Neural Ordinary Differential Equations

Jimmy Chen and Laurie Luo
Math 179, Harvey Mudd College
jimchen@hmc.edu, hluo@hmc.edu

Abstract

In this paper, we introduce the concepts and implementation details of a newer class of machine learning algorithms, Neural Ordinary Differential Equation (NODE). We replicate and discuss a specific implementation that compares ResNet and NODE, which uses the electrocardiogram (ECG) dataset from the MIT Beth Israel Hospital (BIH). We successfully replicated the result of the original authors of the implementation, which demonstrates the advantages and disadvantages of NODE.

1 Neural Network and ResNet

Neural Networks (NN) are a family of algorithms that models over datasets and provides predictions. They play a central role in modern machine learning process, particularly deep learning.

For example, in a classical data modelling problem, we have N pairs of data point $X = (\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_N, \mathbf{x}_N))$, where $\mathbf{x}_i \in X$ is the input domain and $\mathbf{y}_i \in Y$ is the output domain, and some sort of relationship exists between the two variables:

$$X \longrightarrow ? \longrightarrow Y$$

We would like to utilize the existing dataset and make prediction of the output $\mathbf{y}^*$ giving a new data point $\mathbf{x}^*$. The highly flexible NN can be iteratively trained on the dataset, ultimately describing the dataset with accuracy. Using an NN turns the classical situation into the situation below:

$$\begin{tikzcd} X \arrow[rr] \arrow[dr] & & ? \arrow[rr] & & Y \\ & NeuralNetwork \arrow[urrr] & & & \end{tikzcd}$$

Internally, a generic NN consists of several layers of neurons, mimicking the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Figure 1 is a detailed illustration of common, fully connected neural network. The input data $\mathbf{x}_i$ is first fed into the first layer, where it is transformed by the layer to some intermediate values, which are further transformed by the next layer, and so on. In this way, the neural network can be viewed as a giant composite function and, thus, the more layers there are, the deeper the network it is. For a fully connect layer, the input of each neuron is a weighted summation over the previous values with a constant term as bias, and then the result is passed through a scalar activation function, usually nonlinear.

Midterm Project for Math 179 in Harvey Mudd College.

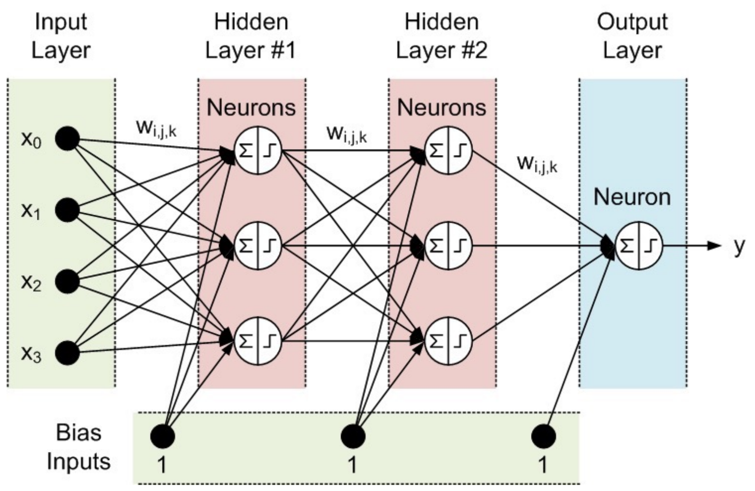


Figure 1: Detailed Illustration of Neural Network

Hence, the number of layers deeply affects the performance of the neural network. Too few layer, meaning too few parameters, will cause under-fitting, wile too many layers will cause over-fitting (shown in figure 2).

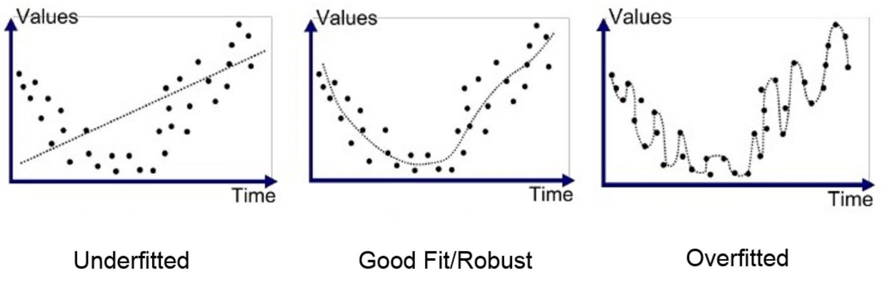
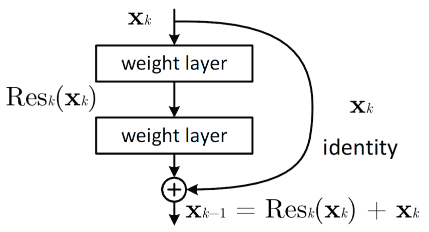


Figure 2: Underfitting and Overfitting

One class of NNs effectively allows the number of layers to be adaptively optimized for the dataset: they are the *Residual Neural Networks (ResNet)*. Whereas regular multi-layer perceptions and many other NNs chain layers together linearly through composition only, ResNets introduce shortcuts that skip over groups of layers by adding intermediate results from a few layers prior to the current output. One can consider a group of layers that has such a shortcut over it as a single residual block (or group). For the kth residual block, the output is the addition $\text{Res}_k(\mathbf{x}_k) + \mathbf{x}_k$, between the processed output of the internal layers and the original input.

$$\mathbf{x}_{k+1} = \text{ResBlock}_k(\mathbf{x}_k) = \text{Res}_k(\mathbf{x}_k) + \mathbf{x}_k, \tag{1.1}$$


$\mathbf{x}_k$
weight layer
$\text{Res}_k(\mathbf{x}_k)$
weight layer
$\mathbf{x}_k$
identity
$\mathbf{x}_{k+1} = \text{Res}_k(\mathbf{x}_k) + \mathbf{x}_k$
Figure 3: ResNet Block

The transition to Neural ODE begins by noting that equation 1.1 bears some resemblance to an unrelated concept. If we treat the internal process $\mathrm{Res}_k(\mathbf{x}_k)$ as giving the derivative of a unified "variable" $\mathbf{x}$ at "time" k, then the equation effectively represents the Euler method for solving the ODE (with unit time step)

$$\left.\frac{d\mathbf{x}}{dt}\right|_{t=k} = \mathrm{Res}(\mathbf{x}_k, k) \tag{1.2}$$

where $\mathrm{Res}(\mathbf{x}_k, k) = \mathrm{Res}_k(\mathbf{x}_k)$ represents Res_k at all ks.

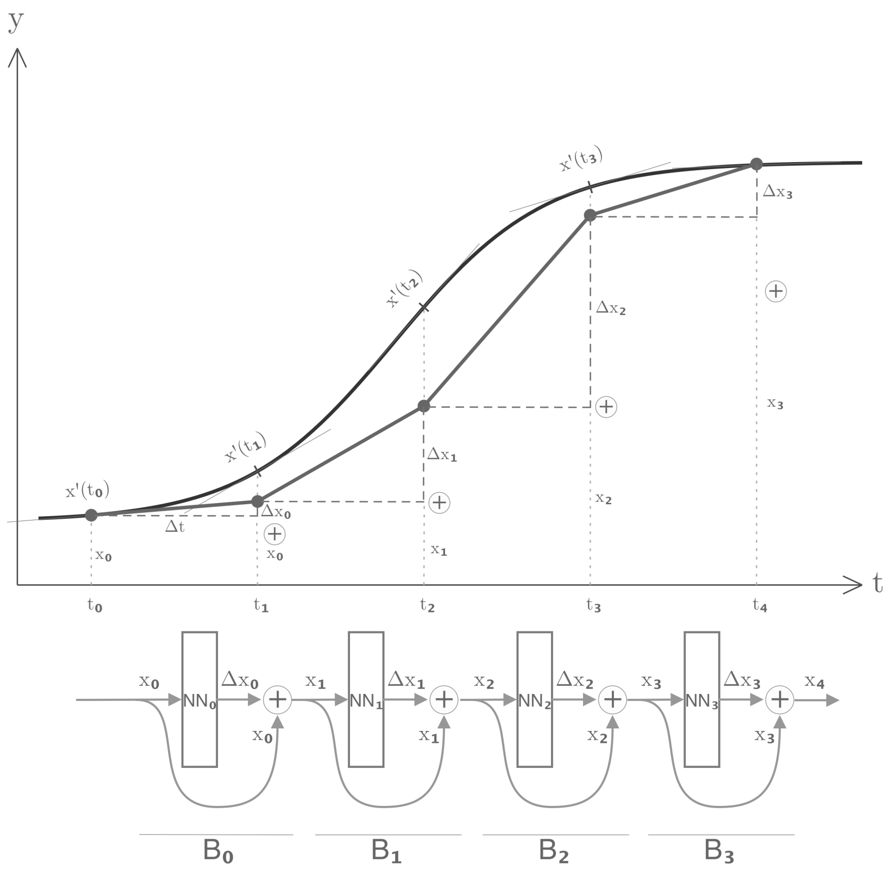


Figure 4

Figure 4 further illustrates the correlation between ResNet and the Euler method. Four ResNet blocks $\mathrm{B}_{0\ldots3}$ are shown, with internal networks $\mathrm{NN}_{0\ldots3}$ and shortcuts as arrow from each input to output. Now we can reinterpret each the network output as the derivative to some function of t, shown as the dark curved line. Take the first step for example. For the first block B_0, its internal network NN_0 takes in the initial point of $\mathbf{x}$, $\mathbf{x}_0$ at t_0, and produces some $\mathbf{x}'(t_0)$. To approximate the value of the function at the next time point t_1, we multiply it by the time step Δt which is assumed to be unit, so $\mathbf{x}'(t_0) = \Delta\mathbf{x}_0$, the change in value in this time step. The final estimate is the sum of the original $\mathbf{x}_0$ and the change $\Delta\mathbf{x}_0$. In the ResNet block, this is the sum of the network output and the original input through the shortcut. The step is then repeated three more times by different ResNet blocks, each corresponding to a specific time point.

This does assume that the intermediate results x_k to share the same dimensions, but the assumption is less important than our purpose of illustration. The point is this: if ResNet approximates an ODE solution by the Euler method, then we can naturally extended ResNet by considering other, and better, integration methods.

2 Neural ODE

Neural ODEs (NODE) are a continuous re-imagination of ResNet. They model curves in arbitrary dimensions as solutions to first order ODEs, and are thus inherently applicable to time series. When ResNet is thought as the Euler method, each residual block corresponds to one discrete time step in the Euler method: the kth group takes the previous result $\mathbf{x}_k$ and gives the derivative of $\mathbf{x}$ at time t_k. Note that each residual group is only responsible for a single time point. Neural ODE instead uses a single NN to produce all derivatives by taking the time point as explicit input:

$$\frac{d\mathbf{x}}{dt} = \mathrm{NN}_{\mathrm{NODE}}(\mathbf{x}, t). \tag{2.1}$$

It follows that the solution to this ODE is the integral over time:

$$\mathbf{x}(t) = \int \mathrm{NN}_{\mathrm{NODE}}(\mathbf{x}, t)dt \tag{2.2}$$

From a computational standpoint, this integral can be approximated with any numerical integration

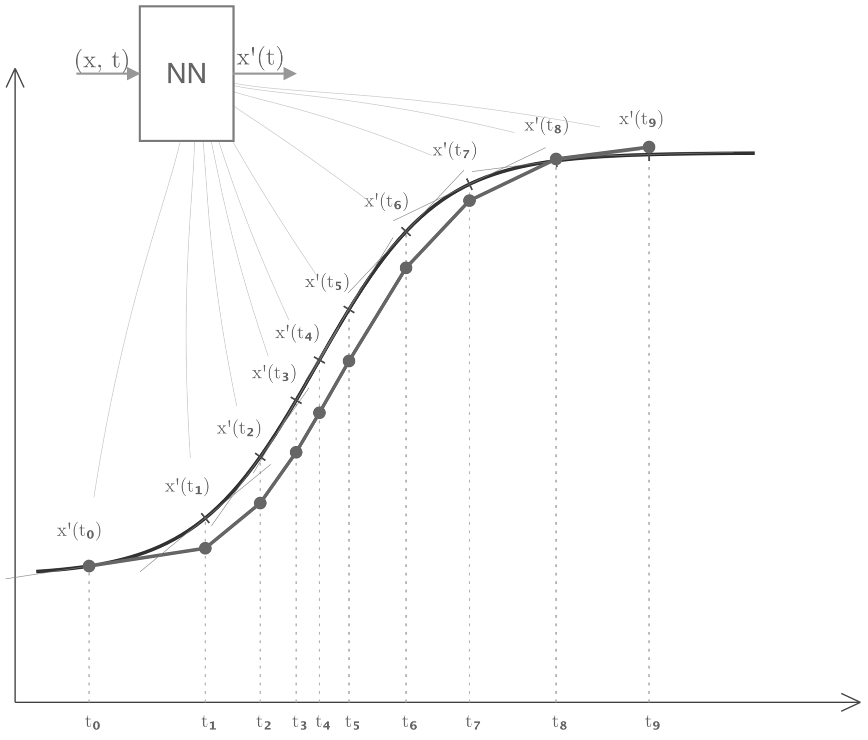


Figure 5

Figure 5 illustrates the operation of a Neural ODE. A single neural network produces all derivatives of the ODE it models: at each time point t, it takes in t and the current value $\mathbf{x}$, and the output of the NN is used as the derivative $\mathbf{x}'(t)$. The network parameters remain constant regardless of time, while the input t varies. This contrasts with ResNet as the Euler method, which uses entirely different networks for different time points.

As Neural ODEs are flexible in integration methods, integration points, the time points at which the derivative is evaluated, can be freely chosen in principle, both in amount and in value. In practice, this can reflect in adaptive strategies in choosing integration points such as reducing the step size when derivatives are large in magnitude.

Such generality has immediate advantages. Numerical methods for solving ODEs has been a rich and much-explored field, with optimized methods for accuracy, speed, and specific ODE types, most significantly superior to the Euler method. For instance, a number of fourth-order Runge–Kutta methods exist where the accumulated error that scales with the fourth power of step size, whereas the error the Euler method scales linearly with step size.

Neural ODE also opens up the flexibility of choosing algorithms and of balancing between accuracy and speed. As a ResNet block corresponds to a single time step of the Euler method, the number of steps taken is tied to the ResNet's structure, namely the depth of the network. For Neural ODEs the number of steps is variable and controllable, which effectively establishes flexible depth. Larger network depth makes it more expressible and accuracy while a smaller depth cuts the time and resource cost of iterative computation. Additionally, for ResNet the memory cost of storing layers of parameters scales with depth, while Neural ODE has a constant memory profile. Finally, Neural ODES can be evaluated at any point along the solution curve, which is ideal for modeling data with irregular time point.

3 Implementation Example

In this section we replicate the result an implementation given in `https://github.com/abaietto/neural_ode_classification`. The author of this implementation compares the accuracy, time cost and memory cost of similarly structured ResNet and NODE models in the context of a supervised learning task, namely the classification of an ECG signal.

3.1 Electrocardiogram Dataset

This implementation uses MIT Beth Israel Hospital (BIH) electrocardiogram (ECG) dataset. The data is obtained from Kaggle [1], containing about 110,000 labeled data points about heartbeat classification. Each sample is annotated into the following five categories: normal (0), supraventricular premature beat (1), premature ventricular contraction (2), fusion of ventricular and normal beat (3), and finally unclassifiable beat (4). The ECGs were recorded at a frequency of 360 Hz. Thus, each sample was taken over 0.52 seconds since there are 187 measurements per sample. To better understand each heartbeat type, we obtained examples for each category, as shown below.

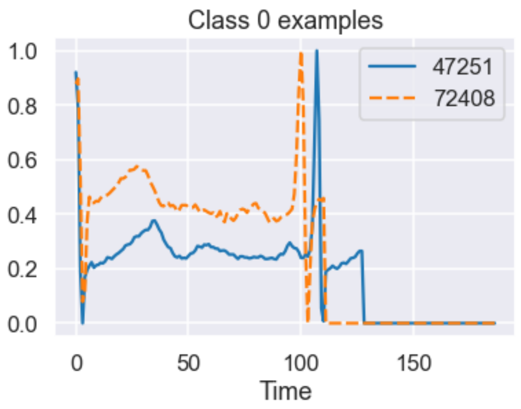


Figure 6: Heartbeat Example of Category 0

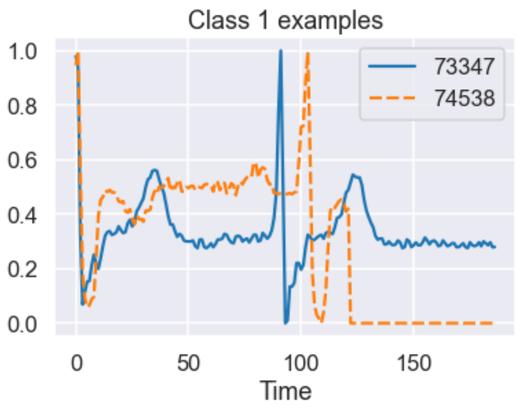


Figure 7: Heartbeat Example of Category 1

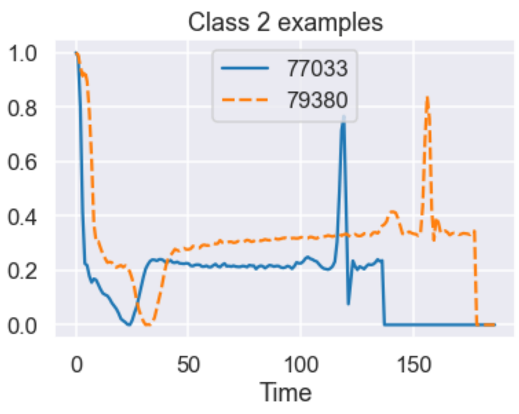


Figure 8: Heartbeat Example of Category 2

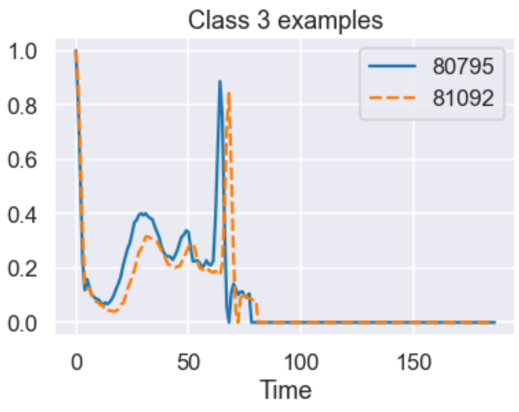


Figure 9: Heartbeat Example of Category 3

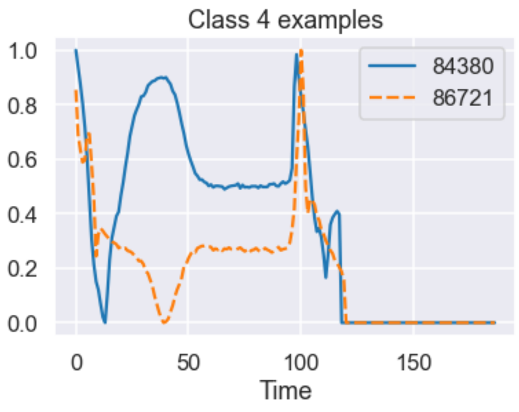


Figure 10: Heartbeat Example of Category 4

From the data, the proportion of each category is shown in Table 1, which shows normal heartbeats are the most common category.

Category	Proportion
0	0.83
1	0.03
2	0.07
3	0.01
4	0.07

Table 1: Proportion of each category of heartbeat

3.2 Network Architecture

For the purpose of comparison, the ResNet model and the NODE model here are identical in many layers. Each cardiogram sample consists of 187 values. For both models, the first three layers are one dimensional convolutions, who together results in 64 channels of time-local series, each now with 46 values. For the ResNet, this is followed by six ResNet blocks identical in structure, each with two convolutions layers plus normalization. For the NODE, the six blocks of the ResNet are replaced by one Neural ODE block, which has an internal NN consisting of two convolutions also with normalization. From this point, both models go through a fully connected layer which outputs to a size of 5 representing the categorization task.

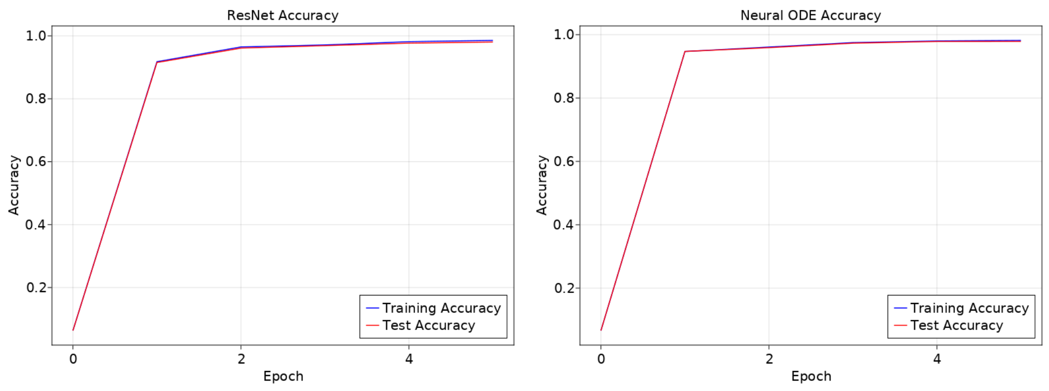


Figure 11: Comparison of accuracy between the ResNet and the Neural ODE

3.3 Characteristics and Comparison Between ResNet and NODE

We are able to fully replicate the results of the original authors. Training each network for 5 epochs, the ResNet produced a final accuracy of 0.985 on the training dataset and 0.980 on the test set, while the Neural ODE produced 0.982 and 0.979 for training and test respectively. Figure 11 shows historical accuracies by epochs. We see that the final accuracies are similar between the two models.

An advantage of Neural ODEs is demonstrated when we compare the size of the models. Namely, while the ResNet contains 182853 parameters, the NODE has only 59333, thus producing the same level of accuracy as the ResNet with less than one third of the memory cost. The difference in model size is directly reflected in the network structure: the ResNet notably uses 6 residual blocks each containing 24832 parameters while the NODE uses one Neural ODE block with 25472 parameters. This verifies the aforementioned theoretical advantage of NODE networks in memory.

The implementation also reflects a potential disadvantage of Neural ODES, namely training and evaluation time. For evaluation, the ResNet takes around 9 seconds on our hardware with the test set as input while the NODE takes around 70 seconds; for training with the selected optimizer (stochastic gradient descent with momentum optimization), the ResNet takes around 1.6 to 1.8 minutes per epoch while the NODE spends around 11.7 to 16.6 minutes per. The multiplied time cost of Neural ODEs has been implicitly noted by several others, namely in [4], [5] and [6] all motivated by efficiency. An immediate area for future work are to incorporate these efficiency optimizations to this specific dataset.

References

[1] https://www.kaggle.com/datasets/shayanfazeli/heartbeat

[2] https://github.com/abaietto/neural_ode_classification

[3] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural Ordinary Differential Equations. arXiv. https://doi.org/10.48550/ARXIV.1806.07366

[4] Kidger, Patrick, Ricky T. Q. Chen, Terry J. Lyons. ''Hey, that's not an ODE': Faster ODE Adjoints with 12 Lines of Code'. CoRR abs/2009.09457 (2020): n. pag. Web.

[5] Bilos, Marin .. 'Neural Flows: Efficient Alternative to Neural ODEs'. CoRR abs/2110.13040 (2021): n. pag. Web.

[6] Kelly, Jacob .. 'Learning Differential Equations that are Easy to Solve'. CoRR abs/2007.04504 (2020): n. pag. Web.