Summary of "Neural Ordinary Differential Equations" [2]

by Emma Lenze and Kai Rothe

The paper "Neural Ordinary Differential Equations" from 2019 by Chen et. al. [2] introduced a new family of deep neural networks, in short so called NeuralODEs.

1. Architecture of NeuralODEs

NeuralODEs can be thought of as generalizations of residual neural networks (ResNets). For $t \in \{0 \dots T\}$ with network depth T, ResNets (as well as RNN decoders and normalizing flows) repeatedly transform data $\boldsymbol{h}_t \in \mathbb{R}^N$ to a hidden state $\boldsymbol{h}_{t+1} \in \mathbb{R}^M$

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t). \tag{1}$$

by adding output of the activation function f with parameters θ_t to the output $\boldsymbol{h}_t$ of the skip connections. This can be seen as Euler discretizations of a continuous transformation. In the limit of continuously decreased step sizes and increasing layers the dynamics of the hidden units become continuous and their derivative can be described by an ordinary differential equation (ODE)

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta) \tag{2}$$

The fundamental idea of NeuralODEs is to parametrize the derivative of the hidden state using a neural network f and then compute the output of the neural network using a differential equation solver

$$\text{ODESolve}(\boldsymbol{h}(t_0), f, t_0, t_1, \theta) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t, \theta) dt \tag{3}$$

given the initial input $\boldsymbol{h}(t_0)$.

Algorithm 1 ResNets as Euler integrators [3]

function $f(\boldsymbol{h}, t, \theta_t)$
    return $\text{NeuralNetwork}_{\theta_t}(\boldsymbol{h})$
end function

function $\textsc{ResNet}_\theta(\boldsymbol{h})$
    for $t \in \{1, \dots, T\}$ **do**
        $\boldsymbol{h} \leftarrow \boldsymbol{h} + f(\boldsymbol{h}, t, \theta(t))$
    end for
    return $\boldsymbol{h}$
end function

Algorithm 2 Neural ODEs [3]

function $f(\boldsymbol{h}, t, \theta)$
    return $\text{NeuralNetwork}_\theta(\boldsymbol{h}, t)$
end function

function $\textsc{NeuralODE}_\theta(\boldsymbol{h})$

    $\boldsymbol{h} \leftarrow \text{ODESolver}(f, \boldsymbol{h}, t_0, t_1, \theta)$

    return $\boldsymbol{h}$
end function

Comparing algorithm 1 and 2 shows that NeuralODEs are "continuous depth-models" [2] in the sense that the depth $t \in \{1, \dots, T\}$ of a ResNet becomes a real-valued input, typically describing time. And the implicit Euler integration scheme of a ResNet is replaced by a general ODESolver, thereby increasing numerical accuracy [3].

2. Training NeuralODEs

Since the loss $L(\boldsymbol{h}(t_1)) = L(\text{ODESolve}(\boldsymbol{h}(t_0), f, t_0, t_1, \theta)$ depends only indirectly through the output of the ODE solver on f, the main technical difficulty Chen et. al. faced was training the neural network f without backpropagating through the black-box ODE solver, which has high memory cost and introduces numerical inaccuracy. Instead, the adjoint sensitivity method was used, and a PyTorch implementation released by the authors makes them easily accessible for various ODE solvers [1]. The main idea is to backpropagate the so called adjoint states

$$\boldsymbol{a}(t) = \frac{\partial L}{\partial \boldsymbol{h}(t)} \tag{4}$$

which dynamics (backwards in time) only depends on the derivative of f and not on the derivative of the ODESolver. The adjoint also determines $\frac{\partial L}{\partial \theta}$ independently of the derivative of the ODESolver.

3. Discussion and Applications of NeuralODEs

NeuralODEs allow training models with depth-independent memory cost, a bottleneck in training deep neural networks. After training, they allow the control and adaption of numerical accuracy (in a trade-off with computational cost) through the ODE solver, e.g. to decrease the computational cost during testing or in application. By comparing a small NeuralODE with a ResNet and an equivalent model trained by backpropagating through the operation of the ODEsolver, Chen et. al. were able to verify those claims and showed that NeuralODE save parameters while still achieving the same performance as a ResNet.

They also use the idea of continuous transformation of NeuralODEs to design a new class of scalable and invertible normalizing flows, which can directly train by maximum likelihood.

Concerning dynamical systems reconstruction, NeuralODEs easily train on irregularly-sampled data, e.g. due to missing values, without the need of binning input data beforehand and discretizing the latent dynamics. Compared to pure RNNs (where time-stemps have also given as input), generative NeuralODEs have been shown to achieve a lower predictive root-mean-square error, especially for irregular, and despite of noisy, data.

4. References

[1] Ricky T. Q. Chen. *torchdiffeq*. 2018. URL: `https://github.com/rtqichen/torchdiffeq`.

[2] Ricky T. Q. Chen et al. *Neural Ordinary Differential Equations*. 2019. arXiv: `1806.07366 [cs.LG]`.

[3] David Duvenaud. *Neural Ordinary Differential Equations at NeurIPS 2018*. URL: `https://www.youtube.com/watch?v=V6nGT0Gakyg`. (accessed: 24.02.2024).