CCIMI & Morgan Stanley Collaborative Project Presentation

Neural network approximation to the SABR option pricing model

Parley Ruogu Yang & Russell Barker

22 March 2021

1. Introduction
2. Simulation and computational costs
3. Neural Network
4. Results
5. Evaluations and Extensions

Key reference: Horvath, Blanka, Aitor Muguruza, and Mehdi Tomas (2019). *Deep Learning Volatility*, arXiv:1901.09647v2

The set-up from Horvath et al (2019)

- Model $M(\theta)$ with parameter $\theta \in \Theta$
- Approximate calibration between a numerical approximation $\tilde{P}$ of the pricing map, and price data observed from the market P^{MKT}:

$$\hat{\theta} = \underset{\theta \in \Theta}{\arg\min}\, \delta(\tilde{P}^{M(\theta)}(\zeta), P^{MKT}(\zeta)) \tag{1}$$

  where ζ denotes the choice of exotic product attribute.
- We learn $F^*(\theta) = \{\tilde{P}^{M(\theta)}(\zeta_j)\}_{j=1}^{N}$ via NN (step 1), then proceed into an inverse mapping study $P^{MKT} \mapsto \hat{\theta}$ (step 2).
- Here we investigate on the step 1.

SABR model

The Fundamental Pricing Theorem (FPT) with a constant interest rate r is specified as

$$\mathbb{E}\left[\max\{P^{asset}(T;\theta)-K,0\}e^{-rT}\right] = P^{call}(K;P^{asset}) =: F(K;\theta) \tag{2}$$

In the SABR model, $\theta \in \mathbb{R}^5$, specified as $\theta = (P^{asset}(0), \alpha(0), \beta, \rho, v)$

$$dP^{asset} = \alpha {P^{asset}}^{\beta} dW_1 \tag{3a}$$

$$d\alpha = v\alpha dW_2 \tag{3b}$$

$$dW_1 dW_2 = \rho dt \tag{3c}$$

Without further ambiguity, we denote P as the asset price P^{asset}.

SABR model — parameters

We fix certain probability distribution on $\mathbb{R}^5$, in particular,

$$P(0) \sim U[0, 200] \tag{4a}$$
$$\alpha(0) \sim U[0, 1] \tag{4b}$$
$$\beta \sim U[0, 1] \tag{4c}$$
$$\rho \sim U[-1, 1] \tag{4d}$$
$$v \sim U[0, 1] \tag{4e}$$

Motivation for computational analysis

- In practice, we face a large dimension of Θ [1]
- Consider $\theta = (\theta_0, \theta_1)$ where θ_1 is a fixed calibration. Then the ultimate aim becomes learning $P^{MKT} \mapsto (\hat{\theta}_0(\theta_1), \theta_1)$
- We note that θ_1 needs to be updated rather regularly — the contemporary choice of the calibrated θ_1 depends on the historical market data and managerial decisions, which can change over the time.

Therefore, the learning needs to be done repetitively, as for every θ_1, the training of neural network needs to proceed thoroughly.

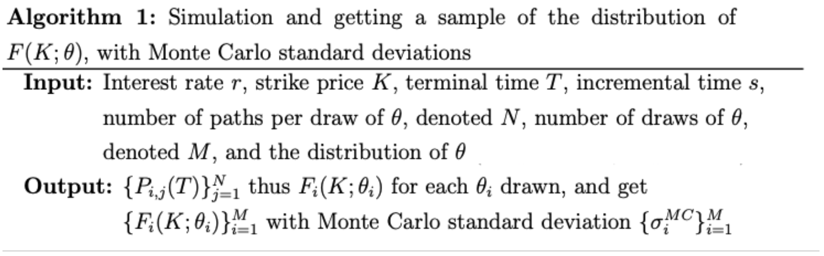
[1] E.g. HJM for FX forwards.

Contribution

- Deepen the understanding of the approximation behaviour towards SABR pricing model
- Computationally observe the trade-off amongst Monte Carlo sample size (M), Price paths per sample (N) and step size (s) given computing constraints.
- Further the observations on NN approximations[2] and potential failures.

[2] Harry and Valeria would have more to talk about on this part.

Algorithm



Algorithm 1: Simulation and getting a sample of the distribution of $F(K;\theta)$, with Monte Carlo standard deviations

Input: Interest rate r, strike price K, terminal time T, incremental time s, number of paths per draw of θ, denoted N, number of draws of θ, denoted M, and the distribution of θ

Output: $\{P_{i,j}(T)\}_{j=1}^{N}$ thus $F_i(K;\theta_i)$ for each θ_i drawn, and get $\{F_i(K;\theta_i)\}_{i=1}^{M}$ with Monte Carlo standard deviation $\{\sigma_i^{MC}\}_{i=1}^{M}$

```
1. For $i \in \{1, ..., M\}$:
    (a) Sample $\theta_i$
    (b) Repeat below for $N$ times to get $\{P_{i,j}(T)\}_{j=1}^{N}$:
        i. Sample Brownian Motion $W_1(t), W_2(t)$:
            A. Sample $\Delta W_2(t) \sim N(0, s)$
            B. Sample $\Delta W_1(t)$
        ii. Compute $\alpha(t), P(t)$:
            A. Compute $\alpha(t)$
            B. Compute $\Delta P(t) = \alpha(t) P(t)^{\beta} \Delta W_1(t)$
    (c) Obtain $F_i(K; \theta_i), \sigma_i^{MC}$
        i. Compute $P_{i,j}^{call}(K; P_{i,j}) := \max\{P_{i,j}(T; \theta_i) - K, 0\} e^{-rT}$ for all $j$
        ii. Obtain $F_i(K; \theta_i)$ as the mean of $\{P_{i,j}^{call}(K; P_{i,j})\}_{j=1}^{N}$
        iii. Obtain $\sigma_i^{MC}$ as the standard deviation of $\{P_{i,j}^{call}(K; P_{i,j})\}_{j=1}^{N}$
        iv. Reject the sample and re-run (i.e. return to 1(a) and keeping the
            same index $i$) if $F_i(K; \theta_i) > 400$
2. Therefore obtain $\{F_i(K; \theta_i), \sigma_i^{MC}\}_{i=1}^{M}$
```

Remarks and computational costs

- $F_i(K;\theta_i)$ is an approximation to the true $F(K;\theta_i)$
- With some pre-determined $\{K_j\}_{j=1}^{J}$, usually as a function of $F(0)$, we simulate a dataset $\{(F(K_j;\theta_i))_{j=1}^{J},\theta_i\}_{i=1}^{M}$. The cost of creating this dataset is $O(NM(J+s^{-1}))$.
    - The draw of normal distributions cost $O(NMs^{-1})$. This comes from the sampling of Brownian motions.
    - The computation with J strike prices needs $O(NMJ)$. This comes from standard statistical operations for mean and variance.

Specifications in this experiment

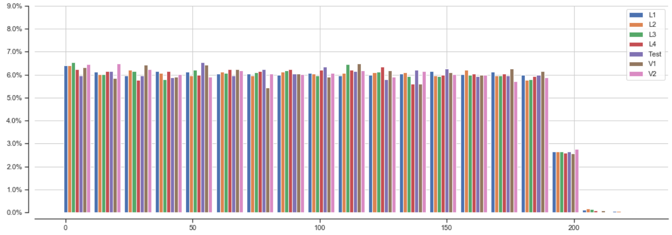
Here we simply fix J and the specification of K_j: we let $K_1, ..., K_8$ to be $0.7F(0), ..., 1.4F(0)$, respectively.

Choices of combinations for training and validation set:

N	M	J	s^{-1}	Name
50	12K	8	100	V1
500	12K	8	20	V2
50	100K	8	10	L1
25	100K	8	20	L2
50	50K	8	20	L3
100	25K	8	20	L4

Configuration for the fixed test set, which will be used for final evaluation:

N	M	J	s^{-1}	Name
500	10K	8	100	TestData



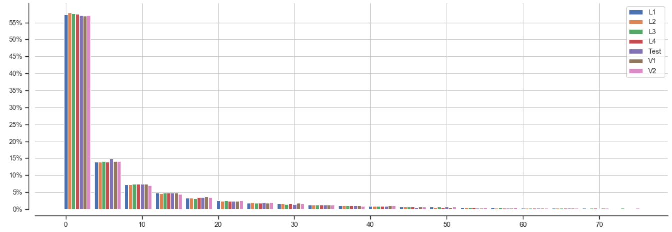


Figure: Histogram of $F_i(K;\theta_i)$ (top) and σ_i^{MC} (bottom)

Aim

Big picture:

$$\phi : \Theta \to \mathbb{R}^J \text{ s.t. } \phi(\cdot) \approx F(K; \cdot)$$

In practice:
fix a test dataset D^{test} and with a evaluation formula $l(D, \phi)$,

$$\min l(D^{\text{test}}, \phi)$$

Questions:

- How to determine the architecture?
- Given an architecture, how to train the neural network?

Architecture

Recall that a component-wise Exponential Linear Units (ELU) stands for

$$ELU(x) = \mathbb{1}[x > 0]x + \mathbb{1}[x < 0](e^x - 1)$$

and that a component-wise Rectified ELU (RELU) stands for

$$RELU(x) = \max\{0, x\}$$

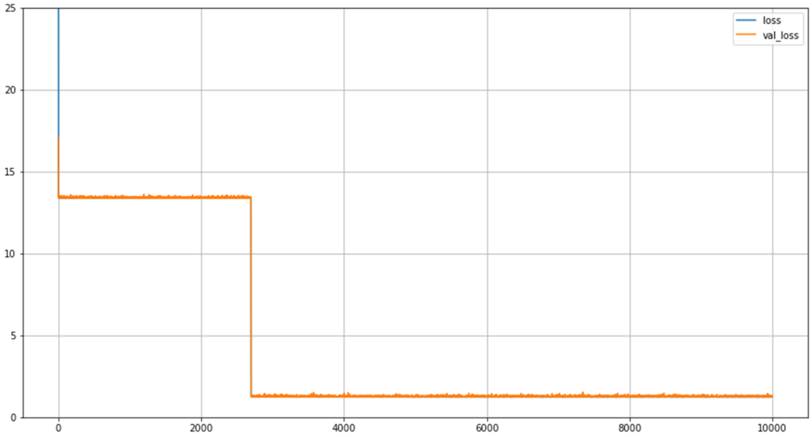
We consider

$$\phi = \sigma_3 \circ W_3 \circ \sigma_2 \circ W_2 \circ \sigma_1 \circ W_1$$

where $\sigma_1 = \sigma_2 = ELU$ and $\sigma_3 \in \{ELU, RELU\}$ and affine maps $W_l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$. We thus have neuron vector $n = (n_0, n_1, n_2, n_3)$ with $n_0 = \dim(\Theta)$ and $n_3 = J$

Training & Empirical troubles

Training: ADAM with MAE loss. Weight initialisation: the trouble.



Validation — Evaluation function

Consider a function that outputs the percentage of predictions errors that are larger than $k\sigma^{MC}$, where $k \in \{1, 2, 3\}$. This can be mathematically written as:

$$\ell_k^{MC}(\sigma, D, \phi) = 100(|D|J)^{-1} \sum_{(x,y,\sigma^{MC})\in D} \sum_{j=1}^{J} \mathbb{1}[|y_j - (\phi(x))_j| > k\sigma] \quad (5)$$

The choice of σ varies — here we consider the σ_L, σ_S, noting the maximum and minimum, respectively, of the generated σ^{MC}

Validation — Evaluation procedure

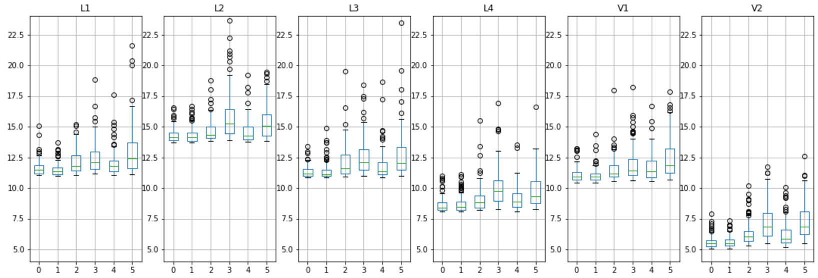
1. For each ϕ_z, for each randomisation $i \in \{1, 2, ..., 99, 100\}$,[3] we obtain $I(D; z, i) = \ell_2^{MC}(\sigma_L, D, \phi_z)$.
2. Summarise them by obtaining $I^{median}(D; z)$ as the median of $\{I(D; z, i)\}_{i=1}^{100}$.
3. Select the top-performing model of each dataset, that is, $z^*(D) = \arg\min_{z \in Z}\{I^{median}(D; z)\}$

[3] This can be completed by the `setseed` in python.

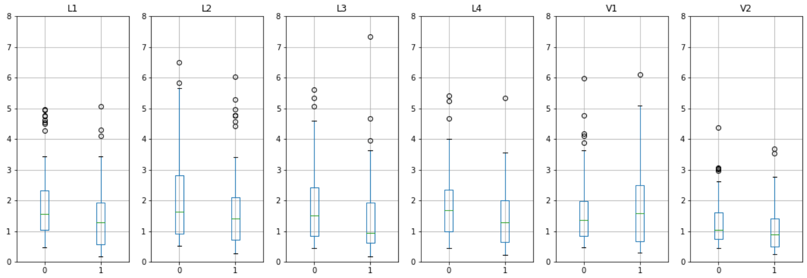
Practical set-ups of the architecture

Label	n	Total number of parameters	σ_3
0	(5,6,7,8)	149	ELU
1	(5,6,7,8)	149	RELU
2	(5,10,40,8)	828	ELU
3	(5,10,40,8)	828	RELU
4	(5,40,40,8)	2208	ELU
5	(5,40,40,8)	2208	RELU

Validation results



Test results



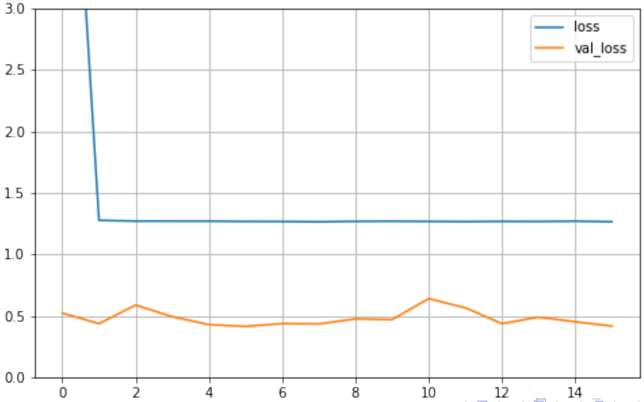
From above, we see more stable output in V2 compared to V1 — this implies that, from the simulation, an increased path size (N) at a cost of increased step size (s) could be beneficial. The rests have no contribution to a determined conclusion.

Interesting observations

MC-NN Error decomposition:

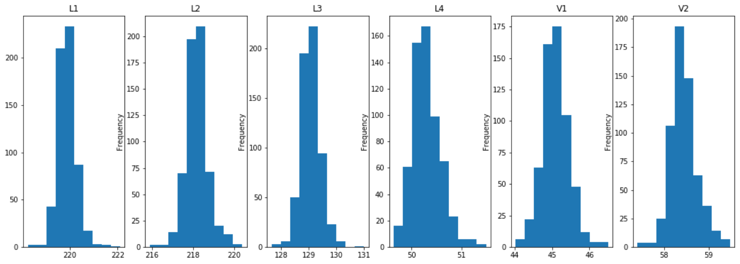
$$F_i(K;\theta_i) - \phi(K;\theta_i) = (F_i(K;\theta_i) - F(K;\theta_i)) + (F(K;\theta_i) - \phi(K;\theta_i))$$

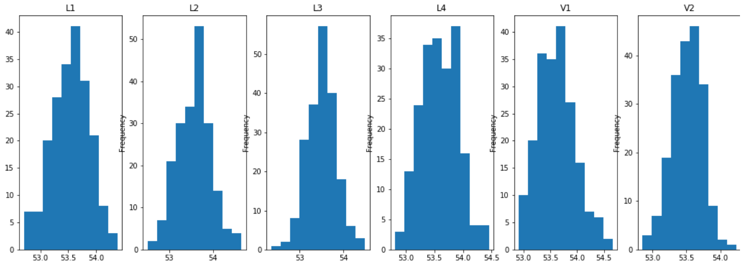
It seems that the MC-Truth error occupies heavily in some dataset, as during the testing stage, the within-data loss could be higher than the test-data loss.



Interesting observations & Future extensions

Irreducible error on the max error — similar to Dr Hansen's arguments.





Future extensions

1. One may try approximating the true option prices via polynomial expansions.
2. Surprised about how fragile and delicate the training of NN can be. Potential extension is to back check on earlystop and potentially other training methods
3. The simulated dataset can be fructified and diversified with put options and grid sample of θ
4. It is a headache doing experiments with Colab / local machines as the computing power was so limited.