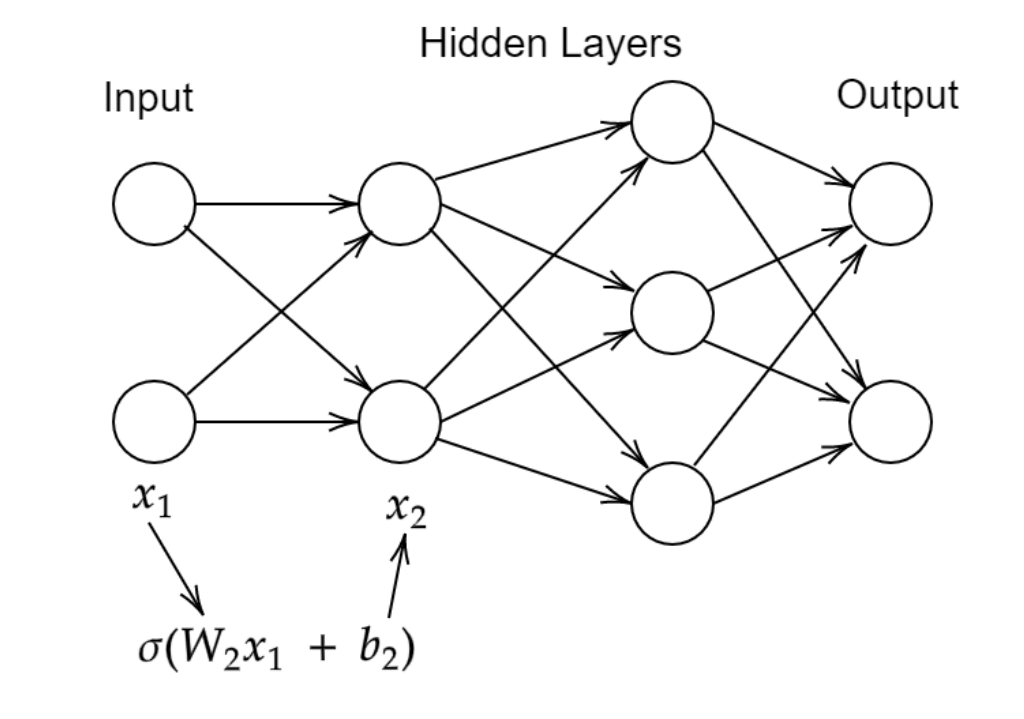
Physics Informed Neural Networks for Numerical Analysis

Robin Pfeiffer

Supervised by Niall Madden

June 2022



Overview of Presentation

- Introduction
- Activation Function
- Setup of Neural Network
- Stochastic Gradient Descent
- Classification Problem
- Interpolation Problems
- Physics Informed Problems
- What I have Learned and What I would do next

Introduction

In its simplest form, a **neural network** is just a function with some parameters that can be chosen.

Given some '**training data**', i.e., some inputs and their desired output, we want to choose these parameters so that this function fits the training data.

Evaluating the function at new points then gives **predicted** output.

The networks we use follow the most standard format:

- The network has a number of **layers,** which have different possible numbers of inputs and outputs.
- Each layer has number of ***"neurons"*** per layer; each "neuron" is an *activation function,* which evaluates as zero or one for most inputs, with a rapid transition between values. That is, “fires” for some inputs, and not for others.

Each layer can be encoded by a **weight matrix** and **bias vector:** "training" the network means choosing values for the entries in these matrices and vectors.

Activation Function

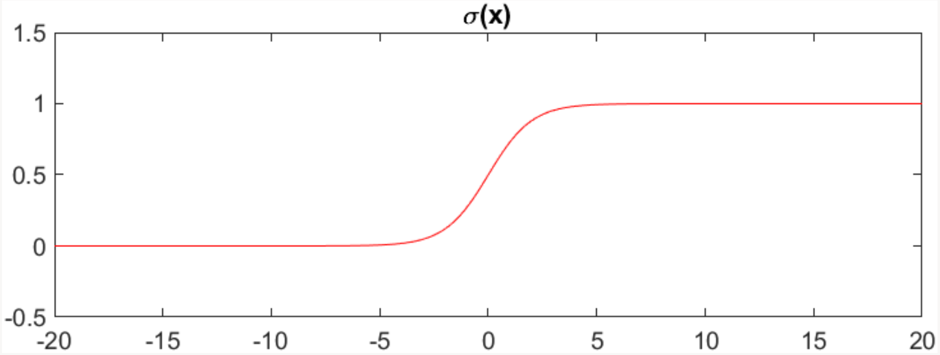
The artificial neural network approach uses repeated application of a simple, nonlinear function. In this case, we chose to base our neural network on the **sigmoid function**.

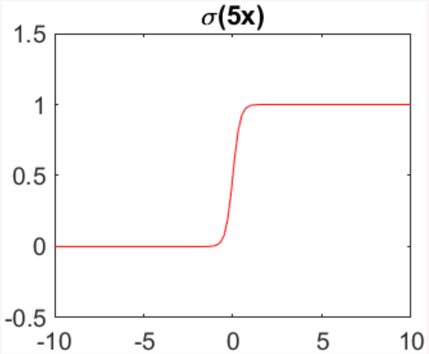
$$\sigma(x) = \frac{1}{1-e^{-x}}$$

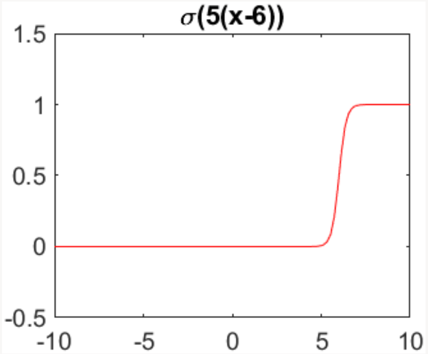
This can be scaled and shifted to control

- how quickly it increases from zero to one,
- at what point it does so.

However, by chaining multiple of these together, we can create some more complex functions, as will be shown in later sections.







Activation Function

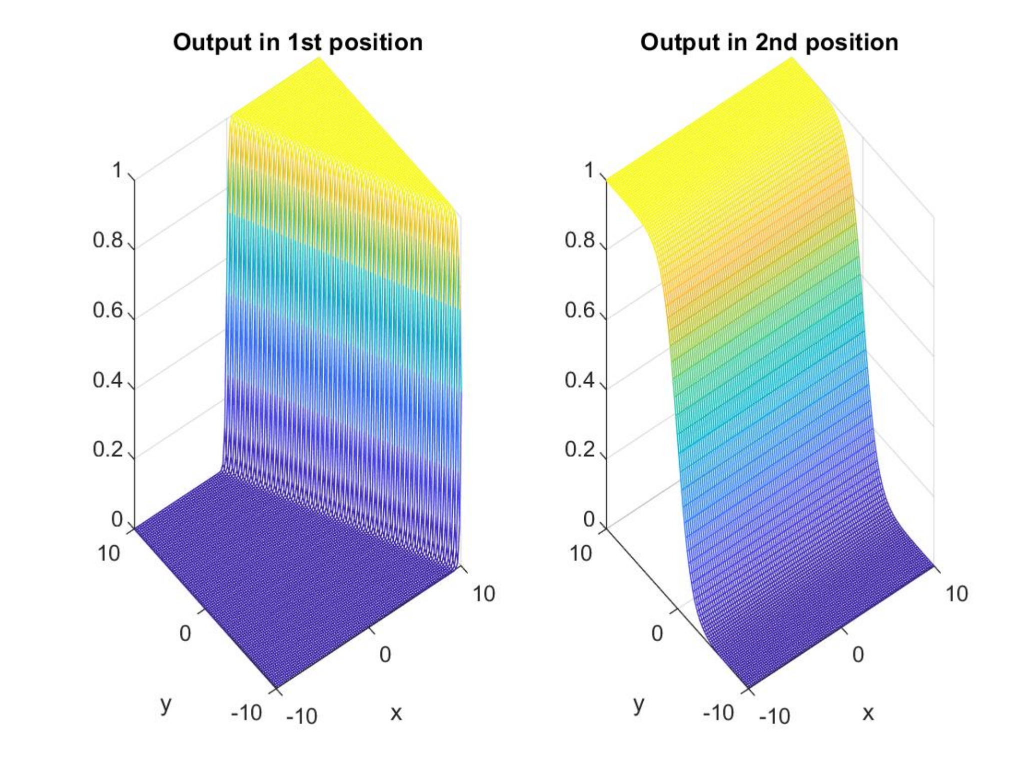
It is also possible to have a vector based sigmoid function. To map from *m* inputs to *n* outputs, the activation function is

$$\sigma = \sigma(Wx + b)$$

where W is an *m* x *n* matrix, and b is a n-vector.

In the following example, I am mapping 2 inputs to 2 outputs.

$$\sigma(Wx+b), \quad W = \begin{pmatrix} 10 & 5 \\ 0 & 1 \end{pmatrix}, b = \begin{pmatrix} -50 \\ 0 \end{pmatrix}$$



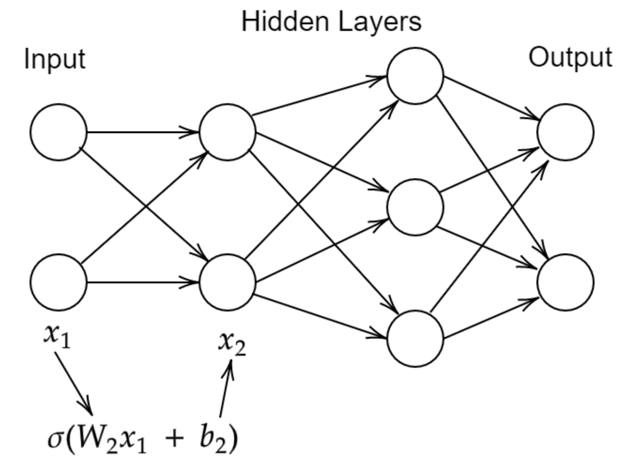
Setup of Neural Network

A neural network is built on multiple layers, with multiple nodes in each layer. In order to go from one layer to another, we first multiply it by a matrix of weights W and add some bias b, and then run it through our activation function.

We now have a function that symbolises our neural network. We let X be the input of our training data, and Y the required output. We can now create a measure for how close our network is to the required solution. We let the cost function be

$$Cost = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left|\left|y(x^i) - f(x^i)\right|\right|^2$$

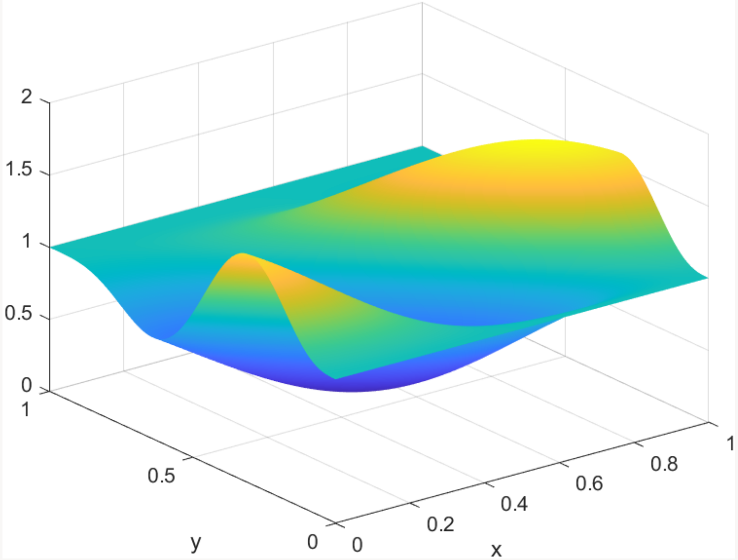
We now need to minimise this cost function, by changing the weights and biases in our network.

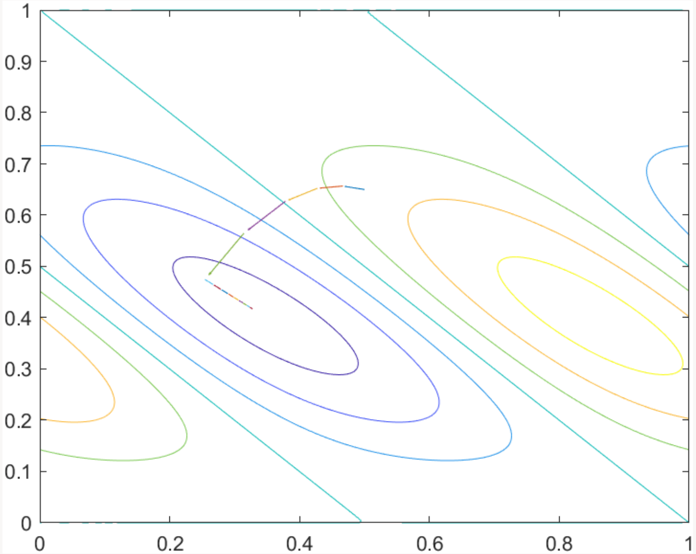


Stochastic Gradient Descent

We use gradient descent to minimise our cost function. The basic idea of gradient descent is, if $x_{(i-1)}$ is an estimate for x_{min}, then $x_{(i)} = x_{(i-1)} - \eta \nabla f$ should be a better one.

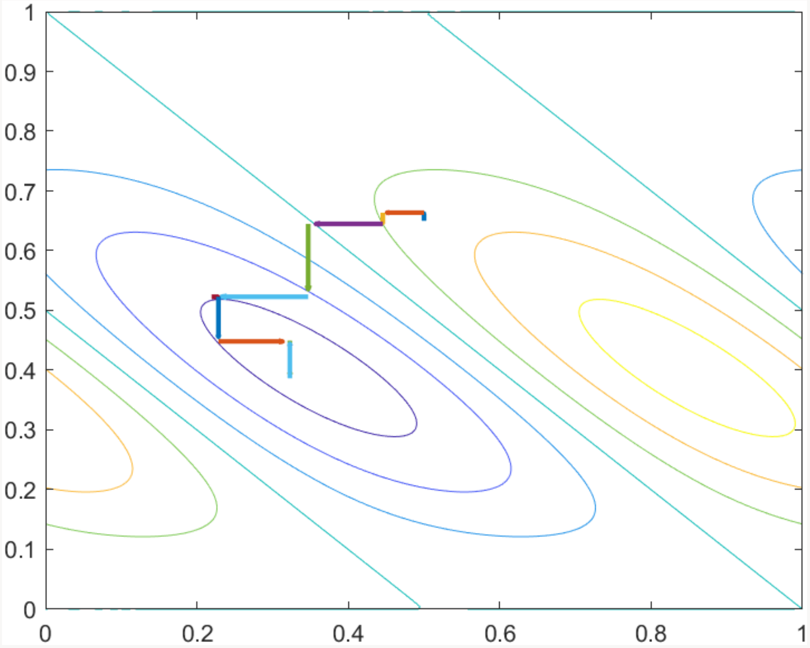
We can see this in action in the following example.





Stochastic Gradient Descent

Calculating the total derivative can be quite expensive, so in stochastic gradient descent we only look at one variable at a time.



Stochastic Gradient Descent

Using the fact that

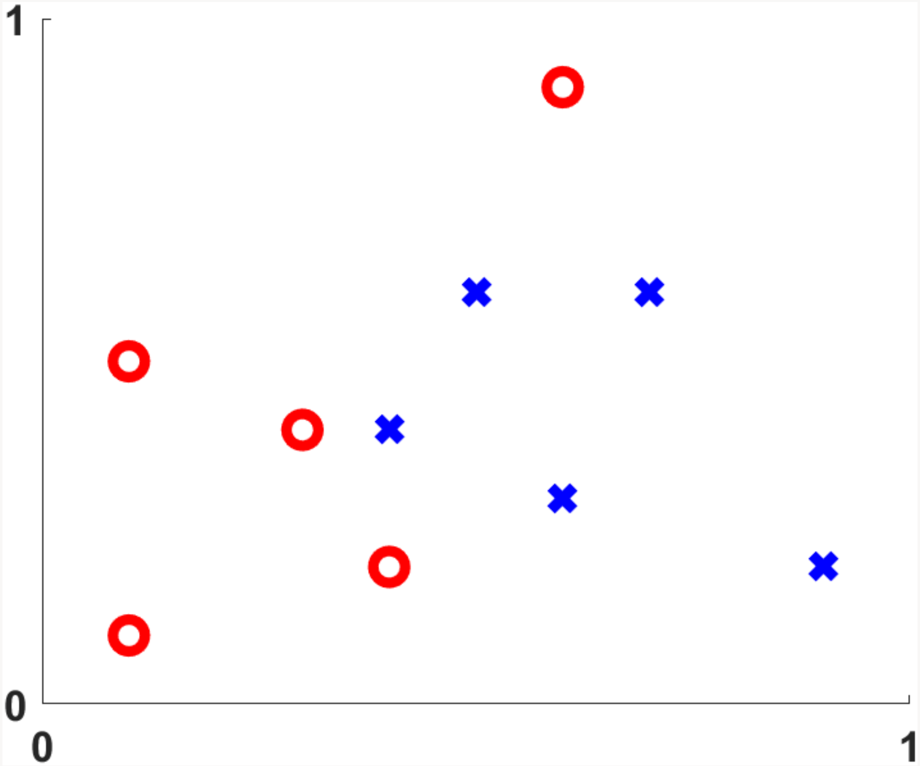
$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

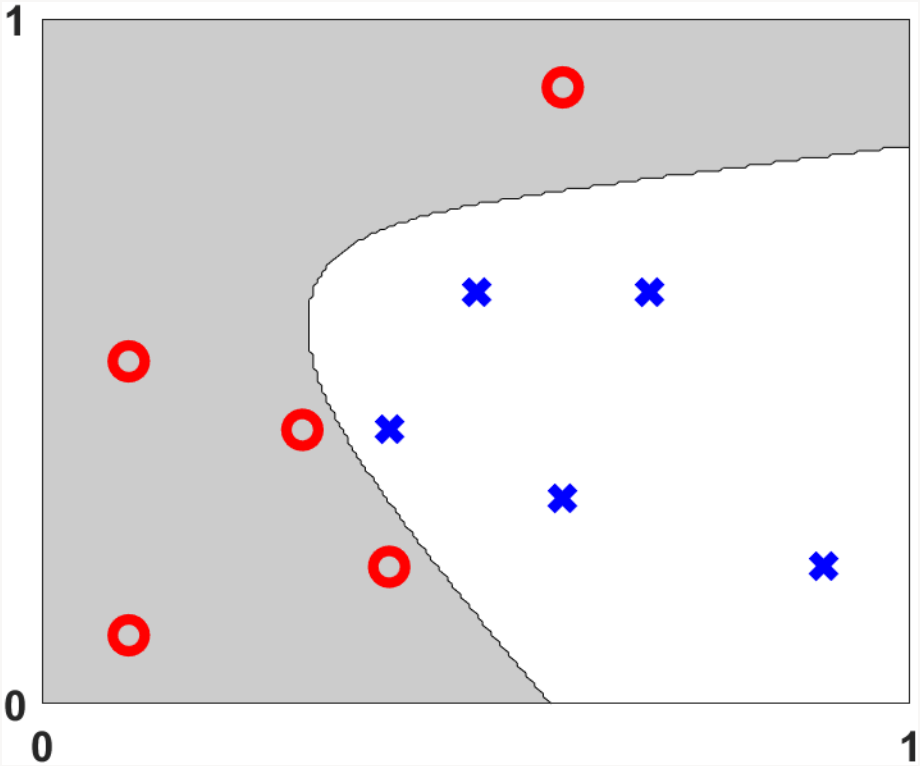
And through back propagation, we can find the derivative δ_i for each layer.

Extending this to our cost function, with p total variables and N layers, our method becomes

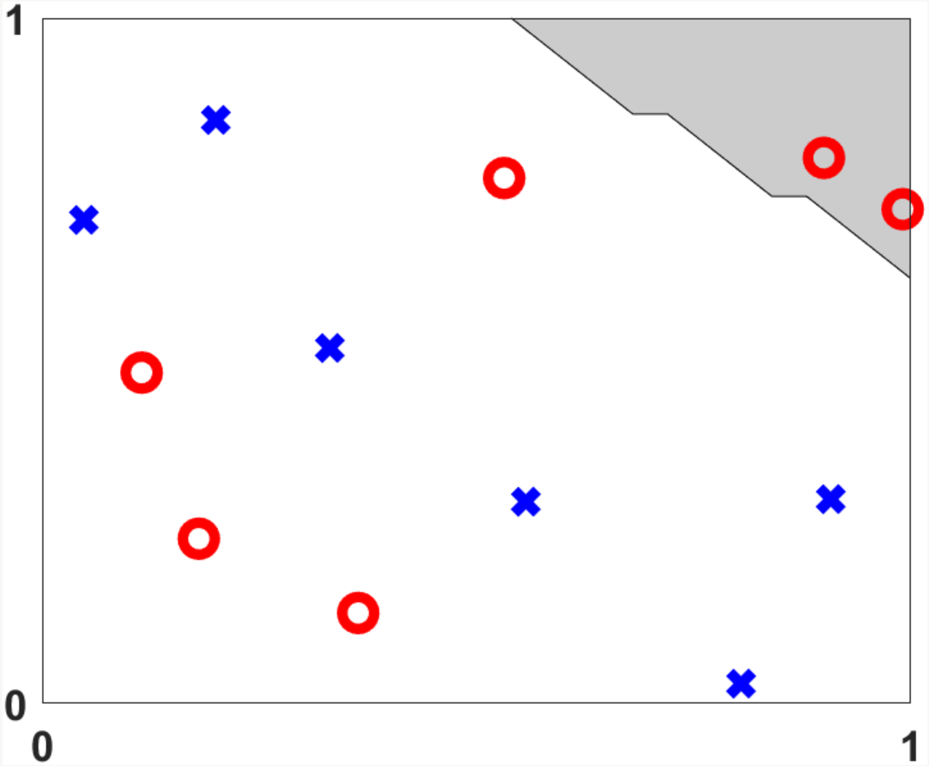
1. Choose an integer k between 1 and p
2. Let $a_1 = k$
3. $a_{i+1} = \sigma(W_{i+1} \cdot a_i + b_{i+1})$, for $i = 2, 3, \ldots, N$
4. $\delta_N = a_N(1 - a_N)(a_N - Y(k))$
5. $\delta_i = a_i(1 - a_i)(W_{i+1}\delta_{i+1})$ for $i = N - 1, N - 2, \ldots, 2$
6. $W_i = W_i - \eta \cdot \delta_i \cdot a_i$
7. $b_i = b_i - \eta \cdot \delta_i$

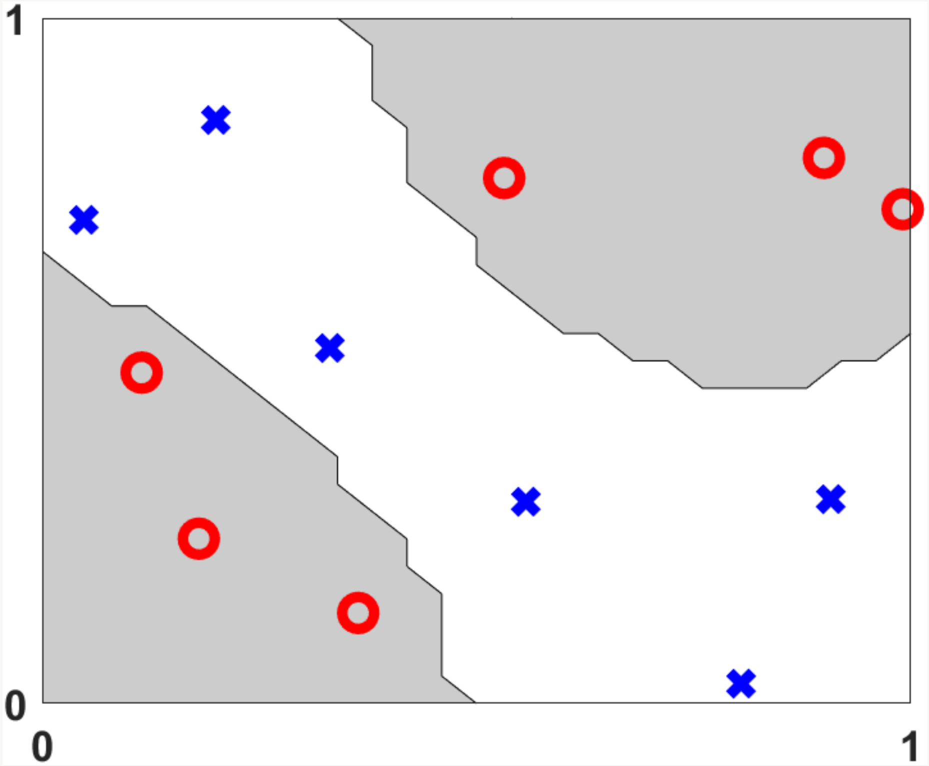
Classification Problem





Classification Problem

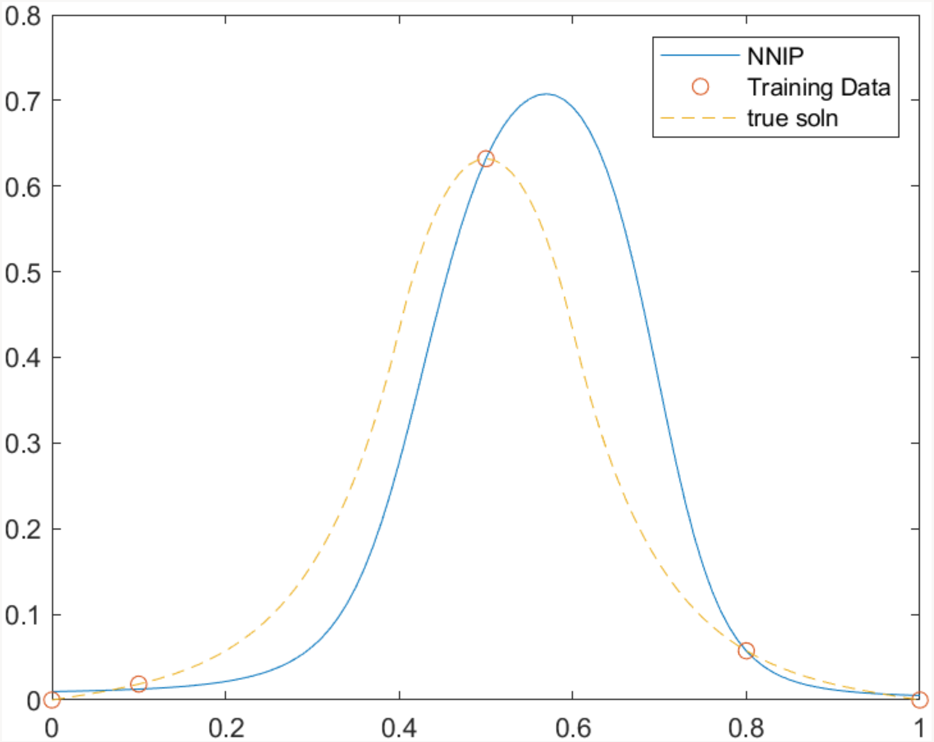




Interpolation Problems

In numerical analysis, the process of interpolation means to find a function that agrees with a data set. We can use the same setup of a neural network to interpolate a data set.

x	0	0.1	0.5	0.8	1
$y(x)$	0	0.018	0.632	0.057	0



Physics Informed Neural Network

PINNs are used in a similar way to solve differential equations, with a crucial difference in using the residual in the cost function.

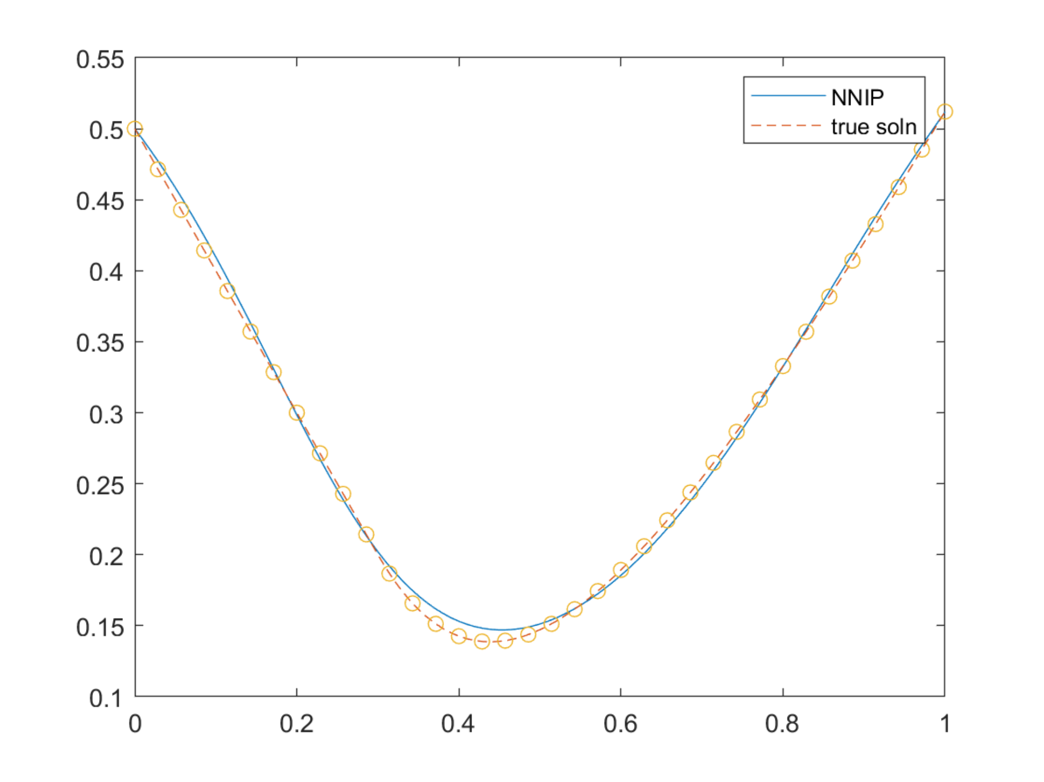
Given a differential equation in the form $D_E(u(x), x) = f(x)$ and some initial condition, our cost function is

$$Cost = [\,|D_E - f(x)|, |u(0) - 1|].$$

Now we have just another optimisation function.

In the following example, our differential equation is

$$0.3\frac{du}{dx} + u(x) = |x - 0.3|,\ u(0) = 0.5$$



What I have Learned and What I would do next

- Much of what I've learned has been based on Higham + Higham (2018), PINNs were covered in Raissi (2019)
- All code is in MATLAB, and uses the chebfun toolbox

If I had more time:

- More general ODEs (boundary value problems; coupled systems; non-linear problems; noisy data);
- Develop code in Python (for example) so it can be shared more easily.
- The ODE example uses a built-in non-linear solver; work on Gradient Descent method.
- Need to experiment with the number of layers and neurons.

References

- Higham, C. F. and Higham, D. J. (2019) Deep learning: an introduction for applied mathematicians. SIAM Review, 61(4), pp. 860-891. (doi: 10.1137/18M1165748)
- Raissi, P. Perdikaris, G.E. Karniadakis, (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comp Phys 37, pp686-707. (doi: 10.1016/j.jcp.2018.10.045).