Using Dynamical Systems to Help ResNets Learn

Presentation & Notes by Ian Joffe
Based on *Stable Architectures for Deep Neural Networks*
by Eldad Haber and Lars Ruthotto, 2017

Presented November 2023 for Stat 31410

1 Introduction

A neural network approximates (or "learns an approximation to") a function that maps features $Y_0 = [y_1, y_2, ..., y_s]^T$ to labels $C = [c_1, c_2, ..., c_s]^T$, where each sample may have many features and many labels. A neural network operates by sending the input Y through several layers of linear and nonlinear operations. Usually the last nonlinearity differs from other layers, but we will neglect that detail.

Each layer performs an affine transformation and then a nonlinear transformation on its input. For an N-layer neural network where the input and output have the same length and each layer has the same *width* (an unnecessary but simplifying assumption for the notation), we write that

$$Y_{j+1} = \sigma(Y_j K_j + b_j) \quad \text{for } j = 1, ..., N-1 \tag{1}$$

where Y_N is the output, the neural network's prediction for C. Each $K_j \in \mathbb{R}^{sxs}$ and $b_j \in \mathbb{R}^s$ are trained to perform the affine transformation at layer j. There are many choices for the nonlinearity σ. Today, the most common is the ReLU, but until recently researchers often used tanh.

The training of a neural network involves both *forward propagation* and *backward propagation*. After the 8-layer AlexNet first fully demonstrated the usefulness of neural networks for image recognition in 2012, practitioners figured they could continue to improve its accuracy by adding more and more layers. But in doing so, they ran into vanishing gradients, which I don't have time to cover in this presentation but concerns a stability issue during backward propagation, especially when using the tanh nonlinearity, and have their own fascinating interpretation via dynamical systems. One partial resolution came about in 2015, with the introduction of the residual neural network (ResNet), where the layers are put together with

$$Y_{j+1} = Y_j + h\sigma(Y_j K_j + b_j) \quad \text{for } j = 1, ..., N-1 \tag{2}$$

The *step size* h is a scalar weight, usually set as a hyperparameter. In a ResNet, each layer is the weighted sum of the current layer's output and the previous layer's output. While this was helpful in reducing instability due to vanishing gradients, it causes potential new instabilities during forward propagation, which is just the process of sending an input through the neural network. Resolving these instabilities is the motivation of Haber and Ruthotto's paper.

2 Instability of ResNets

While the ResNet equation (2) introduces new stability issues in forward propagation, it also implicitly hints at a path to a solution. The ResNet equation is the Euler discretization of the ODE

$$\dot{y}(t) = \sigma(K^T(t)y(t) + b(t)) \tag{3}$$

integrated from 0 to the number of layers N and with the input Y_0 as the initial condition. The ODE is stable if the real parts of the Jacobian of the RHS are nonpositive. This is true if $K(t)$ is sufficiently smooth and if, for eigenvalues λ,

$$Re(\lambda_i(K(t))) \leq 0 \quad \text{for } i = 1, 2, ..., s \text{ and } t \in [0, N]$$

Haber and Ruthotto consider three examples of constant $K(t)$ to illustrate their point

$$K_+ = \begin{bmatrix} 2 & -2 \\ 0 & 2 \end{bmatrix}, K_- = \begin{bmatrix} -2 & 0 \\ 2 & -2 \end{bmatrix}, K_0 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

The given stability condition may be one hypothesis for ensuring a ResNet is a good learner. To investigate, Huber and Ruthotto solve out the ODE each K, and plot the results for three initial conditions on their Figure (1).

K_+, with $\lambda_1 = \lambda_2 = 2$, is unstable about its equilibrium, and close initial conditions end up far away after being passed through the ODE. K_-, with $\lambda_1 = \lambda_2 = -2$ is stable about its equilibrium. Far-away initial conditions become much closer after being run through the ODE. K_0's equilibrium is Lyapunov stable, but not asymptotically stable. Its eigenvalues have zero real part, and its associated ODE performs *rotations* on initial conditions and tends to preserve distance between them.

When solving machine learning problems with deep ResNets, weight matrices like K_0 turn out to be superior. A learned model like a neural net should produce similar outputs from similar inputs, so K_+ fails. However, if it produces the same output for every input like K_-, it fails to be useful. K_0 strikes the correct balance, producing similar outputs from similar inputs and dissimilar outputs from dissimilar inputs. Thus, our true desired property is a modification on equation (3). We want some stability, but don't want forward propagation to be "overly stable", so we want

$$Re(\lambda_i(K(t))) \approx 0 \quad \text{for } i = 1, 2, ..., s \text{ and } t \in [0, N] \tag{4}$$

3 Proposed Solutions

Haber and Ruthotto propose a few alterations to the ResNet architecture to guarantee the proper amount of stability in forward propagation. Observe that K_0 from the example above is skew-symmetric. In fact, the real parts of the eigenvalues of any skew-symmetric matrix are always 0. So, consider revising the ResNet equation (2) to be

$$Y_{j+1} = Y_j + h\sigma(\frac{1}{2}Y_j(K_j - K_j^T) + b_j) \quad \text{for } j = 1, ..., N-1 \tag{5}$$

The modified weight matrix $K_j - K_j^T$ is guaranteed to be skew-symmetric and have a zero real part of its eigenvalues, so at any step in training the ResNet will hvae the correct level of stability. Note that Haber and Ruthotto also add some small quantity to the modified weight matrix to assist with other stability issues when discretizing or numerically integrating.

An alternative method to enforce skew-symmetry is to split each vector Y into x and z. Then re-concatenating x and z into one vector yields the augmented system

$$\frac{\partial}{\partial t}\begin{bmatrix} x \\ z \end{bmatrix}(t) = \sigma\left(\begin{bmatrix} 0 & K(t) \\ -K(t)^T & 0 \end{bmatrix}\begin{bmatrix} x \\ z \end{bmatrix}(t) + b(t)\right) \tag{6}$$

which is skew-symmetric, and therefore obeys (4) and has the proper amount of stability. This formulation is inspired by the symplectic structure of Hamiltonian systems, so clever integration techniques developed for such systems can be applied.

Of course, like any other real neural network, there is no guarantee that these architectures will be able to learn difficult relationships. But, they important offer advantages over vanilla ResNets that may may make training more effective. Section 6 of Haber and Rothotto's paper for figures that evaluate the effectiveness of their implementations.

It is not recommended to straightforwardly implement a ResNet using these techniques. While this work applies directly to the training of a *neural ODE*, the continuous analogue of a ResNet, alterations are needed to make things work in the discrete world. Haber and Ruthotto discuss this in detail, but time constraints require that I delay my discussion of it to another presentation.