

High-Dimensional Approximation in AI

Lecture 2: AI, High-Dimensional PDEs, and a Look into the Future

Gitta Kutyniok (LMU Munich)

also

University of Tromsø & DLR – German Aerospace Center

CIME School on High-Dimensional Approximation
Cetraro, Italy, September 22 –27, 2024



Bavarian AI Chair for
Mathematical Foundations
of Artificial Intelligence

Some Facts about PDE Solvers:

- ▶ Precise physical models exist.
- ▶ The discretization process is very well understood.
- ▶ Often optimal solvers are available.
- ▶ A rich bouquet of highly sophisticated solvers are developed:
 - ▶ Finite-element methods
 - ▶ Wavelet-based approaches
 - ▶ ...

Some Facts about PDE Solvers:

- ▶ Precise physical models exist.
- ▶ The discretization process is very well understood.
- ▶ Often optimal solvers are available.
- ▶ A rich bouquet of highly sophisticated solvers are developed:
 - ▶ Finite-element methods
 - ▶ Wavelet-based approaches
 - ▶ ...

Why do we need deep neural networks?

Some Facts about PDE Solvers:

- ▶ Precise physical models exist.
- ▶ The discretization process is very well understood.
- ▶ Often optimal solvers are available.
- ▶ A rich bouquet of highly sophisticated solvers are developed:
 - ▶ Finite-element methods
 - ▶ Wavelet-based approaches
 - ▶ ...

Why do we need deep neural networks?

~> Deep neural networks can *beat the curse of dimensionality in high dimensional problems!*

Deep Learning Approaches to PDEs

Common Approach to Solve PDEs with Neural Networks:

Approximate the solution u of a PDE

$$\mathcal{L}(u) = f$$

by a neural network Φ , i.e., determine

$$\mathcal{L}(\Phi) \approx f.$$

Key Ideas:

- ▶ Sampling of points in the spatial domain
- ▶ Incorporate PDE in the loss functions

Incomplete List of Contributions:

[Lagaris, Likas, Fotiadis; 1998], [E, Yu; 2017], [Czarnecki, Osindero, Jaderberg, Swirszcz, Pascanu; 2017], [Sirignano, Spiliopoulos; 2017], [Han, Jentzen, E; 2017], [Raissi, Perdikaris, Karniadakis; 2020], [Grohs, Herrmann; 2021],

Let's Now Enter the World of Parametric PDEs

Why Parametric PDEs?

Parameter dependent families of PDEs arise in basically any branch of science and engineering.

Some Exemplary Problem Classes:

- ▶ Complex design problems
- ▶ Inverse problems
- ▶ Optimization tasks
- ▶ Uncertainty quantification
- ▶ ...



The number of parameters can be

- ▶ finite (physical properties such as domain geometry, ...)
- ▶ infinite (modeling of random stochastic diffusion field, ...)

Parametric Map:

$$\mathcal{Y} \ni y \mapsto u_y \in \mathcal{H} \quad \text{such that} \quad \mathcal{L}(u_y, y) = f_y.$$

Parametric Partial Differential Equations

Our Setting: We will consider parameter-dependent equations of the form

$$b_y(u_y, v) = f_y(v), \quad \text{for all } y \in \mathcal{Y}, \quad v \in \mathcal{H},$$

where

- ▶ $\mathcal{Y} \subseteq \mathbb{R}^p$ (p large) is the *compact parameter set*,
- ▶ \mathcal{H} is a Hilbert space,
- ▶ $b_y: \mathcal{H} \times \mathcal{H} \rightarrow \mathbb{R}$ is a *symmetric, uniformly coercive, and uniformly continuous bilinear form*,
- ▶ $f_y \in \mathcal{H}^*$ is the *uniformly bounded, parameter-dependent right-hand side*,
- ▶ $u_y \in \mathcal{H}$ is the *solution*.

We assume the *solution manifold*

$$S(\mathcal{Y}) := \{u_y : y \in \mathcal{Y}\}$$

to be compact in \mathcal{H} .

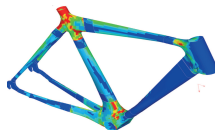
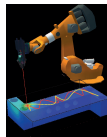
Multi-Query Situation

Many applications require solving the parametric PDE multiple times for *different* parameters:

$$\mathbb{R}^p \supset \mathcal{Y} \ni y = (y_1, \dots, y_p) \mapsto u_y \in \mathcal{H}$$

Examples:

- ▶ Design optimization
- ▶ Optimal control
- ▶ Routine analysis
- ▶ Uncertainty quantification
- ▶ Inverse problems



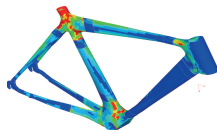
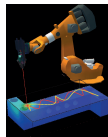
Multi-Query Situation

Many applications require solving the parametric PDE multiple times for *different* parameters:

$$\mathbb{R}^p \supset \mathcal{Y} \ni y = (y_1, \dots, y_p) \mapsto u_y \in \mathcal{H}$$

Examples:

- ▶ Design optimization
- ▶ Optimal control
- ▶ Routine analysis
- ▶ Uncertainty quantification
- ▶ Inverse problems



Curse of Dimensionality:

Computational cost often much too high!

High-Fidelity Approximations

Galerkin Approach: Instead of $b_y(u_y, v) = f_y(v)$, we solve

$$b_y(u_y^h, v) = f_y(v) \quad \text{for all } v \in U^h,$$

where $U^h \subset \mathcal{H}$ with $D := \dim(U^h) < \infty$ is the *high-fidelity discretization* and $u_y^h \in U^h$ is the solution.

Cea's Lemma: u_y^h is (up to a constant) a best approximation of u_y by elements in U^h .

High-Fidelity Approximations

Galerkin Approach: Instead of $b_y(u_y, v) = f_y(v)$, we solve

$$b_y(u_y^h, v) = f_y(v) \quad \text{for all } v \in U^h,$$

where $U^h \subset \mathcal{H}$ with $D := \dim(U^h) < \infty$ is the *high-fidelity discretization* and $u_y^h \in U^h$ is the solution.

Cea's Lemma: u_y^h is (up to a constant) a best approximation of u_y by elements in U^h .

Galerkin Solution: Let $(\varphi_i)_{i=1}^D$ be a basis for U^h . Then u_y^h satisfies

$$u_y^h = \sum_{i=1}^D (\mathbf{u}_y^h)_i \varphi_i \quad \text{with} \quad \mathbf{u}_y^h := (\mathbf{B}_y^h)^{-1} \mathbf{f}_y^h \in \mathbb{R}^D,$$

where $\mathbf{B}_y^h := (b_y(\varphi_j, \varphi_i))_{i,j=1}^D$ and $\mathbf{f}_y^h := (f_y(\varphi_i))_{i=1}^D$.

What about Deep Neural Networks?

Parametric Map:

$$\mathcal{Y} \ni y \mapsto \mathbf{u}_y^h \in \mathbb{R}^D \quad \text{such that} \quad b_y(u_y^h, v) = f_y(v) \quad \forall v \in U^h.$$

Can a Neural Network Approximate the Parametric Map?

What about Deep Neural Networks?

Parametric Map:

$$\mathcal{Y} \ni y \mapsto \mathbf{u}_y^h \in \mathbb{R}^D \quad \text{such that} \quad b_y(u_y^h, v) = f_y(v) \quad \forall v \in U^h.$$

Can a Neural Network Approximate the Parametric Map?

Advantages:

- ▶ After training, extremely rapid computation of the map.
- ▶ Flexible, universal approach.

Questions: Let $\epsilon > 0$.

(1) Does there exist a neural network Φ such that

$$\|\Phi - \mathbf{u}_y^h\| \leq \epsilon \quad \text{for all } y \in \mathcal{Y}$$

(2) How does the complexity of Φ depend on p and D ?

(3) How do neural networks perform numerically on this task?

Deep Learning Approaches to (Parametric) PDEs

Solving PDEs with Neural Networks:

- ▶ Lagaris, Likas, Fotiadis; 1998
- ▶ E, Yu; 2017
- ▶ Sirignano, Spiliopoulos; 2017
- ▶ Han, Jentzen, E; 2017
- ▶ Berner, Grohs, Jentzen; 2018
- ▶ Regazzoni, Dedè, Quarteroni; 2019
- ▶ Reisinger, Zhang; 2019
- ▶ ...

Deep Learning Approaches to (Parametric) PDEs

Solving PDEs with Neural Networks:

- ▶ Lagaris, Likas, Fotiadis; 1998
- ▶ E, Yu; 2017
- ▶ Sirignano, Spiliopoulos; 2017
- ▶ Han, Jentzen, E; 2017
- ▶ Berner, Grohs, Jentzen; 2018
- ▶ Regazzoni, Dedè, Quarteroni; 2019
- ▶ Reisinger, Zhang; 2019
- ▶ ...

Solving Parametric PDEs with Neural Networks:

- ▶ K. Lee, K. Carlberg; 2018
Learn a parametrization of $S(\mathcal{Y})$ represented by neural networks.
- ▶ J.S. Hesthaven, S. Ubbiali; 2018
Find reduced basis and then train neural networks to predict coefficients of solution in that basis.
- ▶ Schwab, Zech; 2018
Assume that there is a reduced basis of polynomial chaos functions. These and the coefficients can be efficiently represented by neural networks.

Approximation of the Parametric Map
by Deep Neural Networks

What about Deep Neural Networks?

Parametric Map:

$$\mathcal{Y} \ni y \mapsto \mathbf{u}_y^h \in \mathbb{R}^D \quad \text{such that} \quad b_y(u_y^h, v) = f_y(v) \quad \forall v \in U^h.$$

Can a Neural Network Approximate the Parametric Map?

Advantages:

- ▶ After training, extremely rapid computation of the map.
- ▶ Flexible, universal approach.

Questions: Let $\epsilon > 0$.

(1) Does there exist a **ReLU** neural network Φ such that

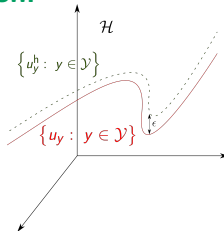
$$\|\Phi - \mathbf{u}_y^h\| \leq \epsilon \quad \text{for all } y \in \mathcal{Y}?$$

(2) How does the complexity of Φ depend on p and D ?

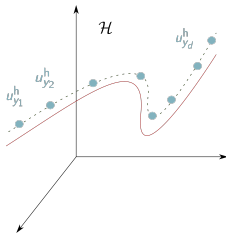
(3) How do neural networks perform numerically on this task?

Reduced Basis Method: Key Ideas

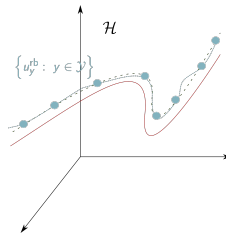
High-Fidelity Discretization:



Key Idea:



Offline (slow):
Compute snap shots



Online (fast):
Compute solutions for new parameters

Reduced Basis Method: Details

Assumption: For all $\epsilon > \epsilon_0$, there exists $U^{\text{rb}} \subset \mathcal{H}$,
 $d(\epsilon) := \dim(U^{\text{rb}}) \ll D$ such that

$$\sup_{y \in \mathcal{Y}} \inf_{w \in U^{\text{rb}}} \|u_y - w\|_{\mathcal{H}} \leq \epsilon.$$

\leadsto Optimality through *Kolmogorov N-width*!

Reduced Basis Method: Details

Assumption: For all $\epsilon > \epsilon_0$, there exists $U^{\text{rb}} \subset \mathcal{H}$,
 $d(\epsilon) := \dim(U^{\text{rb}}) \ll D$ such that

$$\sup_{y \in \mathcal{Y}} \inf_{w \in U^{\text{rb}}} \|u_y - w\|_{\mathcal{H}} \leq \epsilon.$$

\leadsto Optimality through *Kolmogorov N-width*!

Transfer to Reduced Basis:

► Let $U^{\text{rb}} := \text{span}(\psi_i)_{i=1}^{d(\epsilon)}$ with $(\psi_i)_{i=1}^{d(\epsilon)} = \left(\sum_{j=1}^D \mathbf{v}_{j,i} \varphi_j \right)_{i=1}^{d(\epsilon)}$.

Reduced Basis Method: Details

Assumption: For all $\epsilon > \epsilon_0$, there exists $U^{\text{rb}} \subset \mathcal{H}$,
 $d(\epsilon) := \dim(U^{\text{rb}}) \ll D$ such that

$$\sup_{y \in \mathcal{Y}} \inf_{w \in U^{\text{rb}}} \|u_y - w\|_{\mathcal{H}} \leq \epsilon.$$

\leadsto Optimality through *Kolmogorov N-width*!

Transfer to Reduced Basis:

- ▶ Let $U^{\text{rb}} := \text{span}(\psi_i)_{i=1}^{d(\epsilon)}$ with $(\psi_i)_{i=1}^{d(\epsilon)} = \left(\sum_{j=1}^D \mathbf{v}_{j,i} \varphi_j \right)_{i=1}^{d(\epsilon)}$.
- ▶ Set $\mathbf{B}_y^{\text{rb}} := (b_y(\psi_j, \psi_i))_{i,j=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{B}_y^{\text{h}} \mathbf{V} \in \mathbb{R}^{d(\epsilon) \times d(\epsilon)}$.
- ▶ Set $\mathbf{f}_y^{\text{rb}} := (f_y(\psi_i))_{i=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{f}_y^{\text{h}} \in \mathbb{R}^{d(\epsilon)}$.

Reduced Basis Method: Details

Assumption: For all $\epsilon > \epsilon_0$, there exists $U^{\text{rb}} \subset \mathcal{H}$, $d(\epsilon) := \dim(U^{\text{rb}}) \ll D$ such that

$$\sup_{y \in \mathcal{Y}} \inf_{w \in U^{\text{rb}}} \|u_y - w\|_{\mathcal{H}} \leq \epsilon.$$

\leadsto Optimality through *Kolmogorov N-width*!

Transfer to Reduced Basis:

- ▶ Let $U^{\text{rb}} := \text{span}(\psi_i)_{i=1}^{d(\epsilon)}$ with $(\psi_i)_{i=1}^{d(\epsilon)} = \left(\sum_{j=1}^D \mathbf{V}_{j,i} \varphi_j \right)_{i=1}^{d(\epsilon)}$.
- ▶ Set $\mathbf{B}_y^{\text{rb}} := (b_y(\psi_j, \psi_i))_{i,j=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{B}_y^{\text{h}} \mathbf{V} \in \mathbb{R}^{d(\epsilon) \times d(\epsilon)}$.
- ▶ Set $\mathbf{f}_y^{\text{rb}} := (f_y(\psi_i))_{i=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{f}_y^{\text{h}} \in \mathbb{R}^{d(\epsilon)}$.

Galerkin Solution: $(\sup_{y \in \mathcal{Y}} \|u_y - u_y^{\text{rb}}\|_{\mathcal{H}} \leq C\epsilon)$

$$u_y^{\text{rb}} =$$

Reduced Basis Method: Details

Assumption: For all $\epsilon > \epsilon_0$, there exists $U^{\text{rb}} \subset \mathcal{H}$, $d(\epsilon) := \dim(U^{\text{rb}}) \ll D$ such that

$$\sup_{y \in \mathcal{Y}} \inf_{w \in U^{\text{rb}}} \|u_y - w\|_{\mathcal{H}} \leq \epsilon.$$

\leadsto Optimality through *Kolmogorov N-width*!

Transfer to Reduced Basis:

- ▶ Let $U^{\text{rb}} := \text{span}(\psi_i)_{i=1}^{d(\epsilon)}$ with $(\psi_i)_{i=1}^{d(\epsilon)} = \left(\sum_{j=1}^D \mathbf{V}_{j,i} \varphi_j \right)_{i=1}^{d(\epsilon)}$.
- ▶ Set $\mathbf{B}_y^{\text{rb}} := (b_y(\psi_j, \psi_i))_{i,j=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{B}_y^{\text{h}} \mathbf{V} \in \mathbb{R}^{d(\epsilon) \times d(\epsilon)}$.
- ▶ Set $\mathbf{f}_y^{\text{rb}} := (f_y(\psi_i))_{i=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{f}_y^{\text{h}} \in \mathbb{R}^{d(\epsilon)}$.

Galerkin Solution: $(\sup_{y \in \mathcal{Y}} \|u_y - u_y^{\text{rb}}\|_{\mathcal{H}} \leq C\epsilon)$

$$u_y^{\text{rb}} = \sum_{i=1}^{d(\epsilon)} \left(\mathbf{u}_y^{\text{rb}} \right)_i \psi_i =$$

Reduced Basis Method: Details

Assumption: For all $\epsilon > \epsilon_0$, there exists $U^{\text{rb}} \subset \mathcal{H}$, $d(\epsilon) := \dim(U^{\text{rb}}) \ll D$ such that

$$\sup_{y \in \mathcal{Y}} \inf_{w \in U^{\text{rb}}} \|u_y - w\|_{\mathcal{H}} \leq \epsilon.$$

\leadsto Optimality through *Kolmogorov N-width*!

Transfer to Reduced Basis:

- ▶ Let $U^{\text{rb}} := \text{span}(\psi_i)_{i=1}^{d(\epsilon)}$ with $(\psi_i)_{i=1}^{d(\epsilon)} = \left(\sum_{j=1}^D \mathbf{V}_{j,i} \varphi_j \right)_{i=1}^{d(\epsilon)}$.
- ▶ Set $\mathbf{B}_y^{\text{rb}} := (b_y(\psi_j, \psi_i))_{i,j=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{B}_y^{\text{h}} \mathbf{V} \in \mathbb{R}^{d(\epsilon) \times d(\epsilon)}$.
- ▶ Set $\mathbf{f}_y^{\text{rb}} := (f_y(\psi_i))_{i=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{f}_y^{\text{h}} \in \mathbb{R}^{d(\epsilon)}$.

Galerkin Solution: $(\sup_{y \in \mathcal{Y}} \|u_y - u_y^{\text{rb}}\|_{\mathcal{H}} \leq C\epsilon)$

$$u_y^{\text{rb}} = \sum_{i=1}^{d(\epsilon)} (\mathbf{u}_y^{\text{rb}})_i \psi_i = \sum_{j=1}^D (\mathbf{V} \mathbf{u}_y^{\text{rb}})_j \varphi_j =$$

Reduced Basis Method: Details

Assumption: For all $\epsilon > \epsilon_0$, there exists $U^{\text{rb}} \subset \mathcal{H}$,
 $d(\epsilon) := \dim(U^{\text{rb}}) \ll D$ such that

$$\sup_{y \in \mathcal{Y}} \inf_{w \in U^{\text{rb}}} \|u_y - w\|_{\mathcal{H}} \leq \epsilon.$$

\leadsto Optimality through *Kolmogorov N-width*!

Transfer to Reduced Basis:

- ▶ Let $U^{\text{rb}} := \text{span}(\psi_i)_{i=1}^{d(\epsilon)}$ with $(\psi_i)_{i=1}^{d(\epsilon)} = \left(\sum_{j=1}^D \mathbf{V}_{j,i} \varphi_j \right)_{i=1}^{d(\epsilon)}$.
- ▶ Set $\mathbf{B}_y^{\text{rb}} := (b_y(\psi_j, \psi_i))_{i,j=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{B}_y^{\text{h}} \mathbf{V} \in \mathbb{R}^{d(\epsilon) \times d(\epsilon)}$.
- ▶ Set $\mathbf{f}_y^{\text{rb}} := (f_y(\psi_i))_{i=1}^{d(\epsilon)} = \mathbf{V}^T \mathbf{f}_y^{\text{h}} \in \mathbb{R}^{d(\epsilon)}$.

Galerkin Solution: $(\sup_{y \in \mathcal{Y}} \|u_y - u_y^{\text{rb}}\|_{\mathcal{H}} \leq C\epsilon)$

$$u_y^{\text{rb}} = \sum_{i=1}^{d(\epsilon)} (\mathbf{u}_y^{\text{rb}})_i \psi_i = \sum_{j=1}^D (\mathbf{V} \mathbf{u}_y^{\text{rb}})_j \varphi_j = \sum_{j=1}^D \left(\mathbf{V} (\mathbf{B}_y^{\text{rb}})^{-1} \mathbf{V}^T \mathbf{f}_y^{\text{h}} \right)_j \varphi_j$$

Our Results: Discrete Version

Theorem (K, Petersen, Raslan, Schneider; 2019):

We assume the following:

- For all $\epsilon > 0$, there exists $d(\epsilon) \ll D$, $\mathbf{V} \in \mathbb{R}^{D \times d(\epsilon)}$, such that for all $y \in \mathcal{Y}$ there exists $\mathbf{B}_y^{\text{rb}} \in \mathbb{R}^{d(\epsilon) \times d(\epsilon)}$ with

$$\|\mathbf{V}(\mathbf{B}_y^{\text{rb}})^{-1}\mathbf{V}^T\mathbf{f}_y^{\text{h}} - \mathbf{u}_y^{\text{h}}\| \leq \epsilon.$$

- There exist ReLU neural networks Φ^B and Φ^f of size $O(\text{poly}(p)d(\epsilon)^2\text{polylog}(\epsilon))$ such that, for all $y \in \mathcal{Y}$,

$$\|\Phi^B - \mathbf{B}_y^{\text{rb}}\| \leq \epsilon \quad \text{and} \quad \|\Phi^f - \mathbf{f}_y^{\text{rb}}\| \leq \epsilon.$$

Then there exists a ReLU neural network Φ of size $O(d(\epsilon)^3\text{polylog}(\epsilon) + D + \text{poly}(p)d(\epsilon)^2\text{polylog}(\epsilon))$ such that

$$\|\Phi - \mathbf{u}_y^{\text{h}}\| \leq \epsilon \quad \text{for all } y \in \mathcal{Y}.$$

Our Results: Continuous Version

Theorem (K, Petersen, Raslan, Schneider; 2019):

Let $(\psi_i)_{i=1}^{d(\epsilon)}$ denote the reduced basis. We assume in addition the following:

- There exist ReLU neural networks $(\Phi_i)_{i=1}^{d(\epsilon)}$ of size $O(\text{polylog}(\epsilon))$ such that $\|\Phi_i - \psi_i\|_{\mathcal{H}} \leq \epsilon$ for all $i = 1, \dots, d(\epsilon)$.

Then there exists a ReLU neural network Φ of size $O(d(\epsilon)^3 \text{polylog}(\epsilon) + \text{poly}(p)d(\epsilon)^2 \text{polylog}(\epsilon))$ such that

$$\|\Phi - u_y\|_{\mathcal{H}} \leq \epsilon \quad \text{for all } y \in \mathcal{Y}.$$

Our Results: Continuous Version

Theorem (K, Petersen, Raslan, Schneider; 2019):

Let $(\psi_i)_{i=1}^{d(\epsilon)}$ denote the reduced basis. We assume in addition the following:

- ▶ There exist ReLU neural networks $(\Phi_i)_{i=1}^{d(\epsilon)}$ of size $O(\text{polylog}(\epsilon))$ such that $\|\Phi_i - \psi_i\|_{\mathcal{H}} \leq \epsilon$ for all $i = 1, \dots, d(\epsilon)$.

Then there exists a ReLU neural network Φ of size $O(d(\epsilon)^3 \text{polylog}(\epsilon) + \text{poly}(p)d(\epsilon)^2 \text{polylog}(\epsilon))$ such that

$$\|\Phi - u_y\|_{\mathcal{H}} \leq \epsilon \quad \text{for all } y \in \mathcal{Y}.$$

Remark: The hypotheses are fulfilled, for example, by

- ▶ Diffusion equations,
- ▶ Linear elasticity equations.

Key Idea of the Proof

Main Task: Approximate $\mathbf{V}(\mathbf{B}_y^{\text{rb}})^{-1}\mathbf{V}^T\mathbf{f}_y^{\text{h}}$ by a ReLU neural network and control its size!

Key Idea of the Proof

Main Task: Approximate $\mathbf{V}(\mathbf{B}_y^{\text{rb}})^{-1}\mathbf{V}^T\mathbf{f}_y^{\text{h}}$ by a ReLU neural network and control its size!

Step 1 (Scalar Multiplication from Yarotsky; 2017):

For $g(x) := \min\{2x, 2 - 2x\}$ and $g_s := g \circ \dots \circ g$ (s times), we have

$$x^2 = \lim_{n \rightarrow \infty} x - \sum_{s=1}^n \frac{g_s(x)}{2^{2s}} \quad \text{for all } x \in [0, 1].$$

Key Idea of the Proof

Main Task: Approximate $\mathbf{V}(\mathbf{B}_y^{\text{rb}})^{-1}\mathbf{V}^T\mathbf{f}_y^{\text{h}}$ by a ReLU neural network and control its size!

Step 1 (Scalar Multiplication from Yarotsky; 2017):

For $g(x) := \min\{2x, 2 - 2x\}$ and $g_s := g \circ \dots \circ g$ (s times), we have

$$x^2 = \lim_{n \rightarrow \infty} x - \sum_{s=1}^n \frac{g_s(x)}{2^{2s}} \quad \text{for all } x \in [0, 1].$$

Also, g can be represented by a neural network due to

$$g(x) = 2\varrho(x) - 4\varrho(x - \frac{1}{2}) + 2\varrho(x - 2) \quad \text{for all } x \in [0, 1].$$

Key Idea of the Proof

Main Task: Approximate $\mathbf{V}(\mathbf{B}_y^{\text{rb}})^{-1}\mathbf{V}^T\mathbf{f}_y^{\text{h}}$ by a ReLU neural network and control its size!

Step 1 (Scalar Multiplication from Yarotsky; 2017):

For $g(x) := \min\{2x, 2 - 2x\}$ and $g_s := g \circ \dots \circ g$ (s times), we have

$$x^2 = \lim_{n \rightarrow \infty} x - \sum_{s=1}^n \frac{g_s(x)}{2^{2s}} \quad \text{for all } x \in [0, 1].$$

Also, g can be represented by a neural network due to

$$g(x) = 2\varrho(x) - 4\varrho(x - \tfrac{1}{2}) + 2\varrho(x - 2) \quad \text{for all } x \in [0, 1].$$

Moreover,

$$xz = 1/4((x + z)^2 - (x - z)^2) \quad \text{for all } x, z \in \mathbb{R}.$$

\implies *Scalar multiplication on $[-1, 1]^2$ can be ϵ -approximated by a neural network of size $\mathcal{O}(\log_2(1/\epsilon))$.*

Key Idea of the Proof

Step 2 (Multiplication):

A matrix multiplication of two matrices of size $d \times d$ can be performed by d^3 scalar multiplications.

\Rightarrow *Matrix multiplication can be ϵ -approximated by a neural network of size $\mathcal{O}(d(\epsilon)^3 \log_2(1/\epsilon))$.*

Key Idea of the Proof

Step 2 (Multiplication):

A matrix multiplication of two matrices of size $d \times d$ can be performed by d^3 scalar multiplications.

\implies *Matrix multiplication can be ϵ -approximated by a neural network of size $\mathcal{O}(d(\epsilon)^3 \log_2(1/\epsilon))$.*

Step 3 (Inversion):

- ▶ Neural networks can approximate matrix polynomials.
- ▶ Neural networks can the inversion operator $\mathbf{A} \mapsto \mathbf{A}^{-1}$ using

$$\sum_{s=0}^m \mathbf{A}^s \longrightarrow (\mathbf{Id}_{\mathbb{R}^d} - \mathbf{A})^{-1} \quad \text{as } m \rightarrow \infty.$$

\implies *Matrix inversion can be ϵ -approximated by a neural network of size $\mathcal{O}(d(\epsilon)^3 \log_2^q(1/\epsilon))$ for a constant $q > 0$.*

Key Idea of the Proof

Step 4 (Discrete Parametric Map w.r.t Reduced Basis):

► Now use the assumptions on \mathbf{B}_y^{rb} and \mathbf{f}_y^{rb} .

⇒ *The map $y \mapsto (\mathbf{B}_y^{\text{rb}})^{-1} \mathbf{f}_y^{\text{rb}}$ can be ϵ -approximated by a neural network Φ^{rb} of size $\mathcal{O}(d(\epsilon)^3 \log_2^q(1/\epsilon) + \text{poly}(p)d(\epsilon)^2 \log_2^q(1/\epsilon))$.*

Key Idea of the Proof

Step 4 (Discrete Parametric Map w.r.t Reduced Basis):

- ▶ Now use the assumptions on \mathbf{B}_y^{rb} and \mathbf{f}_y^{rb} .

\implies *The map $y \mapsto (\mathbf{B}_y^{\text{rb}})^{-1} \mathbf{f}_y^{\text{rb}}$ can be ϵ -approximated by a neural network Φ^{rb} of size $\mathcal{O}(d(\epsilon)^3 \log_2^q(1/\epsilon) + \text{poly}(p)d(\epsilon)^2 \log_2^q(1/\epsilon))$.*

For Theorem 1:

- ▶ Now use the assumption that every element from the reduced basis can be approximately represented in the high-fidelity basis.
- ▶ Consider then $\mathbf{V} \circ \Phi^{\text{rb}}$.

\implies *The discrete parametric map can be ϵ -approximated by a neural network of size $\mathcal{O}(d(\epsilon)^3 \log_2^q(1/\epsilon) + d(\epsilon)D + \text{poly}(p)d(\epsilon)^2 \log_2^q(1/\epsilon))$.*

Key Idea of the Proof

Step 4 (Discrete Parametric Map w.r.t Reduced Basis):

- ▶ Now use the assumptions on \mathbf{B}_y^{rb} and \mathbf{f}_y^{rb} .

⇒ *The map $y \mapsto (\mathbf{B}_y^{\text{rb}})^{-1} \mathbf{f}_y^{\text{rb}}$ can be ϵ -approximated by a neural network Φ^{rb} of size $\mathcal{O}(d(\epsilon)^3 \log_2^q(1/\epsilon) + \text{poly}(p)d(\epsilon)^2 \log_2^q(1/\epsilon))$.*

For Theorem 1:

- ▶ Now use the assumption that every element from the reduced basis can be approximately represented in the high-fidelity basis.
- ▶ Consider then $\mathbf{V} \circ \Phi^{\text{rb}}$.

⇒ *The discrete parametric map can be ϵ -approximated by a neural network of size $\mathcal{O}(d(\epsilon)^3 \log_2^q(1/\epsilon) + d(\epsilon)D + \text{poly}(p)d(\epsilon)^2 \log_2^q(1/\epsilon))$.*

For Theorem 2:

- ▶ Now use the assumption that neural networks can approximate each element of the reduced basis.

⇒ *The continuous parametric map can be ϵ -approximated by a neural network of size $\mathcal{O}(d(\epsilon)^3 \log_2^q(1/\epsilon) + \text{poly}(p)d(\epsilon)^2 \log_2^q(1/\epsilon))$.*

Numerical Experiments

Practical Learning Problems: A Different World?

Approximation-Theoretical Explanation for a Practical Learning Problem:

- ▶ Tightness of the upper bounds
- ▶ Optimization and sampling prevent approximation theoretical effect from materializing
- ▶ Asymptotic estimates

Goal:

We aim to analyze the approximation-theoretical effect of the architecture on the overall performance of the learning problem in practice!

Numerical Results

(Geist, Petersen, Raslan, Schneider, K; 2020)

Problem with Comparability of Tests:

- ▶ Effect of the sampling procedure
- ▶ Effect of the optimization procedure
- ▶ Quantification of the intrinsic complexity

Numerical Results

(Geist, Petersen, Raslan, Schneider, K; 2020)

Problem with Comparability of Tests:

- ▶ Effect of the sampling procedure
- ▶ Effect of the optimization procedure
- ▶ Quantification of the intrinsic complexity

Our Set-up:

- ▶ Keeping the architecture fixed
- ▶ Analyzing the convergence behavior a posteriori
- ▶ Establishing independence of sample generation
- ▶ Design of semi-ordered test-cases

Numerical Results

(Geist, Petersen, Raslan, Schneider, K; 2020)

Problem with Comparability of Tests:

- ▶ Effect of the sampling procedure
- ▶ Effect of the optimization procedure
- ▶ Quantification of the intrinsic complexity

Our Set-up:

- ▶ Keeping the architecture fixed
- ▶ Analyzing the convergence behavior a posteriori
- ▶ Establishing independence of sample generation
- ▶ Design of semi-ordered test-cases

Hypotheses:

- ▶ *The performance does not suffer from the curse of dimensionality.*
- ▶ *The performance is very sensitive to parametrization.*
- ▶ *Learning is efficient also for non-affinely parametrized problems.*

Numerical Study of Deep Learning for Parametric PDEs:

- ▶ Hesthaven and Ubbiali; 2018
Train neural networks to predict coefficients of solution in a precomputed reduced basis
- ▶ Tripathy and Bilonis; 2018
Solution of the parametric PDE is learned at point evaluations in fixed spatial coordinates
- ▶ Dal Santo, Deparis, and Pegolotti; 2019
Solution of the parametric PDE is learned in a precomputed reduced basis

Study of the Relation of Approximation-Theoretical Findings and Practise:

- ▶ Bölcskei, Grohs, K, and Petersen; 2019
Reproduction of approximation rates of optimally memory-efficient NNs
- ▶ Fokina and Oseledets; 2019
Reproduction of certain exponential convergence rates of NNs
- ▶ Adcock and Dexter; 2020
Numerical study of visibility of approximation theoretic results in practise

Parametric Diffusion Equation:

We will consider the following parametric diffusion equation:

$$-\nabla \cdot (a_y(\mathbf{x}) \cdot \nabla u_y(\mathbf{x})) = f(\mathbf{x}), \quad \text{on } \Omega = (0, 1)^2, \quad u_y|_{\partial\Omega} = 0,$$

where $f \in L^2(\Omega)$ and $a_y \in L^\infty(\Omega)$ is a diffusion coefficient depending on a parameter $y \in \mathcal{Y}$.

Parametric Map:

We learn a discretization of the map $\mathbb{R}^p \supset \mathcal{Y} \ni y \mapsto u_y$, where $p \in \mathbb{N}$, for various choices of parametrizations

$$\mathbb{R}^p \supset \mathcal{Y} \ni y \mapsto a_y.$$

What We Vary...

- ▶ Type of parametrization
- ▶ Dimension of parameter space
- ▶ Complexity of hyper-parameters

Parametric Diffusion Equation

Parametric Diffusion Equation:

$$-\nabla \cdot (a(\mathbf{x}) \cdot \nabla u_a(\mathbf{x})) = f(\mathbf{x}), \quad \text{on } \Omega = (0,1)^2, \quad u|_{\partial\Omega} = 0,$$

where

$$a \in \mathcal{A} = \{a_y : y \in \mathcal{Y}\} \subset L^\infty(\Omega) \quad \text{and} \quad f(x) = 20 + 10x_1 - 5x_2.$$

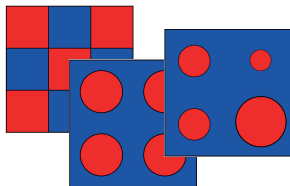
Affine Parametrization: For fixed functions $(a_i)_{i=0}^p \subset L^\infty(\Omega)$,

$$\mathcal{A} = \left\{ a_y = a_0 + \sum_{i=1}^p y_i a_i : y = (y_i)_{i=1}^p \in \mathcal{Y} \right\}.$$

- ▶ Trigonometric polynomials
- ▶ Chessboard partition
- ▶ Cookies with fixed radii

Non-Affine Parametrization:

- ▶ Cookies with variable radii
- ▶ Clipped polynomials



Further Set-Up

Finite Element Space:

- ▶ $\Omega = [0, 1]^2$ with 101×101 equidistant grid points

Fixed Neural Network:

- ▶ $(p, 300, \dots, 300, 10201)$ with $L = 11$ layers
- ▶ Activation function: 0.2-LReLU.

Fixed Training Procedure:

- ▶ Training set: 20000 i.i.d. parameter samples
- ▶ Neural network: Initialized according to a normal distribution with mean 0 and standard deviation 0.1
- ▶ Loss function: Relative error on the finite-element discretization of \mathcal{H}
- ▶ Optimization: Batch gradient descent

Dimension:

- ▶ Various dimensions of the parameter set up to 91.

Numerical Experiments, I

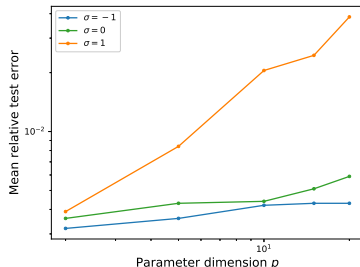
Trigonometric Polynomials:

$$\mathcal{A}^{\text{tp}}(p, \sigma) := \left\{ \mu + \sum_{i=1}^p y_i \cdot i^{\sigma} \cdot (1 + a_i) : y \in \mathcal{Y} = [0, 1]^p \right\},$$

for some fixed shift $\mu > 0$, scaling coefficient $\sigma \in \mathbb{R}$, and

$$a_i(\mathbf{x}) = \sin \left(\left\lfloor \frac{i+2}{2} \right\rfloor \pi x_1 \right) \sin \left(\left\lfloor \frac{i+2}{2} \right\rfloor \pi x_2 \right), \quad \text{for } i = 1, \dots, p.$$

Numerical Results:



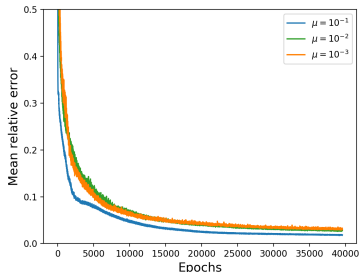
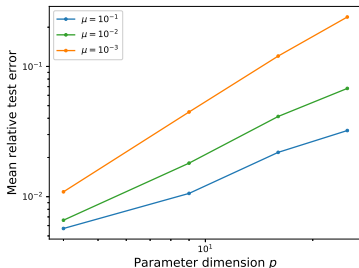
Numerical Experiments, II

Chessboard Partition: Let $p = s^2$ for some $s \in \mathbb{N}$. Then

$$\mathcal{A}^{\text{cb}}(p, \mu) := \left\{ \mu + \sum_{i=1}^p y_i \chi_{\Omega_i} : y \in \mathcal{Y} = [0, 1]^p \right\},$$

where $(\Omega_i)_{i=1}^p$ forms a $s \times s$ chessboard partition of $(0, 1)^2$ and $\mu > 0$ is a fixed shift.

Numerical Results:



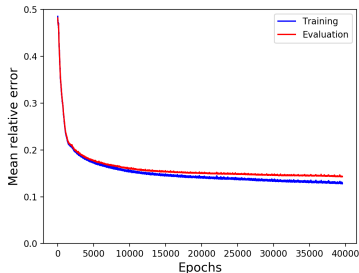
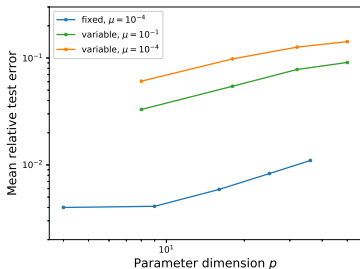
$p = 25$

Numerical Experiments, III

Cookies with Variable Radii: For $s \in \mathbb{N}$ and every $i = 1, \dots, s$, we are given disks $\Omega_{i, y_{i+s^2}}$ with centers $((2k+1)/(2s), (2\ell-1)/(2s))$, where $i = ks + \ell$ for uniquely determined $k \in \{0, \dots, s-1\}$ and $\ell \in \{1, \dots, s\}$ and radius $y_{i+s^2}/(2s)$:

$$\mathcal{A}^{\text{cvr}}(p, \mu) := \left\{ \mu + \sum_{i=1}^p y_i \chi_{\Omega_{i, y_{i+s^2}}} : y \in \mathcal{Y} = [0, 1]^p \times [0.5, 0.9]^p \right\}.$$

Numerical Results:



$p = 50$ and $\mu = 10^{-4}$

Hypotheses and Results:

- ▶ *The performance does not suffer from the curse of dimensionality.*
 - ▶ *True*, we never observed an exponential scaling.

Hypotheses and Results:

- ▶ *The performance does not suffer from the curse of dimensionality.*
 - ▶ *True*, we never observed an exponential scaling.
 - ▶ *The performance is very sensitive to parametrization.*
 - ▶ *True*, there are strong differences in the performance.
 - ▶ More complex parametrized sets yield higher errors, whereas simpler sets or spaces with intuitively lower intrinsic dimensionality yield smaller errors.
- ~> *The approximation theoretical intrinsic dimension of the parametric problem is a main factor in determining the hardness!*

Hypotheses and Results:

- ▶ *The performance does not suffer from the curse of dimensionality.*
 - ▶ *True*, we never observed an exponential scaling.
- ▶ *The performance is very sensitive to parametrization.*
 - ▶ *True*, there are strong differences in the performance.
 - ▶ More complex parametrized sets yield higher errors, whereas simpler sets or spaces with intuitively lower intrinsic dimensionality yield smaller errors.
- ~> *The approximation theoretical intrinsic dimension of the parametric problem is a main factor in determining the hardness!*
- ▶ *Learning is efficient also for non-affinely parametrized problems.*
 - ▶ *True*, there is no fundamental difference of the performance for non-affinely parametrized problems.

But...

Observation: Despite *enormous amounts of money* we still have *major robustness problems!*

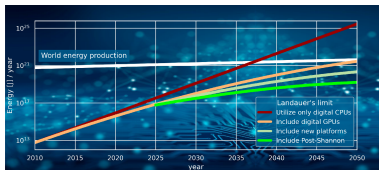


But...

Observation: Despite *enormous amounts of money* we still have *major robustness problems!*



Even More Serious Problem: Sustainability/Energy Efficiency



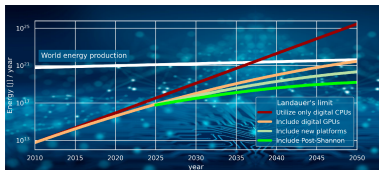
Source: Decadal Plan of the Semiconductor Research Corporation for the Biden (US) Administration, 2021

But...

Observation: Despite *enormous amounts of money* we still have *major robustness problems!*



Even More Serious Problem: Sustainability/Energy Efficiency



Source: Decadal Plan of the Semiconductor Research Corporation for the Biden (US) Administration, 2021

Something much deeper must be going on...

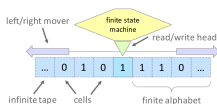
*Towards Reliable and Sustainable AI:
Next Generation AI Computing!*

A Serious Problem

A Serious Problem

Computability on Digital Machines/Turing Machines (informal):

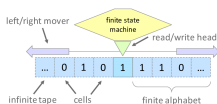
A *computable problem (function)* is one for which the input-output relation can be computed on a digital machine for any given accuracy.



A Serious Problem

Computability on Digital Machines/Turing Machines (informal):

A *computable problem (function)* is one for which the input-output relation can be computed on a digital machine for any given accuracy.



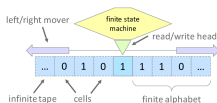
Solution Set of Inverse Problem: For $A \in \mathbb{C}^{m \times N}$ and $y \in \mathbb{C}^m$ let

$$\Psi(A, y) := \arg \min_{x \in \mathbb{C}^N} \|x\|_{\ell^1} \text{ such that } \|Ax - y\|_{\ell^2} \leq \varepsilon.$$

A Serious Problem

Computability on Digital Machines/Turing Machines (informal):

A *computable problem (function)* is one for which the input-output relation can be computed on a digital machine for any given accuracy.



Solution Set of Inverse Problem: For $A \in \mathbb{C}^{m \times N}$ and $y \in \mathbb{C}^m$ let

$$\Psi(A, y) := \arg \min_{x \in \mathbb{C}^N} \|x\|_{\ell^1} \text{ such that } \|Ax - y\|_{\ell^2} \leq \varepsilon.$$

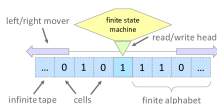
Theorem (Boche, Fono, K; 2023):

The problem described by the function $\Psi : \mathbb{C}^{m \times N} \times \mathbb{C}^m \rightarrow \mathbb{C}^N$ for fixed parameters $\varepsilon \in (0, 1)$, $N \geq 2$, and $m < N$, is *not computable on a Turing machine*.

A Serious Problem

Computability on Digital Machines/Turing Machines (informal):

A *computable problem (function)* is one for which the input-output relation can be computed on a digital machine for any given accuracy.



Solution Set of Inverse Problem: For $A \in \mathbb{C}^{m \times N}$ and $y \in \mathbb{C}^m$ let

$$\Psi(A, y) := \arg \min_{x \in \mathbb{C}^N} \|x\|_{\ell^1} \text{ such that } \|Ax - y\|_{\ell^2} \leq \varepsilon.$$

Theorem (Boche, Fono, K; 2023):

The problem described by the function $\Psi : \mathbb{C}^{m \times N} \times \mathbb{C}^m \rightarrow \mathbb{C}^N$ for fixed parameters $\varepsilon \in (0, 1)$, $N \geq 2$, and $m < N$, is *not computable on a Turing machine*.

~> Interesting work by Hansen et al. on oracle Turing machines!

The Hardware is the Problem!

Remark: There does not exist an algorithm on digital hardware which yields neural networks Φ_A approximating $\Psi(A, \cdot)$ for any given accuracy and all $A \in \mathbb{C}^{m \times N}$.

General Barrier:

- ▶ *Limits of computability* on today's hardware
- ▶ Reason for *problems with non-robustness*?



The Hardware is the Problem!

Remark: There does not exist an algorithm on digital hardware which yields neural networks Φ_A approximating $\Psi(A, \cdot)$ for any given accuracy and all $A \in \mathbb{C}^{m \times N}$.

General Barrier:

- ▶ *Limits of computability* on today's hardware
- ▶ Reason for *problems with non-robustness*?



Theorem (Boche, Fono, K; 2023):

Fix parameters $\varepsilon \in (0, \frac{1}{4})$, $N \geq 2$, and $m < N$. There *does not exist* a (Banach–Mazur-)computable function $\hat{\Psi} : \mathbb{C}^{m \times N} \times \mathbb{C}^m \rightarrow \mathbb{C}^N$ such that

$$\sup_{(A,y) \in \mathbb{C}^{m \times N} \times \mathbb{C}^m} \|\Psi(A,y) - \hat{\Psi}(A,y)\|_{\ell^2} < \frac{1}{4}.$$

Remark: The statement does not depend on the unboundedness of the input domain, but holds true on a compact input set.

What now? ... Mathematics Tells Us the Answer!

What now? ... Mathematics Tells Us the Answer!

Theorem (Boche, Fono, K; 2023):

The solution of a finite-dimensional inverse problem is *computable* (by a deep neural network) *on an analog (Blum-Shub-Smale) machine!*

What now? ... Mathematics Tells Us the Answer!

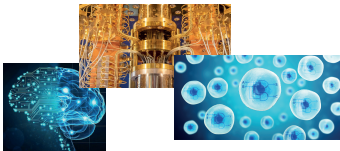
Theorem (Boche, Fono, K; 2023):

The solution of a finite-dimensional inverse problem is *computable* (by a deep neural network) *on an analog (Blum-Shub-Smale) machine!*

Reliability for certain problem settings requires novel hardware!

Possible Future Developments:

- ▶ *Neuromorphic computing*
- ▶ Biocomputing
- ▶ *Quantum computing*



...more Computability Problems!

...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable!*

...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable!*

Theorem (Boche, Fono, K; 2024): The Pseudo Inverse is *not (Turing) computable!*

...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable!*

Theorem (Boche, Fono, K; 2024): The Pseudo Inverse is *not (Turing) computable!*

Theorem (Bacho, Boche, K; 2024): Computing the solutions to the Laplace and the diffusion equation on digital hardware causes a *complexity blowup*.

...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable*!

Theorem (Boche, Fono, K; 2024): The Pseudo Inverse is *not (Turing) computable*!

Theorem (Bacho, Boche, K; 2024): Computing the solutions to the Laplace and the diffusion equation on digital hardware causes a *complexity blowup*.

Theorem (Lee, Boche, K; 2024): Finding the solution of most optimization problems is *not (Turing) computable*!

...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable!*

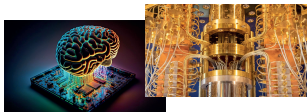
Theorem (Boche, Fono, K; 2024): The Pseudo Inverse is *not (Turing) computable!*

Theorem (Bacho, Boche, K; 2024): Computing the solutions to the Laplace and the diffusion equation on digital hardware causes a *complexity blowup*.

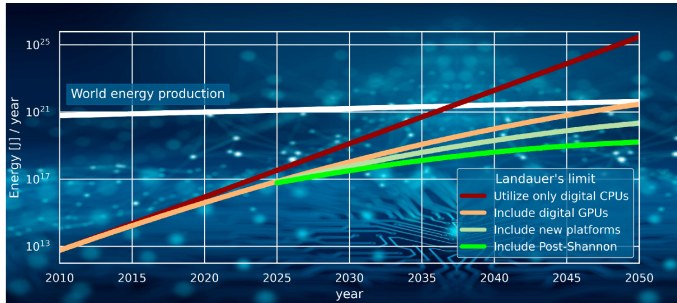
Theorem (Lee, Boche, K; 2024): Finding the solution of most optimization problems is *not (Turing) computable!*

Vision for the Future:

Mathematically Reliable...by Analog Computing!



Problem with Enormous Energy Consumption



Source: Decadal Plan of the Semiconductor Research Corporation for the Biden (US) Administration, 2021

...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable!*

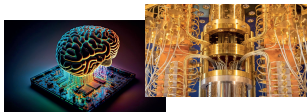
Theorem (Boche, Fono, K; 2024): The Pseudo Inverse is *not (Turing) computable!*

Theorem (Bacho, Boche, K; 2024): Computing the solutions to the Laplace and the diffusion equation on digital hardware causes a *complexity blowup*.

Theorem (Lee, Boche, K; 2024): Finding the solution of most optimization problems is *not (Turing) computable!*

Vision for the Future:

Mathematically Reliable...by Analog Computing!



...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable!*

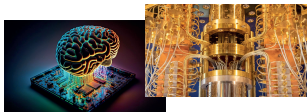
Theorem (Boche, Fono, K; 2024): The Pseudo Inverse is *not (Turing) computable!*

Theorem (Bacho, Boche, K; 2024): Computing the solutions to the Laplace and the diffusion equation on digital hardware causes a *complexity blowup*.

Theorem (Lee, Boche, K; 2024): Finding the solution of most optimization problems is *not (Turing) computable!*

Vision for the Future:

Mathematically Reliable & Energy Efficient AI...by Analog Computing!



...more Computability Problems!

Theorem (Boche, Fono, K; 2024): Many classification problems are also *not (Turing) computable!*

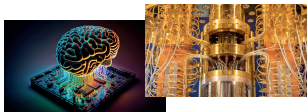
Theorem (Boche, Fono, K; 2024): The Pseudo Inverse is *not (Turing) computable!*

Theorem (Bacho, Boche, K; 2024): Computing the solutions to the Laplace and the diffusion equation on digital hardware causes a *complexity blowup*.

Theorem (Lee, Boche, K; 2024): Finding the solution of most optimization problems is *not (Turing) computable!*

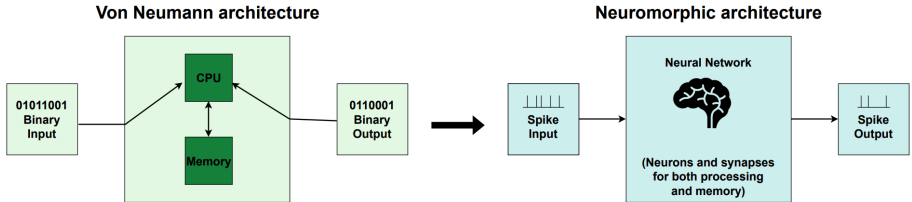
Vision for the Future:

Mathematically Reliable & Energy Efficient AI...by Analog Computing!



Neural Networks for Next Generation Computing

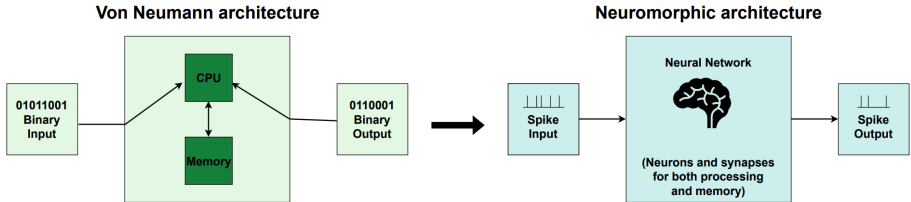
Neuromorphic Computing



Features of neuromorphic hardware:

- ▶ Closer to the human brain.
- ▶ Energy efficiency.
- ▶ Execution speed.
- ▶ Robustness.
- ▶ ...

Neuromorphic Computing

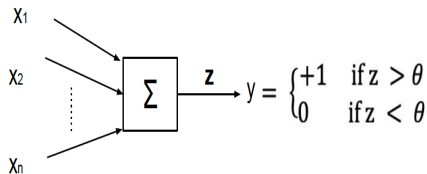


Features of neuromorphic hardware:

- ▶ Closer to the human brain.
- ▶ Energy efficiency.
- ▶ Execution speed.
- ▶ Robustness.
- ▶ ...

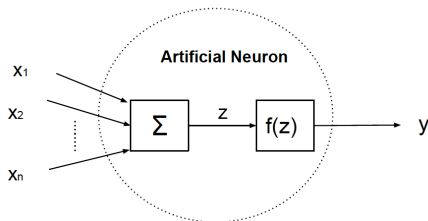
↪ *Suitable neural networks for neuromorphic computing?*

First and Second Generation of Artificial Neurons



First Generation

Perceptron model or Threshold gate



Second Generation

Sigmoid neuron

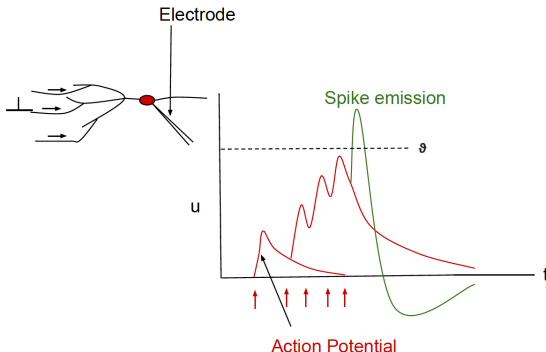
Some General Remarks:

- ▶ Both computational models of a neuron are *biologically inspired*, but rather *crude models*.
- ▶ Both lead to networks, which are *Universal Approximators*.

Third Generation

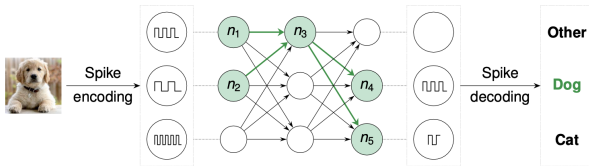
Spiking Neuron:

- ▶ More *biologically realistic* than the first and second generation.
- ▶ *Fires/generates* short electrical pulse known as *action-potential* or *spike*.
- ▶ *Spike* or *action-potential* is the elementary unit of signal transmission.

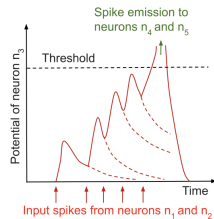


Spiking Neural Network

Computation graph associated with a spiking neural network

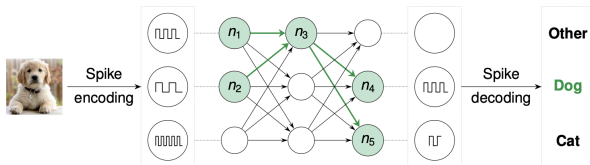


Spike dynamics of neuron n_3

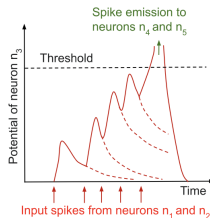


Spiking Neural Network

Computation graph associated with a spiking neural network



Spike dynamics of neuron n_3



Some Remarks:

- ▶ *Asynchronous* transmission of information via *spikes*.
- ▶ Information is encoded in the *timing* of *individual spikes*.
- ▶ Numerous models for spiking neurons exist; one of those is the *Spike Response Model*.

Time is one crucial factor in this model!

Spiking Neurons under Spike Response Model, I

Definition: A *spiking neural network* Φ is a directed graph (V, E) and consists of a finite set V of spiking neurons, a subset $V_{\text{in}} \subset V$ of input neurons, and a set $E \subset V \times V$ of synapses. Each *synapse* $(u, v) \in E$ is associated with

- ▶ a *synaptic weight* $w_{uv} \geq 0$,
- ▶ a *synaptic delay* $d_{uv} \geq 0$,
- ▶ and a *response function* $\varepsilon_{uv} : \mathbb{R} \rightarrow \mathbb{R}$.

Spiking Neurons under Spike Response Model, I

Definition: A *spiking neural network* Φ is a directed graph (V, E) and consists of a finite set V of spiking neurons, a subset $V_{\text{in}} \subset V$ of input neurons, and a set $E \subset V \times V$ of synapses. Each *synapse* $(u, v) \in E$ is associated with

- ▶ a *synaptic weight* $w_{uv} \geq 0$,
- ▶ a *synaptic delay* $d_{uv} \geq 0$,
- ▶ and a *response function* $\varepsilon_{uv} : \mathbb{R} \rightarrow \mathbb{R}$.

Each *neuron* $v \in V \setminus V_{\text{in}}$ is associated with

- ▶ a *firing threshold* $\theta_v > 0$,
- ▶ and a *membrane potential* $P_v : \mathbb{R} \rightarrow \mathbb{R}$,

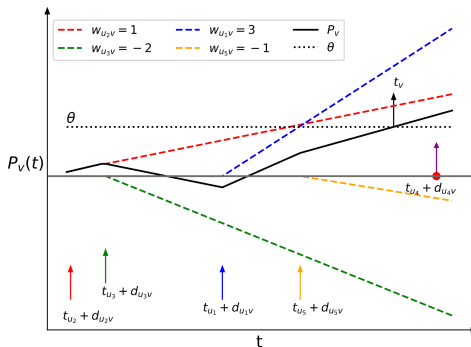
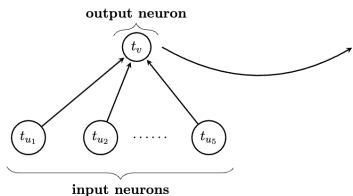
which is given by

$$P_v(t) = \sum_{(u,v) \in E} \sum_{t_u^f \in F_u} w_{uv} \varepsilon_{uv}(t - t_u^f),$$

with $F_u = \{t_u^f : 1 \leq f \leq n \text{ for some } n \in \mathbb{N}\}$ being the set of *firing times* of neuron u , i.e., times t whenever $P_u(t)$ reaches θ_u .

Neuronal Response to Input Spikes

Evolution of the membrane potential as a result of incoming spikes:



Spiking Neurons under Spike Response Model, II

Remark: Given that a neuron *spikes only once*, the state of a neuron v in a spiking neural network is given as

$$P_v(t) = \sum_{(u,v) \in E} w_{uv} \varepsilon_{uv}(t - t_u), \quad (1)$$

where t_u = firing time of presynaptic neuron u .

Spiking Neurons under Spike Response Model, II

Remark: Given that a neuron *spikes only once*, the state of a neuron v in a spiking neural network is given as

$$P_v(t) = \sum_{(u,v) \in E} w_{uv} \varepsilon_{uv}(t - t_u), \quad (1)$$

where t_u = firing time of presynaptic neuron u .

Additional Assumption: Assume that the response function ε is linear and satisfies the following condition

$$\varepsilon_{uv}(t) = \begin{cases} 0, & \text{if } t \notin [d_{uv}, d_{uv} + \delta], \\ (t - d_{uv}), & \text{if } t \in [d_{uv}, d_{uv} + \delta], \end{cases}$$

where $\delta > 0$ is the length of the linear segment of the response function ε . Then (1) simplifies to

$$P_v(t) = \sum_{(u,v) \in E} \mathbf{1}_{\{0 < t - t_u - d_{uv} \leq \delta\}} w_{uv} (t - t_u - d_{uv}).$$

Spiking Neurons under Spike Response Model, III

Assuming δ is large or even infinite in some cases, then

$$\begin{aligned}\inf_{t \in \mathbb{R}} P_v(t) &= \inf_{t \in \mathbb{R}} \sum_{(u,v) \in E} \mathbf{1}_{\{t > t_u + d_{uv}\}} w_{uv} (t - t_u - d_{uv}) \\ &= \inf_{t \in \mathbb{R}} \sum_{(u,v) \in E} w_{uv} \sigma(t - t_u - d_{uv}) = \theta_v,\end{aligned}$$

where $\sigma(x) = \max\{0, x\}$. Then,

$$t_v = \frac{\theta_v + \sum_{(u,v) \in E: t_v > t_u + d_{uv}} w_{uv} (t_u + d_{uv})}{\sum_{(u,v) \in E: t_v > t_u + d_{uv}} w_{uv}}.$$

Spiking Neurons under Spike Response Model, III

Assuming δ is large or even infinite in some cases, then

$$\begin{aligned}\inf_{t \in \mathbb{R}} P_v(t) &= \inf_{t \in \mathbb{R}} \sum_{(u,v) \in E} \mathbf{1}_{\{t > t_u + d_{uv}\}} w_{uv} (t - t_u - d_{uv}) \\ &= \inf_{t \in \mathbb{R}} \sum_{(u,v) \in E} w_{uv} \sigma(t - t_u - d_{uv}) = \theta_v,\end{aligned}$$

where $\sigma(x) = \max\{0, x\}$. Then,

$$t_v = \frac{\theta_v + \sum_{(u,v) \in E: t_v > t_u + d_{uv}} w_{uv} (t_u + d_{uv})}{\sum_{(u,v) \in E: t_v > t_u + d_{uv}} w_{uv}}.$$

Proposition (Singh, Fono, K; 2024):

The output firing time is a *piecewise linear function* of the input firing time. Under the additional mild assumption it is even *continuous*.

Classical Neural Networks as Spiking Neural Networks

Main Question:

*Can we quantify the advantage of spiking neural networks
over classical neural networks?*

Classical Neural Networks as Spiking Neural Networks

Main Question:

Can we quantify the advantage of spiking neural networks over classical neural networks?

Theorem (Singh, Fono, K; 2024):

Let $L, d \in \mathbb{N}$, $[a, b]^d \subset \mathbb{R}^d$ and let Ψ be a *classical ReLU-neural network* of depth L and width d . Then there exists a *spiking neural network* Φ with $N(\Phi) = N(\Psi) + L(2d + 3) - (2d + 2)$ and $L(\Phi) = 3L - 2$ that realizes Ψ on $[a, b]^d$.

Classical Neural Networks as Spiking Neural Networks

Main Question:

Can we quantify the advantage of spiking neural networks over classical neural networks?

Theorem (Singh, Fono, K; 2024):

Let $L, d \in \mathbb{N}$, $[a, b]^d \subset \mathbb{R}^d$ and let Ψ be a *classical ReLU-neural network* of depth L and width d . Then there exists a *spiking neural network* Φ with $N(\Phi) = N(\Psi) + L(2d + 3) - (2d + 2)$ and $L(\Phi) = 3L - 2$ that realizes Ψ on $[a, b]^d$.

Main Ingredient of Proof:

The ReLU-activation function can be realized by a 2-layer spiking neural network on $[a, b]$.

Classical Neural Networks as Spiking Neural Networks

Main Question:

Can we quantify the advantage of spiking neural networks over classical neural networks?

Theorem (Singh, Fono, K; 2024):

Let $L, d \in \mathbb{N}$, $[a, b]^d \subset \mathbb{R}^d$ and let Ψ be a *classical ReLU-neural network* of depth L and width d . Then there exists a *spiking neural network* Φ with $N(\Phi) = N(\Psi) + L(2d + 3) - (2d + 2)$ and $L(\Phi) = 3L - 2$ that realizes Ψ on $[a, b]^d$.

Main Ingredient of Proof:

The ReLU-activation function can be realized by a 2-layer spiking neural network on $[a, b]$.

Lemma:

There does not exist a 1-layer spiking neural network that realizes ReLU.

Spiking Neural Networks as Classical Neural Networks

Theorem (Singh, Fono, K; 2024): For $d \geq 2$, $\ell := \lceil \log_2(d+1) \rceil + 1$.

Let Φ be a 1-layer spiking neural network with one output neuron v and d input neurons u_1, \dots, u_d with $w_{u_i v} \in \mathbb{R}_{>0}$ for $i \in \{1, \dots, d\}$. Then:

- (a) t_Φ can be realized by a classical ReLU-neural network Ψ with $L(\Psi) = \ell$ and $N(\Psi) \in \mathcal{O}(\ell \cdot 2^{2d^3+3d^2+d})$.
- (b) t_Φ can be realized by a classical ReLU-neural network Ψ with $L(\Psi) \in \mathcal{O}(d)$ and $N(\Psi) \in \mathcal{O}(8^d)$.

Spiking Neural Networks as Classical Neural Networks

Theorem (Singh, Fono, K; 2024): For $d \geq 2$, $\ell := \lceil \log_2(d+1) \rceil + 1$. Let Φ be a 1-layer spiking neural network with one output neuron v and d input neurons u_1, \dots, u_d with $w_{u_i v} \in \mathbb{R}_{>0}$ for $i \in \{1, \dots, d\}$. Then:

- (a) t_Φ can be realized by a classical ReLU-neural network Ψ with $L(\Psi) = \ell$ and $N(\Psi) \in \mathcal{O}(\ell \cdot 2^{2d^3+3d^2+d})$.
- (b) t_Φ can be realized by a classical ReLU-neural network Ψ with $L(\Psi) \in \mathcal{O}(d)$ and $N(\Psi) \in \mathcal{O}(8^d)$.

Main Ingredient of Proof:

Proposition: Let Φ be a spiking neural network as above with arbitrary weights. Then t_Φ partitions the input domain into at most $2^d - 1$ linear regions. The following are equivalent:

- (1) The maximal number of linear regions is attained.
- (2) All synaptic weights are positive.

Spiking Neural Networks as Classical Neural Networks

Theorem (Singh, Fono, K; 2024): For $d \geq 2$, $\ell := \lceil \log_2(d+1) \rceil + 1$. Let Φ be a 1-layer spiking neural network with one output neuron v and d input neurons u_1, \dots, u_d with $w_{u_i v} \in \mathbb{R}_{>0}$ for $i \in \{1, \dots, d\}$. Then:

- (a) t_Φ can be realized by a classical ReLU-neural network Ψ with $L(\Psi) = \ell$ and $N(\Psi) \in \mathcal{O}(\ell \cdot 2^{2d^3+3d^2+d})$.
- (b) t_Φ can be realized by a classical ReLU-neural network Ψ with $L(\Psi) \in \mathcal{O}(d)$ and $N(\Psi) \in \mathcal{O}(8^d)$.

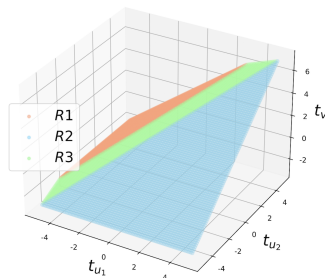
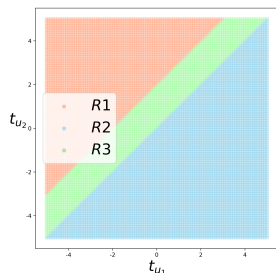
Architecture of Proof:

- ▶ Thus, t_Φ is continuous piecewise linear with $2^d - 1$ linear regions.
- ▶ Then apply upper bounds on the size of classical ReLU-neural networks to realize continuous piecewise linear mappings with a fixed number of linear regions.

Toy Example

Consider a spiking neuron v with two input units u_1 and u_2 . Set $\theta_v = w_{u_1 v} = d_{u_2 v} = 1$ and $d_{u_1 v} = 2$. Then, the firing time of v on the corresponding linear regions equals

$$t_v = \begin{cases} t_{u_1} + 3, & \text{if } t_{u_2} \geq t_{u_1} + 2, \\ t_{u_2} + 2, & \text{if } t_{u_2} \leq t_{u_1}, \\ \frac{1}{2}(t_{u_1} + t_{u_2}) + 2, & \text{if } t_{u_1} < t_{u_2} < t_{u_1} + 2. \end{cases}.$$



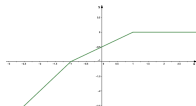
- ▶ A spiking neuron partitions the input space into 3 different regions.
- ▶ The output firing time t_v is continuous piecewise linear.

Example for Superiority of Spiking Neural Networks

A Special Continuous Piecewise Linear Function:

For $a < 0 < \theta < b$, consider the continuous piecewise linear function $f : [a, b] \rightarrow \mathbb{R}$ given by

$$f(x) = -\frac{1}{2}\sigma(-x-\theta) - \frac{1}{2}\sigma(-x+\theta) = -\frac{1}{2}\max(-x-\theta, 0) - \frac{1}{2}\max(-x+\theta, 0).$$



Lemma:

- ▶ A *1-layer spiking neural network* with *1 output neuron* and *2 input neurons* can realize f .
- ▶ Any *classical ReLU-neural network* requires at least *2 layers* and a total number of *4 neurons* to realize f .

Approximating the Minimum Function

Proposition: Let $d \in \mathbb{N}$ such that $d \geq 2$. Then, there exists a spiking neural network Φ with 1 output neuron v and d input neurons such that

$$|\Phi(x_1, \dots, x_d) - \min\{x_1, \dots, x_d\}| \leq \frac{(d-1)\theta}{2dw} \quad \text{for all } x_1, \dots, x_d \in \mathbb{R},$$

with $\theta > 0$ the threshold of v and $w > 0$ the weight of each connection.

Approximating the Minimum Function

Proposition: Let $d \in \mathbb{N}$ such that $d \geq 2$. Then, there exists a spiking neural network Φ with 1 output neuron v and d input neurons such that

$$|\Phi(x_1, \dots, x_d) - \min\{x_1, \dots, x_d\}| \leq \frac{(d-1)\theta}{2dw} \quad \text{for all } x_1, \dots, x_d \in \mathbb{R},$$

with $\theta > 0$ the threshold of v and $w > 0$ the weight of each connection.

Idea for Proof: Minimize the time gap between the earliest subset of neurons that can cause firing and the subset of neurons that actually induce firing by adjusting the ratio of threshold and weight appropriately.

Approximating the Minimum Function

Proposition: Let $d \in \mathbb{N}$ such that $d \geq 2$. Then, there exists a spiking neural network Φ with 1 output neuron v and d input neurons such that

$$|\Phi(x_1, \dots, x_d) - \min\{x_1, \dots, x_d\}| \leq \frac{(d-1)\theta}{2dw} \quad \text{for all } x_1, \dots, x_d \in \mathbb{R},$$

with $\theta > 0$ the threshold of v and $w > 0$ the weight of each connection.

Idea for Proof: Minimize the time gap between the earliest subset of neurons that can cause firing and the subset of neurons that actually induce firing by adjusting the ratio of threshold and weight appropriately.

Comparison with Classical ReLU-Neural Network:

- ▶ For any classical ReLU-neural network, irrespective of depth, to approximate *min*, each hidden layer must have *at least d neurons*.
- ▶ Under certain assumptions on the weights and data distribution, a classical ReLU-neural network of *depth 3 is necessary* to efficiently approximate *min*.

Spiking Neural Networks: Next Generation AI Computing



Open Questions:

- ▶ When and to which extent are spiking neural networks *more expressive* than classical neural networks?
- ▶ What is an optimal *training procedure*?
- ▶ Can we quantify *generalization abilities* (Neuman, Petersen; 2024)?
- ▶ What is a suitable measure for their *energy consumption*?
- ▶ ...

Mathematics of spiking neural networks is a wide open field!

Some Final Thoughts...

Conclusions

AI and High-Dimensional PDEs:

- ▶ We derive *upper bounds on the complexity* of ReLU neural networks to approximate parametric maps.
- ▶ Those neural networks *do not suffer from the curse of dimensionality*.
- ▶ The error on the training set *converges always smoothly*.
- ▶ The performance *does not suffer from the curse of dimensionality*.

Next Generation AI Computing:

- ▶ *Computability:*
 - ▶ Problems of continuum nature are computed on digital hardware.
 - ▶ Several key AI problems are not (Turing) computable.
 - ▶ Analog hardware can solve the computability problem.
- ▶ *Spiking Neural Networks:*
 - ▶ Spiking neural networks are suitable for neuromorphic chips.
 - ▶ Very few mathematical results exist at present.
 - ▶ *Vision:* Next generation AI computing!

THANK YOU!

References available at:

www.ai.math.lmu.de/kutyniok

Survey Paper (arXiv:2105.04026):

Berner, Grohs, K, Petersen, *The Modern Mathematics of Deep Learning*.

Related Book:

- Grohs and K, eds.,
Mathematical Aspects of Deep Learning
Cambridge University Press, 2022.

