Author

Bhavesh Shrimali

Introduction

This blog covers Augmented Neural ODEs, an improved and more expressive version of the celebrated Neural ODEs paper. Let's start by revisiting the Neural ODEs idea, and even before that let us revisit the ResNet update, which is given by the relation

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

where $\mathbf{h}_t$ corresponds to the hidden state vector at the t-th layer, and $\mathbf{f}_t$ corresponds to the residual mapping. This looks surprisingly similar to a forward euler discretization of an ODE

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

It is plain that with $\Delta t = 1$, we recover the ResNet update step. Now, if you instead consider t as a time-like variable, then I can take h_t on the LHS and take the limit of the step size going to zero, i.e.

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

We now have the hidden state parameterized by an ODE, $\mathbf{x} \mapsto \phi\left(\mathbf{x}\right)$,

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

The corresponding flow can be visualized to get an intuition of the transition from a ResNet to a Neural ODE (NODE),

To put things in perspective,

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

ODE Flows

Before introducing the idea of Augmented Neural ODEs (ANODEs), we briefly revisit the notion of an ODE flow. The flow corresponding to a vector field $\mathbf{f}(\mathbf{h}(t), t)$ is given by $\phi(t)$, such that,

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

It is worth noting that the flow resulting from a Neural ODE is homeomorphic, i.e. it is continuous and bijective with a continuous inverse. Physically, the flow measures how the states of the ODE at a given time t depend on the initial conditions $\mathbf{x}$. Note that for classification/regression problems, we often define a NODE $g : \mathbb{R}^d \to \mathbb{R}$ as $g(\mathbf{x}) = \mathcal{L}(\phi(\mathbf{x}))$, where $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is a linear map and $\phi : \mathbb{R}^d \to \mathbb{R}^d$ is the mapping from data to features.

Limitations of Neural ODEs/ODE Flows

It is important to note that not all functions can be approximated by a NODE/ODEFlow. Consider for instance $g_{1\text{ d}} : \mathbb{R} \to \mathbb{R}$, such that $g_{1\text{ d}}(-1) = 1$ and $g_{1\text{ d}}(1) = -1$. It can be seen clearly from the figure below that a NODE cannot approximate this function, no matter how small a timestep or how large the terminal time T. This is due to the fact that the ODE trajectories cannot cross each other. A formal proof can be found in the appendix in the paper, however it is simply built around the uniqueness of a solution to an ODE. An ODE cannot have two solutions that are different everywhere but at point. That is, the solutions are either identical or they do not intersect at any point. ResNets on the other hand do not suffer from this, as can be seen from the figure on the top-right.

Having motivated through a 1D example, let us now consider the 2D version of it, i.e.

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

In theory Neural ODEs cannot represent the above function, since the red and blue regions are not linearly separable. In this case too ResNets can approximate the function. Plotting the loss function gives a more complete picture

As it can bee seen from the above figure, in practice, Neural ODEs are able to approximate the function, but the resulting flow is much more complicated (see the time taken by the NODE to reach the same loss for the 2D example problem)

This motivates exploring an augmented space and seeing its effect the learned ODE. In other words, it turns out that zero padding the input, say with a p dimensional vector, dramatically improves the learning and the resulting Neural ODE (known as an `Augmented Neural ODE`) is able to gain expressivity and lead to simpler flows.

Augmented Neural ODEs (ANODEs)

As motivated above the idea is to augment the space on which the ODE is learned. In other words, $\mathbb{R}^d \to \mathbb{R}^{d+p}$ which 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

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

Plotting the loss function corresponding to each of the two toy examples verifies that ANODEs learn much simpler flows and the resulting loss function decays much faster compared to vanilla-Neural ODEs.

It can be seen that the corresponding flows are almost linear for ANODEs and therefore the number of function evaluations are much fewer compared to NODEs. This point is further reinforced when we plot the number of function evaluations (and resulting evolution of the features) corresponding to each of the two approaches

As we can see the number of function evaluations almost doubles for NODEs but remains roughly the same for ANODEs.

Generalization

In order to see the generalization properties of ANODEs the authors train both ANODE and NODE to have zero training loss and then visualize the points in the output space to which each point in the input gets mapped to.

ANODEs again lead to flows that are much more plausible compared to NODEs. This is because NODEs can only continuously deform the input space. Therefore, the learned flow must squeeze points in the inner circle through the annulus leading to poor generalization. In order to test the generalization properties of ANODEs, the authors 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. The same thing emerges out, that is, ANODEs generalize better!

Experiments

The authors carry out generative modeling experiments on the popular MNIST, CIFAR10 and SVHN datasets. The same story emerges from there as well. ANODEs outperform NODEs for the most part. For the figure below, $p = 0$ corresponds to the base case (NODEs), where p denotes the number of extra channels in the augmented space. Results for MNIST and CIFAR 10 are given below

Conclusions

Bottlenecks/limitations of ANODEs

A few additional insights that emerge from the experiments carried out by the authors are as follows

- While ANODEs are faster than NODEs, they are still slower than ResNets (see the figure from their appendix below)
- 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

The above figure corresponds to the 2D toy example, namely,

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

Conclusion

- There are classes of functions NODEs cannot represent and, in particular, that NODEs only learn features that are *homeomorphic* to the input space
- This leads to *slower learning and complex flows* which are computation-all expensive
- Augmented Neural ODEs learn the flow from input to features in an augmented space and can therefore model more complex functions using simpler flows while at the same time *achieving lower losses, incurring lower computational cost, and improved*

Code

The code to reproduce key findings from the paper is developed on top of a PyTorch library `torchdiffeq` and can be accessed at the authors' `git repository`.

Several other open source implementations are available online. A fast and flexible implementation in `Julia` is available in the `DiffEqFlux` library here, which builds on top of the `Flux.jl` framework and as part of the larger `SciML` ecosystem in Julia.

Citation

```
@misc{dupont2019augmented,
  title={Augmented Neural ODEs},
```

```
  author={Emilien Dupont and Arnaud Doucet and Yee Whye Teh},
  year={2019},
  eprint={1904.01681},
  archivePrefix={arXiv},
  primaryClass={stat.ML}
}

@misc{chen2019neural,
  title={Neural Ordinary Differential Equations},
  author={Ricky T. Q. Chen and Yulia Rubanova and Jesse Bettencourt and David Duvenaud},
  year={2019},
  eprint={1806.07366},
  archivePrefix={arXiv},
  primaryClass={cs.LG}
}

@misc{grathwohl2018ffjord,
  title={FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models},
  author={Will Grathwohl and Ricky T. Q. Chen and Jesse Bettencourt and Ilya Sutskever and D
  year={2018},
  eprint={1810.01367},
  archivePrefix={arXiv},
  primaryClass={cs.LG}
}
```