Session 1

Jamie M. Taylor (CUNEF Universidad, Madrid, Spain)

jamie.taylor@cunef.edu

BCAM - In-Deep workshop

Aims and outline

- The use of NNs to solve problems from PDEs is new, and full of challenges.
- There is no “magic bullet” - each problem requires its own solutions.
- The aim is to give an overview of common methods, common issues and potential solutions in sufficient generality that you can adapt the ideas to your work.
- We lack a complete theory - How to design your problem is often based on intuition and trial and error.
- The solutions we discuss will not apply to everything, and the simulations I show will be far from optimised
- Perhaps leaving the simulation longer or changing training parameters could lead to other results, but this is the nature of NNs - there are many moving parts and the interactions between them are rarely clear.

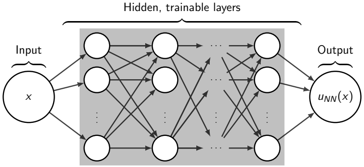
Why use Neural Networks?

- When solving a single, low-dimensional PDE, classical methods (Finite Elements, Finite Differences, etc.) will generally “win” - Solutions can be obtained faster, to higher accuracy, with better control and understanding of errors.
- The principal advantage of Neural Networks is their scalability - In high-dimensional problems, they can describe solutions without suffering the *curse of dimensionality* (exponential increase in necessary degrees of freedom as the dimension increases).
- These may be PDEs in high-dimensional spaces (E.g. Schrödinger, Fokker-Plank, Black-Scholes equations in large systems).
- The problem may be high-dimensional due to a latent parameter space (Parametric PDEs, **Inverse problems**).

Training a Neural Network

- Training a Neural Network for any purpose is ultimately about approximating functions (interpolation of data, solutions of a PDE, operators between spaces...)
- The process admits three general steps:

1) An architecture is chosen (e.g. fully-connected feed-forward Neural Network)



2) A loss function is chosen (e.g. PINNs)

$$\mathcal{L}(u_{NN}) = \int_\Omega |\Delta u_{NN}(x) - f(x)|^2 \, dx + \int_\Gamma |u_{NN}|^2 \, dx$$

3) An optimisation strategy is employed (e.g. SGD)

$$\theta_{i+1} = \theta_i - \delta \frac{\partial \mathcal{L}}{\partial \theta}$$

Toy problem

- To begin, let us consider a simple, 1D problem - Find $u : [0, 1] \to \mathbb{R}$ such that

$$u''(x) = f(x),$$

  satisfying the boundary conditions $u(0) = u(1) = 0$.
- Whilst this is a very simple problem, it suffices to highlight many issues that arise when solving PDEs using Neural Networks.
- If one wishes to solve inverse problems, or more complicated problems involving PDEs, it is essential that one can solve a single ODE.

Architectures

Architectures

Loss functions

The architecture of a Fully-Connected Feed-Forward Neural Network

- We define a sequence of *layers*, functions $L_i : \mathbb{R}^{N_i} \to \mathbb{R}^{N_{i+1}}$, given by

$$L_i(x) = \sigma_i(A_i x + b_i).$$

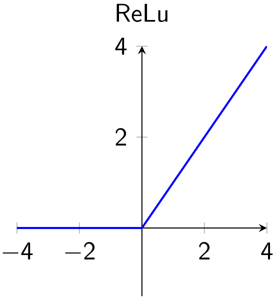
- The functions $\sigma_i : \mathbb{R} \to \mathbb{R}$ act componentwise and are chosen by the user. The matrices $A_i \in \mathbb{R}^{N_{i+1} \times N_i}$ and vectors $b_i \in \mathbb{R}^{N_{i+1}}$ are the *weights* and *biases*, and correspond to the trainable parameters.
- The Neural Network is then given as

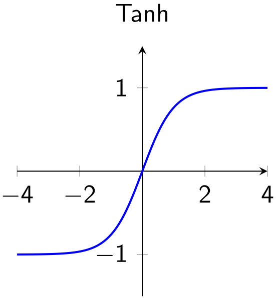
$$u(x) = L_M \circ L_{M-1} \circ ...L_2 \circ L_1 \circ (x).$$

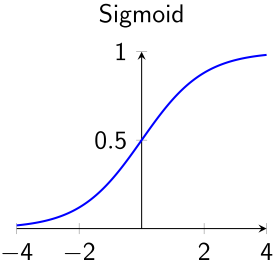
- The last activation function is taken as the identity, $\sigma_M(x) = x$.
- Provided the other σ_i are not polynomials, a sufficiently large network can approximate any continuous function on a compact subset of $\mathbb{R}^d$ uniformly.

Activation functions

- Commonly used activation functions include tanh, sigmoid, relu, elu.
- The regularity of the NN corresponds to that of the activation function - For example, if $ReLU$ is used, the network will define a continuous function, but it will not be differentiable.
- In this course, as we are thinking of PDEs, I will avoid ReLu, as it is not smooth enough for our problems.





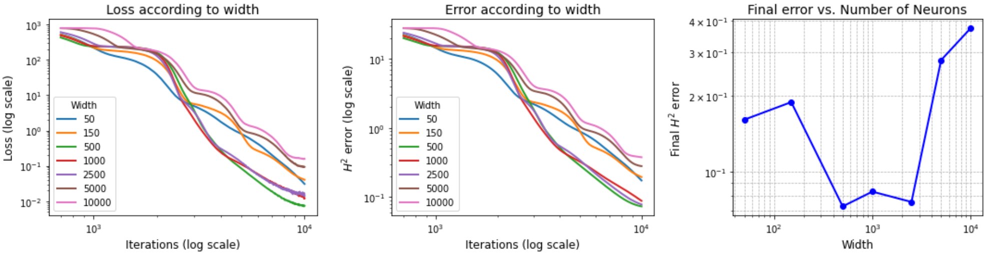


The effect of size

- The “size” of a fully-connected, feedforward neural network has two aspects - the depth, which is the number of layers; and the width, which is the number of neurons per layer (i.e., the dimensions N_i)
- Variants of the Universal Approximation Theorem guarantee that a sufficiently wide or sufficiently deep neural network can approximate any continuous function.
- The mathematical *existence* of such a network does not imply that we can easily find it.
- There are theoretical results that state that there *exist* neural networks that can solve certain problems, but there is *no* appropriate algorithm that can actually find it (Colbrook, Antun and Hansen PNAS).
- Let us consider an empirical study on size.

The effect of width

- We consider a 1D ODE with PINNs loss.
- Our NN has a single hidden layer, and we consider how the training is affected by the width of the NN.



Depth and the problem of vanishing gradients

- In a Deep Neural Network (many layers), the gradients of the loss with respect to the variables of the initial layers are often very small.
- This arises with sigmoid-type activation functions, the gradients are very small for large values, and the effect accumulates over the composition of layers. The gradients correspond to products of many terms (chain rule), so if each is small, the gradient is very small.
- This leads to very small updates in a gradient-based optimiser, even if large ones are needed to make small solutions.
- This is *the problem of vanishing gradients*
- One method to overcome this issue is incorporate the weights into shallower layers as well.

Residual Neural Networks (ResNets)

- In a fully connected, feed-forward neural network, the layers have the form

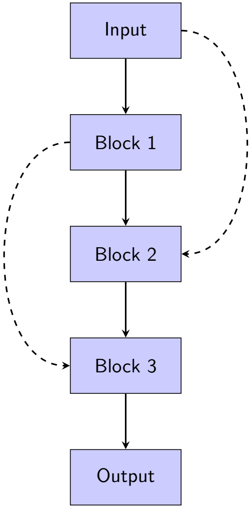
$$L_k(x) = \sigma_k(A_k x + b_k).$$

- In a ResNet, layers can be taken of the form

$$L_k(x) = \sigma_k(A_k x + b_k) + P_k x,$$

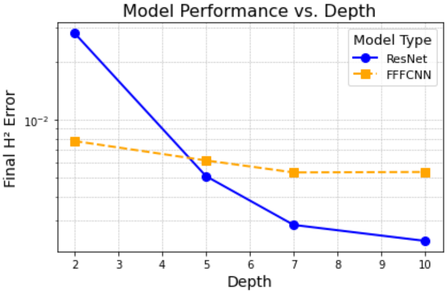
  with P_k a trainable matrix of appropriate dimensions. If the input and output dimension of the layer are equal, we can take $P_k = I$.

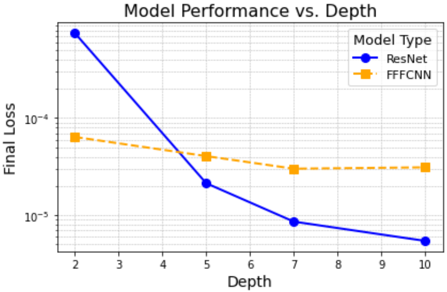
- The idea is that if σ_k "reduces" the information carried by x, the extra term can "preserve" it.
- "Connections" between more separated, rather than sequential, layers are possible.



ResNet example

- We consider the same 1D PINN example as before, and consider the final errors and losses.





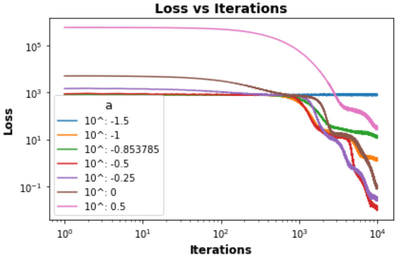
Initialisations

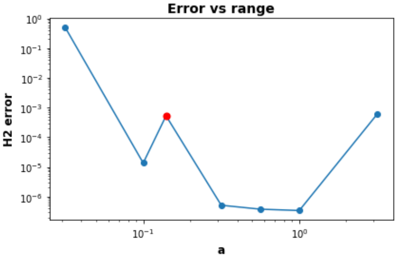
- Training methods for NNs are *iterative*, and loss functions often have many local minima. This means that they can be highly influenced by the initial chosen values of the trainable parameters.
- A well- or poorly-chosen initialisation can mean the difference between converging in far fewer iterations or getting trapped in an undesirable local minimum.
- Typically, one takes the parameters randomly according to some distribution.

Glorot initialisation

- The default initialisation in Tensorflow is the Glorot uniform initialisation - it takes the biases to be zero $b_k = 0$, and A_k to be taken from a uniform distribution on $(-a, a)$ with $a = \sqrt{\frac{6}{N_k + N_{k+1}}}$.
- The choice arises from the argument that if a tanh activation function is used and the inputs of the layers are treated as random variables, then this choice "maintains" the variance of the derivative (wrt. trainable parameters) across layers - they do not explode nor decrease.
- Whilst widely used, when the loss function contains derivatives of the NN, as in our problems, it seems unclear (to me) if this argument still holds.

Experiments with initialisations





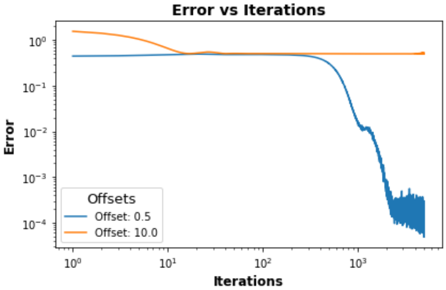
- We take a 1D PINN example, taking the kernel to be initialised from a uniform distribution in $[-a, a]$.
- A large a gives a poor initialisation, a small a barely moves during training.
- The Glorot initialisation $(10^{-0.85})$ does not appear to be optimal.

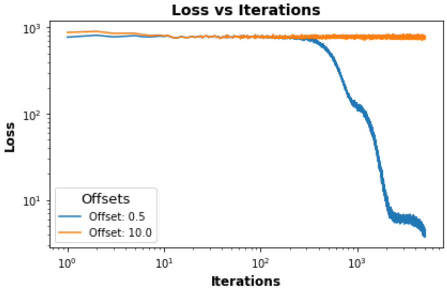
Initialisations

- Most activation functions have their “features” at zero - tanh,sigmoid and ReLu have their “transitions” at zero.
- If a layer is of the form $\sigma(A_k x + b_k)$, then the most “features” of the network will be where the components of $A_k x + b_k \approx 0$.
- If b_k is initialised as 0, then all the features are localised at 0 (or, more accurately, hyperplanes that pass through 0).
- By “offsetting” the Neural Network, we retain an equivalent parametrisation, but the initial “features” are removed from the domain.
$$u_{NN}(x) = \tilde{u}_{NN}(x - x_0)$$

Initialisations

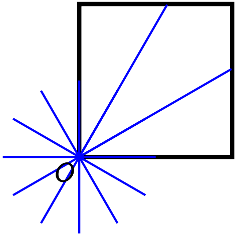
- I consider the same 1D PINN problem, $u''(x) = f(x)$ with zero BCs, on $(-0.5, 0.5)$ (offset=0.5) and on $(-10, -9)$ (offset=10).
- The solutions are the same, up to translations, and the NN is just as capable (in principle) of solving each problem.
- As the biases are initialised at zero, all the “features” are at 0 in each case.

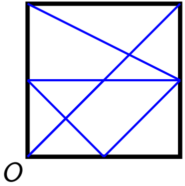




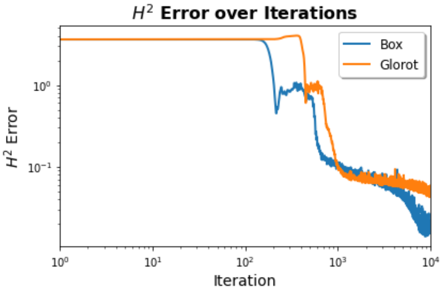
Box initialisation

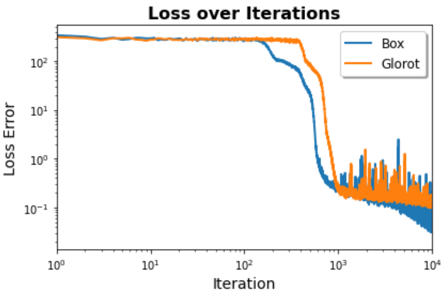
- Cyr et. al. proposed the idea of a “box initialisation” - The idea is that the “features”, understood as hyperplanes where components of $A_k x + b_k = 0$, should be well-distributed across the domain of interest.
- The idea is that each component of $A_k x + b_k$ can be written as $\beta_k(x - p_k) \cdot n_k$ for scalar β_k, a vector p_k and unit vector n_k. Taking p_k and n_k at random and a “good” choice of β_k gives more desirable initialisation.
- Their work was based on ReLus - I will follow the spirit of their idea, but not their exact method.





Example - 2D PINN

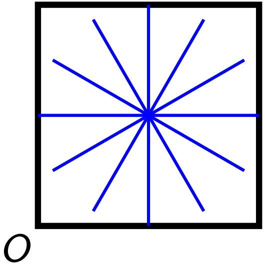




- This example is not the exact method of Cyr, but qualitatively similar, and only on the first layer.
- We see not only a faster convergence at the start of training, but even towards the end the results appear to be improved.

Comment

- The precise Box Initialisation involves an auxiliary optimisation problem, and becomes more complex in deeper networks. It also becomes more complex in more complex geometries.
- A very simple method that I find to help, in the same spirit, is simply to "offset" the neural network.
- If Ω is the domain of interest, take some point $x_0 \in \Omega$ that is "central" (e.g., centre of mass), and rescale the input as $u_{NN}(x) = \tilde{u}_{NN}(x - x_0)$, where $\tilde{u}_{NN}$ is a classical NN initialised with zero bias.



Some concluding remarks on architectures

- Bigger isn't always better - Whilst, in principle, a bigger network has a better approximation capability, in practice, it can make training expensive, slow, or introduce many unwanted local minima.
- Small, simple changes to the architecture can introduce significant gains in training.
- A "good" initialisation can mean the difference between reaching a good solution quickly, or not at all.

Loss functions

Architectures

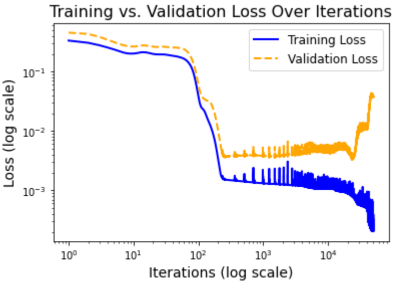
Loss functions

Fixed integration rules/Finite data

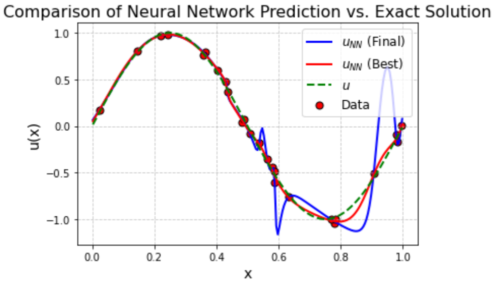
- (Almost?) every loss function related to PDEs is described in the continuum via integrals.
- There are no exact rules to integrate neural networks, so we must employ a quadrature rule.
- First, we consider problems with fixed integration rules.
- Having a fixed integration rule is morally equivalent to having a finite data set - In both cases, this can lead to *overfitting*.
- The ability to make “correct” conclusions at unseen data is called *generalisation*. The *generalisation error* is the error away from the data points.
- When data is finite, we can split it into *training data* (employed by the optimiser) and *validation data* (used to estimate the generalisation error). A 90/10 split is common.

Example - Overfitting and monitoring

- We have a simple interpolation problem with noise - We approximate $||u - u_{NN}||^2_{L^2(0,1)}$ with a small sample of data.



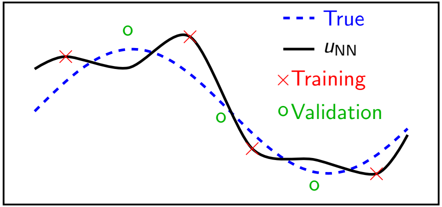
Training vs. Validation Loss Over Iterations



Comparison of Neural Network Prediction vs. Exact Solution

Overfitting and noise

- Often, the “true” solution is believed to be smooth. When noise is present, overfitting can fit to the irregular noise.
- If we overfit on the training data, the loss will be high on the validation data.
- The “best” solution will admit errors in both.

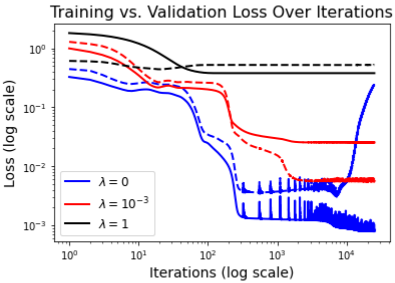


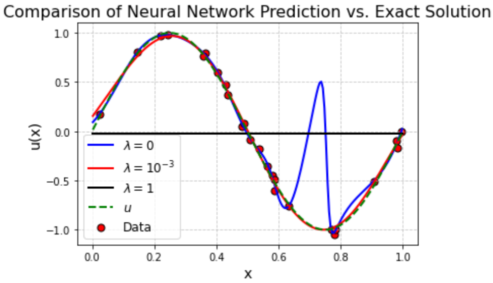
Regularisers - L^2

- One method to avoid overfitting when data is finite is to use a *regulariser*
- The idea is the following: If the true solution is smooth, an oscillatory solution will be “bad”. If u_{NN} is oscillatory, then the trainable parameters must be large.
- We penalise large parameters by augmenting the loss. We take the L^2 regulariser, so that we train with
$$\mathcal{L}(u_{NN}) + \lambda||\theta||^2,$$
where θ represents the trainable parameters and $\lambda > 0$ is a weight to be chosen.
- As the weights A_k and biases b_k play different roles, they can be weighted differently - the biases typically are not regularised at all.

Examples - Regularisers

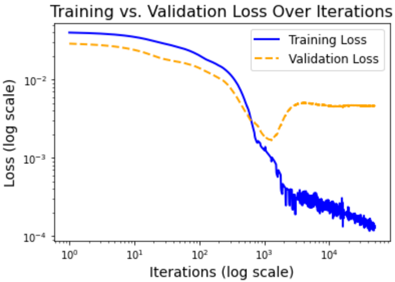
- We consider our interpolation problem with L^2 regularisation with weight λ.

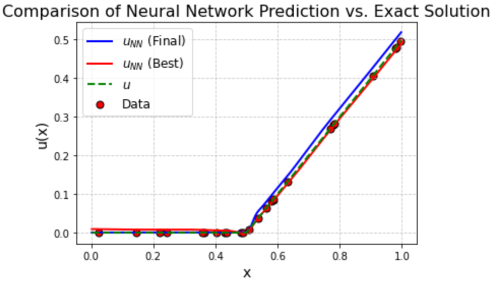




Example - ODE

- We now consider an ODE example, to solve $u'(x) = H(x - 0.5)$ with $u(0) = 0$.
- We overfit on the *derivative* - After the singularity where approximation is poor, the solution is parallel.





Stochastic integration

- Using a fixed quadrature rule can lead to overfitting - stochastic rules are favourable to avoid it.
- The standard choice is Monte Carlo - We take a random, uniform sample of N points in the domain at each iteration, and evaluate the integrand at each,

$$\int_\Omega f(x)\,dx \approx \frac{|\Omega|}{N}\sum_{n=1}^{N} f(x_n).$$

- The expected value of this random variable is the exact integrand, and its variance behaves as $\frac{1}{N}$.
- It is often cited that the *error* is of order $\frac{1}{\sqrt{N}}$ - in reality, it is the *average (squared) error* that is of order $\frac{1}{N}$. Monte Carlo does **not** admit *uniform* error estimates.

The good and the bad of Monte Carlo integration

- Monte Carlo is perhaps the most used integration method in NNs.
- If $Q_N(f)$ is an N-point Monte Carlo quadrature rule of f, and the exact integral is $I(f)$, then

$$\mathrm{Var}(f) = \frac{|\Omega|}{N} \left(\int_\Omega \left| \frac{1}{|\Omega|} I(f) - f(x) \right|^2 dx \right).$$

- Oscillatory integrands imply high variance.
- The variance is not *explicitly* dependent on the dimension (avoiding the *curse of dimensionality*), however, the integral *implicitly* depends on the dimension.
- This can still be big - to approximate $\int_0^1 \sin(2\pi x)\, dx$ with an expected error of 10^{-3}, you need 500,000 points!

An alternative - stratified Monte Carlo

- Ideally, we want a stochastic quadrature rule with far lower error.
- *Stratified* Monte Carlo is the simplest answer in low dimensions - Divide the domain into N subdomains Ω_n, take one point from each x_n, and approximate

$$\int_\Omega f(x)\,dx \approx Q_N^S(f) = \sum_{n=1}^{N} f(x_n)|\Omega_n|.$$

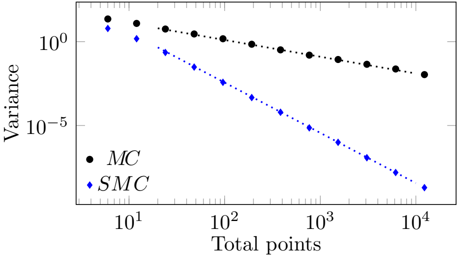
- If the partition is uniform, the variance then behaves asymptotically as

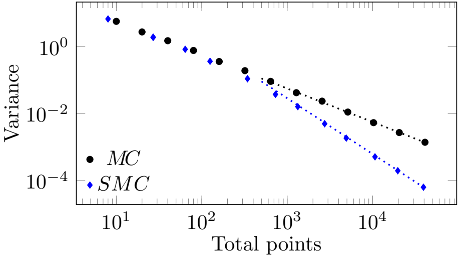
$$\mathrm{Var}(Q_N^S) \approx \frac{1}{N^{1+\frac{2}{d}}} \int_\Omega |\nabla f(x)|^2\,dx.$$

- d is the dimension - The benefit decreases in higher dimensions. If $d = 1$, this is *much* better.

Comparison - MC and SMC - Fixed integrand

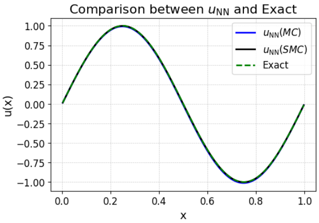
- We can see the difference in variances in a 1D (left) and 3D (right) example with a fixed integrand.
- We see clearly the curse of dimensionality.

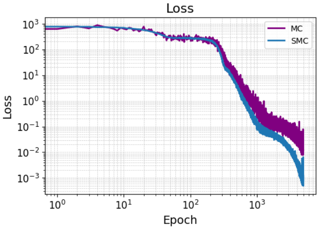


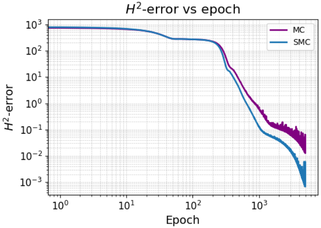


Comparison - MC and SMC - PINNs

- Let's see the difference between the two integration schemes when training an NN.
- Using the same number of points (same cost) and architecture, we see less noise in the loss, and the error final solution is orders of magnitude better.







The holy grail

- We are aiming to find a Neural Network with $u_{NN} \approx u$, with u a function of interest.
- We employ a loss function $\mathcal{L}$ whose minima are solutions of the PDE.
- Ideally, we want to say more: If $\mathcal{L}(u_{NN}) \approx \min \mathcal{L}$, then $u_{NN} \approx u$.
- We are in infinite dimensions - norms are not equivalent, and whether we have a "good" approximation depends on the norm we are interested in.
- The formulation of the PDE often suggests which norms are, or aren't, appropriate.
- Depending on what you are trying to approximate, you may wish to employ a formulation that is better equipped to approximate in a distinct norm.
- An "ideal" loss function satisfies a relation like

$$\mathcal{L}(u_{NN}) - \min \mathcal{L} \sim ||u_{NN} - u||.$$

Function spaces

- Understanding the formulation of the PDE and the way we measure or control errors all depends on the underlying function space.
- For the sake of this course, we will limit ourselves to the simpler Hilbert spaces $H^k(\Omega)$.
- $H^0(\Omega) = L^2(\Omega)$ is the space of square integrable functions, with the norm

$$||u||^2_{L^2(\Omega)} = \int_\Omega u(x)^2 \, dx.$$

- Then $H^k(\Omega)$ are the space of functions whose first k (weak) derivatives are in L^2.
- We also consider $H^1_0(\Omega)$, which is the (closed) subspace of $H^1(\Omega)$ consisting of functions that vanish on the boundary. We may use the norm

$$||u||^2_{H^1_0(\Omega)} = \int_\Omega |\nabla u(x)|^2 \, dx.$$

Strong formulations

- Strong form (classical) PDEs are given in the form $D(u) = f$, with D a differential operator, and $B(u) = g$, with B representing a boundary operator (e.g. trace or normal derivative on the boundary).
- The simplest loss function is to consider comes from the strong formulation

  $$w_1||D(u) - f||^2_{L^2(\Omega)} + w_2||B(u) - g||^2_{L^2(\partial\Omega)},$$

  with weights $w_1 > 0, w_2 > 0$.
- The use of the L^2 norm is attractive - it is smooth, and its computation amounts to integration.
- We consider Poisson's problem with Dirichlet boundary conditions, so $D(u) = \Delta u$ and $Bu = u|_{\partial\Omega}$.

Strong formulations - function spaces

- The "natural" function space for the strong formulation is $H^2(\Omega)$ - functions with two square integrable (weak) derivatives with norm
$$||u||^2_{H^2(\Omega)} = \int_\Omega |\nabla^2 u(x)|^2 + u(x)^2 \, dx.$$
- For sufficiently nice domains (e.g. C^2 boundary) if $u = g$ on $\partial\Omega$
$$||u - u^*||_{H^2(\Omega)} \leq C||\Delta u - f||_{L^2(\Omega)}$$
- In practice, we may use
$$||\Delta u - f||^2_{L^2(\Omega)} + w||u - g||^2_{L^2(\partial\Omega)}$$
(issues arise in boundary spaces - we'll see this later).
- In non-convex sets (e.g. the famous L-Shaped domain), it may be that $\Delta u \in L^2(\Omega)$, but $u \notin H^2(\Omega)$.

The effect of weighting

- We will consider a simple problem: Find $u \in H^2(\Omega)$ such that

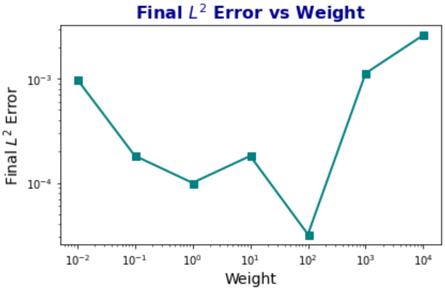
$$\begin{aligned} \Delta u(x) =& f(x) \quad (x \in \Omega) \\ u(x) =& g(x) \quad (x \in \Omega). \end{aligned} \tag{1}$$

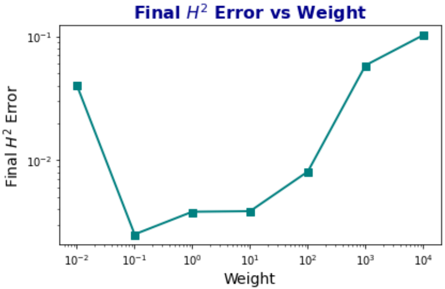
- We will employ the loss function

$$\mathcal{L}(u) = ||\Delta u - f||^2_{L^2(\Omega)} + w||u - g||^2_{L^2(\partial\Omega)}$$

- Our problem will be on the disk in $\mathbb{R}^2$, which is a smooth domain.
- The first question to ask is: what should w be?
- In reality, nobody has come up with a satisfactory answer yet.

The effect of weighting - 2D PINN





- Note - The “best” weight for L^2 approximation and the “best” weight for H^2 approximation differ by 3 orders of magnitude.
- Any change in the architecture or optimisation strategy will probably change the “best” empirical weight.

A comment on scalings

- We can one observation on weightings - Consider

$$\mathcal{L}(u) = ||\Delta u - f||^2_{L^2(\Omega)} + w||u - g||^2_{L^2(\partial\Omega)}$$

  and rescale $u(x) = \tilde{u}(rx)$, $\tilde{f}(rx) = \frac{1}{r^2} f(x)$, $\tilde{g}(rx) = g(x)$. Then we are solving

$$\begin{aligned} \Delta\tilde{u}(x) =& \tilde{f}(x) \quad (x \in \Omega) \\ \tilde{u}(x) =& \tilde{g}(x) \quad (x \in \Omega). \end{aligned} \tag{2}$$

- We haven't really *changed* the problem, but we see that

$$\mathcal{L}(u) = \tilde{\mathcal{L}}(\tilde{u}) = r^6 \left(||\Delta\tilde{u} - \tilde{f}||^2_{L^2(\Omega)} + \frac{w}{r^5}||\tilde{u} - \tilde{g}||^2_{L^2(\partial\Omega)} \right). \tag{3}$$

- The global factor r^6 is not as important, but the weighting of the boundary term can change significantly - the "optimal" weight is even sensitive to the length scale we employ.

Comments from the literature

- Wang, Yu and Perdikaris (2002) did a very technical, but very enlightening, paper based on *Neural tangent kernels*. It is morally a linearisation argument in an appropriate limit.
- Their method provides a way of estimating the optimal weights during training (that should change during training). Empirically, they reached the conclusion that higher order terms need lower order weights - E.g., Dirichlet BCs should have weights order 10^4, Neumann order 10^2, a second order PDE order 1.
- The cost of the method however puts the practicality in doubt.
- De Ryck et. al. (2024) considered the problem of weighting as an issue on the conditioning of the PDE, and finding an appropriate preconditioner is instructive as to how one should chose the weights.
- Again, their methodology is mathematically elegant, but not yet practical.

A comment on boundary spaces

- In higher dimensions, the boundary condition corresponds to an element of an infinite dimensional function space.
- In a strong formulation, where $u \in H^2(\Omega)$, the trace onto the boundary is in $L^2(\partial\Omega)$, but *not all elements of $L^2(\partial\Omega)$ are boundary values of functions in $H^2(\Omega)$*.
- The Dirichlet boundary condition naturally lives in the space $H^{\frac{3}{2}}(\partial\Omega)$, and the corresponding norm is "natural" to include in a loss function.
- The problem is that the $H^{\frac{3}{2}}(\partial\Omega)$ norm involves a singular, non-local integral.
- Let's find a way to avoid such problems....

Strong imposition/Pinning

- We can avoid issues with weighting by forcing the NN to satisfy the boundary condition in the architecture.
- We choose a smooth function $\varphi : \bar{\Omega} \to \mathbb{R}$ that satisfies $\varphi(x) = 0$ on $\partial\Omega$ and $\varphi(x) > 0$ on Ω, and any u_0 in our function space that satisfies $u_0(x) = g(x)$, then consider

$$u_{NN}(x) = \varphi(x)\tilde{u}_{NN}(x) + u_0(x),$$

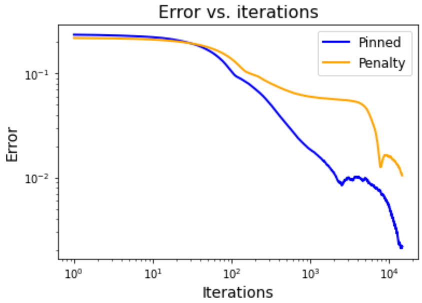
  where $\tilde{u}_{NN}$ is a "classical" NN (e.g. Fully-Connected Feed Forward).
- E.g., if the BC is $u(0) = 1$, $u(1) = 3$, we may take

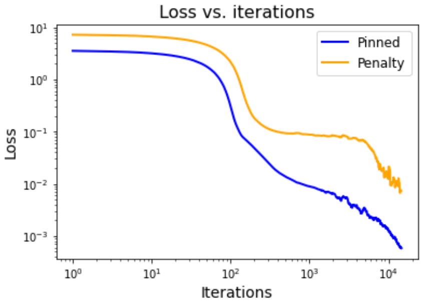
$$u_{NN}(x) = x(1-x)\tilde{u}_{NN}(x) + (1+2x).$$

- φ and u_0 are a cutoff and lift, respectively.

Pinning 2D

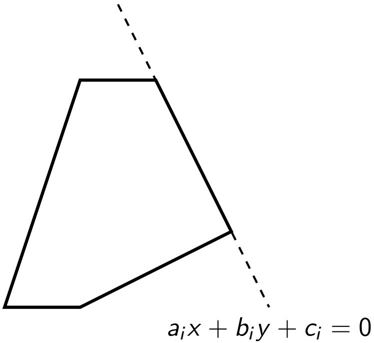
- We consider a PDE in the domain $\Omega = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 < 1\}$.
- There is a simple cutoff function, $\varphi(x, y) = 1 - x^2 - y^2$.
- We compare it with a penalty method, where the weight is the optimal one from the previous example.





Pinning 2D

- In *very* simple geometries, the choice of the cutoff and lift may be simple.
- In realistic geometries, this may be more complex.
- in $(-1,1)^2$, for example, we can consider the cutoff $\varphi(x,y) = (1-x^2)(1-y^2)$.
- Generally, if Ω is a convex polygon, we can use a cutoff of the form $\varphi(x,y) = \prod_{n=1}^{N}(a_i x + b_i y + c_i)$.

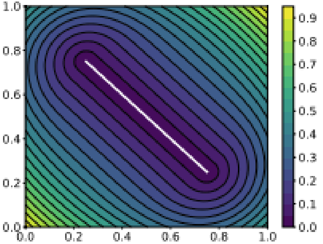


Alternative approach

- In polygons (including non-convex), we can build a cutoff fuction out of cutoffs defined on each edge (C.f. Berrone et. al.)
- The functions $\varphi_n(x, y)$ on each edge are necessarily singular at the vertices - this may be good (if there is a singularity) or bad (if there isn't).
- We can construct the full cutoff as a square-harmonic-type mean of the edge cutoffs,

$$\varphi(x, y) = \left(\sum_{n=1}^{N} \frac{1}{\varphi_n(x, y)^2} \right)^{-\frac{1}{2}} .$$

- The normal derivative of φ at the boundary is (although indeterminate) 1, so the normal derivative can be reclaimed.



(c) $\phi(\mathbf{x})$.

Some concluding remarks on BCs

- Imposing essential boundary conditions is very delicate.
- There is a question of natural function spaces given the formulation that may require difficult norms.
- When using a penality method, the quality of your solution depends highly on the choice of weight.
- Some theory exists as to how to choose the weights, but they are not yet competitive for practical problems.
- Pinning is an alternative approach, but requires the definition of a cutoff function, which is non-trivial in complex geometries.

PDEs as regularisers

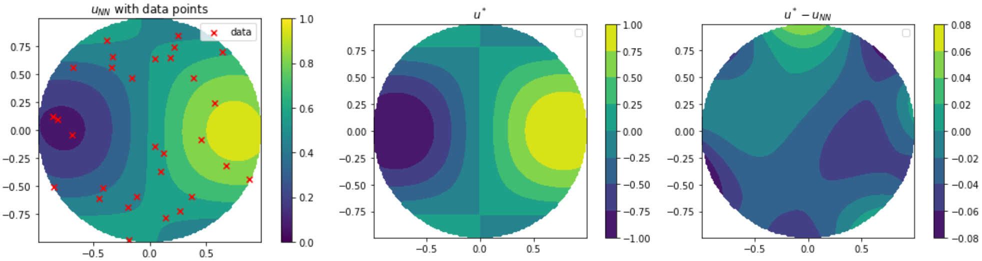
- Given (potentially noisy) data that comes from a known process, we may be able to interpret the PDE as a regulariser for an interpolation problem.
- Say, e.g., we know that $u(x)$ satisfies the PDE $k\Delta u(x) = f(x)$, with k unknown and f known. The boundary data $g(x)$ is known, and we have noisy measurements (x_j, u_j). We may consider a loss function

$$\mathcal{L}(u,k) = \sum_{j=1}^{J}(u(x_j) - u_j)^2 + w_1 \int_{\Omega} |k\Delta u(x) - f(x)|^2 \, dx + w_2 \int_{\partial\Omega} |u(x) - g(x)|^2 \, dx,$$

  approximating the integrals by MC.
- We train over k as well.
- The MC integrals have "infinite data" - they should not overfit.

PDEs as regularisers

- Even with very little data, we recover a reasonable approximation.
- We also recover the value of k to within 5% error.
- This approach can also be understood as solving an inverse problem for k.



Some concluding remarks on losses in strong formulations

- Phrasing the (continuum) PDE in the correct function spaces indicates what your loss function should look like.
- Boundary conditions are tricky - Penalty methods are sensitive to the chosen weight, the "correct" norms are non-trivial, and pinning can be difficult in non-trivial geometries.
- When (noisy) data is known to follow physical laws, a hybrid interpolation/PDE problem with trainable parameters can act as either a regulariser for interpolation orr an inverse problem to find the parameters.

Weak solutions - Function spaces

- Returning to Poisson's problem, we can consider the PDE to find $u \in H^1(\Omega)$ satisfying $\Delta u = f$ in weak form,

  $$\int_\Omega \nabla u(x) \cdot \nabla v(x) + f(x)v(x)\,dx = 0$$

  for all $v \in H_0^1(\Omega)$, where u is required to satisfy a Dirichlet condition, $u|_{\partial\Omega} = g$.
- The "natural" space for the boundary condition is $H^{\frac{1}{2}}(\partial\Omega)$ (again, this is defined by a singular non-local integral).
- We will take $g = 0$ for simplicity.

The Deep Ritz Method

- For positive definite, symmetric, elliptic PDEs, we can use the Ritz Method. The idea is that the minimiser over $H_0^1(\Omega)$ of

$$\mathcal{F}(u) = \int_\Omega \frac{1}{2}|\nabla u(x)|^2 + f(x)u(x)\,dx$$

  solves the weak form PDE.

- It also satisfies the error estimate

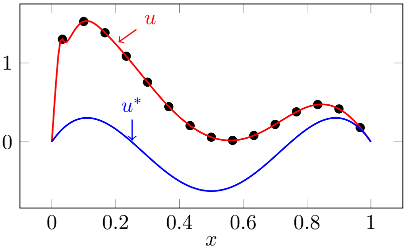
$$\mathcal{L}(u) - \min \mathcal{L} = \frac{1}{2}||u - u^*||^2_{H_0^1(\Omega)}.$$

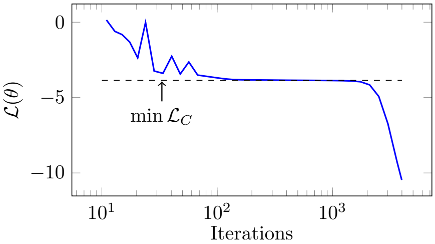
- $\min \mathcal{L}$ is unknown - it's hard to estimate how good our solution is.

Integration in the Deep Ritz Method

- The Deep Ritz Method is *highly* susceptible to overfitting when training.
- With a fixed midpoint rule, the discrete loss is

$$\int_0^1 \frac{1}{2}u'(x)^2 + f(x)u(x)\,dx \approx \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}u'(x_n)^2 + f(x_n)u(x_n)$$

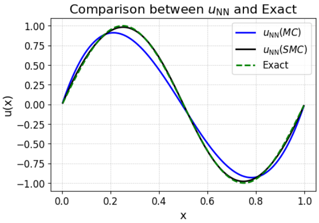


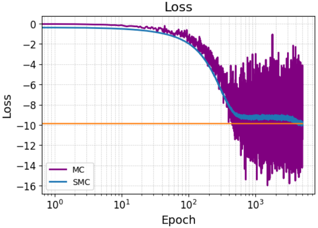


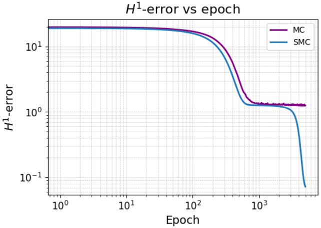
- This highlights a recurring problem with losses in weak formulations - We don't minimise an *integrand* pointwise, rather *integrals* - we need to integrate "well".

Integration in DRM

- Recall that MC is the “standard” for stochastic integration. It is, however, very noisy, and noise can impede convergence. Stratified MC can help here.
- We see a significant difference in behaviour betwen MC and stratified MC







DRM and irregular solutions

- If $\sigma : \Omega \to (0, \infty)$ is piecewise continuous, then we can consider weak solutions in $H_0^1(\Omega)$ of

$$\int_\Omega \sigma(x) \nabla u(x) \cdot \nabla v(x) + f(x) v(x) \, dx = 0.$$

- These are also minimisers of a Ritz energy,

$$\int_\Omega \frac{1}{2} \sigma(x) |\nabla u(x)|^2 + f(x) u(x) \, dx,$$

- They satisfy an estimate in the energy norm:

$$\mathcal{L}(u) - \min \mathcal{L} = \frac{1}{2} \int_\Omega \sigma(x) |\nabla u(x) - \nabla u^*(x)|^2 \, dx. \tag{4}$$

- There is no strong formulation (as such...) for this problem - The solutions are in $H^1 \setminus H^2$.
- Sometimes weak solutions are the *only* method, and we need techniques for them.

Weak formulations - Dual spaces

- The DRM only applies to very specific problems. To solve more general problems, we need a functional analysis framework.
- Given a Hilbert space H, its dual H^* is defined to be the set of continuous, linear functionals on H. That is, $l \in H^*$ if $l : H \to \mathbb{R}$ (or $\mathbb{C}$) such that $l(au + bv) = al(u) + bl(v)$, and
$$|l(v)| \leq M||v||_H$$
for all $v \in H$.
- The norm on H^* is given by
$$||l||_{H^*} = \sup_{v \in H \setminus \{0\}} \frac{l(v)}{||v||_H}.$$

Understanding the weak form

- We then understand the PDE via linear operators, if we define $l \in H_0^1(\Omega)^*$ and $B : H^1(\Omega) \to H_0^1(\Omega)^*$ by

$$\begin{aligned} l(v) &= \int_\Omega f(x)v(x)\,dx, \\ Bu(f) &= \int_\Omega \nabla u(x) \cdot \nabla v(x)\,dx, \end{aligned} \tag{5}$$

  then the PDE is

$$Bu = l.$$

- Choosing the norms appropriately, one obtains

$$||u - u^*||_{H^1(\Omega)}^2 = ||Bu - l||_{H_0^1(\Omega)^*}^2 + ||u||_{H^{\frac{1}{2}}(\partial\Omega)}^2$$

Loss function - Weak formulation

- The equality
$$||u - u^*||^2_{H^1(\Omega)} = ||Bu - l||^2_{H^1_0(\Omega)^*} + ||u||^2_{H^{\frac{1}{2}}(\partial\Omega)}$$
suggests an ideal loss function,
$$\mathcal{L}(u) = ||Bu - l||^2_{H^1_0(\Omega)^*} + ||u||^2_{H^{\frac{1}{2}}(\partial\Omega)}.$$
- Evaluating the dual norm is effectively a PDE itself, and the boundary term involves singular, non-local integral.
- Evaluating it is not easy.
- We can remove the problem of the boundary term via pinning. The dual norm is harder.

Method 1 - (R)VPINNs

- Our aim is to estimate $||Bu - l||_{H^*}$. This is defined via a maximisation problem which may be as hard or harder as the original PDE. We can only "access" $Bu - l$ via a duality pairing,

$$(Bu - l)(v) = \int_{\Omega} \nabla u(x) \cdot \nabla v(x) + f(x)v(x)\, dx.$$

- The idea of VPINNs (Variational Physics Informed Neural Networks) is to choose a set of test functions $(v_k)_{k=1}^{K}$ and consider the loss

$$\mathcal{L}(u) = \sum_{k=1}^{K} |(Bu - l)(v_k)|^2$$

- The loss is *highly* sensitive to the choice of test functions.
- The details are technical (ask us later if you want them), but we want $(v_k)_{k=1}^{K}$ to be H_0^1 orthonormal - Finding such a set can be very expensive, outside of simple geometries.

(R)VPINNs

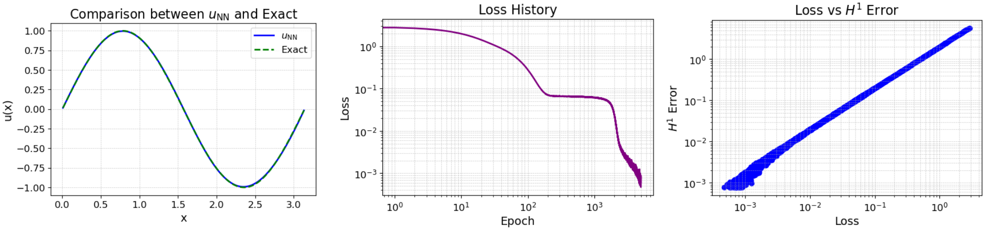
- When the test functions are orthonormal in H_0^1, we have that

$$||Bu - l||^2_{H_0^1(\Omega)^*} \sim \sum_{k=1}^{K} |(Bu - l)(v_k)|^2 + \text{Truncation error}$$

- This yields (R)VPINNs (Robust-VPINNs). If the test functions are not orthonormal, the $\sim$ becomes very weak and lead to poor training.
- We also have to integrate K times, and errors in the integral have an effect as well.
- We will consider the simplest case - in $H_0^1(0,\pi)$, the functions $\sqrt{\frac{2}{\pi}}\frac{\sin(kx)}{k}$ are H_0^1-orthonormal.
- This choice of test functions is the Deep Fourier Residual method.

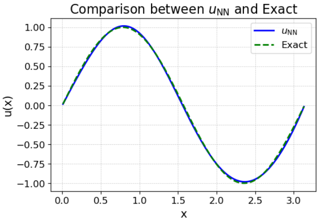
DFR

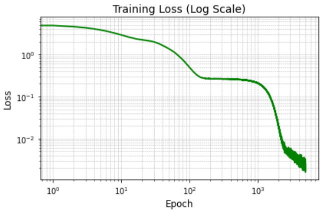
- We see a good convergence to the exact solution.
- The most attractive feature is the right-hand plot - There is a strong correlation between the loss and the H_0^1 error.

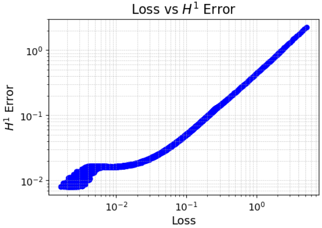


DFR

- We can see the effect of the test functions - I take the same as before, but multiply them by random coefficients. The space of test functions is the same, but the loss behaves differently.
- We no longer have such a nice correlation.

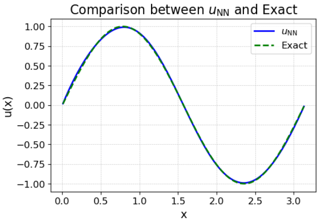


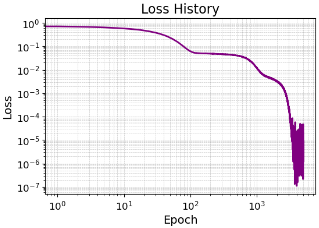


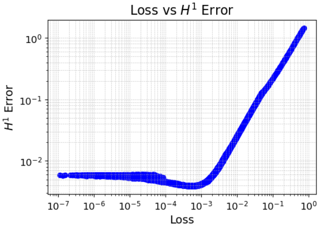


DFR

- Let's see what happens when there are very few test functions - I take only 5 modes.
- The loss is very small, but the errors are on higher frequencies, breaking the relation between the loss and error.







The issues with VPINNs

- In higher dimensions, you need a lot of test functions - these are expensive to evaluate.
- Even finding the test functions (or making them orthonormal to ensure robustness) is expensive outside of simple geometries
- The test functions must naturally be highly oscillatory - this means integration errors can be significant.
- How about we try to do it with *one*, very carefully chosen, test function?

WANs

- Another approach is WANs (weak adversarial neural networks). The idea is the following. As

$$||Bu - l||_{H_0^1(\Omega)^*} = \sup_{v \neq 0} \frac{(Bu - l)(v)}{||v||_{H_0^1(\Omega)}},$$

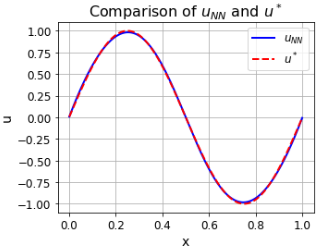
  we can consider a min/max problem using two neural networks:

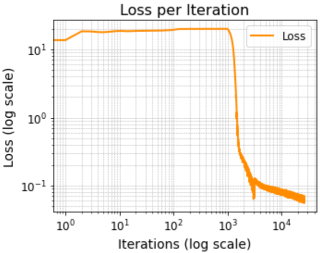
$$\min_u \max_v \frac{(Bu - l)(v)}{||v||_{H_0^1(\Omega)}}.$$

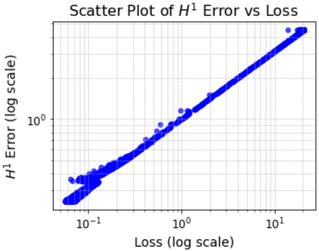
- We perform gradient ascent on v and descent on u.
- Like many other weak formulations, this is very sensitive to integration
- The min/max problem is also analytically and numerically much harder to solve for many reasons.

WANs results

- We consider the same 1D problem.
- Note: This example required a *lot* of tuning, using different optimisation strategies for u and v.

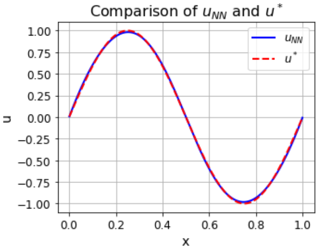


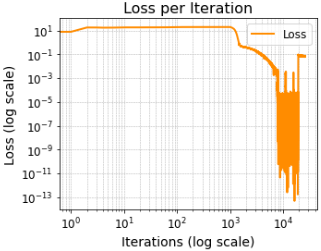


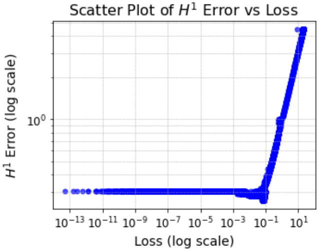


WANs results 2

- This is the behaviour when a “poor” optimisation strategy is chosen.
- Things work up to a point, but then the loss collapses towards zero and no improvement is made.
- One cause is related to the instability of the min/max problem near the exact solution - $(Bu - l)(v) = 0$ for all v.







Some concluding remarks on weak formulations

- Weak formulations are sometimes the only option, and may be more natural depending on the norm you want to minimise.
- The functional analysis framework to study weak formulations is based on dual norms, which are far more difficult to handle than L^2 norms.
- As weak formulations are based on integrals with potentially complex integrands, integration becomes a more significant challenge.