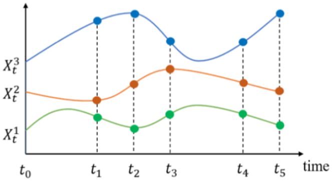
Let $G : \varepsilon \mapsto z(t+\varepsilon)$. We can apply the Chain rule and we have

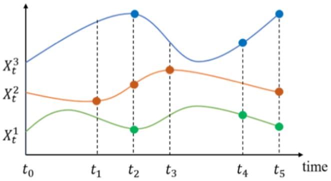
$$\frac{\partial \mathcal{L}}{\partial z(t)} = \frac{\partial \mathcal{L}}{\partial z(t+\varepsilon)} \frac{\partial z(t+\varepsilon)}{\partial z(t)}.$$

In other words

$$a(t) = a(t+\varepsilon)\frac{\partial G(\varepsilon)}{\partial z(t)}. \tag{4}$$

Modelling Irregular time series with measurements





Solutions exists for irregular synchronous time-series such as *Latent ODE, Latent SDE, Neural SDEs and Probabilistic Recurrent Network*

Our aim is to model irregular asynchronous time-series

Figure: Irregular time series with randomness in observations time-points, where X_t^1, X_t^2, X_t^3 represents measurements, including PSA, Gland volume and Max. tumour diameter

References I

Roger Grosse.
Csc321 lecture 6: Backpropagation.
http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/slides/lec6.pdf.

Yaser S. Abu-Mostafa, Malik Magdon-Ismael, and Hsuan-Tien Lin.
Learning from data: A short course.
2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
Deep residual learning for image recognition, 2016.

References II

David Francis Griffiths and Desmond J Higham.
Numerical methods for ordinary differential equations: initial value problems, volume 5.
Springer, 2010.

Zico Kolter, David Duvenaud, and Matt Johnson.
Deep implicit layers - neural odes, deep equilibirum models, and beyond.
https://implicit-layers-tutorial.org/.

Ayan Das.
Neural ordinary differential equation (neural ode).
https://ayandas.me/blog-tut/2020/03/20/neural-ode.html.

References III

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud.
Neural ordinary differential equations.
Advances in neural information processing systems, 31, 2018.

Cauchy Problems

Definition

An *initial condition* (IC) is a condition of the type

$$u(t_0) = u_0$$

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

Definition

A *Cauchy problem* is an ODE with IC

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

One-step methods

It is not always possible to explicitly find a solution to a Cauchy problem.

However, let $T > 0$ such that the solution u exists on $[t_0, t_0 + T]$ and let $n \geqslant 2$ be a natural. Let $t_0 < ... < t_n \in [t_0, t_0 + T]$ where $t_n = t_0 + T$. We obtain a finite number of points $(u_0, \ldots, u_n)$ such that:

$$\forall i \in \{0, \ldots, n\}, u_i \approx u(t_i).$$

To compute those points, we use *one-step methods* which compute the points u_{i+1} from the previous point u_i, the time t_i and the *step* $h_i := t_{i+1} - t_i$.

Euler's Method

Euler's method is a one-step method with a constant step h.
It is similar to a Taylor development, the idea is to compute $u(t_{i+1})$ using the formula

$$u(t_{i+1}) \approx u(t_i) + h\frac{\partial u}{\partial t}(t_i) \tag{5}$$

where

$$\frac{\partial u}{\partial t}(t_i) = f(t_i, u(t_i)).$$

for a function f.

Adjoint method

$$
\begin{aligned}
\frac{\partial a}{\partial t}(t) &= \lim_{\varepsilon\to 0^+} \frac{a(t+\varepsilon)-a(t)}{\varepsilon} \\
&= \lim_{\varepsilon\to 0^+} \frac{a(t+\varepsilon)-a(t+\varepsilon)\frac{\partial G(\varepsilon)}{\partial z(t)}}{\varepsilon} \\
&= \lim_{\varepsilon\to 0^+} \frac{a(t+\varepsilon)-a(t+\varepsilon)\frac{\partial z(t)+\varepsilon f(z(t),t,\theta)+O(\varepsilon^2)}{\partial z(t)}}{\varepsilon} \\
&= \lim_{\varepsilon\to 0^+} \frac{a(t+\varepsilon)-a(t+\varepsilon)(\mathbb{1}+\varepsilon\frac{\partial f(z(t),t,\theta)}{\partial z(t)}+O(\varepsilon^2))}{\varepsilon} \\
&= \lim_{\varepsilon\to 0^+} \frac{-\varepsilon a(t+\varepsilon)\frac{\partial f(z(t),t,\theta)}{\partial z(t)}+O(\varepsilon^2)}{\varepsilon} \\
&= -a(t)\frac{\partial f(z(t),t;\theta)}{\partial z(t)}
\end{aligned}
$$

Adjoint method

We now have the dynamics of $a(t)$

$$\frac{\partial a(t)}{\partial t} = -a(t)\frac{\partial f(z(t), t; \theta)}{\partial z(t)} \tag{6}$$

As we are searching for $a(t_0) = \frac{\partial \mathcal{L}}{\partial z(t_0)}$, we need to solve an ODE for the adjoint backwards in time because the value for $a(t_N)$ is already known. The constraint on the last time point, which is simply the gradient of the loss with respect to $z(t_N)$,

$$a(t_N) = \frac{\partial \mathcal{L}}{\partial z(t_N)},$$

has to be specified to the ODE solver.

Adjoint method

Then, the gradients with respect to the hidden state can be calculated at any time, including the initial value.

If we want to compute the gradient with respect to the parameters θ, we have to evaluate another integral, which depends on both $z(t)$ and $a(t)$,

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\int_{t_N}^{t_0} a(t) \frac{\partial f(z(t), t; \theta)}{\partial \theta} dt. \tag{7}$$

Adjoint method

To avoid computing each ODE on its own, we can do all of them at the same time. To do that we can generalize the ODE to

$$\frac{\partial}{\partial t}\begin{bmatrix} z \\ \theta \\ t \end{bmatrix}(t) = f_{aug}([z(t), \theta, t]) := \begin{bmatrix} f([z(t), \theta, t]) \\ 0 \\ 1 \end{bmatrix},$$

$$a_{aug}(t) := \begin{bmatrix} a \\ a_\theta \\ a_t \end{bmatrix}(t),\ a(t) = \frac{\partial \mathcal{L}}{\partial z(t)},\ a_\theta(t) = \frac{\partial \mathcal{L}}{\partial \theta(t)},\ a_t(t) := \frac{\partial \mathcal{L}}{\partial t(t)}.$$

Adjoint method

The jacobian of f_{aug} has the form

$$\frac{\partial f_{aug}}{\partial[z(t), \theta, t]}([z(t), \theta, t]) = \begin{bmatrix} \frac{\partial f}{\partial z} & \frac{\partial f}{\partial \theta} & \frac{\partial f}{\partial t} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}(t)$$

where each $\mathbf{0}$ is a matrix of zeros with the corresponding dimensions. We can inject a_{aug} in (6) and we get

$$\begin{aligned} \frac{\partial a_{aug}(t)}{\partial t} &= -[a(t)\ a_\theta(t)\ a_t(t)] \frac{\partial f_{aug}}{\partial[z(t), \theta, t]}([z(t), \theta, t]) \\ &= -\left[a\frac{\partial f}{\partial z}\ a\frac{\partial f}{\partial \theta}\ a\frac{\partial f}{\partial t}\right](t). \end{aligned}$$

Adjoint method

We can also get gradients with respect to t_0 and t_N by integrating the last component, $-a(t)\frac{\partial f(z(t),t;\theta)}{\partial t(t)}$, and by using the Chain rule. We have

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial t_0} &= a_t(t_0) = a_t(t_N) - \int_{t_N}^{t_0} a(t)\frac{\partial f(z(t),t,\theta)}{\partial t}dt; \\
\frac{\partial \mathcal{L}}{\partial t_N} &= \frac{\partial \mathcal{L}}{\partial z(t_N)}\frac{\partial z(t_N)}{\partial t_N} = a(t_N)f(z(t_N),t_N,\theta).
\end{aligned}$$

With this generalised method, we have gradients for all possible inputs to a Cauchy problem solver.