Random features: methods & applications

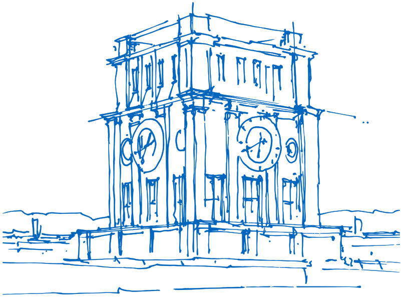
Woudschoten conference, talk #2

Felix Dietrich

Physics-Enhanced Machine Learning
School of Computation, Information and Technology
Technical University of Munich

2025-09-26

Collaboration with
Erik Bolager, Iryna Burak,
Chinmay Datar, Ana Cukarska,
Qing Sun, Anna Veselovska,
and **Massimo Fornasier**



Reminder: Connection to previous talk

Linear operator decomposition

Given $\mathcal{L} : A \to B$, assume we can perform a decomposition

$$\mathcal{L} = \sum_{k=1}^{\infty} \psi_k \lambda_k \phi_k =: \Psi \Lambda \Phi,$$

where $\psi_k \in B$, $\lambda_k \in \mathbb{C}$, $\phi_k \in A^*$.

Connection to this talk

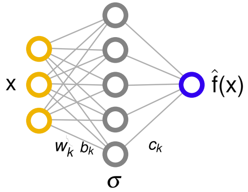
Given a sequence of neurons $\{g_k = \sigma(w_k \cdot + b_k)\}_k$, can we find a unitary transformation Q with

$$\mathcal{L} = \sum_{k=1}^{\infty} \psi_k \lambda_k \phi_k =: \Psi Q \Lambda^{1/2} \Lambda^{1/2} Q^* \Phi,$$

where $[\Psi Q \Lambda^{1/2}]_k = g_k$? Maybe we can even define Q as a random basis?

Reminder: Feed-forward neural networks

Picture



Compact

$$\hat{f}(x) = \sum_{k=1}^{m} c_k \sigma(w_k^T x + b_k)$$

Weights w_k, biases b_k, coefficients c_k.

Can we find $\sigma(w_k^T x + b_k) = g_k$? (Let's leave the previous talk behind now...)

Reminder: Stochastic gradient descent

ML example: Stochastic gradient descent [1]

Given a data set $X = \{x_i \in \mathbb{R}^d\}_{i=1}^N$ and a learning rate $\eta > 0$, optimize parameters θ of a model $\hat{f}$ to minimize the loss function $\mathcal{L}(X,\theta) = 1/|X| \sum_{x_i \in X} \|\hat{f}(\theta)(x_i) - f_{\text{true}}(x_i)\|^2$.

1. Randomly initialize $\theta_0 \sim N(0,1)$.
2. Update $\theta_{n+1} \leftarrow \theta_n - \eta \nabla \mathcal{L}(B_n, \theta_n)$, with $B_n \subset X$ chosen randomly.
3. Stop if $n >$ #iterations or $\mathcal{L}(B_n, \theta_n) <$ bound.

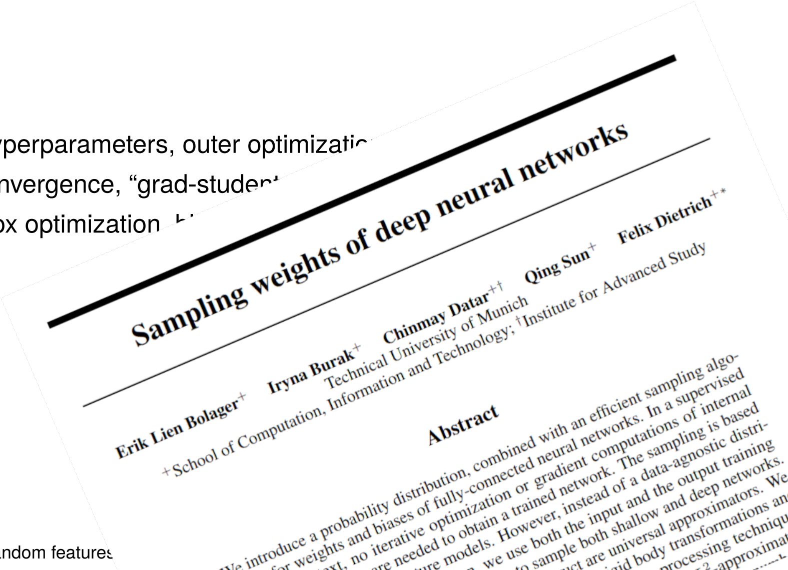
[1] Kingma and Ba, “Adam: A Method for Stochastic Optimization,” in ICLR 2015.

Challenges with backpropagation & SGD

- Many hyperparameters, outer optimization loop required.
- Slow convergence, “grad-student descent”.
- Black-box optimization, black-box models.

Challenges with backpropagation & SGD

- Many hyperparameters, outer optimizatio
- Slow convergence, “grad-studen
- Black-box optimization, bi



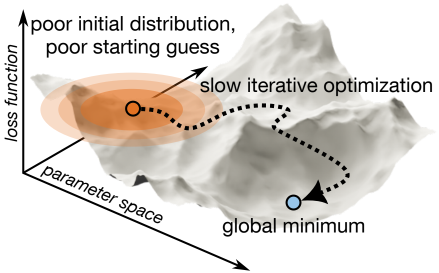
Sampling weights of deep neural networks

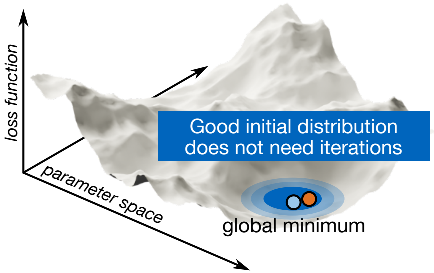
Erik Lien Bolager[+] Iryna Burak[+] Chinmay Datar[+†] Qing Sun[+] Felix Dietrich[+*]
Technical University of Munich
[+]School of Computation, Information and Technology; [†]Institute for Advanced Study

Abstract

We introduce a probability distribution, combined with an efficient sampling algo-
for weights and biases of fully-connected neural networks. In a supervised
ext, no iterative optimization or gradient computations of internal
are needed to obtain a trained network. The sampling is based
ture models. However, instead of a data-agnostic distri-
n, we use both the input and the output training
to sample both shallow and deep networks.
uct are universal approximators. We
igid body transformations an
processing techniqu
L^2-approximat

Random feature methods (data-driven)





Left: stochastic gradient descent slowly and iteratively refines a poor initial guess.
Right: good initial distributions of weights do not need iterative training!

Main question: How do we construct good initial distributions of parameters?

Feed-forward neural networks

Our setting

Feed-forward networks $\hat{u}$, (for now) with one hidden layer,

$$\hat{u}(x) = \sum_{k=1}^{m} c_k \sigma(w_k^T x + b_k),\ x \in \Omega,$$

with m hidden neurons and activation function $\sigma = \tanh$.

Main argument

If weights and biases (w_k, b_k) were given, we would only need to solve a (regularized) linear problem:

$$\min_{c \in \mathbb{R}^m} \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{k=1}^{m} c_k \sigma(w_k^T x_i + b_k) - f(x_i) \right)^2 + \lambda \|c\|^2 .$$

Parameters of feed-forward neural networks

Approaches to construct internal weights and biases

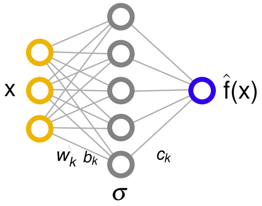
[1] "Classic": SGD, Adam	$\theta_{n+1} = \theta_n - \eta \nabla \text{Loss}(\theta_n)$
[2] Weight reconstruction	Approximate Hessian of f
[3] Random features, reservoirs	$w \sim N(0,1)$, $b \sim U(-1,1)$
[4] Using the data	$w = x^{(2)} - x^{(1)}$
[5] Sample weights "where it matters"	$w = (x^{(2)} - x^{(1)})/\|x^{(2)} - x^{(1)}\|^2$

[1] Kingma and Ba, "Adam: A Method for Stochastic Optimization," in ICLR 2015.
[2] Fornasier, Klock, Mondelli, and Rauchensteiner. "Finite Sample Identification of Wide Shallow Neural Networks with Biases." pre-print, 2022. arxiv.org/abs/2211.04589.
[3] Rahimi and Recht, "Uniform approximation of functions with random bases," IEEE, 2008.
[4] Galaris, Fabiani, Gallos, Kevrekidis, and Siettos, "Numerical Bifurcation Analysis of PDEs From Lattice Boltzmann Model Simulations" J. Sci. Comput., 2022.
[5] **Bolager, Burak, Datar, Sun, and D., "Sampling weights of deep neural networks," NeurIPS 2023.**

Feed-forward neural networks, revisited

Illustrations and notation

Picture



Compact

$\sum_{k=1}^{m} c_k \sigma(w_k^T x + b_k)$

Basis function view

$$\begin{aligned} &c_1\sigma(\langle w_1, x\rangle + b_1) &+\\ &c_2\sigma(\langle w_2, x\rangle + b_2) &+\\ &\dots\\ &c_m\sigma(\langle w_m, x\rangle + b_m) \end{aligned}$$

Main insights

1. Weights w_k are in the dual space of the data*, i.e., they are vectors.
2. Biases b_k shift the input of the activation function.
3. Coefficients c_k combine the activation functions linearly.

* Related work: Spek, Heeringa, Schwenninger, and Brune. 2025. "Duality for Neural Networks through Reproducing Kernel Banach Spaces." ACHA, 2025.

Feed-forward neural networks, revisited

Given the ReLU function ($\sigma(x) = \max(0, x)$), $\Omega = \mathbb{R} \times \mathbb{R}^D \times \mathbb{R}$ and a probability measure μ over Ω.

Mathematical framework: Barron spaces

The Barron space $\mathcal{B}$ is defined as

$$\mathcal{B} = \left\{ f : f(x) = \int_\Omega c\sigma(\langle w, x\rangle - b) d\mu(b, w, c) \text{ and } \|f\|_\mathcal{B} < \infty \right\},$$

with the Barron norm defined as

$$\|f\|_\mathcal{B} = \inf_\nu \max_{b,w,c \in \text{supp}(\nu)} \left\{ |c|(\|w\|_1 + |b|) \right\}.$$

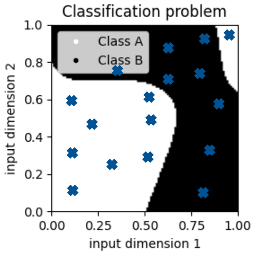
Main question in machine learning: Given f, how do we approximate the integral with a finite number of m parameters $(b_k, w_k, c_k)_{k=1}^m$?

“Sample where it matters”

Algorithm (Bolager et al., 2023)

1. Sample many random data pairs $(x^{(i)}, x^{(k)})$ in the domain $\mathcal{X}$.
2. Given data from target f, evaluate finite difference $d_{i,k} = \frac{\|f(x^{(i)}) - f(x^{(k)})\|}{\|x^{(i)} - x^{(k)}\|}$.
3. Draw m pairs with probability proportional to $d_{i,k}$.
4. Construct weights and biases: $w = s_1 \frac{x^{(i)} - x^{(k)}}{\|(x^{(i)} - x^{(k)})\|^2}$, $b = w^T x^{(i)} + s_2$.

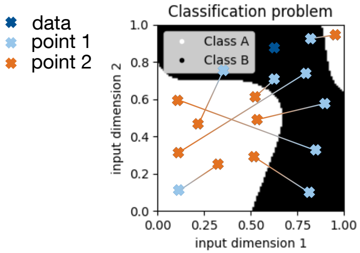
✖ data



“Sample where it matters”

Algorithm (Bolager et al., 2023)

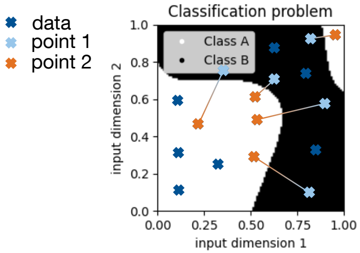
1. Sample many random data pairs $(x^{(i)}, x^{(k)})$ in the domain $\mathcal{X}$.
2. Given data from target f, evaluate finite difference $d_{i,k} = \frac{\|f(x^{(i)})-f(x^{(k)})\|}{\|x^{(i)}-x^{(k)}\|}$.
3. Draw m pairs with probability proportional to $d_{i,k}$.
4. Construct weights and biases: $w = s_1 \frac{x^{(i)}-x^{(k)}}{\|(x^{(i)}-x^{(k)})\|^2}$, $b = w^T x^{(i)} + s_2$.



“Sample where it matters”

Algorithm (Bolager et al., 2023)

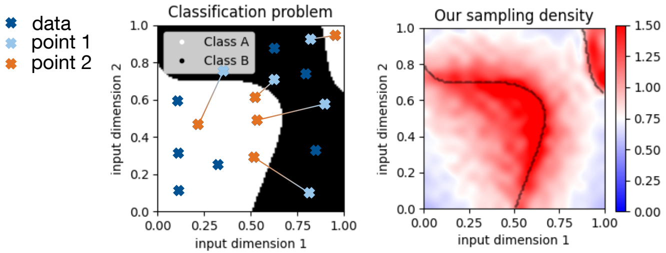
1. Sample many random data pairs $(x^{(i)}, x^{(k)})$ in the domain $\mathcal{X}$.
2. Given data from target f, evaluate finite difference $d_{i,k} = \frac{\|f(x^{(i)}) - f(x^{(k)})\|}{\|x^{(i)} - x^{(k)}\|}$.
3. Draw m pairs with probability proportional to $d_{i,k}$.
4. Construct weights and biases: $w = s_1 \frac{x^{(i)} - x^{(k)}}{\|(x^{(i)} - x^{(k)})\|^2}$, $b = w^T x^{(i)} + s_2$.



“Sample where it matters”

Algorithm (Bolager et al., 2023)

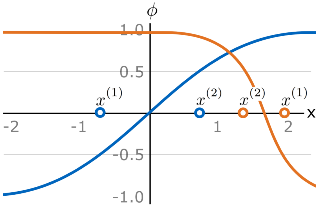
1. Sample many random data pairs $(x^{(i)}, x^{(k)})$ in the domain $\mathcal{X}$.
2. Given data from target f, evaluate finite difference $d_{i,k} = \frac{\|f(x^{(i)})-f(x^{(k)})\|}{\|x^{(i)}-x^{(k)}\|}$.
3. Draw m pairs with probability proportional to $d_{i,k}$.
4. Construct weights and biases: $w = s_1 \frac{x^{(i)}-x^{(k)}}{\|(x^{(i)}-x^{(k)})\|^2}$, $b = w^T x^{(i)} + s_2$.

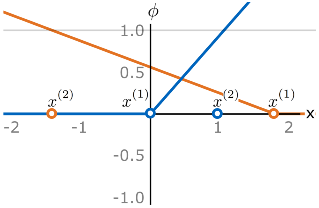


“Sample where it matters”

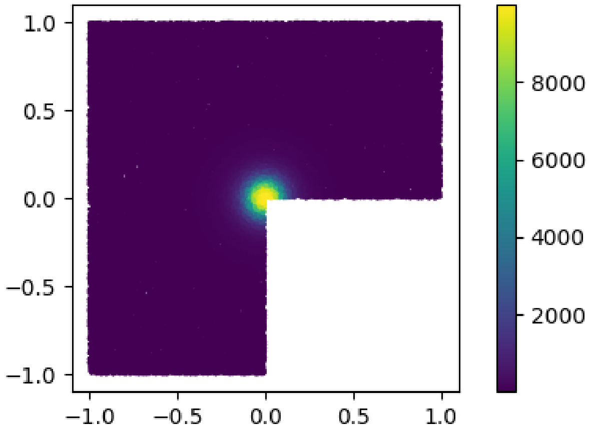
Algorithm (Bolager et al., 2023)

1. Sample many random data pairs $(x^{(i)}, x^{(k)})$ in the domain $\mathcal{X}$.
2. Given data from target f, evaluate finite difference $d_{i,k} = \frac{\|f(x^{(i)})-f(x^{(k)})\|}{\|x^{(i)}-x^{(k)}\|}$.
3. Draw m pairs with probability proportional to $d_{i,k}$.
4. Construct weights and biases: $w = s_1 \frac{x^{(i)}-x^{(k)}}{\|(x^{(i)}-x^{(k)})\|^2},\ b = w^T x^{(i)} + s_2$.



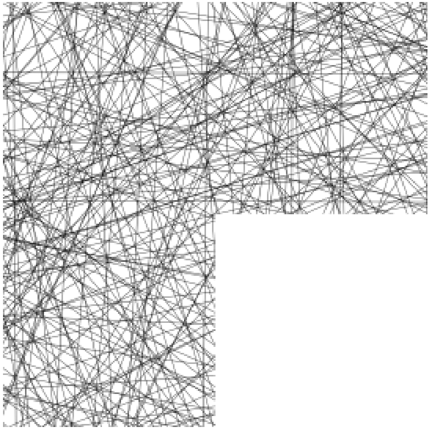


“Sample where it matters”



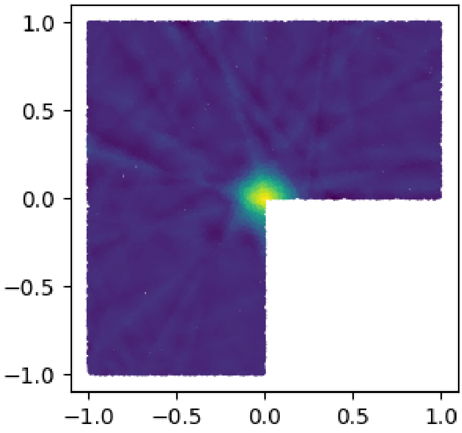
How to discretize functions on this space?
ReLU: $\max(w_k x + b_k, 0)$

“Sample where it matters”



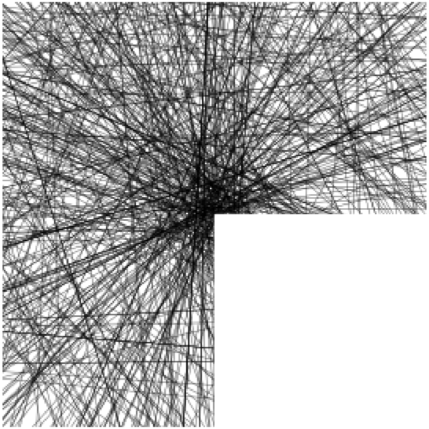
How to discretize functions on this space? Random ReLU functions?
ReLU: $\max(w_k x + b_k, 0)$

“Sample where it matters”



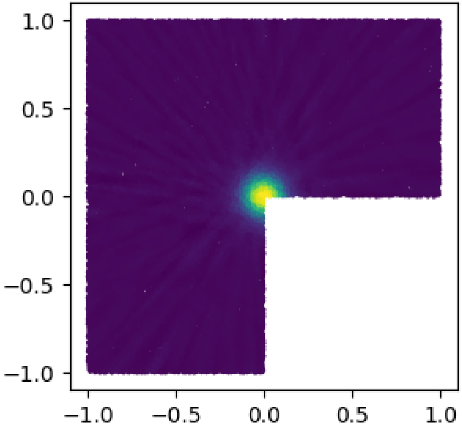
How to discretize functions on this space? Random ReLU functions?
ReLU: $\max(w_k x + b_k, 0)$

“Sample where it matters”



How to discretize functions on this space? Sample functions where it matters!
ReLU: $\max(w_k x + b_k, 0)$

“Sample where it matters”



How to discretize functions on this space? Sample functions where it matters!
ReLU: $\max(w_k x + b_k, 0)$ Also see “He and Xu: Deep Neural Networks and Finite Elements of Any Order on Arbitrary Dimensions, 2024. arXiv:2312.14276”.

“Sample where it matters”

Universal approximation

Theorem 1. *For any number of layers $L \in \mathbb{N}$, the space of sampled networks with L hidden layers and ReLU activation function is dense in $C(\mathcal{X}, \mathbb{R})$.*

Idea of proof: Show that we can approximate any neural network with one hidden layer with a sampled network that has slightly more neurons. This works differently for ReLU (homogeneous) and tanh (not homogeneous).

“Sampling data pairs from $\mathcal{X} \times \mathcal{X}$ is enough to be expressive.”

Bolager et al., “Sampling weights of deep neural networks,” NeurIPS 2023.

“Sample where it matters”

Fast convergence (breaking CoD)

Theorem 2. *Let* $f \in \mathcal{B}$ *(Barron) and* $\mathcal{X} = [0,1]^D$*. For any* $N_1 \in \mathbb{N}$*,* $\epsilon > 0$*, and an arbitrary probability measure* π*, there exist sampled networks* Φ *with one hidden layer,* N_1 *neurons, and ReLU activation function, such that*

$$\|f - \Phi\|_2^2 = \int_{\mathcal{X}} |f(x) - \Phi(x)|^2 d\pi(x) < \frac{(3+\epsilon)\|f\|_{\mathcal{B}}^2}{N_1}.$$

Idea of proof: Similar to Theorem 1, together with convergence results for general neural networks.

“The error scales independently of the input dimension.”

Bolager et al., “Sampling weights of deep neural networks,” NeurIPS 2023.

“Sample where it matters”

Equivariance

Theorem 3. *Let f be the target function and H be a scalar and rigid body transformation, equivariant with respect to f. If we have two sampled networks, $\Phi, \hat{\Phi}$, with the same number of hidden layers L and neurons $N_1, \dots, N_L$, where $\Phi \colon \mathcal{X} \to \mathbb{R}^{N_{L+1}}$ and $\hat{\Phi} \colon H(\mathcal{X}) \to \mathbb{R}^{N_{L+1}}$, then the following statements hold:*

(1) If $\hat{x}_{0,i}^{(1)} = H(x_{0,i}^{(1)})$ and $\hat{x}_{0,i}^{(2)} = H(x_{0,i}^{(2)})$, for all $i = 1, 2, \dots, N_1$, then $\Phi^{(1)}(x) = \hat{\Phi}^{(1)}(H(x))$, for all $x \in \mathcal{X}$.

(2) If H is invariant w.r.t. f, then for any parameters of Φ, there exists parameters of $\hat{\Phi}$ such that $\Phi(x) = \hat{\Phi}(H(x))$ for all $x \in \mathcal{X}$, and vice versa.

(3) The probability measure P over the parameters is invariant under H.

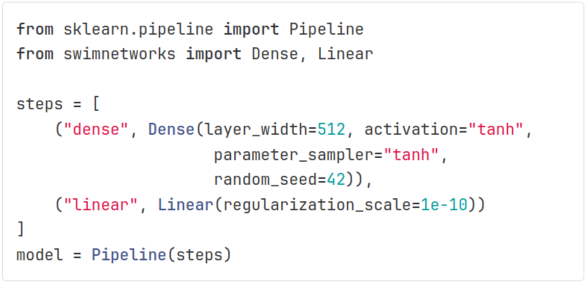
Idea of proof: Neurons can be written as $\phi(\left\langle s_1 w/\|w\|^2, x - x^{(1)} \right\rangle - s_2)$. As we divide by the $\|w\|^2$, the scalar in H cancels. There is a difference between two points, which means the translation cancels. Orthogonal matrices cancel due to isometry.

“Rigid motion and scaling of the input space does not matter.”

Sampling weights: Code contributions

It is super fast - try it out!

https://www.gitlab.com/felix.dietrich/swimnetworks



```python
from sklearn.pipeline import Pipeline
from swimnetworks import Dense, Linear

steps = [
    ("dense", Dense(layer_width=512, activation="tanh",
                    parameter_sampler="tanh",
                    random_seed=42)),
    ("linear", Linear(regularization_scale=1e-10))
]
model = Pipeline(steps)
```

Computational experiments (1/3)

Comparison to iterative training: OpenML classificaton benchmark

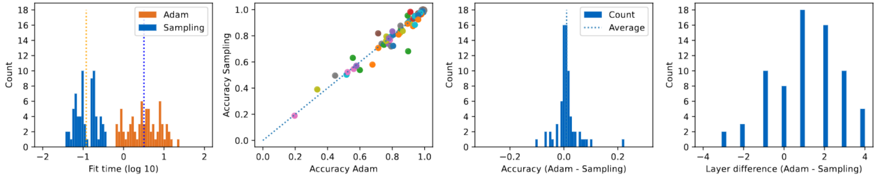
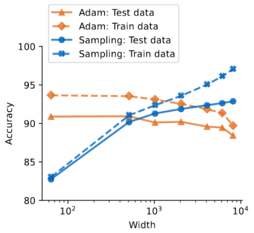


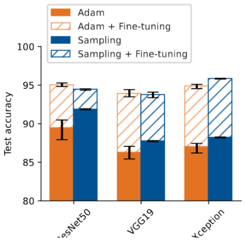
Figure 1 Fitting time, accuracy, and number of layers using weight sampling, compared to training with the Adam optimizer. The best architecture is chosen separately for each method and each problem, by evaluating 10-fold cross-validation error over $1-5$ layers with 500 neurons each.

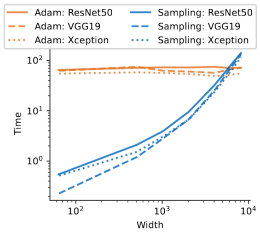
"Sampled networks are as accurate as iteratively trained ones."

Computational experiments (2/3)

Sampling for transfer learning with CNNs (work by Chinmay Datar)







Left: Train and test accuracies with different widths for ResNet50 (averaged over 5 random seeds). Middle: Test accuracy with different models with and without fine-tuning (width = 2048). Right: Adam training and sampling times of the classification head (averaged over 5 experiments).

"Fine-tuning with iterative optimization is possible."

Computational experiments (3/3)

Sampling neural operators (work by Iryna Burak and Qing Sun)

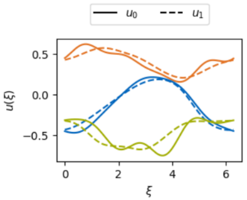
Algorithm to sample POD-DeepONet $G(u_0, x) = u(t, x)$
(a similar trick also works for sampling Fourier Neural Operators):

1. Given: collection of initial and final solutions $\{u_i(0, x), u_i(t, x)\}_{i=1}^N$ to a PDE.
2. Construct fixed, orthonormalized POD modes $T(x)$ from given solution data.
3. Define POD-DeepONet as $\hat{G}(u, x) = T(x)^T B(u)$, with “trunk” T and “branch”-net B.
4. Project final data to POD modes:
   $T(x)u(t, x) \approx T(x)\hat{G}(u(t, \cdot), x) = B(u(t, \cdot))$.
5. **Sample** branch-net $\hat{B}(u(t, \cdot)) \approx T(x)u(t, \cdot)$. This is now a supervised-learning problem!

“We can design probability distributions for network parameters that are useful to solve specific problems.”

Computational experiments (3/3)

Sampling neural operators (work by Iryna Burak and Qing Sun)

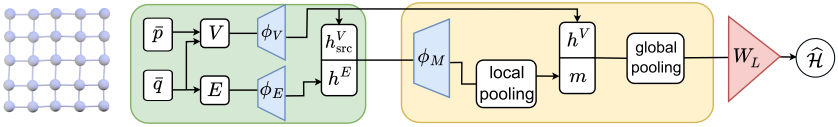


	Model	width	layers	mean rel. L^2 error	Time	
Adam	FCN; signal space	1024	2	4.48×10^{-3}	644s	GPU
	FCN; Fourier space	1024	1	3.29×10^{-3}	1725s	
	POD-DeepONet	2048	4	1.62×10^{-3}	4217s	
	FNO	n/a	4	$\mathbf{0.38 \times 10^{-3}}$	3119s	
Sampling	FCN; signal space	4096	1	0.92×10^{-3}	20s	CPU
	FCN; Fourier space	4096	1	1.08×10^{-3}	16s	
	POD-DeepONet	4096	1	$\mathbf{0.85 \times 10^{-3}}$	21s	
	FNO	4096	1	0.94×10^{-3}	387s	

Left: Samples of initial conditions u_0 and corresponding solutions u_1 for Burgers' equation. Right: Parameters of the best model for each architecture, the mean relative L^2 error on the test set, and the training time. We average the metrics across three runs with different random seeds.

"Other architectures can be sampled, too."

Learning Hamiltonian dynamics on graphs

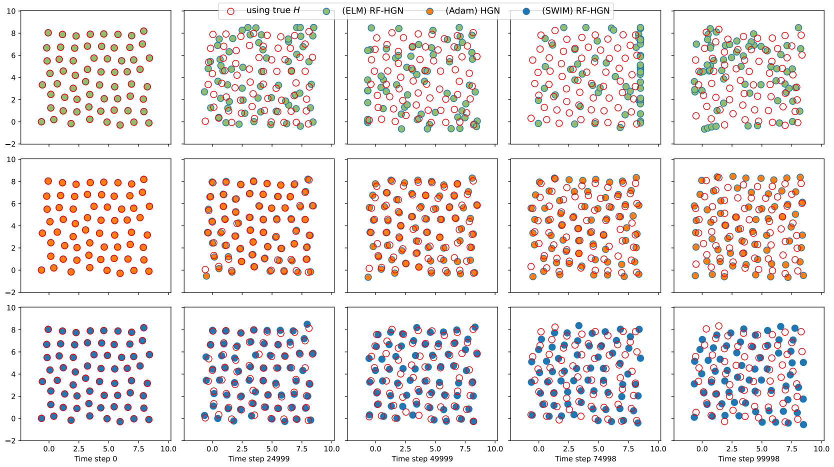


Hamiltonian **graph neural network** constructed with random feature layers. The network solves a linear PDE defined on a high-dimensional base space (all nodes of the graph combined) for H, so that

$$\dot{q} = \frac{\partial H}{\partial p},\ \dot{p} = -\frac{\partial H}{\partial q}.$$

Rahma, Datar, Cukarska, and D. “Rapid Training of Hamiltonian Graph Networks without Gradient Descent.” Preprint, `http://arxiv.org/abs/2506.06558`.

Learning Hamiltonian dynamics on graphs

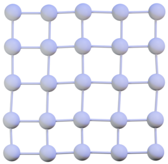


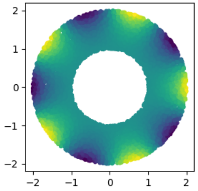
Rahma, Datar, Cukarska, and D. “Rapid Training of Hamiltonian Graph Networks without Gradient Descent.” Preprint, `http://arxiv.org/abs/2506.06558`.

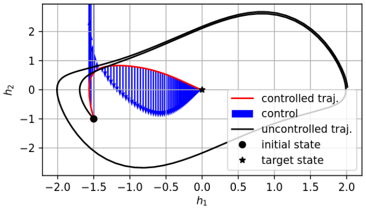
Machine learning without SGD

We now turn this idea into a scientific program.

1. Feed-forward networks (Bolager et al. [2023]).
2. Solving partial differential equations (pre-print Datar et al. [2024]).
3. Training recurrent neural networks (pre-print Bolager et al. [2024]).
4. Learning Hamiltonian dynamics on graphs (Rahma et al. [2024, 2025]).
5. Training spiking neural networks (pre-print submitted).
6. “Soon”: training convolutional networks and transformers.







Forward problem: Solving PDE

Setting

Domain $\Omega \subset \mathbb{R}^d$, function space $\mathcal{F} = \{h : \Omega \to \mathbb{R}\}$, forcing $f : \Omega \to \mathbb{R}$, boundary data $g : \partial\Omega \to \mathbb{R}$, linear operators L, B. **Find** $u \in \mathcal{F}$, s.t.

$$\begin{aligned} Lu(x) &= f(x),\ \forall x \in \Omega, \\ Bu(x) &= g(x),\ \forall x \in \partial\Omega. \end{aligned}$$

Numerical solution (Galerkin method)

1. Choose basis functions $\{h_i \in \mathcal{F}\}_i$.
2. Write ansatz as linear combination $\hat{u} = \sum_i c_i h_i$.
3. Apply L and B to ansatz (and thus individual bases h_i).
4. Obtain (large) linear system: $L\hat{u} = \sum_i c_i L h_i = f, B\hat{u} = \sum_i c_i B h_i = g$.
5. Solve for c_i.

Classical choices for h_i: splines, Gaussians, (trigonometric) polynomials.

Neural networks as ansatz functions

Mathematical perspective: Why are networks useful?

- Universal approximation (for C^0 functions) [1,4].
- Breaking the curse of dimensionality (for Barron functions) [2,3,4,5].
- Better than any linear method (at breaking the curse for certain Barron functions) [5].
- Overparametrization improves test error [6].
- For PDE: Global basis functions, spectral convergence.
- For PDE: Mesh-free approach, easy to use.

[1] Cybenko, "Approximation by superpositions of a sigmoidal function," Math. Cont. Sig. Sys., 1989.
[2] Barron, "Universal approximation bounds for superpositions of a sigmoidal function," IEEE Trans. Inform. Theory, 1993.
[3] Rahimi and Recht, "Uniform approximation of functions with random bases," in Allerton Conf. on Communication, Control, and Computing, 2008.
[4] E, "Towards a Mathematical Understanding of Neural Network-Based Machine Learning: What We Know and What We Don't," CSIAM-AM, 2020.
[5] Wu and Long, "A Spectral-Based Analysis of the Separation between Two-Layer Neural Networks and Linear Methods," J. Mach. Learn. Res., 2022.
[6] Belkin, Hsu, Ma, and Mandal, "Reconciling Modern Machine Learning Practice and the Bias-Variance Trade-Off." PNAS, 2019.

Solving PDE with neural networks

We can still use the Galerkin method!

1. Choose basis functions $\{\phi_i \in \mathcal{F}\}_i$. $\phi_i(x) = \sigma(w_i^T x + b_i)$.
2. Write ansatz as linear combination $\hat{u} = \sum_i c_i \phi_i$. $\hat{u}(x) = \sum_i c_i \sigma(w_i^T x + b_i)$.
3. Apply L and B to ansatz (and thus individual bases ϕ_i).
4. Obtain (large) linear system: $L\hat{u} = \sum_i c_i L\phi_i = f, B\hat{u} = \sum_i c_i B\phi_i = g$.
5. Solve for c_i.

Solving PDE with neural networks

We can still use the Galerkin method!

1. Choose basis functions $\{\phi_i \in \mathcal{F}\}_i$. $\phi_i(x) = \sigma(w_i^T x + b_i)$.
2. Write ansatz as linear combination $\hat{u} = \sum_i c_i \phi_i$. $\hat{u}(x) = \sum_i c_i \sigma(w_i^T x + b_i)$.
3. Apply L and B to ansatz (and thus individual bases ϕ_i).
4. Obtain (large) linear system: $L\hat{u} = \sum_i c_i L\phi_i = f, B\hat{u} = \sum_i c_i B\phi_i = g$.
5. Solve for c_i.

[1] Suchuan Dong, ad Zongwei Li. "Local Extreme Learning Machines and Domain Decomposition for Solving Linear and Nonlinear Partial Differential Equations." Computer Methods in Applied Mechanics and Engineering 387 (2021): 114129.
[2] Suchuan Dong, and Jielin Yang. "On Computing the Hyperparameter of Extreme Learning Machines: Algorithm and Application to Computational PDEs, and Comparison with Classical and High-Order Finite Elements." Journal of Computational Physics 463 (2022): 111290.
[3] Yiran Wang, and Suchuan Dong. "An Extreme Learning Machine-Based Method for Computational PDEs in Higher Dimensions." arXiv:2309.07049, 2023.
[4] Yong Shang, and Fei Wang. "Randomized Neural Networks with Petrov–Galerkin Methods for Solving Linear Elasticity and Navier–Stokes Equations." Journal of Engineering Mechanics 150, no. 4, 04024010, 2024.
[5] Jingbo Sun, Suchuan Dong, and Fei Wang. "Local Randomized Neural Networks with Discontinuous Galerkin Methods for Partial Differential Equations." Journal of Computational and Applied Mathematics 445, 115830, 2023.

Solving PDE with neural networks
Example: Poisson equation

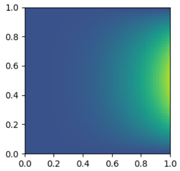
Solve $\Delta u = f$ in $\Omega = B(0,1) \subset \mathbb{R}^d$, with boundary $u = g$ on $\partial\Omega$.

$$\begin{aligned}
\text{Ansatz: } \hat{u} &= \sum_{i=1}^{m} c_i \sigma(\langle w_i, x\rangle + b_i) \\
\nabla \hat{u} &= \sum_{i=1}^{m} c_i \sigma^{'}(\langle w_i, x\rangle + b_i) w_i \\
\nabla^2 \hat{u} &= \sum_{i=1}^{m} c_i \sigma^{''}(\langle w_i, x\rangle + b_i)(w_i \times w_i) \\
f = \Delta \hat{u} = \mathrm{Tr}(\nabla^2 u) &= \sum_{i=1}^{m} \|w_i\|_2^2 c_i \sigma^{''}(\langle w_i, x\rangle + b_i).
\end{aligned}$$

Sample weights w_i and biases b_i, then solve linear system for coefficients c_i!

Solving PDE: experiments (summary)

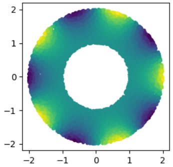
Poisson: $\Delta u = f$



#Neurons: 512
#Points: 2500

Runtime: .4s

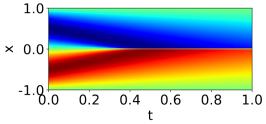
Harmonics: $\Delta u = 0$



#Neurons: 512
#Points: 2500

Runtime: .6s

Time-dependent (Burgers)

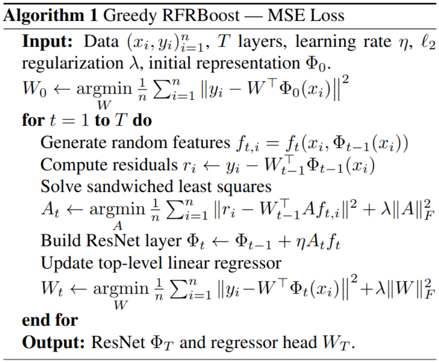


#Neurons: 450
#Points: 6000

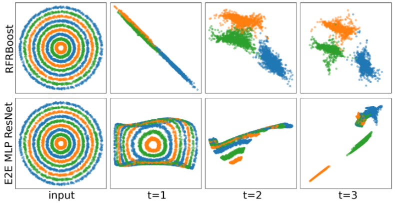
Runtime: 5s,
Rel. L^2 error: 1e-7

Datar et al. "Solving partial differential equations with sampled neural networks," 2024, pre-print: arxiv.org/abs/2405.20836

Feature learning (work of Lukas Gonon et al.)



```
Algorithm 1 Greedy RFRBoost — MSE Loss
  Input: Data $(x_i, y_i)_{i=1}^n$, $T$ layers, learning rate $\eta$, $\ell_2$
  regularization $\lambda$, initial representation $\Phi_0$.
  $W_0 \leftarrow \operatorname*{argmin}_W \frac{1}{n}\sum_{i=1}^n \|y_i - W^\top \Phi_0(x_i)\|^2$
  for $t = 1$ to $T$ do
    Generate random features $f_{t,i} = f_t(x_i, \Phi_{t-1}(x_i))$
    Compute residuals $r_i \leftarrow y_i - W_{t-1}^\top \Phi_{t-1}(x_i)$
    Solve sandwiched least squares
    $A_t \leftarrow \operatorname*{argmin}_A \frac{1}{n}\sum_{i=1}^n \|r_i - W_{t-1}^\top A f_{t,i}\|^2 + \lambda \|A\|_F^2$
    Build ResNet layer $\Phi_t \leftarrow \Phi_{t-1} + \eta A_t f_t$
    Update top-level linear regressor
    $W_t \leftarrow \operatorname*{argmin}_W \frac{1}{n}\sum_{i=1}^n \|y_i - W^\top \Phi_t(x_i)\|^2 + \lambda \|W\|_F^2$
  end for
  Output: ResNet $\Phi_T$ and regressor head $W_T$.
```



Zozoulenko, Cass, and Gonon. 2025. “Random Feature Representation Boosting.” ICML 2025. https://icml.cc/virtual/2025/poster/44355.

Computing the gradient w.r.t. the representation, not weights!

Reconstructing neural networks

What is required to identify network parameters?

$$u(x) = \sum_k^m c_k \sigma(W^T x + b) \implies u''(x) = \sum_k^m c_k \sigma''(w_k^T x + b_k)(w_k \otimes w_k)$$

Knowing the Hessian of a network allows to extract the hidden weights!
Algorithm:

1. Sample Hessians $Hu(x_i)$ at random locations x_i.
2. Compute projection operator P onto the subspace spanned by those Hessians.
3. Solve $\max_w \|P(w \otimes w)\|$ with random initial w.

Fornasier, Klock, Mondelli, and Rauchensteiner. 2022. "Finite Sample Identification of Wide Shallow Neural Networks with Biases." arXiv:2211.04589. Preprint, arXiv, November 8. `http://arxiv.org/abs/2211.04589`.

Open questions

Open questions

- Can we incorporate the PDE itself into the sampling?
- Can we also sample useful parameters of deep networks?
- Can we (improve) sampling of neural operators?

Challenges and future work

- Better theory for generalization and convergence beyond data-agnostic features.
- Higher-dimensional base spaces.
- Random features for other architectures (CNN, Transformer).

Summary and future work

Summary

Data-driven random features offer a good basis to learn functions.

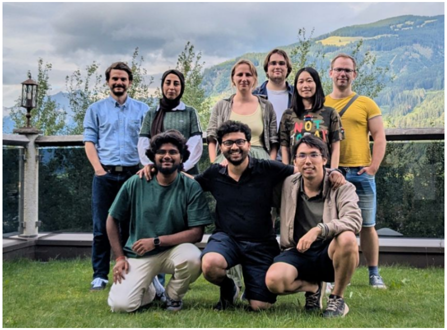
- Fast and efficient training (sampling + linear solve)
- Interpretable network parameters (one pair of data points per neuron)
- Allow us to use trustworthy optimization methods (linear solvers)

Many challenges remain!

Literature

- Sampling: Bolager et al., NeurIPS 2023 (arxiv.org/abs/2306.16830)
- Learning Hamiltonians pre-print: arxiv.org/abs/2506.06558
- Solving PDE pre-print: arxiv.org/abs/2405.20836

Discussion


Felix, Zahra, Iryna, Erik, Qing, Vladyslav, Shyam, Chinmay, Hessel. Not in the picture: Ana, Atamert, Berkay, Felix S., Nadiia, Rahul.

Summary

Data-driven random features offer a new perspective on neural networks.

Contact

`felix.dietrich@tum.de`
`www.cs.cit.tum.de/en/scml`

Funding

DFG
SPONSORED BY THE Federal Ministry of Education and Research
MDSI
Georg Nemetschek Institute

Literature I

A.R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993. ISSN 0018-9448, 1557-9654. doi: 10.1109/18.256500.

Erik Lien Bolager, Iryna Burak, Chinmay Datar, Qing Sun, and Felix Dietrich. Sampling weights of deep neural networks. Advances In Neural Information Processing Systems, NeurIPS 2023, arXiv: 2306.16830, June 2023.

Erik Lien Bolager, Ana Cukarska, Iryna Burak, Zahra Monfared, and Felix Dietrich. Gradient-free training of recurrent neural networks, October 2024.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989. ISSN 1435-568X. doi: 10.1007/BF02551274.

Chinmay Datar, Taniya Kapoor, Abhishek Chandra, Qing Sun, Iryna Burak, Erik Lien Bolager, Anna Veselovska, Massimo Fornasier, and Felix Dietrich. Solving partial differential equations with sampled neural networks, May 2024.

Suchuan Dong and Zongwei Li. Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 387:114129, December 2021. ISSN 00457825. doi: 10.1016/j.cma.2021.114129.

Suchuan Dong and Jielin Yang. On computing the hyperparameter of extreme learning machines: Algorithm and application to computational PDEs, and comparison with classical and high-order finite elements. *Journal of Computational Physics*, 463:111290, August 2022. ISSN 00219991. doi: 10.1016/j.jcp.2022.111290.

Weinan E. Towards a Mathematical Understanding of Neural Network-Based Machine Learning: What We Know and What We Don’t. *CSIAM Transactions on Applied Mathematics*, 1(4):561–615, June 2020. ISSN 2708-0560, 2708-0579. doi: 10.4208/csiam-am.SO-2020-0002.

Juncai He and Jinchao Xu. Deep Neural Networks and Finite Elements of Any Order on Arbitrary Dimensions, January 2024.

D. P. Kingma and L. J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations ICLR 2015*, 2015.

Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561, Monticello, IL, USA, September 2008. IEEE. ISBN 978-1-4244-2925-7. doi: 10.1109/ALLERTON.2008.4797607.

Atamert Rahma, Chinmay Datar, and Felix Dietrich. Training Hamiltonian neural networks without backpropagation, November 2024.

Atamert Rahma, Chinmay Datar, Ana Cukarska, and Felix Dietrich. Rapid training of Hamiltonian graph networks without gradient descent, June 2025.

Yong Shang and Fei Wang. Randomized Neural Networks with Petrov–Galerkin Methods for Solving Linear Elasticity and Navier–Stokes Equations. *Journal of Engineering Mechanics*, 150(4):04024010, April 2024. ISSN 0733-9399, 1943-7889. doi: 10.1061/JENMDT.EMENG-7463.

Literature II

Jingbo Sun, Suchuan Dong, and Fei Wang. Local randomized neural networks with discontinuous Galerkin methods for partial differential equations. *Journal of Computational and Applied Mathematics*, 445:115830, August 2024. ISSN 03770427. doi: 10.1016/j.cam.2024.115830.

Yiran Wang and Suchuan Dong. An Extreme Learning Machine-Based Method for Computational PDEs in Higher Dimensions, September 2023.

Lei Wu and Jihao Long. A Spectral-Based Analysis of the Separation between Two-Layer Neural Networks and Linear Methods. *Journal of Machine Learning Research*, 23(1), January 2022. ISSN 1532-4435.