Solving ODEs with Neural Networks

SON TU AND BEN WRIGHT

Why this project?

There has been lots of recent interest in solving differential equations with neural networks. Namely:

1. J. Han, A. Jentzen, and W. E, “Solving high-dimensional partial differential equations using deep learning,” Proceedings of the National Academy of Sciences, vol. 115, no. 34, pp. 8505–8510, 2018.
2. M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part i): Datadriven solutions of nonlinear partial differential equations,”
3. A. Singh, M. Bauer, and S. Joshi, “Physics informed convex artificial neural networks (PICANNs) for optimal transport based density estimation,”
4. T. Nguyen-Thien and T. Tran-Cong, “Approximation of functions and their derivatives: A neural network implementation with applications,” Applied Mathematical Modelling, vol. 23, no. 9, pp. 687–704, 199

The Trick

An ODE consists of a function $F\left(x, y, y', y'', \dots, y^{(n)}\right) = 0$ and sometimes initial conditions $y(a) = c, y(b) = d$.

Assume the solution to the ODE has the structure that $y(x) = NN(x)$, that is, the solution can be modeled on – or approximated by – a neural network output. Since the derivatives of a neural network can be computed with either back propagation or finite difference method, one can measure the L^2 norm of the approximate solution $\left|\left|F\left(x, NN(x), NN(x)', \dots, NN(x)^{(n)}\right)\right|\right|_2$ and use this as a loss function!

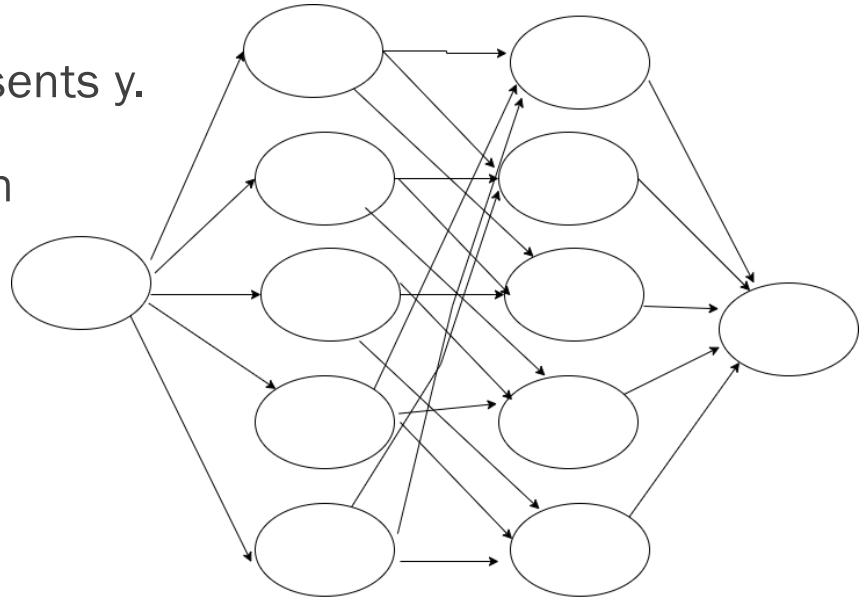
The parameters of the network can then be updated according to an algorithm designed to optimize this loss!

What is a neural network/what was our network architecture?

We studied a feed-forward, fully connected neural network generally with two hidden layers of 10 nodes with hyperbolic tangent nonlinear activation function.

The input node represents x and the output node represents y.

The value at each node is a weighted linear combination of all nodes leading into it, with the tanh() activation function then applied.



Ex. 1: $y'(x) + y(x) - x = 0, y(0) = 1$

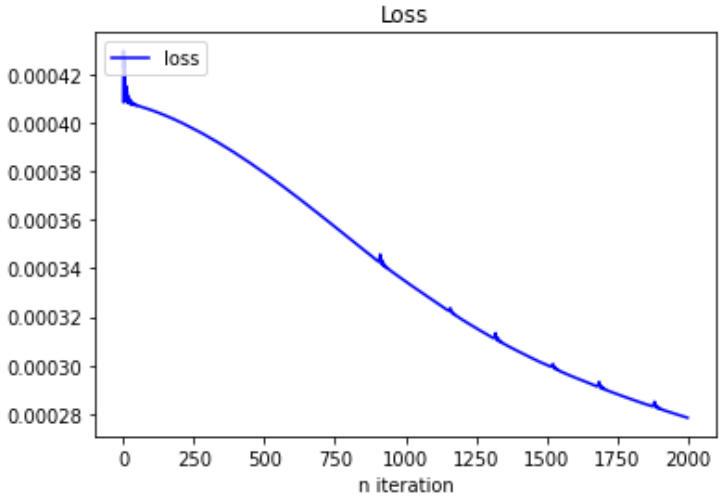
In this case, we use the loss function

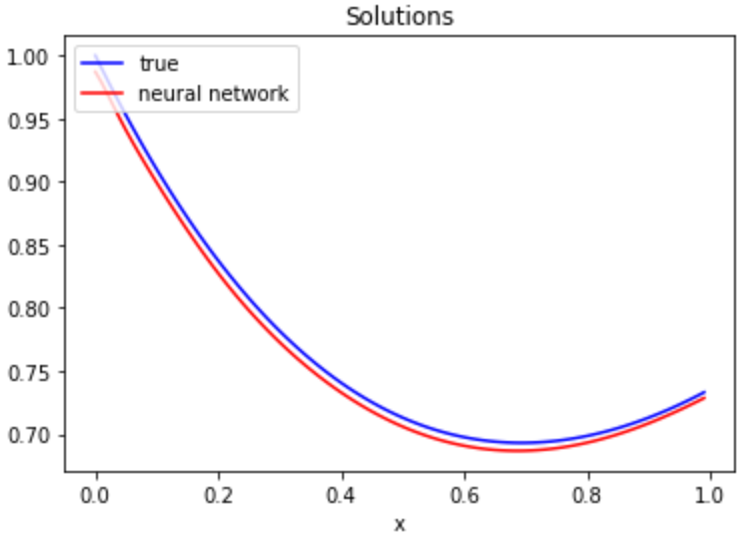
$$L(u) = \frac{1}{N}\sum_{i=1}^{n}(|u'(x_i) + u(x_i) - x_i|^2) + |u(0) - 1|^2$$

where u is the approximation for y and N represents the mesh size used to discretize the interval of interest [0,1].

Results:

Loss: 6.25e-05
Time: 192.21





Ex. 2: $-y''(x) + y(x) = \sin(4\pi x)\,,\ y(0) = y(1) = 0$

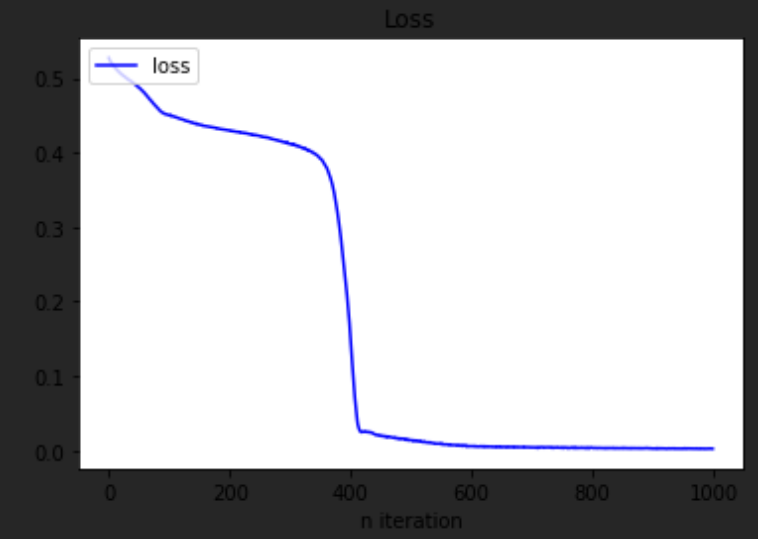
In this case, the loss function is

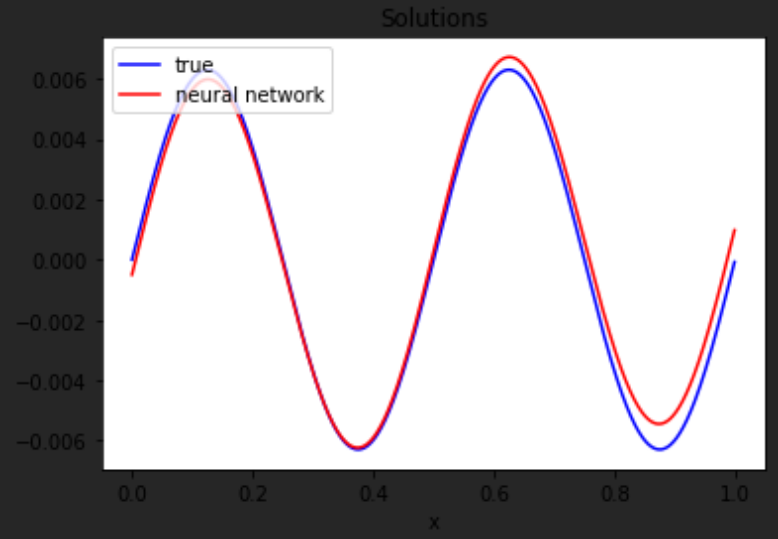
$$L(u) = \frac{1}{N}\sum_{i=1}^{n}(|-u''(x_i) + u(x_i) - \sin(4x_i\pi)|^2) + \frac{|u(0)|^2}{2} + \frac{|u(1)|^2}{2}$$

where the second derivative was computed via finite-difference method.

Results:

Loss: 2.63e-07





Ex. 3: $y'' + y = e^{-x}, y(0) = 1,$ $y(1) = 1/2(\cos(1) + \sin(1) + e^{-1})$

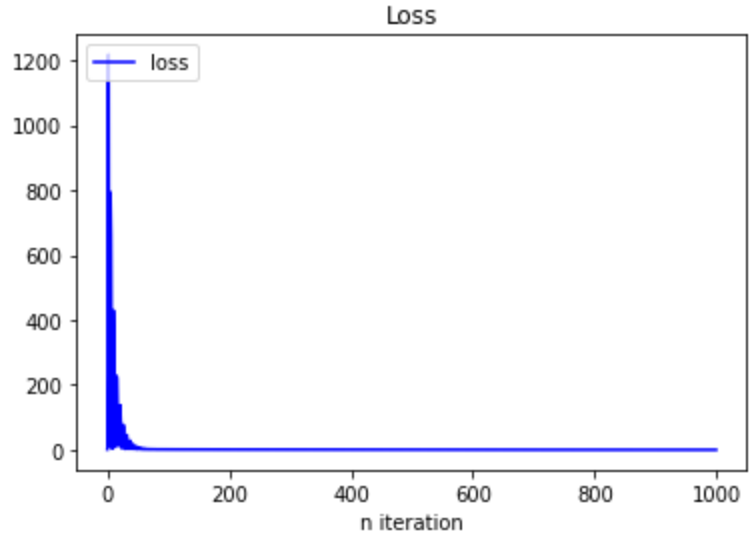
In this case, the loss function is

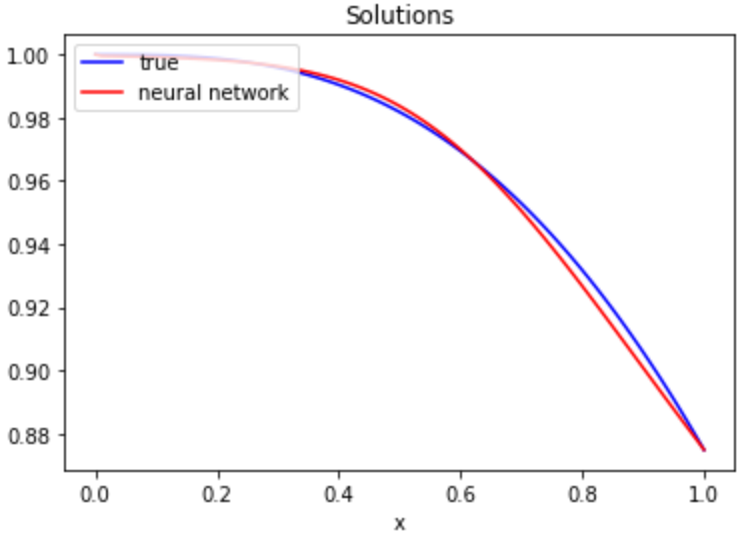
$$L(u) = \frac{1}{N}\sum_{i=1}^{n}(|u''(x_i) + u(x_i) - \mathrm{e}^{-\mathrm{x_i}}|^2) + \frac{|u(0)|^2}{2} + \frac{|u(1)|^2}{2}$$

Results:

Loss:

6.28e-06





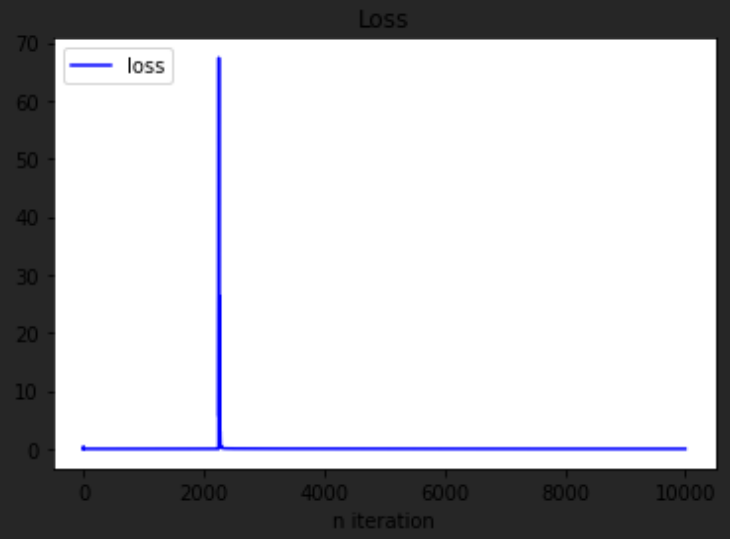
Ex. 4: $|y'| = 1, y(-1) = y(1) = 0.$

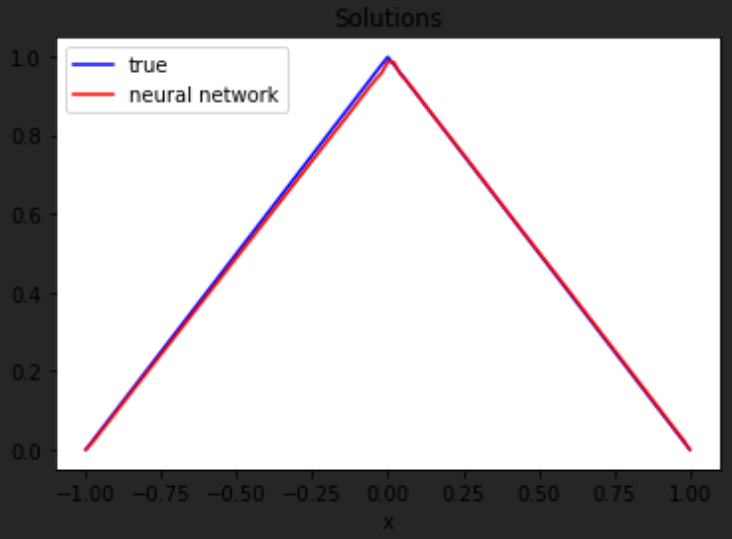
The loss function we used for this was not as one might expect – we introduced a small viscosity term $\varepsilon y''$ to give the neural network more wiggle room. So,

$$L(u) = \frac{1}{\mathrm{N}}\sum_{i=1}^{N}\big||u'(x_i)| - \varepsilon u''(x_i) - 1\big| + 1/2(u(0)^2 + u(1)^2)$$

Results:

Loss: 7.37e-05
Time: 432.71





Future directions:

01

Generalize to more classes of functions.

02

Generalize to PDEs.

03

Investigate the best network architecture for solving certain classes of functions.

04

Study results on convergence of the method.

Thank you!

Are there any questions?