PEPR IA / PDE AI

Greedy training for neural networks

Applications to PINNs

Contents

PDE and numerical methods

Motivation

- **PDE modeling**: Most physical phenomena are modeled by implicit constraints on the desired function $u(t,x)$, known as a PDE (Partial Differential Equation).

$$\mathcal{N}(\partial_t u, \partial_x u, \partial_{xx} u) = f(t,x)$$

- **Simulations**: To simulate these phenomena, we use numerical methods to construct an approximation of the solution $u(t,x)$
- **ML and numerical methods**: Machine learning, like numerical methods, aims to approximate functions of the form $u(t,x)$ using a parametric model $u_\theta(t,x)$. ML approaches achieves this by solving

$$\min_\theta \sum_{i=1}^{N} d(u_\theta(t_i,x_i), u_i)$$

with a limited number N of data points, and numerical methods do so by solving

$$\min_\theta \sum_{i=1}^{N} d(\mathcal{N}(\partial_t u_\theta, \partial_x u_\theta, \partial_{xx} u_\theta)(t_i,x_i), f(t_i,x_i))$$

where the constraint can be evaluated at as many points as needed.

Classical vs Neural numerical methods

- Classical vs neural methods for spatial PDE like $-\Delta u = f$

- Approximation trial space:

$$V_n = \left\{ u_{\boldsymbol{\theta}}(\boldsymbol{x}), \quad \text{such that } u_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{n} \theta_i \varphi_i(\boldsymbol{x}) \right\}$$

- We solve:

$$J(\theta) = \min_{\boldsymbol{\theta}} \int_{\Omega} \sum_{i=1}^{n} \mid (-\Delta u_{\boldsymbol{\theta}}(\boldsymbol{x})) - f(\boldsymbol{x}))\psi_i(\boldsymbol{x}) \mid^2 \, d\boldsymbol{x}$$

with $W_n = \text{Span }(\psi_1, ..., \psi_n)$ the test space.

- Since the problem is quatradic in θ we solve it with normal equation.

- Approximation trial space:

$$V_n = \{ u_{\boldsymbol{\theta}}(\boldsymbol{x}), \quad \text{such that } u_{\boldsymbol{\theta}}(\boldsymbol{x}) = A_L \sigma(A_{l-1} ... + \boldsymbol{b}_{l-1}) + \boldsymbol{b}_l) \}$$

- We solve:

$$J(\theta) = \min_{\boldsymbol{\theta}} \int_{\Omega} \sum_{i=1}^{n} \mid (-\Delta u_{\boldsymbol{\theta}}(\boldsymbol{x})) - f(\boldsymbol{x}))\psi_i(\boldsymbol{x}) \mid^2 \, d\boldsymbol{x}$$

with $W_n = \text{Span }(\psi_1, ..., \psi_n)$ the test space.

- Since the problem is nonlinear we solve it using gradient methods and automatic differentiation.

- For time problems we can make the same with t a dimension like ther others.
- In general we prefer choose $\theta(t)$ and write a continuous time process which describes the evolution of the parameters.

Why Neural numerical methods ?

Result (Convergence): The set of numerical methods admits a result of this type:

$$\| u(\boldsymbol{x}) - u_\theta(\boldsymbol{x}) \| < C_{\text{pde}} C_u \left(\frac{1}{n}\right)^p$$

- The neural based methods (PINNs, discrete PINNs, Neural Galerkin) admit a limited accuracy and no convergence results.
- Why, in this case, use neural networks ?

Question (Dimension): In uncertainty propagation or optimal control problems, we aim to understand the influence of the parameter $\boldsymbol{\mu}$ of the PDE on the solution, thus capturing an approximation of $u(\boldsymbol{x}, \boldsymbol{\mu})$.

Result (Curse of dimensionality): We consider a problem of dimension d. We set a target error ε. The number of degrees of freedom (dof) is very roughly given by: $O(\frac{1}{\varepsilon^d})$.

Greedy Training for NNs and PINNs

How improve the performance ?

- The limiting point seems to be the optimization of the neural networks.
- Promising results have been obtained by using:
  - Preconditionning (Natural gradient, Gauss-Newton, Leverberg-Marquardt like methods):

$$\theta_{k+1} = \theta_k - A^{+}\nabla J(\theta_k)$$

with for example $A = \sum_i^N \nabla_\theta u(\boldsymbol{x}_i) \otimes \nabla_\theta u(\boldsymbol{x}_i)$.

  - Subspace/Least Square approaches: we see the network as a basis expension with Apdative basis functions

$$u_{\alpha,\theta}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \varphi_i(\boldsymbol{x}; \theta_i)$$

    Alternatively we project onto the basis (least square solver) and we adapt the basis (nonlinear optimization).

  - Greedy approach: As subspace approach we consider the network as a sum of adaptive basis functions. The basis functions are constructed one by one to minimize the error.

Greedy Algorithm

Definition (Greedy method): We consider a problem like:

$$u = \text{argmin}_{v \in V} \mathcal{E}(v)$$

We consider $\mathbb{D}$ a dictionary of functions (subspace of V). The greedy algorithm is:

- Initialization: $u_0 = \varphi_0$
- Iteration:

$$\varphi_k = \text{argmin}_{\varphi \in \mathbb{D}} \mathcal{E}(u_{k-1} + \varphi), \quad \text{and} \quad (\alpha_1, ...\alpha_n) = \text{argmin}_{\beta_1,...,\beta_n} \mathcal{E}\left(\sum_{i=1}^{n} \beta_i \varphi_i\right)$$

- update:

$$u_n = \sum_{i=1}^{n} \alpha_i \varphi_i$$

- The greedy algorithm is a sequential method in which we construct a sum of basis functions, each chosen to minimize the error of the previous approximation.

Greedy methods and PINNs

- **Difficulty**:
  - As the algorithm progresses, the error we aim to capture becomes smaller and corresponds to higher frequencies.
- **References**:
  - **Seigel and al**: Shallow Neural networks and convergence in $n^{-\frac{1}{2}}$. $O(100)$ steps.
  - **M. Ainsworth and al**: single hidden-layer NN with increasing number of neurons for the **high-frequency** capturing. $O(10)$ steps.
  - **Z. Aldirany and al**: Deep networks with fourier features for the **high-frequency** capturing. machine error with 4 networks.
  - **Y. Wang and al**: Deep networks with fourier features for the **high-frequency** capturing with heuristic for the frequencies choice machine error with 4 networks.
  - **J. Ng and al**: Deep networks with fourier features for the **high-frequency** capturing with FFT for the frequencies choice machine error with 2-3 networks.

Question (Greedy methods and PINNs):

- Use only for simple ellitpic problems. How extend it to more complex problems: nonlinear PDE, complex geometries.
- How extend the theoretical proofs ?

Theoretical results for deep networks

Result (Convergence (V. Ehrlacher)): We assume that the functional to minimize is strongly convex. If $\mathbb{D}$ the dictionary satisfy:

- Span($\mathbb{D}$) is dense in V (the functional space of the solution like $H^1(\Omega)$)
- $\mathbb{D}$ is weakly closed in V
- $\forall \lambda \in \mathbb{R}, z \in \mathbb{D}$ then $\lambda z \in \mathbb{D}$

the the sequence $(u_n)_{n\in\mathbb{N}^*}$ converge toward the solution u.

- If we cannot have the second condition we can add a Ridge penalization of the θ_n parameters where $\mathbb{D} = \{f_\theta(\boldsymbol{x}), \text{such that } \theta \in U \subset \mathbb{R}^n\}$

Remark: The main point in the density of Span($\mathbb{D}$).

Numerical results for deep networks

Result (Seigel): The shallows networks are dense in $H^m(\Omega)$ if $v(x) = \sigma(x+1) - \sigma(x)$ with σ the activation function admit a polynomial decay at infinity.

Result (L. Navoret, V. Ehrlacher, E; Franck, V. Michel-Dansac): We consider the space of deep neural networks with a specified archicture and L hidden layer and classical activation as tanh or sinus is dense in $H^m(\Omega)$

- There exist a set of weights that all network

$$u_\theta = \sum_i^n \alpha_i \sigma^L(\langle \boldsymbol{\theta}_i, \boldsymbol{x}\rangle + \boldsymbol{b}_i)$$

with σ^L the composition of all the activation functions of the deep network.

- So the Span of the deep network contains the space of Shallows networks associated σ^L
- For many classical activation functions σ^L satisfy the condition of Seigel
- Therefore, we have the density of the deep network in $H^m(\Omega)$ using Siegel's results.

Results Laplacian I

- **PDE**: 2D laplacian + 2D parametric source term

$$-\Delta u = e^{-\frac{(x_1-\mu_1)^2+(x_2-\mu_2)^2}{2\sigma^2}}$$

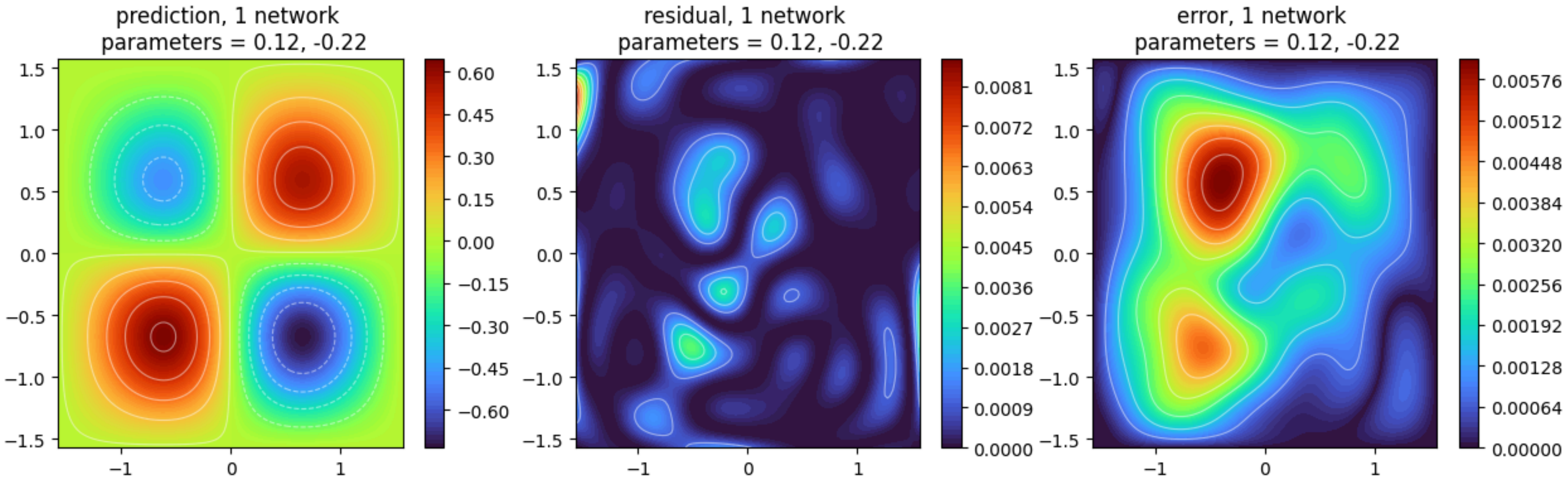

Figure 1: Network used: First step

Results Laplacian II

- **PDE**: 2D laplacian + 2D parametric source term

$$-\Delta u = e^{-\frac{(x_1-\mu_1)^2+(x_2-\mu_2)^2}{2\sigma^2}}$$

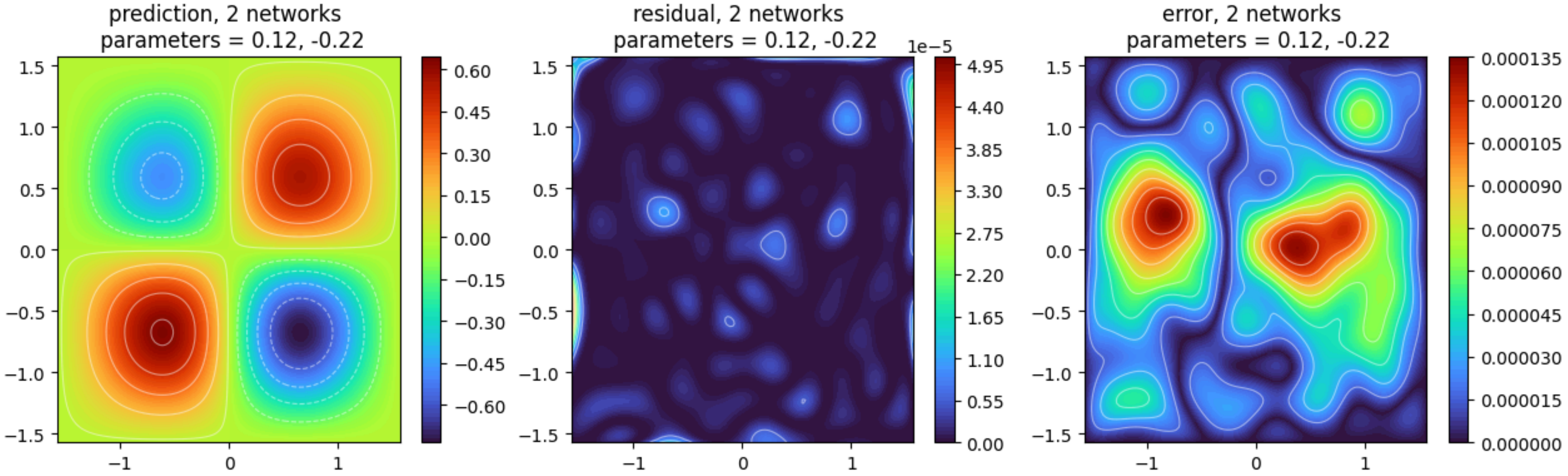

Figure 2: Network used: Second step

Results Laplacian III

- **PDE**: 2D laplacian + 2D parametric source term

$$-\Delta u = e^{-\frac{(x_1-\mu_1)^2+(x_2-\mu_2)^2}{2\sigma^2}}$$

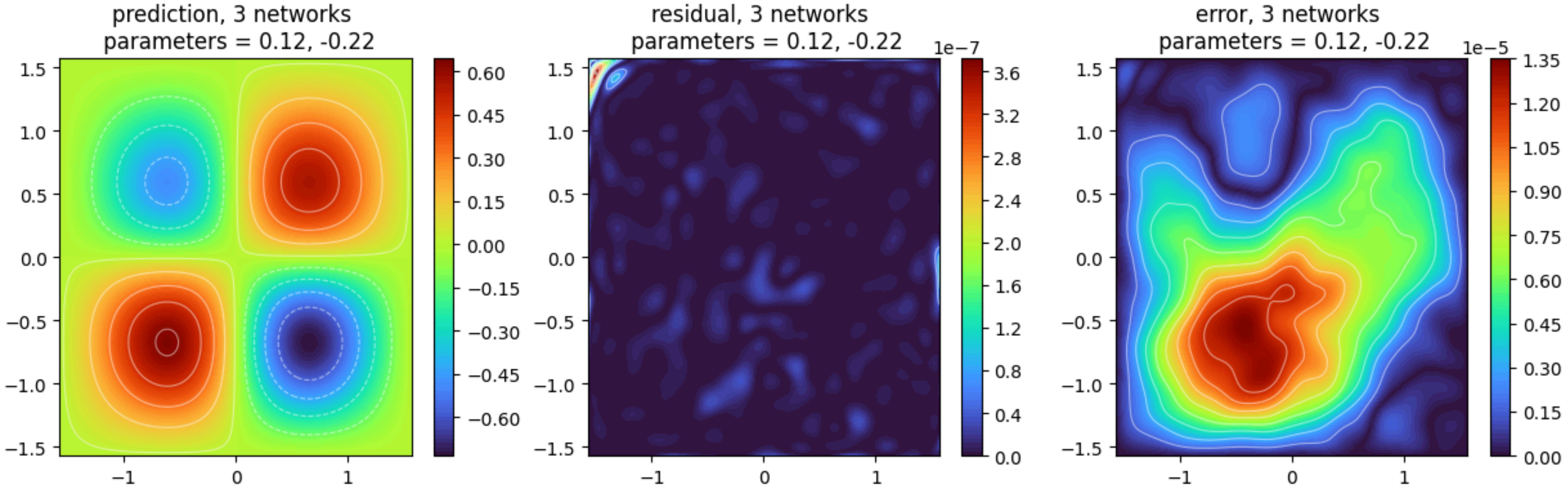

Figure 3: Network used: Third step

Results Laplacian IV

- **PDE**: 2D laplacian + 2D parametric source term

$$-\Delta u = e^{-\frac{(x_1-\mu_1)^2+(x_2-\mu_2)^2}{2\sigma^2}}$$

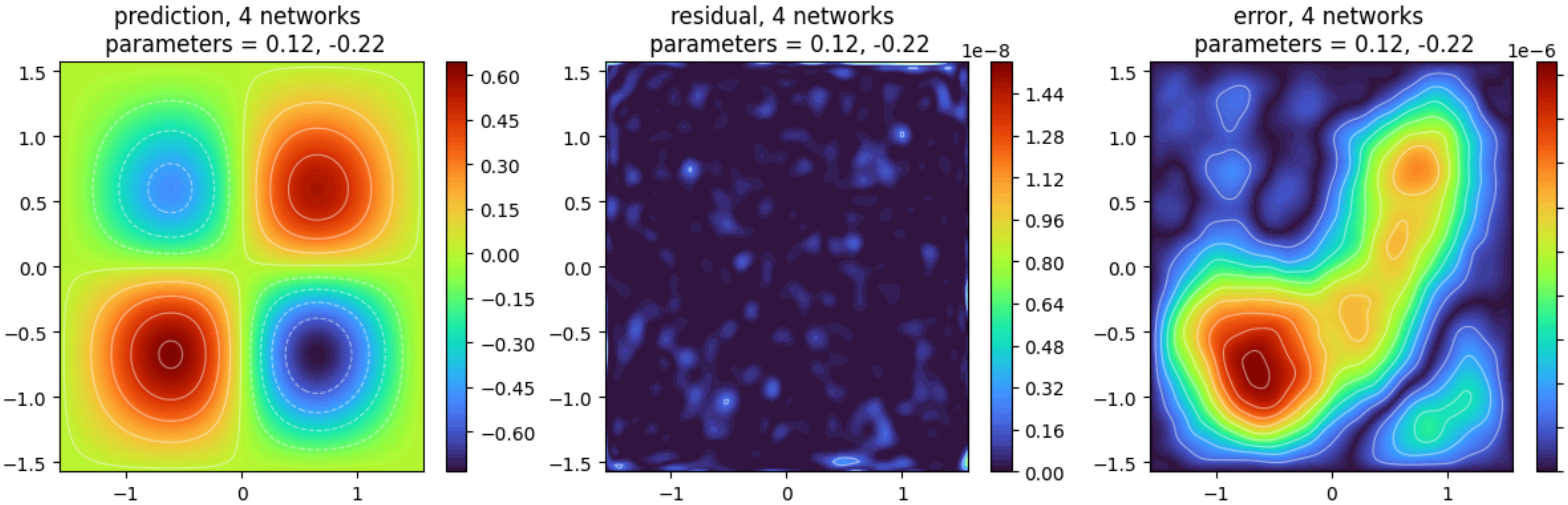

Figure 4: Network used: Fourth step

Results Grad-Shafranov I

- **PDE**: 2D linear Grad-Shafranov (Plasma tokamak equilibrium) + 1D parametric source term

$$-\partial_{rr}\psi + \frac{1}{r}\partial_r\psi - \partial_{zz}\psi = e^{f_0}(r^2 + r_0^2)$$

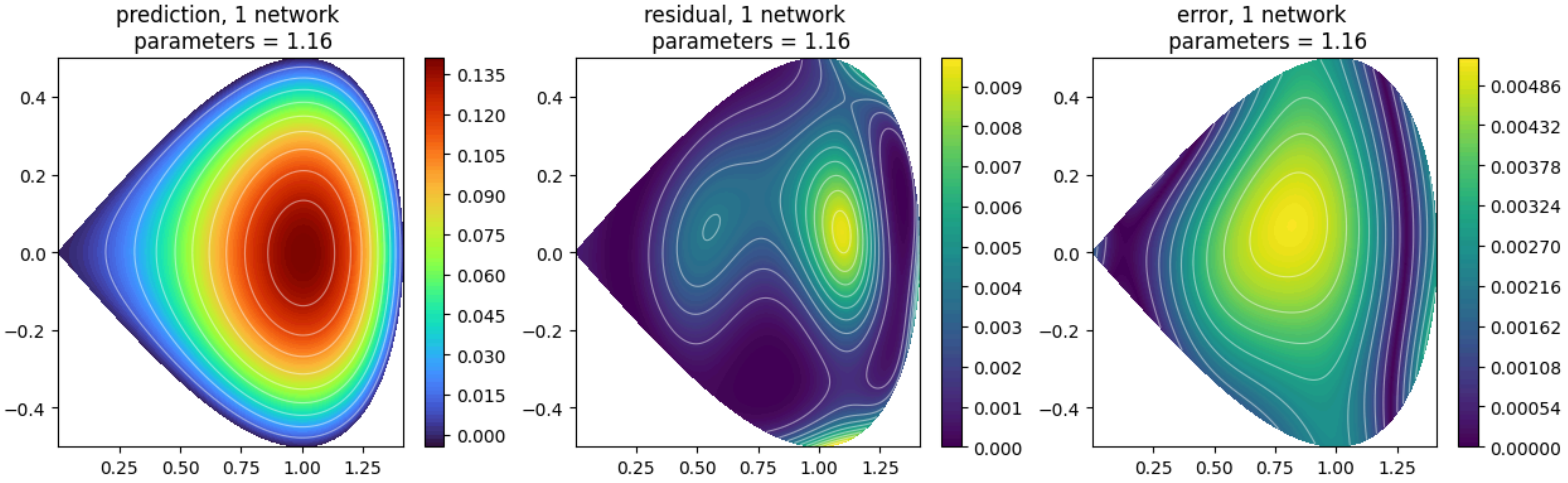

Figure 5: Network used: First step

Results Grad-Shafranov II

- **PDE**: 2D linear Grad-Shafranov (Plasma tokamak equilibrium) + 1D parametric source term

$$-\partial_{rr}\psi + \frac{1}{r}\partial_r\psi - \partial_{zz}\psi = e^{f_0}(r^2 + r_0^2)$$

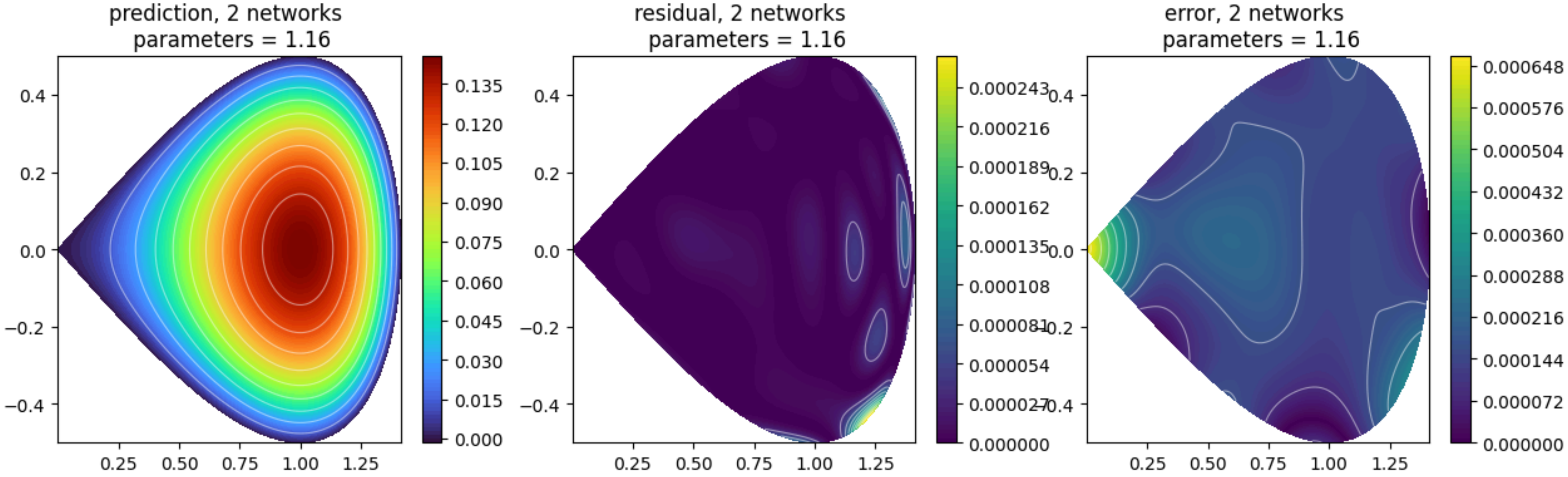

Figure 6: Network used: Second step

Results Grad-Shafranov III

- **PDE**: 2D linear Grad-Shafranov (Plasma tokamak equilibrium) + 1D parametric source term

$$-\partial_{rr}\psi + \frac{1}{r}\partial_r\psi - \partial_{zz}\psi = e^{f_0}(r^2 + r_0^2)$$

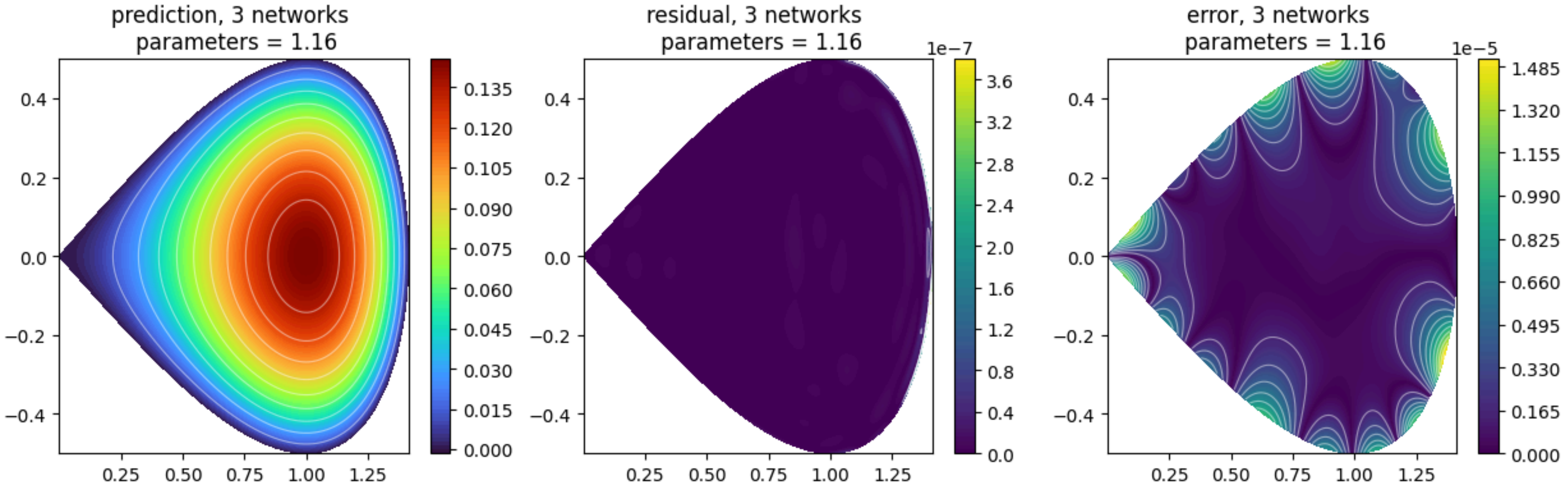

Figure 7: Network used: Third step

Results Grad-Shafranov IV

- **PDE**: 2D linear Grad-Shafranov (Plasma tokamak equilibrium) + 1D parametric source term

$$-\partial_{rr}\psi + \frac{1}{r}\partial_r\psi - \partial_{zz}\psi = e^{f_0}(r^2 + r_0^2)$$

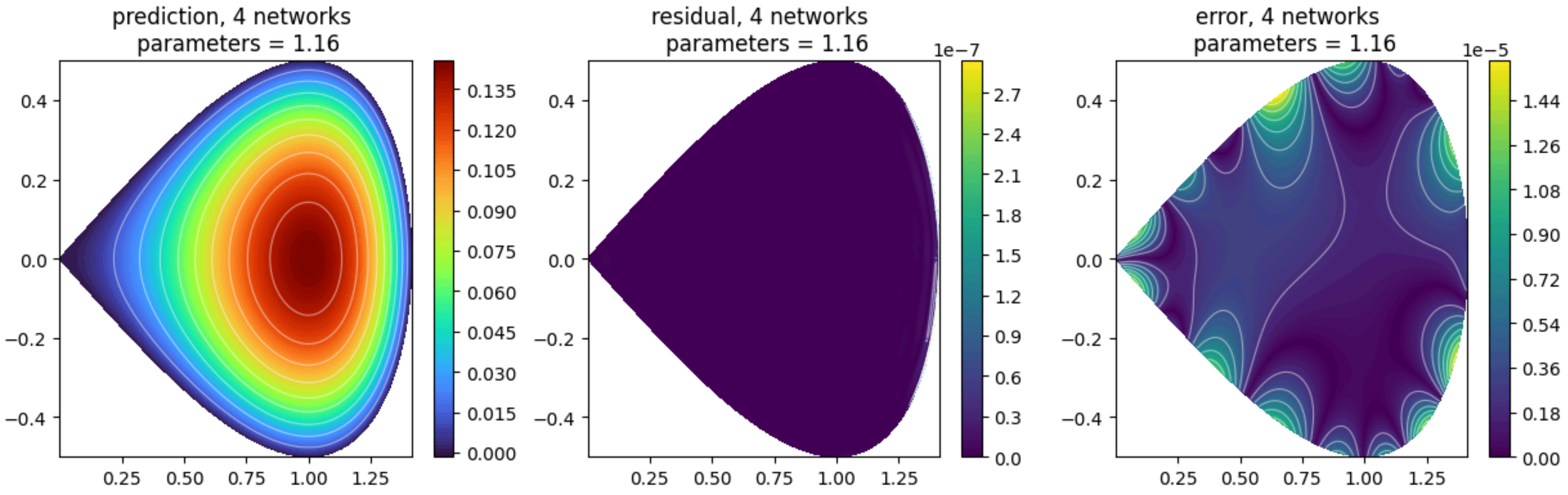

Figure 8: Network used: Fourth step

Next steps and PEPR IA

Project in the PEPR IA

- Post doc of F. Salin (beginning first april 2025).
- ▸ **Step 1**:
  - Extend the proof of convergence with error estimates for greedy methods applied to shallow networks with Fourier features.
  - Propose an efficient strategy for complex geometries to initialize the frequencies of Fourier features.
  - Extension to one-hidden-layer networks ?
  - Couple Greedy methods with natural gradient for each step.

  ▸ **Step 2**
  - We conside high-dimensional transport equations with a neural Semi-Lagrangian scheme (in redaction paper):

$$\theta_{k+1} = \text{argmin}_\theta \sum_{i=1}^{N} \parallel u_\theta(\boldsymbol{x}_i) - u_{\theta_n}(\boldsymbol{x}_i - \boldsymbol{v}\Delta t) \parallel^2$$

  - Coupling this method with the greedy projection.
  - Demonstrate the convergence of the greedy method for this problem.
  - Applications: Hamilton-Jacobi-Bellman equation (shape optimization, continuous RL), Vlasov equation (Plasma), Radiative transfer.

Neural operators and greedy methods

Neural Operator

- We consider a PDE problem like:

$$-\Delta u = f$$

Definition (Neural Operator): A neural operator is a neural network that approximates operators like $-\Delta$. It takes as input the function f and output the function u.

- In practice we work with numerical approximations of u and f
- We speak about Neural operator where the result is independent of the resolution and possibly the discretization of the inputs and outputs.

Definition (Continuous neural operator layer): We consider $\boldsymbol{v}_{l(x)} \in \mathbb{R}^{d_l}$ and $\boldsymbol{v}_{l+1}(x) \in \mathbb{R}^{d_{l+1}}$. A layer of neural operator is given by:

$$\boldsymbol{v}_{l+1}(x) = \sigma\left(W\boldsymbol{v}_l(x) + \int_\Omega K_l(x, y, \boldsymbol{v}_l(x), \boldsymbol{v}_l(y))\boldsymbol{v}_l(y)dy + \boldsymbol{b}_l(x)\right)$$

with W, $\boldsymbol{b}_l$ and K_l are learnable.

Neural Operator and Greedy methods

- Simpler case: the GreenNet which is a single linear layer neural operator:

$$\boldsymbol{v}_{l+1}(x) = \int_\Omega K_\theta(x,y)\boldsymbol{v}_l(y)dy + \boldsymbol{b}_l(x)$$

with K_θ is a MLP or similar network and the integration is discretized using Monte Carlo.

Objective (Greedy methods for neural Operator): A first result with randomized neural networks and greedy methods for the construction of K was obtained. We want extend this to shallow and single hidden NNs with theoretical results.

- It will be also interesting to consider numerically deeper neural operators and coupling these with greedy methods.

Conclusion

Conclusion

- **Greedy** methods are a promising approach to improve the performance of neural networks for PDEs.
- **Theoretical** results are available for shallow networks and we obtain partial results for deep networks.

Objective: Provide more theoretical results with error estimates

Objective: Extend the methodology to time-evolutionary neural networks and neural operators.

Objective: Find automatic way to choose the frequencies of the Fourier features and other hyper-parameters.