Rule Discovery using Neural ODEs - ASL Classification

Subhaditya Mukherjee (17BCE2193)
Supratim Sarkar (17BCE2203)

1. Guide

Professor Vijaya Rajan V will be guiding us through our project.

2. Research Proposal

In any field, data modeling is an extremely important factor in understanding a system. In classical cases, Ordinary differential equations were used to perform this discovery. A recent paper[1] merged the concepts of ODEs with Deep learning and created a neural ODE in which a ResNet architecture's res-blocks were substituted with an ODE solver which led to not only better results, but also more memory efficiency. This project aims to attempt to come up with a better architecture that gives SOTA results on the ASL classification dataset.

3. Workflow

The first part of the project will involve going through multiple papers and doing a comprehensive survey of the work that is already there. The objective is to identify which architectures are being used and possibly what memory requirements they come with in comparison to their non ODE counterparts. Standard metrics will be used to compare the architectures and possibly other metrics to identify how well the architectures generalize.

The second part of the project will involve testing multiple architectures on classification tasks and coming up with one that does better on a variety of these tasks and is more suited to being used with an ODE solver. We will start with a standard ResNet architecture and then branch out to similar ones. Since this is a relatively new paper, there are many architectures that have not been worked on and those will be targeted first. The insights from these will be used to come up with a better architecture and find out what influences it.

4. System

Since this is a Deep learning experiment, a system with a GPU will be used.

OS - Linux (Arch)

Language - Python

5. Libraries that will be used

1. numpy

2. matplotlib
3. pytorch
4. pytorch lightning
5. torchdiffeq

6. Literature Survey

Neural Networks are important for the approximation of bounded continuous functions in the field of Machine Learning. Such networks provide various frameworks and procedures for solving ODEâ and PDEâs in a numerical manner. The paper[9] discusses about the use of a solver for ODEâs and PDEâs using Neural Network frameworks. Such Neural Networks use function approximation for performing the tasks. The solver has proven to show high accuracy for both initial value problems and boundary value problems.it works for the functions as well as the derivatives. The solver is tested using collocation points using the Burgers equation and the heat equations. The ODE solver works well for the ODEâs and shows great accuracy. Boundary value problems are challenging to solve using the equation mentioned and another drawback is the computational efficiency of the ODE solver. Accuracy must be improved in the case of boundary value problems. The paper [5] discusses that the stochastic gradient for a given ResNet neural network is made to converge to the stochastic gradient descent for a Neural ODE. All the layers in the ResNet neural network share the same weighted matrix. It proves that Neural ODEâs are the deep limit of such neural networks. Several equations have been tried and the result has been obtained using the Fokker-Plank equationsâ equation and theorems resulted in a large number of convergence results. Convergence rates are high and are optimal for the results. It only considered the convergence of minimizers and not the optimization procedure. Regularization cannot solely explain the numerical simulation. The use of DiffEqFlux library is described in [10] and an explanation for how it is used for solving several ODEâs and other type of differential equations is given. All the ODEâs are taken as a flux defined neural network. The purpose of using the DiffEqFlux library is to solve such ODEâs where simple integration strategies are not sufficient to get optimal results and solutions with high accuracy. The ODEâs are defined as neural networks and proper description of the Flux model zoo is provided that includes neural stochastic differential equations. The library helps in developing highly accurate ODE solvers that can play a major role in various applications of Machine Learning and Data Science. Various architectures of Deep Learning have now been considered as Neural ODEâs. It helps bring closer the gap between deep learning and systems that are dynamic in nature. The major challenges faced by [4] are figuring out the working of the models and the dynamics associated with them. The design choices for building the modules have to be elucidiated.The framework is set up for building a general Neural ODE and the components associated with it are considered. Infinite dimensional problem has been solved using numerical approximations that lead to better models for solving ODEâs. There are a variety of linear Ordinary differential equations that exist today for which various methods exist to solve them. The paper[3] discusses about the multiquadric radial basis function networks and its use in solving linear ODEâs in an efficient manner. By approximating functions and its derivatives using radial basis function networks, another RBFN is formed which can be efficient in solving Linear ODEâs with better accuracy. It is the most optimal method to determine the existing parameters. It fixes the problem of not having enough data samples as it deals with only the domain and the existing boundaries associated with it.The method can be used to solve even PDEâs.A sample set is selected from the training set and positions of centers of basis functions are selected-means clustering algorithm is used on the training set and the centers of the clusters are used as the centers of the basis functions. It is useful for large-scale problems that exist in several engineering fields.The major challenge faced by this approach is that the number of basis fucntions and data set must be provided with a priori. Function approximation has always been an unstable task when it comes to using differential equations. The challenge of achieving a stable flow and also having a low computational cost has been tackled by the authors[8]. The techniques proposed in this paper use an energy function to guarantee the asymptomatic stability of the solution. Every deep learning algorithm is function composition between an input and an output space and is basically a mapping between them. This process was known but a proper optimization process was not identified so far. This paper proposes an algorithm to reduce the instability and stiffness that a solver might face by using a parametrized energy function. This takes care of of the floor of the network States and moves it along the negative gradient. Instead of trying to learn a single function, the algorithm tries to learn functions from the function space. There are

multiple types of models that are possible and observed. Autonomous port Hamiltonian model, second order and stochastic models are also defined. In training the model smooth scaler cost function was used to to understand how well the model performs and also to minimize the loss in an optimal setting. The computing of gradients is is efficient with respect to to the choice of the parameters. To ensure that the energy function remains steady, a soft constraint was also added. Neural network architectures are composed of states as well as hidden layers. Using this concept, it is possible to understand the mapping between the vector spaces of the input and outputs. We can further take this idea and replace the entire component of the hidden states by using a differential equation solver. And framing the network problem as an ODE problem. This is the premise of a paper by Chen et al.[6] The major advantage of using models success this is the near constant memory costs as well as a higher customization by means of allowance for trade off between speed as well as precision values. The second advantage is that of huge boost in efficiency due to the process of back propagation not being required as a differential equation solver is being used instead. This also leads to pretty high guarantees about stagnating errors as well as the ability to be trained using the maximum likelihood model. It is also possible to to train on data that is not static and arrives non concurrently due to its nature. The method used to solve the equations makes use of the calculation of a secondary equation backwards which allows for efficiency in memory as well as the ability to define the error threshold. These benefits do not generally come with standard deep learning models. Making use of this black box idea, the authors replace the standard ResNet models with an architecture that uses the solver instead of hidden layers. The only issue is that the analogue to the depth of a network cannot be obtained in this scenario. The model RK-Net performs similar to the original ResNet on the MNIST dataset and many others as described in the paper. Using a solver also allows the model to have multiple hidden units without affecting the time complexity of the training. Being able to discover new equations using data has always been a dream when it comes to the field of computer science. A recent paper [1] shows the possibility of harnessing deep learning for the same by augmenting the capabilities of scientific modeling. This leads to the creation of an UDE which would potentially enable the user to extrapolate factors that do not exist in the data as well as be able to simulate better models more accurately. Using the principles of physics, it is also possible to incorporate the existing knowledge into the network. The combination of differential equations in an ML concept to create a PINN using the UAT has also been shown. The paper [2] demonstrates the ability to obtain the equations that define a system using just data and also the possibility of recovering an equation from the time series data to extrapolate it. It also talks about the possibility of using the physics laws of conservation to to identify new systems by means of transferring previous knowledge. The approach of using physics and form networks allows the creation of solvers that can efficiently create gradient related calculations that are required to to perform any kind of machine learning task. Even though the existence of residual networks have been present for a long time the the technique for or constructing such models for a continuous and invertible function was not present. This paper[7] shows how to prove any homeomorphism that can be approximated by using a combination of neural networks and ODEs in Euclidean space. This leads to the ability of using residual connections and blocks to create a reverse mapping between the output space as well as input space and allow for applications such as deep learning oriented ones.

7. Components that will be used

7.1. Dataset

The dataset that will be used is the ASL dataset from https://www.kaggle.com/grassknoted/asl-alphabet. This has about 29 classes of which, 10 classes will initally be used to test the system.

7.2. Reading the data

This will be done using pandas and a dataframe will be created from the dataset to allow for easier processing. Stratify will also be used to ensure that the classes remain the same count while sending it to the training loop.

7.3. Training details

These have not been finalized but an inital system involves the following.

1. Save hyperparameters
2. Learning rate of 1e-14
3. Weight decay of 0.0001
4. Adam Scheduler with default parameters and weight decay
5. Step Learning rate scheduling.
6. Loss function of cross entropy.
7. Batch size of 128
8. Image size of 64x64x3
9. Float 16 precision
10. Distribution plugins with ddp and sharded plugins for single GPU parallelism
11. Data augmentations as follows

    RandomResizedCrop(img_size, img_size, p=1.0),

    Transpose(p=0.5),

    HorizontalFlip(p=0.5),

    VerticalFlip(p=0.5),

    ShiftScaleRotate(p=0.5),

    HueSaturationValue(hue_shift_limit=0.2,sat_shift_limit=0.2,val_shift_limit=0.2,p=0.5),

    RandomBrightnessContrast(brightness_limit=(-0.1, 0.1),contrast_limit=(-0.1, 0.1),p=0.5),

    Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225],max_pixel_value=255.0,p=1.0),

    CoarseDropout(p=0.5),

    Cutout(p=0.5),

    ToTensorV2(p=1.0),

8. References

[1] Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., ... & Ramadhan, A. (2020). Universal differential equations for scientific machine learning. arXiv preprint arXiv:2001.04385.

[2] Filici, C. (2008). On a neural approximator to ODEs. IEEE transactions on neural networks, 19(3), 539-543.

[3] Jianyu, L., Siwei, L., Yingjian, Q., & Yaping, H. (2003). Numerical solution of elliptic partial differential equation using radial basis function neural networks. Neural Networks, 16(5-6), 729-734.

[4] Massaroli, S., Poli, M., Park, J., Yamashita, A., & Asama, H. (2020). Dissecting neural odes. arXiv preprint arXiv:2002.08071.

[5] Avelin, B., & NystrÃ¶m, K. (2019). Neural ODEs as the deep limit of ResNets with constant weights. arXiv preprint arXiv:1906.12183.

[6] Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary differential equations. arXiv preprint arXiv:1806.07366.

[7] Zhang, H., Gao, X., Unterman, J., & Arodz, T. (2020, November). Approximation capabilities of neural ODEs and invertible residual networks. In International Conference on Machine Learning (pp. 11086-11095). PMLR.

[8] Massaroli, S., Poli, M., Bin, M., Park, J., Yamashita, A., & Asama, H. (2020). Stable neural flows. arXiv preprint arXiv:2003.08063.

[9] Liu, Z., Yang, Y., & Cai, Q. (2019). Neural network as a function approximator and its application in solving differential equations. Applied Mathematics and Mechanics, 40(2), 237-248.

[10] Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., & Dixit, V. (2019). Diffeqflux. jl-A julia library for neural differential equations. arXiv preprint arXiv:1902.02376.