Second-Order Differential Machine Learning

Neil Kichler

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

Domain: Surrogate Models

Monte Carlo methods for sampling option pricing models are expensive. Instead:

- Find efficient **neural-network** based **surrogates** for option pricing models.
- Can we go **beyond Differential Machine Learning** as proposed by Huge & Savine [1], using second-order information?

Key Idea

Add **second-order differential data** to the learning process [2]:

- Use PCA via SVD to find meaningful directions.
- Apply Hessian Vector Product on found principal components.
- (optionally) take k_v most important principal components.
- Adaptive loss balancing using singular value information.
- Implementation on the GPU/TPU using JAX.
- Potential parallel, on-the-fly data generation on the CPU with C++ using AD [3] via, e.g., dco/c++.

Algorithm

Require: Initialized…

- Surrogate model $\mathcal{N}(\boldsymbol{\vartheta})$ with parameters $\boldsymbol{\vartheta}$.
- Reference model $\mathcal{S}$.
- Optimizer G.
- hyperparameter κ, for principal components.
- Loss function $\mathcal{L}$.
- loss balancing parameters $\lambda_0, \lambda_1, \lambda_2$.

while $\boldsymbol{\vartheta}$ not converged **do**

$\{(\boldsymbol{x}_i, \boldsymbol{y}_i, \nabla_{\boldsymbol{x}}\boldsymbol{y}_i)\}_{i=1}^m \sim \mathcal{S}$ ▷ Sample training data

$\boldsymbol{\mu} \leftarrow \{\frac{1}{m}\sum_{i=1}^m \nabla_{\boldsymbol{x}}\boldsymbol{y}_i\}$ ▷ Mean of pathwise gradients

$\{\nabla_{\boldsymbol{x}_i}\tilde{\boldsymbol{y}}_i\}_{i=1}^m \leftarrow \{\nabla_{\boldsymbol{x}}\boldsymbol{y}_i - \boldsymbol{\mu}\}_{i=1}^m$ ▷ Mean subtracted data

$(\boldsymbol{U}, \boldsymbol{s}, \boldsymbol{V}^\mathsf{T}) \leftarrow \mathrm{SVD}(\{\nabla_{\boldsymbol{x}_i}\tilde{\boldsymbol{y}}_i\}_{i=1}^m)$

$\{\tilde{\boldsymbol{v}}_k\}_{k=1}^{n_0} \leftarrow \mathtt{diag}(\boldsymbol{s})\boldsymbol{V}$ ▷ Principal components

$\{\boldsymbol{v}_k\}_{k=1}^{n_0} \leftarrow \{\tilde{\boldsymbol{v}}_k + \boldsymbol{\mu}\}_{k=1}^{n_0}$ ▷ mean adjusted

$\boldsymbol{s}_{\sigma^2} \leftarrow \boldsymbol{s}^2/\mathtt{sum}(\boldsymbol{s}^2)$ ▷ Scaled $\boldsymbol{s}$ to represent % of variance

$k_{\boldsymbol{v}} \leftarrow \arg\max(\mathtt{cumsum}(\boldsymbol{s}_{\sigma^2}) > \kappa)$

Gradient $\hat{\boldsymbol{g}}$ of minibatch:

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\vartheta}}\sum_{i=1}^{m}\Bigg[\lambda_0\mathcal{L}(f_{\boldsymbol{\vartheta}}(\boldsymbol{x}_i), \boldsymbol{y}_i) + \lambda_1\mathcal{L}(\nabla_{\boldsymbol{x}} f_{\boldsymbol{\vartheta}}(\boldsymbol{x}_i), \nabla_{\boldsymbol{x}_i}\boldsymbol{y}_i) + \lambda_2\sum_{k=1}^{k_v}\mathcal{L}(\partial^2_{\boldsymbol{x}}(f_{\boldsymbol{\vartheta}})(\boldsymbol{x}_i, \boldsymbol{v}_k), \partial^2_{\boldsymbol{x}}(f)(\boldsymbol{x}_i, \boldsymbol{v}_k))\Bigg]$$

$\boldsymbol{\vartheta} \leftarrow G(\boldsymbol{\vartheta}, \hat{\boldsymbol{g}})$ ▷ Update surrogate parameters

end while

return $\mathcal{N}$

Details

How to balance the loss parameters?

⇒ Use k_v (most important principal components)

$$c = 1 + \alpha n + \beta n^2, \quad \lambda_0 = \frac{1}{c}, \quad \lambda_1 = \frac{\alpha n}{c}, \quad \lambda_2 = \frac{\beta n^2}{c},$$

where e.g., $\alpha = 1, \beta = 2k_{\boldsymbol{v}}/n^2$.

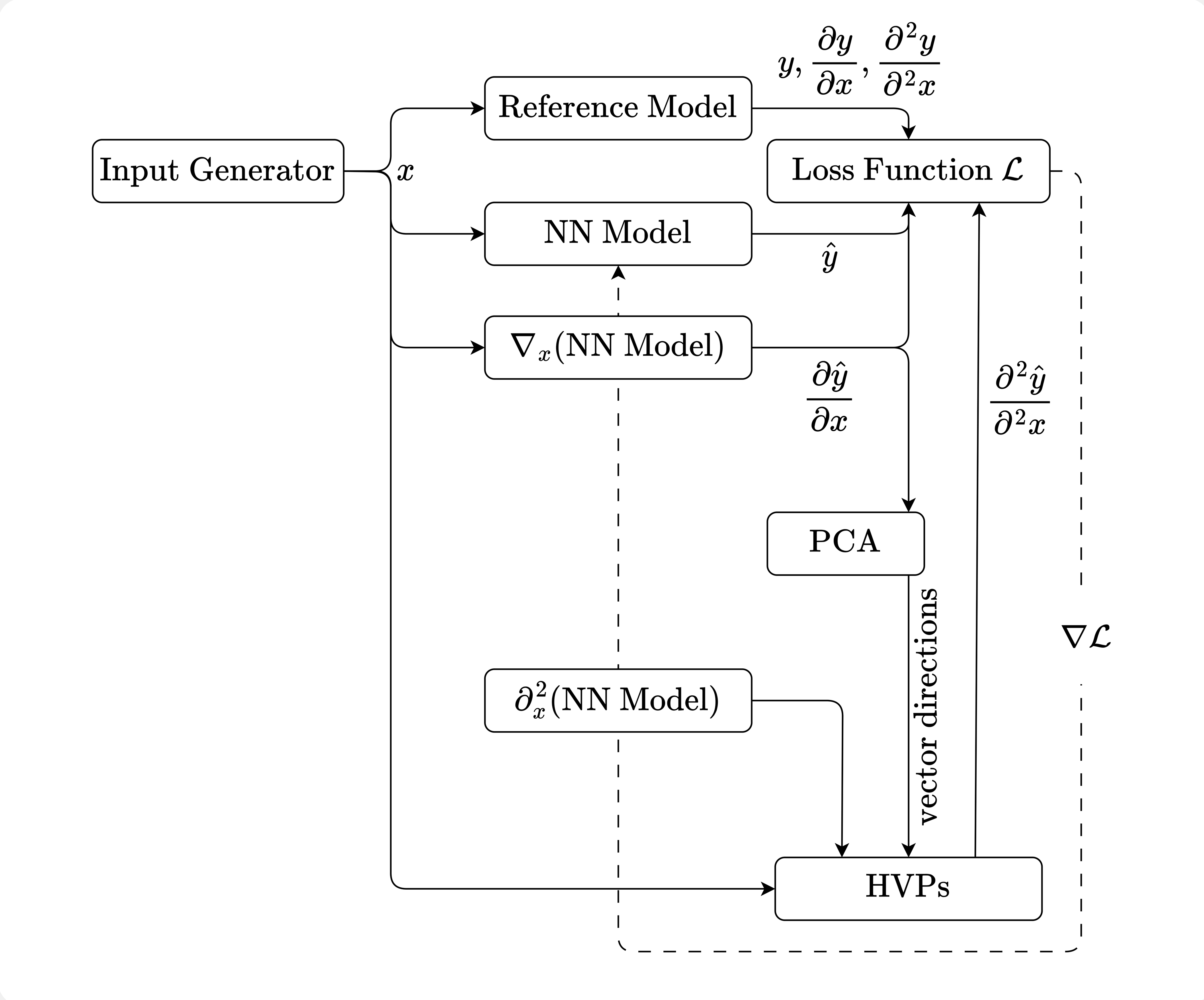
How to deal with pathwise (derivative) payoff discontinuities?

⇒ Use smoothing, e.g., sigmoidal smoothing.

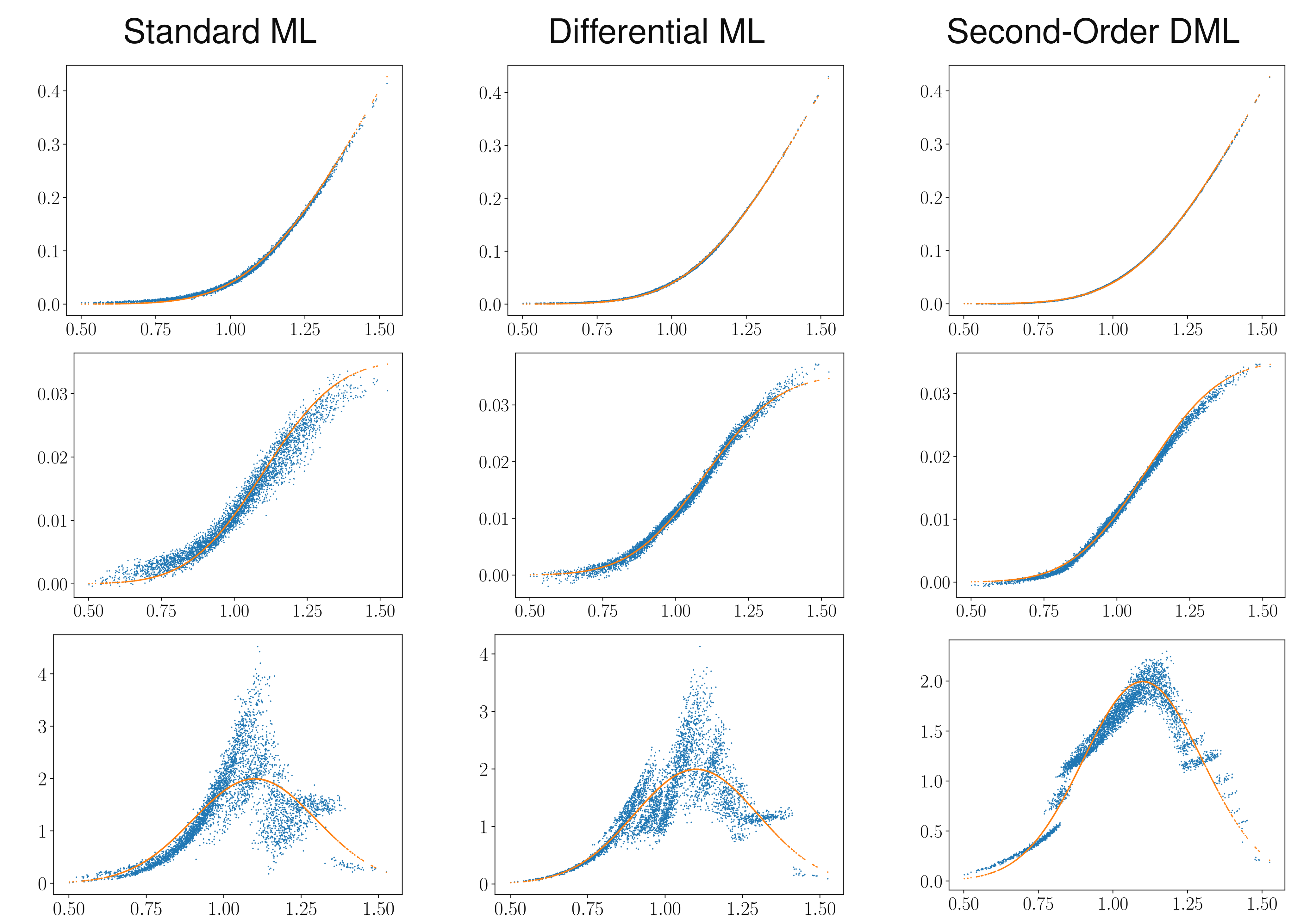
References:

1. B. N. Huge and A. Savine. "Differential Machine Learning". *Risk*, 2020.
2. N. Kichler. "Second-Order Differential ML". MA thesis. RWTH, 2023.
3. U. Naumann. *The art of differentiating computer programs.* SIAM, 2011.

Visualization of Method



Results for Bachelier Basket Option



RMSE for 30 runs with 8192 samples, maturity $T = 1$ year:

Predict	# Dim	Standard ML	Differential ML	2nd-Order PCA	2nd-Order RNG
Price	7	0.320 ± 0.022	0.123 ± 0.009	**0.101 ± 0.016**	**0.099 ± 0.004**
Delta	7	0.532 ± 0.009	0.261 ± 0.025	**0.155± 0.018**	0.231± 0.010
Gamma	7	97.625 ± 0.33	87.390 ± 2.20	**75.54± 0.51**	87.392 ± 1.31

Tested on baskets with up to **100 assets**, resulting in similar improvements.

Future Directions

- More complicated models? (E.g., Heston? ⇒ requires variance reduction)
- PCA using Krylov subspace iteration solver
- Alternatives to PCA capturing non-linearities? (e.g., Kernel PCA, Autoencoder)
- Even higher-order differential data?

Poster online: