WIP: Energy Optimized Piecewise Polynomial Approximation Utilizing Modern Machine Learning Optimizers

Hannes Waclawek Stefan Huber

Josef Ressel Center for Intelligent and Secure Industrial Automation
Department for Information Technologies and Digitalisation

Salzburg University
of Applied Sciences

AI4IP Workshop, 36th DEXA conferences and workshops
Bangkok, Thailand
August 27, 2025

Interreg Bayern-Österreich

Kofinanziert von der Europäischen Union

Problem setting

Question

Can we leverage piecewise polynomial (PP) models as parameterized machine learning models in a supervised learning setting, while also taking domain-specific optimization goals into account?

Motivating examples from Interreg AI4GREEN:

- Optimizing cam motion profiles for energy consumption
- Spectral shaping of motion profiles for vibration reduction

Why polynomials?

One step back: Supervised learning in action

In a nutshell, in supervised learning we are given

- a dataset $D = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ of input-output pairs $(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}^m$.

Goal: Given a hypothesis class $H_\Theta = \{h_\theta : \theta \in \Theta\}$ of parameterized hypothesis $h_\theta : \mathbb{R}^n \to \mathbb{R}^m$, where Θ is the parameter space,

- find the most likely hypothesis h_{θ^*} that explains[1] the data D best, i.e.,
- where the loss $\ell(\theta|D) = \sum_{(x,y)\in D}(h_\theta(x) - y)^2$ is minimized:

$$\theta^* = \arg\min_{\theta\in\Theta} \ell(\theta|D).$$

This optimization task is performed by modern, powerful, gradient-based optimizers, like Adam. All they need:

- gradients $\nabla_\theta \ell(\theta|D)$,
- which are provided by so-called auto-diff engines in ML frameworks like PyTorch and TensorFlow.

[1] Assuming normal distributed data, minimizing this ℓ results in the maximum liklihood estimator θ^*.

One step back: Supervised learning in action

In a nutshell, in supervised learning we are given

- a dataset $D = \{(x_1, y_1), \ldots, (x_k, y_k)\}$ of input-output pairs $(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}^m$.

Goal: Given a hypothesis class $H_\Theta = \{h_\theta : \theta \in \Theta\}$ of parameterized hypothesis $h_\theta : \mathbb{R}^n \to \mathbb{R}^m$, where Θ is the parameter space,

- find the most likely hypothesis h_{θ^*} that explains[1] the data D best, i.e.
- where the loss $\ell(\theta|D)$

This is a general setting, for any concrete hypothesis class H_Θ.

This optimization task is performed by modern, powerful, gradient-based optimizers, like Adam. All they need:

- gradients $\nabla_\theta \ell(\theta|D)$,
- which are provided by so-called auto-diff engines in ML frameworks like PyTorch and TensorFlow.

[1] Assuming normal distributed data, minimizing this ℓ results in the maximum liklihood estimator θ^*.

Neural networks

In general, the go-to model. But for specific applications we may find better ways, e.g.,

- for low-dimensional supervised learning in cyber-physical systems with domain-specific optimization objectives?

Pros

- Very versatile, including for high-dimensional input spaces $\mathbb{R}^n$ or output spaces $\mathbb{R}^m$, at least for deep neural networks
- Highly optimized software and hardware

Cons

- Black-box models, i.e., the obtained h_θ hard to interpret and explain
  - Deep nets $\Rightarrow$ concatenation of non-linear activation functions
  - (Symbolic) derivatives for physical interpretation difficult to obtain
- Efficient deployment requires dedicated hardware, like GPUs and NPUs, not ubiquitous in cyber-physical systems
- Approximation theory of limited use

Alternatives to neural networks

Our approach

Treat the universal approximation property as first principle.[2]

Mathematical modeling in classical engineering is typically done with polynomials:

- Are well understood, e.g., best approximation theory, Stone-Weierstrass theorem for uniform approximation of continuous functions
- Algebraically and computationally tangible, e.g., form an algebraic ring, easy to symbolically derive
- Are well supported in existing cyber-physical systems, e.g., the cam models in servo drives

$$\text{neural nets : GPU/NPU} \quad \widehat{=} \quad \text{piecewise polynomials : servo drives}^3$$

Question

Can piecewise-polynomial functions act as H_Θ in practice in PyTorch or TensorFlow?

[2] Neural nets were inspired by biological neural networks, i.e., McCulloch–Pitts neuron, Rosenblatt perceptron.

[3] The cam profile automaton (e.g., in B&R ACOPOS drive) is a state machine, where states have cam profiles and, in a sense, resembles a computational model that executes cam profiles. For ACOPOS, each cam can comprise 128 polynomials of degree 6.

How?

Piecewise-polynomial models

For a bounded interval $I \subset \mathbb{R}$ we denote by $\mathcal{C}^k(I)$ the set of k-times continuous differentiable functions $I \to \mathbb{R}$.

We call $f\colon I \to \mathbb{R}$ a piecewise polynomial (PP) function of degree d, if

- there are split points $\xi_0 < \cdots < \xi_m$ such that we can tessellate I into m sub-intervals

$$I = \underbrace{[\xi_0, \xi_1)}_{I_1} \cup \underbrace{[\xi_1, \xi_2)}_{I_2} \cup \cdots \cup \underbrace{[\xi_{m-1}, \xi_m]}_{I_m}$$

- and there are m polynomial functions $p_1, \ldots, p_m$ of degree d, with

$$p_i(x) = \sum_{j=0}^{d} \alpha_{i,j}\, x^j$$

- such that $f(x) = p_i(x)$ when $x \in I_i$ for all $x \in I$.

For simplicity, we fix ξ_i to be equidistant over I. Then f is fully determined by the $m \cdot (d+1)$ model parameters $\alpha_{i,j}$, i.e., the hypothesis h_θ is f. We will write f_θ to emphasize this.

Piecewise-polynomial models

For a bounded interval $I \subset \mathbb{R}$ we denote by $\mathcal{C}^k(I)$ the set of k-times continuous differentiable functions $I \to \mathbb{R}$.

We call $f : I \to \mathbb{R}$ a piecewise polynomial (PP) function of degree d, if

- there are s… ls
- and there a…
- such that $f(x) = p_i(x)$ when $x \in I_i$ for all $x \in I$.

Research programme:

- Implement them into a ML framework PyTorch or ThensorFlow, whose auto-diff engines compute $\nabla_\theta f_\theta$
- Leverage modern, powerful SGD optimizers, like Adam or AMSGrad, to train these models
- Enable domain-specific optimization objectives, like energy reduction

For simplicity, we fix ξ_i to be equidistant over I. Then f is fully determined by the $m \cdot (d+1)$ model parameters $\alpha_{i,j}$, i.e., the hypothesis h_θ is f. We will write f_θ to emphasize this.

Prior work: $\mathcal{C}^k$-continuous approximation objective

For many industrial applications, we require the PP function f_θ to be $\mathcal{C}^k$-continuous:

- If f_θ is a cam profile then $k = 2$ means continuous acceleration, $k = 3$ means continuous jerk.

Assume we are given a dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n points $(x_i, y_i) \in \mathbb{R} \times \mathbb{R}$.

In [HW23] we defined two loss terms:

$$\ell_2(\theta) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 \quad \text{and} \quad \ell_{\mathrm{CK}}(\theta) = \frac{1}{m-1}\sum_{i=1}^{m-1}\sum_{j=0}^{k}\Delta_{i,j}^2 \text{ with } \Delta_{i,j} = \underbrace{p_{i+1}^{(j)}(\xi_i) - p_i^{(j)}(\xi_i)}_{\text{jump in } j\text{-th deriv. at } \xi_i}$$

Note that $\ell_{\mathrm{CK}}(\theta) = 0$ means that $f_\theta \in \mathcal{C}^k(I)$.[4] Hence, we are interested in the constrained optimization

$$\arg\min_\theta \ell_2(\theta) \quad \text{s.t.} \quad \ell_{\mathrm{CK}}(\theta) = 0.$$

[4] We can make f_θ periodic by adding $i = 0$ in ℓ_{CK}. We might be interested in the weaker property "cyclicity" for cams, where we add $i = 0$ only for $j \geq 1$.

Prior work: $\mathcal{C}^k$-continuous approximation objective

In [HW23] we solved the constrained problem as follows:

1. Turn it into an unconstrained version

$$\arg\min_\theta \lambda \ell 2(\theta) + (1-\lambda)\ell_{\mathsf{CK}}(\theta),$$

   compatible to auto-diff engines.
2. Apply our algorithm CKMIN (from [HW23]), which establishes $\ell_{\mathsf{CK}}(\theta) = 0$ by adding certain correction polynomials. (Requires $d \geq 2k+1$.)

Key insights from [HW23] and [WH25]:

- Practical implementation with Horner scheme plus shift-and-scale of I_i to $[-1, 1]$.
- Switching from the power basis to Chebyshev polynomials is beneficial, cf. [WH25].
- In [HW23] and [WH25], regularization techniques were presented to enable/improve convergence.
- Initializing f_θ with the explicitly computed ℓ_2-optimum improves performance.

This paper: How about domain-specific objectives?

What remained open, both in [HW23] and [WH25], is this:

- Utilizing auto-diff engines of ML frameworks would allow us to easily incorporate various domain-specific optimization goals.
- How would these techniques perform with energy-optimized PP functions?

We define the energy loss ℓ_{E} as

$$\ell_{\mathrm{E}}(\theta) = \int_I f_\theta''(x)^2 \mathrm{d}x.$$

Note that ℓ_{E} is related to "energy" in different ways:

- It is the elastic strain energy of a rod bent like the graph of f_θ.
- If f_θ is a position signal of a servo drive then the acceleration f_θ'' is proportional to the motor coil current, i.e., ℓ_{E} represents the copper losses.

Multi-objective optimization with energy loss

In the auto-diff framework (e.g., TensorFlow), we implement this closed-form formula for $\ell_{\mathsf{E}}(\theta)$:

$$\ell_{\mathsf{E}}(\theta) = \int_I f_\theta''(x)^2 \mathrm{d}x = \sum_{i=1}^{m} \sum_{j=2}^{d} \sum_{k=2}^{d} \alpha_{i,j}\alpha_{i,k} \frac{jk(j-1)(k-1)}{j+k-3} \left(\xi_i^{j+k-3} - \xi_{j-1}^{j+k-3} \right).$$

Our energy optimization problem then becomes a two-objective, constrained optimization:

$$\begin{aligned} \min_{\theta} \quad & \beta \ell_2(\theta) + (1-\beta)\ell_{\mathsf{E}}(\theta) \\ \text{s.t.} \quad & \ell_{\mathsf{CK}}(\theta) = 0 \end{aligned}$$

The parameter $\beta \in [0, 1]$ sets the preference of ℓ_2 over ℓ_{E}. Has to be provided by the user.

Integration into auto-diff framework

We generalize [HW23]: Turn the original problem into an unconstrained 3-object optimization problem

$$\min_{\theta} \quad \alpha \ell_{\mathsf{CK}}(\theta) + \beta \ell_2(\theta) + (1 - (\alpha + \beta)) \ell_{\mathsf{E}}(\theta)$$

with $\alpha, \beta \geq 0$ and $\alpha + \beta \leq 1$.

Implementation[5]:

- We implemented this into TensorFlow, enabling the whole battery of optimizers.
- We again apply the CKMIN algorithm to establish $\ell_{\mathsf{CK}}(\theta) = 0$.

[5] Available at https://github.com/JRC-ISIA/paper-2025-energy-ml-optimized-pp

Experimental evaluation: A toy dataset

Requirements on the dataset:

- Data that displays enough "variety" such that ℓ_2-minimization does not implicitly solve ℓ_E-minimization en passant.
- That is, ℓ_2 and ℓ_E should compete each other

100 points x_i equidistant and

- $y_i = \sin(4\pi x_i^2) + \mathcal{N}(0, 0.1)$.
- Gaussian noise $\mathcal{N}(0, 0.1)$ makes f_θ "squiggle" when ℓ_2 becomes low and hence competes with ℓ_E.

Reminder: Loss was
$\alpha\ell_{CK} + \beta\ell_2 + (1 - (\alpha + \beta))\ell_E$

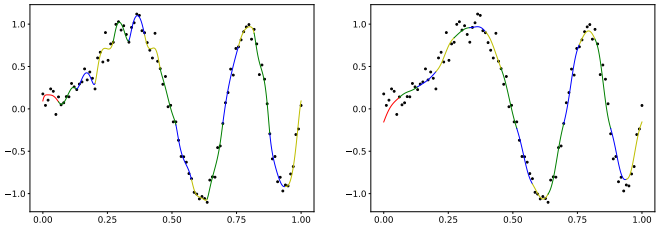

Figure: $m = 16$ polynomials of degree $d = 7$. We have $k = 2$ and $\alpha = 0.1$. Left: $\beta = 0.9$ (no ℓ_E), right: $\beta = 0.45$.

Experimental evaluation: Preliminary tests

A couple of preliminary tests reveal:

- Adam- and AMSGrad-based optimizers perform best over different input data, with default parameters.
- $\mathcal{C}^k$-regularization helps, but variance in the input data makes the gap to close.
- When f_θ is initialized wit ℓ_2-optimum, power basis is competitive with Chebyshev basis when ℓ_E is not optimized for. (Still Chebyshev converges faster.)
- Optimizing for ℓ_E, i.e., $\alpha + \beta < 1$, makes power basis more preferable than Chebyshev.

Experimental evaluation: Pareto front

- AMSGrad in TensorFlow, 1000 epochs with early stopping (patience of 100 epochs)
- $k = 2$ for $\mathcal{C}^k$-continuity, degree $d = 7$ with $m = 16$ polynomials (power basis)
- CKMIN applied to establish $\ell_{\mathrm{CK}} = 0$

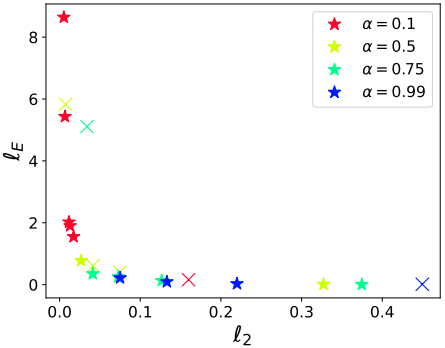

Figure: ℓ_{E} over ℓ_2 for different α and $\beta \in (0, 1-\alpha]$. $\star$ is Pareto optimal.

Parameters	ℓ_2	$\ell_{\mathbf{E}}$
$\alpha = 0.10, \beta = 0.900$	0.005	8.682
$\alpha = 0.10, \beta = 0.450$	0.014	1.970
$\alpha = 0.50, \beta = 0.500$	0.007	5.468
$\alpha = 0.50, \beta = 0.250$	0.046	0.616
$\alpha = 0.75, \beta = 0.250$	0.007	5.852
$\alpha = 0.75, \beta = 0.125$	0.077	0.282
$\alpha = 0.99, \beta = 0.010$	0.043	4.977
$\alpha = 0.99, \beta = 0.005$	0.135	0.121

What's next?

Polynomial models for uni-variate functions

Open question: Why is convergence of Chebyshev impaired with ℓ_E added?

Generalization to other polynomial bases:

- Legendre polynomials are orthogonal with constant weight function
- Bernstein basis polynomials favorable from a numerical point of view: Of same degree, but add up to the constant-1 function.

Other domain-specific optimization goals:

- Spectral vibration minimization. (See [FS12] for the Fourier transform of Chebyshev and Legendre polynomials.)

Experimental validation:

- Implementation and experimental validation on a real servo drive.

Polynomials as machine learning models

Polynomials as parameterized hypothesis of the form $\mathbb{R} \to \mathbb{R}$.

- Easy to generalize to $\mathbb{R} \to \mathbb{R}^m$, by having m such polynomials
- But in general, we want $\mathbb{R}^n \to \mathbb{R}^m$...
- ...which is why we generalized to multi-variate Chebyshev polynomials in a follow-up work

Current NeurIPS 2025 submission:

- Introduced multi-variate Chebyshev polynomials as model for Reinforcement Learning, in particular policy-gradient algorithms (e.g., PPO).
- On the mountain car benchmark problem, we improved the regret (gap to optimality) by 350 %, which is a significant improvement in past 35 years.
  - Previous SOTA trained by ARS and PPO with neural nets with some 4000 parameters
  - Our model has 268× less parameters and trained with baseline REINFORCE in 13 % of steps
  - Explainable and interpretable, symbolic derivatives are trivial
- Primary application domain: Low-dimensional control tasks

Thank you

Bibliography I

[FS12] A. S. Fokas and S. A. Smitheman. *The Fourier Transforms of the Chebyshev and Legendre Polynomials*. 2012. URL: https://arxiv.org/abs/1211.4943.

[HW23] Stefan Huber and Hannes Waclawek. "C^k-continuous Spline Approximation with TensorFlow Gradient Descent Optimizers." In: *2022*. Cham: Springer Nature Switzerland, 2023-02, pp. 577–584. ISBN: 978-3-031-25312-6. DOI: 10.1007/978-3-031-25312-6_68. URL: https://arxiv.org/abs/2303.12454.

[WH25] Hannes Waclawek and Stefan Huber. "Machine Learning Optimized Orthogonal Basis Piecewise Polynomial Approximation." In: *Learning and Intelligent Optimization*. Cham: Springer Nature Switzerland, 2025, pp. 427–441. ISBN: 978-3-031-75623-8. DOI: 10.1007/978-3-031-75623-8_33.