Multilevel Monte Carlo learning

Daniel Roth

droth@qontigo.com

Joint work with Thomas Gerstner, Bastian Harrach and Martin Simon

MCM

Paris, June 30th, 2023.

Motivation

- Replace the Monte Carlo estimation with the evaluation of a deep neural network.
- Once the neural network training is done, the evaluation of the resulting approximating function is computationally highly efficient.
- Training of a suitable neural network is likely to be prohibitive in terms of computational cost.

Goal:

- Study complexity of the training process.
- Reduce complexity by a combination of:
  - Solving the Kolmogorov PDE by means of deep learning (Beck et al.)
  - Multilevel Monte Carlo (Giles)

Example practical usage: Replacing complex computations:

- new market data
- calibrate implied volatility surface
- new local volatility function $\sigma(S, t, b)$ with parameter vector $b = (b_1, \ldots, b_s) \in \mathbb{R}^s$
- Compute options' price through e.g. Monte Carlo

Reformulation:

$$dS(t) = \mu(S, t, a)\,\mathrm{d}t + \sigma(S, t, b)\,\mathrm{d}W(t)$$

$$\begin{aligned} a = (a_1, \ldots, a_m) \in \mathbb{R}^m : &\quad \text{parameter vector} \\ b = (b_1, \ldots, b_s) \in \mathbb{R}^s : &\quad \text{parameter vector} \\ \mu(S, t, a) : &\quad \text{drift} \\ \sigma(S, t, b) : &\quad \text{volatility} \end{aligned}$$

Training set $Y \subset \mathbb{R}^{m+s+r} \times \mathbb{R}^2_+$.

We will be interested in the expected value of

$$P : y \mapsto V\left(S_{a,b,s_0,T}(T), v\right),$$

for fixed

$$y := (a, b, v, s_0, T) \in Y$$

Goal:

Search neural network $\mathcal{N} : Y \to \mathbb{R}$ minimizing

$$\|\mathbb{E}[P] - \mathcal{N}\|_{L^p(Y)},$$

for $1 \leq p \leq \infty$.

Training approaches:

- Deterministically select inputs and compute outputs (e.g. via multilevel MC)
- Randomly select inputs and compute outputs (Stochastic-batch)
- Randomly select inputs and use approximations of outputs. (Beck et al. 2018).
  If Monte Carlo is used for the approximation: Monte Carlo learning

Training approaches I

1: initialize training data (Y, Z)
2: split (Y, Z) into a training set $(Y^{(1)}, Z^{(1)})$ and validation set $(Y^{(2)}, Z^{(2)})$
3: initialize neural network
4: **for** $i = 1, \ldots, K$ **do**
5: randomly select training data $(y_i^{(1)}, z_i^{(1)})$ from $(Y^{(1)}, Z^{(1)})$
6: calculate loss
7: update weights
8: **end for**

Training approaches II

Approach 1:

1: initialize training data (Y, Z)
2: split (Y, Z) into a training set $(Y^{(1)}, Z^{(1)})$ and validation set $(Y^{(2)}, Z^{(2)})$
3: initialize neural network
4: **for** $i = 1, \ldots, K$ **do**
5: randomly select training data $(y_i^{(1)}, z_i^{(1)})$ from $(Y^{(1)}, Z^{(1)})$
6: calculate loss
7: update weights
8: **end for**

Approach 2:

1: ~~initialize training data (Y, Z)~~
2: ~~split (Y, Z) into a training set $(Y^{(1)}, Z^{(1)})$ and validation set $(Y^{(2)}, Z^{(2)})$~~
3: initialize neural network
4: **for** $i = 1, \ldots, K$ **do**
5: randomly sample inputs $y_i^{(1)}$ and compute outputs $z_i^{(1)}$
6: calculate_loss
7: update_weights
8: **end for**

Training approaches III

Approach 1:

1: initialize training data (Y, Z)
2: split (Y, Z) into a training set $(Y^{(1)}, Z^{(1)})$ and validation set $(Y^{(2)}, Z^{(2)})$
3: initialize neural network
4: **for** $i = 1, \ldots, K$ **do**
5: randomly select training data $(y_i^{(1)}, z_i^{(1)})$ from $(Y^{(1)}, Z^{(1)})$
6: calculate loss
7: update weights
8: **end for**

Approach 2:

1: ~~initialize training data (Y, Z)~~
2: ~~split (Y, Z) into a training set $(Y^{(1)}, Z^{(1)})$ and validation set $(Y^{(2)}, Z^{(2)})$~~
3: initialize neural network
4: **for** $i = 1, \ldots, K$ **do**
5: randomly sample inputs $y_i^{(1)}$ and compute outputs $z_i^{(1)}$
6: calculate_loss
7: update_weights
8: **end for**

Approach 3:

1: ~~initialize training data (Y, Z)~~
2: ~~split (Y, Z) into a training set $(Y^{(1)}, Z^{(1)})$ and validation set $(Y^{(2)}, Z^{(2)})$~~
3: initialize neural network
4: **for** $i = 1, \ldots, K$ **do**
5: randomly sample inputs $y_i^{(1)}$ and compute sample paths $z_i^{(1)}$
6: calculate_loss
7: update_weights
8: **end for**

Example: Approach 2 vs. Approach 3

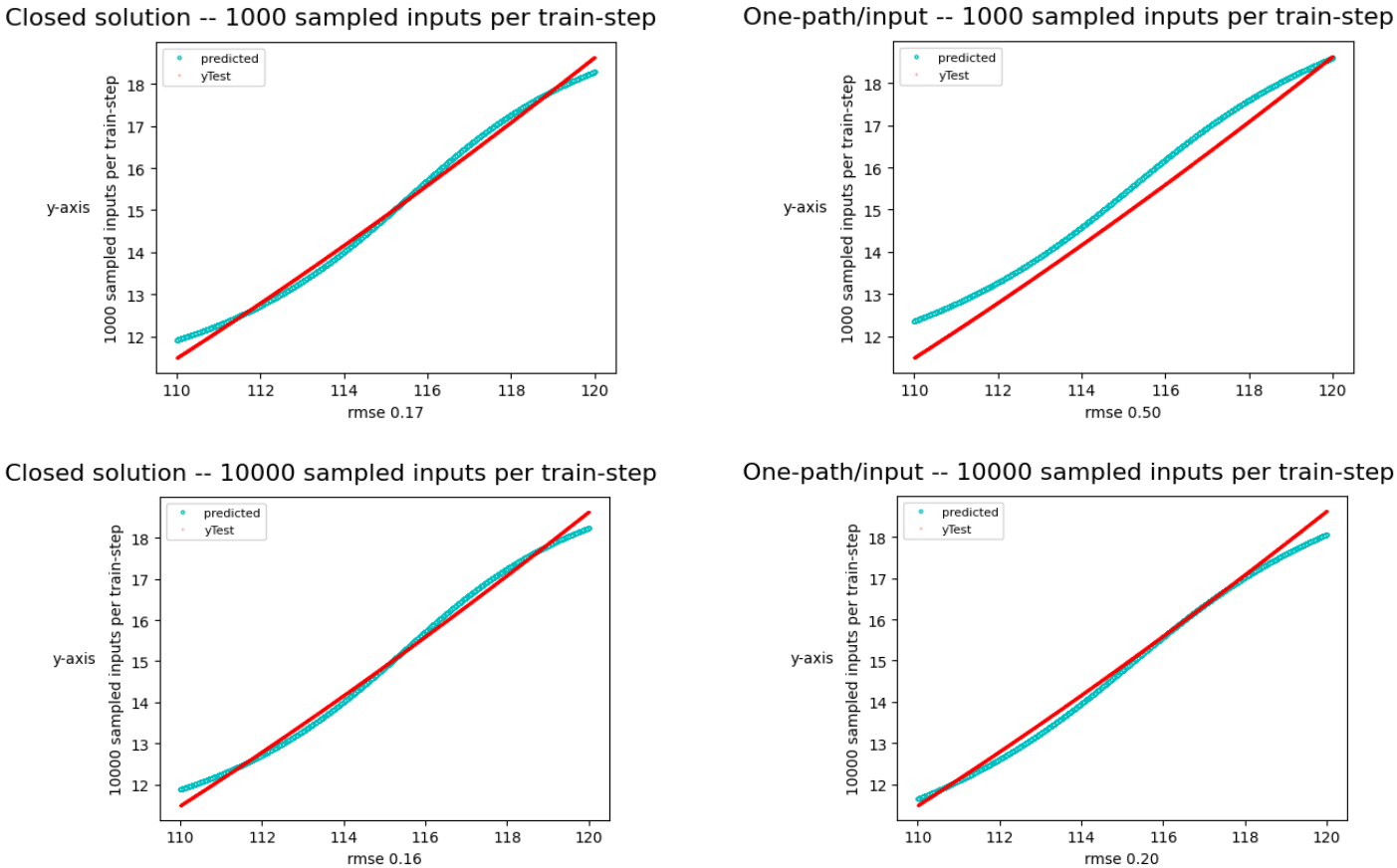

Figure: First line 1000 samples per training step. Second line 10000 samples per step. Fixed neural network architecture and fixed amount of training steps used for all.

Statistical error:

Under certain assumptions (Beck et al.), there exists a a neural network $N : [0,1]^d \to \mathbb{R}$ and a unique continuous function f such that

$$\inf_{f \in \mathcal{C}([0,1]^d, \mathbb{R})} \int_{[0,1]^{d+H}} \left(P^h(u) - f(u)\right)^2 \mathrm{d}u = \int_{[0,1]^{d+H}} \left(P^h(u) - N(u)\right)^2 \mathrm{d}u$$

and it holds for every $u \in [0,1]^d$ that

$$N(u) = \int_{[0,1]^H} P^h(u) \, \mathrm{d}u = \mathbb{E}\left[P^h(u)\right].$$

Using approx. outputs usually requires more input-samples per training step.

Example: observed convergence rates: batch size and training steps

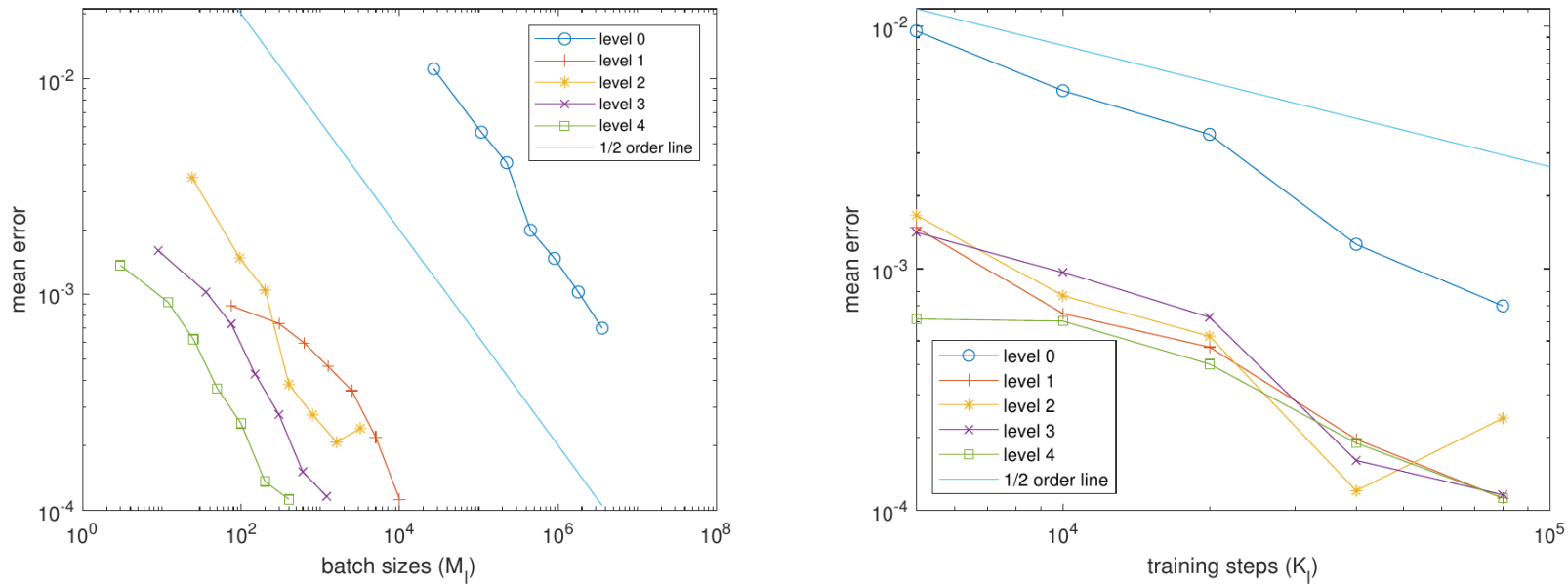

Figure: For levels $l = 0, \ldots, 4$: fixed input evaluation. For comparibility step rate $=$ trainings steps $/5$. Samples(level)/train-step: $(3600000, 10000, 3200, 1200, 400)$

Complexity studies I

Sequence of functions:

$$\mathcal{N}_{\nu,\theta_i,\tilde{P},\mathbb{L}_M} : Y \to \mathbb{R},$$

ν :	network structure (layers and neurons)
$\theta_i \in \mathbb{R}^\nu$:	weights, $i = 1, \ldots, K$ (training steps)
$\tilde{P}$:	approach to generate training data
$\mathbb{L}_M$:	loss function
M :	number of samples used to evaluate the loss function (batch size)

Goal:

Minimize the mean-squared error (MSE)

$$\left\| \mathbb{E}\left[\left(\mathbb{E}[P] - \mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}_M} \right)^2 \right] \right\|_{L^p(Y)} < \epsilon^2,$$

Complexity studies II

Error sources:

$$\mathbb{E}[P - \mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}_M}]$$

$$\begin{aligned}
&= \mathbb{E}[P - \widetilde{P}] && \text{discretization error}\\
&+ \mathbb{E}[\widetilde{P} - \mathcal{N}_{\nu,\Theta,\mathbb{E}[\widetilde{P}],\mathbb{L}}] && \text{approximation error}\\
&+ \mathbb{E}[\mathcal{N}_{\nu,\Theta,\mathbb{E}[\widetilde{P}],\mathbb{L}} - \mathcal{N}_{\nu,\Theta,\widetilde{P},\mathbb{L}}] && \text{statistical error}\\
&+ \mathbb{E}[\mathcal{N}_{\nu,\Theta,\widetilde{P},\mathbb{L}} - \mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}}] && \text{optimization error}\\
&+ \mathbb{E}[\mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}} - \mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}_M}]. && \text{generalization error}
\end{aligned}$$

Computational cost:

$$C_{\mathcal{N}} \leq c\,(K \cdot \text{cost}_{\text{training step}})$$

Bring things together

Conjectures:

For $\mathbb{E}[\dots]$	to achieve the following error bounds:		
$P - \widetilde{P}$	ϵ	ϵ	$\epsilon + h$
$\widetilde{P} - \mathcal{N}_{\nu,\Theta,\mathbb{E}[\widetilde{P}],\mathbb{L}}$	ϵ	ϵ	ϵ
$\mathcal{N}_{\nu,\Theta,\mathbb{E}[\widetilde{P}],\mathbb{L}} - \mathcal{N}_{\nu,\Theta,\widetilde{P},\mathbb{L}}$	0	0	0
$\mathcal{N}_{\nu,\Theta,\widetilde{P},\mathbb{L}} - \mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}}$	$K^{-0.5}$	$K^{-0.5}$	$K^{-0.5}$
$\mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}} - \mathcal{N}_{\nu,\theta_K,\widetilde{P},\mathbb{L}_M}$	M^{-1}	$M^{-0.5}$	$M^{-0.5}$
	Approach 1 (det./MLMC)	Approach 2 (rnd./MLMC)	Approach 3 (rnd./1 path)
requires a complexity $\mathcal{C}_{\mathcal{N}}$ of	$K\epsilon^{-2}M^d$	$K\epsilon^{-2}M$	$Kh^{-1}M$

Complexity studies IV

Lemma (shortened, considering the conjectures)

Then, there exists a values M and K for which the mean squared error can be bounded by ϵ^2 with a computational complexity $C_{\mathcal{N}}$ with bound

$$C_{\mathcal{N}} \leq \begin{cases} \epsilon^{-5.0}, & \text{for "Approach 1 (1-dim)"}, \\ \epsilon^{-4.0-d}, & \text{for "Approach 1"}, \\ \epsilon^{-6.0}, & \text{for "Approach 2"}, \\ \epsilon^{-5.0}, & \text{for "Approach 3"}. \end{cases}$$

Multilevel Monte Carlo learning

- The multilevel estimator $\hat{Y}$ is given by the sum of the *level estimators*:

$$\hat{Y} = \sum_{l=0}^{L} \hat{Y}_l.$$

- Level estimators:

$$\hat{Y}_l : y \mapsto \begin{cases} P_{h_0}(y) & \text{for } l = 0, \\ P_{h_l}(y) - P_{h_{l-1}}(y)), & \text{for } l > 0. \end{cases}$$

- Train a network $\mathcal{N}_l$ for each level estimator to obtain the estimator:

$$\hat{\mathcal{N}} := \sum_{l=0}^{L} \mathcal{N}_l,$$

Multilevel Monte Carlo learning

Theorem (shortened, only selected assumption)

If amongst others

$$\left\| \mathbb{V}\left[\hat{\mathcal{N}}_{\nu,\theta^l_{K_l},\hat{Y}_l,\mathbb{L}_{M_l}} \right] \right\|_{L^p(Y)} \leq c_2 (h_l^2 K_l^{-1} + h_l M_l^{-1}),$$

$$C_l \leq c_3 h_l^{-1} M_l K_l.$$

Then, the MSE can be bounded with a computational complexity of

$$C_{\hat{\mathcal{N}}} \leq \epsilon^{-4.0},$$

Reminder Singlelevel:

$$C_{\mathcal{N}} \leq \begin{cases} \epsilon^{-5.0}, & \text{for "Approach 1 (1-dim)"} , \\ \epsilon^{-4.0-d}, & \text{for "Approach 1"} , \\ \epsilon^{-6.0}, & \text{for "Approach 2"} , \\ \epsilon^{-5.0}, & \text{for "Approach 3"} . \end{cases}$$

Training example:

$$Y = [0.02, 0.05] \times [0.1, 0.2] \times [80, 120] \times [109, 110] \times [0.9, 1.0]$$
$$y = (\mu, \sigma, K, s_0, T) \in Y.$$

Parameter	Value
neurons	$(50, 50, 1)$
decay rate	0.1
initial learning rate	0.01
step rate	40.000
training steps	150.000

Table: Network structure and training parameters.

Comparing approach 3 and the new multilevel idea

Setting:

- repeated 10 times
- Nvidia K80 GPU
- Max-absolute-error is estimated using 20.000.000 randomly sampled inputs and the closed solution formula
- Batch sizes are calculated through a transformation of the multilevel sample size estimator

Comparison:

single-level		multilevel	
mean error	time	mean error	time
0.0273	2.32h	0.0290	4.15h
0.0182	7h	0.0184	5.29h
0.0119	26.66h	0.0103	11.18h

Further works may include:

- Algorithm for individual training parameter detection
- Individual network structure for each network
- Complexity studies including the network structure
- Conjecture revision
- Differential machine learning

- https://github.com/da-roth/NeuronalNetworkTensorflowFramework
- T. Gerstner, B. Harrach, D. Roth, M. Simon *Multilevel Monte Carlo learning*, arXiv

---

Thank you for your attention!

Batch sizes per net:

Estimated needed Monte Carlo samples N_l for the Multilevel Monte Carlo approach for a fixed y and $\epsilon = 0.01$:

level l	0	1	2	3	4	5	6	7
N_l	3.000.000	72695	27756	10550	3691	1308	476	182

Estimated needed batch-size for the training of the specific level nets for $l = 0, \ldots, 7$:

multilevel id	level l	0	1	2	3	4	5	6	7
1	M_l	75.000	1817	690	264	93	33	12	5
2	M_l	300.000	7268	2760	1056	372	132	48	20
3	M_l	1.200.000	29072	11040	4224	1488	528	192	80

```
###
### 1. Training data Generator/Importer
###

Generator = GBM(GBM_Case.Standard)
Generator.set_inputName('S')
Generator.set_outputName('10000 sampled inputs per train-step')

###
### 2. Set Nueral network structure / Hyperparameters
###

Regressor = Neural_Approximator()
Regressor.set_Generator(Generator)
Regressor.set_hiddenNeurons(20)
Regressor.set_hiddenLayers(2)
Regressor.set_activationFunctionsHidden(tf.nn.sigmoid)
Regressor.set_activationFunctionOutput(tf.nn.sigmoid)
Regressor.set_weight_seed(1)

#hiddenNeurons = 20               # we use equal neurons for each hidden layer
#hiddenLayers = 3                # amount of hidden layers
#activationFunctionsHidden = [tf.nn.tanh]   # activation functions of hidden layers


###
### 3. Set Nueral network structure / Hyperparameters
###

TrainSettings = TrainingSettings()
TrainSettings.set_learning_rate_schedule([(0.0, 0.001),(0.333, 0.0005),(0.666, 0.00025)])
TrainSettings.set_nTest(20000)
TrainSettings.set_samplesPerStep(10000)
TrainSettings.set_trainingSteps(2000)


###
### 3. Train network and Study results
### Comment: For different trainingSetSizes the neural network reset and not saved, hence t
###
xTest, yTest, yPredicted = train_and_test(Generator, Regressor, TrainSettings)
plot_results("One-path/input", yPredicted, xTest, yTest, Generator)
```