Deep Learning in Finance

option pricing

Damien Challet
damien.challet@centralesupelec.fr

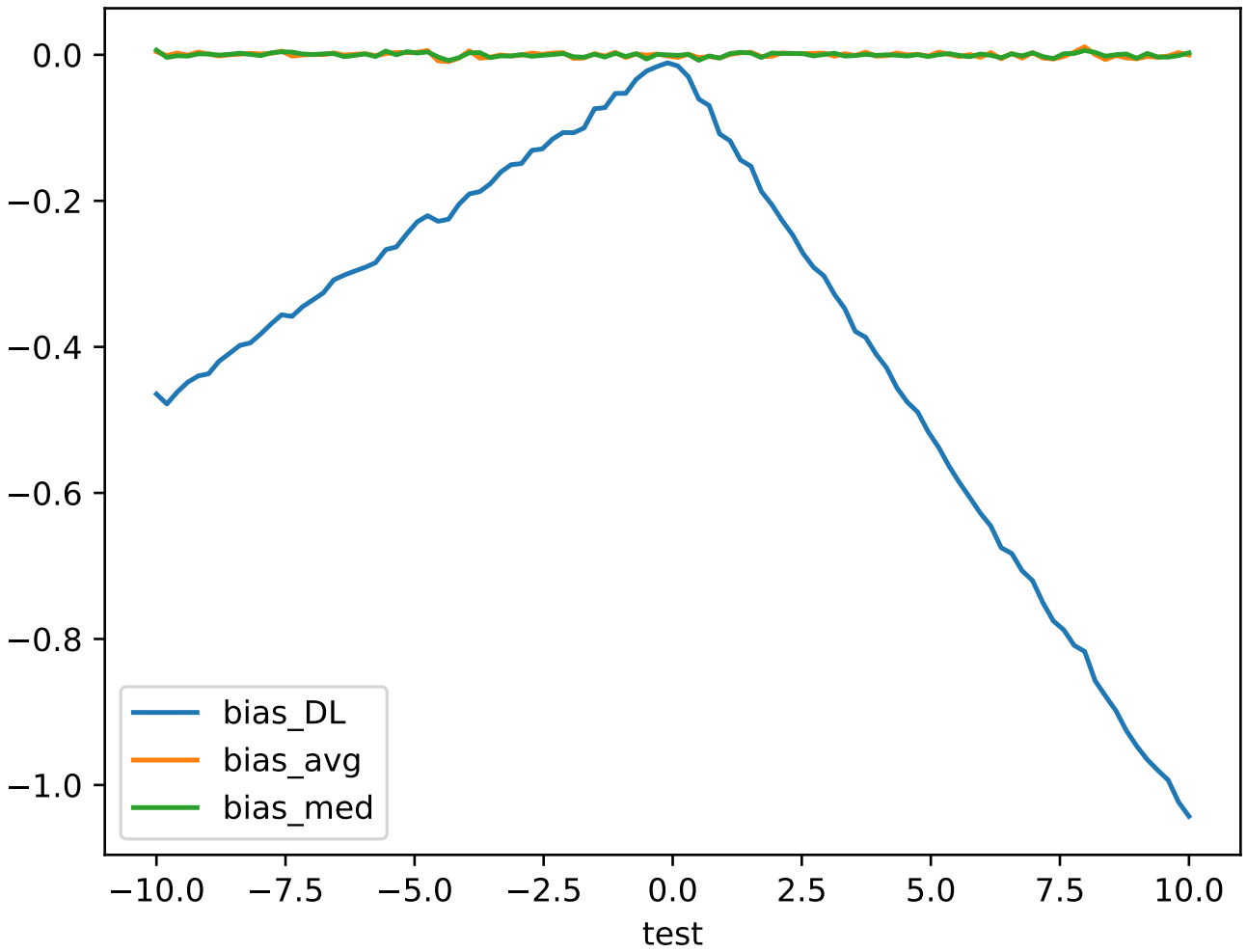
29th October 2024

Predictors: NNs vs trees

1. Tree-based methods cannot add, multiply, compare and lag columns
    - Additional predictors must be added
    - NNs can do that (learning to multiply is harder)
2. Tree-based methods cannot extrapolate beyond previously seen values
    - NNs can do that (not very well)

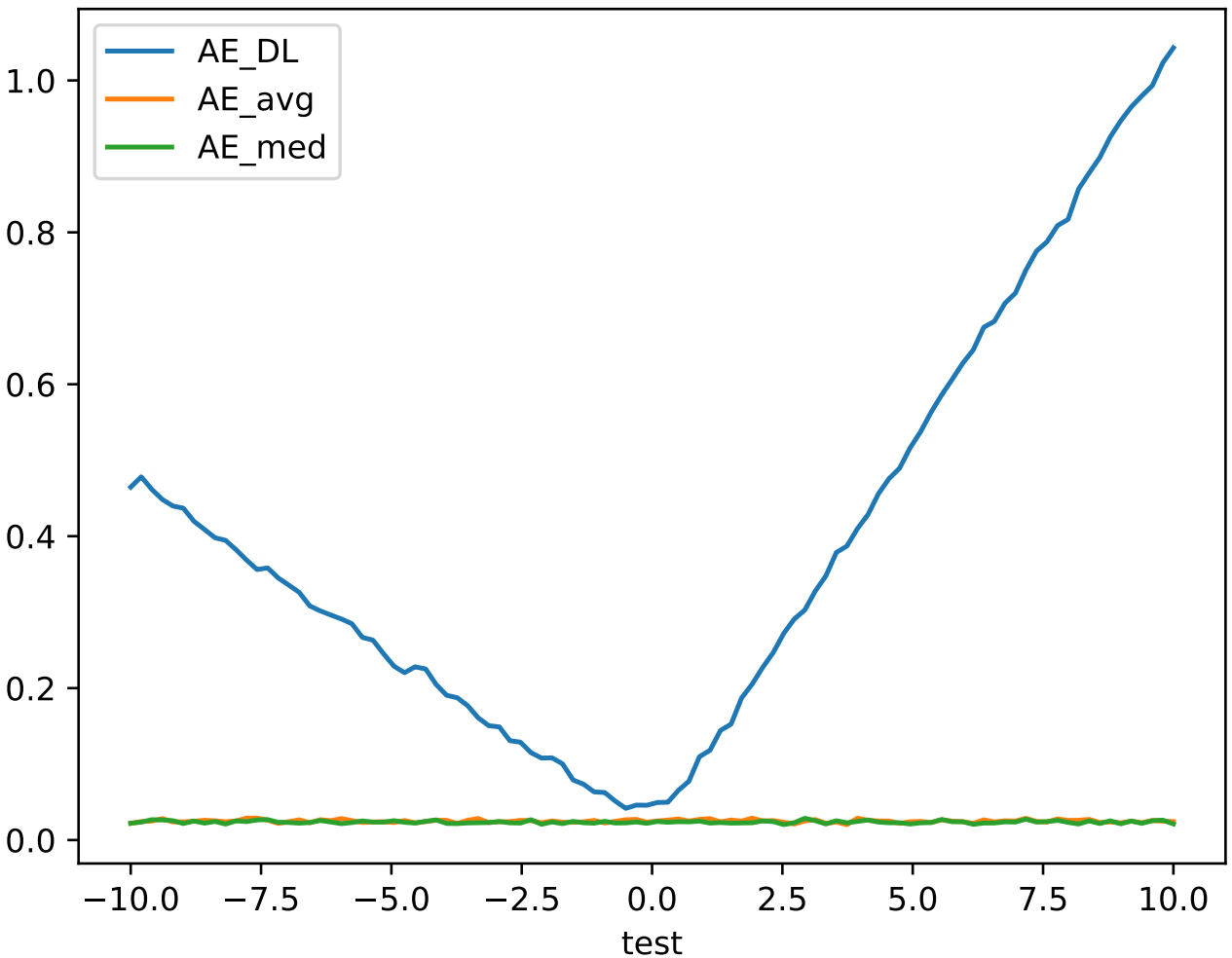
DL and sample mean: bias

DL: train for $\mu \in [-2, +2]$



DL and sample mean: variance

Mean absolute error: mainly from bias



Hard problems in Finance

1. Modelling and calibration
2. Solving high-dimensional equations
3. Prediction
4. Portfolio construction

Options

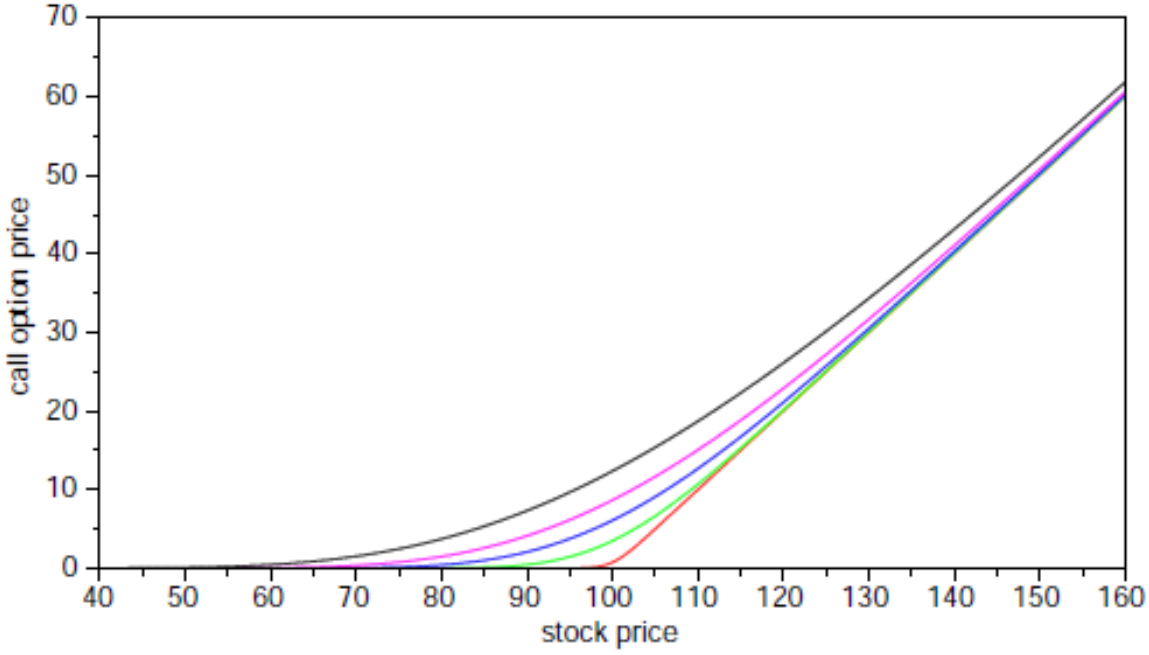
Insurance / bet: price/volatility change

option $\equiv$ right to buy (*call*) or to sell (*put*) an asset at price K (strike).

- European option: single exercise time T (maturity)
- American option: any time until expiry
- barrier options
- exotic options

Payoff shape

Strike $K = 100$



from [link]

Options variables

- asset price p_t
- strike price K
- time to maturity T
- interest rate r

$\rightarrow$ option price at time $t = 0$?

European option price

- Hyp: call option, strike K, time to maturity T
- Option price C = fair price
- Naively,

$$C = E[\max(p_T - K, 0)]$$

- Interest rate?

  C can be invested in a risk-free asset

$$C(1+r)^T = E[\max(p_T - K, 0)]$$
$$= \int_K^\infty P(p_T)(p_T - K)dp_T$$

$\rightarrow$ model of price needed

Bachelier's model

- Price evolution

$$p_{t+1} = p_t + \eta_t$$
$$\eta_t \sim \mathcal{N}(0, \sigma_B^2)$$

- Normal diffusion

$$P(p_T|p_0) = \frac{1}{\sqrt{2\pi T}\sigma_B} e^{-\frac{(p_T - p_0)^2}{2T\sigma_B^2}}$$

- Additive world

Black-Scholes model

- Price evolution

$$\log p_{t+1} = \log p_t + \eta_t$$
$$\eta_t \sim \mathcal{N}(0, \sigma^2_{BS})$$

- Log-normal diffusion

$$P(\log p_T | \log p_0) = \frac{1}{\sqrt{2\pi T}\sigma} e^{-\frac{(\log p_T - \log p_0)^2}{2\sigma T}}$$

- N.B.: no jumps, constant volatility

Fair price in discrete time with hedging

- $t \in \{0, \cdots, T\}$
- v_t: number of shares invested in underlying asset

$$C(1+r)^T = E[\max(K - p_T, 0)] - \sum_{t=0}^{t-1} E\left[(1+r)^{T-k-1} v_t (p_{t+1} - p_t - r p_t)\right]$$

- v_t: hedging (reduce or eliminate risk)

Optimal hedging

- Capital evolution of option underwriter:

$$\Delta W = C(1+r)^T - [\max(K - p_T, 0)] + \sum_{t=0}^{t-1} \left[(1+r)^{T-k-1} v_t (p_{t+1} - p_t - r p_t) \right]$$

- Residual risk $= \min_{\{v_t\}} \mathrm{var}(\Delta W)$

- Continuous time & Gaussian increments

  $\rightarrow$ Black-Scholes miracle:

  zero residual risk with optimal hedging

Black-Scholes option price

Continuous-time limit: $T = N\tau$, $\tau, 1/N \to 0$,

call price

$$C_{BS}(p_0, K, T, r, \sigma) = p_0 P_> [y_-] - K e^{-rT} P_> [y_+]$$

$$P_>(x) = \frac{1 + \mathrm{erf}(x/\sqrt{2})}{2}$$

$$y_\pm = \frac{\log(p_0/K \cdot e^{-rT}) \pm \sigma^2 T/2}{\sigma\sqrt{T}}$$

Price scaling

$$C_{BS}(p_0, K, T, r, \sigma) = p_0 P_> [y_-] - K e^{-rT} P_> [y_+]$$

$$P_>(x) = \frac{1 + \mathrm{erf}(x/\sqrt{2})}{2}$$

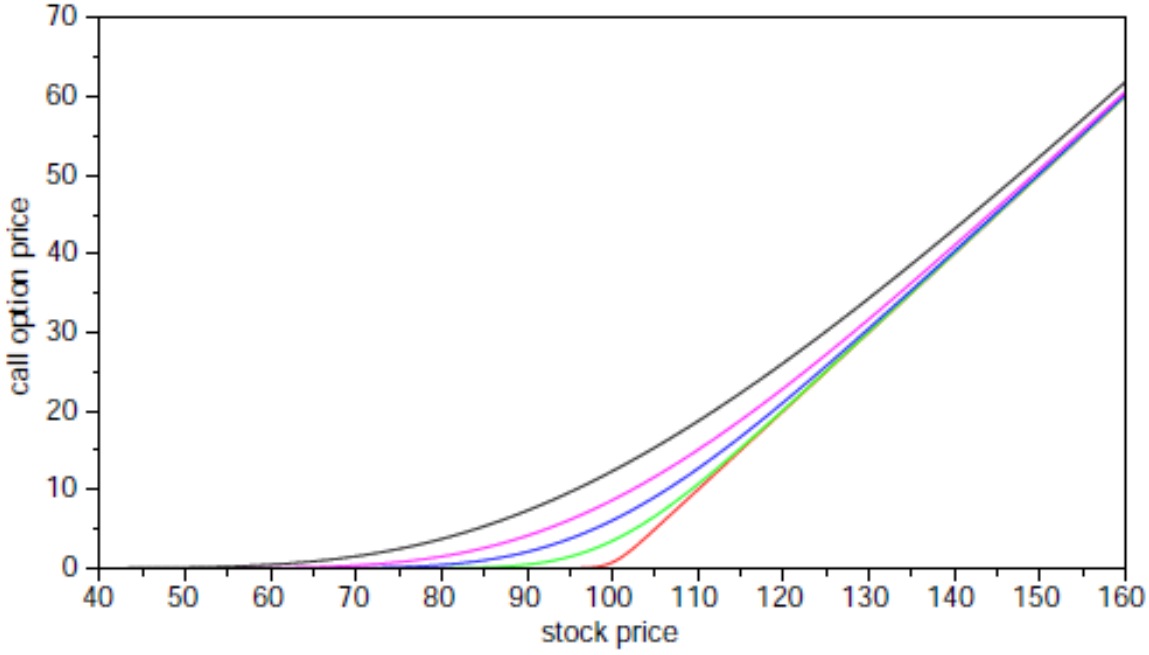
$$y_\pm = \frac{\log(p_0/K e^{-rT}) \pm \sigma^2 T/2}{\sigma\sqrt{T}}$$

Setting $k = \frac{K e^{-rT}}{p_0}$

$$\frac{C_{BS}(k, T, r, \sigma)}{p_0} = P_> [y_-] \quad - k P_> [y_+]$$

Payoff shape

Strike $K = 100$



from [link]

Black-Scholes option price

Continuous-time limit: $T = N\tau$, $\tau, 1/N \to 0$,

Put price

$$P_{BS}(p_0, K, T, r, \sigma) = -p_0 P_{>}[-y_{-}] + Ke^{-rT} P_{>}[-y_{+}]$$

$$P_{>}(x) = \frac{1 + \mathrm{erf}(x/\sqrt{2})}{2}$$

$$y_{\pm} = \frac{\log(p_0/K \cdot e^{-rT}) \pm \sigma^2 T/2}{\sigma\sqrt{T}}$$

Parameters of Black and Scholes

- Known: p_0, K, T
- Possibly known: r
- Optimised: hedging $\{v_t\}$
- **Estimated**: σ
- **Assumed:** continuous time, Gaussian returns

Black-Scholes vs reality: returns

Heavy-tailed skewed price returns+stochastic volatility

$\rightarrow$ implicit volatility:

$$\underbrace{C_{BS}(p_0, K, T, r, \sigma_{impl})}_{BS \text{ world}} = \underbrace{\text{option price}(p_0, K, T, r)}_{\text{real world}}$$

- Heavy tails $\rightarrow$ smile
- Skewed returns, leverage effect $\rightarrow$ skewed smile

Non-constant volatility

$\rightarrow$ stochastic volatility model

$\rightarrow$ non-explicit option price formulae (e.g. Heston, etc.)

N.B.: additional risk: volatility of volatility

Deep learning and option pricing

1. Model-driven — speed up calibration
2. Data-driven approach — Ito who?
3. Model + Data — improve on 1. and 2.

1. Deep learning option prices from model

1. NN fitted to reproduce expensive model output
   - 1.1 Monte-Carlo
   - 1.2 Heston
   - 1.3 Rough Bergomi [link]

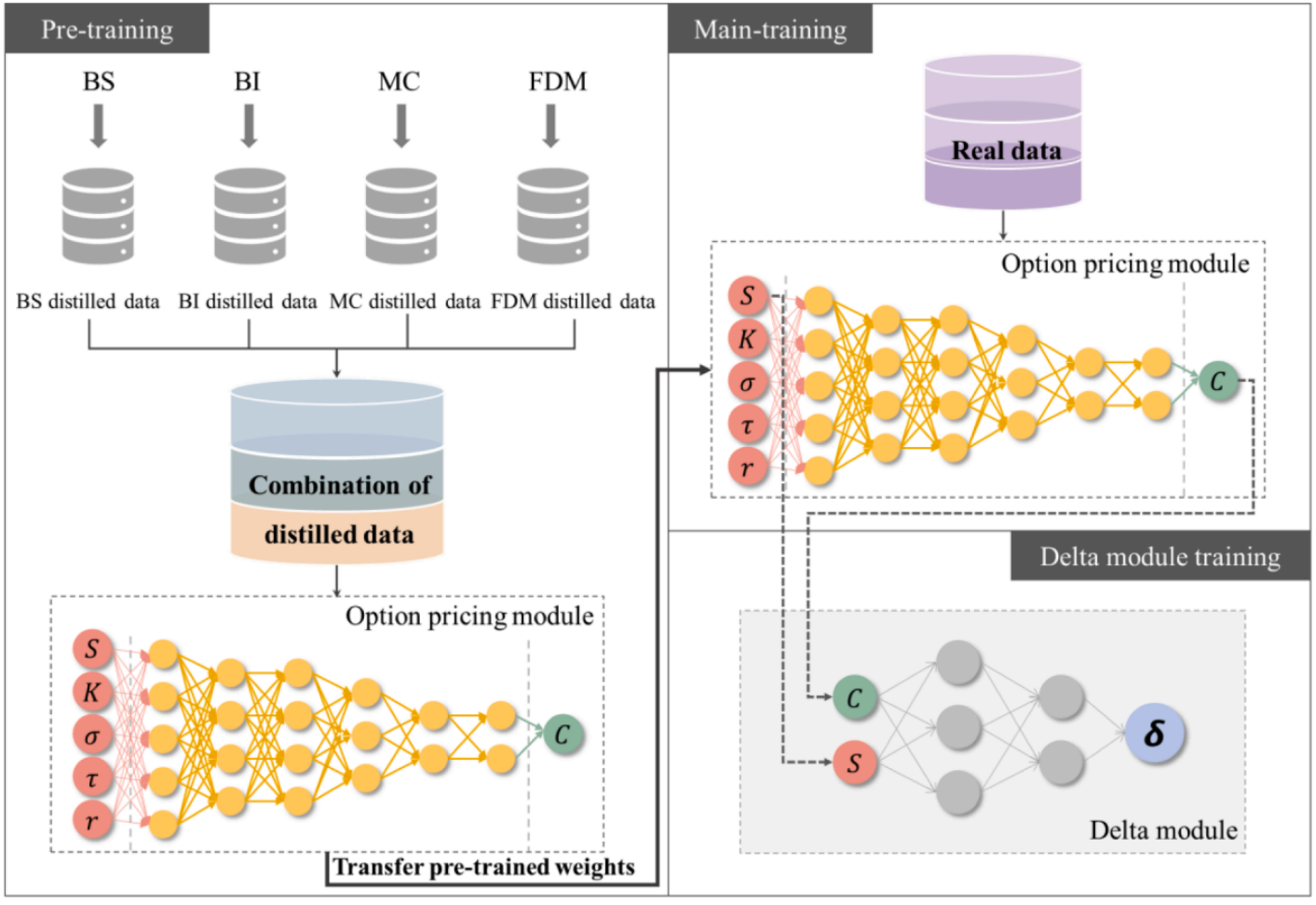
2. Find best parameters of the NN-fitted model
   - 2.1 with standard optimisation
   - 2.2 add no-arbitrage conditions
   - 2.3 NN [link]

2. Data-driven option price

1. Fixed parameters (K, p_0, r, T)
2. Add estimated parameters σ, skew, tail exponent
3. Fit option prices with DL $\rightarrow$ option pricer

3. Deep learning option prices: prices from model and data

Example: Jang *et al.* (2020)
[https://doi.org/10.1016/j.inffus.2020.12.010]



3. Deep learning option prices from model

Improvement:

1. Give what you know as inputs

2. Learn residuals

Example: Funahashi (2021) [link]

- Learn $C_3 - C_{data}$, where $C_3 =$ third order approximation of C_{BS}
  - Smaller training time
  - More robust and stable
  - Faster calculations

Solving complex option pricing equations

Wang et al (2022) [link]

- Physics informed NN (PINNS) (2019) [link] (11300 citations)

To descibe the Physics-Informed Neural Networks for solving PDEs, let us consider a general PDE with the solution $u(\boldsymbol{x})$ defined on a domain $\Omega \subset \mathcal{R}^d$:

$$f\left(\boldsymbol{x}; \frac{\partial u}{\partial x_1}, \cdots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \cdots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \cdots\right) = 0, \quad \boldsymbol{x} := (x_1, \cdots, x_d)' \in \Omega, \tag{2}$$

supplemented with proper boundary conditions

$$\mathcal{B}(u, \boldsymbol{x}) = 0 \quad \text{on} \quad \partial\Omega, \tag{3}$$

where $\mathcal{B}(u, \boldsymbol{x})$ can be Dirichlet, Neumann, Robin, or even periodic boundary con-

Solving complex option pricing equations

Physics informed NN (PINNS) (2019) [link]

The major steps of the PINNs algorithm for solving a PDE are as follows (cf. Lu et al. 2021):

Step 1 Construct a neural network $\hat{u}(\boldsymbol{x};\theta)$ with parameters $\theta = \{W_l, \boldsymbol{b}_l\}_{1 \le l \le L}$.

Step 2 Specify the training data $\mathcal{T}_{pde}$ and $\mathcal{T}_{ib}$ of sizes N_{pde} and N_{ib}, respectively, to satisfy the physics imposed by the PDE and the boundary/initial conditions. Here $\mathcal{T}_{pde} \subset \Omega$ and $\mathcal{T}_{ib} \subset \partial\Omega$ are points inside the domain Ω and on the domain boundary $\partial\Omega$, respectively.

Step 3 Specify a loss function defined as a weighted sum of the L^2 errors for the PDE and the boundary/initial conditions:

$$\begin{aligned}\mathcal{L}(\theta) :=& \omega_{pde} \cdot \frac{1}{N_{pde}} \sum_{\boldsymbol{x} \in \mathcal{T}_{pde}} \left|\left| f\left(\boldsymbol{x}; \frac{\partial \hat{u}}{\partial x_1}, \cdots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \cdots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \cdots\right)\right|\right|^2_{L^2(\Omega)} \\ &+ \omega_{ib} \cdot \frac{1}{N_{ib}} \sum_{\boldsymbol{x} \in \mathcal{T}_{ib}} ||\mathcal{B}(\hat{u}; \boldsymbol{x})||^2_{L^2(\partial\Omega)},\end{aligned} \tag{4}$$

where ω_{pde} and ω_{ib} are the weights. Those derivatives in the loss function (4) are evaluated by the automatic differentiation (AD) provided by Tensorflow (Abadi

Solving complex option pricing equations

Wang et al (2022) [link]

Non-linear BS equation (with transaction costs)

Let us consider the nonlinear Black–Scholes equation (Lesmana et al., 2013):

$$\frac{\partial V}{\partial \tau} - \frac{1}{2}\sigma^2(\tau, S, \frac{\partial V}{\partial S}, \frac{\partial^2 V}{\partial S^2})S^2\frac{\partial^2 V}{\partial S^2} - rS\frac{\partial V}{\partial S} + rV = 0, \text{ on } (0,\infty)\times(0,T], \quad (10)$$

where the volatility σ is a function of time τ, the underlying stock price S, the Delta $\frac{\partial V}{\partial S}$, and the Gamma $\frac{\partial^2 V}{\partial S^2}$.

Two-asset case

$$\frac{\partial V}{\partial \tau} - \frac{\sigma_1^2 S_1^2}{2}\frac{\partial^2 V}{\partial S_1^2} - (r-\delta_1)S_1\frac{\partial V}{\partial S_1} + rV$$
$$-\frac{\sigma_2^2 S_2^2}{2}\frac{\partial^2 V}{\partial S_2^2} - (r-\delta_2)S_2\frac{\partial V}{\partial S_2} - \rho\sigma_1\sigma_2 S_1 S_2\frac{\partial^2 V}{\partial S_1 \partial S_2} = 0, \quad \text{on } (0,\infty)^2\times(0,T], \quad (16)$$

where the option value $V(S_1, S_2, \tau)$ depends on the asset prices S_i $(i = 1, 2)$ and the time to maturity τ. The parameter r is the interest rate, σ_i and δ_i $(i = 1, 2)$ denote the volatilities and the dividend rates for the two assets, respectively, and ρ represents the correlation coeffcient between S_1 and S_2.

Solving complex option pricing equations

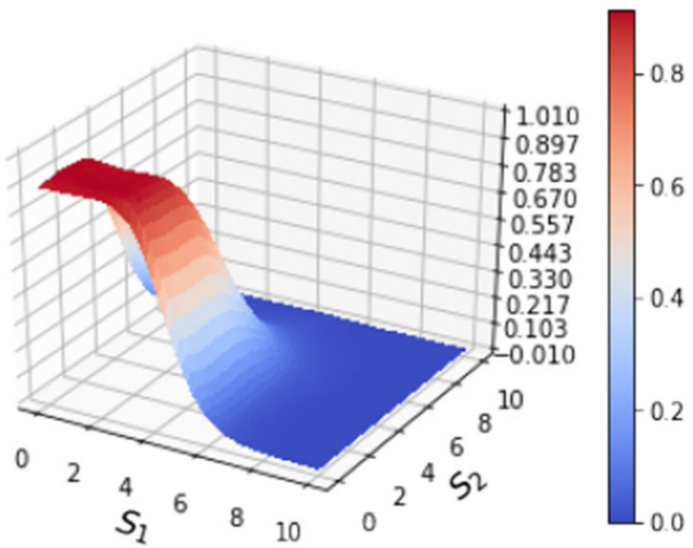
Wang et al (2022) [link]

Two-asset case

$$\frac{\partial V}{\partial \tau} - \frac{\sigma_1^2 S_1^2}{2}\frac{\partial^2 V}{\partial S_1^2} - (r-\delta_1)S_1\frac{\partial V}{\partial S_1} + rV$$
$$- \frac{\sigma_2^2 S_2^2}{2}\frac{\partial^2 V}{\partial S_2^2} - (r-\delta_2)S_2\frac{\partial V}{\partial S_2} - \rho\sigma_1\sigma_2 S_1 S_2\frac{\partial^2 V}{\partial S_1 \partial S_2} = 0, \quad \text{on } (0,\infty)^2 \times (0,T], \tag{16}$$

where the option value $V(S_1, S_2, \tau)$ depends on the asset prices S_i $(i = 1, 2)$ and the time to maturity τ. The parameter r is the interest rate, σ_i and δ_i $(i = 1, 2)$ denote the volatilities and the dividend rates for the two assets, respectively, and ρ represents the correlation coeffcient between S_1 and S_2.

Fig. 5 Example 3. Plots of the numerical solution at the final time T (Top); the training history (Bottom)



Deep learning: regression + no-arbitrage constraints

Ackerer, Takasovska and Vatter (2020),
Deep Smoothing of the Implied Vol Surface [link]

- Implied volatility ansatz (residuals)

$$\sigma = \sigma_{NN}\sigma_{prior}$$
$$\omega(k,\tau) = \sigma^2\tau$$

  where σ_{prior} comes from a model and τ: time to maturity
- 6 shape and no-arbitrage constraints

DL: regression + constraints

Proposition 2.2 *Roper [48, Theorem 2.9] Let $S > 0$, $r = \delta = 0$, and $\omega : \mathbb{R} \times [0, \infty) \mapsto \mathbb{R}$. Let ω satisfy the following conditions:*

C1) (Positivity) for every $k \in \mathbb{R}$ and $\tau > 0$, $\omega(k, \tau) > 0$.
C2) (Value at maturity) for every $k \in \mathbb{R}$, $\omega(k, 0) = 0$.
C3) (Smoothness) for every $\tau > 0$, $\omega(\cdot, \tau)$ is twice differentiable.
C4) (Monotonicity in τ) for every $k \in \mathbb{R}$, $\omega(k, \cdot)$ is non-decreasing, $\ell_{\text{cal}}(k, \tau) = \partial_\tau \omega(k, \tau) \geq 0$, where we have written ∂_τ for $\partial / \partial \tau$.
C5) (Durrleman's Condition) for every $\tau > 0$ and $k \in \mathbb{R}$,

$$\ell_{\text{but}}(k, \tau) = \left(1 - \frac{k\, \partial_k \omega(k, \tau)}{2\omega(k, \tau)}\right)^2 - \frac{\partial_k \omega(k, \tau)}{4} \left(\frac{1}{\omega(k, \tau)} + \frac{1}{4}\right) + \frac{\partial^2_{kk} \omega(k, \tau)}{2} \geq 0,$$

where we have written ∂_k for $\partial / \partial k$ and ∂_{kk} for $\partial^2 / (\partial k \partial k)$
C6) (Large moneyness behaviour) for every $\tau > 0$, $\sigma^2(k, \tau)$ is linear for $k \to \pm\infty$.

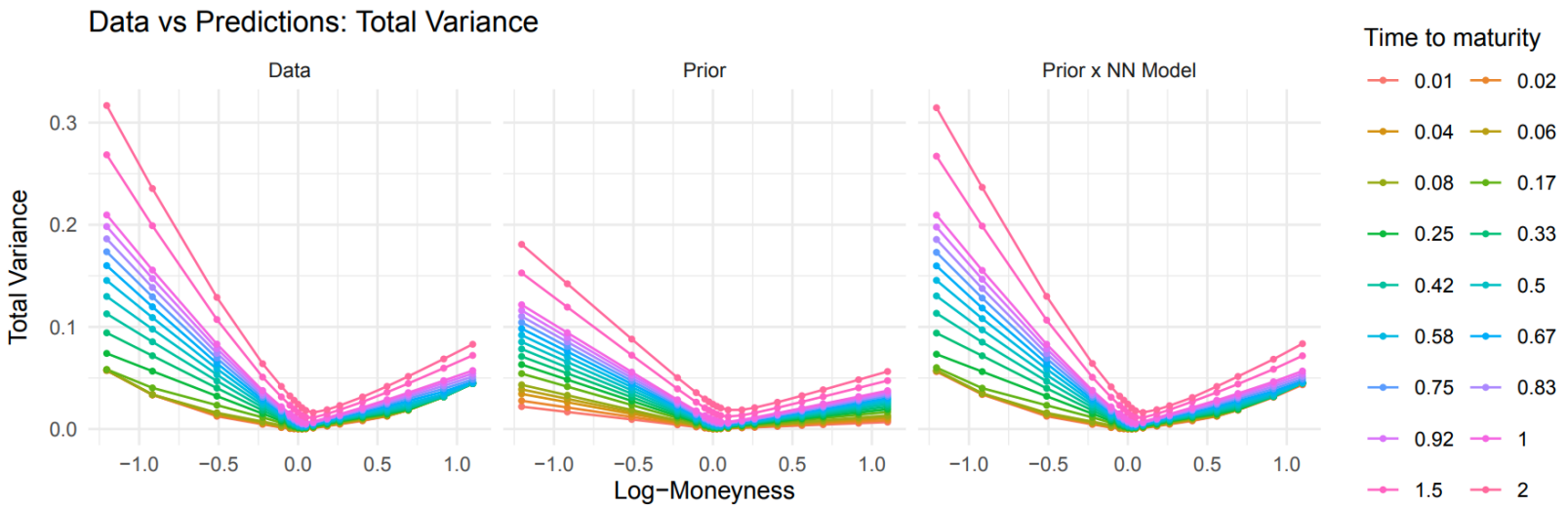
C1, C2, C3 are guaranteed by activation function

Constraints help generalization abilities

DL: regression + constraints

Loss function. We fit the network parameters and prior parameters θ by minimizing the loss function

$$\mathcal{L}(\theta) = \mathcal{L}_0(\theta) + \sum_{j=1}^{6} \lambda_j \, \mathcal{L}_{\mathrm{C}j}(\theta) \tag{6}$$



Nonarbitrage constraint: put-call parity

- C : call price
  - P : put price
  - S_t : spot (current) price
  - K : strike price
  - T : time to maturity
  - r : interest rate

$$C - P = S - e^{-rT}K$$

Custom Loss in Keras

- `Usual losses [doc]`

  `model.compile(...,loss='mse') or 'mae' or ...`

- Generically [doc][tf math]

```
import tensorflow as tf

def myloss(y_true,y_pred):
    diff2 = tf.math.square(y_true - y_pred)
    return tf.math.reduce_mean(diff2, axis=-1)


model.compile(loss=myloss)
```

(advanced): gradients in custom loss

- myloss(y_pred, y_true)
- custom loss with gradients: use GradientTape [doc] [stackoverflow]
- other ways to impose hard/soft constraints: Donti, Rolnik, Zolter (2021) [link]

Early stopping

- Until now, fix number of epochs
- Idea: stop when learning is useless
- Early stopping [link]: stop when loss or val_loss does not decrease
  - minimum/average over the last $time_{\text{patience}}$ steps
  - either train or validation loss
- code

```
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='min', patience=10)
model.fit(        ,validation_split=0.3,callbacks=[es])
```