Solving differential equations with unknown constitutive relations as recurrent neural networks

Tobias Hagge
PNNL
tobias.hagge@pnnl.gov

Panos Stinis
PNNL

Enoch Yeung
PNNL

Alexandre M. Tartakovsky
PNNL

Abstract

We solve a system of ordinary differential equations with an unknown functional form of a sink (reaction rate) term. We assume that the measurements (time series) of state variables are partially available, and use a recurrent neural network to "learn" the reaction rate from this data. This is achieved by including discretized ordinary differential equations as part of a recurrent neural network training problem. We extend TensorFlow's recurrent neural network architecture to create a simple but scalable and effective solver for the unknown functions, and apply it to a fedbatch bioreactor simulation problem. Use of techniques from recent deep learning literature enables training of functions with behavior manifesting over thousands of time steps. Our networks are structurally similar to recurrent neural networks, but differ in purpose, and require modified training strategies.

1 Introduction

Neural networks have an extensive history as tools for numerical solution of differential equations, with recurrent neural networks playing a role from the start. An early example was [6], which described how to construct, given a set of solvable difference equations, a Hopfield network, the minimal energy states of which are time series satisfying those differential equations. Scalability was advertised as a feature of these networks; the authors argued that their algorithm was highly parallel and relied only on simple operations.

We are interested in scalable solutions to the problem of training neural networks to estimate unknown functions in ordinary differential equations (ODEs)[2]. An early application of neural networks for this purpose is found in the 1992 work of Psichogios and Ungar [8], in which a fedbatch reactor model predicts total volume, substrate volume, and reactant volume over time. In this work, an ODE model is used to describe mass conservation with an unknown kinetic term μ, which is represented as a two-layer feed-forward neural network with sigmoidal activations. Psichogios and Ungar demonstrated that their model could learn μ correctly even when parameters are unmeasurable, but at the cost of solving additional sensitivity ODEs. Also, they gave their model an advantage by providing a training corpus over which the domain of μ was well-covered by the first few time steps, leaving open the question of whether long time-series behavior can be used to train an unknown function in a process of known functional form.

There are at least two motivations for "deep" hidden-state process models such as ours. One motivation is empirical: in some applications, some variables cannot be measured quickly and/or inexpensively thus are unsuitable as model input. A second motivation is statistical; noisy data can bias the training of a function, but but effects can be mitigated when long-term behavior is incorporated into the definition of loss.

In this work, we re-examine the problem of Psichogios and Ungar in light of recent advances in deep learning research. Psichogios and Ungar considered the network training problem as an application of the sensitivity equations, and it was necessary to integrate the neural network training software with a differential equation solver in order to backpropagate the difference of predicted and observed values. We recast the problem as a recurrent neural network training problem and extend TensorFlow's recurrent neural network architecture to solve it. This strategy eliminates the need for solving sensitivity equations as a part of the backpropagation procedure, and allows training functions using long time-series training sets with missing data. Our approach inherits the scalability advantages of the TensorFlow framework.

2 Implementation

Our solver implements a subclass EulerCell of the TensorFlow python class RNNCell. An instance of this class is passed to TensorFlow's dynamic_rnn method to construct a graph. Despite its name, dynamic_rnn is in fact an implementation of a batched discrete-time finite dynamical system. Our invocation constructs a graph that performs Euler integration of an ODE. Each "unrolled layer" in the constructed graph iterates a single timestep of a finite difference equation.

The resulting network is, formally, a recurrent neural network, and can be trained using backpropagation if the dynamics contain trainable parameters. We call it a *state-based* recurrent network as the internal state represents the evolving physical state of the system rather than a memory. For training purposes, state-based networks enjoy advantages over memory-based networks. In state-based networks, correct internal-state behavior is highly or entirely constrained by the dynamics and the data. Furthermore, though the networks are deep networks and there is the potential for issues with exploding and vanishing gradients [3], many important physical systems can be modelled by non-delayed differential equations, and in such systems the short-term dynamics of the system determine the long-term dynamics. Thus, state-based networks always have access to short-time feedback which informs training.

The difference between our approach and most control-theoretic approaches to process modelling lies in the difference in how the loss function is defined; in a process control application short-term errors are used to provide corrections, typically for the purpose of restricting a physical state space. We allow the process to continue without correction until termination and incorporate the long-term effects of model error into the loss function.

3 Fedbatch bioreactor model

As a proof of concept, we tested our approach on a system of ODEs describing the bioreactor of [8], which has dynamics described by the ODEs:

$$\frac{\partial X}{\partial t}(t) = \mu(t)X(t) - \frac{F(t)X(t)}{V(t)}, \tag{1}$$

$$\frac{\partial S}{\partial t}(t) = -k_1\mu(t)X(t) + \frac{F(t)(S_{in}(t) - S(t))}{V(t)}, \tag{2}$$

$$\frac{\partial V}{\partial t}(t) = F(t). \tag{3}$$

Here S is a substrate concentration, X is the reactant concentration, and V is the total material volume. The material is fed in with a known rate F and substrate concentration S_{in}. The reaction rate μ is assumed, for the model, to depend only on state variables X and S. Our goal is to learn an approximation of μ sufficient to predict S and X over a range of time steps, given initial state (X_0, S_0, V_0) and time-varying input (S_{in}, F). As in [8], we assume that the feed rate F is constant.

Following [8], for the ground truth we use the Haldane model:

$$\mu_g(X, S) = \frac{S\mu^*}{S + K_m + \frac{S^2}{K_i}}. \tag{4}$$

Here μ^*, K_m, and K_i are constants. Note that μ_g does not in fact depend on X; in the trained model this behavior must be learned.

If the domain of μ is well-covered by the initial time-steps of a long-time-series data set, as in [8], it is debatable, and situational, whether there is any point in training the series in their entirety, rather

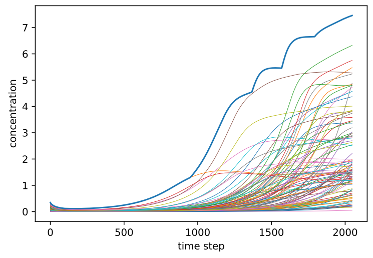

Figure 1: Concentrations of reactant X for several test-set samples. The topmost curve is the maximum over all samples.

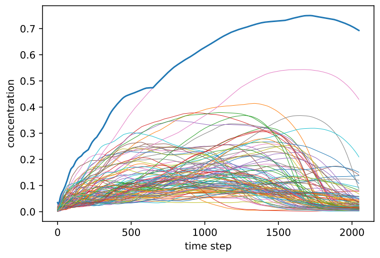

Figure 2: Concentrations of substrate S for several test-set samples. The topmost curve is the maximum over all samples.

than just over the initial time-steps. Accordingly, we have chosen values for k_1, μ^*, K_m, and K_i, along with distributions for the initial values S_0, X_0, V_0 and time-dependent input $(S_{in})_0$, that result in growth in the domain of S and X over a large number of time steps. Figures 1 and 2 show the substrate and reactant concentrations, respectively, for fifty samples from the training corpus.

4 Results

4.1 Methodology

Our networks were trained using Adam optimization [5], with a fixed learning rate of 10^{-4}, and L_2 loss. For easier comparisons between experiments, losses are averaged per-sample and per-timestep. Training, validation, and test sets each contained 1024 entries, consisting of a starting-state triple (X_0, S_0, V_0), and a time-varying-input vector S_{in} with 2048 entries. Initial state values were chosen from normal distributions centered at zero with variances $(.1, .01, 2)$; values for S_n were chosen by choosing $(S_{in})_0$ from a distribution and computing $(S_{in})_k$ by adding a normally distributed value with mean zero and variance .01 at each time step. The distributions were chosen to exercise interesting behavior during training, without regard for physical plausibility. We tried to ensure that novel values of the reactant concentration X and the substrate concentration S did not all occur within the first few hundred time steps. This has consequences for training: at the final time steps the most extreme-valued time series lie in regions of the domain of μ which is relatively poorly trained; and can be a significant source of loss.

Networks were trained until adequate performance (loss per sample per time step less than 3×10^{-5}) or improvement failure (12 epochs without improvement) were achieved. We did not encounter significant issues with generalization failure (the generalization ratio was greater than 2, see [7]).

4.2 Long-time runs

A two-layer feed-forward network model using ReLU activations was selected to represent the unknown function μ and trained using time series training data. It follows from [4] that with enough nodes in the hidden layer, any function on a compact domain can be well-approximated by a network of this type. Strictly speaking, the result in [4] only applies to bounded activation functions. ReLU activations, however, are Lipschitz functions, and one can apply the theorem after noting that $f_t(x) = ReLU(x+t) - ReLU(x)$ is bounded.

The network was trained in two stages. First, in order to produce a reasonable approximation of μ, the input data was coarsened to produce time series of length 256, spanning the same temporal interval. The network was trained on this input until termination (which was by improvement failure at about 3 times adequate performance). At this point, the original time series were used to train μ until improvement failure occurred (just above adequate performance loss). As in later experiments in [8], at both stages of computation, loss was computed only for S. This simulated the situation in which X is unmeasurable. Ground truth and predicted values for X and S for the median-loss test sample are shown in Figure 3.

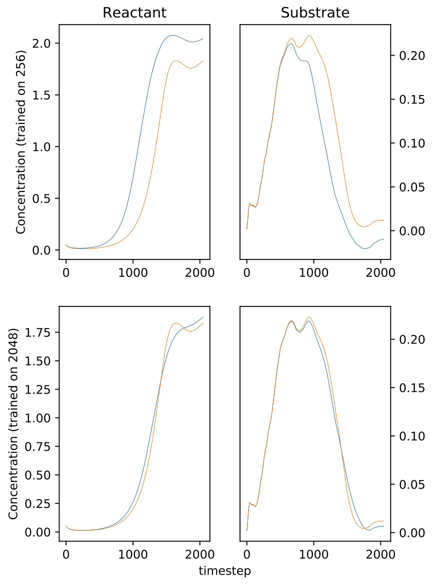

Figure 3: Median-loss time-series reactant (X) and substrate (S) concentrations for pretraining (256 time steps) and the full sequence (2048 time steps), with loss function computed from substrate concentration only. Orange is ground truth; blue is predicted.

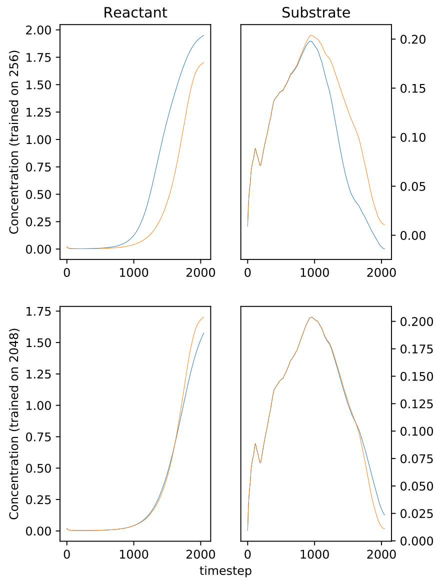

Figure 4: Median-loss time-series reactant (X) and substrate (S) concentrations for pretraining (256 time steps) and the full sequence (2048 time steps), with loss function computed at pretraining steps. Orange is ground truth; blue is predicted.

Next, the loss function was altered so that only every eighth time step, (i.e. each pretraining step) was counted in the loss function. Both X and S were factored into the loss. This simulated complete state measurements on a time scale too large to accurately capture correct behavior. The network was trained using the same two-stage procedure as before. The first stage terminated with improvement failure at about thirty-eight-times-adequate (squared) loss performance. This translates, at the finer time scale, to roughly five-hundred-times adequate loss; the network reaches improvement failure in stage two at about twenty times adequate loss. Time-series values for the median-loss sample are shown in Figure 4.

We have omitted graphs of trained μ vs ground μ. In this work, we consider the network to be trained to be the RNN, not μ. A priori there is no guarantee in our method that a correctly trained unknown function resembles the ground truth; there may be multiple unknown functions which produce correct dynamics. Also, the (four) graphs would require space to interpret; an analysis will appear in [2].

5 Discussion

Our networks have the usual issues with choosing good initializations. To establish a baseline, we used Glorot initialization [1]. Unfortunately, our network does not satisfy the hypotheses of [1]. Correct initialization choices depend on the (problem-specific) dynamics; developing an algorithm that makes these choices automatically is a direction of future research.

Our networks solve useful problems and provide use cases for very deep neural networks which are structurally simpler and easier to study than other use cases.

We are interested in applying our approach to applications in dynamical systems involving missing data, and in developing tools to work with PDEs. Also, as recurrent neural networks are formally quite similar to iterated difference equations, we are interested in exploring opportunities for technology transfer between the deep learning and numerical analysis communities.

References

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[2] T. Hagge, P. Stinis, E. Yeung, and A. M. Tartakovsky. Solving differential equations with unknown constitutive relations as recurrent neural networks. arXiv:1710.02242 – Update to appear.

[3] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[4] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, March 1991.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[6] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110 – 131, 1990.

[7] Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[8] Dimitris C. Psichogios and Lyle H. Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10):1499–1511, 1992.