CS2305: Presentation-2

Akash Tadwai

Krishna Srikar Durbha

Veluvarthi Narasimha Reddy

16th March 2020

Ordinary Differential Equations

An ordinary differential equation (ODE) is an equation that involves some ordinary derivatives (as opposed to partial derivatives) of a function. Often, our goal is to solve an ODE, i.e., determine what function or functions satisfy the equation.

Solving an ODE is more complicated than simple integration. Even so, the basic principle is always integration, as we need to go from derivative to function. Usually, the difficult part is determining what integration we need to do.

Finite Difference Method

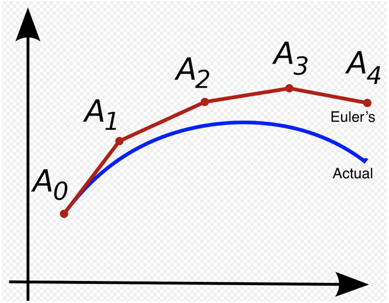
In numerical analysis, finite-difference methods (FDM) are discretizations used for solving differential equations by approximating them with difference equations that finite differences approximate the derivatives.

FDMs convert a linear ordinary differential equations (ODE) or non-linear partial differential equations (PDE) into a system of equations that can be solved by matrix algebra techniques.

Example : `https://mathforcollege.com/nm/mws/gen/08ode/mws_gen_ode_spe_finitedif.pdf`

Euler's Method I

Often times, differential equations are large, relate multiple derivatives, and are practically impossible to solve analytically.The value of the function y(t) at time t is needed, but we don't necessarily need the function expression itself. Even more convenient is the fact that we are given a starting value of y(x) in an initial value problem(problem consisting of a differential equation and an initial value), meaning we can calculate y'(x) at the start value with our DE.



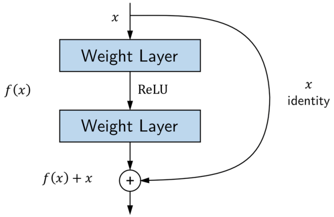
Euler's Method II

The process of evaluation of y(t) is recursive. The following equations describe it.

$$
\begin{aligned}
y(0) &= y(0) \\
y(1) &= y(0) + s \cdot y'(0) \\
y(2) &= y(1) + s \cdot y'(1) \\
&\vdots \\
y(t) &= y(t-1) + s \cdot y'(t-1)
\end{aligned}
$$

Residual Networks I

Residual Networks are progenitor of Neural ODEs. In a vanilla neural network, the transformation of the hidden state through a network is $h(t+1) = f(h(t), \theta(t))$, where f represents the network, h(t) is the hidden state at layer t (a vector), and $\theta(t)$ are the weights at layer t (a matrix). The hidden state transformation within a residual network is similar and can be formalized as $h(t+1) = h(t) + f(h(t), \theta(t))$. The difference is we add the input to the layer to the output of the layer to prevent important information to be discarded.



Residual Networks II

Residual Networks achieve higher accuracy because their architecture. Unlike Vanilla Neural Networks, Residual Networks can be stacked more deeper as there is not problem of Vanishing and Exploding Gradients through Backpropagation. Thus ResNets can learn their optimal depth, starting the training process with a few layers and adding more as weights converge, mitigating gradient problems. Thus the concept of a ResNet is more general than a vanilla NN, and the added depth and richness of information flow increase both training robustness and deployment accuracy.

However, ResNets still employ many layers of weights and biases requiring much time and data to train. On top of this, the backpropagation algorithm on such a deep network incurs a high memory cost to store intermediate values. ResNets are thus frustrating to train on moderate machines.

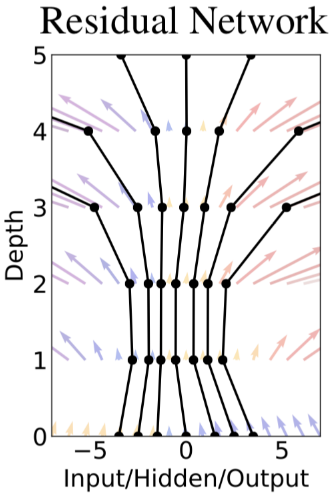
Residual Networks and ODEs I

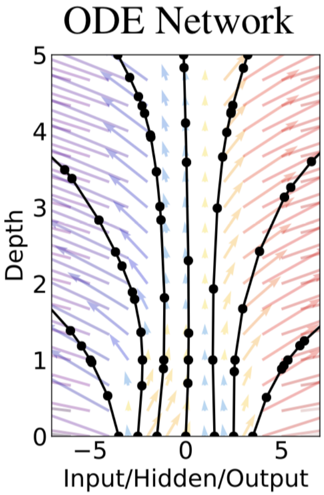
Hidden State Transformation in Residual Network.

$$
\begin{aligned}
h_1 &= i + f\left(i, \Theta_1\right) \\
h_2 &= h_1 + f\left(h_1, \Theta_2\right) \\
h_3 &= h_2 + f\left(h_2, \Theta_3\right) \\
&\vdots \\
h_{t+1} &= h_t + f\left(h_t, \Theta_t\right)
\end{aligned}
$$

Hidden State Transformation in Residual Network is similar to that of Euler's Method of solving Differential Equations with $y' = f(y,t)$ and Input to the Network is $h(0)$ with a step size of "1".Even though the underlying function to be modeled is continuous, the neural network is only defined at natural numbers t, corresponding to a layer in the network. The following Diagram explains it

Residual Networks and ODEs II





NeuralODE

Differential equations are defined over a continuous space and do not make the same discretization as a neural network, so we modify our network structure to capture this difference to create an ODENet.

ResNet

```
def f(z,t,$\theta$):
  return nnet(z,$\theta_t$)
def resnet(z,$\theta$):
  for t in range(1,T):
    z = z + f(z,t,$\theta$)
  return z
```

ODENet

```
def f(z,t,$\theta$):
  return nnet([z,t],$\theta$)
def ODENet(z,$\theta$):
  for t in range(1,T):
    z = z + f(z,t,$\theta$)
  return z
```

Differences between ODENet and Resnet and Introduction to Adpative Solvers

The primary differences between these two code blocks is that the ODENet has shared parameters θ across all layers. Without weights and biases which depend on time, the transformation in the ODENet is defined for all t, giving us a continuous expression for the derivative of the function we are approximating. Another difference is that, because of shared weights, there are fewer parameters in an ODENet than in an ordinary ResNet.

Better Numerical Solutions and be found using other ODE solvers. **Adaptive Solvers** restrict the error below predefined thresholds with intelligent trial and error. These methods modify the step size during execution to account for the size of the derivative. For example, in a t interval on the function where f(z, t, θ) is small or zero, few evaluations are needed as the trajectory of the hidden state is barely changing. But when the derivative f(z, t, θ) is of greater magnitude, it is necessary to have many evaluations within a small window of t to stay within a reasonable error threshold.

Adpative ODENet I

The number of times "d" an adaptive solver has to evaluate the derivative has an importance. If d is high, it means the ODE learned by our model is very complex and if d is low, then the hidden state is changing smoothly without much complexity. In terms of evaluation time, the greater d is the more time an ODENet takes to run, and therefore the number of evaluations is a proxy for the depth of a network. In adaptive ODE solvers, a user can set the desired accuracy themselves, directly trading off accuracy with evaluation cost, a feature lacking in most architectures. Adaptive ODE solvers packages are available in most programming languages can be used by setting the ODE Solver with a with an error tolerance to determine the appropriate method and number of evaluation points

Adpative ODENet II

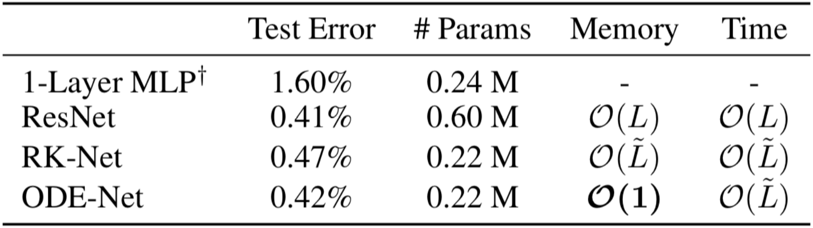
The pseudocode for AdaptiveODENet is as follows:

AdaptiveODENet

```
def f(z,t,$\theta$):
    return nnet(z,$\theta_t$)
def resnet(z,$\theta$):
    for t in range(1,T):
        z = ODESolve(f,z,0,t,$\theta$)
    return z
```

Neural ODEs for Supervised Learning I

A Residual Network Model with few downsampling layers, 6 residual blocks, and a final fully connected layer is used on MNIST Dataset. Similarly a NeuralODE Model is used whose architecture is same as Residual Network but replacing 6 residual layers with a ODE Block. NeuralODE is trained by directly backpropagating through the operations in the ODE solver. Along with these 2 Models an old classification technique from paper by Yann LeCun called 1-Layer MLP. The results are as follows:



	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\tilde{L})$

Neural ODEs for Supervised Learning II

Observations:

- ODE based methods are much more parameter efficient.
- ODENet is using the adjoint method, does away with such limiting memory costs and takes constant memory.

  Adjoint Method:

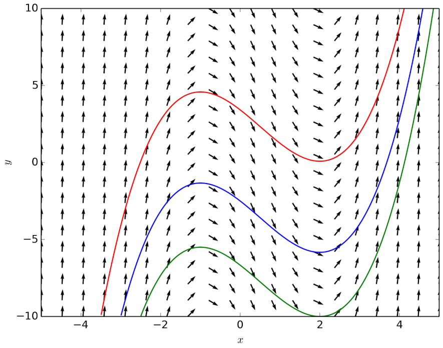
  The adjoint state method is a numerical method for efficiently computing the gradient of a function or operator in a numerical optimization problem.

Uses of NeuralODE:

- In mobile applications, there is potential to create smaller accurate networks using the Neural ODE architecture that can run on a smartphone or other space and compute restricted devices.
- In Physics where Differential Equations are involved.

Example of Neural ODE Modelling a Vector Field

Example of an NeuralODE that models trajectory of a vector field, and the corresponding smoothness in the trajectory of points, or hidden states in the case of Neural ODEs, moving through it:



Here the data is continous.

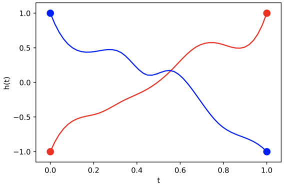
Limitations of NeuralODE I

Example-1:

If Data given or Funtion modelled is discontinuous then results are as follows:

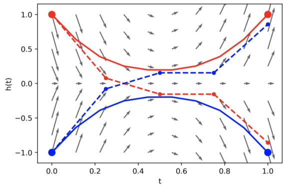
Let g_{1d} :R $\rightarrow \mathbb{R}$ be a function such that $g_{1d}(1) = 1$ and $g_{1d}(1) = 1$.

Function:



Neural ODE's Output:

Limitations of NeuralODE II



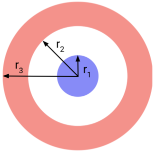
Solutions of the ODE are shown in lines and solutions using the Euler method (which corresponds to ResNets) are shown in dashed lines. As can be seen, the model easily learns the identity mapping but cannot represent $g_{1d}(x)$. Indeed, since the trajectories cannot cross, the model maps all input points to zero to minimize the mean squared error.

Any data that is not linearly separable within its own space breaks the architecture.

Limitations of NeuralODE III

Example-2:

Let $0 < r_1 < r_2 < r_3$ and let $g : \mathbb{R}^a \to \mathbb{R}$ be a function such that



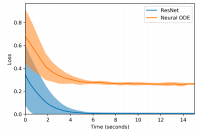
$$\begin{cases} g(\mathbf{x}) = -1 & \text{if } \|\mathbf{x}\| \leq r_1 \\ g(\mathbf{x}) = 1 & \text{if } r_2 \leq \|\mathbf{x}\| \leq r_3 \end{cases}$$

The function maps all points inside the blue sphere to 1 and all the points in the red annulus to 1.

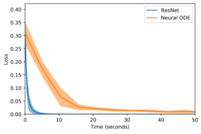
In order for the linear layer to map the blue and red points to 1 and 1 respectively, the features $\phi(x)$ for the blue and red points must be linearly separable. Since the blue region is enclosed by the red region, points in the blue region must cross over the red region to become linearly separable, requiring the trajectories to intersect, which is not possible in Neural ODE.

Augumented Neural ODE I

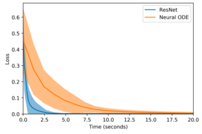
Neural ODE fails to model data which isnt linearly seperable.



(a) $g(\mathbf{x})$ in $d = 1$



(b) $g(\mathbf{x})$ in $d = 2$



(c) Separable function in $d = 2$

Compared to ResNets, NODEs struggle to fit g(x) both in d = 1 and d = 2. The difference between ResNets and NODEs is less pronounced for the separable function.

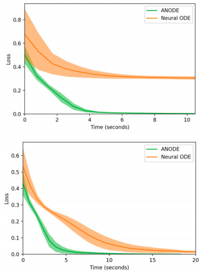
Augmented Neural ODE II

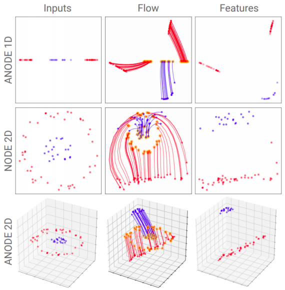
Augmented Neural ODEs (ANODEs) which provide a simple solution to the exceptions provide. Augmented Neural ODEs (ANODEs) which provide a simple solution to the problems we have discussed. We augment the space on which we learn and solve the ODE from R^d to R^{d+p} , allowing the ODE flow to lift points into the additional dimensions to avoid trajectories intersecting each other. Letting a(t) $\in \mathbb{R}^p$ denote a point in the augmented part of the space, we can formulate the Augmented ODE problem as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \mathbf{h}(t) \\ \mathbf{a}(t) \end{bmatrix} = \mathbf{f}\left(\begin{bmatrix} \mathbf{h}(t) \\ \mathbf{a}(t) \end{bmatrix}, t\right), \quad \begin{bmatrix} \mathbf{h}(0) \\ \mathbf{a}(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}$$

Results of ANODE and Visualisations I

Figure: (Left)Loss plots for NODEs and ANODEs trained on g(x) in d = 1 (top) and d = 2 (bottom). ANODEs easily approximate the functions and are consistently faster than NODEs. (Right) Flows learned by NODEs and ANODEs. ANODEs learn simple nearly linear flows while NODEs learn complex flows that are difficult for the ODE solver to compute.





Results of ANODE and Visualisations II

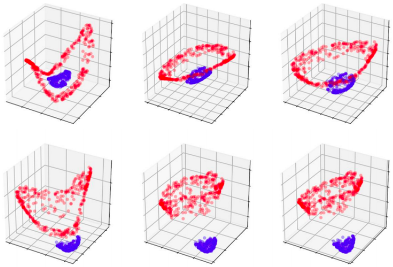


Figure: Evolution of features during training for ANODEs. The top left tile shows the feature space for a randomly initialized ANODE and the bottom right tile shows the features after training.

Summary of Neural ODE I

- A neural network is a popular type of machine learning model.
- Neural networks are built with Linear Algebra and optimized using Calculus.
- Neural network consists of series of "Layers" which are repetition of Matrix operations and Activation functions.
- Each layer introduces a little bit of error that compounds through the network.
- Adding more layers reduces error.
- But adding more than necessary layers may increase error because of over fitting.
- Residual Networks reduces output of previous layers to the output of new layers thus reducing the error.
- Ordinary Differential equations involve one or more ordinary derivatives of unknown functions.
- Euler method is a numerical method to solve univariable Differential equations.

Summary of Neural ODE II

- ResNet output of a layer is of form "$x_{k+1} = x_k + hF(x)$" is similar to Euler equation for solving ODE's.
- More efficient than Euler's method is the Adjoint method used for optimization purposes.
- The advantage of a Neural ODE is there is no need to specify the number of layers required, based on the required accuracy specified it trains itself choosing the number of layers by itself.
- Instead of Discrete layers, a continuous computational block of specified memory is used.
- Faster testing time, but slower training time (Good for Low Power Edge Computing)
- More accurate results for continuous time models

Summary of Neural ODE III

- Limitation of Neural ODE is it doesn't work for data that is not linearly separable in its defined space
- We can try to avoid this limitation by increasing dimensions of data

Acknowledgements

- https://medium.com/@ml.at.berkeley/neural-ordinary-differential-equations-and-dynamics-models-1a4277fbb80
- https://arxiv.org/pdf/1904.01681.pdf
- https://arxiv.org/pdf/1806.07366.pdf