

# CS598 NODE 2: Augmented Neural ODEs – Bhavesh Shrimali

Emilien Dupont, Arnaud Doucet, Yee Whye Teh

November 4, 2021

# Table of Contents

- 1 **NODE**
- 2 ODE Flows
- 3 ANODE
- 4 Experiments
- 5 Conclusions
- 6 Code

# Introduction: Neural ODEs

- ResNet update

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

# Introduction: Neural ODEs

- ResNet update

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

- More generally

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \Delta t \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

# Introduction: Neural ODEs

- ResNet update

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

- More generally

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \Delta t \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

- Interpret  $\mathbf{h}_{t+1} - \mathbf{h}_t$  as finite-difference discretization of  $\dot{\mathbf{h}}(t)$  with the time step  $\Delta t = 1$

$$\lim_{\Delta t \rightarrow 0^+} \frac{\mathbf{h}_{t+\Delta t} - \mathbf{h}_t}{\Delta t} = \frac{d\mathbf{h}(t)}{dt} = \mathbf{f}(\mathbf{h}(t), t)$$

# Introduction: Neural ODEs

- ResNet update

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

- More generally

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \Delta t \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

- Interpret  $\mathbf{h}_{t+1} - \mathbf{h}_t$  as finite-difference discretization of  $\dot{\mathbf{h}}(t)$  with the time step  $\Delta t = 1$

$$\lim_{\Delta t \rightarrow 0^+} \frac{\mathbf{h}_{t+\Delta t} - \mathbf{h}_t}{\Delta t} = \frac{d\mathbf{h}(t)}{dt} = \mathbf{f}(\mathbf{h}(t), t)$$

- The hidden state can then be parameterized by an ODE,  $\mathbf{x} \mapsto \phi(\mathbf{x})$

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{f}(\mathbf{h}(t), t), \quad \mathbf{h}(0) = \mathbf{x} \quad t \in (0, T]$$

# Introduction: Neural ODEs

- ResNet update

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

- More generally

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \Delta t \mathbf{f}_t(\mathbf{h}_t) \quad \mathbf{h}_t \in \mathbb{R}^d \quad \text{and} \quad \mathbf{f}_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

- Interpret  $\mathbf{h}_{t+1} - \mathbf{h}_t$  as finite-difference discretization of  $\dot{\mathbf{h}}(t)$  with the time step  $\Delta t = 1$

$$\lim_{\Delta t \rightarrow 0^+} \frac{\mathbf{h}_{t+\Delta t} - \mathbf{h}_t}{\Delta t} = \frac{d\mathbf{h}(t)}{dt} = \mathbf{f}(\mathbf{h}(t), t)$$

- The hidden state can then be parameterized by an ODE,  $\mathbf{x} \mapsto \phi(\mathbf{x})$

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{f}(\mathbf{h}(t), t), \quad \mathbf{h}(0) = \mathbf{x} \quad t \in (0, T]$$

- $\mathbf{h}(T)$  is the learned features by the model at  $T$ .

# Introduction: Neural ODEs

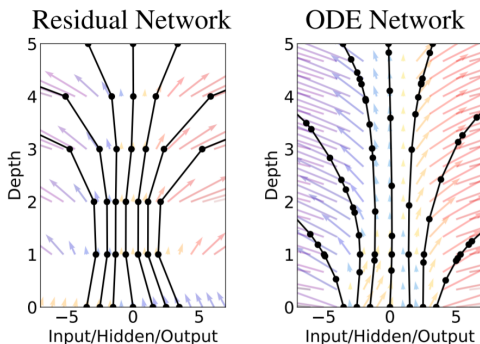


Figure: Neural ODEs (right) as continuous-time ResNets (left) [CRBD19]

- In ResNets: map an input  $x$  to output  $y$  by a forward pass



# Introduction: Neural ODEs

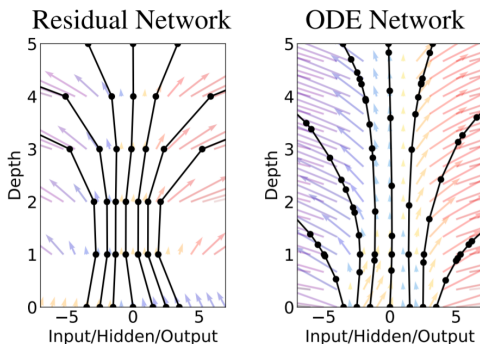


Figure: Neural ODEs (right) as continuous-time ResNets (left) [CRBD19]

- In ResNets: map an input  $x$  to output  $y$  by a forward pass
- Tune the weights of the network to minimize  $d(y, y_{\text{true}})$

# Introduction: Neural ODEs

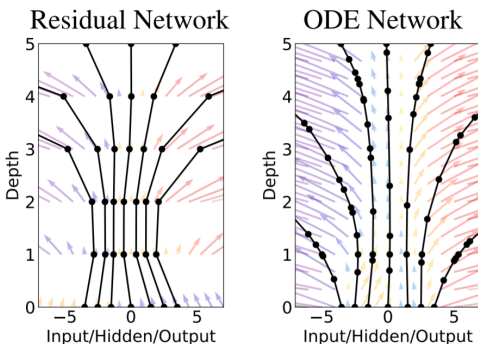


Figure: Neural ODEs (right) as continuous-time ResNets (left) [CRBD19]

- In ResNets: map an input  $x$  to output  $y$  by a forward pass
- Tune the weights of the network to minimize  $d(y, y_{\text{true}})$
- For NODEs: adjust the dynamics of the system encoded by  $f$  such that the ODE transforms input  $x$  to  $y$  to minimize  $d(y, y_{\text{true}})$

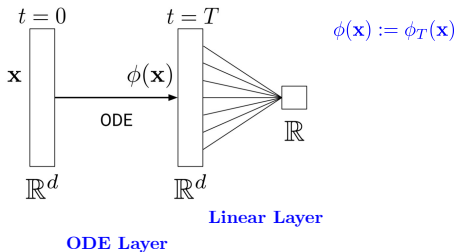
# Table of Contents

- 1 NODE
- 2 ODE Flows
- 3 ANODE
- 4 Experiments
- 5 Conclusions
- 6 Code

# Introduction: ODE flows

- The flow associated to  $\mathbf{f}(\mathbf{h}(t), t)$  is given by  $\phi(t)$

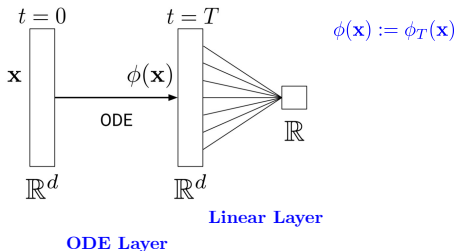
$$\phi_t : \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad \phi_t(\mathbf{x}) = \mathbf{h}(t) \quad \text{with} \quad \mathbf{h}(0) = \mathbf{x}$$



# Introduction: ODE flows

- The flow associated to  $\mathbf{f}(\mathbf{h}(t), t)$  is given by  $\phi(t)$

$$\phi_t : \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad \phi_t(\mathbf{x}) = \mathbf{h}(t) \quad \text{with} \quad \mathbf{h}(0) = \mathbf{x}$$

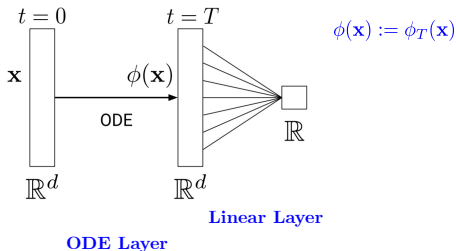


- The flow measures how the states of the ODE at a given time  $t$  depend on the initial conditions  $\mathbf{x}$ .

# Introduction: ODE flows

- The flow associated to  $\mathbf{f}(\mathbf{h}(t), t)$  is given by  $\phi(t)$

$$\phi_t : \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad \phi_t(\mathbf{x}) = \mathbf{h}(t) \quad \text{with} \quad \mathbf{h}(0) = \mathbf{x}$$

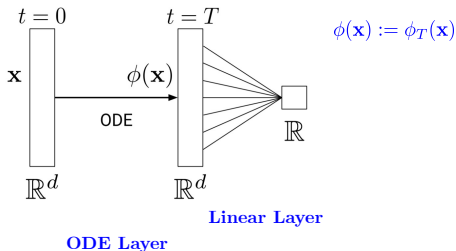


- The flow measures how the states of the ODE at a given time  $t$  depend on the initial conditions  $\mathbf{x}$ .
- The features of the ODE,  $\phi(\mathbf{x})$ , are the “flow at the final time  $T$ ”

# Introduction: ODE flows

- The flow associated to  $\mathbf{f}(\mathbf{h}(t), t)$  is given by  $\phi(t)$

$$\phi_t : \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad \phi_t(\mathbf{x}) = \mathbf{h}(t) \quad \text{with} \quad \mathbf{h}(0) = \mathbf{x}$$



- The flow measures how the states of the ODE at a given time  $t$  depend on the initial conditions  $\mathbf{x}$ .
- The features of the ODE,  $\phi(\mathbf{x})$ , are the “flow at the final time  $T$ ”
- For classification/regression problems, often define a NODE  $g : \mathbb{R}^d \rightarrow \mathbb{R}$  as  $g(\mathbf{x}) = \mathcal{L}(\phi(\mathbf{x}))$ , where  $\mathcal{L} : \mathbb{R}^d \rightarrow \mathbb{R}$  is a linear map and  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^d$  is the mapping from data to features.

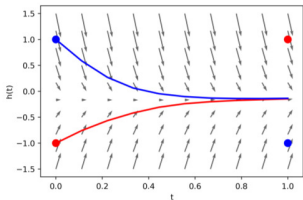
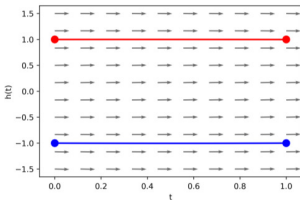
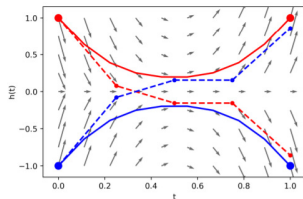
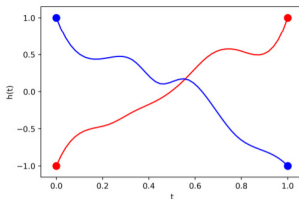
# Limitations of Neural ODEs/ODE Flows

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



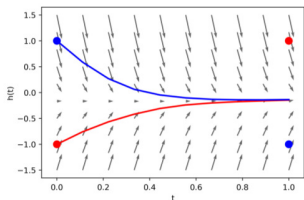
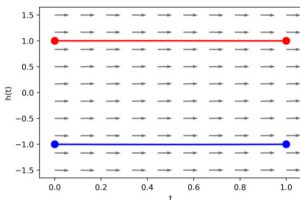
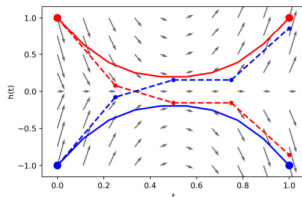
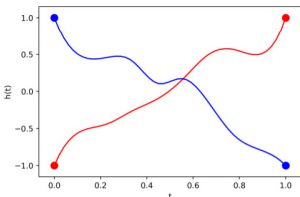
# Limitations of Neural ODEs/ODE Flows

- Let  $g_{1d} : \mathbb{R} \rightarrow \mathbb{R}$  be a function such that  $g_{1d}(-1) = 1$  and  $g_{1d}(1) = -1$ .
- Proposition 1: The flow of an ODE cannot represent  $g_{1d}(x)$ .**



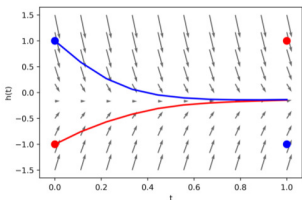
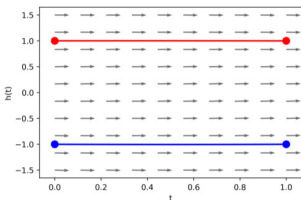
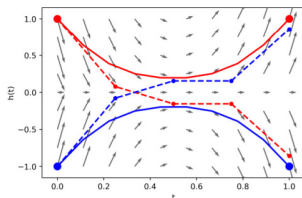
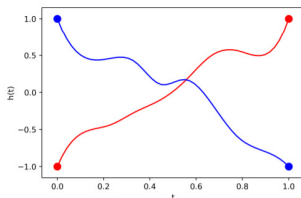
# Limitations of Neural ODEs/ODE Flows

- ResNets, on the other hand, can!



# Limitations of Neural ODEs/ODE Flows

- ResNets, on the other hand, can!



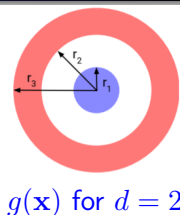
- ResNets are a discretization of the ODE, allowing the trajectories to make discrete jumps to cross each other

# Limitations of Neural ODEs/ODE Flows

- Let  $0 < r_1 < r_2 < r_3$  and let  $g : \mathbb{R}^d \mapsto \mathbb{R} \ni$

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

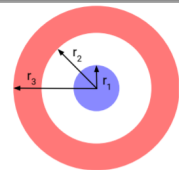
where  $\|\cdot\|$  is a Euclidian norm



# Limitations of Neural ODEs/ODE Flows

- Let  $0 < r_1 < r_2 < r_3$  and let  $g : \mathbb{R}^d \mapsto \mathbb{R} \ni$

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



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

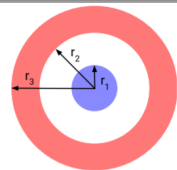
where  $\|\cdot\|$  is a Euclidean norm

- In order for the linear layer to map the blue and red points to  $-1$  and  $1$  respectively, **the features  $\phi(\mathbf{x})$  for the blue and red points must be linearly separable.**

# Limitations of Neural ODEs/ODE Flows

- Let  $0 < r_1 < r_2 < r_3$  and let  $g : \mathbb{R}^d \mapsto \mathbb{R} \ni$

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



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

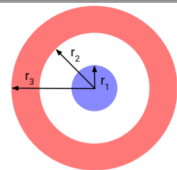
where  $\|\cdot\|$  is a Euclidean norm

- In order for the linear layer to map the blue and red points to  $-1$  and  $1$  respectively, **the features  $\phi(\mathbf{x})$  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

# Limitations of Neural ODEs/ODE Flows

- Let  $0 < r_1 < r_2 < r_3$  and let  $g : \mathbb{R}^d \mapsto \mathbb{R} \ni$

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



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

where  $\|\cdot\|$  is a Euclidean norm

- In order for the linear layer to map the blue and red points to  $-1$  and  $1$  respectively, **the features  $\phi(\mathbf{x})$  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
- The feature mapping  $\phi(\mathbf{x})$  is a homeomorphism**

# Limitations of Neural ODEs/ODE Flows

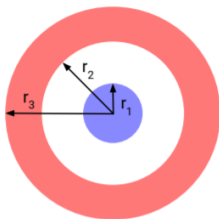


Figure:  $g(x)$  for  $d = 2$

- **Discrete points and continuous regions:** In practice, NODEs are trained on inputs sampled from the continuous regions of the annulus (and the sphere). The flow could then squeeze through the gaps between sampled points making it possible for the NODE to learn a good approximation of the function.



# Limitations of Neural ODEs/ODE Flows

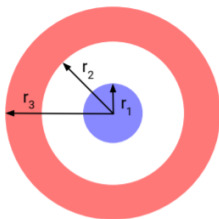


Figure:  $g(\mathbf{x})$  for  $d = 2$

- **Discrete points and continuous regions:** In practice, NODEs are trained on inputs sampled from the continuous regions of the annulus (and the sphere). The flow could then squeeze through the gaps between sampled points making it possible for the NODE to learn a good approximation of the function.
- The resulting flows lead to **ill-posed** ODE problems which are expensive to solve.

<https://github.com/EmilienDupont/augmented-neural-odes>

# Limitations of Neural ODEs/ODE Flows

Figure: Flow visualization NODEs vs ANODEs<sup>1</sup>

---

<sup>1</sup><https://github.com/EmilienDupont/augmented-neural-odes>

# Table of Contents

- 1 NODE
- 2 ODE Flows
- 3 ANODE**
- 4 Experiments
- 5 Conclusions
- 6 Code

# Augmented Neural ODE

- **Main idea:** Augment the space on which the ODE is learned

# Augmented Neural ODE

- **Main idea:** Augment the space on which the ODE is learned
- $\mathbb{R}^d \rightarrow \mathbb{R}^{d+p} \implies$  allows the ODE to lift points into additional dimensions to avoid trajectories from intersecting each other.

# Augmented Neural ODE

- **Main idea:** Augment the space on which the ODE is learned
- $\mathbb{R}^d \rightarrow \mathbb{R}^{d+p} \implies$  allows the ODE to lift points into additional dimensions to avoid trajectories from intersecting each other.
- Let  $\mathbf{a}(t) \in \mathbb{R}^p$  be a point in the augmented part of the space, the reformulation can be written as

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

# Augmented Neural ODE

- **Main idea:** Augment the space on which the ODE is learned
- $\mathbb{R}^d \rightarrow \mathbb{R}^{d+p} \implies$  allows the ODE to lift points into additional dimensions to avoid trajectories from intersecting each other.
- Let  $\mathbf{a}(t) \in \mathbb{R}^p$  be a point in the augmented part of the space, the reformulation can be written as

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

- The authors hypothesize that this will make the **learned  $\mathbf{f}$  smoother, giving rise to simpler flows** that the ODE solver can compute in fewer steps

# Augmented Neural ODE

- **Main idea:** Augment the space on which the ODE is learned
- $\mathbb{R}^d \rightarrow \mathbb{R}^{d+p} \implies$  allows the ODE to lift points into additional dimensions to avoid trajectories from intersecting each other.
- Let  $\mathbf{a}(t) \in \mathbb{R}^p$  be a point in the augmented part of the space, the reformulation can be written as

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

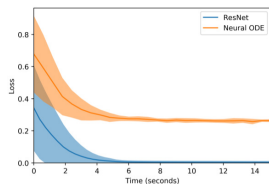
- The authors hypothesize that this will make the **learned  $\mathbf{f}$  smoother, giving rise to simpler flows** that the ODE solver can compute in fewer steps
- Run empirical experiments to substantiate the above hypotheses



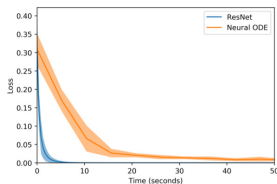
# Table of Contents

- 1 NODE
- 2 ODE Flows
- 3 ANODE
- 4 Experiments**
- 5 Conclusions
- 6 Code

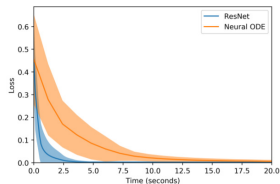
# Experiments



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



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

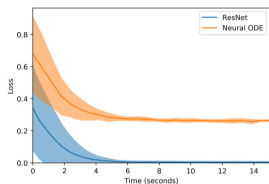


(c) Separable function in  $d = 2$

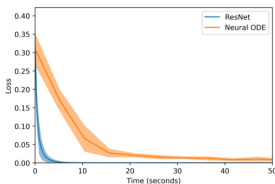
Figure: Training losses for NODEs and ResNets.

- **Baseline comparison:** Train on data which can be “**linearly separable**”

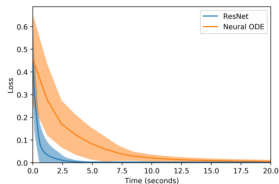
# Experiments



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



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



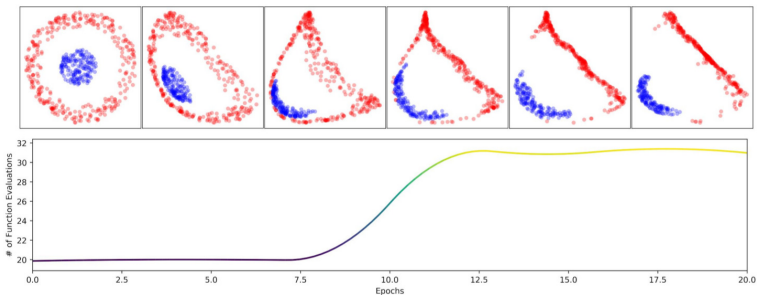
(c) Separable function in  $d = 2$

Figure: Training losses for NODEs and ResNets.

- **Baseline comparison:** Train on data which can be “**linearly separable**”
- ResNet easily fits  $g(\mathbf{x})$  for both  $d = 1, 2$ , but NODE cannot. For  $d = 2$  NODE eventually learns but is far more costly compared to ResNets.

# Computational Cost and NFE

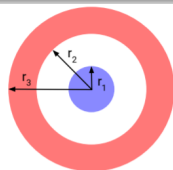
- As the training progresses and flows get increasingly complex, the number of steps for the ODE solver increases  
 ([CRBD19, GCB<sup>+</sup>18])  $\implies$  NFE  $\uparrow$



**Figure:** Evolution of the feature space as training progresses and the corresponding NFE required to solve the ODE. As the ODE needs to break apart the annulus, NFE shoots up

# Augmented Neural ODE: Experiments (Toy)

- ANODEs fit the functions that NODEs can't



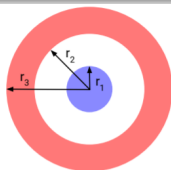
$g(\mathbf{x})$  for  $d = 2$

# Augmented Neural ODE: Experiments (Toy)

- ANODEs fit the functions that NODEs can't
- Much faster despite higher input dim

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

$$\begin{bmatrix} \mathbf{h}(0) \\ \mathbf{a}(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}$$



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

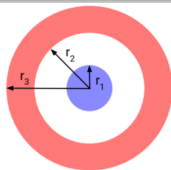
# Augmented Neural ODE: Experiments (Toy)

- ANODEs fit the functions that NODEs can't
- Much faster despite higher input dim

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

$$\begin{bmatrix} \mathbf{h}(0) \\ \mathbf{a}(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}$$

- Learn simpler flows and thus lower NFEs



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

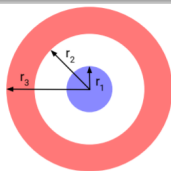
# Augmented Neural ODE: Experiments (Toy)

- ANODEs fit the functions that NODEs can't
- Much faster despite higher input dim

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

$$\begin{bmatrix} \mathbf{h}(0) \\ \mathbf{a}(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}$$

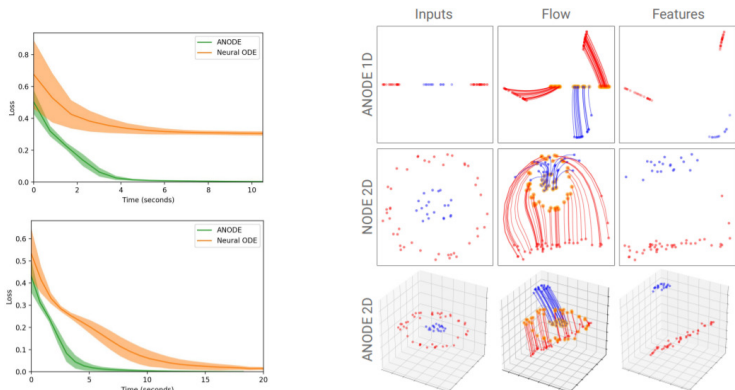
- Learn simpler flows and thus lower NFEs
- Better generalization



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



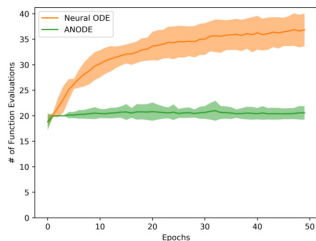
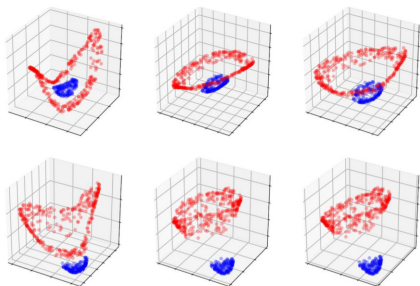
# Augmented Neural ODE: Experiments (Toy)



**Figure:** Loss function for NODEs and ANODEs trained on  $g(x)$  in  $d = 1$  (top) and  $d = 2$  (bottom) dimensions. **ANODEs are faster**. On the right are the flows learned by NODEs and ANODEs. **ANODEs learn simple nearly linear flows**, while NODEs learn complex (expensive) flows

# Augmented Neural ODE: Experiments (Toy)

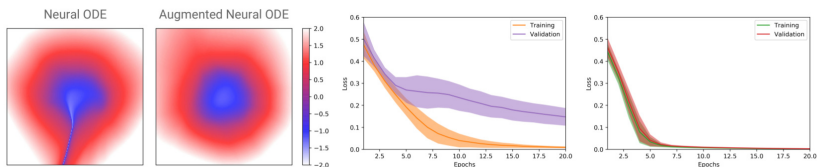
- Number of function evaluations:** As ANODEs learn simpler flows, the NFEs required by ANODEs hardly increase during training whereas it doubles for NODEs



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

# Augmented Neural ODE: Experiments (Toy)

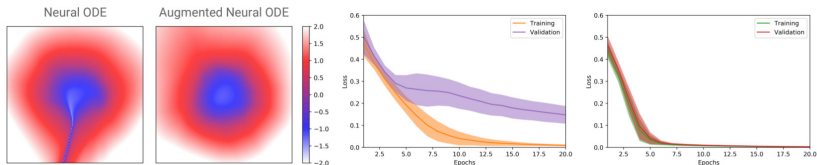
- Generalization:** In order to test if indeed ANODEs generalize better  $\rightarrow$  visualize to which value each point in the input space gets mapped by a NODE and an ANODE (both are optimized to approximately zero training loss).



**Figure:** (Left) Plots of how NODEs and ANODEs map points in the input space to different outputs (for zero training loss). ANODEs generalize better. (Middle) Training and validation losses for NODE, (Right) for ANODE

# Augmented Neural ODE: Experiments (Toy)

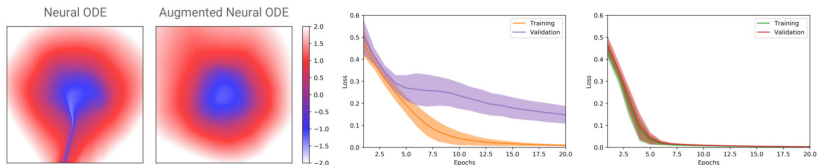
- Generalization:** In order to test if indeed ANODEs generalize better  $\rightarrow$  visualize to which value each point in the input space gets mapped by a NODE and an ANODE (both are optimized to approximately zero training loss).
- NODEs (below) can only continuously deform the input space, the learned flow must squeeze points in the inner circle through the annulus leading to poor generalization



**Figure:** (Left) Plots of how NODEs and ANODEs map points in the input space to different outputs (for zero training loss). ANODEs generalize better. (Middle) Training and validation losses for NODE, (Right) for ANODE

# Augmented Neural ODE: Experiments (Toy)

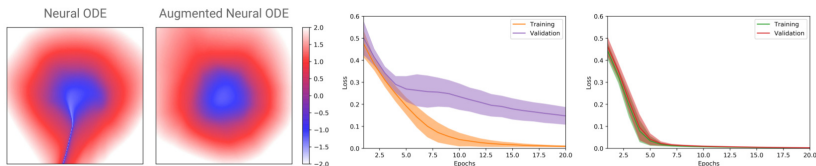
- **Generalization:** ANODEs, in contrast, map all points in the input space to reasonable values.



**Figure:** (Left) Plots of how NODEs and ANODEs map points in the input space to different outputs (for zero training loss). ANODEs generalize better. (Middle) Training and validation losses for NODE, (Right) for ANODE

# Augmented Neural ODE: Experiments (Toy)

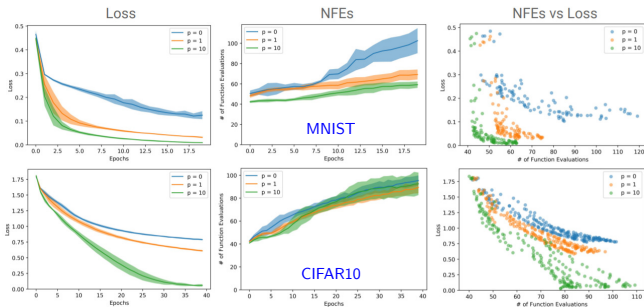
- **Generalization:** ANODEs, in contrast, map all points in the input space to reasonable values.
- The authors [DDT19] also consider a further test. They create a validation set by removing random slices of the input space and train both NODEs and ANODEs on the training set and plot the evolution of the validation loss during training.



**Figure:** (Left) Plots of how NODEs and ANODEs map points in the input space to different outputs (for zero training loss). ANODEs generalize better. (Middle) Training and validation losses for NODE, (Right) for ANODE

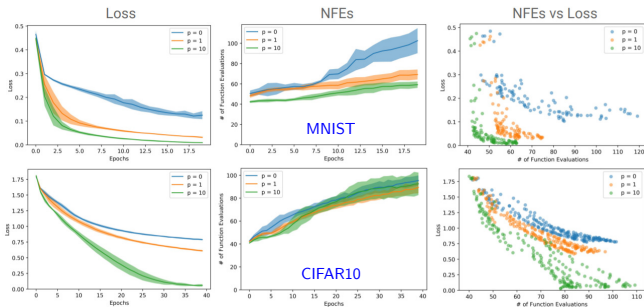
# Augmented Neural ODE: Experiments (Images)

- Convolutional architectures for  $\mathbf{f}(\mathbf{h}(t), t)$



# Augmented Neural ODE: Experiments (Images)

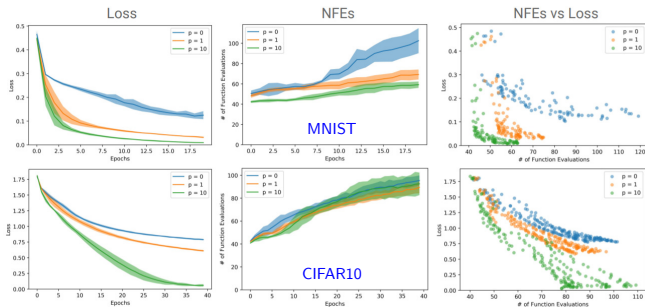
- Convolutional architectures for  $\mathbf{f}(\mathbf{h}(t), t)$
- $\mathbf{h}(t)$  is now  $\mathbb{R}^{c \times h \times w}$





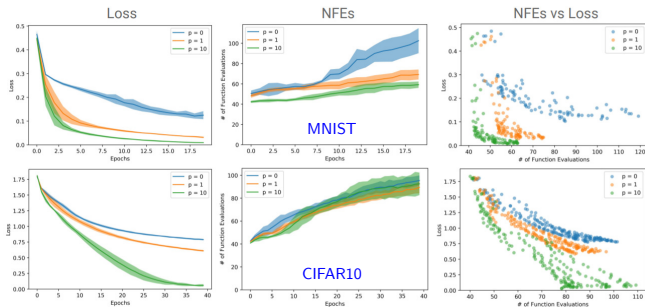
# Augmented Neural ODE: Experiments (Images)

- Convolutional architectures for  $\mathbf{f}(\mathbf{h}(t), t)$
- $\mathbf{h}(t)$  is now  $\mathbb{R}^{c \times h \times w}$
- Augmented space  $\mathbb{R}^{c \times h \times w} \rightarrow \mathbb{R}^{(c+p) \times h \times w}$ , i.e. add  $p$  channels of zeros to the input image

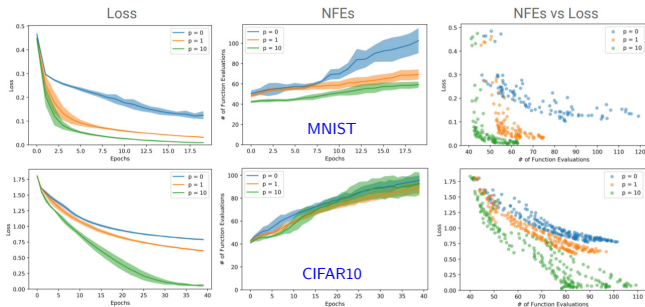


# Augmented Neural ODE: Experiments (Images)

- Convolutional architectures for  $\mathbf{f}(\mathbf{h}(t), t)$
- $\mathbf{h}(t)$  is now  $\mathbb{R}^{c \times h \times w}$
- Augmented space  $\mathbb{R}^{c \times h \times w} \rightarrow \mathbb{R}^{(c+p) \times h \times w}$ , i.e. add  $p$  channels of zeros to the input image
- ANODEs train faster and obtain lower losses at a smaller computational cost than NODEs

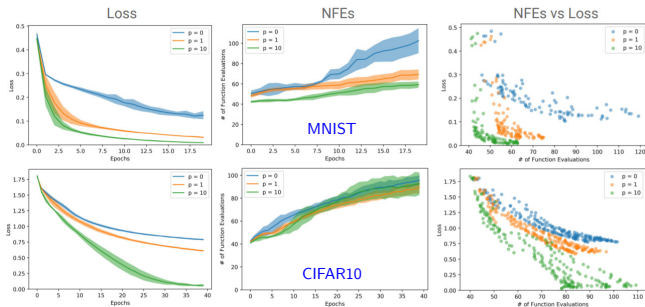


# Augmented Neural ODE: Experiments (Images)



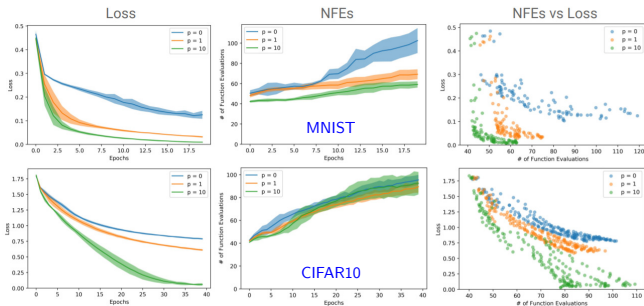
- On MNIST, ANODEs with  $p = 10$  augmented channels achieves the same loss in  $\sim 10x$  fewer iterations (for CIFAR10, ANODEs are roughly 5 times faster)

# Augmented Neural ODE: Experiments (Images)



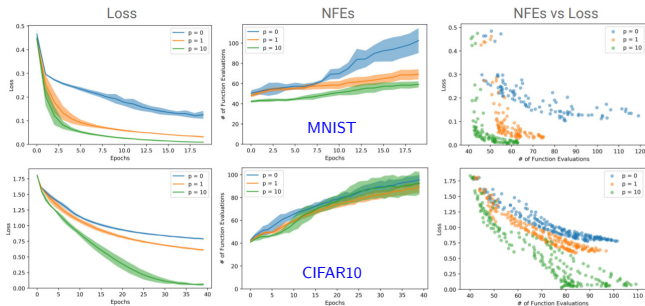
- On MNIST, ANODEs with  $p = 10$  augmented channels achieves the same loss in  $\sim 10x$  fewer iterations (for CIFAR10, ANODEs are roughly 5 times faster)
- Plot of the NFEs against the loss helps in inferring how complex a flow (i.e. how many NFEs) are required to model a function that achieves a certain loss.

# Augmented Neural ODE: Experiments (Images)



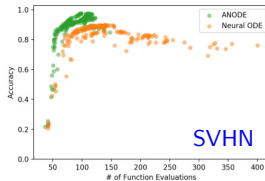
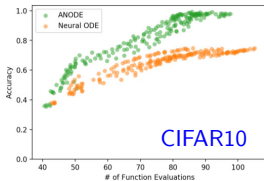
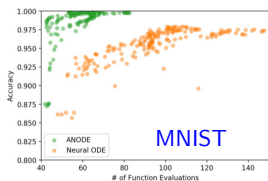
- For example, to compute a function which obtains a loss of 0.8 on CIFAR10, a NODE requires approximately 100 function evaluations whereas ANODEs only require 50

# Augmented Neural ODE: Experiments (Images)



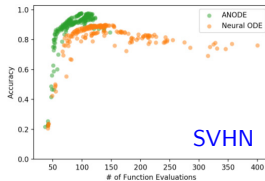
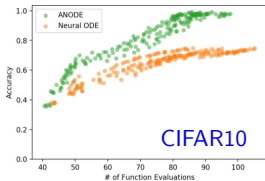
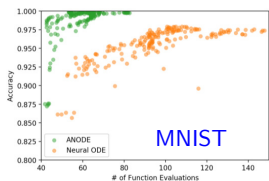
- For example, to compute a function which obtains a loss of 0.8 on CIFAR10, a NODE requires approximately 100 function evaluations whereas ANODEs only require 50
- Note that both ANODEs and NODEs have the same number of parameters for a fair comparison

# Augmented Neural ODE: Experiments (Images)



- ANODEs achieve higher accuracy at a lower computational cost than NODEs

# Augmented Neural ODE: Experiments (Images)

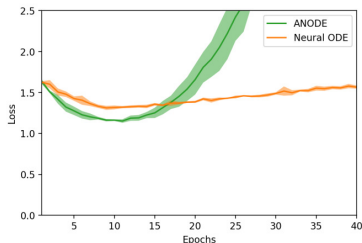
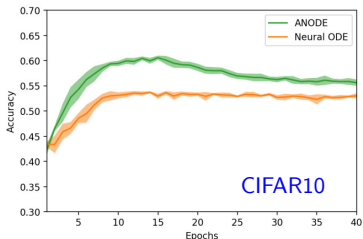
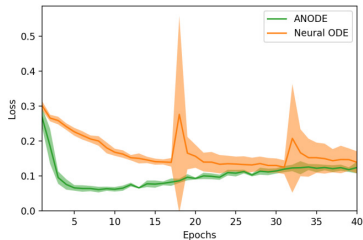
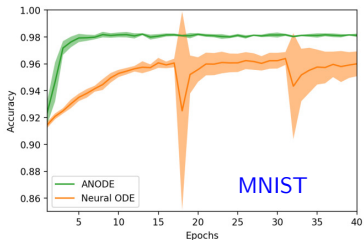


- ANODEs achieve higher accuracy at a lower computational cost than NODEs
- Note that both ANODEs and NODEs have the same number of parameters for a fair comparison



# Augmented Neural ODE: Experiments (Images)

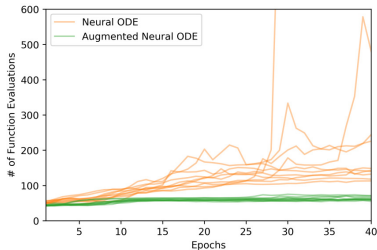
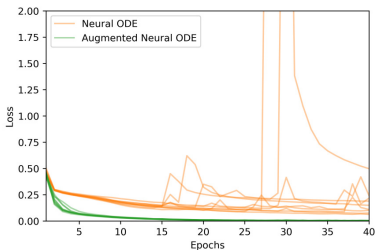
**Generalization:** ANODEs achieve lower test losses and higher accuracy on image datasets as well



# Augmented Neural ODE: Experiments (Images)

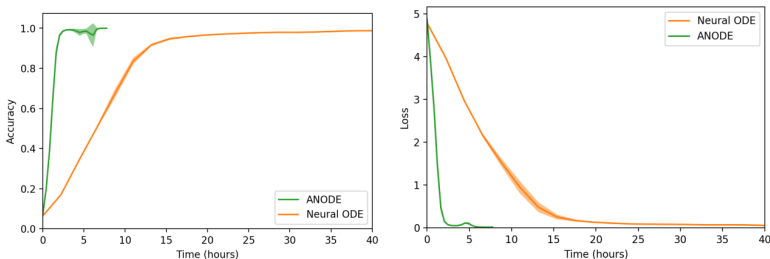
## Instabilities:

- NFEs could often become prohibitively large (in excess of 1000, which roughly corresponds to a 1000-layer ResNet). For example, when overfitting a NODE on MNIST, the learned flow can become so ill posed the ODE solver requires timesteps that are smaller than machine precision resulting in underflow.
- Further, this complex flow often leads to unstable training resulting in exploding losses. Augmentation consistently leads to stable training and fewer NFEs, even when overfitting



# Augmented Neural ODE: Experiments (Images)

**Scaling:** Both NODEs and ANODEs on 200 classes of  $64 \times 64$  ImageNet



**Figure:** ANODEs **scale better**, **achieve lower training losses** and have **roughly 10x faster training times**

# Table of Contents

- 1 NODE
- 2 ODE Flows
- 3 ANODE
- 4 Experiments
- 5 Conclusions**
- 6 Code

# Augmented Neural ODE: Conclusions

- While ANODEs are faster than NODEs, **they are still slower than ResNets.**

# Augmented Neural ODE: Conclusions

- While ANODEs are faster than NODEs, **they are still slower than ResNets.**
- Augmentation changes the dimension of the input space which, depending on the application, may not be desirable

# Augmented Neural ODE: Conclusions

- While ANODEs are faster than NODEs, **they are still slower than ResNets.**
- Augmentation changes the dimension of the input space which, depending on the application, may not be desirable
- The augmented dimension  $p$  can be seen as an extra hyperparameter to tune.

# Augmented Neural ODE: Conclusions

- While ANODEs are faster than NODEs, **they are still slower than ResNets.**
- Augmentation changes the dimension of the input space which, depending on the application, may not be desirable
- The augmented dimension  $p$  can be seen as an extra hyperparameter to tune.
- For excessively large augmented dimensions (e.g. adding 100 channels to MNIST), the model tends to perform worse with higher losses and NFEs



# Augmented Neural ODE: Conclusions

- There are classes of functions NODEs cannot represent and, in particular, that NODEs only learn features that are homeomorphic to the input space

# Augmented Neural ODE: Conclusions

- There are classes of functions NODEs cannot represent and, in particular, that NODEs only learn features that are homeomorphic to the input space
- Showed through empirical experiments that this leads to slower learning and complex flows which are expensive to compute

# Augmented Neural ODE: Conclusions

- There are classes of functions NODEs cannot represent and, in particular, that NODEs only learn features that are homeomorphic to the input space
- Showed through empirical experiments that this leads to slower learning and complex flows which are expensive to compute
- Proposed Augmented Neural ODEs which learn the flow from input to features in an augmented space and show that ANODEs can model more complex functions using simpler flows while achieving lower losses, reducing computational cost, and improving stability and generalization.




# Table of Contents

- 1 NODE
- 2 ODE Flows
- 3 ANODE
- 4 Experiments
- 5 Conclusions
- 6 Code

# Augmented Neural ODE: Code

- Pytorch: <https://github.com/EmilienDupont/augmented-neural-odes>
- Julia (DiffeqFlux): [https://diffeqflux.sciml.ai/dev/examples/augmented\\_neural\\_ode/](https://diffeqflux.sciml.ai/dev/examples/augmented_neural_ode/)

# References

-  R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud.  
Neural ordinary differential equations, 2019, [1806.07366](#).
-  E. Dupont, A. Doucet, and Y. W. Teh.  
Augmented neural odes, 2019, [1904.01681](#).
-  W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud.  
Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018, [1810.01367](#).