Learning Evolution Operators of Partial Differential Equations with Spiking Neural Networks

Spencer Powers[1]

Faculty Advisor: Dr. Guillermo Reyes Souto[2]

Submitted to satisfy the requirements of the AME Capstone Course

March 28, 2021

[1]Department of Aerospace and Mechanical Engineering, Viterbi School of Engineering, University of Southern California

[2]Department of Mathematics, Dornsife College of Letters, Arts, and Sciences, University of Southern California

Abstract

Partial differential equations (PDEs) form an integral part of our understanding of the natural world. Common techniques using machine learning to predict the time evolution of systems governed by PDEs rely on prior knowledge of the equation structure, but a recent work estimated the evolution operator of an unknown PDE from time series data alone by working purely in modal space. While this recent approach used a conventional neural network architecture, it was hypothesized that the spatio-temporal nature of this problem allows it to be effectively learned by a biologically plausible spiking neural network (SNN). Using the SuperSpike learning rule, a multilayer SNN successfully learned the evolution operator of a one-dimensional heat equation problem from data alone in modal space.

Contents

List of Figures

1 Introduction

Partial differential equations (PDEs) govern a wide variety of fundamental processes in nature. Their influence spans fields such as fluid mechanics, thermodynamics, and quantum mechanics, among many others. Thus, it follows that the ability to predict how systems governed by PDEs evolve is of significant importance to engineers and scientists alike. The majority of recent efforts to apply machine learning to this problem have found success by estimating coefficients of terms from a dictionary using data alone [1, 2]. However, this approach is not ideal because the equation structure must be known ahead of time.

Wu et al. [3] took a unique approach to this problem by training a deep neural network to learn the evolution operator of an unknown PDE from data in modal space. While their approach does not allow for the explicit reconstruction of the PDE from the data, it provides a highly flexible method of predicting how the system evolves over time without prior knowledge of the equation structure.

Wu et al. chose a residual neural network (ResNet) for their task because this architecture has previously demonstrated an aptitude for learning unknown dynamical systems from data [3]. ResNets are an especially biologically–inspired variant of deep artificial neural networks. They take queues from pyramidal cells in the neocortex by allowing for skip connections, which allow for the output of one layer to be fed back into the input to the second-deeper layer [4]. As a result, the network attempts to minimize the "residual" or difference between the input and the underlying mapping instead of directly approximating the mapping. This was motivated by efforts to overcome the problem of vanishing gradients in training extremely deep networks [4]. ResNets are built up from modules called blocks, which can take a variety of forms. This architecture is visualized below:

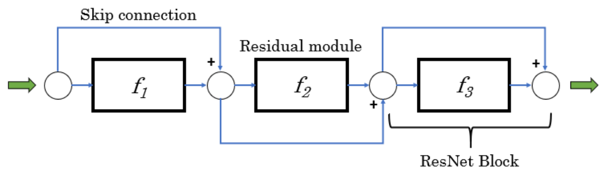


Figure 1: A standard ResNet architecture with three ResNet blocks (adapted from [5]).

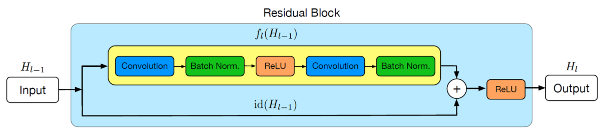


Figure 2: A sample configuration for the internals of a ResNet block [6].

While ResNets attempt to mimic brain-like computational architectures, they are limited in their similarity by their neuronal models. Artificial neural networks (ANNs) such as ResNets use neurons with continuous-output activation functions, such as the sigmoid or rectifier functions. Biological neurons, however, only produce activity in the form of spikes or electrical discharges when their membrane potentials reach a firing threshold. In reality, activity in neural circuits is sparse, meaning only a small fraction of all neurons are firing at any given time. Furthermore, ANNs have no concept of time because they run on the CPU clock cycle. Biological neural networks are asynchronous, event-driven networks, so information is encoded in the timings of the inputs, allowing rich information to be encoded in sparse activity.

The biological shortcomings of ANNs are resolved by spiking neural networks (SNNs), which are referred to as the third generation of neural networks [7]. SNNs use neuronal models directly drawn from computational neuroscience; the firing dynamics of each neuron are governed by differential equations that vary in complexity based on how biologically–realistic the model is. Due to their spiking nature, time now plays an important role in SNNs. For example, the build and decay of the membrane potentials of the neurons imbues them with a form of short-term memory, which makes them ideal for learning time–series problems [8]. Furthermore, since computations are only performed locally at neurons that fire, the number of computations required for learning tasks in SNNs is drastically smaller than ANNs; instead of processing every neuron in lower layers before sending information to higher layers, SNNs propagate spikes to higher layers asynchronously.

Spiking neural networks are a natural next step from the work of Wu et al. due to their choice of the ResNet architecture. Instead of approximating biologically–plausible neural circuits with ANNs, SNNs directly model neural circuits. In addition, the task of learning the evolution operator of an unknown PDE from temporal data falls into the class of time–series problems that SNNs are naturally suited for.

2 Objective

The goal of this research was to explore if spiking neural networks are capable of learning the evolution operators of partial differential equations from data alone. Learning these operators would allow for computationally-efficient prediction of the underlying dynamics of systems described by PDEs as they evolve over time if implemented on neuromorphic hardware.

3 Methods

3.1 Network Architecture

Because the SNN will be learning a temporal prediction problem, it is logical that the input and readout layers have the same shape; the output activity is redirected into the input layer when testing the trained network.

The network utilized for this project was a modified version of the network used by Zenke and Ganguli in their work presenting SuperSpike, a nonlinear, voltage-based, three-factor learning rule for supervised learning in multilayer spiking neural networks [9]. The code for the implementations presented in the original SuperSpike work ran on the Auryn SNN simulation library [10], and at the time of writing, the SuperSpike code base could only support a single hidden layer. Thus, given the short duration of this project, it follows that the network used in this project has the same constraint.

In addition, the same deterministic leaky integrate-and-fire neuron models used by Zenke and Ganguli were used for this project to allow the SuperSpike supervised learning rule to be implemented in a similar manner.

3.2 Encoding Scheme

Because the problem is cast in modal space, the network input and output will represent vectors of modal coefficients. A simple encoding scheme was adopted where the values of the coefficients are encoded into population response vectors by mapping digits to regions of 10 neurons, one for each digit. Figure 3 illustrates this approach:

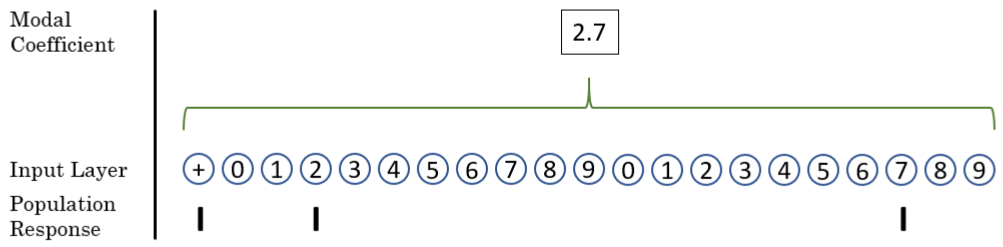


Figure 3: Digit encoding scheme. The bars in the population response level represent spikes.

The size of the input and output layers is then a function of the number of coefficients and how many significant digits are required for each coefficient. Note that while the number of neurons can grow to hundreds relatively quickly, the activity in the input and output layers is guaranteed to be sparse, which is characteristic of biological neural networks.

3.3 Learning Approach

The problem of learning the underlying evolution operator is cast as a spatio-temporal pattern learning problem, a class of problems at which SNNs have been proven to excel [9]. The SuperSpike learning algorithm is well suited for this task, as demonstrated in the following example from the original paper:

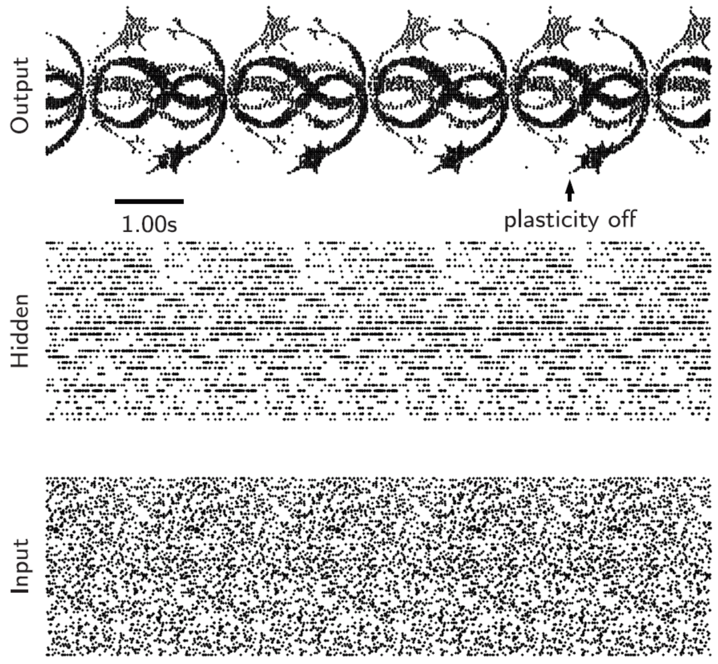


Figure 4: Example of network output at each layer over a number of exposure cycles after learning a spatio-temporal pattern, shown repeated in the output layer [9]. The initial output is near-random, but the SuperSpike algorithm trains the network to eventually produce the desired pattern.

For the specific problem of learning coefficients, the network will take in the numerical values of the coefficients at time t encoded via Figure 3 and output the coefficients at time $t + \Delta t$. In training, the output will be compared against the known coefficients for the next time step, while in testing the output will be directly fed into the input layer.

4 Learned Problems

4.1 One-Dimensional Heat Equation – Single Trajectory

The first PDE problem of interest was the one-dimensional heat equation with homogeneous Dirichlet boundary conditions and triangular initial conditions, due to its readily available analytical solution. This problem is summarized in Equation 1:

$$\begin{cases} u_t = \sigma u_{xx} \text{ in } (0,\pi) \times [0,\infty) \\ u(0,t) = u(\pi,t) = 0 \\ u(x,0) = f(x) \end{cases} \tag{1}$$

where

$$f(x) = \begin{cases} x \text{ for } x \in (0, \frac{\pi}{2}) \\ \pi - x \text{ for } x \in (\frac{\pi}{2}, \pi) \end{cases} \tag{2}$$

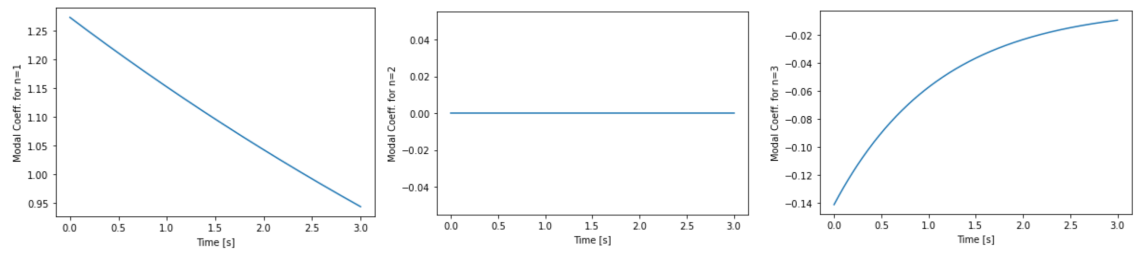
By choosing the basis of sines, the solution of Equation 1 takes the following form:

$$u_t(x,t) = \sum_{n=1}^{\infty} B_n sin(nx) e^{-n^2\sigma t} \tag{3}$$

where

$$B_n = \begin{cases} 0 \ \text{ if } n \text{ is even} \\ \frac{4}{\pi n^2} \ \text{ if } \ n = 1, 5, 9, ... \\ -\frac{4}{\pi n^2} \ \text{ if } \ n = 3, 7, 11, ... \end{cases} \tag{4}$$

The goal of the SNN is to learn the decaying Fourier coefficients in Equation 1, namely $B_n e^{-n^2\sigma t}$ for the the first three coefficients. These coefficients were computed for each time step and are visualized over the target 3 second interval in Figure 5:



(a) $n = 1$ (b) $n = 2$ (c) $n = 3$

Figure 5: Evolution of $B_n e^{-n^2\sigma t}$ for the first 3 modal coefficients over 3 seconds.

The input to the network is a temporal sequence of modal coefficient vectors, where the value of each coefficient is mapped to the input layer via an array of neurons representing digits (Figure 3). In the training process, the input will be a set of pairs of vectors. In each pair, the first element is the vector of known coefficients at time t for given boundary conditions. The second element is the vector of known coefficients at time $t + \Delta t$, and this is compared against the network output to compute the error.

Note that due to the predictive nature of the problem, the input and target spike rasters for the SuperSpike code base are near-identical; the latter is just the former shifted by one time step, and thus the former can only include spikes up until the second-to-last time step. For this reason, only the target spike raster is presented below:

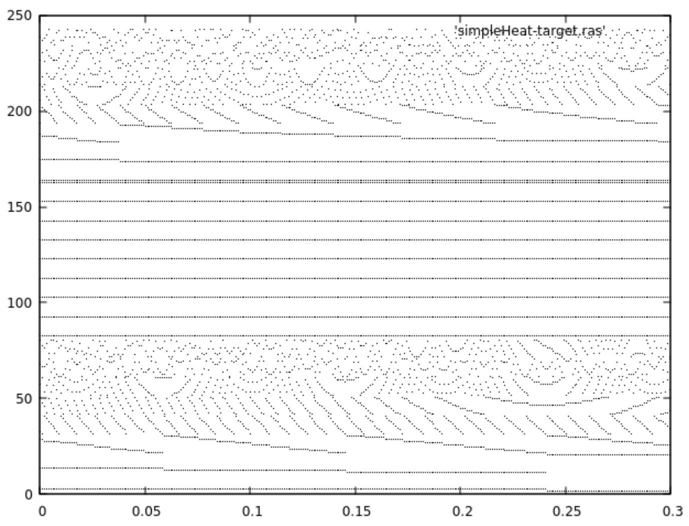


Figure 6: Target spike raster encoding the first 3 modal coefficients for a single trajectory over 3 seconds to be learned by the SNN. Y-axis denotes neuron ID, x-axis denotes time.

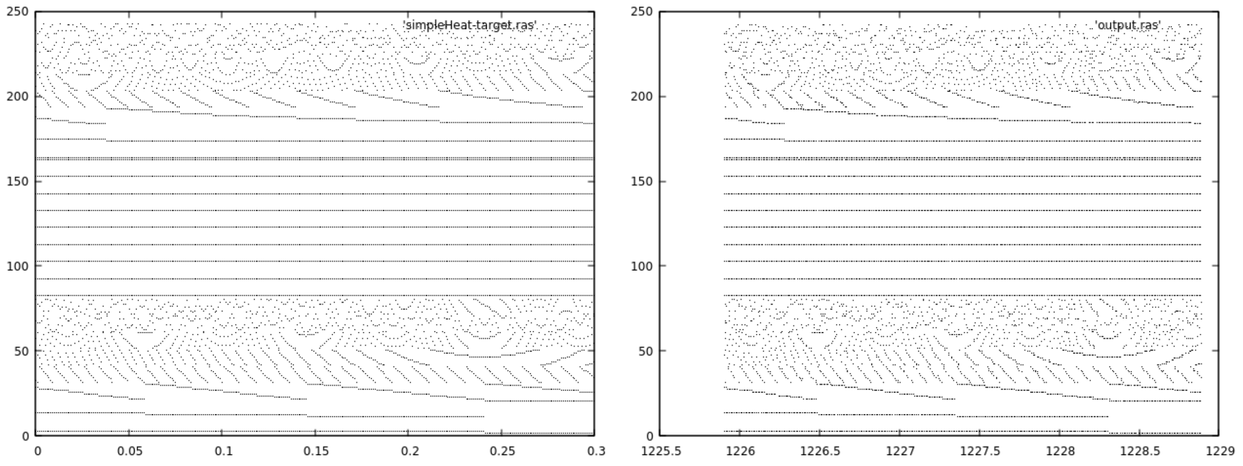
Note that Figure 6 shows a total of 243 neurons, meaning 81 neurons are used to represent a single coefficient. This corresponds to 7 digits after the decimal place, which ensures that the network is learning a one-to-one mapping, or that the activity pattern at each time step is unique. If fewer digits were used, this one-to-one mapping behavior would disappear at the end of the pattern due to the decaying rate of change.

The network parameters for the model that successfully learned this problem are presented in Table 1.

Table 1: Network parameters for learning the evolution operator of a single trajectory of the first three modal coefficients using the Auryn SNN simulation library.

Parameter	Value
Input/Output Layer Size	243
Hidden Layer Size	500
Number of Hidden Layers	1
Neurons Per Coefficient	81
Time Step	0.01 sec
Grid Length	2.99 sec
Simulation Time Per Block	29.9 sec
Number of Blocks	40
Learning Rate (η)	10^{-3}
SuperSpike Connection Parameter (ϵ)	10^{-12}

The SuperSpike Connection Parameter (ϵ) is used in place of zero in plasticity calculations to ensure that a divide-by-zero computation does not occur. Note that the values of η and ϵ are the standard values used in the original SuperSpike implementation.



(a) Target activity pattern. (b) Output layer activity after training.

Figure 7: Comparison of target and output layer activity patterns for the single trajectory problem. Y-axis denotes neuron ID, x-axis denotes time.

Figure 7 visually confirms that the network successfully learned the evolution operator for the first three coefficients. It should be noted that the SuperSpike implementation provided by Zenke and Ganguli only allows for visual confirmation of learning for spatio-temporal patterns. However, when comparing the unique patterns shared across the target and output layer activity at corresponding neurons, it is clear that the network as successfully learned the underlying operator required to produce the target pattern.

4.2 One-Dimensional Heat Equation – Multiple Trajectories

Wu et al. noted that training on a variety of coefficient trajectories improved generalizability, so this was a natural next step after learning a single trajectory. The problem setup is identical to the system presented in Eq. 1, except that the initial condition function $f(x)$ is randomly generated. More specifically, the values of B_n are randomly drawn from a uniform distribution between 0 and 1. The evolution operator is the same as in the single trajectory case, but now the network is tasked with learning from ten distinct trajectories at once. Using the same number of neurons for each coefficient as for the single trajectory (Table 1), the new target spike raster is presented below:

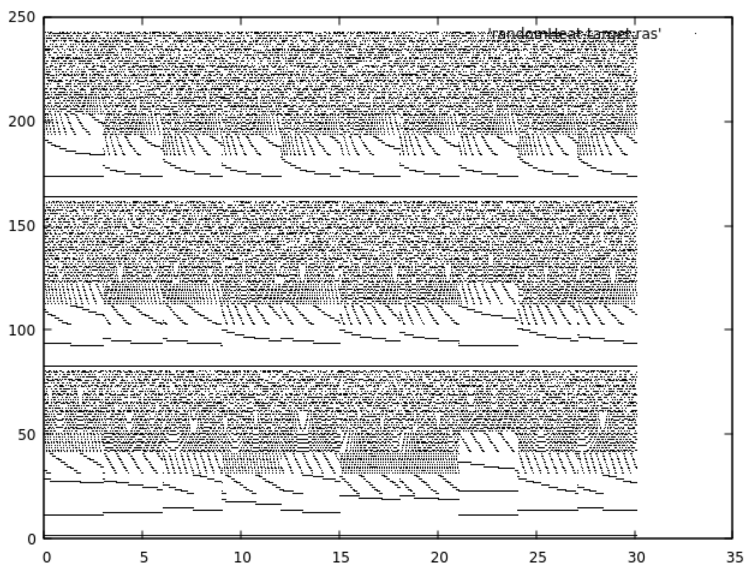


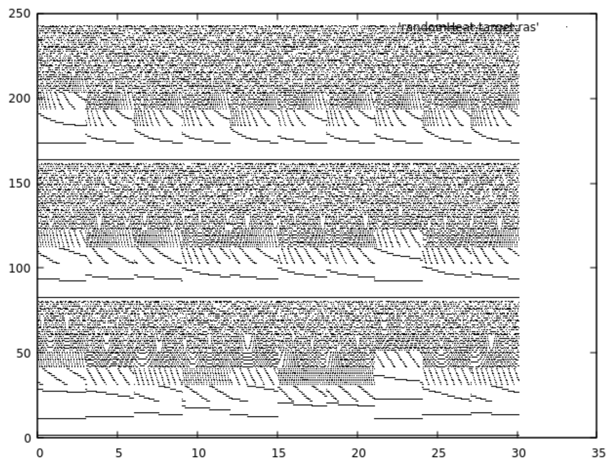
Figure 8: Target spike raster encoding the first three modal coefficients for 10 random trajectories over 3 seconds each to be learned by the SNN. Y-axis denotes neuron ID, x-axis denotes time.

Note that the 10 trajectories are easily distinguishable in Figure 8 as vertical slices. The network parameters used to successfully learn the evolution operator from 10 trajectories are shown in Table 2:

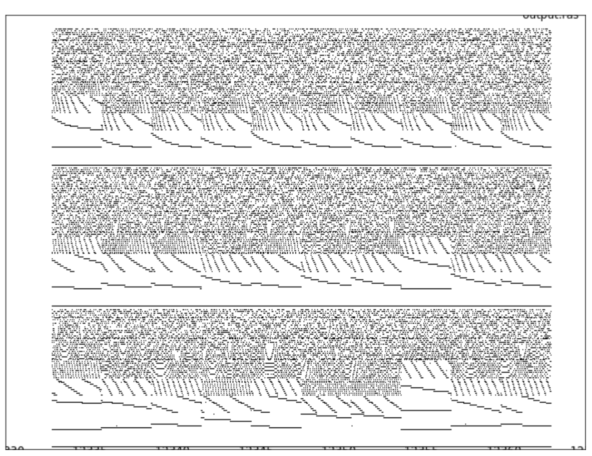
Table 2: Network parameters for learning the evolution operator of 10 trajectories of the first three modal coefficients using the Auryn SNN simulation library.

Parameter	Value
Input/Output Layer Size	243
Hidden Layer Size	500
Number of Hidden Layers	1
Neurons Per Coefficient	81
Time Step	0.01 sec
Grid Length	30.08 sec
Simulation Time Per Block	300.8 sec
Number of Blocks	40
Learning Rate (η)	10^{-3}
SuperSpike Connection Parameter (ϵ)	10^{-12}

Visual confirmation of successful learning is provided in Figure 9:



(a) Target activity pattern.



(b) Output layer activity after training.

Figure 9: Comparison of target and output layer activity patterns for the multiple trajectory problem. Y-axis denotes neuron ID, x-axis denotes time.

Axes were not rendered in Figure 9(b) due to issues with the gnuplot software, but examination of the raw output layer spike raster output shows appropriate neuron ID ranges. The time window displayed is the last 30.08 seconds of the training process, per the grid definition in Table 2.

5 Discussion

The SuperSpike learning rule was successfully used to learn the evolution operator for a one-dimensional heat equation problem solely from a set of modal coefficients over time. It was further demonstrated that the network could learn the correct evolution operator from multiple random trajectories to prevent potential overfitting on a single trajectory. However, the evolution matrix that the network learned in this particular problem was diagonal, which makes it a more easily learned task. Most physical systems will not have such simple evolution operators, so this should be seen as a stepping stone towards more realistic learning tasks involving unknown modal coefficient data.

6 Next Steps

The next task is to produce modal coefficients for the same heat equation problem using a different basis, thus requiring the network to learn a more complex and non-diagonal operator matrix. This can be done using existing spectral Galerkin method packages such as Shenfun (https://github.com/spectralDNS/shenfun).

In addition, learning more complex evolution operators will likely require more than one hidden layer, which is the limit currently set by the SuperSpike authors' software implementation of their learning rule. Thus, the source code provided by the authors must be modified to ensure proper functionality with multiple hidden layers.

As an additional test, the network would then be tasked to learn the Neumann problem for the same heat equation with arbitrary bases. Once the heat equation is rigorously tested through a variety of problem formulations and bases, the intuitive follow-on step is to consider equations that are more general but still linear. Such equations include the diffusion-absorption equation or the diffusion-convection equation.

Following demonstrated learning across multiple linear equations, the next task to be considered is to tackle one-dimensional nonlinear problems such as Burger's equation, with and without viscosity. The non-viscous case is particularly challenging because the network must learn to predict the formation of shocks, just as the ResNet proposed in [3] was able to do. Finally, any further steps would consider two-dimensional problems, likely starting from simpler problems such as the heat equation and increasing in complexity.

7 References

[1] A. Hasan, J. M. Pereira, R. Ravier, S. Farsiu, and V. Tarokh, "Learning partial differential equations from data using neural networks," *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020.

[2] Z. Long, Y. Lu, Z. Ma, and B. Dong, "Pde-net: Learning pdes from data," *International Conference on Machine Learning*, 2018.

[3] K. Wu and D. Xiu, "Data-driven deep learning of partial differential equations in modal space," *Journal of Computational Physics*, 2020.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[5] V. Fung, "An overview of resnet and its variants," *Towards Data Science*, 2017.

[6] J. D. Seo, "Implementing simple resnet (deep networks with stochastic depth) for mnist classification," *Towards Data Science*, 2018.

[7] N. Zheng and P. Mazumder, "Online supervised learning for hardware-based multilayer spiking neural networks through the modulation of weight-dependent spike-timing-dependent plasticity," *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[8] M. Pfeiffer and T. Pfeilr, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in Neuroscience*, 2018.

[9] F. Zenke and S. Ganguli, "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks," *Neural Computation*, vol. 30, pp. 1514–1541, Apr. 2018.

[10] F. Zenke and W. Gerstner, "Limits to high-speed simulations of spiking neural networks using general-purpose computers," *Front Neuroinform*, vol. 8, 2014.