Session 2 part 1- Optimisation strategies

Jamie M. Taylor (CUNEF Universidad, Madrid, Spain)
jamie.taylor@cunef.edu

BCAM - In-Deep workshop

(Stochastic) Gradient Descent

Continuum gradient descent

- For now, let us consider an abstract, differentiable loss function $\mathcal{L} : \mathbb{R}^N \to \mathbb{R}$ and try to minimise it.
- As with loss functions, we will start by thinking "in the continuum", with the gradient descent: Given an initial state θ_0, we aim to find the solution to

$$\begin{aligned} \theta'(t) &= -\nabla \mathcal{L}(\theta(t)). \\ \theta(0) &= \theta_0. \end{aligned} \tag{1}$$

- The chain rule gives that

$$\frac{d}{dt}\mathcal{L}(\theta(t)) = \theta'(t) \cdot \nabla \mathcal{L}(\theta(t)) = -||\nabla \mathcal{L}(\theta(t))||^2.$$

- The loss decreases.

Continuum gradient descent

- We don't just want it to decrease, we want it to decrease *quickly*.
- Lets keep things simple - Say $\mathcal{L}(\theta) = \frac{1}{2} A\theta \cdot \theta$, A a PD symmetric matrix. Then $\theta^* = 0$ is the minimiser, and $\theta'(t) = -A\theta(t)$.
- If $\lambda_{\min}$ is the smallest eigenvalue of A, then we expect

$$||\theta(t)|| \leq C \exp(-\lambda_{\min} t).$$

- That is, a (nearly) degenerate matrix A will lead to slow convergence.

Discrete gradient descent

- Of course, we have to discretise. Effecttively, we use finite differences: We initialise at θ_0, and update via

$$\theta_{k+1} = \theta_k - \gamma \nabla \mathcal{L}(\theta_k).$$

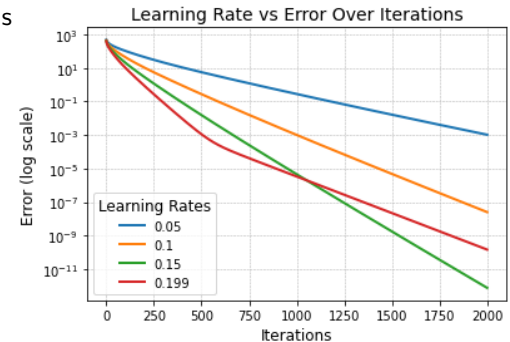
- Again, if $\mathcal{L}$ is quadratic, we obtain

$$\theta_{k+1} = (I - \gamma A)\theta_k.$$

- The solution is $\theta_k = (I - \gamma A)^k \theta_0$, so the norm is $||\theta_k - \theta * || \sim ||I - \gamma A||^k$ (in operator norm).
- We can see two problems - if the smallest eigenvalue of $1 - \gamma \lambda_{min}(A) \approx 1$, or $1 - \gamma \lambda_{max}(A) \approx -1$, then convergence will be very slow. If $1 - \gamma \lambda_{max}(A) > 1$, the norm will explode.
- That is, γ too big means things explode, γ too small means it takes forever.

Examples with learning rates

- We consider our linear problem, where the matrix A is such that it has eigenvalues distributed log-uniformly between 10^{-1} and 10.
- If $\gamma < 0.2$, we should have exponential convergence.
- Let's see the effect of the learning rate.
- If $\gamma > 0.2$, then the error explodes exponentially.



Stochastic gradient descent

- In reality, we can't always access the full gradient $\mathcal{L}$.
- In interpolation problems, it may be too memory-intensive to evaluate the loss over the entire data set (and we only have a finite sample).
- In PDEs, we cannot integrate the loss exactly, we can only use a quadrature rule with error.
- This yields a *stochastic gradient descent*. We have a collection of "data" ξ, and a loss function that can be written as

$$\mathcal{L}(\theta) = \frac{1}{J}\sum_{j=1}^{J} \mathcal{L}(\theta;\xi_j) \quad \text{or} \quad \mathcal{L}(\theta) = \mathbb{E}\left(\mathcal{L}(\theta;\xi)\right).$$

- We then perform updates as

$$\theta_{k+1} = \theta_k - \gamma \mathcal{L}(\theta_k, \xi_k),$$

where ξ_k is a sample of the data.

The good and the bad of SGD

- By introducing noise into the system, this can help prevent overfitting when data is finite, as it is harder for the NN to "cheat".
- It can also help the NN "jump" out of undesirable local minima.
- Noisy gradients can also impede convergence.
- The "best" we can usually hope for is to reach an equilibrium where we are oscillating around a "good" minimiser.
- If we split into batches, the parameters update per batch, not per epoch, and this adds an "overhead" cost (potentially big, potentially small).

Finite vs. infinite data

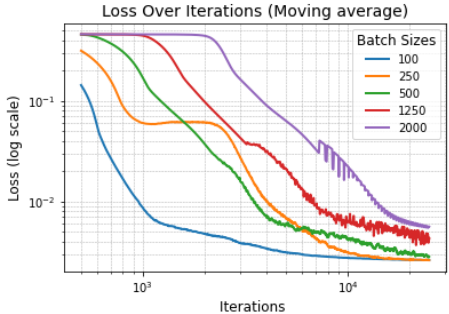
- In more "data science" approaches, data is expensive and finite. When we think about random quadrature (e.g. MC) in PDEs, data is infinite. This leads to some mild differences in approaches.

	Finite data	Quadrature
Batch sizes	Each epoch, divide data into minibatches, subsample	No batches as such.
Validation	Need to split data to monitor overfitting	Overfitting unlikely (not impossible) but a fixed validation should be used.
Minimisers	Minimiser of average loss will *probably* overfit	Minimiser of average loss *should* be good.
Noise	The data itself and batch effects	Integration errors

- These are not rigorous statements, but conceptual ideas that distinguish the cases.

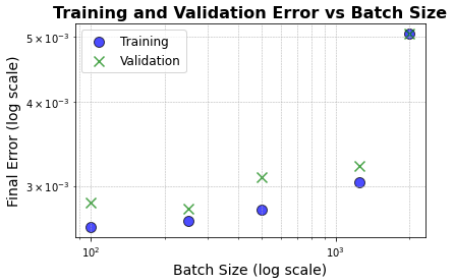
The effect of batch size

- We consider an interpolation problem in 1D with noisy data.
- We have 2000 samples and consider the effect of batch size.



The effect of batch size

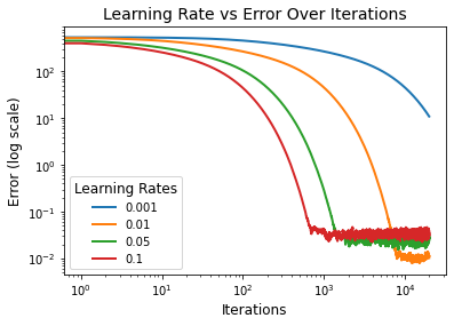
- We can also compare the final training and validation losses.
- We see less disparity with largest batch sizes, but the final values are lower with smaller batches.
- As we update our parameters after every batch, this introduces a larger "overhead" cost. For a "small" problem like this, it is significant.



Batch size	100	250	500	1000	2000
Time (100 epochs)	6.4s	4.9s	4.8s	4.4s	4.0s

SGD with noisy, infinite data

- Let's return to a toy linear problem, but now with a noisy gradient.
- We take $\theta_{k+1} = \theta_k - \gamma(A\theta_k + \epsilon_k)$, where ϵ_k is a random variable with mean zero and variance 10^{-2}.
- Recall that in the case without noise, we had exponential convergence, achieving errors of order 10^{-30} in some cases in a few thousand iterations.



SGD with noisy data

- The noise impedes the convergence significantly.
- One method to overcome this is to reduce the learning rate during training, so $\gamma = \gamma_k$.
- Whilst exponential decrease is common in the literature, there are various works that suggest an algebraic decay is better - $\gamma_k \sim k^{-\alpha}$ with $\alpha \in \left(\frac{1}{2}, 1\right]$.
- When algebraic, under technical hypotheses (that probably don't hold in NNs...), we have that

$$\frac{\theta_k - \theta^*}{\gamma_k} \to \mathcal{N}(0, V),$$

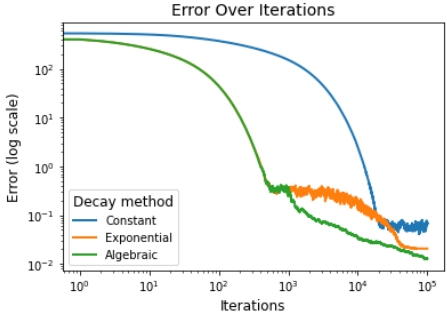
where V solves a Lyupanov equation,

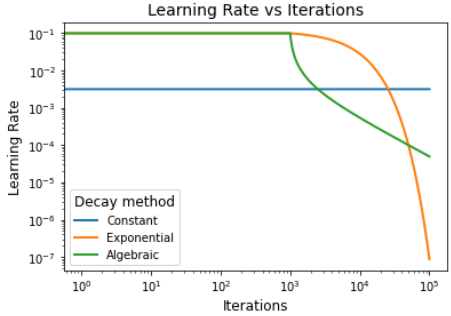
$$V H_{\mathcal{L}}(\theta^*) + H_{\mathcal{L}}(\theta^*) V = \mathrm{Var}(\nabla \mathcal{L}(\theta^*))$$

- This suggests two things: Degeneracy of the Hessian or large variances make V big (poor convergence).

Decaying learning rates

- We consider our toy, linear, noisy model, taking $\theta_{k+1} = \theta_k - \gamma_k(A\theta_k + \epsilon_k)$.
- We consider 3 regimes - A constant learning rate, an exponentially decaying learning rate, and an algebraically ($\sim \frac{1}{k}$) decaying learning rate.





Some concluding remarks on SGD

- SGD for simple, deterministic problems has exponential convergence.
- When noise is present, this significantly impedes convergence.
- Decaying learning rates can overcome this, but convergence will be much slower - if it converges at all.
- When data is used, small batches may lead to a larger cost, but better solutions.

Momentum and related optimisers

(Stochastic) Gradient Descent

Momentum and related optimisers

Other strategies

Momentum

- (S)GD can be understood as a discretisation of a continuum gradient descent, $\theta' = -\nabla\mathcal{L}(\theta)$.
- Momentum methods can be understood as discretisations of a second order system,

$$\theta'' + \mu\theta' = -\nabla\mathcal{L}(\theta).$$

- The updates in the discrete system are written as

$$\begin{aligned} m_{k+1} &= \beta m_k - \gamma\nabla\mathcal{L}(\theta_k) \\ \theta_{k+1} &= \theta_k + m_{k+1}. \end{aligned} \tag{2}$$

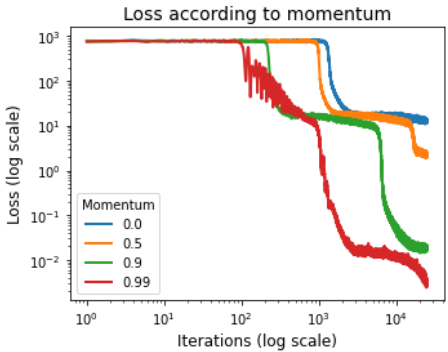
- $0 < \beta < 1$ is typically taken close to 1.

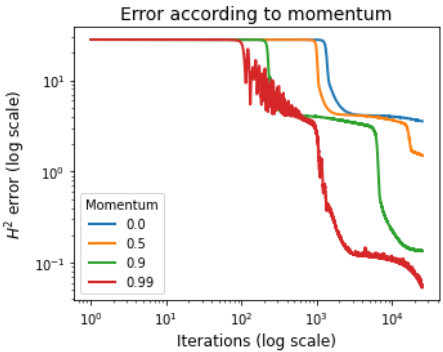
The good and bad of Momentum

- The momentum vector m acts like an "average" of recent gradients - This can smooth out noisy gradients.
- When the loss is very "flat", momentum can accelerate convergence (think of a ball rolling down a soft slope)
- Near a minimum, the use of momentum may lead to more oscillatory behaviour (think simple harmonic oscilator, $y'' = -\omega y$ versus exponential decay $y' = -\omega y$).
- When the gradient is changing a lot, the momentum "remembers" previous gradients that are not as useful, this can lead to instability.

Examples SGD(+M)

- We revisit our 1D PINN problem, using SGD with a fixed learning rate (10^{-3}) and varying the momentum parameter.





The holy grail - Newton methods

- Each component of the trainable parameters θ has a different "role", and will generally perform better if a correct learning rate is defined for each one.
- The Newton method goes further, we approximate the loss locally via

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta_k) + \nabla\mathcal{L}(\theta_k) \cdot (\theta - \theta_k) + \frac{1}{2} H_{\mathcal{L}}(\theta_k) \cdot (\theta - \theta_k) \cdot (\theta - \theta_k),$$

  whose exact minimum is attained at

$$\theta = \theta_k - H_{\mathcal{L}}(\theta_k)^{-1} \nabla\mathcal{L}(\theta_k).$$

- Newton methods are usually *very* fast to converge.
- The problem is the cost - if we have 5,000 variables, the Hessian involves evaluating $O(10^7)$ derivatives, then we have to solve the corresponding linear system.
- Nonetheless, they tell us that we should be thinking of "learning rates" that act on different parameters differently and depend on the local structure of the loss.

Adam

- There are many different optimisers with varying levels of sophisitication and applicability - One of the most popular is Adam (which I have used in almost every simulation I have shown), employing adaptive moment estimates to obtain per-parameter learning rates.
- The precise algorithm has many ingredients, but the key objects are a momentum estimate m_k and a second-moment estimate v_k of $\nabla\mathcal{L}(\theta_k)^2$, then

$$\theta_{k+1} = \theta_k - \gamma \frac{1}{\sqrt{v_k} + \epsilon} m_k.$$

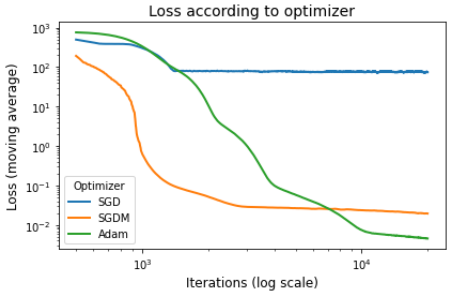
- $\gamma \frac{1}{\sqrt{v_k}+\epsilon}$ acts as an effective learning rate, morally like $H_{\mathcal{L}}(\theta_k)^{-1}$.

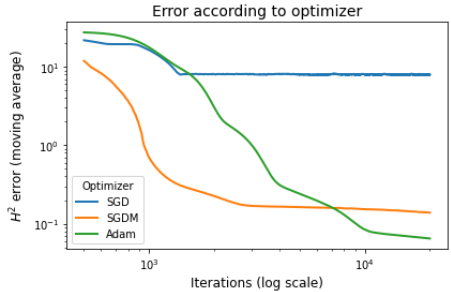
The good and the bad of Adam

- Adam is typically significantly faster to converge than SGD (in my experience, orders of magnitude).
- It is not as sensitive as SGD to the chosen learning rate - it adapts the learning rate itself.
- Adam can be very unstable - You may have noticed that in the previous simulations the loss occasionally "jumps" and then quickly returns to its original value.
- Adam contains many parameters that need to be tuned (learning rate, decay for first moment estimates, decay for second moment estimates, regulariser ϵ)

Comparison Adam vs SGD

- We employ SGD, SGDM and Adam on the same problem with the same initial learning rate for a 1D PINN problem.
- Note: I have shown moving averages - All three are noisy.
- SGDM required fine-tuning of the momentum parameter, Adam used the default settings.





Some concluding remarks on momentum-based methods

- Momentum and Adam can yield significant gains in convergence for the same cost.
- Adam tends to be the optimiser of choice for problems with NNs.
- Momentum/Adam methods have internal hyperparameters that need to be tuned, and this may heavily effect convergence.
- Different trainable parameters have different roles, and may need different approaches (simplest case - distinct learning rates)

Other strategies

(Stochastic) Gradient Descent

Momentum and related optimisers

Other strategies

Least-square solver

- The issue with Newton methods is the cost, the system is huge, the derivatives expensive to compute, and if the loss is not convex, it may be unstable.
- Take a fully-connected feedforward NN, and we will write it as

$$u_{NN}(x) = \sum_{n=1}^{N} c_n u_n(x).$$

- Each u_n corresponds to a node in the last layer.
- We can rewrite the mean-squared error loss as

$$\begin{aligned}\mathcal{L}(u_{NN}) =& \frac{1}{J}\sum_{j=1}^{J}(u_{NN}(x_j) - y_j)^2 \\ =& \sum_{n,m=1} c_n c_m \left(\frac{1}{J}\sum_{j=1}^{J} u_n(x_j)u_m(x_j)\right) - 2\sum_{n=1}^{N} c_n \left(\frac{1}{J}\sum_{j=1}^{J} u_n(x_j)y_j\right) + \text{constant}.\end{aligned}$$

Least-squares solver

- We can write the loss, defining ω to be the other trainable parameters of u_{NN} apart from c, as

$$\mathcal{L}(u_{NN}) = A(\omega)c \cdot c - 2b(\omega) \cdot c + \text{constant},$$

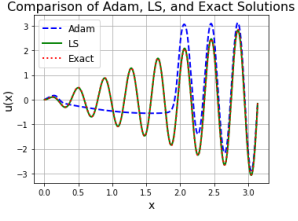
  whose exact minimiser over c is $A(\omega)^{-1}b(\omega)$.
- By defining $c = c(\omega) = A(\omega)^{-1}b(\omega)$, we obtain a (very non-linear) loss to be optimised over ω, which can be done via (e.g.) SGD or Adam.
- The idea is that the loss is quadratic in a small number of variables, so we can find the minimum via linear algebra, whilst the remaining variables can be solved by a gradient-based method.
- This is the idea of the hybrid LS-solver of Cyr et. al.

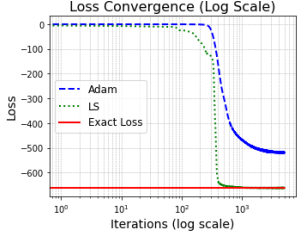
The good and the bad of LS

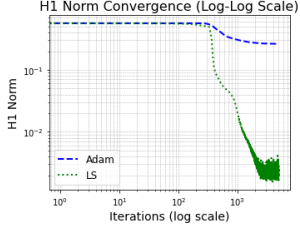
- The method can be very good at approximating oscillatory solutions, where many gradient-based methods get trapped easily in local minima.
- It can highly accelerate convergence, even improving the initial loss by orders of magnitude.
- Matrix inverses are very sensitive, errors in $A(\omega)$ can lead to large errors in $A(\omega)^{-1}$ - and this can produce erroneous results.
- It is unclear how to apply the idea to minibatches with data - Constructing the full LS system is expensive, but lacking data points may lead to significant errors.

LS Example

- We take a Deep Ritz Method in 1D with an oscillatory solution.
- Note - there is little noise, this is because LS is very unstable with poor integration, so I used a very precise integration method (not feasible in 3D).
- We see Adam fails - this highlights the issue of *spectral bias*.

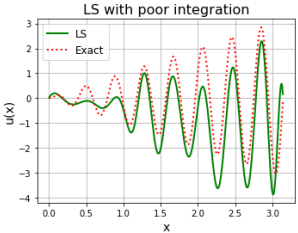


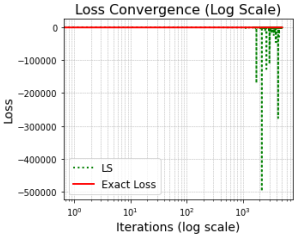


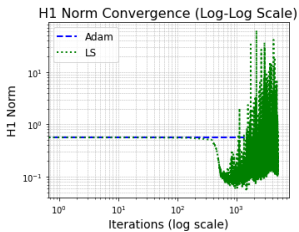


LS Example

- If we use less integration points, we can see how sensitive the LS method is to integration errors







Spectral bias

- Neural Networks suffer from *spectral bias*, also referred to as the *frequency principle* (F-principle).
- It states that, during training, first the low-frequency features are learned, and then the high frequency features.
- The "corollary" of this is that when solutions are naturally high frequency, it can be hard to train an NN to approximate it.
- It also means that the error at the end of training should be dominated by high-frequency components.
- We can try to take advantage of this idea to develop a new training approach.

Multilevel NNs

- Multi-level Neural Networks (Aldirany et. al. 2024) are based on a multi-step training process.
- We will not follow their exact methodology, but instead the spirit of it here.
- Suppose we have a PINNs type-problem, $u''(x) = f(x)$, and we train a neural network u_0 to approximate the solution.
- The error, $u_1 = u - u_0$, also satisfies an ODE,

$$u_1''(x) = u''(x) - u_0''(x) = f(x) - u_0''(x).$$

- We can then view u_0 as fixed (no more training), and try to approximate the error. This is an ODE of the same type and $u_0 + u_1$ is our approximate solution.
- We train u_1 with this loss and consider $u_0 + u_1$, and repeat for the error $u_2 = u - (u_1 + u_2)$.

Multilevel NNs

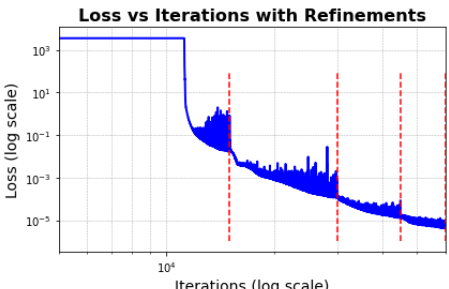
- Due to the F-principle, we expect u_1 to have high-frequency modes, u_2 to have higher-frequency modes and so on.
- As such, we use an architecture that facilitates high-frequency modes.
- We take the first layer of our NN to be

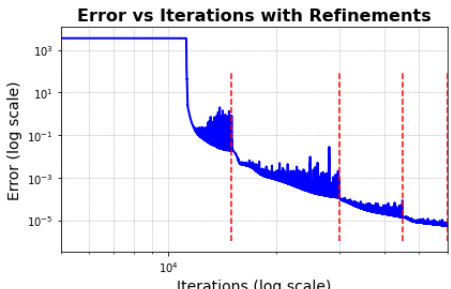
$$L_1(x) = (x, \cos(k_1 x), \sin(k_1 x), \cos(k_2 x), \sin(k_2 x), ...)$$

  for some $(k_i)_{i=1}^K$. The remaining layers are those of a fully-connected feed-forward NN.
- We take higher frequencies in the later refinements.

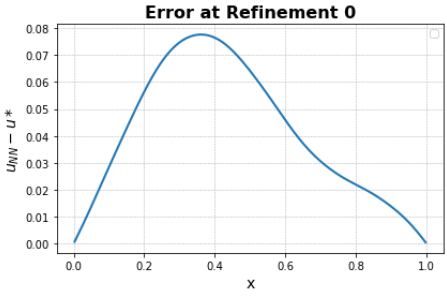
Multilevel NNs

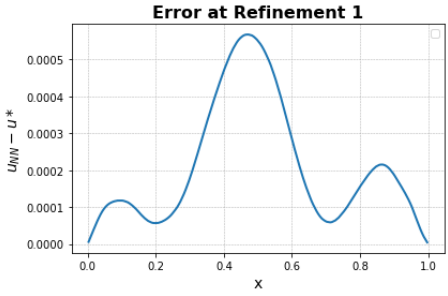
- We show the results of training below, with refinements marked in red.
- We see not only a reduction in the loss, but a reduction in the noise during training.

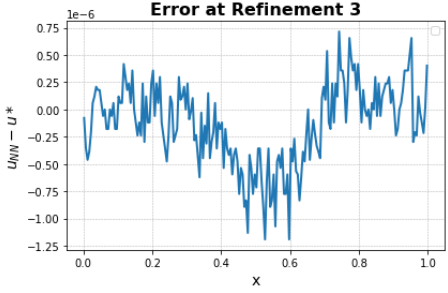




Multilevel NNs

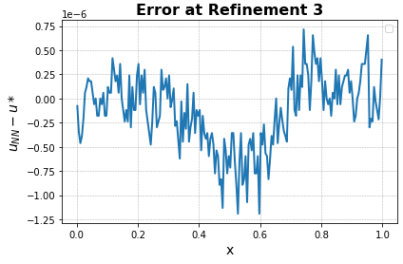


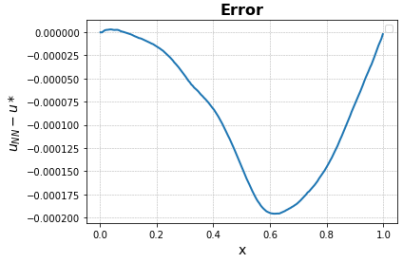


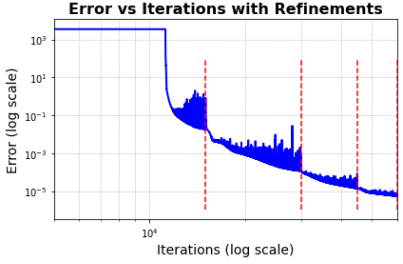


Comparison

- We can compare this with training the full neural network together.







Some concluding remarks on other strategies

- Going beyond gradient-based strategies can lead to significant gains in convergence.
- Neural networks suffer from spectral bias - low frequency features are learned easily, high frequency features are slower, if learned at all
- LS-based solvers can be a huge help, but it is unclear how they work with batches/data, and are sensitive to errors.
- Iterative strategies can train following the F-principle - this may include "forcing" high-frequency modes into your NN.