# Numerical ODE approximator using sine activation gates

**Yash Ramani, Hans Shi**

## Abstract

Neural networks have proved to be a powerful tool in finding solutions to ordinary and partial differential equations. We propose to leverage periodic activation functions to approximate numerical solutions to lower as well as higher-order Ordinary Differential equations. Motivated by their success in representing images, wavefields, video, sound, and their derivatives, we modify sinusoidal representation networks (SIREN) (Sitzmann et al. 2020) to our particular task. We also experiment the application of the $\tanh$ and ReLU activations for the same, and demonstrate how $\sin$ activations learn solutions at least as good as the others.

## 1 Introduction

We shall use the sine activation function (along with other activation functions) to create a neural network to solve the initial value problem (IVP) defined as follows: given an open set $D \subset \mathbb{R}^n$, and a function $f : D \times \mathbb{R} \to \mathbb{R}^n$, we wish to find functions $x : [a, b] \to D$ such that

$$x'(t) = f(x(t), t) \tag{1}$$

subject to the condition $x(t_0) = x_0$ for $t_0 \in (a, b)$, $x_0 \in D$. More generally, we shall consider the $n^{\text{th}}$ order IVP:

$$\frac{\mathrm{d}^n x}{\mathrm{d}t^n} = f\left(x, \frac{\mathrm{d}x}{\mathrm{d}t}, \frac{\mathrm{d}^2 x}{\mathrm{d}t^2}, \ldots, \frac{\mathrm{d}^{n-1} x}{\mathrm{d}t^{n-1}}, t\right) \quad t \in (a, b)$$

with $n$ initial conditions in total on any of the higher order derivatives of $f$. [1]

## 2 Related Work

Initial attempts of solving ODEs and PDEs made use of multi-layered perceptrons (Malek and Shekari Beidokhti 2006), using the sigmoid and hyperbolic-tangent as activation functions (Chakraverty and Mall 2014). A recent approach employed Legendre Neural Network (LeNN) to solve initial value problems (Mall and Chakraverty 2016).

## 3 Method

Before describing our models, we shall describe how we shall use the output of our neural network $N(t)$ (here $t$ is time - the input to our network) in solving the IVP. For convenience, we consider the IVP $x' = f(x(t), t)$ described above. In this case, $N(t) \in \mathbb{R}^n$ for all $t \in [a, b]$. We propose a trial solution to the ODE: $x(t) = x_0 + (t - t_0)N(t)$. The motivation behind defining this as the trail solution is to ensure that the initial condition is met since now we already have $x(t_0) = x_0$.

[1] Please see appendix B for the contributions of each member, and visit https://github.com/hanss314/CSC413-Final-Project to see our code base.

As part of the training data, we're provided with samples $X = \{t_i \mid i = 1, \ldots, N\} \subset [a, b]$. We define our loss function as

$$\mathcal{L}(X) = \sum_{i=1}^{N} \|x'(t_i) - f(x(t_i), t_i)\|^2 \quad (2)$$

where as discussed above, $x(t) = x_0 + (t - t_0)N(t)$ and so $x'(t) = (t - t_0)N'(t) + N(t)$ by the chain rule. In practise, we train the network over multiple epochs and average over the above loss over the number of samples encountered in each epoch. We note that even for the more general $n^{\text{th}}$ order ODE, we can similarly cleverly define our trial solutions $x(t)$ in terms of $N(t)$ so that initial condition are satisfied.

We obtain $N'(t)$, and more generally, $N^{(k)}(t)$ for any $k \geq 1$, by successively using `torch.jacrev` which allows us to take the derivative of any function of the inputs of a neural network w.r.t. the inputs.

Now, we describe our models, beginning first with SIREN. SIREN is a neural network all of whose layers are SineLayer except the last one which is linear. Each SineLayer has a hyperparameter $\omega \in \mathbb{R}$ associated with it. Given that the input to a SineLayer is $\mathbf{x}$, it outputs

$$\sin(\omega \cdot \mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ is the bias term. We set $\omega$ to be the same for all layers except the first SineLayer (i.e. the first layer of SIREN). For the first SineLayer, we initialize the weights uniformly in the interval $[-\frac{1}{n}, \frac{1}{n}]$ where $n$ is the size of the layer. For the subsequent hidden layers, we initialise the weights uniformly in the interval $[-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}]$. The motivation behind the hyperparameter $\omega$ and the initialization of weights can be found at Sitzmann et al. 2020

We compare SIREN with two networks: one with ReLU as the activation and another with $\tanh$ as the choice of activation function. Each of the hidden layers in each of these two networks outputs:

$$f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\mathbf{x}$ is the input to the layer, and $f$ is the respective activation function. As before, in each of these two networks, the output layer in linear.

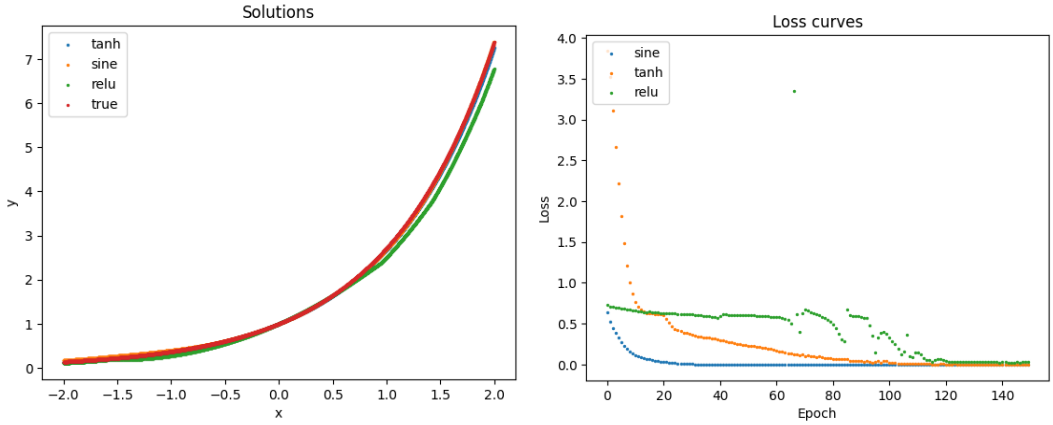
To compare the performance of the models (and against each other), we plot $x(t)$ for each of them against the actual solution to the ODE; this works if $x(t)$ is a scalar. If instead it is a vector, we plot $x_1(t)$, where $x_1(t)$ is the first component of $x(t)$. To create these plots, we sample a large number of times $t$ uniformly in $[a, b]$.

## 4 Results

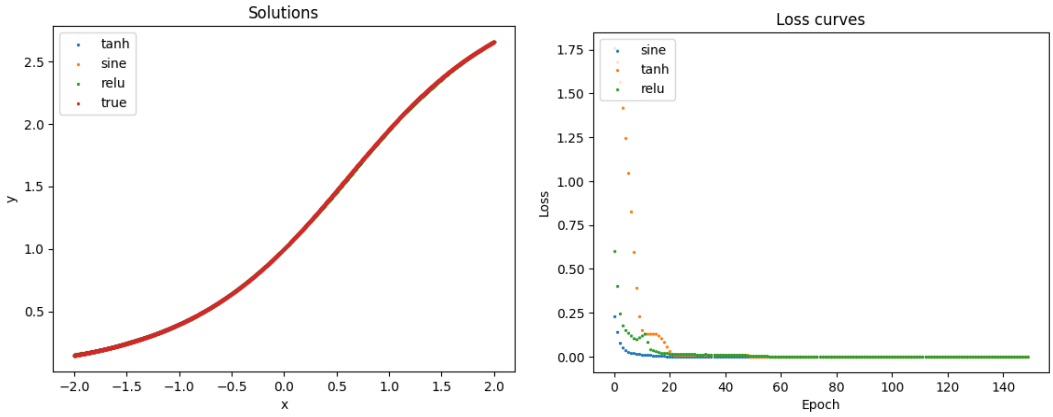
### 4.1 Comparison against other activation functions

In all our experiments, each of the three neural networks had 10 hidden layers each of size 10; we use the RAdam optimizer and have a 90-10 split of the set `np.arange(-2, 2, 0.001)` between the training sets and validation sets. For each ODE considered below, we include the validation loss curves and the solutions learnt from each of the three models. We also plot the actual solution of the ODE for comparison. The hyperparameters used, the proposed trial solution in terms of $N(t)$, as well as the actual solution to each ODE considered, are included in Appendix A.

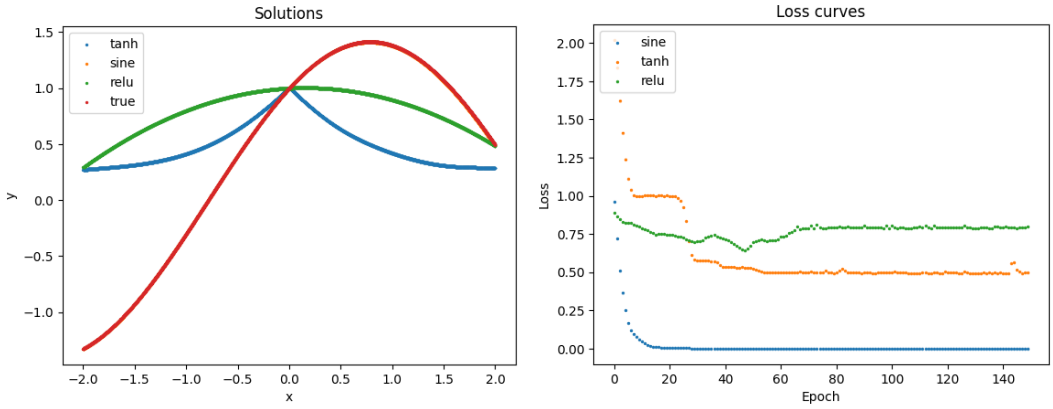
#### 4.1.1 $x' = x$ with $x(0) = 1$



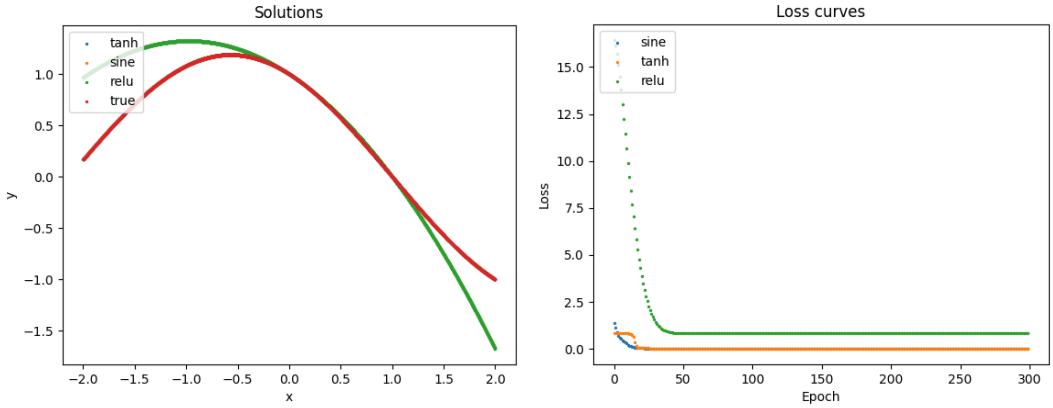

#### 4.1.2 $x' = \sin(x)$ with $x(0) = 1$




#### 4.1.3 $x' = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} x$ with $x(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$



#### 4.1.4 $x'' = -x$ with $x(0) = 1$ and $x(1) = 0$

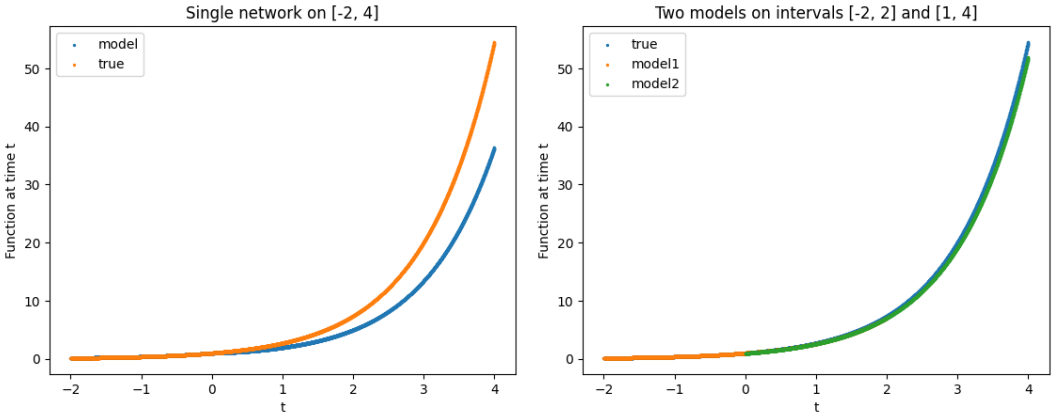



## 5 Discussion

We see that in all the ODEs considered above, the model with the $\sin$ activation performs the best even though in some cases, $\tanh$ comes close. The improvement in using $\sin$ activation is especially pronounced in the third ODE considered. However, we do note that the $\sin$ activation doesn't always work well, as discussed below.

### 5.1 Limitations of large intervals

SIREN fails to converge on larger intervals of the ODE $x' = x$ (with $x(0) = 1$) due to the limitations of network size, and also because the actual solution $x(t) = e^t$ explodes with time. Fitting to a larger interval requires an exponentially larger network size, which can be impractical. However, an easy way to grow the coverage linearly is to train an initial network $N_0$ on a small interval. $N_0$ is then capable of producing an initial value near the right boundary of its interval, which can be used to train another network on an overlapping interval. This can be repeated to cover as large of an interval as desired. We demonstrate this for the ODE $x' = x, x(0) = 1$ below:




## 6 Conclusion

We demonstrated how $\sin$ as an activation function allows us to create an ODE solver. In particular, we showed an improvement in performance when using $\sin$ as compared to other commonly used activation functions. Finally, we demonstrated one limitation of using the $\sin$ activation, and one possible fix to this problem.

## References


Chakraverty, S. and Susmita Mall (Sept. 2014). "Regression-based weight generation algorithm in neural network for solution of initial and boundary value problems". In: *Neural Computing and Applications* 25.3, pp. 585–594. ISSN: 1433-3058. DOI: 10.1007/s00521-013-1526-4.

Malek, A. and R. Shekari Beidokhti (2006). "Numerical solution for high order differential equations using a hybrid neural network—Optimization method". In: *Applied Mathematics and Computation* 183.1, pp. 260–271. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2006.05.068.

Mall, Susmita and S. Chakraverty (2016). "Application of Legendre Neural Network for solving ordinary differential equations". In: *Applied Soft Computing* 43, pp. 347–356. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2015.10.069.

Sitzmann, Vincent et al. (2020). *Implicit Neural Representations with Periodic Activation Functions*. arXiv: 2006.09661 [cs.CV].


## Appendix

**A.** For each of the ODEs considered in the Results, we describe the hyperparameters used, the proposed trial solution in terms of $N(t)$, as well as the actual solution to each ODE considered, are included in the appendix.

**A.1.** $x' = x$ with $x(0) = 1$

We set the number of epochs to 150 and batch size to 200. We set $\omega_f = 2$ and $\omega_h = 1$ where $\omega_f$ and $\omega_h$ are the value of $\omega$ for the first and hidden layers respectively. The trial solution is defined to be

$$x(t) = 1 + tN(t)$$

where $N(t)$ is the output of the network; note that this ensures $x(0) = 1$. The actual solution to this ODE is

$$x(t) = e^t$$

**A.2.** $x' = \sin(x)$ with $x(0) = 1$

We set the number of epochs to 150 and batch size to 200. We set $\omega_f = 1$ and $\omega_h = 1$ where $\omega_f$ and $\omega_h$ are the value of $\omega$ for the first and hidden layers respectively. The trial solution is defined to be

$$x(t) = 1 + tN(t)$$

where $N(t)$ is the output of the network; note that this ensures $x(0) = 1$. The actual solution to this ODE is

$$x(t) = 2\text{arctan}\left(\tan\left(\frac{1}{2}\right)e^t\right)$$

**A.3.** $x' = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} x$ with $x(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

We set the number of epochs to 150 and batch size to 200. We set $\omega_f = 2$ and $\omega_h = 1$ where $\omega_f$ and $\omega_h$ are the value of $\omega$ for the first and hidden layers respectively. The trial solution is defined to be

$$x(t) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + tN(t)$$

where $N(t)$ is the output of the network; note that this ensures $x(0) = 1$. As remarked earlier, for each of the models, we only just plot $x_1(t)$ where $x(t)$ is the trial solution. The actual solution to this ODE is

$$x(t) = \begin{pmatrix} \cos(t) + \sin(t) \\ \cos(t) - \sin(t) \end{pmatrix}$$

**A.4** $x'' = -x$ with $x(0) = 1$ and $x(1) = 0$

We set the number of epochs to 350 and batch size to 200. We set $\omega_f = 1$ and $\omega_h = 1$ where $\omega_f$ and $\omega_h$ are the value of $\omega$ for the first and hidden layers respectively. The trial solution is defined to be

$$x(t) = 1 - t + t(t-1)N(t)$$

where $N(t)$ is the output of the network; note that this ensures $x(0) = 1$ and $x(1) = 0$. The actual solution to this ODE is

$$x(t) = \cos(t) - \cot(1)\sin(t)$$

Note that unlike in the above scenarios, our loss function is a bit different from the loss defined in (2). It is defined as below:

$$\mathcal{L}(X) = \sum_{i=1}^{N} \|x''(t_i) - f(x(t_i), t_i)\|^2$$

where $f(x, t) = -x$.

**B.** The link to our code base and individual member contributions can be found below

**B.1** Project github: https://github.com/hanss314/CSC413-Final-Project

**B.2 Contributions**

Yash wrote the code for the SIREN model and the training loops. He also wrote the all sections of the report except the Discussions section which was written by Hans.

Hans contributed to debugging of the network Yash initially wrote, to get it actually working, and prototyping some attempted solutions not featured in the writeup. He also wrote the comparisons and graphs for them.