Learning Functions with Structured Jacobians

Jonathan Lorraine, Safwan Hossain

lorraine@cs.toronto.edu, hossain@cs.toronto.edu

VECTOR INSTITUTE

UNIVERSITY OF TORONTO

Motivation

- We want to enforce global properties on neural networks (NN)
- It is easy to enforce some NN properties by choice of output activation. Ex., softmax for classification.
- But, it is difficult to enforce properties of the NNs Jacobian.
- Some NN properties are easiest to phrase as constraints on the Jacobian. Ex., being invertible or Lipschitz.
- Let's directly parameterize the Jacobian to easily constrain with an appropriate output activation.

Contributions

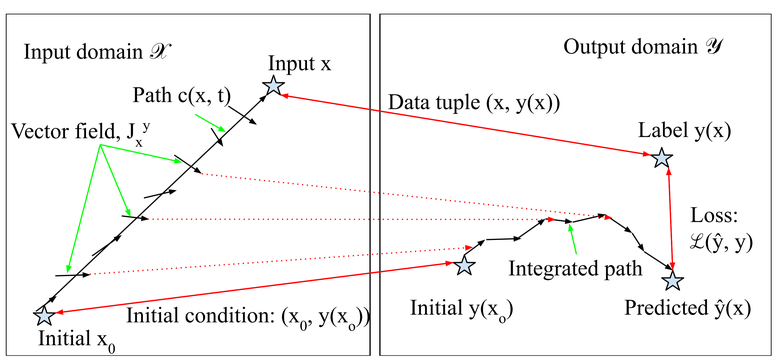
- A way to learn functions by parameterizing their Jacobian and integrating along paths.

Function Evaluation as Line Integration

- We can evaluate a function y by combining initial conditions with an integral of the Jacobian $J_{\mathrm{x}}^{\mathrm{y}}$ on a differentiable path c between the initial and final values, x_o and x.

$$\mathrm{y}(\mathrm{x}) = \underbrace{\mathrm{y}(\mathrm{x}_o)}_{\text{Initial condition}} + \underbrace{\int_{t=0}^{t=1} \overbrace{J_{\mathrm{x}}^{\mathrm{y}}}^{\text{Jacobian}} \left(\overbrace{c(t, \mathrm{x}_o, \mathrm{x})}^{\text{Path}} \right) \overbrace{c^{(t)}(t, \mathrm{x}_o, \mathrm{x})}^{\text{Rectification}} dt}_{\text{Path integral}}$$

- Use a linear path $c(t, \mathrm{x}_o, \mathrm{x}) = (1-t)\mathrm{x}_o + t\mathrm{x}$ for simplicity.



A visualization of our training procedure.

Structuring the Jacobian

- We propose to parameterize a vector/matrix field $J_\theta(\mathrm{x})$ to approximate the Jacobian $J_{\mathrm{x}}^{\mathrm{y}}$ via a network with weights θ.
- We can evaluate the loss by doing a forward pass through the integrator to get a predicted label.
- We leverage differentiable integrators from Chen et al. [1] to backpropagate through integration to train θ.
- We use a k-scaled tanh output activation to ensure our Jacobian norm lies in $[-k, k]$ for k-Lipschitz.
- We constrain the Jacobian determinant to be non-zero everywhere for invertibility, by using an output activation which is guaranteed to be positive definite:

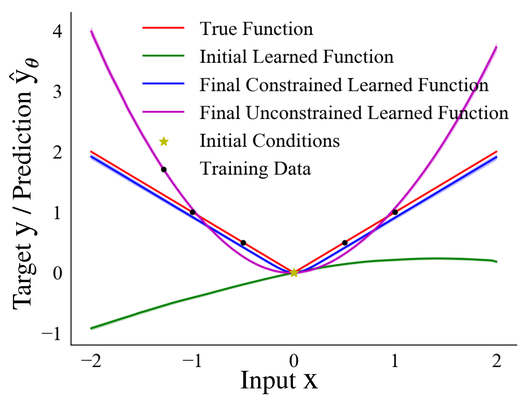
$$J'_\theta(\mathrm{x}) = J_\theta(\mathrm{x}) J_\theta^T(\mathrm{x}) + \epsilon I$$

Note that $J_\theta(\mathrm{x}) J_\theta^T(\mathrm{x})$ is a flexible PSD matrix, while adding ϵI makes it positive definite.

- By the Implicit Function Theorem, we can compute y^{-1} by integrating the inverse Jacobian along a path $c(t, \mathrm{y}_o, \mathrm{y})$:

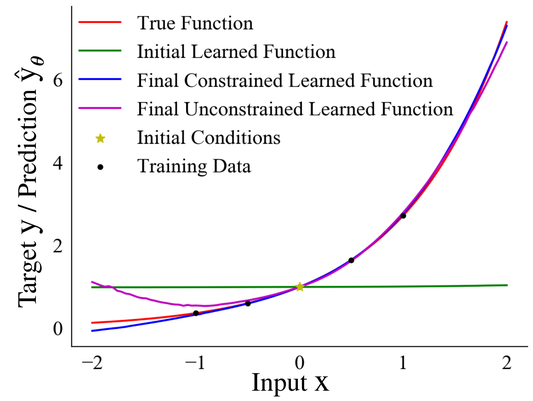
$$\mathrm{x}(\mathrm{y}, \theta) = \mathrm{x}(\mathrm{y}_o) + \int_{t=0}^{t=1} \left(J_\theta \left(c(t, \mathrm{y}_o, \mathrm{y}) \right) \right)^{-1} c^{(t)}(t, \mathrm{y}_o, \mathrm{y}) dt$$

Learning Lipschitz Functions



A graph of the 1-Lipschitz target function $|\mathrm{x}|$, the NN after training, and the NN at initialization. The initial NN is inaccurate, while the final NN closely matches the target function. Additionally, the NN is 1-Lipschitz at initialization and after training. We include graphs of an unconstrained NN which - in contrast - does not generalize to unseen data, in-part because the derivative is not bounded by $[-1, 1]$.

Learning Invertible Functions



A graph of the invertible target function $\exp(\mathrm{x})$, the NN after training, and the NN at initialization. The initial NN is inaccurate, while the final NN closely matches the target function. We include graphs of an unconstrained NN whose Jacobian can be zero. Note how the unconstrained function is not invertible everywhere, because the Jacobian determinant is 0 for some arguments.

Challenges

- In higher dimensions when parameterizing the Jacobian of $\mathrm{y} : \mathbb{R}^n \to \mathbb{R}^m$ if $n > 1, m = 1$ we are parameterizing a vector field J_θ, which may not be conservative. Thus, our NN may not be the Jacobian of a function.
- The invertible output activation $\mathrm{a}(\mathrm{b}) = \mathrm{b}^T\mathrm{b} + \epsilon I$ is restrictive, because we can only parameterize PSD Jacobians while general invertible functions have a non-zero determinant.
- Inverting J_θ may be a computational bottleneck in high dimensions. Can we learn an easily invertible matrix?

References

[1] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In Advances in Neural Information Processing Systems, pages 6571–6583, 2018.