Name of the Thesis

Candidate No:

University of Oxford

A thesis submitted in partial fulfillment of the MSc in

Mathematical and Computational Finance

June 23, 2022

Abstract

TO BE COMPLETED

Contents

1 Introduction

Motivation

Consider a typical workflow for a end-user in a trading desk: a price is needed for a particular product, a pricing function is invoked, which takes in a collection of (calibrated) input parameters, which in turn invokes the underlying numerical method, and finally the outputs, being the price and sensitivities are returned. Arguably, a key business objective for derivatives quants at investment banks and market makers is to streamline and improve this process: for any given product and the given *market generator model* (in other words, the modelling assumptions in terms of volatility or underlying processes), produce a pricing function that outputs its *prices* and its *risks* (sensitivities), to a high degree of accuracy and with minimal latency.

To accomplish this, derivatives quants have broadly three families of numerical methods: analytic and semi-analytic form solutions, Monte Carlo (MC) Methods, and Partial Differential Equation solvers (e.g. finite-difference methods). Analytic expression and expansions are only available for few market models and payoffs, for example the Black-Scholes formula for European options, the Heston characteristic transform, and SABR expansion. For many payoffs and market models, this leaves MC and PDE methods, which given a set of input model parameters (for example the beta and rho in SABR), can return a collection of prices over some time discretisation and state / strike discretisation. However, we should note that MC and PDE method are themselves approximation methods with approximation error, given the need for an appropriate state and time discretisation.

Approximation error to the 'true' model price can be reduced to some extent with greater computational effort. The computational effort may grow very large when the underlying dimensionality of the problem increases, for example with more complex payoffs and volatility models. When the *risk* sensitivities (which may also represent the hedging strategy) also need to be computed, the computational effort may grow even larger, and in addition the convergence rates in price and sensitivities may not necessarily be the same.

In the Monte Carlo and PDE settings, the solutions are only for one set of model parameters, and as model parameters change the numerical method must be reinvoked. Thus for models whose pricing depends on these schemes, calibration and other downstream tasks becomes expensive. In particular, for the pricing. Hence towards some practical applications, there may be acceptable tradeoffs between accuracy in exchange for speed, for example in the context for electronic market-making or risk / VaR calculations.

Neural-networks may be one method to accomplish this. Neural networks, in this setup, may be viewed as as a model agnostic (in the sense of volatility or market generator models) and method agnostic (MC, PDE) extension to existing numeric pricing methods. Neural Networks present several desirable properties: universal approximation, and ability to 'learn' non-parametric data-driven relationships, sensitivities can be obtained quickly with automatic differentiation, and a fast inference time can be obtained, amortising a computationally expensive training period. A vast literature on the application of neural networks toward derivatives modelling has emerged, for example [5]

In addition to any pricing function returning an accurate price for the market model and payoff, another key requisite for any pricing function is adherence to no-arbitrage, and similarly, smoothness of the pricing function and sensitivities in all inputs. Naturally, if *static* arbitrage opportunities exists, this presents the possibility of a loss to the trading desk. In terms of smoothness, an exotics / OTC desk may have to price derivatives beyond standard strikes and maturities, and a non-smooth pricing function is likely to admit static arbitrage. In addition, non-smooth sensitivities functional may also present P&L impact from hedging. On key concern, is that a standard neural network construction may not necessarily adhere to no-arbitrage and smoothness conditions.

The aim of this dissertation is to investigate how to construct a production-level neural network to approximate derivatives pricing and risks under any general payoff and market generator model. We aim to investigate:

- How to construct and train a neural network efficiently, in terms of both approximation error and minimal no-arbitrage violations? What is the empirical computational complexity required / convergence rate?
- How do neural network approximations perform under various market models, and similarly, under higher dimensionalities in terms of the number of inputs (Black-Scholes, Heston, SABR, Rough Vol)?
- How well does it perform under different payoffs (Calls and Digitals, Asians, Barriers), and exercise policy (Bermudan, European)

We aim to investigate two contexts:

- In the case of offline training of a network for calibration / electronic quotes, how can we train a network more efficiently in terms of reducing total training time and error convergence?
- In the case of difficult payoffs, e.g. exercisable products (Bermudans, Callables), path-dependent payoffs (Asians, Barriers), high dimensionality, and expensive applications (XVA, VaR), under what contexts does on-the-fly neural network training outperform standard methods?

In a bank, not only prices are needed but the first-order sensitivies, and in some cases the second-order sensitivities [32]. This is given that the sensitivities represent exposures to the input parameters, and also in some cases, the potential hedging strategy (the option delta). In some cases, higher order sensitivities need to be hedged or accounted for as well (e.g. optin gamma).

Article Structure

In section two, we review the relevant pricing problem formulations. In section three, we review neural networks. In section four, we review the relevant literature on applying neural networks for. In section five, we focus on numerical settings, examining how to construct and efficiently train neural networks that are able to approximate European and Bermudan payoffs. Finally, we conclude our observations in

2 Preliminaries

In this section, we review the formulation of derivatives pricing and risk calculations.

Problem Setup

Consider the problem of computing derivative prices and sensitivities with respect to some input parameters $\mathbf{X} = (\mathbf{F}_t, \tau = T - t, \ldots)$. First, consider a filtered probability space $(\Omega, \mathcal{F}, \mathcal{F}_t, \mathbb{P})$. Suppose there is a $\mathcal{F}_t$-adapted d-dimensional stochastic process $\mathbf{F}_t \in \mathbb{R}^d, t \in [0, T]$, representing the value of the underlying assets or cash flows through time whose evolution is determined by a fixed vector of parameters $\mathbf{X} \in \mathbb{R}^f$ (or $\mathbf{X}$ is $\mathcal{F}_t$) measurable. Thus in effect, we consider $\mathbf{F}_t(\mathbf{X})$.

Suppose that there exists some equivalent local martingale measure $\mathbb{Q}$, such that $\mathbf{F}_t$ is a $\mathbb{Q}$-local martingale. In the most general case, consider a payoff determined by a continuous discounted payoff and a terminal discounted payoff:

$$f(\tau, \mathbf{S_t}, \mathbf{X}) = \mathbb{E}^{\mathbb{Q}}\left[\int_t^T h_1(u, S_u)dt + h_2(S_T)|\mathbf{S}_t, \mathbf{X}\right]$$

Remark: In a 'real-life' scenario, consider only the case of cash flows at discrete times $\mathbb{E}^{\mathbb{Q}}\left[\sum_{i=1}^n h_i(t_i, \mathbf{S}_i)|\mathbf{S}_{t_0}, \mathbf{X}\right]$. Thus without any loss of generality, we consider pricing a single payoff; then for any derivatives, for example interest rate caps and floors , we can be priced by decomposing them as the sum of payoffs.

$$f(\tau, \mathbf{S_t}, \mathbf{X}) = \mathbb{E}^{\mathbb{Q}}\left[h_2(S_T)|\mathbf{S}_t, \mathbf{X}\right] \qquad \text{(Pricing as Conditional Expectation)}$$

Suppose that f is defined, i.e. $\mathbb{E}^{\mathbb{Q}}\left[|h_2(F_T)|\mathbf{S}_t, \mathbf{X}|\right] < \infty$ is bounded for any $\mathbf{F}_T$

Suppose f is sufficiently smooth, in particular C^1 with respect to time-to-maturity $\tau = T - t$ and C^2 with respect to each $\mathbf{S}_t$. Then by Feynman-Kac, the corresponding (parametric) PDE is given by:

$$f_\tau = \mathcal{L}f, f(0, s, x) = h(s) \quad \forall \tau \in [0, T], \mathbf{X} \in \mathcal{X}, s \in \mathcal{S} \tag{1}$$

subject to some initial (in terms of time-to-maturity) condition corresponding to the European Payoff, and $\mathcal{L}$ is the generator of $\mathbf{S}$

Thus f denotes the true value function, which represents the no-arbitrage price of the , which can be expressed as a conditional expectation or solution to a PDE.

In general, both the pricing function f and $\frac{\partial f}{\partial \theta}$ in (4), cannot be obtained in closed-form with the exception of several cases. Thus in practice we have the approximation g:

$$g(T - t, \mathbf{X}) \approx f(T - t, \mathbf{X}), \quad f(T - t, \mathbf{X}) = g(\mathbf{X}) + \epsilon$$

Where ϵ denotes the error, which may itself depend on θ and the choice of numerical method. We look for a approximating function g such that ϵ is small over a relevant range of parameters $\mathbf{X}$.

In addition to the value f, the sensitivities with respect to the parameters $\mathbf{X}$ are also of interest. Thus we also require

$$\frac{\partial g}{\partial \mathbf{X}} \approx \frac{\partial f}{\partial \mathbf{X}}, \dots, \frac{\partial^d g}{\partial \mathbf{X}}$$

Path-Dependent Payoffs: Payoffs may also be path dependent, and depend on the entire history of $\mathbf{S_t}$:

$$f(\tau, (\mathbf{S}_u)_{u\in[0,t]}, \mathbf{X}) = \mathbb{E}^{\mathbb{Q}}\left[h_2((\mathbf{S_u})_{u\in[0,T]})|(\mathbf{S}_u)_{u\in[0,t]}\mathbf{X}\right] \tag{2}$$

More generally, we also consider the underlying parameters $\mathbf{X}$ which determine $\mathbf{S}_t$

and that $h(\mathbf{X}_t)$ is a $\mathcal{F}_T$-measurable random variable representing the (discounted) total sum of cash flows, as some function of the time t state $\mathbf{X}$, under some risk-neutral measure $\mathbb{Q}$.

Stochastic Control approach

For example, if $\theta = S_0$, then $\frac{\partial f}{\partial S_0}$ gives the sensitivity to the underlying, and the hedging strategy.

American

$$\sup_{\tau} \mathbb{E}[\int_t^{\tau} B_{t,T} h_1(X_t)dt + h_2(X_\tau)|X] \tag{3}$$

$$f(T-t, \mathbf{X}_t) = \mathbb{E}^{\mathbb{Q}}[h(\mathbf{X}_t)|\mathbf{X}] \tag{4}$$

Then f is the conditional expectation and value function for a derivative with latest cash flow at time T. Under some risk-neutral measure $\mathbb{Q}$, or equivalently the assumptions of no-arbitrage and market completeness, then f is the pricing function with respect to the parameters θ.

Adjoint Automatic Differentiation

The adjoint automatic differentiation method was first brought to [**giles2004**]

In the most general case, consider the pricing function in the form of 4. Given regularity constraints

$$\frac{\partial f}{\partial \theta} = \frac{\partial}{\partial \theta}\mathbb{E}^{\mathbb{Q}}[]$$

The sensitivities of f with respect to θ, can be computed through the application of the chain rule backwards:

$$\frac{\partial S}{\partial \theta} \cdot \frac{\partial f}{\partial \theta} = \frac{\partial f}{\partial \theta}$$

where the partials are evaluated in reverse order. Programming language wise, it can be implemented via computional graphs and implementing *adjoint* operators.

No-Arbitrage Constraints

We consider the conditions for our approximating function g to be free of static arbitrage in the case of European calls. Suppose that $g \in C^{1,2}$ is continuously and and once-differentiable with respect to $\tau = T - t$ and twice differentiable with respect to K. Equivalently, we could consider moneyness $x = S/K > 0$, or log-moneyness $y = \log(S/K)$, which may be a more convenient representation in some cases, and assume g is twice differentiable with respect to:

$$\frac{\partial X}{\partial K} = \frac{S}{-K^2} = \frac{-X}{K} \qquad , \quad \frac{\partial y}{\partial K} = \frac{-1}{K}$$

$$\frac{\partial g}{\partial K} = \frac{\partial g}{\partial x}\frac{\partial x}{\partial K} = \frac{-x}{K}\frac{\partial g}{\partial X} \qquad , \quad \frac{\partial g}{\partial y}\frac{\partial y}{\partial K} = -\frac{1}{K}\frac{\partial g}{\partial y}$$

$$\frac{\partial}{\partial K}(-\frac{x}{K}\frac{\partial g}{\partial x}) = \frac{1}{K^2}(x\frac{\partial g}{\partial X} + x^2\frac{\partial^2 g}{\partial X^2})$$

$$\frac{\partial}{\partial K}(-\frac{1}{K}\frac{\partial g}{\partial y}) = \frac{1}{K^2}(\frac{\partial g}{\partial y} + \frac{\partial^2 g}{\partial y^2})$$

Then in terms of K, x, y we have:

No Static Arbitrage for European Calls: From [13] [6]

$\frac{\partial g}{\partial \tau} \geq 0$	carry, time-value
$\frac{\partial g}{\partial K} \leq 0, \frac{\partial g}{\partial x} \geq 0$	monotonically decreasing (increasing) in strike (moneyness / underlying)
$\frac{\partial^2 g}{\partial K^2} \geq 0, \frac{\partial^2 g}{\partial x} \geq 0, \frac{\partial^2 g}{\partial y} \geq 0$	convexity in strike, moneyness
$g(T-t, K) \geq (S_t - K)^+ \geq 0$	intrinsic value
$\lim_{K\to\infty} g(K, T-t) = 0$	1

We also consider no static arbitrage for European digitals. From the Breeden-Litzenberger formula we have:

Breeden-Litzenberg formula. We now have an additional approach: solve for the risk-neutral transition density $p(y, T; s, t)$ and obtain the price of any european payoff via numerical integration.

Since the payoff of a European digital $\mathbb{Q}[]$

$\frac{\partial g}{\partial \tau} \geq 0$	carry, time-value
$\frac{\partial g}{\partial K} \leq 0, \frac{\partial g}{\partial x} \geq 0$	monotonically decreasing in strike, increasing in moneyness/underlying
$\frac{\partial^2 g}{\partial K^2} \geq 0, \frac{\partial^2 g}{\partial x} \geq 0, \frac{\partial^2 g}{y} \geq 0$	convexity in strike
$g(T-t, K) \geq (S_t - K)^+ \geq 0$	intrinsic value
$\lim_{K\to\infty} g(K, T-t) = 0$	1

$$\int_{\mathbb{R}} (y - K)^+ p(y, T; s, t) dy = \int_K^\infty (y - K) p(y, T; s, t) dy$$

In the context of put-call parity, we can simply price one of the European payoffs and obtain the rest via replication, or integration from the derived transition density.

No Dynamic Arbitrage: No dynamic arbitrage indicates the lack of a replication strategy with zero initial wealth that leads to positive wealth $\mathbb{P}$-almost surely. For Dynamic Arbitrage, a sufficient condition may be that the function g satisfies the relevant pricing PDE.

$$\mathcal{L}g = g_\tau, \quad \text{for all } \tau, K$$

If $\mathcal{L}g \neq 0$, the dynamic arbitrage strategy is given by longing the payoff if $\mathcal{L}g$ and shorting the replicating portfolio, and $\mathcal{L}g < 0$, and shorting the payoff if $\mathcal{L}g < 0$ and longing the replicating strategy.

No arbitrage bounds for non-vanilla payoffs: For non-european options, we can consider replication, and in some case lower bounds. For example, in the case of Americans we must have a value greater than the equivalent european $V^A(t,s) \geq B^B(t,s)V^E(t,s)$, and for barries, we must have $V_{down} + V_{up} =$

One question is how to incorporate these no-arbitrage constraints into a neural network approximation.

Longstaff-Schwartz / Least Squares Monte Carlo

Suppose $\mathbb{E}[g(X_n)^2] < \infty$ the payoff is L^2 integrable.

[28] introduced the Longstaff-Schwartz method, or Least Squares Monte Carlo for valuing American Options. In the one step case, consider

$$f(X_t(\theta)) = \mathbb{E}^{\mathbb{Q}}[h(X_T(\theta))|X_t(\theta)] \tag{5}$$

Where X is some appropriately chosen (Markov) state process such that $f(X_t(\theta))$. Then let g denote an approximation for the value function. In [28], they consider basis functions such as Chebyshev and Legendre polynomials.

$$g(X_t) = \sum_{b=1}^{B} \phi_b(X)$$

In the multi-period case , we consider for $t_0 < t_i < \dots t_n = T$:

$$f(X_{t_i}) = \max\{\mathbb{E}^{\mathbb{Q}}[f(X_{t_{i+1}})\}|X_{t_i}], h(X_{t_i})\}, \quad f(X_T) = h(X_T)$$

Where h is the exercise value for the pay

[28] argues that as the number of basis functions B to infinity, g approximates the true value function, and as the number of timesteps N goes to infinity. Further to this, in the Monte Carlo setting, we also require the number of samples M to go to infinity.

$$f(S_t) = \mathbb{E}[(S_T - K)^+|S_t] \tag{Black-Scholes}$$

$$\mathbb{E}[(f(S_T) - \mathbb{E}[g(S_t)|S_t])^2] = 0$$

Then by the tower property of expectation

$$\mathbb{E}[(g(S_T) - \mathbb{E}[g(S_t)|S_t])^2] = 0$$

3 Neural Networks

In this section, we review some definition of relevant neural network concepts.

Neural Network Definition

A comprehensive outline of neural networks is better found in [17], and a reference on practical implementations using `Tensorflow/Keras` can be found in [8]. An overview of neural networks and their applications towards finance can be found in [11].

Consider a dataset $\mathbf{X}, \mathbf{y}, \mathbf{X} \in \mathbb{R}^{N \times d}, \mathbf{y} \in \mathbb{R}^n$. The inputs, or in machine learning jargon 'features' $\mathbf{X}$ consists of N samples of a d-dimensional vector, and $\mathbf{y} = f(\mathbf{X})$ are the corresponding outputs or 'targets' to approximate. A n-layer feed-forward neural network g (equivalently, a neural network with $n-1$ hidden layers) is characterised by:

$$\begin{aligned} \mathbf{Z}_1 &= g_1(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1\mathbf{1}_1^\top) \\ \mathbf{Z}_2 &= g_2(\mathbf{Z}_1\mathbf{W}_2 + \mathbf{b}_2\mathbf{1}_2^\top) \\ &\vdots \\ g(\mathbf{X};\boldsymbol{\theta}) &= g_n(\mathbf{Z}_{n-1}\mathbf{W}_n + \mathbf{b}_n\mathbf{1}_n^\top) \end{aligned} \tag{Feed-Forward Neural Network}$$

Weights and Biases: Let $\mathbf{W}_i \in \mathbb{R}^{H_{i-1} \times H_i}, i = 1, \ldots n-1$ denotes the *weights*, and $\mathbf{b_i} \in \mathbb{R}^{H_i}$ denotes the *bias* term respectively for the i-th hidden layer. Note that much of the neural network consists of affine transformations $\mathbf{XW} + \mathbf{b}\mathbf{1}^\top$, which are likely optimised in the underlying programming framework. Further speedups can be potentially obtained via dedicated hardware such as Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs), inference can be relatively fast.

Activation Functions: Let $g_i, i = 1, \ldots, n$, denote the activation function for the i-th hidden layer. Activation functions are applied element-wise to the inputs, such that $g_i(\mathbf{Z}_{i-1})_{j,k} = g_i(\mathbf{Z}_{i-1,j,k})$, introducing non-linear transformations into the network. Typically in a regression setting (where the output takes values in $\mathbb{R}$), the final layer is the identity activation $g_n(\mathbf{Z}_{n-1,j}) = \mathbf{Z}_{n-1,j}$, such that the output may attain any real value. However, if knowledge is available about the range of the outputs, for example if we know the output takes values $y_i \in [0,1]$, we could consider applying a final non-linear transformation g_n, for example in this case $g_n(x) = \frac{1}{1+e^{-x}}$ could be considered. Table 1 displays some common activation functions and their first and second order gradients for a single input $x \in \mathbb{R}$.

Learnable Parameters: Let $\boldsymbol{\theta}$ denotes all *learnable* parameters of the neural network. In a feed-forward network, this amounts to the collection of all weights and biases $(\mathbf{W}_1, \ldots \mathbf{W}_n, \mathbf{b}_1, \ldots, \mathbf{b}_n)$. A neural network is non-parametric in the sense that we do not necessarily have to make any assumptions about or specify the functional or distributional relationships between inputs and targets $\mathbf{X}, \mathbf{y}$, but the values of $\boldsymbol{\theta}$ need to be determined or 'learnt' over some space of possible parameters Θ. For example, we could simply take the product of all real-valued matrices and vectors with the corresponding dimension as the weights and biases $\Theta = \prod_{i=1}^n \{\mathbb{R}^{H_{i-1} \times H_i}\} \prod_{i=1}^n \mathbb{R}^{H_i}$.

Activation	$g_i(x)$	$g_i'(x)$	$g_i''(x)$
ReLU	$\max\{x, 0\}$	$1_{x>0}$	0
LeakyReLU	$\max\{0, x\} + \alpha \min\{0, x\}$	$1_{x>0} + \alpha 1_{x<0}$	0
ELU	$\alpha(e^x - 1)1_{x<0} + x1_{x>0}$	$1_{x>0} + \alpha e^x 1_{x<0}$	$\alpha e^x 1_{x<0}$
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{(1+e^{-x})^2}$	$\frac{e^{-x}(e^{-x}-1)}{(1+e^{-x})^3}$
SoftPlus	$\log(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{(1+e^{-x})^2}$
Swish	$\frac{x}{1+e^{-x}}$	$\frac{1+((x+1))e^{-x}}{(1+e^{-x})^2}$	$\frac{((2-x)+(x-2)e^{-x})e^{-x}}{(1+e^{-x})^3}$
GeLU	$x\Phi(x)$	$x\phi(x) + \Phi(x)$	$(2 - x^2)\phi(x)$
tanh	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$\frac{4}{(e^x+e^{-x})^2}$	$\frac{-8(e^x-e^{-x})}{(e^x+e^{-x})^3}$
RBF	$\exp(-\frac{x^2}{2})$	$-x\exp(-\frac{x^2}{2})$	$(x^2 - 1)\exp(-\frac{x^2}{2})$

Where $\alpha > 0$ denotes a hyperparameter, and Φ, ϕ denotes the cumulative density and probability density functions for the Gaussian distribution respectively.

Table 1: Activation Functions and their Derivatives

Neural Networks as basis functions: One way of viewing a neural network may be to consider it as non-parametric 'learning' a collection of non-linear basis (also referred to as 'latent representation' or 'features') $\mathbf{Z}_n$ from inputs $\mathbf{X}$ that is useful for the given. Consider that if we have no final activation g_n, then the output is simply $\mathbf{Z}_n\mathbf{W} + \mathbf{b}\mathbf{1}^\top$ a ordinary least squares regression of $\mathbf{y}$ on $\mathbf{Z}$. However, compared with standard basis regression, there is no need to specify the basis functions $\mathbf{Z}_n$ explicitly.

Neural Networks as a composition of neural networks: We also note that the neural network g is a composition of $g_n(g_{n-1}(\cdots))$, in effect we can regard any general neural network as a composition of neural networks. The implication of this, is that we could consider neural networks with multiple outputs instead of 1, or combine neural networks by considering for example a new neural network with $g^3 = g_n((\mathbf{Z}_n^1\mathbf{Z}_n^2)\mathbf{W}_n + \mathbf{b}_n\mathbf{1}^\top)$

Theorem 1 (Hornik (1990)) *Let $\mathcal{N}_{H_0,H_1,g}$ be the set of neural networks mapping from $\mathbb{R}^{H_0} \to \mathbb{R}^{H_1}$, with activation function $g : \mathbb{R} \to \mathbb{R}$. Suppose $f \in C^k$ is continuously k-times differentiable. Then if $g \in C^k(\mathbb{R})$ is continuously k-times differentiable, then $\mathcal{N}_{H_0,H_1,g}$ arbitrarily approximates f and its derivatives up to order n.*

The universal approximation of theorem [**Hornik**] suggests there exists some neural network that arbitrarily approximates. This informs the choices of potential activation functions: for our derivatives modelling application, this suggests to obtain k-th order sensitivities, we must use a continuous k-times differentiable C^k activation function g. This rules out the commonly used Rectified Linear Unit (ReLU) activation function, and the LeakyReLU function. However, the universal approximation theorem only proves existence of such a neural network, and not how to construct it. For example, we have the questions of the architectural choices: the number of hidden layers, hidden units in each layer, and the (smooth) activation function, and how to determine the learnable parameters θ.

Training Neural Networks and Practical Issues

Suppose we a fix a neural network architecture, i.e. the number of hidden layers n, the hidden units H_i, and the activation functions g_i. Then we define optimality with respect to some objective (loss) function L.

$$\arg\min_{\theta \in \Theta} L(\mathbf{y}, g(\mathbf{X})) \tag{6}$$

the minimizer of some objective ('loss') function' L, over a space of feasible parameters Θ. Thus we define the 'optimality' of a neural network with respect to a particular loss function. For example, we could consider the expected mean squared error

$$\arg\min_{\theta \in \Theta} \mathbb{E}[\|\mathbf{y} - f(\mathbf{X}; \theta)\|^2] \tag{7}$$

In equation (7), $\mathbf{X}$ denotes a random variable over some probability space. Thus in effect, we want the $L^2 \times \mathbb{P}$ norm. However, in actuality $\mathbf{X}$ is a finite sample $\mathbf{X}$. Thus we aim to minimise the L^2 error over the empirical measure.

This overall approach can be referred to as *supervised machine learning.* In this case, we have explicit input output pairs $\mathbf{X}, \mathbf{y}$, which we want the neural network to learn some relationship.

From 7, this amounts to finding the optimal Θ. This loss function is generally non-convex with respect to the parameters θ; as such there are no guarantees of convergence to a global minima except under certain conditions.

Firstly, we initialise the parameters $\boldsymbol{\theta}_0$. He initialisation:

In practice the gradients with respect to the neural network parameters $\nabla_\theta L$ also need to be estimated. Mini-batch stochastic gradient descent

Algorithm 1 Mini-Batch Stochastic Gradient Descent

```
Initialise parameters $\theta_0$ randomly via PRNG, $t = 0$
while $t \leq$ EPOCHS $\times \lceil \frac{N}{\text{BatchSize}} \rceil$ or NOT StoppingCriterion do
    Compute loss $L(y^{batch}, g(\mathbf{X}^{batch}))$ for batch = $\lceil 1, \ldots, \frac{N}{\text{BatchSize}} \rceil$
    Compute gradient w.r.t. loss $\nabla L$ via AAD
    $t \leftarrow t + 1$
    Set $\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla_\theta L$
    if StoppingCriterion then Break
    end if
end while
```

The gradient of the loss L with respect to the parameters $\boldsymbol{theta}$, $\frac{\partial L}{\partial \boldsymbol{\theta}}$ are obtained via in-built AAD in the neural network framework of choice.

Then and updated via backpropogation algrithm.

This procedure is repeated over all batches in the training dataset $\mathbf{X}$, and repeated until the earlier of a given number of iterations (epochs) or some stopping criterion.

In practice, several neural network techniques have been found empirically to improve training speed and generalisation: *Other Optimizers*: Adam *learning rate scheduling*: have been found to lead to empirically faster training

Early Stopping

Batch Normalisation: Samples are normalised $(\mathbf{X}_i \ominus \mu_i)/ \oslash \sigma_i$

Dropout: [35] proposed the *Dropout* method as a form of neural network regularisation. During training, fraction p of hidden units are set to zero, and the remaining units are scaled by $1/p$ to preserve the expected mean and variance. $\mathbf{X} \otimes \mathbf{R}\frac{1}{p}$, where theh entries of $R_{ij} \sim Bin(p)$ are bernoulli distributed with probability p.

Regularisation: setting constraints on the weights, for example a penalty on the L^1 or L^2 norm on the hidden layer weights $\mathbf{W}_i$. This amounts to modifying the loss function, for example: $L + \lambda \sum_{i=1}^{n} \|\mathbf{W}_i\|_2$

Activation Functions

The first neural network design choice is the choice of activation function.

In the neural network, in lieu of x we apply g_i element-wise to the affine transformation $\mathbf{Z}_{i-1}\mathbf{W}_i + \mathbf{b}_i\mathbf{1}^\top$. From **??**, we need to compute the gradients of g_i with respect to

Although we could explicitly compute the gradients of a activation function (as shown above), we do not need to do so explicitly with standard neural network framework as an optimised adjoint version is implemented, and its gradients are obtained quickly via AAD.

Neural Network Architectures

The universal approximation theorem of [**hornik**] only suggests that there *exists* a neural network. Thus in practice, we need to determine the network architecture, in particular, the key choices are the number of layers (depth), the number of hidden units in each layer (width) and the activation function. Empirically, architectures that have been selected for domain-specific problems as opposed to . We consider several hidden layer or block architectures we will utilise in later sections.

Gated unit: In the context, this can help learn feature interactions, and also facilitate feature *selection.*

$$\mathbf{Z}_1 = g_1(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1\mathbf{1}^\top) \tag{8}$$

$$\mathbf{Z}_2 = g_2(\mathbf{X}\mathbf{W}_2 + \mathbf{b}_2\mathbf{1}^\top) \tag{9}$$

$$\mathbf{Z}_3 = \mathbf{Z}_1 \otimes \mathbf{Z}_2 \tag{gated block}$$

Where $\otimes$ denotes element-wise multiplication. [38] uses gated units to construct a neural network that has the requisite monotonicity and convexity constraints with respect to strike and moneyness.

Residual block: Residual blocks allow for the *flow* of information from earlier layers to later layers

$$\mathbf{Z}_1 = g_1(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1\mathbf{1}^\top) \tag{10}$$
$$\mathbf{Z}_2 = g_2(\mathbf{Z}_1\mathbf{W}_2 + \mathbf{b_2}\mathbf{1}^\top) \tag{11}$$
$$\mathbf{Z}_3 = g_3(\begin{pmatrix}\mathbf{Z}_1 & \mathbf{Z}_2\end{pmatrix}\mathbf{W}_3 + \mathbf{b}_3\mathbf{1}^\top) \tag{gated block}$$

Recurrent neural networks: Recurrent neural networks (RNNs) capture sequential, or temporal dependencies, and have had applications in text and time series modelling.

$$\mathbf{Z}_1 = g_1(\mathbf{X}\mathbf{W}_1 + \mathbf{b}\mathbf{1}^\top) \tag{12}$$
$$\mathbf{Z}_2 = g_2(\mathbf{Z}_1\mathbf{W}_2 + \mathbf{b}\mathbf{1}^\top) \tag{13}$$
$$\mathbf{y} = f(\mathbf{X};\theta) = g_n(\mathbf{Z}_n\mathbf{W}_n + \mathbf{b}\mathbf{1}^\top) \tag{gated block}$$

Convolutional Neural Networks: Convolutional Neural Networks (ConvNets) are able to capture local dependencies, and have had great success in image and video modelling.

$$\mathbf{Z}_1 = g_1(\mathbf{X}\mathbf{W}_1 + \mathbf{b}\mathbf{1}^\top) \tag{14}$$
$$\mathbf{Z}_2 = g_2(\mathbf{Z}_1\mathbf{W}_2 + \mathbf{b}\mathbf{1}^\top) \tag{15}$$
$$\mathbf{y} = f(\mathbf{X};\theta) = g_n(\mathbf{Z}_n\mathbf{W}_n + \mathbf{b}\mathbf{1}^\top) \tag{gated block}$$

? considers the use of ConvNets to model the implied volatility surface as an image.

Determining Optimal Architecture

AutoML, Hyperparameter Tuning: One way to determine the optimal architecture is to consider neural architecture search or automatic machine learning (AutoML). In effect, we consider some space of neural networks $\mathcal{N}$, for example in terms of the number of hidden layers or hidden units. We then evaluate some finite subset of $\mathcal{N}_{sub} \subset \mathcal{N}$ and consider:

$$\arg \min_{g_i \in \mathcal{N}_{sub}} \min_{\theta \in \Theta_{g_i}} L \tag{16}$$

A naive method to evaluate $\mathcal{N}$ could be to define some small finite search space (grid-search) $\mathcal{N}_{sub} = \mathcal{N}$, for example through a cartesian product on a finite set of possible hidden units, layers, and activation functions

$$\text{Hidden Units} \times \text{Hidden Layers} \times \text{Activation} \in \{32, 64, 128\} \times \{32, 64, 128\} \times \{\text{ReLU}, \text{ELU}\}$$

However, for a larger search space (or for example, with continuous parameters), and also in the case of a fixed time constraint, a brute-force search becomes infeasible. Instead, a randomised subset must be considered through a random search, or more sophisticated optimisation methods such as Bayesian optimisation. In `Tensorflow/Keras`, this can be implemented through the `KerasTuner` API.

[20] used a neural network to calibrate swaption and yield curve as input to . They discussed that even if the architecture choice is fixed (for example whether to use a ResNet or Feed-forward neural network), there is still the architectural choices of number of hidden units, layers, and other hyperparameters.

Ensembling: Consider that even for the same architecture and training scheme, there is some randomisation and results may differ due to the initial random seed for weight initialisation. Rather than considering a single neural network, we could consider multiple neural networks. Supposing we have n neural networks $g^1, \ldots, g^n$, we could take a simple or weighted average of the price and gradient outputs

$$\hat{f}(\mathbf{X}; \theta) = \sum_{i=1}^{n} w_i g^i(\mathbf{X}; \theta), \frac{\partial \hat{f}}{\partial \mathbf{X}}$$

Where the weightings w_i could be further learnt.

This can be also used to obtain uncertainty bounds. For example, [16] considers initialising multiple random seeds, to obtain a confidence interval on prices. A somewhat related concept is Bayesian Neural Networks.

$$\Big[\min\{g^1(\mathbf{X}; \theta_1), \ldots g^n(\cdot; \theta_2)\} = \hat{f}(\mathbf{X}; \theta^-), \hat{f}(\mathbf{X}; \theta^+) = \max\{g^1(\mathbf{X}; \theta_1), \ldots g^n(\mathbf{X}; \theta_2)\}\Big]$$

A potential drawback of this approach is that this may potentially lead to a more complex implementation, and reduce inference speed somewhat, given the need to evaluate n neural networks instead of 1.

Transfer Learning: Transfer Learning is a method has been applied to some success in image-related modelling, which involves retraining a neural network (usually the last few layers) which has already been trained on some other similar task. In the image context, for example, this may be retraining a neural network trained to classify cats to classify dogs instead. In the derivatives pricing context, we could assume for example, that the learnt basis functions in the final layer in a neural network for European Calls may also be useful for European Puts, and retrain only the final layer accordingly. [3] [15] explore this application.

4 Neural Networks for Derivatives Pricing

Having defined the pricing and sensitivities and neural networks, in this section we review some of the existing literature and methods for neural networks in derivatives pricing and risks.

Modelling Choices, Literature Review

A machine learning setup generally consists of several components: the modelling objective, the dataset, the model, training, and inference.

The first question is how to approach modelling for the pricing approximation task. To date, various approaches utilising neural networks for derivatives pricing and hedging have been presented, of which a comprehensive review of methods can be further found in [33]:

Supervised: Direct approximation of model prices. [23] was one of the first papers use neural networks for option pricing, although their application was on real options pricing data. Their approach to use neural networks as a direct funcntion approximation between inputs, and S&P500 options price data.

However, [23] did not consider sensitivities, automatic differentiation can be used as in .

Neural SDEs: Represent the dynamics of a SDE $\mathbf{S}_t$ as a nueral network, as in [16] and Generative Adversarial Networks, and calibrate by minimising the MC pricing error between observed prices $\|P(T,K) - \mathbb{E}[h(S_T, K)]\|$. This approach has the potential advantage of allowing more realistic dynamics to be captured; however, being a Monte-Carlo based approach, the actual inference time may be slow.

Model hedging strategy; Approximation for a hedging or replicating strategy; In effect this can be seen as pricing via replication or stochastic control [5]. The *Deep Hedging* approach of [5] considers approximating the replicating strategy. Thus in this case, the neural network represents the delta.

$$y - \sum_{i=1}^{N} \Delta_i(X_{t_i})(S_{i+1} - S_i)$$

Learn the hedging strategy with a neural network, and price can be obtained via superhedging. However the drawback is that this hedging strategy is tuned to a single payoff and requires MC simulations for pricing. Other papers that explore this reinforcement learning based approach include

We focus on the supervised learning approach. In addition, the hedging and neural PDE approaches can also be incorporated to some extent, by including them in the loss function for the supervised task.

However, even within the supervised learning ask. In effect, the choice of input-output pairs $(\mathbf{X}, \mathbf{y})$.

Model risk-neutral implied distribution: Let $g(T, y/K; t, X)$ be a estimate for the conditional density at time T. Thus in effect, we approximation the price of a European digital and obtain (in the 1D case).

$$g(T-t,X) \approx DF_{t,T} \int_K^{\infty} (\frac{y}{K}-1)^+ g(T,y;t,X)dy$$

Model Control Variate Residuals:

Supervised - Calibration: learn mappings for implied volatilities/prices to parameters, or vice-versa [21]. [21] also considers a grid based approach, where a fix set of strikes are predicted, and an image based approach where a fixed collection of maturities and strikes are modelled.

Dataset Construction

Machine Learning Approach: In the machine learning framework, typically three datasets are constructed: A *train* dataset, which, the model is directly trained. In theory, given that $\mathbf{X}$ is sampled from the same distribution. In our case, we know the true distribution of $\mathbf{X}$, given that for a known volatility model / SDE, we can simulate it. [20] mentions the possibility of sampling from historical joint distributions parameter, to capture a meaningful space of parameter relationships (as opposed to sampling uniformly in a hypercube).

In [22], their ys are based on sample Monte Carlo payoffs. However, given that in our application we must also consider

In the general case of a standard diffusion process, we could consider

$$123123 \tag{17}$$

Data Generation Method. Train, Val: Hypercube / Grid sampling vs Random sampling, Random sampling within boundary

In the case of machine learning Quasi Monte Carlo, Sobol Sequences, Control Variates

Test: Another random sample within boundary, random sample outside of boundary

In a real life setting, next-day prices could be used, although this does not necessarily make sense.

Data Preprocessing

The dataset may also contain arbitrageable prices, for example in Monte Carlo simulations. In the Longstaff-Schwartz paper, they simply set out-of-the money paths to weight zero:

$$\sum_{j=1}^{N} w_j L(f(X_j), g(X_j))$$

In the Longstaff-Schwartz case, paths for which $f(X_j)$ have weight zero.

In this same style, [10]. Preprocessing, remove arbitrage out of the money paths.

Dataset Quality and Risk: The risk of the neural network is also present through *data risk*. Dataset quality For example, how does the NN perform on unseen parameter ranges?, An issue is that the NN may be able to , e.g. when the market regime shifts. Thus the NN must likely be trained on a very large grid of parameters.

In [22] and the standard Longstaff-Schwartz approach,

In the case of hedging, FBSE, and American / Bermudan options modelling,

The neural network approximation approach is more closely aligns with a Monte Carlo method. Thus there is error from both the neural network approximation g and the simulation of state-payoff pairs X_j, y_j

For example, consider the case of very deep in-the-money or out-of-the money strikes for a European all. A proposed solution of is to sample more from these regions, or similarly to set a greater weight w_j in a weighted loss function

Swaption

Neural Network Construction for Derivatives Pricing and Risks

We characterise handcrafted neural networks as described in [33]: in short, we can embed prior knowledge into the construction of the neural network, in the choice of an appropriate loss function or architecture.

[24] propose a modified elu function with: $R(z) = \alpha(e^z - 1)1_{z\leq 0}+$

Smoothness. Firstly, to obtain smooth (or at least, continuous) approximations for the d-th order partial derivatives, we require $g(\cdot)$ to be C^d continuous d-time differentiable with respect to its inputs [24]. Given the loss functions in 20, 21, may contain even higher-order partial derivative terms, we may require even further smoothness constraints.

In order for the neural network output g to be C^d, one method could be to constrain all intermediate activation functions g_i to be sufficiently smooth, such that g as a composition of smoothness will be a smooth function. [7] uses this approach.

Hard constraints [7] *softplus* activation for the strike and sigmoid activation for the time-to-maturity, with non-negative weight constraints. Thus this guarantees that the neural network is non-negative, monotonic in strike and time, and convex in strike. **Asymptotics**: [1] highlights that neural networks are generally able to interpolate within the domain of training, but unable to extrapolate. This is also have particular relevance, given models have asymptotics, and payoff asymptotics correspond to no-arbitrage bounds.

The proposed solution of [1] uses a control-variate-like two-step procedure: first, they fit a *control variate function* , in their case cubic spline, with the appropriate asymptotics, then they fit a neural network with vanishing asymptotics to the residual of the original function and the control variate.

Their proposed architecture consists of the following gaussian kernel, or radial basis (RBF) activation

$$g_i(\mathbf{x}) = \exp(-\frac{1}{2}\|\mathbf{x}\mathbf{W}_i + \mathbf{b}_i\|^2) \tag{18}$$

Such that if any $x_i \to \pm\infty$ we have $g_i(\mathbf{x}) \to 0$. Thus if our neural network consists only of RBF activations, we can obtain the target asymptotics.

Loss Functions for Derivatives Pricing and Risks

We now consider a choice of specific loss functions for the supervised learning task.

In the most simple case, we consider direct approximation, for example with mean squared error as a loss function. For a single observation: (X_i, y_i)

$$L_{price}(y_i, g(X_i)) = (y_i - X_i)^2 \tag{19}$$

The proposed method from [22] is to consider a joint loss function, an approach they describe as *differential machine learning.*

$$L_{price}(g(X_i), f(X_i)) + \lambda L_{greeks}\left(\frac{\partial g(X_i)}{\partial X}, \frac{\partial g(X_i)}{\partial X}\right), \lambda \geq 0 \tag{20}$$

The partial derivative $\frac{\partial f(X_i)}{\partial X}$ can be obtained at some (small) extra computational cost given AAD. [22] argues that the λ_1 is not significant and thhey set it to $\lambda_1 = 1$, but in practice this could be determined by considering prior knowledge of the relative scales, or through trial-and-error.

Neural PDEs: A further extension could be to consider the corresponding PDE associated with the , which can be described as a *Physics-Inspired Neural Network* or a *Neural PDE* approach.

Use of neural networks to solve (high-dimensional) PDEs as in [**sirignano**]. Given that in some cases, pricing problems can be formulated as. In addition, in our application we would need to solve the PDE over a range of model parameters.

$$L_{price}(g(X_i), f(X_i)) + \lambda_1 L_{greeks}\left(\frac{\partial g(X_i)}{\partial X}, \frac{\partial g(X_i)}{\partial X}\right) + \lambda_2 L_{PDE}(\mathcal{L}g) \tag{21}$$

Where $\mathcal{L}$ denotes a PDE operator and $\lambda_1, \lambda_2 \geq 0$ are weights for each loss function. For example, in the case of black scholes, we add the differential:

$$\mathcal{L}g = \frac{\partial g}{\partial t} + rs\frac{\partial g}{\partial s} + \frac{\sigma^2 s^2}{2}\frac{\partial^2 g}{\partial s^2} - rg$$

[37] argues that the inclusion of a PDE loss term leads to self-consistency, given that if the neural network approximation satisfies the pricing PDE (in their application, the Dupire Local Volatility PDE), $\mathcal{L}g = 0$ there is no dynamic arbitrage.

[30] proposes a method to determine the relative weightings between the pricing error and the PDE error

Soft Penalties: Finally, we can also consider any no-arbitrage constraints. An alternative is a *soft penalty* approach [24]. For example, we could consider any of the no-arbitrage constraints, and include a loss term if the . One such example could be a penalty $((S-K)^+ - g(S))^+$, i.e. a penalty for being beneath the intrinsic call bound. let $L_{cons_1}, \ldots L_{cons_P}$ denote P soft penalties. In the most general case, the loss function consists of:

$$L_{price} + \lambda_1 L_{greeks} + \lambda_2 L_{PDE}(\mathcal{L}g) + \sum_{i=1}^{P} \lambda_i L_{cons_i} \tag{22}$$

This is similar to the penalty method for finite differences or optimisation.

In practice, the inclusion of higher order differentials increases the total training time. In the subsequent sections, we explore whether the inclusion of each different penalty leads to better results, in terms of error and no-arbitrage.

Interpreting and Monitoring Neural Networks

We discuss briefly the issues with neural networks which may prevent its practical usage, namely its 'black-box' nature and lack of interepretability. In this case, we only use neural networks as an approximating function and overlay on top of some given market generator model and numerical method, which may alleviate some of the *black-box* issues [9]. However, the neural network may still produce unexpected outputs. We can evaluate and interpret neural networks to some extent. In the simple case, we can use graphical methods, or evaluate behaviour around boundary conditions. In addition, we can leverage machine learning interpretability methods [31], which [4] explores in the context of using deep neural networks for Heston calibration. For example, to interpret the pricing function, we could consider:

- Dependency plots against one or two dependent variables, boundary conditions.
- First order partial derivatives, Second order (Hessian), higher order derivatives.
- Output of penultimate layer (basis functions or latent representation)
- Machine-learning interpretabilty methods: Shapley Values, LIME

For evaluating the correctness in the implementation of the NN greeks, we could compare the NN AAD greeks versus finite differences. In terms of monitoring and deploying the neural network. In the training period, we obtain an estimate of the pricing errors for some range of parameters. We could define a region of parameters $\mathcal{X} \subseteq \mathbb{R}^d$ for which the maximum error $\boldsymbol{e} = \mathbf{y} - g(\mathbf{X}; \boldsymbol{\theta})$ of the neural network is under some ϵ. These could be stored as simply $2d$ linear constraints $d_i < X_i < u_i$ for each parameter. If the pricer is evoked outside this region, we simply revert to the original numerical method. Another method could be to consider confidence intervals, for example with an ensemble as previously described, or simply constructing a confidence interval from the standard deviation of the neural network errors $\sigma = Var(\mathbf{e})$. If however, the market parameters have been consistently away the training region of parameters, in other words, *distribution drift*, or the contract structure of a derivative changes, then this necessitates retraining of the neural network.

[27] mentions frameworks such as the NVIDIA TensorRT, which can further be used to speedup inference. [32] discusses computing the sensitivities / Jacobian / Hessian with respect to $\mathbf{X}$ explicitly as opposed to using AAD, given that AAD computes all derivatives

[19] discusses some strategies for optimising neural networks from a programming leve.

Alternative Methods and Why Neural Networks?

Alternative basis functions: For function approximation, a natural comparison could be made with standard polynomial or basis regression as in [28]. The Stone-Weierstrass theorem asserts that any continuous function on a compact set can be approximated by polynomial

Gaussian Processes / Kernel methods: [26] [] among other papers explored the use of Gaussian Processes, which have similar properties. However, naive implementations of Gaussian Processes have $O(N^3)$ time complexity [11] Chapter, which suggests that they cannot necessarily scale to a large number of training points.

Tree-based Methods: Some papers, for example [12] have also looked at tree-based methods such as Gradient Boosted Trees, such as However, as Gradient Boosted Trees are in effect sums of indicator functions, the corresponding greeks cannot be obtained.

[2] explored the use of Tensor Methods. [36] discussed the use of the Karhunone-louvre basis and signatures. Chebyschev Methods.

To summarise, Neural Networks are advantageous, in that they are scalable to large amounts of training data, able to learn non-parametric relationships. Inference time is relatively fast after training. The disadvantage is that there is no guarantee of convergence or desired properties except in few theoretical cases.

5 Numerical Experiments - Offline

In our numerical experiments, our general workflow is as follows:

- We consider a forward value processes F_t which is a $\mathbb{Q}$-local martingale with zero drift. In practice, if F_t is not a local martingale, we could consider some appropriate change of numéraire, or consider the chain rule / AAD to obtain the relevant sensitiviies,
- We consider exploiting homogeneity, or other properties for an initial dimensionality reduction of the problem.
- We then construct a dataset, consider multiple neural networks and train them, then evaluate them.

The metrics we aim to evaluate our experiments on are:

- Time Complexity: Dataset Construction (Size (Timesteps, Number of MC paths), method), Training time, Inference time of price + sensitivities
- Model Complexity (No. of parameters, (width, depth). The choice of Model Architecture: Standard, Differential, Neural PDE
- Errors (L1, L2 , Linf) in Price, First Order, Second Order sensitivities, against closed form if available.
- No-arbitrage violations, asymptotic behaviour, extrapolation behaviour
- Comparison vs MC, PDE, other approximation methods
- Applications: Pathwise Hedging Error $(V_T - \sum_{i=1}^{N} \Delta_{t_i} \cdot (S_{t_i+1} - S_{t_i}) - P_0)$ (Loss, CVaR / Quantiles), or explained PNL

We consider two setups. In these setups, we fix the neural network architecture in terms of the number of hidden units, and compare the different training methods. In a practical setting, the architecture may be fixed due to computational limitationns, in particular the tradeoff between network size an infnerence time. Thus in this context, we in effect investigate the relative tradeoffs between the training method and different performance metrics.

Geometric Brownian Motion / Black-Scholes

We consider the most simple case: European Call pricing in 1D Black-Scholes

SDE and MC: We have

$$\begin{aligned}
dF_t = F_t\sigma dW_t, \quad F_t &= F_0\exp(-\frac{\sigma^2}{2}t + \sigma\sqrt{t}\frac{W_t}{\sqrt{t}}) \\
dX_t = d(F_t/K) &= \frac{F_t}{K}\sigma dW_t = X_t\sigma dW_t \\
d\log(F_t) &= \frac{-\sigma^2 t}{2}dt + \sigma W_t \\
h(S_T) &= \left(\frac{F_T}{K} - 1\right)^+ = (X_T - 1)^+ = (\exp(y_T) - 1)^+
\end{aligned}$$

In the case of Black-Scholes, we can exploit positive homogeneity

$$\mathbb{E}^{\mathbb{Q}}[\frac{(F_t}{K} - 1)^+] = \frac{\mathbb{E}^{\mathbb{Q}}[(F_T - K)^+|S_t, K, \sigma]}{K} \tag{23}$$

$$\lambda C(F_t, K, \sigma, \tau) = C(\lambda F_t, \lambda K, \sigma, \tau) \tag{24}$$

Thus we can eliminate one of F_t, K by letting $\lambda = \frac{1}{K}$ and fixing. We can further eliminate one of σ, τ by noting that in the Black-Scholes formula:

$$\begin{aligned}
d_\pm &= \frac{\log(F_t/K)}{\sigma\sqrt{\tau}} \pm \frac{(\sigma\sqrt{\tau})^2}{2} \\
C(\frac{F_t}{K}, 1, \sigma, \tau) &= \frac{F_t}{K}\Phi(d_+) - \Phi(d_-)
\end{aligned}$$

The σ and τ terms are grouped together as
PDE: The corresponding PDE is given by:

$$V_\tau + \frac{s^2\sigma^2 V_{ss}}{2} = 0 \tag{25}$$

$$V_{\sigma\sqrt{\tau}} + \frac{V_y}{} \tag{26}$$

Dataset:

$y = \log(F_t/K)$	$\sigma\sqrt{\tau}$
Log-moneyness	Time-scaled Implied Volatility
$[-3, 3]$	$[0, 3]$

	Feed-Forward	Gated	Neural PDE
pred_l1	0.022172	0.055315	**0.011959**
pred_l2	0.028844	0.085589	**0.019299**
pred_l_inf	**0.093755**	1.415525	0.557627
Intrinsic Value	0.027328	0.223175	0.183304
PDE_mean	-0.033249	0.144910	0.002128
PDE_l1	0.111167	0.388884	0.011846
PDE_l2	0.148716	0.707390	0.017840
PDE_l_inf	2.882793	4.383477	0.557461
monotonicity_error	0.000000	0.000000	0.000000
time_value_error	0.022888	0.000000	0.032928
convex_error	0.000000	0.000000	0.000000

We consider the second example from [22], 1D Black-Scholes for a single time-to-maturity $T - t$ with respect to different levels of the underlying S_t.

We parameterise in terms of moneyness $\frac{S_T}{K}$ and exploit the homogeneity of Black-Scholes. $\frac{\partial f}{\partial x}\frac{\partial x}{\partial s} =$

The lack. Thus we consider a parameterisation with respect to moneyness and time-scaled implied volatility

$$f(S_t/K, \sigma\sqrt{T-t})$$

In this case, we can leverage the homogeneity of Black-Scholes, and reduce our problem to the process $\log(S_t/K)$ instead of S_t and time-scaled implied volatility $\sigma\sqrt{\tau}$ instead of τ

$$V_t + rsV_s - + \frac{V_{ss}\sigma^2 s^2}{2} - rV = 0$$

$$\frac{\partial\sigma\sqrt{\tau}}{\partial t} = \frac{\sigma^2}{2\sqrt{\tau}}, \frac{\partial\log(S/K)}{\partial s} = \frac{1}{s},$$

$$-V_{\sigma\sqrt{\tau}}\frac{\sigma^2}{2\sigma\sqrt{\tau}} - \frac{\sigma^2}{2}V_x + \frac{\sigma^2 V_{xx}}{2} = 0$$

$$V_t + rsV_s + \frac{s^2 V_{ss}\sigma^2}{2} - rV = 0$$

$x = \log(S/K), \frac{\partial x}{\partial s} = \frac{1}{s}, \frac{\partial^2 x}{\partial s^2} = \frac{-1}{s^2}$

$$V_{\sigma\sqrt{T-t}}\frac{\sigma}{2\sqrt{T-t}} + rsV_x\frac{\partial x}{\partial s} + \frac{s^2\sigma^2}{2}[V_{xx}(\frac{\partial x}{\partial s})^2 + V_x\frac{\partial^2 x}{\partial s^2}] - rV = 0$$

$$V_{\sigma\sqrt{T-t}}\frac{\sigma}{2\sqrt{T-t}} + rsV_x\frac{\partial x}{\partial s} + \frac{s^2\sigma^2}{2}[V_{xx}(\frac{\partial x}{\partial s})^2 + V_x\frac{\partial^2 x}{\partial s^2}] - rV = 0$$

$$-V_\sigma\sqrt{T-t}\frac{\sigma}{2\sqrt{T-t}} + (r - \frac{\sigma^2}{2})V_x + \frac{\sigma^2 V_{xx}}{2} - rV = 0$$

$$\frac{-V_\sigma\sqrt{T-t}}{\sigma\sqrt{T-t}} + (\frac{r}{2\sigma^2} - 1)V_x + V_{xx} - \frac{r}{2\sigma^2}V = 0$$

$$V_t + rsV_s + \frac{s^2V_{ss}\sigma^2}{2} - rV = 0$$

$x = \log(S/K)$, $\frac{\partial x}{\partial s} = \frac{1}{s}$, $\frac{\partial^2 x}{\partial s^2} = \frac{-1}{s^2}$

$$V_{\sigma\sqrt{T-t}}\frac{\sigma}{2\sqrt{T-t}} + rsV_x\frac{\partial x}{\partial s} + \frac{s^2\sigma^2}{2}[V_{xx}(\frac{\partial x}{\partial s})^2 + V_x\frac{\partial^2 x}{\partial s^2}] - rV = 0$$

$$V_{\sigma\sqrt{T-t}}\frac{\sigma}{2\sqrt{T-t}} + rsV_x\frac{\partial x}{\partial s} + \frac{s^2\sigma^2}{2}[V_{xx}(\frac{\partial x}{\partial s})^2 + V_x\frac{\partial^2 x}{\partial s^2}] - rV = 0$$

$$-V_\sigma\sqrt{T-t}\frac{\sigma}{2\sqrt{T-t}} + (r - \frac{\sigma^2}{2})V_x + \frac{\sigma^2V_{xx}}{2} - rV = 0$$

$$\frac{-V_\sigma\sqrt{T-t}}{\sigma\sqrt{T-t}} + (\frac{r}{2\sigma^2} - 1)V_x + V_{xx} - \frac{r}{2\sigma^2}V = 0$$

$V_K = V_x\frac{\partial x}{\partial K} = -\frac{V_x}{K}, V_{KK} = \frac{V_{xx}}{K^2} + \frac{V_x}{K^2}$
Thus we require: $V_x \geq 0, V_{xx} \geq 0, V_\tau \geq 0$
$V(-\infty, \sqrt{\tau}) = 0, V(x, 0) = (x-1)^+$

$$x = \frac{\log(S/K)}{\sigma\sqrt{T-t}}, \frac{\partial}{\partial S} = \frac{1}{sy}, \frac{\partial}{t} = \frac{\log(S/K)\sigma^2}{2y}$$

SABR

We consider the SABR of [18]

$$dF_t = F_t^\beta \sigma_t dW_{1,t} \tag{27}$$

$$d\sigma_t = \xi\sigma_t dW_{2,t}, dW_{2,t} = \alpha\sigma_t[\rho dW_{1,t} + \sqrt{1-\rho^2}dW_{1,t}^\top] \tag{28}$$

Consider also, the normalised forward or moneyness process F_t/K, and the log-forward process

$$d(F_t/K) = \frac{\sigma_t}{K}F_t^\beta dW_{1,t} = (\sigma_t K^{\beta-1})(F_t/K)^\beta dW_{1,t}$$

$$d\log F_t = \frac{1}{F_t}dF_t + \frac{-1}{2F_t^2}(dF_t)^2 = \sigma_t F_{t-1}^{\beta-1}dW_{1,t} - \frac{\sigma_t^2 F_t^{2\beta-2}}{2}dt$$

$$dX_t = d\log F_t = \sigma_t \exp((\beta-1)X_t)dW_{1,t} - \frac{\sigma_t^2}{2}\exp(2(\beta-1)X_t)dt$$

In the case of SABR (and CEV), we have the following relationship for call prices, we have the relationship:

$$C(\lambda F_t, \lambda K, \lambda^{1-\beta}\sigma_t, \beta, \rho, \xi) = \lambda C(F_t, K, \sigma_t, \beta, \rho, \xi)$$

Thus we can fix one of F_t, K. However, it may be difficult for a neural network to learn the scaling relationship for the volatility input.

PDE: The corresponding SABR PDE is given by:

$$-V_\tau + \frac{\sigma^2 F^{2\beta}}{2} V_{FF} + \frac{\sigma^2 \xi F^\beta}{2} V_{\sigma\sigma} + \frac{\xi^2\sigma^2}{2} V_{F\sigma} = 0, V(F, 0) = (F - K)^+ \tag{29}$$

$$-V_\tau + V_y e^{2(\beta-1)y}\frac{\sigma^2}{2} + \frac{\sigma^2 \exp(2(\beta-1)y}{2} V_{yy} + \sigma^2 \exp((\beta-1)y) V_{y\sigma}, V(y, \tau = 0) = (e^y - 1) \tag{30}$$

$$-V_\tau + \frac{\sigma^2 (F/K)^{2\beta}}{2} V_{FF} + \frac{\sigma^2 \xi F^\beta}{2} V_{\sigma\sigma} + \frac{\xi^2\sigma^2}{2} V_{F\sigma} = 0, f(F/K, 0, \ldots) = (F/K - 1)^+ \tag{31}$$

$$\tag{32}$$

[**hagan**] provides an asymptotic expansion. However,

F_t	K	σ_t	τ	β	ξ	ρ
Forward	Strike	Volatility	Time-to-maturity	CEV	vol-of-vol	correlation
1	[0.5, 5]	[0, 0.2]	[0, 0.3]	[0, 1]	[0, 1]	[0, 0.7]

The asymptotic expansion degrades as σ, ξ, T increase, or for low strikes K (equivalently, high moneyness F/K or $\log(F/K)$)

We simulate a dataset

We generate implied volatilities from the Hagan approximation

$$z = \frac{\xi}{\sigma}(FK)^{(1-\beta)/2}$$
$$x(z) = \log\left((\sqrt{1 - 2\rho z + z^2} + z - \rho)/(1 - \rho)\right)$$

We can simulate the volatility process σ_t exactly, and the

$$V_T = V_t \exp((-0.5\alpha^2(T - t)) + \alpha(W_T - W_t)) \tag{33}$$
$$F_{j+1} - F_j = |V_j F_j^\beta \Delta W_j| \tag{34}$$

[**mcghee**]

Heston

SDE and MC:

$$
\begin{aligned}
dF_t &= F_t V_t dW_{1,t} \\
dV_t &= \kappa(\theta - V_t)dt + \xi\sqrt{V_t} dW_{2,t} \\
d[W_1, W_2]_t &= \rho dt
\end{aligned}
$$

We use the description of the characteristic transform from [14].

PDE:

Dataset:

F_t/K	V_t	τ	κ	θ	ξ	ρ
Moneyness	Variance	Time-to-maturity	Mean Reversion Speed	Mean Variance Level	Vol-of-Vol	Correlation

For the Heston, we also require the Feller Condition: $2\kappa\theta > \sigma^2$

Asian Option

We consider the case of a geometric asian average option and an arithmetic average asian option in the Black-Scholes setting.

For both cases we generate a dataset by sampling paths $(S_{j,i})$, where j denotes the

Geometric Average: We can obtain a closed-form expression for the geometric average asian option with fixed strike:

$$
\begin{aligned}
\frac{1}{N}\sum_{i=1}^{N}\log(S_i) &= \log(S_0) - \frac{1}{N}\sum_{i=1}^{N}\frac{\sigma^2}{4} i\Delta t + \frac{\sigma\Delta t}{N}\sum_{i=1}^{N} iZ_{N-i+1} && (35) \\
&= \log(S_0) - \frac{(N+1)\sigma^2\Delta t}{2} + \sqrt{\frac{(N+1)(2N+1)}{6N}}\sigma\Delta t Z && (36)
\end{aligned}
$$

$$
(\frac{1}{K}\exp(\frac{1}{N}\sum_{i=1}^{n}\log(S_i)) - 1)^+
$$

In the continuous time-limit we have, noting that $\int_0^T W_t dt = \int_0^T (T-t)dW_t$

$$
\frac{1}{T}\int_0^T \log(S_t)dt = \frac{1}{T}\int_0^T [\log(S_0) - \frac{\sigma^2}{2}t + \sigma W_t]dt = \log(S_0) - \frac{\sigma^2 t^2}{4} + \sigma W_T + \frac{\sigma T}{3}
$$

Discrete Time: Consider a discrete-time arithmetic average Asian option on a single underlying

$$
(\frac{1}{N}\sum_{i=1}^{n} S_i/K - 1)^+
$$

In the continuous time limit consider $Y_t = \frac{1}{K}\int_0^t S_t du, dY_t = S_t dt$

$$f(t,s,y) = \mathbb{E}[e^{-r(T-t}h(S_T,Y_T)|S_t = s, Y_t = y]$$

Then the corresponding PDE is given by:

$$f_t + rsf_s + sf_y + \frac{\sigma^2}{2}f_{ss} - rf = 0, f(T,s,y) = \frac{1}{T}(y/K-1)^+$$

The replicating strategy is f_s

Multidimensional Example - Arithmetic Brownian Motion / Bachelier Model

We consider the first example from [22], a standard European Call on a basket option where the underlyings have dynamics Arithmetic Brownian Motion.

SDE and MC: Let $\mathbf{S}_t$ be a d-dimensional Arithmetic Brownian motion driven by $\mathbf{W}_t$ a f-dimensional Brownian Motion over $t \in [0,T]$, with covariance $\mathbf{L}\mathbf{L}^\top dt, \mathbf{L} \in \mathbb{R}^{d\times f}$. For simplicity suppose that $\mathbf{S}_t$ is a $\mathbb{Q}$-local martingale. Suppose there is some weight vector $\mathbf{w} \in \mathbb{R}^d$ representing the index weights for a basket option, such that the value of the underlying basket is $\mathbf{w}^\top\mathbf{S}_t$. Then the payoff of the basket call option with payoff $h(\mathbf{w}^\top\mathbf{S}_t, K) = (\mathbf{w}^\top\mathbf{S}_t - K)^+$ is given by:

$$d\mathbf{S}_t = \mathbf{L}d\mathbf{W}_t, \mathbf{S}_t = \mathbf{S}_0 + \mathbf{L}\mathbf{W}_t, \mathbf{S}_t \sim N(\mathbf{S}_0, \mathbf{L}\mathbf{L}^\top t) \tag{37}$$

$$d\mathbf{X}_t = \mathbf{w}^\top d\mathbf{S}_t = \sigma dW_t^\top, \mathbf{X}_t = \mathbf{w}^\top\mathbf{S}_t, \mathbf{X}_t \sim N(\mathbf{w}^\top\mathbf{S}_0, \mathbf{w}^\top\mathbf{L}\mathbf{L}^\top\mathbf{w}) \tag{38}$$

$$h(\mathbf{w},\mathbf{S}_t,K) = (\mathbf{w}^\top\mathbf{S}_T - K)^+ = (\mathbf{X}_T - K)^+ \tag{39}$$

$$f(\tau,\mathbf{S}_t,K,\mathbf{w}) = \mathbb{E}[(\mathbf{w}^\top\mathbf{S}_t - K)^+|\mathbf{S}_t,\mathbf{w},\tau,K] \tag{40}$$

PDE: Applying Ito's lemma on $\mathbf{X}_t$ and $\mathbf{S}_t$, the corresponding PDE for the price of the basket call option is given by:

$$df(T-t,S_t;\ldots) = -\frac{\partial f}{\partial t}dt + \sum_{i=1}^{d}\frac{\partial f}{\partial S_i}dS_{i,t} + \sum_{i=1}^{d}\sum_{j=1}^{d}\frac{1}{2}\frac{\partial V}{\partial S_i S_j}d[S_i,S_j] \tag{41}$$

$$0 = -\frac{\partial f}{\partial \tau} + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\frac{\partial V}{\partial S_i S_j}, f(0,\mathbf{S}) = (\mathbf{w}^\top\mathbf{S} - K)^+ \tag{42}$$

$$df(T-t,X_t;\ldots) = -\frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}dX_t + \frac{1}{2}d[X]_t \tag{43}$$

$$= \mathbf{w}^\top\mathbf{L}d\mathbf{W}_t + \frac{1}{2}\mathbf{w}^\top\mathbf{L}\mathbf{L}^\top\mathbf{w}dt \tag{44}$$

$$= -\frac{\partial f}{\partial \tau} + \frac{\sigma^2}{2}\frac{\partial f}{\partial x^2}, f(0,\mathbf{X}) = (\mathbf{X} - K)^+ \tag{45}$$

Closed form price: The closed form price is given by the Bachelier normal formula:

For a gaussian random variable with variance a^2 and mean b, We can write:

$$\begin{aligned}
aZ + b - K = a(Z + \frac{b-K}{a}) \geq 0 &\implies Z \geq \frac{K-b}{a} \\
\mathbb{Q}[aZ + b \geq K] &= \Phi(\frac{K-b}{a}) \\
\mathbb{E}[Z \cdot 1_{aZ+b\geq K}] &= \int_{(K-b)/a}^{\infty} z\frac{e^{-z^2/2}}{\sqrt{2\pi}}dz = [\frac{-1}{\sqrt{2\pi}}e^{-z^2/2}]_{(K-b)/a}^{\infty} = \frac{\exp(-((K-b)/a)^2/2)}{\sqrt{2\pi}} \\
\mathbb{E}[(aZ+b-K)^+] &= a\phi(\frac{K-b}{a}) + (b-K)\Phi(\frac{K-b}{a}) \\
&= a\phi(\frac{b-K}{a}) + \frac{b-K}{a}\Phi((\frac{b-K}{a}))]
\end{aligned}$$

In this case, we have $a = \sigma\sqrt{\tau} = \sqrt{\mathbf{w}^\top \mathbf{L}\mathbf{L}^\top \mathbf{w}}\sqrt{\tau}, b = \mathbf{w}^\top \mathbf{S}_0$. Thus the closed-form price for the basket option is given by:

$$f(\mathbf{X}_0, \sigma, \tau, K) = \sigma\sqrt{\tau}[\phi(d_1) + d_1\Phi(d_1)], d_1 = \frac{\mathbf{X}_0 - K}{\sigma\sqrt{\tau}} \tag{46}$$

The analytic greeks are given by, noting that $\frac{\partial d_1}{\partial \tau} = -\frac{d_1}{2\tau}$ and $\frac{\partial d_1}{\partial x} = \frac{1}{\sigma\sqrt{\tau}}$

$$\frac{\partial f}{\partial x} = \sigma\sqrt{\tau}[-\frac{d_1}{\sigma\sqrt{\tau}}\phi(d_1) + \frac{1}{\sigma\sqrt{\tau}}\Phi(d_1) + \phi(d_1)\frac{d_1}{\sigma\sqrt{\tau}}] = \Phi(d_1) \tag{47}$$

$$\frac{\partial f}{\partial \tau} = \frac{f}{2\tau} + \sigma\sqrt{\tau}[\frac{d_1^2}{2\sqrt{\tau}}\phi(d_1) - \frac{d_1^2}{2\sqrt{\tau}}\phi(d_1) - \frac{d_1}{2\sqrt{\tau}}\Phi(d_1)] = \frac{\sigma}{2\sqrt{\tau}}\phi(d_1) \tag{48}$$

$$\frac{\partial f}{\partial x^2} = \frac{1}{\sigma\sqrt{\tau}}\phi(d_1) \tag{49}$$

Experiment: We consider a similar setup as the first code example [1] from [22]. time-to-maturity $\tau = T - t$, strike K, and the covariance $\mathbf{L}\mathbf{L}^\top$ are fixed, and the experiment is repeated with a different dimensionality d and number of sample paths.

In our setup, we consider the following parameter space. We simulate a random covariance matrix by taking the Cholesky decomposition of a matrix of standard normal random variables.

$\mathbf{S_0}$	$d = f$	K	τ	$\mathbf{L}\mathbf{L}^\top$
Underling	No. Assets	Strike	Time-to-maturity	Covariance
$\mathbf{1} + \sqrt{30/250}\mathbf{Z}$	$\{10\}$	{ 1 }	{ 1 }	$\{0.2^2\mathbf{Z}\mathbf{Z}^\top \mathbf{Z}_{ij} \sim N(0,1)\}$

[1] Retrieved from: https://github.com/differential-machine-learning/notebooks/blob/master/DifferentialMLTF2.i

We compare a standard feed-forward neural network trained on the sample payoffs only 19, and a neural network trained on the pathwise differentials 20, and standard Monte Carlo estimation. We also compute the analytic prices and greeks from equations 45-48.

We generate samples $i = 1, \ldots, N_{\text{samples}} = 10,000$ samples of $\mathbf{S_{i,0}}, \mathbf{W}_{i,0} \in \mathbb{R}^d$, simulating $\mathbf{S_{i,T}} = \mathbf{S}_{i,t} + \mathbf{L}\mathbf{W_{i,T}}$ exactly from 39. We obtain pathwise differentials $\frac{\partial h}{\partial \mathbf{S}_{i,T}}$ via automatic differentiation, although in this case the pathwise differentials are known analytically: $\mathbf{w}^\top 1_{\mathbf{X}_T \geq 1.0}$. Given the pathwise differentials are not differentiable, we cannot obtain a Hessian estimate from Monte Carlo with AAD.

We use the same neural network architecture and random seed initialisation for the weights for the feed-forward and differential neural network:

- **Inputs**: the values of the underlying $\mathbf{S}_0 \in \mathbb{R}^d$
- **Outputs**: predicted price for strike $K = 1$ $f(\mathbf{S}_0) \approx \mathbb{E}[(\mathbf{w}^\top \mathbf{S}_T - 1)^+ | \mathbf{S}_0]$
- **Architecture**: Standard feed-forward, 4 Hidden layers of 30 hidden units each, softplus activation in all hidden layers, linear output activation
- **Training:** Batch Size of 32, Learning Rate: 10^{-3}. For feed-forward only, Early Stopping with a validation set of 20%, and patience of 50 epochs.
- **Regularisation** : None.

In the neural network case, we do not have an estimate for $\frac{\partial g}{\partial \tau}$. Thus we only verify the pricing error, percentage of no-arbitrage bound violations, and error in the estimate of the sensitivity to the basket factor. We obtain the following results:

	ffn	**differential**	**MC**
pred, L^1	0.03751	0.01069	0.00173
pred, L^2	0.05742	0.01617	0.00174
pred, L^∞	0.55145	0.32498	0.00290
$\lvert\{i : f(X_i) \geq (X_i - K)^+\}\rvert / N_{\text{samples}}$	0.03460	0.02390	0.00000
grad, L^1	0.00326	0.00118	0.00003
grad, L^2	0.00355	0.00150	0.00003
grad, L^∞	0.01184	0.01096	0.00010
$\lvert\{i : \frac{\partial f(X_i)}{\partial X} > 0\}\rvert / N_{\text{samples}}$	0.06300	0.00000	0.00000
Total Time (s)	24.19039	44.79427	140.00

Table 2: Bachelier Basket example

In this case, it appears. Indeed, as per [22], the differential neural network achieves lower errors in both the price approximation, and gradients. At least for our particular choice of sample seed and the given neural network architecture, both neural network approaches underperform a MC estimate in all error measures and

Remark: The Bachelier Model has a positive homogeneous relationship. In particular:

$$f(x, \sigma\sqrt{\tau}) = \mathbb{E}[(x - K)^+] = K\mathbb{E}[(\frac{x}{K} - 1)^+] = Kf(\frac{x}{K}, \sigma\sqrt{\tau}/K)$$

Thus we could possibly eliminate dependency on strike. In addition, we can eliminate the dependence on time-to-maturity τ by noting that the σ and τ terms are grouped together as $\sigma\sqrt{\tau}$.

We note that if we consider the weighted underlying $w_i S_i$, we are effectively as a single variable, we are effectively able to price. However, in this setup we, the neural learns a pricing function the specific strike K, maturity τ, and covariance $\mathbf{L}\mathbf{L}^\top$.

Finally, it be more appropriate to model the basket as a univariate process.

Rough Bergomi

In the rough volatility setting, there is a lack of closed form expressions for prices, hence prices need to be approximated by Monte Carlo. [21] used a two step approach.

MC and SDE:

The Rough Bergomi model has dynamics:

$$1 \tag{50}$$

We use the code implementation for the Turbocharged Rough Bergomi scheme [2] from [29]

Experiment: Our parameter space consists of:

$\mathbf{S_0}$	$\log(K)$	τ	α	ρ	V_0	ξ
Underlying	Strike	Time-to-maturity	Hurst Exponent	correlation	Volatility	Vol-of-vol
$\{1\}$	$\{-0.5 + 0.1i : i = 0, \ldots, 10\}$	$\{\frac{i}{30} : i = 0, \ldots, 30\}$	$U[-0.4, 0.5]$	$U[-0.95, -0.1]$	$\{0.235\}$	$U[1.5, 2.5]$

Table 3: Rough Bergomi Parameter Space

We can only analyse error in pricing and greeks with respect to the MC estimates. However, we can analyse no-arbitrage violations for calls as before.

We consider two neural network architectures: a standard feed-forward **??**, a gated neural network **??**, and also compare against cubic spline regression.

- **Inputs**: $\mathbf{X} = (-\log(K), \tau, \alpha, \rho, \xi)$
- **Outputs**: Predicted call price $\approx \mathbb{E}[(\mathbf{S}_T - K)^+|\mathbf{X}]$

[2] Accessed from: https://github.com/ryanmccrickerd/rough_bergomi

- **Architecture**: *Feed-Forward*: 2 hidden layers with 100 hidden units each, hidden layers, linear output activation; *Gated*:
- **Training:** Batch Size of 32, Learning Rate: 10^{-3}. For feed-forward only, Early Stopping with a validation set of 20%, and patience of 10 epochs.
- **Regularisation** : None.

We sample $N_{\text{SPACE}} = 100$ random vectors of (α, ρ, ξ) from the distributions above. For each of these, we simulate S_T using [29], with a terminal maturity of $T = 1$ and $N_{\text{Brownian}} = 10000$ paths each, and compute the call prices for the collection of $\tau \times \log(K)$ described above. This produces a dataset of 96100 observations in 18 seconds.

We obtain the following results:

	ffn	gated	polynomial
pred, L^1	**0.00316**	0.00532	0.00447
pred, L^2	**0.00447**	0.00808	0.00601
pred, L^∞	**0.03383**	0.05344	0.04127
$g \geq (S-K)^+$	0.11413	**0.10864**	0.21770
$\frac{\partial g}{\partial K} > 0$	0.05237	**0.00000**	0.12903
[illegible]	0.24469	**0.00000**	0.00000

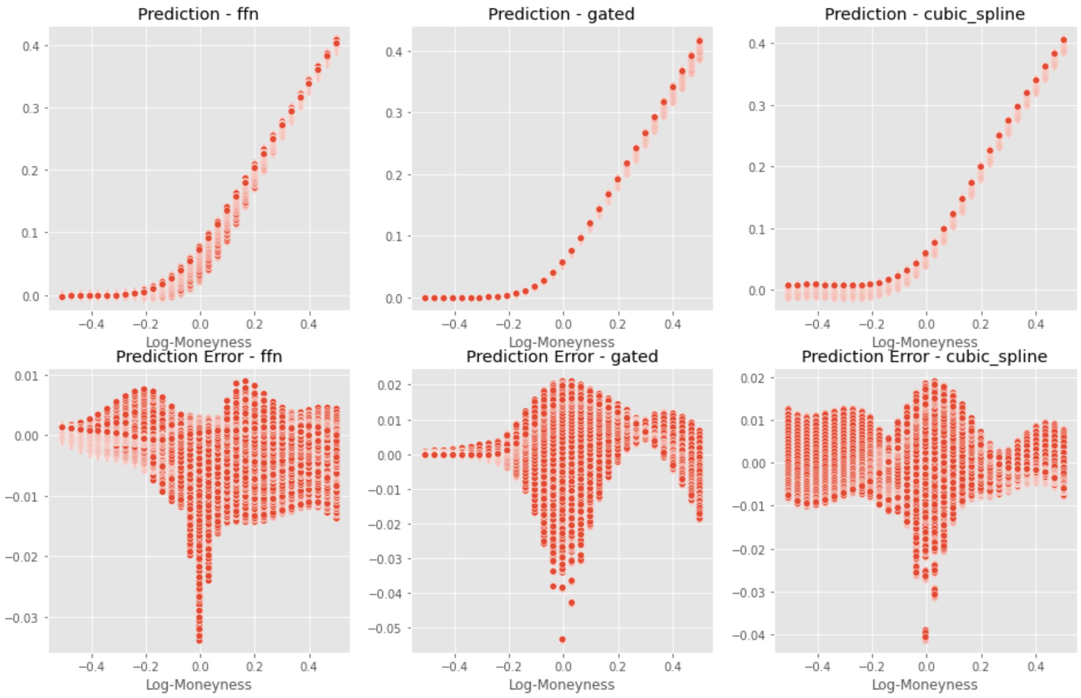

Figure 1: Errors for Rough Bergomi Approximations

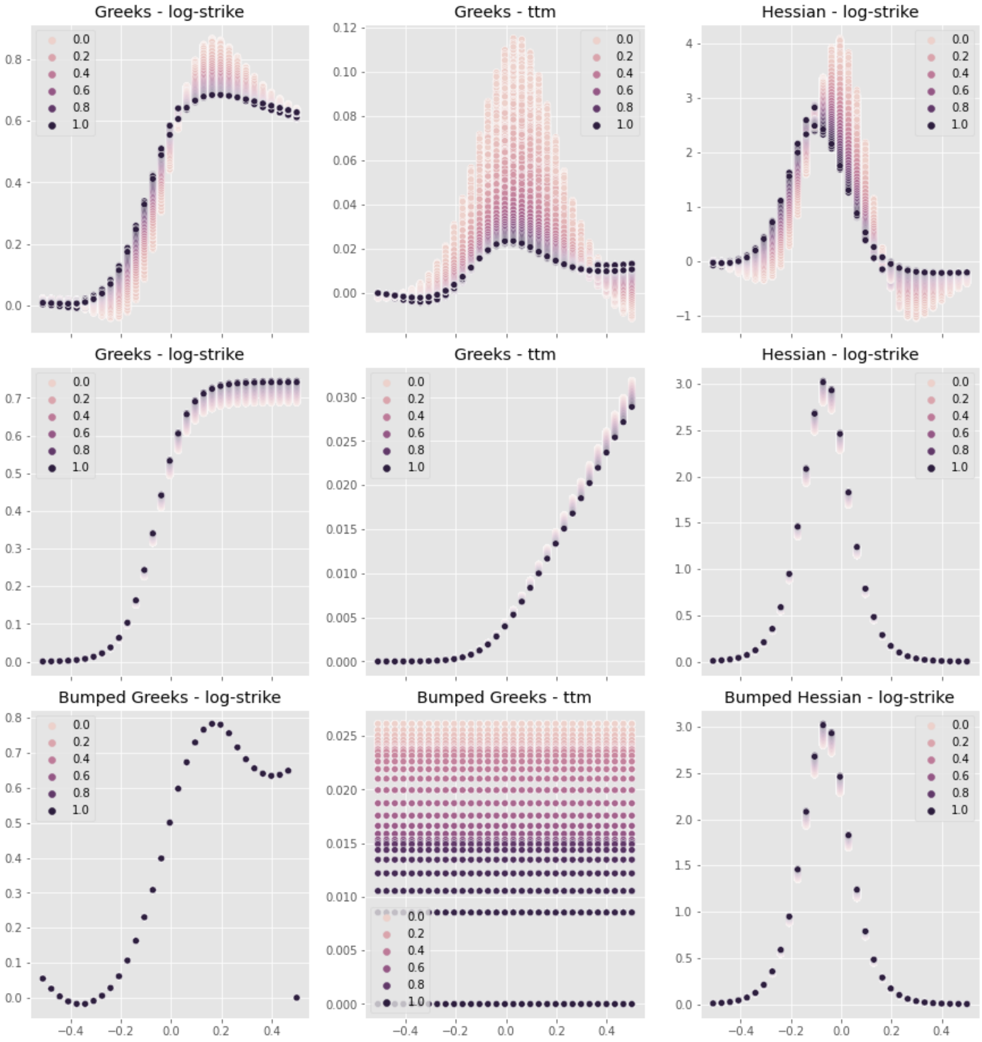

Figure 2: Greeks for Feed-foward, Gated, Cubic-spline respectively

The gated neural network is by construction able to avoid any no-arbitrage in terms of monotonicity in time to maturity in strike, and convexity in strike. The standard neural network however achieves the best result in terms of prediction error. Interestingly, standard cubic spline regression has similar performance to the unconstrained neural network, at a much shorter training time.

Remark: To verify that the neural network pricer has minimal dynamic arbitrage opportunities in this non-Markovian setting, one approach could be to evaluate whether the neural network pricer satisfies the corresponding path-dependent PDE (PPDE). Some literature towards this includes [25] and [34]

Up-and-in Barrier

We consider a barrier option with discrete barrier dates in the Black-Scholes setting

$$\mathbb{E}\left[S_T \prod_{i=1}^{\lfloor \tau/\Delta t \rfloor} 1_{S_{t+i\Delta t}>b} | \mathbf{X}\right]$$

In this case, we require the log-barrier to be greater than F_0/K, otherwise the barrier call is just a standard

$$-\frac{\partial V}{\partial \sigma\sqrt{\tau}} - \frac{\sigma^2\tau}{2}\frac{\partial V}{\partial x} + \frac{\sigma^2\tau}{2}\frac{\partial^2 V}{\partial x^2} V(0,x) = (e^x - 1)1_{x\geq \max\{\log(b/K),1\}} V(\tau, x) \quad = V^{BS}(\tau, x) \forall x >= \log(b \tag{51}$$

A problem in this case is that the payoff is discontinuous at the barrier, which makes computing gree

MC and SDE. However, in this case we must simulate entire sample trajectories as opposed to the prior. If we assume that the barrier dates are precisely the timesteps, then we can evaluate any general . For simplicity we consider the parameters: $\log(F_t/K), \sigma\sqrt{\tau}, \log(b/K)$, being the log-moneyness, time-scaled implied volatility and log-barrier.

Closed Form Solution:

Dataset:

Results:

Bermudans

TODO: Investigate this aspect further

Remark: A practical concern is that a single neural network can be trained to approximate only one specific exercise schedule.

Fixed Income Example

TODO: simulate rates curve: e.g. Vasciek, Cheyette, CIR, Hull White, LIBOR market model / BGM, HJM and analyse swaption approximation in this example, given the high-dimensional sensitivity to all swaps

Spread Payoff

TODO: A 2D example, possibly 2 SABR processes + dependence via copula / correlation matrix

6 Conclusion

Empirical observations thus far:

- Dataset quality and construction likely plays a role in training, convergence, and extrapolation. Cleaner (in terms of estimation error for MC and PDEs) and more data is better, but this increases training time.
- Differential training (whether through explicit greek labels, or the neural PDE term) seems to lead to better results empirically
- Hard constraints can be embedded into a neural network to prevent some no-arbitrage conditions, at the cost of some flexibility.
- Neural networks can be trained to solve complex payoffs and high-dimensional problems / market models, but need to be retrained on new market parameters or contract specifications. In some cases, it can be slower and less accurate than even Monte Carlo.
- Architecture: Hidden Units, Layers, Choice of Activation Function
- Regularisation: Batch Normalisation, Dropout, weight regularisation
- Learning rate, no, of epochs
- Dataset Construction: Parameter sample space

Future Directions: Further investigation into Neural PDE methods. Comparison against the tensor methods of [2], Potz, Glau, as no convenient `Python` implemenntation exists yet. Investigation into Quantum Machine Learning and potential applications for derivatives modelling. On the enginenering side, investigation of alternative frameworks, for example PyTorch, Jax in Python, and Flux in Julia.

Investigation of time-varying real world measure of dynamics, of how calibrated parameters evolves udner real world dynamics. Deep Hedging: Learning Risk-Neutral Implied Volatility Dynamics

References

[1] Alexandre Antonov, Michael Konikov, and Vladimir Piterbarg. “Neural networks with asymptotics control”. In: *Available at SSRN 3544698* (2020).

[2] Alexandre Antonov and Vladimir Piterbarg. “Alternatives to Deep Neural Networks for Function Approximations in Finance”. In: *Available at SSRN 3958331* (2021).

[3] Erik Alexander Aslaksen Jonasson. *Differential Deep Learning for Pricing Exotic Financial Derivatives*. 2021.

[4] Damiano Brigo et al. “Interpretability in deep learning for finance: a case study for the Heston model”. In: *Available at SSRN 3829947* (2021).

[5] Hans Buehler et al. “Deep hedging”. In: *Quantitative Finance* 19.8 (2019), pp. 1271–1291.

[6] Peter Carr and Dilip B Madan. “A note on sufficient conditions for no arbitrage”. In: *Finance Research Letters* 2.3 (2005), pp. 125–130.

[7] Marc Chataigner, Stéphane Crépey, and Matthew Dixon. “Deep local volatility”. In: *Risks* 8.3 (2020), p. 82.

[8] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[9] Samuel N Cohen, Derek Snow, and Lukasz Szpruch. “Black-box model risk in finance”. In: *Available at SSRN 3782412* (2021).

[10] Samuel N. Cohen, Christoph Reisinger, and Sheng Wang. “Detecting and Repairing Arbitrage in Traded Option Prices”. In: *Applied Mathematical Finance* 27.5 (Sept. 2020), pp. 345–373. DOI: 10.1080/1350486x.2020.1846573. URL: https://doi.org/10.1080%2F1350486x.2020.1846573.

[11] Matthew F Dixon, Igor Halperin, and Paul Bilokon. *Machine learning in Finance*. Vol. 1170. Springer, 2020.

[12] Zineb El Filali Ech-Chafiq, Pierre Henry-Labordere, and Jérôme Lelong. “Pricing Bermudan options using regression trees/random forests”. In: *arXiv preprint arXiv:2201.02587* (2021).

[13] Hans Föllmer and Alexander Schied. “Stochastic finance”. In: *Stochastic Finance*. de Gruyter, 2016.

[14] Jim Gatheral. *The volatility surface: a practitioner’s guide*. John Wiley & Sons, 2011.

[15] Gunnlaugur Geirsson. *Deep learning exotic derivatives*. 2021.

[16] Patryk Gierjatowicz et al. “Robust pricing and hedging via neural SDEs”. In: *Available at SSRN 3646241* (2020).

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[18] Patrick S Hagan et al. “Managing smile risk”. In: *The Best of Wilmott* 1 (2002), pp. 249–296.

[19] Horace He. “Making Deep Learning Go Brrrr From First Principles”. In: (2022). URL: `https://horace.io/brrr_intro.html`.

[20] Andres Hernandez. “Model calibration with neural networks”. In: *Available at SSRN 2812140* (2016).

[21] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. “Deep learning volatility”. In: *Available at SSRN 3322085* (2019).

[22] Brian Norsk Huge and Antoine Savine. “Differential machine learning”. In: *Available at SSRN 3591734* (2020).

[23] James M Hutchinson, Andrew W Lo, and Tomaso Poggio. “A nonparametric approach to pricing and hedging derivative securities via learning networks”. In: *The journal of Finance* 49.3 (1994), pp. 851–889.

[24] Andrey Itkin. “Deep learning calibration of option pricing models: some pitfalls and solutions”. In: *arXiv preprint arXiv:1906.03507* (2019).

[25] Antoine Jack Jacquier and Mugad Oumgari. “Deep PPDEs for rough local stochastic volatility”. In: *Available at SSRN 3400035* (2019).

[26] Joerg Kienitz, Nikolai Nowaczyk, and Nancy Qingxin Geng. “Dynamically Controlled Kernel Estimation”. In: *Available at SSRN 3829701* (2021).

[27] Tae-Kyoung Kim, Hyun-Gyoon Kim, and Jeonggyu Huh. “Large-scale online learning of implied volatilities”. In: *Expert Systems with Applications* 203 (2022), p. 117365.

[28] Francis A Longstaff and Eduardo S Schwartz. “Valuing American options by simulation: a simple least-squares approach”. In: *The review of financial studies* 14.1 (2001), pp. 113–147.

[29] Ryan McCrickerd and Mikko S Pakkanen. “Turbocharging Monte Carlo pricing for the rough Bergomi model”. In: *Quantitative Finance* 18.11 (2018), pp. 1877–1886.

[30] Remco van der Meer, Cornelis W Oosterlee, and Anastasia Borovykh. “Optimally weighted loss functions for solving pdes with neural networks”. In: *Journal of Computational and Applied Mathematics* 405 (2022), p. 113887.

[31] Christoph Molnar. *Interpretable machine learning.* Lulu. com, 2020.

[32] Antal Ratku and Dirk Neumann. “Derivatives of feed-forward neural networks and their application in real-time market risk management”. In: *OR Spectrum* (2022), pp. 1–19.

[33] Johannes Ruf and Weiguan Wang. “Neural networks for option pricing and hedging: a literature review”. In: *arXiv preprint arXiv:1911.05620* (2019).

[34] Marc Sabate-Vidales, David Šiška, and Lukasz Szpruch. “Solving path dependent PDEs with LSTM networks and path signatures”. In: *arXiv preprint arXiv:2011.10630* (2020).

[35] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[36] Valentin Tissot-Daguette. *Projection of Functionals and Fast Pricing of Exotic Options*. 2021. DOI: 10.48550/ARXIV.2111.03713. URL: https://arxiv.org/abs/2111.03713.

[37] Zhe Wang, Nicolas Privault, and Claude Guet. “Deep self-consistent learning of local volatility”. In: *Available at SSRN 3989177* (2021).

[38] Yongxin Yang, Yu Zheng, and Timothy Hospedales. “Gated neural networks for option pricing: Rationality by design”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.