Developing parameterizations for climate models using neural differential equations

Janak Agrawal[1] janaka@mit.edu
Manvitha Ponnapati[1] manvitha@mit.edu
Ali Ramadhan[1] alir@mit.edu

[1]Massachusetts Institute of Technology

December 12, 2019

Abstract

Climate models cannot resolve every cloud in the atmosphere and every wave in the ocean. The effect of small-scale physics such as cloud formation in the atmosphere and turbulence in the ocean must thus be parameterized. They represent a large source of uncertainty in climate models. In this project we investigate the feasibility of developing more accurate parameterizations for climate models using neural differential equations. The code is available on GitHub[1].

1 Introduction

Climate models attempt to accurately simulate the flow of the Earth's atmosphere and oceans over long time scales. Due to the short length and sparsity of observations, especially in the ocean, climate models are the main tool used by scientists to understand the Earth's climate and how it might change in the future. However, climate models can only use up a finite amount of computational resources and cannot be expected to resolve every cloud in the atmosphere and every wave in the ocean. Small-scale physics such as cloud formation in the atmosphere and boundary layer turbulence in the ocean are critical to the evolution of the climate and their effects must be accounted for. This is done by *parameterizations*, which is what the climate modeling community calls ad-hoc models used to represent the effects of unresolved physics upon the state of the atmosphere and oceans.

As expected, it is not easy to represent the effects of small-scale physics on the climate. The physics of cloud formation is not completely understood so any parameterization is developed with an incomplete understanding of what it must do. Turbulence remains an unsolved problem in classical problem so any parameterization for ocean turbulence similarly cannot fully rely on physical principles. As a result, climate parameterizations are a major source of uncertainty in climate models. While it is sometimes possible to resolve most of the physics accurately in high-resolution small-scale simulations it's computationally infeasible to simulate Earth's global climate at high-resolution.

Machine learning has been successfully used to learn the parameterization for coarse-grained models by observing data from high-resolution models for radiative transfer, convective transfer, and moist convection. While these models are promising, the neural network architectures used are inflexible and can not incorporate existing deterministic equations to develop and scale hybrid models. Neural differential equations combine differential equations that describe physics and neural networks in an intelligent way. In this work, we discuss the current methods for parameterization, introduce neural differential equations and provide a repository with Julia notebooks for applications of neural ODEs to climate parameterization.

[1]GitHub repository: `https://github.com/ali-ramadhan/6S898-climate-parameterization`

2 Could machine learning help tackle climate parameterization?

Machine learning, particularly neural networks are theoretically universal approximators for any non-linear function given the right hyper parameterization. But can the machine learning model underlying non-linear functions of many real-world physical processes involved in climate models? Research shows that by choosing the right architecture, it is possible to parameterize these processes with machine learning under certain constraints. In this section, we will review some of the existing works for using machine learning to parameterize radiative transfer, moist convection etc

Climate parameterizations are a major source of uncertainity in climate models. One of the physical processes that if parameterized accurately would reduce this uncertainty is convection. But at convection-permittive scales, it is shown that at such a place scale it's possible to parameterize the physical processes accurately. But running such high-resolution simulation at a global process is computationally intensive. It was shown by Gentine et al,2018 that is possible to use a simple fully connected artificial neural network to model temperature and humidity tendencies using temperature, humidity, sensible heat flux, latent flux , surface pressure and meridional wind as inputs to the network. This greatly reduces the amount of data required to parameterize the climate process. Gentine et al, 2018 found that high-frequency global training data was enough for convective parameterization. Turns out this. While this model provided a proof of concept for the use of machine learning for parameterization, it still had some issues in regards to energy and moisture conservation.

Works by Rasp et al,2018 built upon this idea to create a deep neural network for sub-grid parameterization. The network work inputs were similar to the ones described above but outputs included shortwave and longwave flux at TOA and surface, heating rate, moistening rate, and precipitation. The network is trained on inputs from cloud-resolving models which are expensive to run globally for more than a couple of years.The model was able to predict stable mean climate, precipitation extremes and tropical waves. It was also interesting to note that the model learned to conserve energy without explicitly being told to. On the negative side the moisture conservation, positive cloud water concentrations etc was still infeasible. It was also shown that the approximation of parameterizations by neural networks was too smooth and didn't capture all the small variabilities which contribute to uncertainties in the climate models.

The above problems with representing existing physics in neural networks is fundamental to the architecture of neural networks. This is because most often neural networks are used as black boxes to find the nonlinear approximator without taking into account any of the physics. Some models used for climate parameterization have gotten around this issue by choosing a model architecture that fits the needs of the physical process they are trying to parameterize. One example of this is the random forests model used by O'gorman et al, 2018 for moist convection parameterization from RAS simulations. Since random forests take means of subsets of trees from the training data, they automatically preserve energy and ensure surface precipitation non-negativity.

Brenowitz et al ,2018 took a different approach to ensure better models for parameterization by designing a mass-weighted loss. When a neural network with mean squared error as loss is used to predict the tendencies at the next time step, it starts accumulating error for every wrong prediction at a timestep. Instead, Brenowitz et al, created a user defined hyper parameter T which is the number of time steps over which a collective loss is evaluated. The loss function is shown below

$$J(\alpha) = \frac{1}{n_x n_y (n_t - T + 1)} \sum_i \sum_{n=1}^{n_t - T + 1} \sum_{m=0}^{T-1} ||\mathbf{x}_i^{n+m} - \mathbf{F}_\alpha^m \mathbf{x}_i^n||_W$$

The W-norm is defined as, with M as mass of an atmospheric column. J, the loss is similar to the mean absolute deviation

$$\left\| \begin{bmatrix} \mathbf{q} \\ \mathbf{s} \end{bmatrix} \right\| = \frac{1}{M\sigma_q} \int \mathrm{d}z \rho_0 |q(z)| + \frac{1}{M\sigma_s} \int \mathrm{d}z \rho_0 |s(z)|.$$

By using the loss above, and adding the neural network, the advection forcing sequentially - So each time step of the neural network looks like

$$\mathbf{x}^* = \mathbf{x} + \Delta t \frac{\mathbf{g}(t) + \mathbf{g}(t + \Delta t)}{2}$$

$$\mathbf{F}_\alpha \mathbf{x} = \mathbf{x}^* + \Delta t \mathbf{f}_{\mathrm{NN}}(\mathbf{x}^*; \alpha).$$

It was shown that this model performed better than a state of the art traditional climate models

While the models discussed showed promise for machine learning in climate parameterizations, we believe the inability to easily incorporate existing physics into the neural networks makes the case for utilization of neural differential equations architecture described in the next section.

3 Neural ODEs

Existing data-driven parameterization approaches tend to operationally fail when plugged into a climate model due to the lack of ability of data-driven models to generalize to extreme as well as unseen initial conditions. We hypothesize that Neural Differential Equations (NDEs) provide a more powerful framework for designing new procedures where we can use data from high-resolution climate models. Additionally, the NDE framework allows us to utilize insights from dynamical systems that may produce a neural network better suited for long integration times as in a climate model.

Neural Ordinary Differential equations have attracted significant attraction and a paper describing how neural networks can be modelled as ODE and solved with ODE solvers was recognized as one of the best papers in NeurIPS 2018. Neural Differential Equations architecture provides a better approach for parameterization because of its ability to mix ODE and neural networks. These leads to a better framework for scientific computing for climate models and will reduce the dependence on having to produce tons of data if were to use pure neural networks to model the complicated non-linear dynamics of climate.

With Neural ODEs we can replace certain terms in ODE with a neural network/universal approximator and introduce and study the effects of delay and stochastic events on climate models. In a typical machine learning problem, you are given some input x and you want to predict an output y. This generation of a prediction y from x is a machine learning model (let's call it ML). During training, we attempt to adjust the parameters of ML so that it generates accurate predictions. We can then use ML for inference (i.e., produce ys for novel inputs x). This is just a nonlinear transformation $y = ML(x)$. Specifically, $ML(x) = \sigma(W_3\sigma(W_2\sigma(W_1x)))$ is a three-layer deep neural network, where $W = (W_1, W_2, W_3)$ are learnable parameters. You then choose W such that $ML(x) = y$ reasonably fits the function you wanted it to fit. The theory and practice of machine learning confirms that this is a good way to learn non-linearities.

Machine learning models grow and are hungry for larger and larger amounts of data, differential equations have become an attractive option for specifying nonlinearities in a learnable (via the parameters) but constrained form. They are essentially a way of incorporating prior domain-specific knowledge of the structural relations between the inputs and outputs. Given this way of looking at the two, both methods trade-off advantages and disadvantages, making them complementary tools for modeling.

The inspiration of Neural ODEs comes from a neural network architecture known as ResNET. A ResNET differs from other neural network architectures in the form that the input of the previous layer in the neural network is also passed as an input to the current layer as shown in Figure 1. Let l_t be the output of layer t in an arbitrary neural network, then

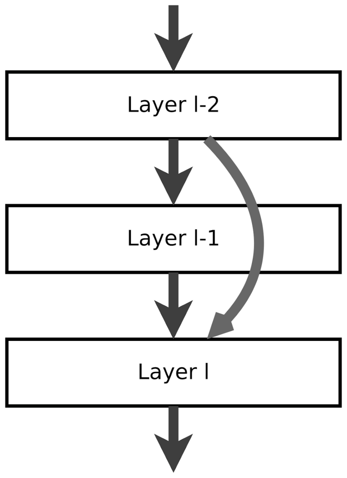
In a traditional neural network:

$$l_{t+1} = W_{t+1} * l_t \tag{1}$$

but for a Residual Network (ResNET):

$$l_{t+1} = W_{t+1} * l_t + l_{t-1} \tag{2}$$

Figure 1: ResNET



The equation for a ResNET can be rewritten as:

$$l_{t+1} = ML_\theta(l_t) + l_{t-1} \tag{3}$$
$$\Delta l_t = ML_\theta(l_t) \tag{4}$$

This is equation is of the same form as the forward Euler method for numerically solving ordinary differential equations:

$$\Delta y = \Delta x f(x) \tag{5}$$

This looks similar in structure to a ResNet, one of the most successful image processing models. The insight of the the Neural ODEs paper was that increasingly deep and powerful ResNet-like models effectively approximate a kind of "infinitely deep" model as each layer tends to zero. Rather than adding more layers, we can just model the differential equation directly and then solve it using a purpose-built ODE solver.

For example, lets consider the following the following class of ODEs:

$$\frac{du}{dt} = \nabla u(t,x)\mu(t,x) + f(t,x,\sigma(t,x)) \tag{6}$$

We assume that we do not know the exact form of $\sigma(t,x)$ and therefore, we need to parameterize it. We can approximate it by using a universal function approximator such as deep neural network.

Therefore, introduce the approximation:

$$\sigma(t,x) = NN(t, x_t|\theta_1) \tag{7}$$

where NN is a universal approximater defined by the parameters θ_1.

Therefore, our equation becomes:

$$\frac{du}{dt} = \nabla u(t,x)\mu(t,x) + f(t,x,NN(t,x_t|\theta_1)) \tag{8}$$

which we can solve to determine the solution of the ODE as well as the parameterization θ_1 of the NN using an ODE solver.

4 Implementation in Julia

As it is widely known, training deep learning models can require significant computational resources making the choice of language to implement this system an important one. We chose Julia language because of the abundance of pure-Julia packages for both machine learning and scientific computing allowing us to test our ideas on fairly diverse problems and the extensive performance bench marking of Julia in training machine learning models compared to languages like C++, Fortran etc.. It also includes other modern features such as GPU acceleration, distributed parallelism and sophisticated event handling.

Within Julia, DiffEqFlux.jl is a library for fusing neural networks and differential equations. This is the first toolbox to combine a fully-featured differential equation solver library and neural networks seamlessly together. It incorporates DifferentialEquations.jl defined differential equation problems into a Flux.jl defined neural network, and vice-versa.

DiffEqFlux.jl also supports the ability to fuse neural networks with stochastic differential equations, delayed differential equations, stiff equations and different methods for adjoint sensitivity analysis calculations. This is a large generalization of the Neural ODE work which is very helpful in integrating Neural ODEs in climate models.

5 Numerical Analysis

In the paper, we try to come up with an efficient and robust parameterization for the reduced Navier-stokes equation in one-dimension. Specifically, the Navier-Stokes equation in two dimensions is:

$$u_t + NN_1(uu_x + vu_y) = -p_x + NN_2(u_{xx} + u_{yy})v_t + NN_1(uv_x + vv_y) = -p_y + NN_2(v_{xx} + v_{yy}) \tag{9}$$

where $u(t, x, y)$ is the x-component of the velocity field, $v(t, x, y)$ is the y-component of the velocity field, $p(t, x, y)$ is the pressure and $\lambda = (\lambda_1, \lambda_2)$ are unknown parameters.

Solutions to the Navier-Stokes equations are searched in the set of divergence-free equations:

$$u_x + v_y = 0 \tag{10}$$

Given noisy measurements t, x, y, u, v at N time steps of the velocity field, the goal is to find a suitable parameterization over the parameters λ.

We reduced these equations to a single dimension and focused on predicting the time derivative of the Temperature profile to demonstrate the viability of Neural ODEs as an efficient parameterization of mesoscale eddy currents.

Therefore, what climate models usually end up solving for mesoscale eddy currents is:

$$\partial_t T + \boldsymbol{u} \cdot \nabla T = \nabla \cdot (\kappa \nabla T) + \frac{Q}{\rho c_p} \tag{11}$$

Which when averaged over certain grid-cells transforms into:

$$\partial_t \overline{T} + \boldsymbol{u} \cdot \nabla \overline{wT} = \nabla \cdot (\kappa \nabla \overline{T}) + \frac{Q}{\rho c_p} \tag{12}$$

We then further transform these equations into the following parameterizations for our Neural ODE solvers:

1. $\partial_t \overline{T} = \mathbb{NN}(T, ...) + \dfrac{Q}{\rho c_p}$
2. $\partial_t \overline{T} = \partial_z \left[\mathbb{NN}(T, ...)\right] + \dfrac{Q}{\rho c_p}$

where $\mathbb{NN}(T, ...)$ is a deep learning architecture with input variables like temperature T etc. Here it is assumed that everything not explicitly addressed in parameterized equations from the original equations is absorbed in the deep learning model.

6 Methodology

High-resolution climate models require a lot of computational resources to run which is why scientists often prefer running low-resolution models which are much easier and faster to compute. But these low-resolution models often approximate some hard to calculate quantities, a technique widely referred to as parameterization. These parameterizations are often hand-tuned making them prone to errors. We try to develop a computer-trained machine learning parameterization for the low-resolution models.

We divided our Neural ODE training pipeline into three different stages to make it efficient to train and validate Neural ODE models:

6.1 Data generation from climate models

To train the deep learning architecture in the Neural ODE, we need high amounts of training data which includes data regarding the different variables such as temperature $T(\boldsymbol{x}, t)$ and velocity $\boldsymbol{u}(\boldsymbol{x}, t)$. Generating this training data from low-resolution climate models would require us to use an existing parameterization which is not ideal for our case since the machine learning model would just learn the structure of the parameterization used instead of coming up with its own parameterization.

Therefore, we generate training data from high-resolution ocean simulations where we don't need any such parameterization and then convert it into low-resolution data by averaging variables over certain grid points and then reducing it to a single dimension. This data is suitable for training our machine learning models as it does not require using an existing parameterization.

The high-resolution ocean simulations were performed using `Oceananigans.jl` [2], a Julia package for simulating incompressible fluids, such as the ocean, on GPUs.

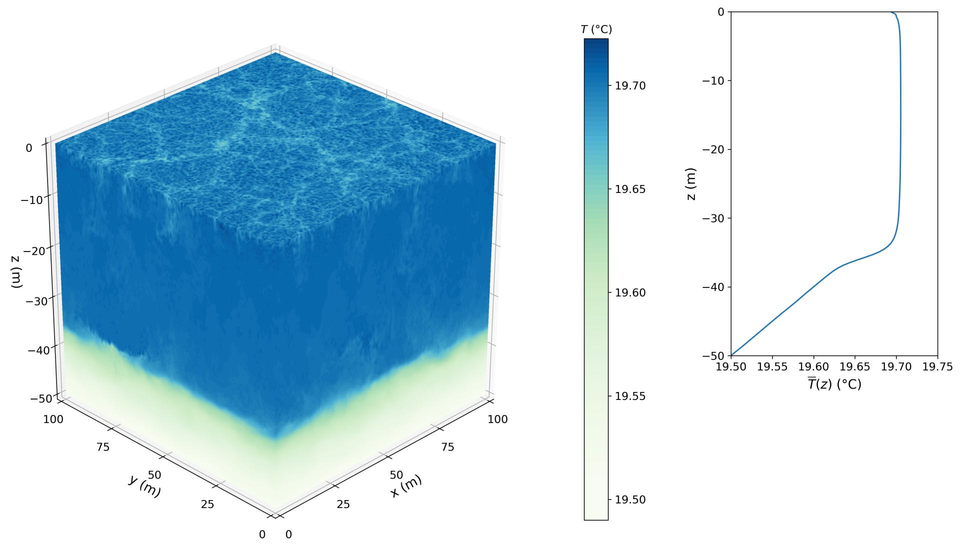


Figure 2: On the left is a snapshot of the temperature field from one of the high-resolution simulations. Boundary layer turbulence and convection sets up a *mixed layer* in the upper layer of the ocean where temperature is homogenously mixed. On the right is a horizontal average of the temperature showing that while the 3D field is turbulent, the statistically averaged temperature of that turbulence produce a coherent vertical profile. It is this vertical profile that we are interested in preidcting.

Training Neural ODEs We use the coarse-grained or low-resolution data generated from the climate models to train different deep learning and Neural ODE architectures on equations mentioned in

[2]`https://github.com/climate-machine/Oceananigans.jl`

section 5. The training data contains a number of variables which can be used in training the neural networks. We divide the training into train and test datasets to calculate the model accuracy.

Validating the models Once a data-driven model has been developed, we will assess its predictive skill by inserting it into a low-resolution ocean simulation. Usually models derived from data-driven approaches leads to blowing up of climate simulations because of the chaotic nature of the system. Therefore, we assess the quality of the parameterization by running the low-resolution ocean simulation with our Neural ODE model and assess how well it simulates the mesoscale eddie currents compared to high-resolution data. We also make sure that the data-driven model doesn't blowup when plugged into the low-resolution climate model.

7 Experiments

In the paper, we test different Neural ODE models such as a standard neural ODE where we model the derivative as a universal approximator and mixed neural ODE where we only approximate certain terms in the ODE with a machine learning model. To compare these models against standard machine learning parameterization that has shown to work in the prior literature.

For all models we use data generated from `Oceananigans.jl` on a 3 dimensional grid of 100 m $\times$ 100 m $\times$ 100 m divided into 256 $\times$ 256 $\times$ 256 grid cells for 8 days of simulation time with output at 10 minute intervals. We use this data to generate coarse resolution data in one dimension in vertical direction of length 100 m divided into 32 grid cells by averaging over all properties in space. This dataset contains a total of 1153 data points spread evenly across time. We use the same dataset for training all the models mentioned here.

7.1 Time series prediction models

Some parameterizations are time series prediction problems with spatial and temporal dependencies. These kinds of problems typically arise in traffic flow prediction problems. We first trained a multivariate LSTM network to predict the next time step of the temperature. But our model had a large mean square error after second test step on the free convection graph shown below. One improvement we hope to try to fix this issue in the future is by changing the loss to something similar to the mass weight loss over a few timesteps like in Brenowitz.C et al, 2018

The next model we tried was diffusion convolution recurrent neural networks used for data driven traffic forecasting by Li et al,2018. The network models the spatial/temporal time series prediction problem as a diffusion process on a directed graph. This model was shown to predict abrupt changes very well in traffic flow prediction problems so we hypothesized that it can capture variability in climate parameterizations better than the neural network architectures in the previous works. We tested this model on a toy problem to see if it captures the variability of the time series model well.

Again, we believe most climate parameterization black box models will benefit from a compounded time series loss and hope to implement this in our future models.

7.2 Neural ODE

We first tried a complete black box neural network $\mathbb{NN}(T)$ replacing the right hand side taking the temperature T as input. The neural differential equation can then be expressed as

$$\partial_t T = \mathbb{NN}(T) \tag{13}$$

so the neural network knows zero physics. As expected such a network did not perform well (figure 4). Presumably the only way to improve this neural network is to show it more data and increase the size of the network as it is responsible for learning all the physics from the data. We should be able to specify some of the physics so that the neural network is only responsible for resolving the unknown physics. However, it is stable during time-stepping which is a nice feature.

We then tried using a neural differential equation with a little more physics

$$\partial_t T = \partial_z \left[\mathbb{NN}(T)\right] + \frac{Q}{\rho c_p} \tag{14}$$

so that the neural network is only trying to predict the convection term and knows about surface cooling Q. It turned out to be necessary to *pre-train* this network on $(T, -wT)$ data pairs before

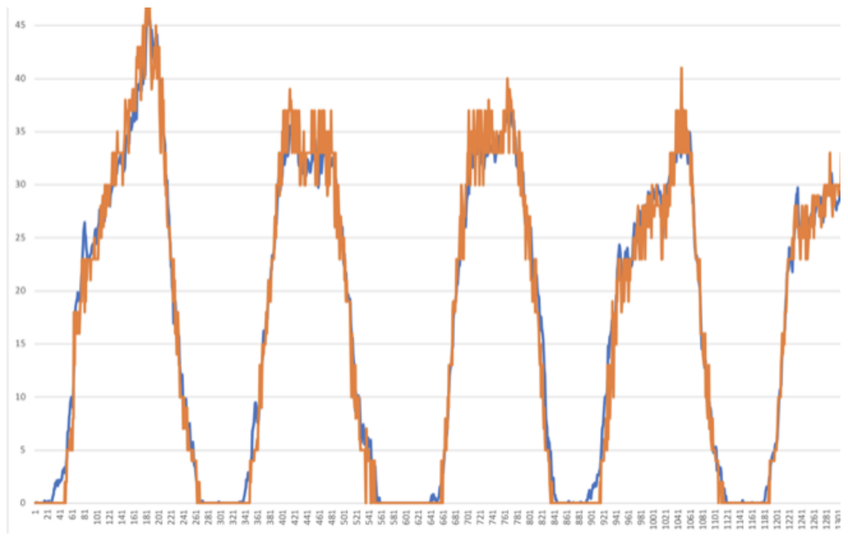


Figure 3: Time series prediction results. Orange - predicted timeseries model, blue - Actual time series curve. Even though the figure shows a good model, the RMSE of the model was very high and didn't generalize very well to the validation hold out set

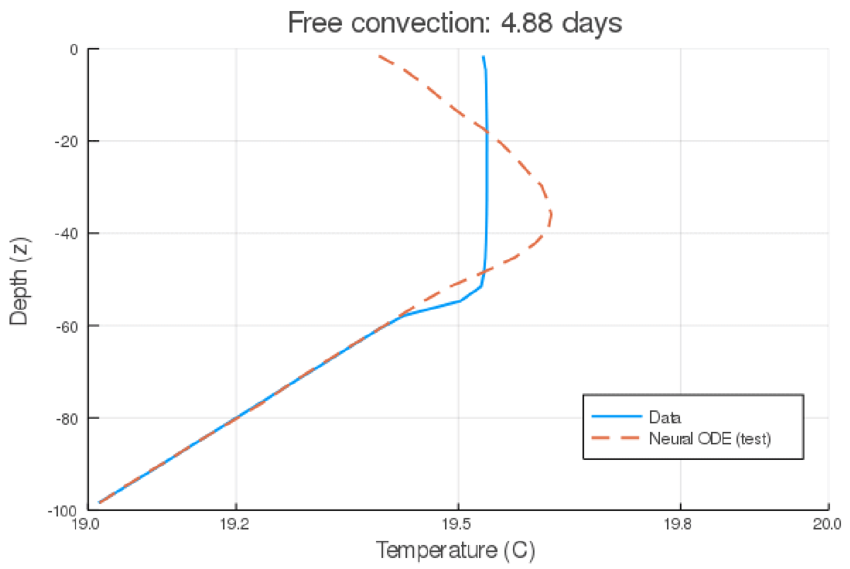


Figure 4: In blue is the temperature profile data from a high-resolution simulation such as the one shown in figure 2. The dashed orange line shows the neural differential equation's prediction for the temperatue profile given an initial condition resembling that of the high-resolution simulation.

using it in a neural differential equation to provide a good first guess for the weights of $\mathbb{NN}$, otherwise time-stepping the neural differential equation produced numerical instabilities. Afterwards, training on 32 iterations worth of data pairs it was stable for over 800 iterations.

Such a neural differential equation (see figure 5 did not perform terribly well, presumably as it need to be trained on much more data (we ran out of time). We hope we can get away with smaller networks if we tell the neural network all the physics we know.

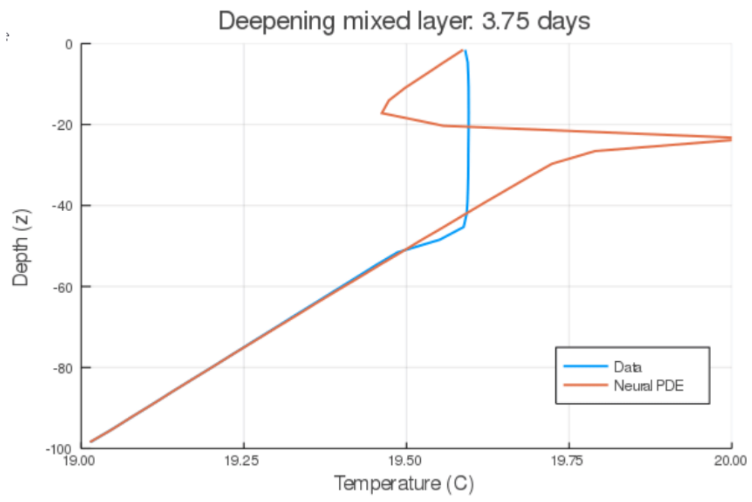
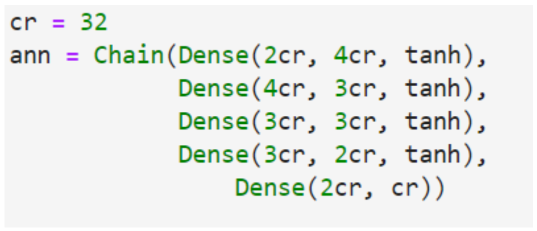


Figure 5: In blue is the temperature profile data from a high-resolution simulation such as the one shown in figure 2. The orange line shows the neural differential equation's prediction for the temperatue profile given an initial condition resembling that of the high-resolution simulation.



```
cr = 32
ann = Chain(Dense(2cr, 4cr, tanh),
            Dense(4cr, 3cr, tanh),
            Dense(3cr, 3cr, tanh),
            Dense(3cr, 2cr, tanh),
               Dense(2cr, cr))
```

Figure 6: Implementation of neural network architecture for Mixed Neural ODE in Equation (15)

7.3 Mixed ODE - Janak

A mixed neural ODE consist of the neural ODE's which contain both the machine learning as well as standard ODE terms. This is a very broad class and are loosely defined but in general, are more complex than the equations we used in section 7.3.

For this experiment, we try to model the following equation also mentioned in section 5:

$$\partial_t \overline{T} = \partial_z \left[\mathbb{NN}(T, ...)\right] + \frac{Q}{\rho c_p}$$

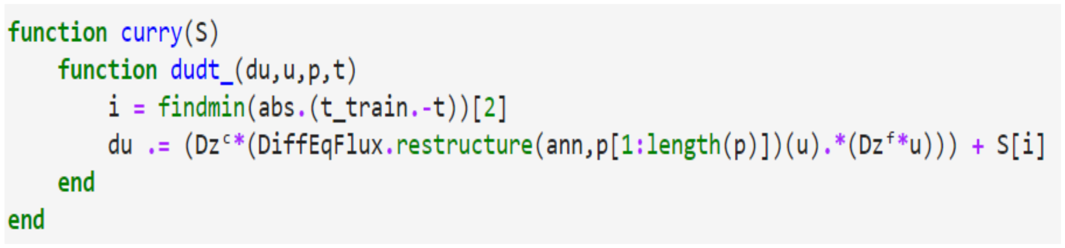
We discretize this equation around z to get the following augmented form of the equation:

$$\partial_t \overline{T} = \left[\mathbb{NN}(T, ...)\right] * \partial_z \overline{T} + \frac{Q}{\rho c_p} \tag{15}$$

We implement the neural network using 5 layers of densely connected neural network layers with tanh activation function. The number of input as well as outputs to the neural networks is 32 which is equal to the number of grid cells in our data. The architecture of the neural network is also shown in Figure 6.

For the implementation of the neural ODE we use the DifferentialEquations.jl package in Julia. We pass We can also see the implementation of the neural ODE in Figure 7. As we see in Figure 7, the gradient of temperature with respect to the z-direction is obtained using Tridiagonal matrices Dz^c and Dz^f. We pass the heat flux to the neural ODE as a parameter to the closure encapsulating the actual differential equation.

We train the aforementioned model for 300+ iterations where the mean-squared error(MSE) loss goes from 10^6 to 10. The results of the model after training are shown in Figure 8. As we can see, 300



```
function curry(S)
    function dudt_(du,u,p,t)
        i = findmin(abs.(t_train.-t))[2]
        du .= (Dz^c*(DiffEqFlux.restructure(ann,p[1:length(p)])(u).*(Dz^f*u))) + S[i]
    end
end
```

Figure 7: Implementation of neural ODE architecture for Mixed Neural ODE in Equation (15)

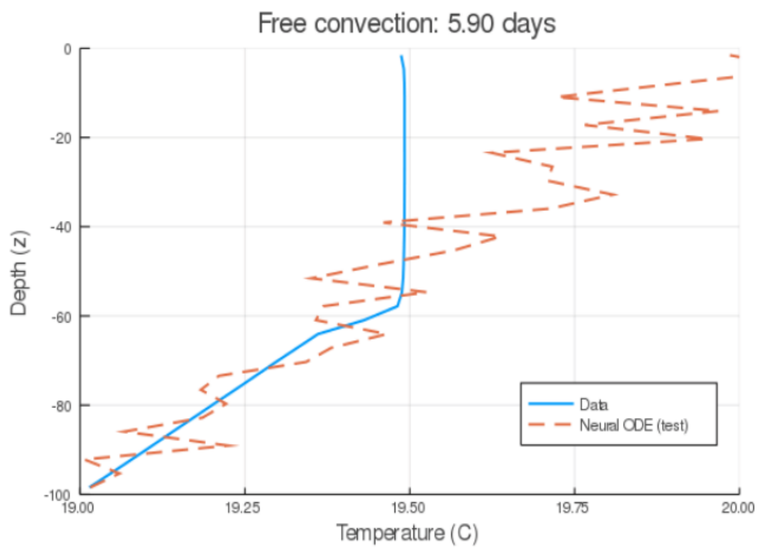


Figure 8: Comparison of Mixed Neural ODE model to actual data after training. The blue line represents actual data whereas the orange line represents the prediction from mixed Neural ODE model.

iterations on a 16GB CPU was enough to make the model stable but not nearly enough to give a good approximation of the changes in temperature.

We also successfully implemented and trained the model on a GPU but since the neural ODE as shown in Figure 7 contains a significant number of matrix as well as scalar(such as division and addition of constant numbers) operations, we weren't able to achieve any training boost over the CPU, thus making it harder to train mixed Neural ODEs.

Based on this, we estimate that mixed neural ODEs can although help incorporate more information about the dynamics of the system into the model, the resources required to train a mixed neural ODE are much higher than simple neural ODE. But in this experiment we were able to show that the amount of time and resources required to stabilize the model are much less than that needed for a traditional machine learning parameterization approach.

8 Conclusion and future work

It took us a while to get up and running so we did not end up reaching any concrete conclusions. However, a few things we learned include

- There are good reasons to try and apply machine learning to climate model parameterizations.
- Neural differential equations provide a flexible framework for scientific machine learning.

- The scalability of Neural differential equations is not clear compared to traditional deep networks.

- Neural differential equations make it easier to work with irregularly spaced time sampled data (e.g. latent differential equations).

We intend to continue working on this project beyond the 6.S898 seminar. Near term future goals include

- Scaling up our operations beyond proof of concepts. This will involve training and testing in many different physical scenarios. For example, surface heating during the day, surface cooling during the night, diurnal and seasonal cycles, and calm, windy, and gusty surface wind conditions.

- We need to train using much more data. The data has been generated and is available but time did not permit us to scale up just yet.

- Identify which parameterizations benefit from neural differential equations vs. a deep network or random forests.

- The neural differential equation parameterization seems stable but we will need to inject it into a climate model and test it *in vivo*.

- Compare performance and accuracy with existing parameterizations.

References

Alexander, J. A. and M. C. Mozer (1995). **?**Template-based algorithms for connectionist rule extraction**?** In: *Advances in neural information processing systems*, pp. 609–616.

Bolton, T. and L. Zanna (2019). **?**Applications of Deep Learning to Ocean Data Inference and Subgrid Parameterization**?** In: *Journal of Advances in Modeling Earth Systems* 11.1, pp. 376–399. DOI: `10/gfsvqg`.

Bower, J. M. and D. Beeman (1995). **?**The Book of Genesis: Exploring Realistic Neural Models with the General Neural Simulation System**?** In:

Brenowitz, N. D. and C. S. Bretherton (2018). **?**Prognostic Validation of a Neural Network Unified Physics Parameterization**?** In: *Geophysical Research Letters* 45.12, pp. 6289–6298. DOI: `10/gdqbbr`.

?Diffusion Convolution Recurrent Neural Networks**?** (2018). In: *ICLR* 11.1. DOI: `10/gfsvqg`.

Gentine, P., M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis (2018). **?**Could Machine Learning Break the Convection Parameterization Deadlock?**?** In: *Geophysical Research Letters* 45.11, pp. 5742–5751. DOI: `10.1029/2018GL078202`.

Hasselmo, M. E., E. Schnell, and E. Barkai (1995). **?**Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3**?** In: *Journal of Neuroscience* 15.7, pp. 5249–5262.

Rasp, S., M. S. Pritchard, and P. Gentine (2018). **?**Deep learning to represent subgrid processes in climate models**?** In: *Proceedings of the National Academy of Sciences* 115.39, pp. 9684–9689. DOI: `10.1073/pnas.1810286115`.

Reichstein, M., G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat (2019). **?**Deep learning and process understanding for data-driven Earth system science**?** In: *Nature* 566.7743, pp. 195–204. DOI: `10.1038/s41586-019-0912-1`.

Rolnick, D. et al. (2019). **?**Tackling Climate Change with Machine Learning**?** In: *arXiv:1906.05433 [cs, stat]*. URL: `http://arxiv.org/abs/1906.05433` (visited on 09/15/2019).