TU Delft

Delft University of Technology

Document Version
Final published version

Licence
CC BY

Citation (APA)
Della Corte, S., Van Mieghem, L., Papapantoleon, A., & Papazoglou-Hennig, J. (2026). Machine learning for option pricing: an empirical investigation of network architectures. *European Journal of Finance*. https://doi.org/10.1080/1351847X.2026.2667910

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.



The European Journal of Finance

ISSN: 1351-847X (Print) 1466-4364 (Online) Journal homepage: www.tandfonline.com/journals/rejf20

Machine learning for option pricing: an empirical investigation of network architectures

Serena Della Corte, Laurens Van Mieghem, Antonis Papapantoleon & Jonas Papazoglou-Hennig

To cite this article: Serena Della Corte, Laurens Van Mieghem, Antonis Papapantoleon & Jonas Papazoglou-Hennig (28 May 2026): Machine learning for option pricing: an empirical investigation of network architectures, The European Journal of Finance, DOI: 10.1080/1351847X.2026.2667910

To link to this article: https://doi.org/10.1080/1351847X.2026.2667910

© 2026 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.

Published online: 28 May 2026.

Submit your article to this journal

Article views: 143

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=rejf20

THE EUROPEAN JOURNAL OF FINANCE
https://doi.org/10.1080/1351847X.2026.2667910

Routledge
Taylor & Francis Group

OPEN ACCESS Check for updates

Machine learning for option pricing: an empirical investigation of network architectures

Serena Della Corte[a], Laurens Van Mieghem[a], Antonis Papapantoleon[a,b] and Jonas Papazoglou-Hennig[c]

[a]Delft Institute of Applied Mathematics, TU Delft, Delft, The Netherlands; [b]Institute of Applied and Computational Mathematics, FORTH, Heraklion, Greece; [c]Department of Mathematics, TU Munich, Garching, Germany

ABSTRACT

We consider the supervised learning problem of learning the price of an option or the implied volatility given appropriate input data (model parameters) and corresponding output data (option prices or implied volatilities). The majority of articles in this literature considers a (plain) feedforward neural network architecture in order to connect the neurons used for learning the function mapping inputs to outputs. In this article, motivated by methods in image classification and recent advances in machine learning methods for PDEs, we investigate empirically whether and how the choice of network architecture affects the accuracy and training time of a machine learning algorithm. We find that highway-type network architectures achieve the best performance, when considering the mean squared error and the training time as criteria, within the considered parameter budgets for the Black–Scholes and Heston option pricing problems. Considering the transformed implied volatility problem, a simplified DGM variant achieves the lowest error among the tested architectures. We also carry out a capacity-normalized comparison for completeness, where all architectures are evaluated with an equal number of parameters. Finally, for the implied volatility problem, we additionally include experiments using real market data.

ARTICLE HISTORY
Received 22 January 2025
Accepted 27 April 2026

KEYWORDS
Option pricing; implied volatility; supervised learning; residual networks; highway networks; DGM networks

2020 MATHEMATICS SUBJECT CLASSIFICATIONS
91G20; 91G60; 68T07

1. Introduction

Machine learning has taken the field of mathematical finance by a storm, and there are numerous applications of machine learning in finance by now. Concrete applications include, for example, the computation of option prices and implied volatilities as well as the calibration of financial models (Horvath, Muguruza, and Tomas 2021; Liu, Oosterlee, and Bohte 2019), approaches to hedging (Buehler et al. 2019), portfolio selection and optimization (Zhang, Zohren, and Roberts 2022), risk management (Boudabsa and Filipović 2022), optimal stopping problems (Becker, Cheridito, and Jentzen 2019), model-free and robust finance (Eckstein and Kupper 2021), stochastic games and optimal control problems (Huré et al. 2021), as well as the solution of high-dimensional PDEs (Sirignano and Spiliopoulos 2018). A comprehensive overview of machine learning applications in mathematical finance appears in the recent volume of Capponi and Lehalle (2023), while an exhaustive overview focussing on pricing and hedging appears in Ruf and Wang (2020).

We are interested in the computation of option prices and implied volatilities using machine learning methods, and thus, implicitly, in model calibration as well. More specifically, we consider the supervised learning problem of learning the price of an option or the implied volatility given appropriate input data (model parameters) and corresponding output data (option prices or implied volatilities). The majority of articles in this literature, see e.g. Horvath, Muguruza, and Tomas (2021), Cuchiero, Khosrawi, and Teichmann (2020) and Liu, Oosterlee, and Bohte (2019), consider a (plain) feedforward neural network architecture in order to connect

CONTACT Antonis Papapantoleon a.papapantoleon@tudelft.nl

© 2026 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. The terms on which this article has been published allow the posting of the Accepted Manuscript in a repository by the author(s) or with their consent.

the neurons used for learning the function mapping inputs to outputs. In this article, motivated by methods in image classification, see e.g. He et al. (2015), Srivastava, Greff, and Schmidhuber (2015) and He et al. (2016), and recent advances in machine learning methods for PDEs, see e.g. Sirignano and Spiliopoulos (2018), we investigate empirically whether and how the choice of network architecture affects the speed of a machine learning algorithm, and we are interested in the "optimal" network architecture for these problems.

More specifically, next to the classical feedforward neural network or multilayer perceptron (MLP) architecture, we consider residual neural networks, highway networks and generalized highway networks. These network architectures have been successfully applied in image classification problems, see e.g. He et al. (2016). Moreover, motivated by the recent work of Sirignano and Spiliopoulos (2018) on the deep Galerkin method (DGM) for solving high-dimensional PDEs, we also consider the DGM network architecture, and, in addition, we construct and test two variants of this architecture.

The empirical results show that the more advanced network architectures consistently outperform the MLP architecture, and should thus be preferred for the computation of option prices and implied volatilities. More specifically, we find that for option pricing problems, where we focus on the Black–Scholes and the Heston model, highway-type network architectures outperform all other variants, when considering the mean squared error and the computational time as criteria. Moreover, we find that for the computation of the implied volatility, after a necessary transformation, a variant of the DGM architecture outperforms all other variants, when considering again the mean squared error and the computational time as criteria. See also Van Mieghem (2022).

The remainder of this paper is organized as follows: in Section 2, we briefly introduce the models used in this article, and in Section 3, we outline the basics of neural networks. In Section 4, we describe the different network architectures that will be used throughout, namely residual networks, highway and generalized highway networks, as well as the DGM network and its two variants. In Section 5, we revisit the main problems and describe the method used for generating sample data. In Section 6, we present the results of the empirical analysis performed, where the different network architectures are compared, with main criteria being accuracy and computational time. Moreover, we perform a capacity-normalized comparison in which all architectures are evaluated with an equal number of parameters, and also apply our results to real market data. Finally, Section 7 contains a synopsis and some concluding remarks from the empirical analysis.

2. Models

In this brief section, we are going to review the models we will use for option pricing and recall the definition of implied volatility, in order to fix the notation for the remainder of this work.

Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{Q})$ denote a complete stochastic basis, where $\mathbb{F} = (\mathcal{F}_t)_{t\in[0,T]}$ is the filtration, $T > 0$ is a fixed and finite time horizon, while $\mathbb{Q}$ denotes an equivalent martingale measure for an asset S with price process $S = (S_t)_{t\in[0,T]}$, adapted to the filtration $\mathbb{F}$.

We first consider the Samuelson / Black and Scholes (1973) model for the evolution of the asset price S, i.e.

$$\mathrm{d}S_t = rS_t\mathrm{d}t + \sigma S_t \mathrm{d}W_t, \quad S_0 > 0, \tag{1}$$

where r denotes the risk-free interest rate, σ the volatility and W denotes a $\mathbb{Q}$-Brownian motion. The price of a European call option with payoff $C = (S_T - K)^+$ in this model is provided by the celebrated Black–Scholes formula, i.e.

$$\begin{aligned} \pi_t(C; K, T, \sigma) &= S_t\Phi(d_1) - \mathrm{e}^{-r(T-t)}K\Phi(d_2), \\ d_1 &= \frac{\log\left(\frac{S_t}{K}\right) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} = d_2 + \sigma\sqrt{T-t}, \end{aligned} \tag{2}$$

where $\Phi(\cdot)$ denotes the cumulative distribution function (cdf) of the standard normal distribution.

Assume that we can observe in a financial market the traded prices $\pi_t^{\mathrm{M}}(C; K, T)$ for a call option C, for various strikes K and maturities T. The implied volatility $\sigma_{\mathrm{iv}}(K, T)$ is the volatility that should be inserted



Figure 1. The market implied volatility for CBOE's SPX options.

into the Black–Scholes Equation (2) such that the model price $\pi_t(C;K,T,\sigma)$ matches the traded market price $\pi_t^{\mathrm{M}}(C;K,T)$, i.e.

$$\pi_t(C;K,T,\sigma_{\mathrm{iv}}(K,T)) = \pi_t^{\mathrm{M}}(C;K,T), \tag{3}$$

see e.g. Gatheral (2006, Ch. 1). The inverse problem in (3) does not admit a closed form solution and needs to be solved numerically, using methods such as the Newton–Raphson or Brent; see e.g. Liu, Oosterlee, and Bohte (2019) for more details. The Black–Scholes model assumes that the volatility is constant; however the implied volatility computed from real market option prices exhibits a so-called smile or skew shape; see e.g. Figure 1 for a visualization.

A popular model that captures the behaviour of market data is the Heston stochastic volatility model, *cf.* Heston (1993). The dynamics of the asset price in this model are provided by the following SDEs

$$\begin{cases} \mathrm{d}S_t = rS_t\mathrm{d}t + \sqrt{v_t}S_t\mathrm{d}W_t, & S_0 > 0 \\ \mathrm{d}v_t = \kappa(\theta - v_t)\mathrm{d}t + \eta\sqrt{v_t}\mathrm{d}\widetilde{W}_t, & v_0 > 0, \end{cases} \tag{4}$$

where κ, θ, η denote positive constants and $[W, \widetilde{W}] = \rho$ denotes the correlation between the two $\mathbb{Q}$-Brownian motions. The tuple of the log-asset price process and the volatility process $(\log S, v)$ is an affine process, see e.g. Filipović (2009, Ch. 10), and the characteristic function of the log-asset price can be computed explicitly. Therefore, option prices in the Heston model can be computed efficiently using Fourier transform methods, see e.g. Eberlein, Glau, and Papapantoleon (2010), or the COS method of Fang and Oosterlee (2008).

3. Fundamentals of neural networks

In this section, we briefly recall the basic building blocks of feedforward neural networks, fix notation, and introduce the multilayer perceptron (MLP) architecture used as a reference throughout the paper.

A (single) *perceptron* is a function that maps an input vector $\mathbf{x} \in \mathbb{R}^d$ to an output $\mathbf{y} \in \mathbb{R}$ of the form

$$\mathbf{y} = H\left(\sum_{i=1}^{d} w_i x_i + b\right) = H\left(\mathbf{w}\cdot\mathbf{x} + b\right), \tag{5}$$

where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector, $b \in \mathbb{R}$ a bias term, and H is a nonlinear activation function. Here $\cdot$ denotes the standard inner product.

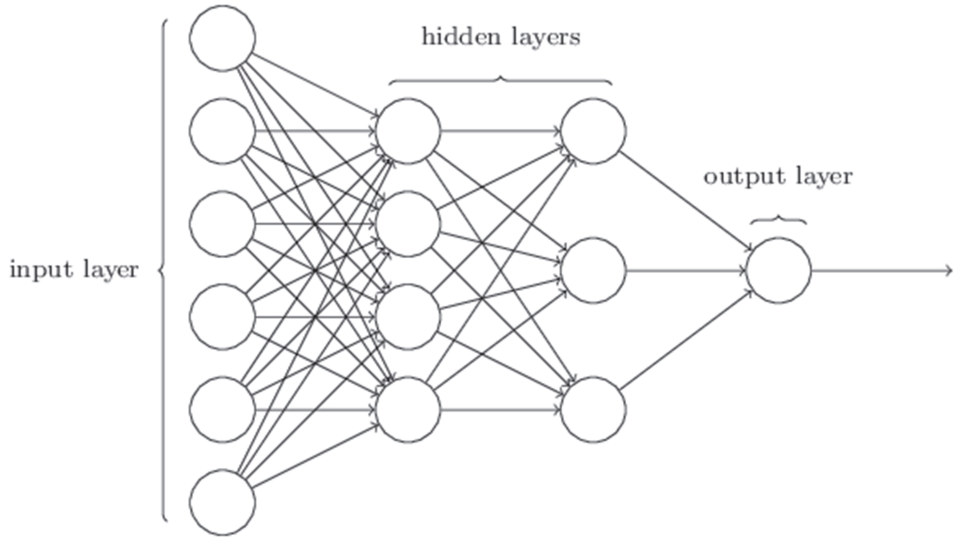


Figure 2. A visual representation of a multilayer perceptron (MLP).

The activation function H is typically nonlinear (to allow the network to represent nonlinear maps) and differentiable (to enable gradient-based training). Common examples include:

- the sigmoid function, $H_{\text{sig}}(x) = \frac{1}{1+\mathrm{e}^{-x}}$,
- the hyperbolic tangent, $H_{\tanh}(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}}$,
- the Rectified Linear Unit (ReLU), $H_{\text{ReLU}}(x) = \max\{0, x\}$,
- the Gaussian Error Linear Unit (GELU), $H_{\text{GELU}}(x) = x\,\Phi(x)$, with Φ the standard normal cdf,
- the softmax function for classification, $H_{\text{sm}}(\mathbf{x})_i = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{K} \mathrm{e}^{x_j}}$.

A *layer* is a collection of perceptrons that share the same input. Let a layer have n units, and collect their outputs into a vector $\mathbf{y} \in \mathbb{R}^n$. Using a weight matrix $W \in \mathbb{R}^{n\times d}$ and bias vector $\mathbf{b} \in \mathbb{R}^n$, we can write

$$y_i = H_i\left(\sum_{j=1}^{d} W_{ij}x_j + b_i\right), \tag{6}$$

where H_i is the activation of unit i. In most applications, the same activation function is used throughout a layer, hence $H_i \equiv H$ for all i.

The simplest example of a feedforward neural network is the *multilayer perceptron* (MLP), where layers are stacked and fully connected: the output of one layer serves as the input to the next; see Figure 2.

Consider a target output $\mathbf{y}$ and a neural network $H(\mathbf{x};\theta)$ with parameters θ (all weights and biases). We measure the discrepancy by a loss function J, e.g. the mean squared error,

$$J(\theta) := \|H(\mathbf{x};\theta) - \mathbf{y}\|^2. \tag{7}$$

The parameters are learned by (stochastic) gradient descent,

$$\theta_{k+1} := \theta_k - \alpha\,\nabla_\theta J(\theta_k), \tag{8}$$

with learning rate $\alpha > 0$. In practice, the training data are partitioned into mini-batches; the update (8) is applied on each batch, and one full pass over all batches is referred to as an *epoch*. We initialize all networks using either the Glorot and Bengio (2010) or the He et al. (2015) schemes.

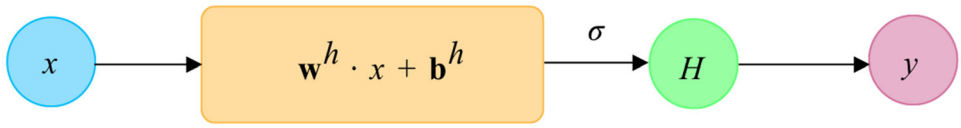


Figure 3. Schematic representation of a single MLP layer, *cf.* (9).

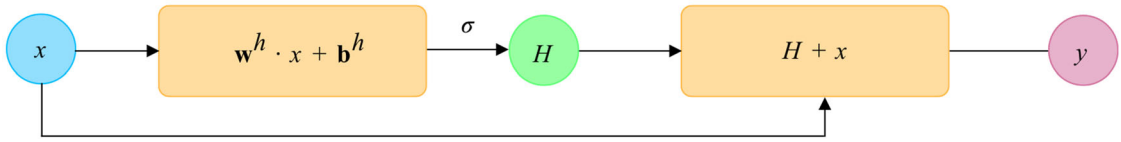


Figure 4. Schematic representation of a single residual layer, *cf.* (10).

4. Network variations

Let us now describe the network architectures used in our experiments. All of them are based on the fully-connected MLP structure from Section 3, and differ only on how each layer transforms its input, and how information is propagated across layers. Throughout this section, we follow the notation of Srivastava, Greff, and Schmidhuber (2015).

A standard MLP layer applies an affine map followed by a nonlinearity,

$$\mathbf{y} = H(W_H\mathbf{x} + \mathbf{b}_H) =: H(\mathbf{x}, W_H), \tag{9}$$

where W_H and $\mathbf{b}_H$ denote the layer-specific weights and biases. A schematic representation is shown in Figure 3; the symbol σ is used to indicate a generic nonlinear activation function.

4.1. Residual network

Residual networks (ResNets; He et al. 2016) augment the MLP layer with an identity skip connection, so that each layer computes

$$\mathbf{y} = H(\mathbf{x}, W_H) + \mathbf{x}. \tag{10}$$

A schematic view appears in Figure 4. The additional skip connection allows information (and gradients) to propagate directly across layers and has been shown to improve trainability in deep architectures.

4.2. Highway network

Highway networks are a generalization of the residual network architecture from the previous subsection, by introducing an additional non-linearity $T(\mathbf{x}, W_T)$, called the transform gate. This non-linearity will determine the amount of information flowing from the non-linearity H inside the layer and the amount of information 'carried' over from the input vector $\mathbf{x}$. The outcome is thus the following transformation

$$\mathbf{y} = H(\mathbf{x}, W_H) \odot T(\mathbf{x}, W_T) + \mathbf{x} \odot [1 - T(\mathbf{x}, W_T)], \tag{11}$$

where $\odot$ denotes the Hadamard product between vectors or matrices. Hence, T creates a convex combination between the input $\mathbf{x}$ and the transformed input $H(\mathbf{x}, W_H)$ before the output. A visual representation of this layer is given in Figure 5.

In the approximation of complex functions using neural networks, the depth of the network is proven to be a significant factor in its success, see e.g. Telgarsky (2016). However, training deeper networks presents additional bottlenecks, such as the vanishing gradients problem. The intuition behind highway networks is to allow

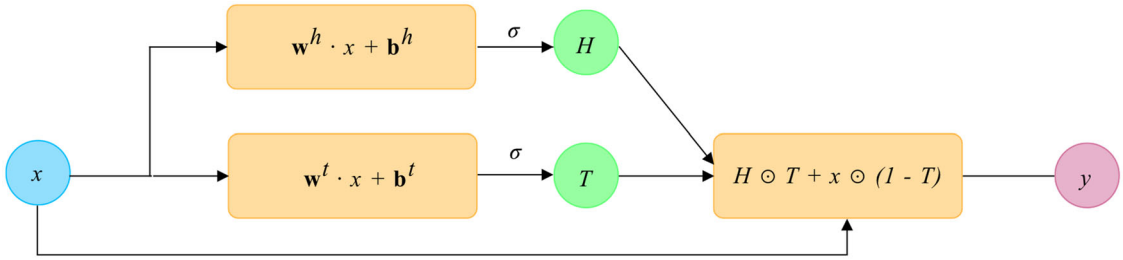


Figure 5. Schematic representation of a single highway layer, *cf.* (11).

for unimpeded information flow across layers; see Srivastava, Greff, and Schmidhuber (2015). Indeed, from Equation (11) we can see that for the boundary values of T we have

$$\mathbf{y} = \begin{cases} \mathbf{x}, & T(\mathbf{x}, W_T) = \mathbf{0} \\ H(\mathbf{x}, W_H), & T(\mathbf{x}, W_T) = \mathbf{1}, \end{cases} \tag{12}$$

therefore

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}} = \begin{cases} I, & T(\mathbf{x}, W_T) = \mathbf{0} \\ H'(\mathbf{x}, W_H), & T(\mathbf{x}, W_T) = \mathbf{1}, \end{cases} \tag{13}$$

where I denotes the identity matrix. Thus, the transform gate T allows the highway layer to vary its behaviour between a standard MLP layer and an identity mapping, leaving $\mathbf{x}$ unaffected.

4.3. Generalized highway network

One can further generalize highway networks by removing the convexity requirement in the outcome. Let us define an additional non-linear transformation alongside $T(\mathbf{x}, W_T)$, which we denote by $C(\mathbf{x}, W_C)$, such that the outcome takes the form

$$\mathbf{y} = H(\mathbf{x}, W_H) \odot T(\mathbf{x}, W_T) + \mathbf{x} \odot C(\mathbf{x}, W_C). \tag{14}$$

We call T and C the transform and carry gate, respectively, as they regulate the amount of information passed into the next layer from the transformed and original input. This gating mechanism allows information to flow along the layers of the network without degradation. Highway networks are obviously a special case of generalized highway networks, once we set $C = 1 - T$. A visual representation of this layer is given in Figure 6.

4.4. DGM network

The DGM network architecture was proposed in the context of the Deep Galerkin Method for the solution of PDEs; see Sirignano and Spiliopoulos (2018) and also Al-Aradi et al. (2018). This layer structure is related to generalized highway networks but incorporates additional nonlinear blocks and an explicit dependence on the original input $\mathbf{x}$ at each layer.

4.4.1. DGM layer

A DGM layer takes as input the feature vector $\mathbf{x}$ and the output S^{old} of the previous layer (or an initial affine transform of $\mathbf{x}$). Using these, it computes four intermediate vectors Z, G, R, H and produces the new state S^{new}

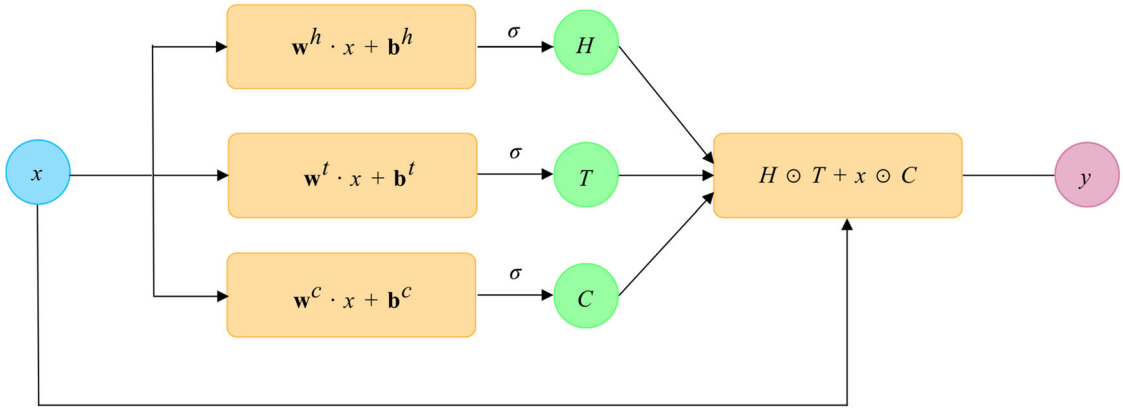


Figure 6. Schematic representation of a single generalized highway layer, *cf.* (14).

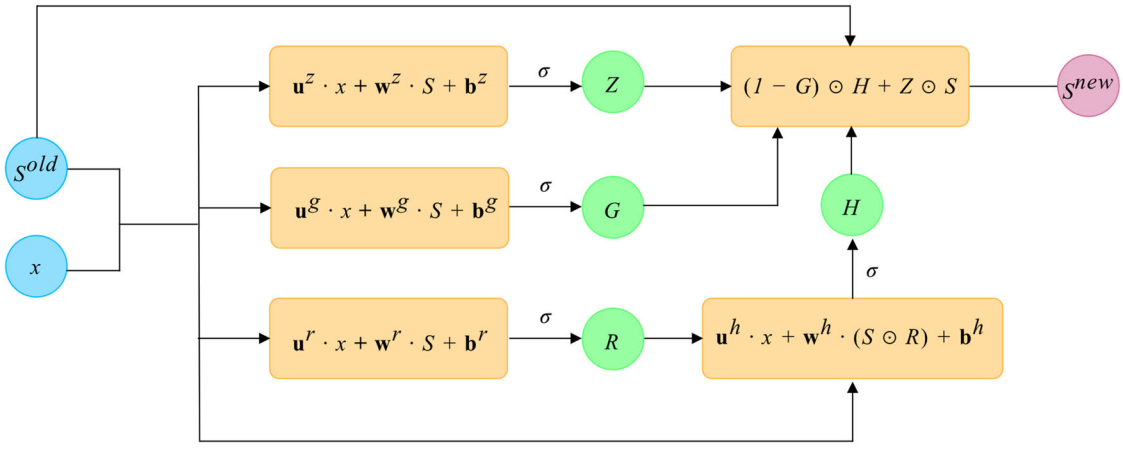


Figure 7. Schematic representation of a single DGM layer. Adopted from Al-Aradi et al. (2018).

via

$$S^{\text{new}} = (1 - G) \odot H + Z \odot S, \tag{15}$$

where S denotes the incoming state. Setting $T = 1-G$ and $C = Z$ shows that this is formally analogous to a generalized highway layer,

$$S^{\text{new}} = T \odot H + C \odot S,$$

but with a more elaborate internal computation of H and recurrent use of the original input $\mathbf{x}$; see Figure 7.

Compared to a highway layer, a DGM layer is more complex in two respects:

(i) The vector H is obtained by composing multiple nonlinear transformations, which can be interpreted as a small "subnetwork" within the layer.
(ii) The original feature vector $\mathbf{x}$ enters each nonlinear block, creating a recurrent-like structure reminiscent of gated recurrent units; see, e.g. Sherstinsky (2020).

These two aspects will be further explored in the next subsections, by introducing and testing new architectures that stress these points. As a consequence, a DGM layer contains substantially more parameters than an

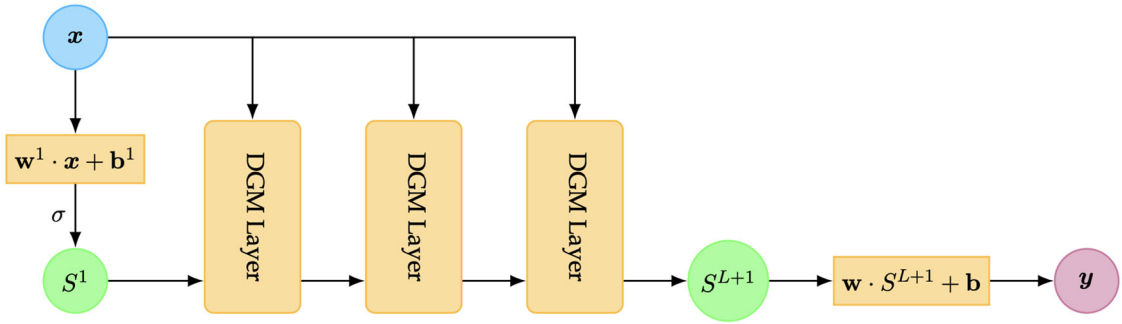


Figure 8. High-level overview of the DGM network. Each block labelled "DGM layer" refers to the operations in Figure 7. Adopted from Al-Aradi et al. (2018).

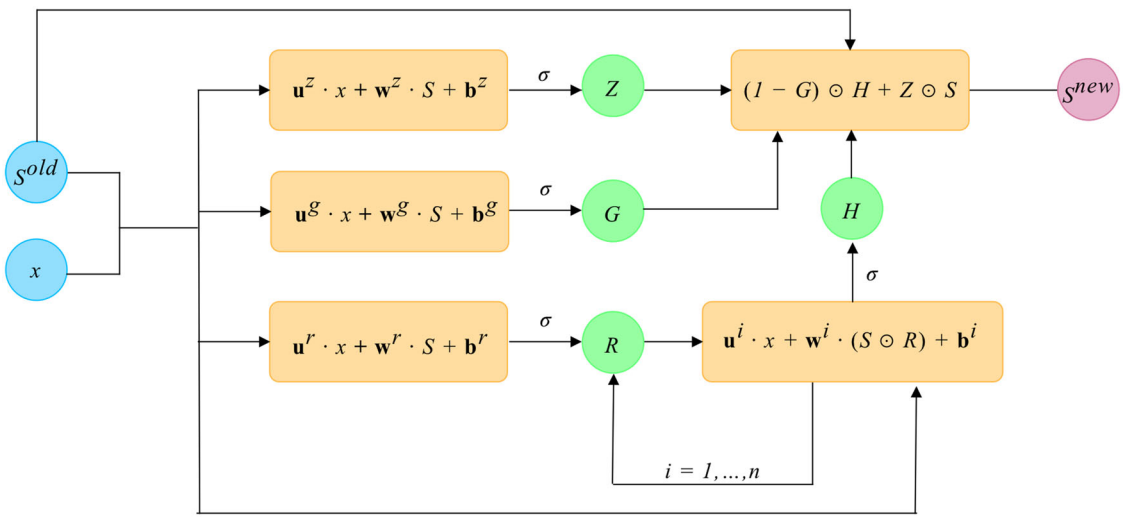


Figure 9. Schematic representation of a deep DGM layer with n internal nonlinear transformations.

MLP or a highway layer and incurs higher computational cost, but offers increased flexibility for approximating functions with sharp local features.

4.4.2. Overall DGM architecture

Since each DGM layer requires both $\mathbf{x}$ and a hidden state S as input, we first compute an initial state S^1 via a dense layer,

$$S^1 = \mathbf{w}^1 \cdot \mathbf{x} + \mathbf{b}^1, \tag{16}$$

which sets the dimension of the subsequent DGM layers. An output layer then maps the final state to the desired prediction. The overall architecture is summarized in Figure 8.

4.5. Deep DGM network

The *deep DGM* variant increases the depth of the internal subnetwork used to compute H, by stacking n nonlinear transformations; see Figure 9. This adds additional weight matrices and biases inside each DGM layer and allows us to study the effect of deeper intra-layer nonlinearities on approximation quality.

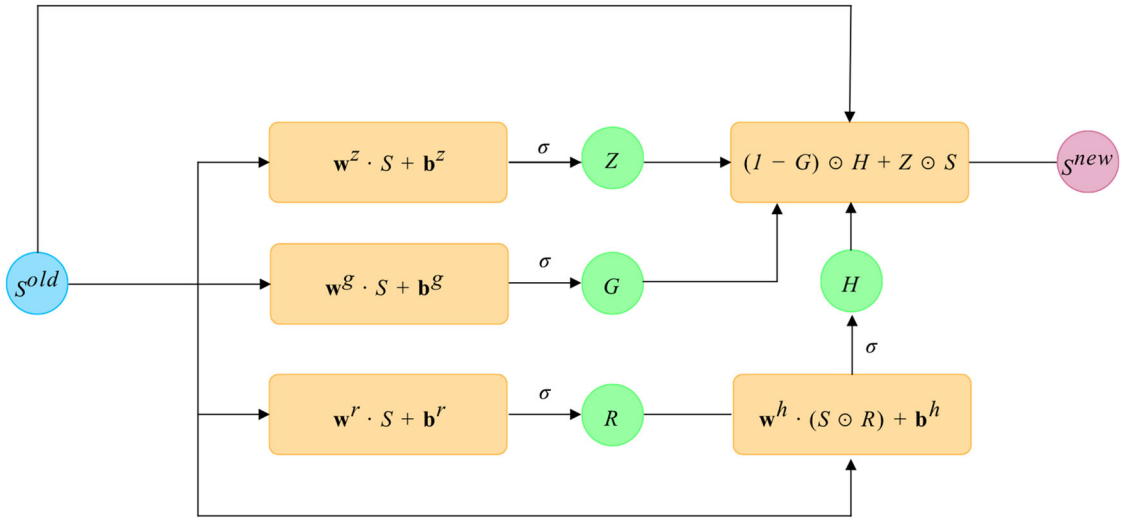


Figure 10. Schematic representation of a single no-recurrence DGM layer.

4.6. No-recurrence DGM network

Finally, we consider a simplified *no-recurrence* DGM network in which the dependence on $\mathbf{x}$ is removed from all layers except the first. The resulting layer structure, shown in Figure 10, closely resembles a highway layer but retains the more complex computation of H:

$$H = \sigma\left(\mathbf{w}^h \cdot \left[S \cdot \sigma\left(\mathbf{w}^r \cdot S + \mathbf{b}^r\right)\right] + \mathbf{b}^h\right). \tag{17}$$

This reduces the parameter count and computational cost per layer, while preserving some of the additional flexibility of the original DGM construction.

5. Problems revisited and data sampling

In this section, we take another look at the problems considered in this work and present the form that will be used in the empirical analysis. We also discuss the methods for generating data and the parameter sets used for training the neural networks.

5.1. Black–Scholes and Heston models

The first problem we consider is the pricing of European call options in the Black–Scholes model. Let us define the following variables, from which we will sample later:

- $m = \frac{S}{K}$: the moneyness value;
- σ: the volatility of the underlying asset;
- r: the risk-free rate;
- $\tau = T - t$: a time transformation to obtain the time to maturity, in years.

Using these variables, we can calculate the price of a European call option using a transformed version of (2):

$$\pi_t(C; m, \tau, \sigma) = \frac{\pi_t(C; K, T, \sigma)}{K} = m\Phi(d_1) - \mathrm{e}^{-r\tau}\Phi(d_2),$$
$$d_1 = \frac{\log(m) + (r + \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}} = d_2 + \sigma\tau. \tag{18}$$

When we train neural networks to compute call option prices in the Black–Scholes model, we will use the solution presented in (18).

In addition to the Black–Scholes pricing problem, we also train neural networks to compute the price of European call options in the Heston stochastic volatility model (4). The prices of call options in the Heston model are computed using the COS method and utilizing the affine structure of the characteristic function. We use this problem in order to investigate whether the various networks show consistent performance across different option pricing problems or not.

5.2. Implied volatility

The next problem we consider is the training and evaluation of neural networks for the computation of the implied volatility, thus continuing the analysis of Liu, Oosterlee, and Bohte (2019), in which only MLPs are trained and evaluated. In addition to the standard dataset for the computation of the implied volatility, in which we use the parameters directly from the inverted Black–Scholes pricing problem (Table 3), we also apply a transformation on the dataset from the scaled call price (π/K) to the scaled time value of the option, already introduced in Liu, Oosterlee, and Bohte (2019). The reasoning behind this transformation is the following: We train the networks to find the solution to Equation (3). However, we know that the option's vega, i.e. its derivative with respect to the volatility, can become arbitrarily small for deep in-the-money (ITM) or deep out-of-the-money (OTM) options. In the context of neural networks, where we take the derivatives to compute the gradient of the network, the instability of vega may lead to large gradients, possibly causing significant estimation errors.

Let us briefly elaborate further on this issue. We denote $\pi =: f(\sigma)$ for convenience, where f is the analytical function representing the price of European call option in the Black–Scholes model, depending on σ. The implied volatility problem concentrates on approximating the inverse of this problem, i.e. $\sigma = f^{-1}(\pi)$. During the backpropagation phase, we must compute the gradient of $f^{-1}(\pi)$ with respect to π. That is, $\frac{\mathrm{d}\sigma}{\mathrm{d}\pi} = \frac{1}{\mathrm{d}\pi/\mathrm{d}\sigma}$, which is the reciprocal of the vega. Hence, for arbitrarily small values of vega, this gradient will explode, leading to convergence problems.

In order to address this issue, we will follow the gradient-squash approach of Liu, Oosterlee, and Bohte (2019). An option value can be divided into its *intrinsic value* (i.e. the no-arbitrage bound) and its *time value*. In order to obtain the time value, we subtract the intrinsic value from the option value, yielding

$$\hat{\pi}_t = \pi_t - (S_t - K\mathrm{e}^{-r\tau})^+, \tag{19}$$

where $\hat{\pi}$ denotes the time value. Finally, we will also apply a log-transform to further reduce the possible steepness of the gradient, resulting in the dataset shown in Table 4.

5.3. Data sampling

We treat each problem discussed above as a *supervised learning* problem. In other words, we require inputs and outputs which we can feed to the network and calculate the loss function. We will follow the configuration outlined in Liu, Oosterlee, and Bohte (2019) for all three problems. We will generate a total of 1 million samples for each problem, which we divide into an 80/20 training and validation set. We will generate an additional 100 thousand samples used for evaluation. When sampling from the space of input parameters, we can either define a joint distribution over the entire domain, or sample each variable separately. Liu, Oosterlee, and Bohte (2019), as well as Papazoglou-Hennig (2020) and Van Mieghem (2022), opt for Latin hypercube sampling (LHS), in

Table 1. Black–Scholes data generating parameters, adopted from Liu, Oosterlee, and Bohte (2019).

	Parameters	Range
Input	Moneyness: S_0/K	[0.4, 1.6]
	Time to Maturity ($\tau = T - t$)	[0.2, 1.1]
	Risk-free rate (r)	[0.02, 0.1]
	Volatility (σ)	[0.01, 1.0]
Output	Scaled call price (π/K)	(0.0, 0.9)

Table 2. Heston data generating parameters, adopted from Liu, Oosterlee, and Bohte (2019).

	Parameters	Range
Input	Moneyness: S_0/K	[0.4, 1.6]
	Time to Maturity ($\tau = T - t$)	[0.2, 1.1]
	Risk-free rate (r)	[0.02, 0.1]
	Correlation (ρ)	[−0.95, 0.0]
	Reversion speed (κ)	[0, 2.0]
	Long term variance ($\bar{v}$)	[0, 0.5]
	Volatility of volatility (γ)	[0, 0.5]
	Initial variance (v_0)	[0.05, 0.5]
Output	Call price (π)	(0.0, 0.67)

Table 3. Implied volatility data generating parameters, adopted from Liu, Oosterlee, and Bohte (2019).

	Parameters	Range
Input	Moneyness: S_0/K	[0.4, 1.6]
	Time to Maturity ($\tau = T - t$)	[0.2, 1.1]
	Risk-free rate (r)	[0.02, 0.1]
	Scaled call price (π/K)	[0, 0.71]
Output	Volatility (σ)	(0.05, 1.0)

which values are sampled from a joint distribution, resulting in a better representation of the parameter space; *cf.* McKay, Beckman, and Conover (1979). Thus, in order to remain consistent with previous research, we use LHS in this work to generate the data as well.

Starting from the Black–Scholes pricing problem, we sample points using the parameters shown in Table 1, and use the analytical solution in Equation (18) to obtain the output labels. We use the same sampling parameters as in Liu, Oosterlee, and Bohte (2019) and Papazoglou-Hennig (2020), allowing us to compare the network performances with the results of both works at a later stage.

The Heston pricing problem obviously requires additional parameters, whose range is listed in Table 2. The range for these parameters is selected again according to Liu, Oosterlee, and Bohte (2019).

In the 'default' implied volatility problem, we reuse the Black–Scholes pricing data and switch the role of the volatility (σ) and the scaled call price (π/K) as input and output, obtaining the parameters listed in Table 3.

Finally, we transform the implied volatility dataset by applying the transformation described in Equation (19), along with a log-transform. The domain of the transformed implied volatility dataset, where we used the scaled time value, is provided in Table 4.

6. Empirical results

In this section, we train neural networks using the architectures discussed in Sections 3 and 4, and apply them to the three different problems outlined in Sections 2 and 5. We then compare the empirical performance of these networks against each other, with the aim of identifying an optimal network architecture for the solution of these and related problems. In addition, we test the method for the computation of implied volatility using real data, and also briefly compare this with existing methods. The implementation for every network was done

Table 4. Transformed implied volatility data generating parameters, adopted from Liu, Oosterlee, and Bohte (2019).

	Parameters	Range
Input	Moneyness: S_0/K	[0.4, 1.6]
	Time to Maturity ($\tau = T - t$)	[0.2, 1.1]
	Risk-free rate (r)	[0.02, 0.1]
	Scaled time value ($\log(\hat{\pi}/K)$)	$[-18.42, -0.95]$
Output	Volatility (σ)	(0.01, 1.0)

Table 5. Overview of MLP configurations.

	Multilayer Perceptron (MLP)
Layers	2, 3
Nodes per layer	50, 100, 150, 200, 250, 500
Activation function	ReLU
Loss function	MSE
Learning rate	10^{-5}
Batch size	64

using Google's *TensorFlow* library for Python (Abadi et al. 2015) and the empirical analysis was carried out using the Delft High Performance Computing Centre (DHPC) (2022) facilities.

6.1. Multilayer perceptrons

We start by training a set of MLPs ranging from a very small network to a large network, with similar configurations. The goal is to evaluate the performance of this network architecture, as well as to determine whether the size of the network influences the performance over a fixed range of epochs. To this end, we define a set of twelve networks with configurations listed in Table 5. We fix the learning rate according to the optimal value found from the study conducted in Liu, Oosterlee, and Bohte (2019). The batch size is chosen as large as possible while keeping the calculations on a CPU fast. We found that a batch size of 64 works well for the supervised learning problems discussed in this work. Furthermore, each network in this work is trained for 200 epochs on the entire training dataset. We found that, in most cases, increasing the number of training epochs is beneficial. However, due to the large amount of network architectures that must be trained, we decided to terminate training after 200 epochs.

We allocate exactly one CPU core from the DHPC (Delft High Performance Computing Centre (DHPC) 2022) (Intel Xeon 3.0 GHz) cluster to each model. Doing so allows us to measure the training time for each model and to compare the performance trade-off to the off-line training time. Since we are training relatively small networks, GPU training does not speed up the process in most cases, as the overhead of distributing across the cores is too large. All networks are trained on the Black–Scholes and Heston pricing problems as well as on the two implied volatility problems. The empirical results for each problem are discussed below.

6.1.1. Black–Scholes model

Figure 11 visualizes the training time alongside the MSE for each network on the test set. We can see a clear relationship between the size of the network, which is increasing on the horizontal axis, the reduction in MSE and the increase in training time. Indeed, increasing the network size decreases the MSE (red) and hence improves the accuracy of the estimation, while the cost associated with this improvement is visible through the increase in training time (blue).

Interestingly, the performance increase of a larger network is not immediately visible during training. Figure 12 visualizes the training and validation losses over the 200 training epochs for the 2 layer networks with 50 nodes (small), 250 nodes (medium) and 500 nodes (large). Notice that even though the largest network is almost 5 times larger than the medium network and performs roughly 3.5 times as well as the medium network, the training loss is almost identical for both networks. This is an indication that losses during training do not represent actual performance on unseen data. Additionally, the validation loss for the medium and large

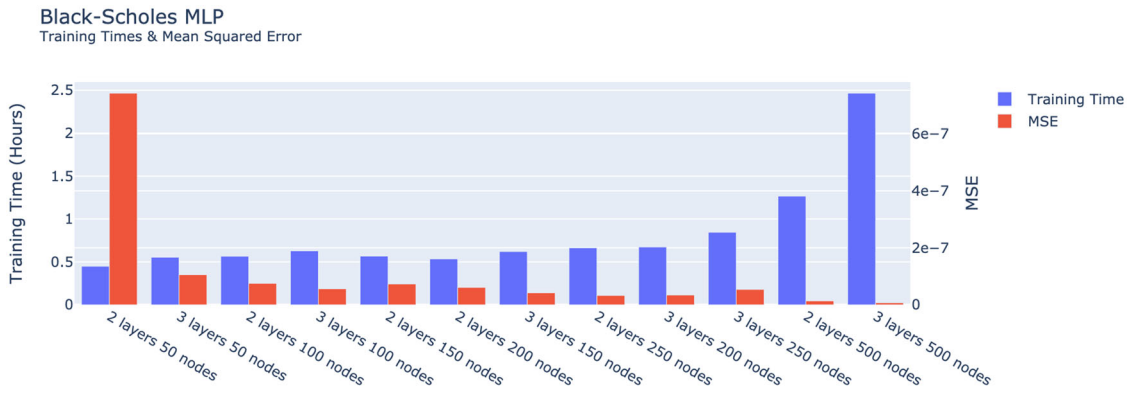


Figure 11. A comparison of the twelve networks from Table 5 including MSE on the test set (red) and training time (blue) for the Black–Scholes problem.

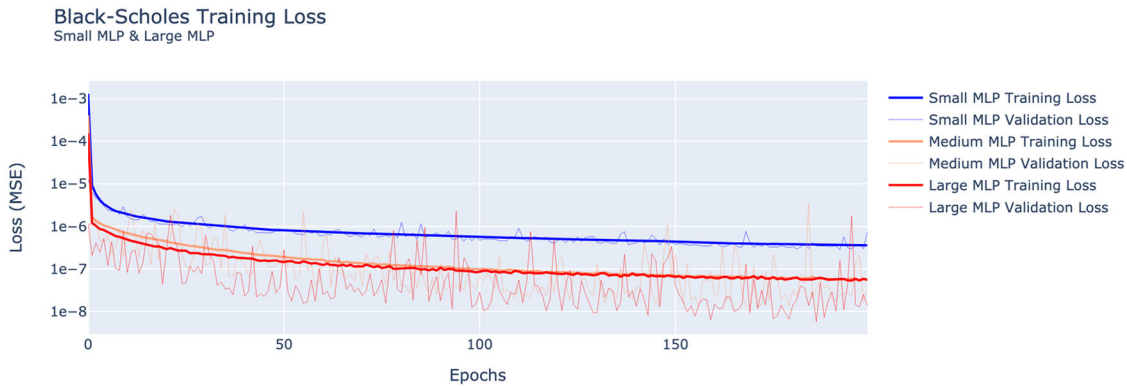


Figure 12. The training and validation losses for the three MLP networks. The small, medium and large networks contain 2851, 64,251 and 504,001 parameters respectively.

networks is lower than the training loss. This indicates a possible performance improvement for both networks if training time is extended for more epochs, which aligns with the results in Liu, Oosterlee, and Bohte (2019). Since the Black–Scholes problem has an analytical solution, we expect to be able to approximate the solution using neural networks to any degree of precision, when choosing an appropriately sized network along with many training cycles. The results from this section support this hypothesis. We select the 3 layer 50 node network to compare against the highway and DGM networks discussed later, as a good trade-off between accuracy and training time. Moreover, for the sake of completeness, we list the configuration of each network in Table 6, including training time and MSE; here, training time is measured in fractions of an hour.

6.1.2. Heston model

The results of the various MLPs trained and evaluated on the Heston pricing problem are presented in Figure 13. Therein we observe that the results for the Heston pricing problem using the MLP network architectures align qualitatively with those for the Black–Scholes problem. Since both problems are very similar in nature and attempt to approximate the same quantity, the similarity in the results is rather expected.

Indeed, the results from both the Black–Scholes as well as the Heston pricing problem exhibit, aside from the magnitude of the MSEs, many similarities; compare Figures 11 and 13. The MSEs and training times for the Heston model can be found in Table 7. The difference in MSEs is not surprising, as the Heston model has 8 parameters compared to the 4 parameters in the Black–Scholes model.

Table 6. An overview of the MLP configurations alongside training time and MSE for the Black–Scholes problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	2851	0.52	$2.3 \cdot 10^{-7}$
2	2	100	10,701	0.57	$7.5 \cdot 10^{-8}$
3	2	150	23,551	0.57	$7.3 \cdot 10^{-8}$
4	2	200	41,401	0.54	$6.1 \cdot 10^{-8}$
5	2	250	64,251	0.66	$3.3 \cdot 10^{-8}$
6	2	500	253,501	1.27	$1.4 \cdot 10^{-8}$
7	3	50	5401	0.55	$1.1 \cdot 10^{-7}$
8	3	100	20,801	0.63	$5.6 \cdot 10^{-8}$
9	3	150	46,201	0.62	$4.2 \cdot 10^{-8}$
10	3	200	81,601	0.67	$3.4 \cdot 10^{-8}$
11	3	250	127,001	0.84	$5.4 \cdot 10^{-8}$
12	3	500	504,001	2.47	$7.0 \cdot 10^{-9}$
MLP (Papazoglou-Hennig 2020)	6	200	202,201	–	$1.3 \cdot 10^{-7}$

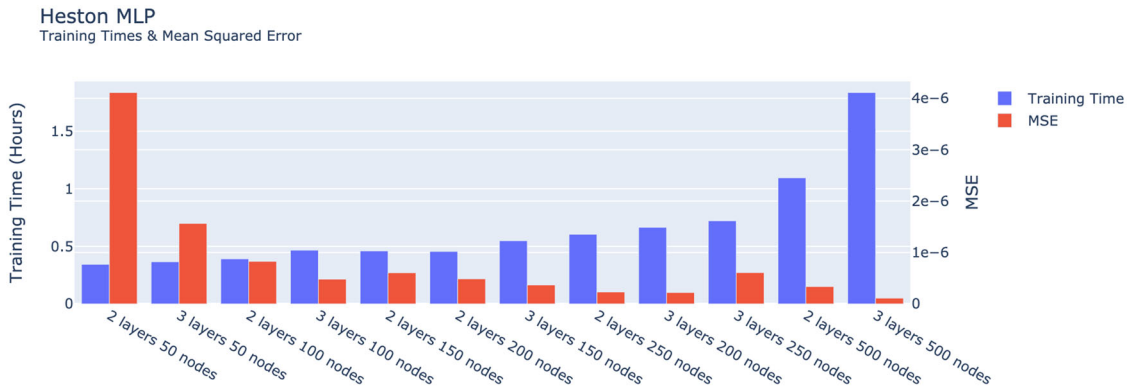


Figure 13. A comparison of the twelve networks from Table 5 including MSE on the test set (red) and training time (blue) for the Heston problem.

Table 7. An overview of the MLP configurations alongside training time and MSE for the Heston pricing problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	3051	0.34	$4.1 \cdot 10^{-6}$
2	2	100	11,101	0.39	$8.3 \cdot 10^{-7}$
3	2	150	24,151	0.46	$6.0 \cdot 10^{-7}$
4	2	200	42,201	0.46	$4.9 \cdot 10^{-7}$
5	2	250	65,251	0.60	$2.3 \cdot 10^{-7}$
6	2	500	255,501	1.09	$3.4 \cdot 10^{-7}$
7	3	50	5601	0.37	$1.6 \cdot 10^{-6}$
8	3	100	21,201	0.47	$4.8 \cdot 10^{-7}$
9	3	150	46,801	0.55	$3.7 \cdot 10^{-7}$
10	3	200	82,401	0.66	$2.2 \cdot 10^{-7}$
11	3	250	128,001	0.72	$6.1 \cdot 10^{-7}$
12	3	500	506,001	1.84	$1.1 \cdot 10^{-7}$
MLP (Liu, Oosterlee, and Bohte 2019)	4	400	646,006	–	$1.7 \cdot 10^{-8}$

In Table 7, we observe that the MLP from Liu, Oosterlee, and Bohte (2019) performs 10 times better than the best performing MLP that we trained. We trained each network for 200 epochs, whereas the authors of Liu, Oosterlee, and Bohte (2019) train for 3000 epochs. Since the architecture of the networks is identical, this indicates once again that longer training on this problem improves performance, similar to what we observed for the Black–Scholes pricing problem.

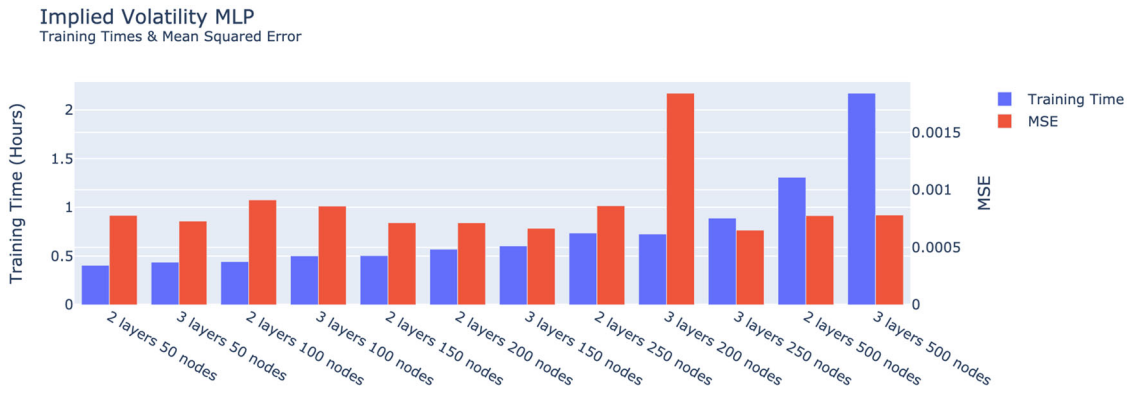


Figure 14. A comparison of the twelve networks from Table 5 including MSE on the test set (red) and training time (blue) for the implied volatility problem.

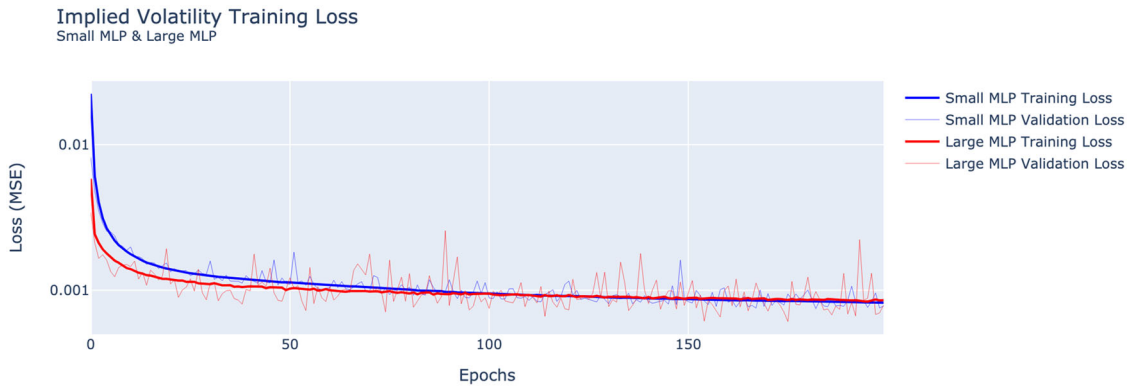


Figure 15. The training and validation loss of the smallest network compared to the largest network on the implied volatility data set.

6.1.3. Implied volatility

The main observation for the implied volatility problem is that, contrary to the Black–Scholes and Heston problems, we cannot find evidence of larger networks performing better for this (inverse) problem. Indeed, the smallest network we train consists of 2 layers and 50 nodes per layer, containing a total of 2851 parameters, and performs just as well after 200 epochs with identical configuration as the largest network with 3 layers and 500 nodes, containing 504,001 parameters. Both networks have an MSE on the test set of $7.7 \cdot 10^{-4}$, while the small network took only 0.4 h to train compared to the 2.17 h for the largest network. All networks show comparable performance, as can be seen in Figure 14.

The identical performance of the networks during training is visualized in Figure 15. The results we find align with the errors found in Papazoglou-Hennig (2020), in which the best performing MLP network containing 202,201 parameters performed worse on the test data than the considerably smaller networks from Figure 14. Table 8 lists all the networks along with the training times, number of parameters and MSE for the implied volatility problem. The results indicate that the networks are dealing with convergence issues that prevent them from further optimization, regardless of their size and architecture.

Next, we consider the transformed implied volatility dataset, where we aim to solve the steep gradient problems possibly causing the convergence issues by introducing a transformation from the scaled call price to the scaled time value of the option, explained in Subsection 5.2.

Figure 16 visualizes the performance of the MLPs on the transformed implied volatility problem. We notice that on the transformed problem, the MSE is decreasing as a function of the network size, similarly to what we

Table 8. An overview of the MLP configurations alongside training time and MSE for the implied volatility problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	2851	0.40	$7.7 \cdot 10^{-4}$
2	2	100	10,701	0.44	$9.1 \cdot 10^{-4}$
3	2	150	23,551	0.50	$7.1 \cdot 10^{-4}$
4	2	200	41,401	0.57	$7.1 \cdot 10^{-4}$
5	2	250	64,251	0.74	$8.6 \cdot 10^{-4}$
6	2	500	253,501	1.31	$7.8 \cdot 10^{-4}$
7	3	50	5401	0.44	$7.3 \cdot 10^{-4}$
8	3	100	20,801	0.50	$8.5 \cdot 10^{-4}$
9	3	150	46,201	0.60	$6.6 \cdot 10^{-4}$
10	3	200	81,601	0.73	$1.8 \cdot 10^{-3}$
11	3	250	127,001	0.94	$7.6 \cdot 10^{-4}$
12	3	500	504,001	2.17	$7.7 \cdot 10^{-4}$

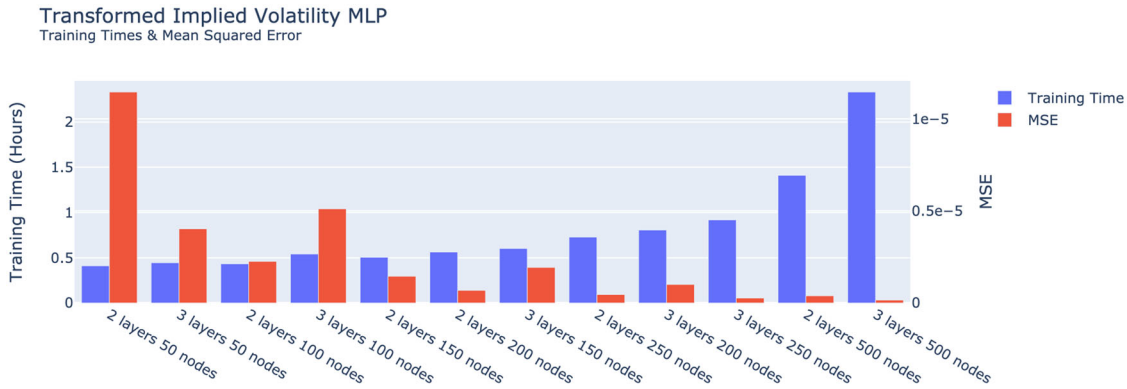


Figure 16. A comparison of the twelve networks from Table 5 including MSE on the test set (red) and training time (blue) for the transformed implied volatility problem.

Table 9. An overview of the MLP configurations alongside training time and MSE for the transformed implied volatility problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	2851	0.41	$1.1 \cdot 10^{-5}$
2	2	100	10,701	0.43	$2.3 \cdot 10^{-6}$
3	2	150	23,551	0.51	$1.5 \cdot 10^{-6}$
4	2	200	41,401	0.56	$1.9 \cdot 10^{-6}$
5	2	250	64,251	0.73	$6.9 \cdot 10^{-6}$
6	2	500	253,501	1.41	$3.9 \cdot 10^{-7}$
7	3	50	5401	0.44	$4.0 \cdot 10^{-6}$
8	3	100	20,801	0.54	$5.1 \cdot 10^{-6}$
9	3	150	46,201	0.60	$1.9 \cdot 10^{-6}$
10	3	200	81,601	0.81	$1.0 \cdot 10^{-6}$
11	3	250	127,001	0.92	$2.7 \cdot 10^{-7}$
12	3	500	504,001	2.33	$1.6 \cdot 10^{-7}$

found for the option pricing problems before. The reduction in MSE is not as consistent as in the previous problems, indicating difficult convergence during training, possibly still due to a steep gradient not being entirely removed by the transformation. The results are considerably better than for the 'default' implied volatility problem, using only a simple transformation. Table 9 shows the results of all MLPs alongside their parameters and training times.

Table 10. Overview of highway network configurations.

	Generalized Highway	Highway	Residual
Layers	3	3	3
Nodes per layer	50	50	50
Total parameters	23,251	15,601	7951
Initializer	Glorot Normal	Glorot Normal	Glorot Normal
Carry/Transform activation function	*tanh*	*tanh*	–
Activation function	*tanh*	*tanh*	*tanh*
Loss function	MSE	MSE	MSE
Learning rate	10^{-5}	10^{-5}	10^{-5}
Batch size	64	64	64

Table 11. An overview of the highway network performance and error reductions for the MLP and highway network architectures on the Black–Scholes pricing problem.

	Time (H)	MSE	Reduction (%)	Layer operation
MLP	0.55	$1.1 \cdot 10^{-7}$	–	H
Residual	0.55	$6.7 \cdot 10^{-8}$	33%	$H + \mathbf{x}$
Highway	0.60	$2.8 \cdot 10^{-8}$	61%	$H \cdot T + \mathbf{x} \cdot (1 - T)$
Generalized Highway	0.68	$7.4 \cdot 10^{-9}$	73%	$H \cdot T + \mathbf{x} \cdot C$

Note: The reductions in the fourth column are relative to the previous row.

In order to quantify the practical impact of the log-transformation we compare the performance of identical architectures on the default and transformed implied volatility datasets. Using the 2-layer MLP with 150 nodes per layer, the default formulation (Table 8) yields a test MSE of 7.1×10^{-4}, whereas the transformed formulation (Table 9) reduces the error to 1.5×10^{-6}. This improvement by almost three orders of magnitude is achieved without changing the architecture, hyperparameters, or training regime. The substantial gain in accuracy confirms empirically that the time-value and log-transform significantly improve numerical conditioning and optimization stability, consistent with the gradient analysis in Section 5.2.

6.2. Highway networks

In this section, we train variants of the highway networks discussed in Section 4.2 with approximately equal configurations. More specifically, each network contains 3 layers each with 50 nodes. We train relatively small networks to compare across various architectures, in order to keep computational times feasible. Table 10 lists each highway network configuration.

6.2.1. Black–Scholes model

Figure 17 visualizes the training times (blue) and MSE on the test set (red) for each highway network. Both the highway and generalized highway networks perform better than their MLP counterpart with an equal number of nodes. More impressive, however, is the fact that both networks required barely more training time than the small MLP, but scored comparably to the large MLP on the test set. This means that the highway architecture improves convergence on the Black–Scholes problem significantly. The difference in convergence speed can also be seen during training, as shown in Figure 18. In this case, training losses are a good indicator for performance on the test set, likely because the performance difference for both networks is so large. The loss during training is roughly one order of magnitude smaller for the generalized highway model compared to the MLP, whereas the training time is only 23% longer; *cf.* Table 11.

The residual layer is closest to the classical dense layer. The only difference is the carry of the input $\mathbf{x}$ to the next layer, *cf.* (10). Surprisingly, only carrying the input vector to the next layer decreases the MSE on the test set by 33% from $1.1 \cdot 10^{-7}$ to $6.7 \cdot 10^{-8}$. Since only one additional operation is added in each layer, the training time is almost identical. Sirignano and Spiliopoulos (2018), in the context of solutions for PDEs, argue that including the input vector in many operations allows the network to make 'sharper turns', resulting in a more flexible network. We see that this feature becomes particularly useful also when approximating the Black–Scholes price.

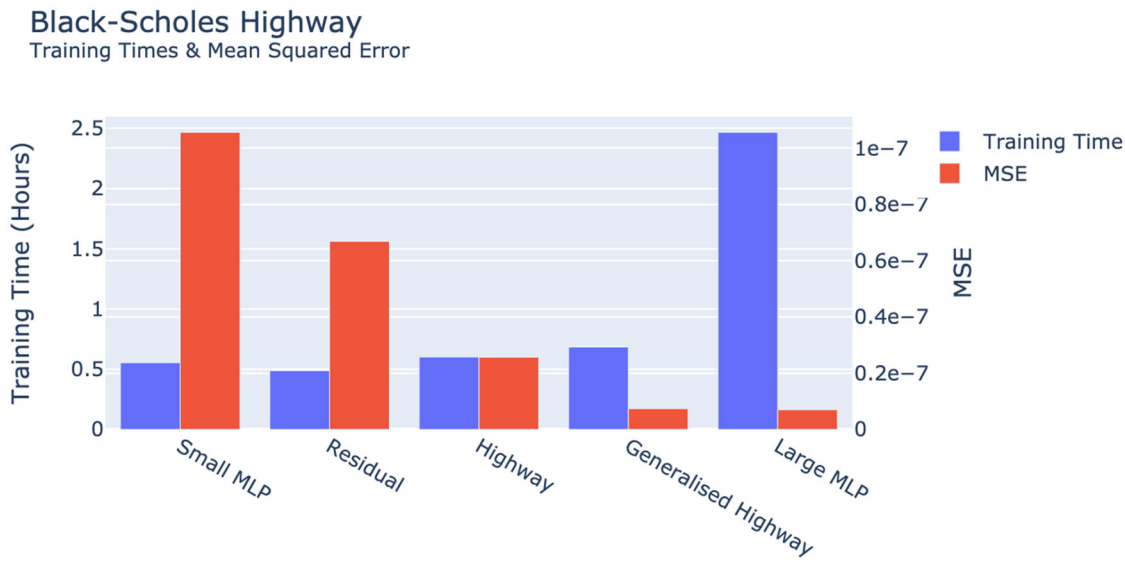


Figure 17. A comparison of highway networks from Table 10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) for the Black–Scholes problem. The small MLP contains 3 layers with 50 nodes, and the large MLP 3 layers with 500 nodes. The results are ordered by the total number of parameters.

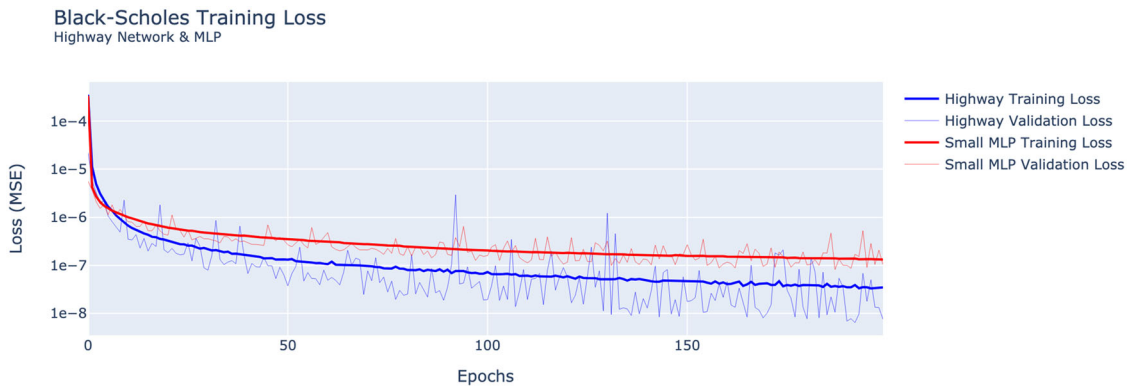


Figure 18. The training and validation loss over 200 epochs. The small MLP consisting of 3 layers and 50 nodes (2851 parameters) in red, compared to the 3 layer 50 node generalized highway network (23,251 parameters) in blue.

The difference in performance between the highway network and the residual network is also significant. The incorporation of one extra weight matrix and one additional non-linear function multiplied with the input vector $\mathbf{x}$ in each layer, *cf.* (11), results in an MSE reduction of 61% from the residual network's $6.7 \cdot 10^{-8}$ to the highway network's $2.6 \cdot 10^{-8}$. Adding one additional weight matrix W_C and non-linearity C, yielding the generalized highway layer, improves MSE performance by another 61% compared to the highway layer. An overview of the error reductions, training times and operations in each layer are given in Table 11.

Finally, we notice that a 23% increase in training time from the small MLP to the generalized highway network leads to a reduction of the MSE by 99%. When comparing the results obtained for the highway networks to the results in Liu, Oosterlee, and Bohte (2019, Table 6), in which a large MLP is trained and evaluated on identical data, we see that our small generalized highway network already outperforms the MLP. In fact, as shown in Table 14, the generalized highway network improves the MSE by 9.8% while reducing the amount of parameters by more than 96% compared to the MLP from Liu, Oosterlee, and Bohte (2019).

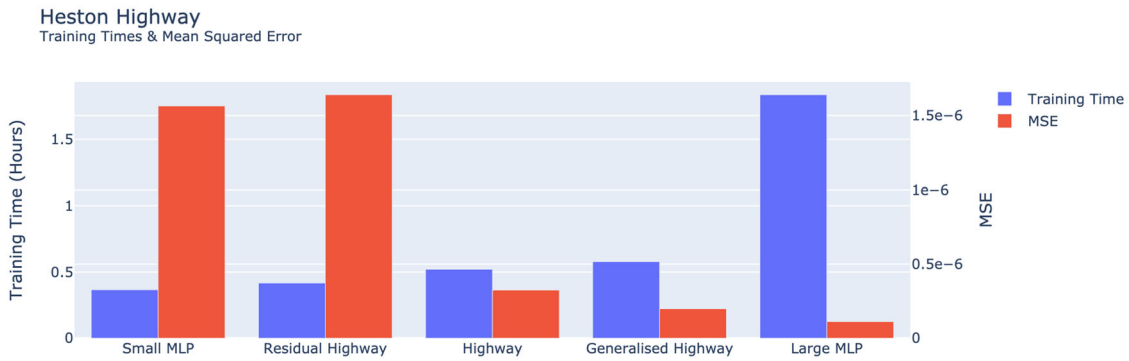


Figure 19. A comparison of highway networks from Table 10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) on the Heston pricing problem. The small MLP contains 3 layers with 50 nodes, and the large MLP contains 3 layers with 500 nodes.

Table 12. An overview of the highway network performance and error reductions for the MLP and highway network architectures on the Heston pricing problem.

	Time (H)	MSE	Reduction (%)	Layer Operation
MLP	0.37	$1.6 \cdot 10^{-6}$	–	H
Residual	0.42	$1.7 \cdot 10^{-6}$	−5%	$H + \mathbf{x}$
Highway	0.58	$3.3 \cdot 10^{-7}$	80%	$H \cdot T + \mathbf{x} \cdot (1 - T)$
Generalized Highway	0.68	$2.0 \cdot 10^{-7}$	39%	$H \cdot T + \mathbf{x} \cdot C$

Note: The reductions in the fourth column are relative to the previous row.

6.2.2. Heston model

Figure 19 visualizes the performance of the highway networks for the Heston pricing problem. We observe again a similar behaviour as in the Black–Scholes pricing problem; compare with Figure 17. Indeed, the highway and generalized highway networks perform comparably to the large MLP in terms of MSE, while requiring the training time of the small MLP. A major difference is the performance of the residual network, which reduced the MSE by 33% compared to the MLP on the Black–Scholes problem; *cf.* Table 11. On the Heston pricing problem, we do not observe an increase in performance. A possible explanation for this could be that the Heston pricing problem might be too involved for the residual architecture to make an improvement.

The highway and generalized highway networks do, in fact, significantly improve performance compared to the MLP networks, with the generalized highway network performing almost as well as the largest MLP, while requiring only 31% of the computational time. Table 12 lists the reductions that each highway variation contributes to the MSE. We notice that the improvement of the generalized highway network on the Heston problem is less significant than on the Black–Scholes problem, where the generalized highway network was the best performing network. In Section 6.3, we compare the highway networks to their DGM counterpart on the Heston pricing problem, and consider equally sized networks in terms of parameters.

6.2.3. Implied volatility

In the previous subsections, we found significant reductions in MSE on the test set when comparing the highway networks to the MLPs for the Black–Scholes and the Heston pricing problems. In this subsection, we perform the same comparison for the implied volatility problems. The results are visualized in Figure 20. In contrast to the Black–Scholes problem, in which the highway networks outperformed the MLPs, this is not the case for the implied volatility problem. None of the highway networks outperforms the MLPs on the test set. As in the MLP scenario, this indicates again that the steep gradient also poses a problem when optimizing the highway networks. In order to mitigate this issue, we consider the performance of the networks on the transformed implied volatility problem.

In Figure 21 the performance of the highway networks on the transformed implied volatility dataset is shown. On this dataset, we find similar relative MSEs and computational times as for the Black–Scholes and Heston



Figure 20. A comparison of the highway networks from Table 10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) on the implied volatility problem. The small MLP contains 3 layers with 50 nodes and the large MLP contains 3 layers with 500 nodes.

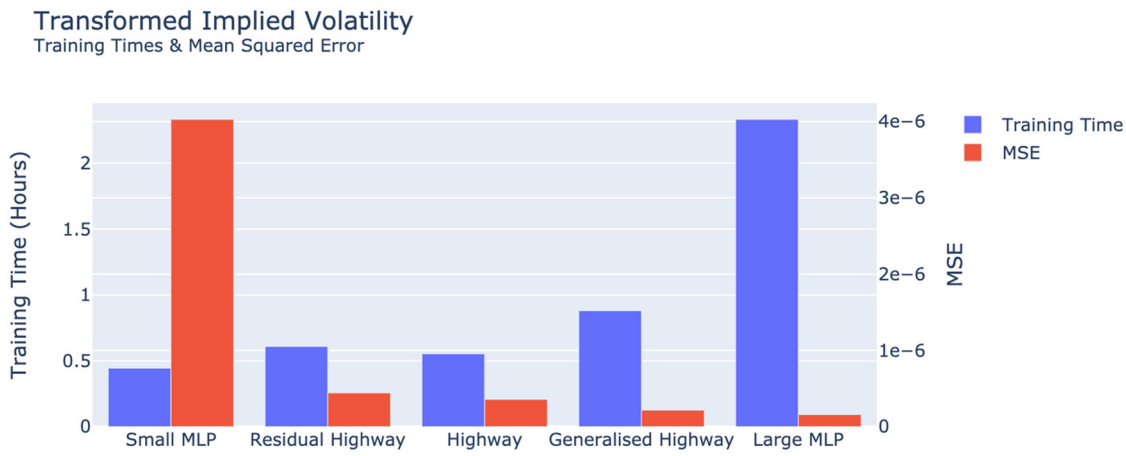


Figure 21. A comparison of the highway networks from Table 10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) on the transformed implied volatility dataset. The small MLP contains 3 layers with 50 nodes and the large MLP contains 3 layers with 500 nodes.

pricing problems. In fact, the generalized highway network is again the superior network relative to the training time. However, just like in the Heston pricing problem scenario, the large MLP scores lower in absolute MSE. This indicates that the network performance on the implied volatility problem is sensitive to larger network sizes. We expect to see performance improvements when training highway networks with more nodes and layers.

These results clarify how much influence the network architecture has on the performance; consider, for example, the difference between the small MLP and the residual highway network. Recall that the operation of a residual network is simply the addition of the input vector to the layer; *cf.* (10). This operation reduces the MSE by roughly 89%, while adding only 38% to the training time. We also notice that the training times for an identical network architecture can differ per dataset. In order to see this, consider the 14% increase in training time from the small MLP to the residual network on the Heston problem (Table 12) and the 0% increase on the Black–Scholes problem (Table 11).

Table 13. Overview of DGM network configurations.

	DGM Network
Layers	3
Nodes per layer	50
Total parameters	33,459
Activation function	*tanh*
Initialiser	Glorot Normal
Loss function	MSE
Learning rate	10^{-5}
Batch size	64

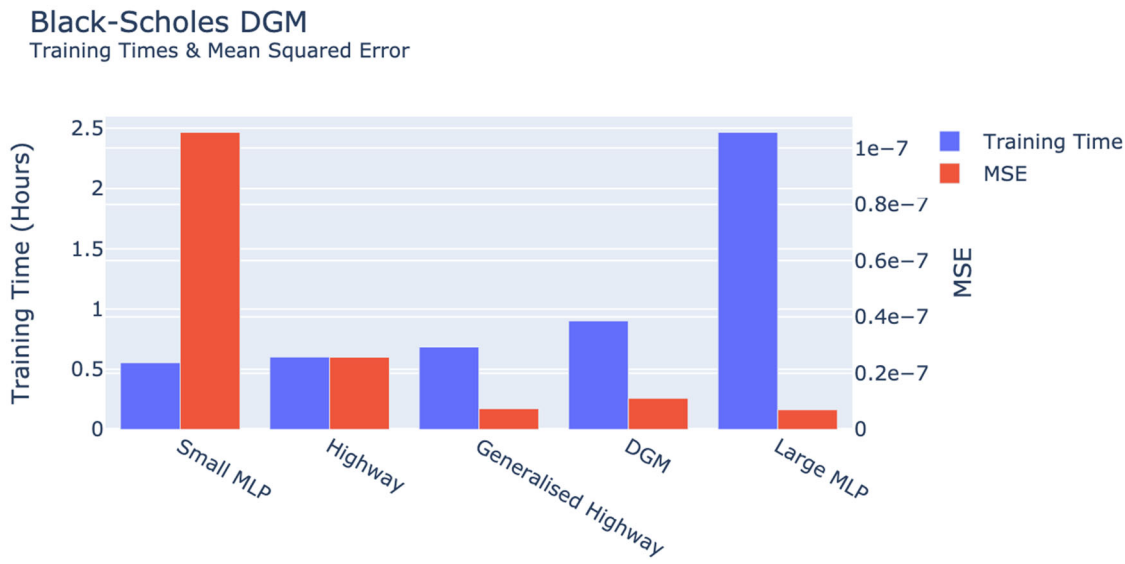


Figure 22. The MLP, highway and DGM networks compared in terms of MSE on the test set (red) and training time (blue) for the Black–Scholes model. The small MLP contains 3 layers with 50 nodes and the large MLP contains 3 layers with 500 nodes.

6.3. DGM networks

In this section, we train the DGM network and its variations presented in Subsections 4.4–4.6 and compare their performance to the highway and MLP networks. We train the DGM network with the configuration listed in Table 13.

6.3.1. Black–Scholes model

The results of the DGM network performance in the Black–Scholes problem are visualized in Figure 22. We note that more operations must be computed inside a DGM network than in a highway network, resulting in slightly more parameters and hence in longer training time. This aligns with the numerical results.

The main difference between the highway layer and the DGM layer is that the original input $\mathbf{x}$ is used in each DGM layer, whereas the highway layer only depends on the output of the previous layer. The dependence on $\mathbf{x}$ introduces 4 additional weight matrices per layer; see Figure 7. In Figure 22 no significant improvements due to this dependence compared to the generalized highway network can be observed. Indeed, the generalized highway network has a smaller MSE and requires less computational time for training. Possibly, the Black–Scholes problem is not complex enough for the additional non-linearity and recurrence in the DGM layers to cause improvements.

Figure 23 reports the training and validation MSE for the DGM, generalized highway, and large MLP architectures applied to the Black–Scholes problem. The training and validation losses decrease concurrently for all

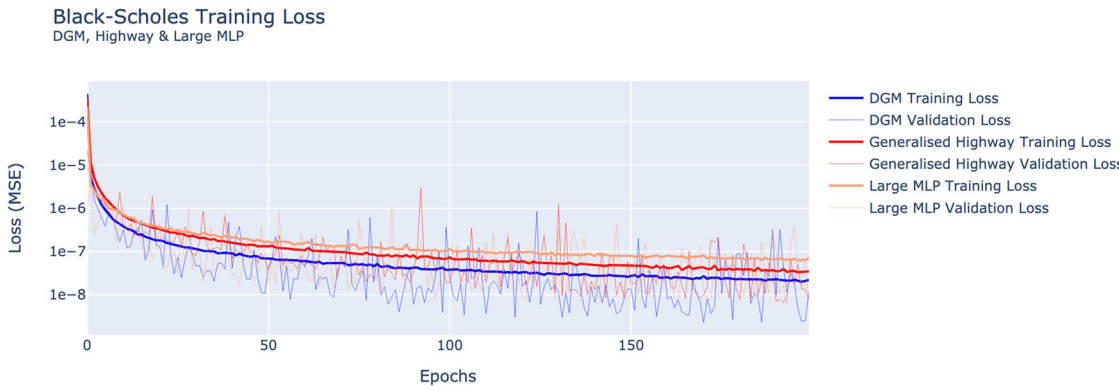


Figure 23. The MLP, highway and DGM networks compared during the training cycles.

models throughout the optimization and remain closely aligned, with no evidence of a systematic divergence between the two curves. This indicates that none of the architectures exhibit overfitting over the training horizon considered. Although the validation losses display higher variance, particularly at later epochs, their overall downward trend mirrors that of the training losses, which is consistent with stochastic simulation and Monte Carlo sampling effects rather than deteriorating generalization. Among the three architectures, DGM attains the lowest error levels, followed by the generalized highway network and the large MLP, reflecting differences in architectural inductive bias rather than generalization failure. Overall, the results demonstrate stable training dynamics and good generalization performance across all models.

In order to gain more insights into the impact of the operations inside the DGM network on the performance, we also train the deep DGM network and the no-recurrence DGM network with the same configuration as in Table 13, with the deep DGM layer containing 3 sublayers. In Figure 24 the MSEs of the DGM variations are compared against the DGM and highway networks. As expected from the previous observations, the addition of sublayers inside the DGM layer, resulting in the deep DGM network (*cf.* Subsection 4.5), does not bring significant performance improvements, while increasing training time by almost 50% compared to the "plain" DGM network. Removing the dependence on the input vector $\mathbf{x}$ in all layers, resulting in the no-recurrence DGM network (*cf.* Subsection 4.6), more than tripled the network error. This suggests that for the Black–Scholes pricing problem, the DGM network admits its performance benefits from the recurrence in the network.

The operations inside the no-recurrence network are very similar to those in a highway network. However, the MSEs on the test set are nowhere close. In order to verify that this was not a numerical issue, the no-recurrence network was trained multiple times, yielding similar results. The layer operations for the no-recurrence network and the generalized highway network after renaming weight matrices and activation functions are

$$\mathbf{y} = [1 - G(\mathbf{x}, W_G)] \odot H\left(R(\mathbf{x}, W_R) \cdot \mathbf{x}\right) + Z(\mathbf{x}, W_Z) \odot \mathbf{x} \tag{20}$$

and

$$\mathbf{y} = [1 - Z(\mathbf{x}, W_Z)] \odot H(\mathbf{x}, W_H) + Z(\mathbf{x}, W_Z) \odot \mathbf{x}, \tag{21}$$

respectively. Hence, the difference in performance likely originates from the additional non-linearity R or G. The fact that the introduction of two additional operations have such a negative influence in accuracy suggests that choosing an optimal network architecture can be a process subject to high sensitivity.

In addition, for the sake of completeness, Table 14 lists the Black–Scholes pricing MSE results for the most important networks, alongside the performance of the networks from Papazoglou-Hennig (2020) and Liu, Oosterlee, and Bohte (2019). We notice that the generalized highway networks achieve the lowest errors among the considered architectures.



Figure 24. The highway and DGM networks, including DGM variations (deep DGM and no-recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the Black–Scholes pricing problem.

Table 14. An overview of MLP, highway and DGM network configurations alongside their training time and MSE for the Black–Scholes pricing problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
Small MLP	3	50	5401	0.55	$1.1 \cdot 10^{-7}$
Highway	3	50	15,601	0.60	$2.6 \cdot 10^{-8}$
Generalized Highway	3	50	23,251	0.68	$7.4 \cdot 10^{-9}$
No-Recurrence DGM	3	50	31,059	0.90	$4.8 \cdot 10^{-8}$
DGM	3	50	33,459	0.90	$1.1 \cdot 10^{-8}$
Deep DGM	3	50	49,959	1.33	$8.8 \cdot 10^{-9}$
Large MLP	3	500	504,001	2.47	$7.0 \cdot 10^{-9}$
MLP (Papazoglou-Hennig 2020)	6	200	202,201	–	$1.3 \cdot 10^{-7}$
DGM (Papazoglou-Hennig 2020)	5	100	210,601	–	$1.6 \cdot 10^{-7}$
MLP∗ (Liu, Oosterlee, and Bohte 2019)	4	400	644,401	–	$8.2 \cdot 10^{-9}$

Note: ∗Trained for 3000 epochs, whereas the other networks are trained for 200. We also consider the networks from Papazoglou-Hennig (2020) and Liu, Oosterlee, and Bohte (2019).

6.3.2. Heston model

In Figure 25 the performance of the DGM network alongside the smallest and largest 3-layer MLPs and highway networks in the Heston pricing problem is shown. We notice that, unlike the Black–Scholes problem, the DGM network achieves a lower MSE than both the large MLP and the generalized highway network. The reason could be that the DGM has slightly more parameters which can lead to performance improvements; see also the discussion in Section 6.4.

Moreover, in the same figure (Figure 25) the results of the deep DGM and the no-recurrence DGM networks are included. We notice that neither the no-recurrence network nor the deep DGM network significantly improve performance or training time. Interestingly, the no-recurrence DGM also requires slightly longer training time, despite a fewer number of parameters in the network, compared to the DGM network. We can conclude that the DGM network outperforms both variations. In addition, we find that while the recurrence in the DGM network has decreased the MSE, the additional complexity it brings along does not make it preferable for option pricing problems in the tested setup.

Concluding, we find that the generalized highway network outperforms all other networks in terms of performance compared to total parameters and training time, similarly to the Black–Scholes problem.

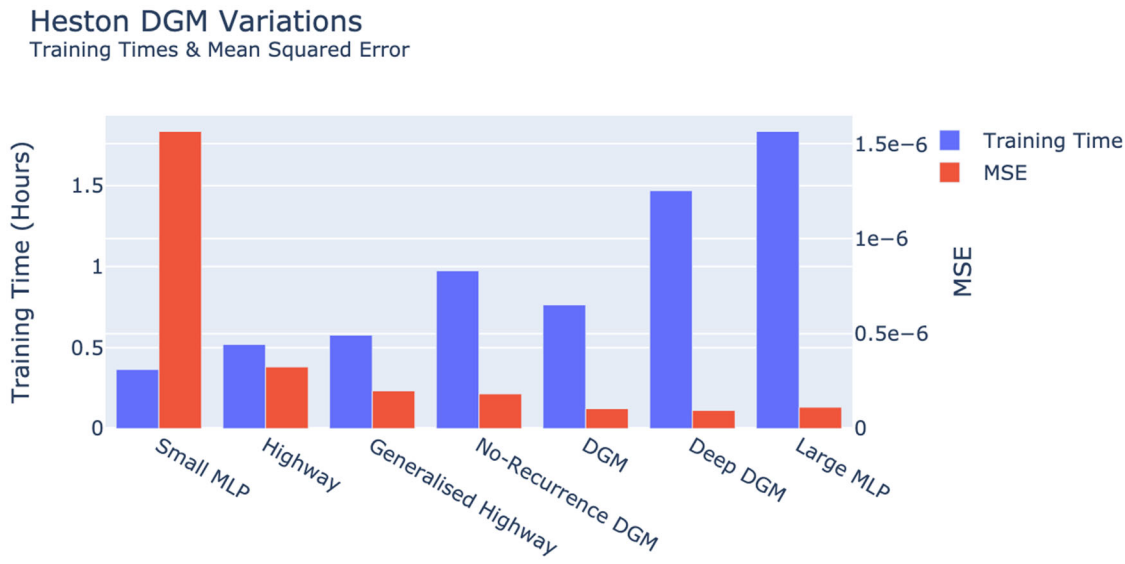


Figure 25. The highway and DGM networks, including DGM variations (deep DGM and no-recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the Heston pricing problem.

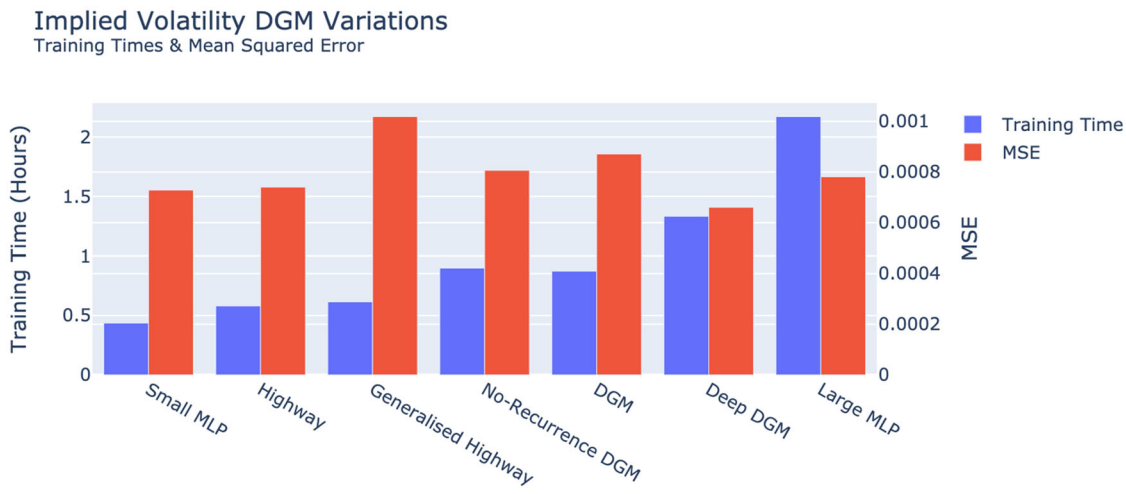


Figure 26. The highway, MLP and DGM network including DGM variations (deep DGM and no-recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the implied volatility problem.

6.3.3. Implied volatility

Finally, we examine the performance of the DGM networks on the implied volatility problem. We have seen in the previous subsections that the implied volatility problem has not been susceptible to performance increases with different architectures, which is likely due to the steep gradient problem present in the 'default' implied volatility formulation. We again find this result when considering the DGM networks. Table 15 lists the training times and MSE on the implied volatility problem for the networks considered in Figure 26. We notice that the 3 layer 150 node MLP performs best in terms of MSE and training time. We find comparable performance from the networks trained in Papazoglou-Hennig (2020) and Liu, Oosterlee, and Bohte (2019).

Since the results we find on the implied volatility dataset is probably a result of convergence problems originating from the formulation, we consider the DGM performance on the transformed implied volatility problem

Table 15. An overview of MLP, highway and DGM network configurations along with training time and MSE for the implied volatility problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
Small MLP	3	50	5401	0.44	$7.3 \cdot 10^{-4}$
Highway	3	50	15,601	0.51	$7.4 \cdot 10^{-4}$
Generalized Highway	3	50	23,251	0.62	$1.0 \cdot 10^{-3}$
No-Recurrence DGM	3	50	31,059	0.90	$8.1 \cdot 10^{-4}$
DGM	3	50	33,459	0.87	$8.7 \cdot 10^{-4}$
Best MLP	3	150	46,201	0.60	$6.6 \cdot 10^{-4}$
Deep DGM	3	50	49,959	1.33	$6.6 \cdot 10^{-4}$
Large MLP	3	500	504,001	2.17	$7.8 \cdot 10^{-4}$
MLP (Papazoglou-Hennig 2020)	6	200	202,201	–	$8.1 \cdot 10^{-4}$
DGM (Papazoglou-Hennig 2020)	5	100	210,601	–	$6.5 \cdot 10^{-4}$
MLP∗ (Liu, Oosterlee, and Bohte 2019)	4	400	644,401	–	$6.4 \cdot 10^{-4}$

Note: ∗Trained for 3000 epochs, whereas the other networks are trained for 200. We also consider the networks from Papazoglou-Hennig (2020) and Liu, Oosterlee, and Bohte (2019).

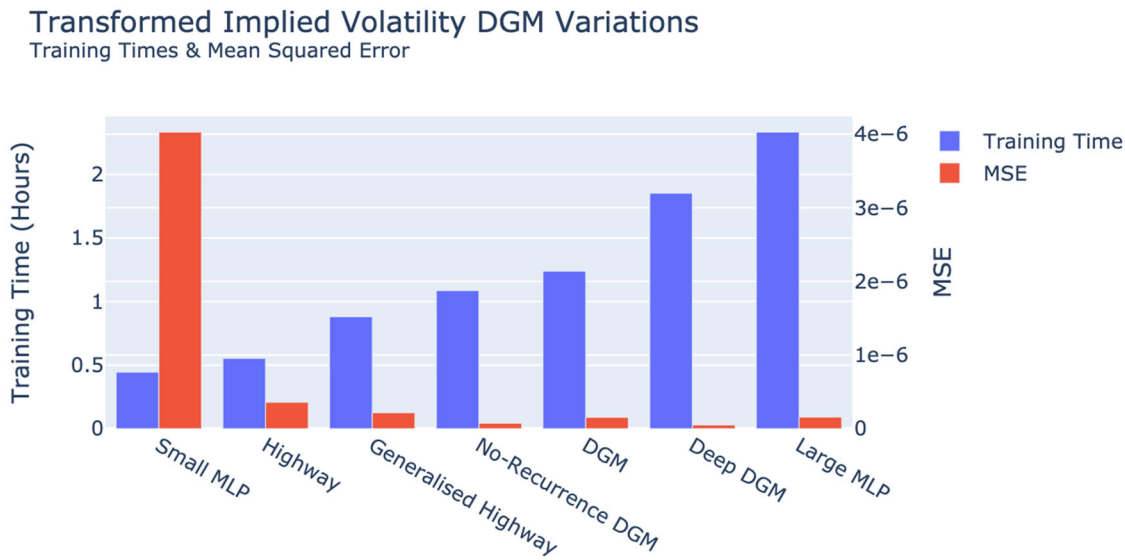


Figure 27. The highway, MLP and DGM networks including DGM variations (deep DGM and no-recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the transformed implied volatility problem.

in the remainder of this section. We observed in Subsection 6.2 that the highway architectures showed strong signs of decreased MSE with little increase in training time. Figure 27 shows the results of the DGM alongside the highway and MLP reference networks.

We notice that, similar to the Heston pricing problem, the DGM network outperforms the other networks in terms of absolute MSE. However, the generalized highway network performs better when the training time is taken into account.

When we also include the results of the DGM variations, see again Figure 27, we find that the no-recurrence DGM shows even better performance than the DGM network. This network architecture is similar to the generalized highway architecture, but contains one additional non-linear operation. In all the previous problems, the additional non-linear operation caused reductions in performance compared to the simpler generalized highway architecture. However, for the transformed implied volatility problem, we observe that the no-recurrence architecture shows a 51% reduction in MSE compared to the DGM network, and a 66% reduction compared to the generalized highway. Table 16 shows the MSEs, training times and parameters for all the networks of interest.

Table 16. An overview of MLP, highway and DGM network configurations along with training time and MSE for the transformed implied volatility problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
Small MLP	3	50	5401	0.44	$4.0 \cdot 10^{-6}$
Highway	3	50	15,601	0.55	$3.6 \cdot 10^{-7}$
Generalized Highway	3	50	23,251	0.88	$2.2 \cdot 10^{-7}$
No-Recurrence DGM	3	50	31,059	1.09	$7.4 \cdot 10^{-8}$
DGM	3	50	33,459	1.24	$1.5 \cdot 10^{-7}$
Deep DGM	3	50	49,959	1.33	$5.0 \cdot 10^{-8}$
Large MLP	3	500	504,001	2.33	$1.6 \cdot 10^{-7}$
MLP∗ (Liu, Oosterlee, and Bohte 2019)	4	400	644,401	–	$1.6 \cdot 10^{-8}$

Note: ∗Trained for 3000 epochs, whereas the other networks are trained for 200. We also consider the network from Liu, Oosterlee, and Bohte (2019).

Table 17. Comparison of DGM variants in terms of structural features, parameter count, training time, and MSE for the Black-Scholes problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
No-Recurrence DGM	3	50	31,059	0.90	$4.8 \cdot 10^{-8}$
DGM	3	50	33,459	0.90	$1.1 \cdot 10^{-8}$
Deep DGM	3	50	49,959	1.33	$8.8 \cdot 10^{-9}$

Taking the training times into account, we can conclude that the no-recurrence DGM network shows the most favourable performance among the architectures considered for the implied volatility problem.

To summarize the results of the architectural variations within the DGM family, Table 17 provides a concise comparison of their structural differences, parameter counts, and empirical performance.

6.4. Capacity-normalized comparison of architectures

Although architectural comparisons across networks such as the residual, highway, generalized highway, and DGM networks are informative, the differences in parameter counts can bias conclusions in favour of higher-capacity models. Therefore, in order to isolate the effect of the architectural design from the raw model capacity, we conducted a capacity-normalized experiment in which all architectures were adjusted to contain approximately the same number of trainable parameters.

6.4.1. Experimental setup.

We designed new experiments in which all architectures were resized to contain roughly the same number of parameters. The following parameter-matched configurations were used:

- **MLP:** 2 layers × 150 units
- **Residual:** 3 layers × 120 units
- **Highway:** 4 layers × 50 units
- **Generalized Highway:** 3 layers × 50 units
- **DGM:** 2 layers × 50 units

All models were trained under identical conditions: Adam optimizer with learning rate 10^{-5}, batch size of 64, and 200 epochs. The same training, validation, and test splits as in the main experiments were used (i.e. 80% / 20% / 10%), and each experiment was repeated with three random seeds to report mean and standard deviation for both mean squared error (MSE) and training time. Tables 18–20 summarize the results for the Black–Scholes, Heston, and implied volatility problems, respectively.

Table 18. Capacity-normalized comparison for the Black–Scholes problem (all networks have approximately 23k parameters).

Model	Layers	Nodes	Parameters	Training Time (h) Mean ± Std	MSE Mean ± Std
MLP	2	150	23,551	1.03 ± 0.016	$(2.76 \pm 0.61) \cdot 10^{-6}$
Residual	3	102	22,881	1.04 ± 0.025	$(1.00 \pm 0.033) \cdot 10^{-6}$
Highway	4	59	22,861	1.07 ± 0.023	$(9.97 \pm 5.15) \cdot 10^{-8}$
Generalized Highway	3	59	22,421	1.08 ± 0.025	$(1.69 \pm 0.27) \cdot 10^{-7}$
DGM	2	50	22,459	1.01 ± 0.009	$(2.38 \pm 1.74) \cdot 10^{-7}$

Table 19. Capacity-normalized comparison for the Heston problem (all networks have approximately 23k parameters).

Model	Layers	Nodes	Parameters	Training Time (h) Mean ± Std	MSE Mean ± Std
MLP	2	150	24,151	0.92 ± 0.05	$(6.2 \pm 5.0) \cdot 10^{-6}$
Residual	3	104	23,713	0.95 ± 0.02	$(1.48 \pm 0.05) \cdot 10^{-5}$
Highway	4	59	22,834	1.04 ± 0.04	$(2.81 \pm 0.35) \cdot 10^{-5}$
Generic Highway	3	59	23,365	1.06 ± 0.05	$(2.31 \pm 0.18) \cdot 10^{-5}$
DGM	2	50	24,467	1.10 ± 0.02	$(3.31 \pm 0.16) \cdot 10^{-5}$

Table 20. Capacity-normalized comparison for the implied volatility problem (all networks have approximately 23k parameters).

Model	Layers	Nodes	Parameters	Training Time (h) Mean ± Std	MSE Mean ± Std
MLP	2	150	23,551	1.09 ± 0.01	$(1.99 \pm 0.01) \cdot 10^{-3}$
Residual	3	102	22,033	1.04 ± 0.07	$(3.35 \pm 0.10) \cdot 10^{-3}$
Highway	4	59	22,126	1.11 ± 0.01	$(2.36 \pm 0.12) \cdot 10^{-3}$
Generalized Highway	3	59	22,421	1.12 ± 0.02	$(3.32 \pm 0.24) \cdot 10^{-3}$
DGM	2	50	22,459	2.15 ± 0.02	$(2.04 \pm 0.13) \cdot 10^{-3}$

6.4.2. Discussion.

The capacity-normalized results show, first of all, that the training time is determined by the number of parameters and not the architecture. Indeed, as expected, all models in the capacity-normalized experiment need approximately the same time to be trained. However, we still observe that architectural differences lead to substantial variation in performance across the three problems. In the Black–Scholes setting, highway-type architectures achieve the highest accuracy, indicating that gating mechanisms are well suited for this problem; this is largely consistent with the results of the previous subsections (see Figure 17). On the other hand, for the more complex Heston model, the simple MLP yields the lowest error, suggesting that the inductive biases of gated or residual architectures may be overly restrictive under a fixed parameter budget. In addition, for the implied-volatility problem, DGM and MLP architectures perform similarly and outperform residual-type models.

Our results show that there is no single best neural network architecture for all problems: each type of network works well for some tasks but not for others. Importantly, even when all models have the same number of parameters, the differences in performance did not disappear. This means that the results are not caused by some models being bigger or having more capacity. Instead, the differences come from the way each architecture is built.

Table 21. Performance of MLP, highway, and DGM architectures on real implied volatility data from OptionMetrics.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
MLP	3	150	46,051	0.12	$3.44 \cdot 10^{-4}$
Residual	3	50	5501	0.12	$5.73 \cdot 10^{-4}$
Highway	3	50	10,801	0.15	$3.80 \cdot 10^{-4}$
Generalized Highway	3	50	16,101	0.13	$3.89 \cdot 10^{-4}$
DGM	3	50	32,757	0.21	$4.64 \cdot 10^{-4}$

6.5. Application to real data and comparison with other methods

In order to complete the analysis, we evaluated the architectures on real implied volatility data obtained from the OptionMetrics database. Each daily implied volatility surface is given on a fixed grid of *maturities*

$$\{10,\ 30,\ 60,\ 91,\ 122,\ 152,\ 182,\ 273,\ 365\} \text{ days},$$

and *put deltas*

$$\{-90,\ -80,\ -70,\ -60,\ -50,\ -40,\ -30,\ -20,\ -10\}.$$

Each column of the dataset corresponds to one maturity–delta pair. We train the architectures with the same layers and nodes as in Table 15; for the MLP architecture, we consider the best one. Table 21 reports the results for all five architectures on the real implied volatility data. The performance on real implied volatility data is highly consistent with the synthetic experiments shown in Table 15. In both cases, the MSE falls in the range of 10^{-3} to 10^{-4}. This indicates that the real dataset, although noisy, does not significantly increase the overall approximation difficulty compared to the synthetic implied-volatility surfaces. The ranking of architectures also follows similar patterns. The real-market experiment therefore provides an important robustness check: despite data imperfections, neural architectures achieve error levels comparable to the synthetic benchmark.

Finally, it is useful to compare the neural-network approaches with classical non-learning methods for computing implied volatility. Numerical techniques such as Jäckel's algorithm, or Chebyshev interpolation can achieve extremely small absolute volatility errors (from about 10^{-6} to 10^{-12}) when applied to well-behaved price inputs; see Glau et al. (2019). These methods are highly precise in controlled settings but do not directly approximate the entire volatility surface across strikes and maturities; instead they apply different schemes depending on the regime of the input (e.g. deep in-the-money, *etc*). Neural networks, by contrast, aim to learn the full surface in a single model, capturing complex patterns across both dimensions. As a result, the neural networks considered in this work reach mean absolute volatility errors around 10^{-2}, reflecting the intrinsic difficulty of modelling a more irregular and high-dimensional function. This comparison highlights that, while classical solvers provide a benchmark for precision, neural networks handle a broader approximation problem.

7. Conclusion

The empirical results obtained in this work allow us to draw several practical conclusions regarding the choice of neural network architectures for option pricing and implied volatility computation:

- Across the Black–Scholes, Heston, and transformed implied volatility problems, we observe that increasing the number of parameters generally improves accuracy when combined with an appropriate architecture. Larger models typically achieve lower MSEs, although the rate of improvement depends strongly on the underlying problem. However, they also require more training time.
- Across all architectures, training and validation losses evolve consistently, indicating stable optimization and good generalization without evidence of overfitting. Although validation losses exhibit higher variance, their overall behaviour closely follows the training performance.
- The validation loss was often lower during training than the training loss. This indicates that we could improve the accuracy of the networks by training them for more epochs. In Liu, Oosterlee, and Bohte (2019),

the results on the Heston pricing problem were more accurate, training for 3000 cycles compared to 200 in this work, while using an identical configuration for the MLPs.

- We observe that highway-type network architectures outperformed the other networks for the Black–Scholes problem, when we compared the MSE relative to the training time. This was consistent also in the capacity-normalized experiment. The same observation is true for the Heston model, but was not consistently observed in the capacity-normalized experiment.
- In the transformed implied volatility problem, the generalized highway network also performed well, but was outperformed by the no-recurrence DGM architecture. Interestingly, this architecture did not perform competitively on the pricing problems, highlighting that architectural benefits are task-dependent.
- When training networks on the implied volatility dataset, we found poor performance for all networks, and no improvements when using more complex architectures. When transforming the dataset as explained in Section 5.2, the prediction accuracy increased significantly, and we found similar results in network architecture performance compared to the Black–Scholes and Heston pricing problems. This observation suggests that the training of networks is highly dependent on the formulation and the dataset. When training neural networks, we should try to ensure that gradients are bounded. A possible way to do this is to transform the input variables so that the gradients with respect to them do not approach zero or infinity.
- We found that for the considered problems, the recurrent behaviour of the DGM networks did not yield benefits when compared to the generalized highway networks. Instead, the computational complexity caused by the increase of operations inside the DGM layers made it less attractive in terms of relative performance to off-line required computational power than the generalized highway network.
- The evaluation on real implied volatility data shows that the architectures maintain error magnitudes and relative performance patterns comparable to the synthetic data experiments, indicating that the task-dependent behaviours observed throughout the study persist under real-market conditions.

Acknowledgements

JPH gratefully acknowledges the hospitality at the Financial Engineering & Mathematical Optimization Lab of the NTUA where this project was initiated.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

AP gratefully acknowledges the financial support from the Hellenic Foundation for Research and Innovation [grant number HFRI-FM17-2152].

Data availability statement

Most of the data used in this research are artificial and can be made available upon request. The market data of Subsection 6.5 are available from OptionMetrics.

Notes on contributors

Serena Della Corte is an Assistant Professor at Leiden University and was previously a Postdoctoral Researcher at the Delft Institute of Applied Mathematics, Delft University of Technology. She received her PhD in Mathematics from TU Delft in 2025. Her research lies at the interface of applied probability, stochastic analysis, partial differential equations, and mathematical finance. Her work focuses on large deviations, Hamilton-Jacobi and Hamilton-Jacobi-Bellman equations, stochastic processes, viscosity solutions, and machine-learning methods for option pricing and calibration.

Laurens Van Mieghem has earned his B.S. and M.S. degrees with honours in Applied Mathematics at Delft University of Technology. His primary field of interest is derivatives pricing. He is currently active as a quantitative trader at a trading firm.

Antonis Papapantoleon is a Professor of Mathematical Finance at the Delft Institute of Applied Mathematics, TU Delft, and the "Machine Learning in Trading" Lead at the Delft FinTech Lab. He is also an Affiliated Researcher at the Institute of Applied and

Computational Mathematics, FORTH. He previously held positions at the National Technical University of Athens, TU Berlin, and the University of Mannheim, and received his PhD in Mathematics from the University of Freiburg. His research interests include limit theorems for stochastic systems, applications of Lévy processes in finance, term structure and LIBOR modelling, systemic risk, model-free methods in finance, and applications of machine learning in finance.

Jonas Papazoglou-Hennig is an Assistant Vice President - Risk Manager at DWS, the largest German asset management firm. He is dealing primarily with quantitative risk management approaches for the fiduciary investment market and liquidity risk. He holds a Master of Science in Mathematics from Technische Universität München (TUM) and a Master of Science in Applied Mathematics from the Ecole Polytechnique Federale de Lausanne (EPFL). He was a visiting research intern at the Financial Engineering and Mathematical Optimization Lab (FEMO Lab) of the National Technical University of Athens (NTUA). Furthermore, he is a certified quantitative finance and risk practitioner, holding the Certificate of Quantitative Finance (CQF).

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, et al. 2015. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." https://www.tensorflow.org/.

Al-Aradi, A., A. Correia, D. Naiff, G. Jardim, and Y. Saporito. 2018. "Solving Nonlinear and High-Dimensional Partial Differential Equations via Deep Learning." Preprint, arXiv:1811.08782.

Becker, S., P. Cheridito, and A. Jentzen. 2019. "Deep Optimal Stopping." *Journal of Machine Learning Research* 20:Paper No. 74, 25.

Black, F., and M. Scholes. 1973. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy* 81 (3): 637–654. https://doi.org/10.1086/260062.

Boudabsa, L., and D. Filipović. 2022. "Machine Learning with Kernels for Portfolio Valuation and Risk Management." *Finance & Stochastics* 26 (2): 131–172. https://doi.org/10.1007/s00780-021-00465-4.

Buehler, H., L. Gonon, J. Teichmann, and B. Wood. 2019. "Deep Hedging." *Quantitative Finance* 19 (8): 1271–1291. https://doi.org/10.1080/14697688.2019.1571683.

Capponi, A., and C. Lehalle. 2023. *Machine Learning and Data Sciences for Financial Markets: A Guide to Contemporary Practices*. Cambridge: Cambridge University Press.

Cuchiero, C., W. Khosrawi, and J. Teichmann. 2020. "A Generative Adversarial Network Approach to Calibration of Local Stochastic Volatility Models." *Risks* 8 (4): 101. https://doi.org/10.3390/risks8040101.

Delft High Performance Computing Centre (DHPC). 2022. "DelftBlue Supercomputer (Phase 1)." https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1.

Eberlein, E., K. Glau, and A. Papapantoleon. 2010. "Analysis of Fourier Transform Valuation Formulas and Applications." *Applied Mathematical Finance* 17 (3): 211–240. https://doi.org/10.1080/13504860903326669.

Eckstein, S., and M. Kupper. 2021. "Computation of Optimal Transport and Related Hedging Problems via Penalization and Neural Networks." *Applied Mathematics & Optimization* 83 (2): 639–667. https://doi.org/10.1007/s00245-019-09558-1.

Fang, F., and C. W. Oosterlee. 2008. "A Novel Pricing Method for European Options Based on Fourier-Cosine Series Expansions." *SIAM Journal on Scientific Computing* 31 (2): 826–848. https://doi.org/10.1137/080718061.

Filipović, D. 2009. *Term-Structure Models: A Graduate Course*. Berlin: Springer.

Gatheral, J. 2006. *The Volatility Surface: A Practitioner's Guide*. Hoboken, NJ: Wiley.

Glau, K., P. Herold, D. B. Madan, and C. Pötz. 2019. "The Chebyshev Method for the Implied Volatility." *Journal of Computational Finance* 23 (3): 1–31.

Glorot, X., and Y. Bengio. 2010. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, edited by Y. W. Teh and M. Titterington, 249–256. PMLR.

He, K., X. Zhang, S. Ren, and J. Sun. 2015. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification." In *2015 IEEE International Conference on Computer Vision (ICCV)*, 1026–1034. IEEE Computer Society.

He, K., X. Zhang, S. Ren, and J. Sun. 2016. "Deep Residual Learning for Image Recognition." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. IEEE Computer Society.

Heston, S. L. 1993. "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options." *Review of Financial Studies* 6 (2): 327–343. https://doi.org/10.1093/rfs/6.2.327.

Horvath, B., A. Muguruza, and M. Tomas. 2021. "Deep Learning Volatility: A Deep Neural Network Perspective on Pricing and Calibration in (rough) Volatility Models." *Quantitative Finance* 21 (1): 11–27. https://doi.org/10.1080/14697688.2020.1817974.

Huré, C., H. Pham, A. Bachouch, and N. Langrené. 2021. "Deep Neural Networks Algorithms for Stochastic Control Problems on Finite Horizon: Convergence Analysis." *SIAM Journal on Numerical Analysis* 59 (1): 525–557. https://doi.org/10.1137/20M1316640.

Liu, S., C. W. Oosterlee, and S. M. Bohte. 2019. "Pricing Options and Computing Implied Volatilities Using Neural Networks." *Risks* 7 (1): 16. https://doi.org/10.3390/risks7010016.

McKay, M. D., R. J. Beckman, and W. J. Conover. 1979. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code." *Technometrics* 21 (2): 239–245.

Papazoglou-Hennig, J. 2020. *Elementary Stochastic Calculus, Application to Financial Mathematics, and Neural Network Methods for Financial Models*. Internship Notes. NTUA-TUM.

Ruf, J., and W. Wang. 2020. "Neural Networks for Option Pricing and Hedging: A Literature Review." *Journal of Computational Finance* 24 (1): 1–46.
Sherstinsky, A. 2020. "Fundamentals of Recurrent Neural Network (RNN) and Long short-term Memory (LSTM) Network." *Physica D: Nonlinear Phenomena* 404:132306, 28. https://doi.org/10.1016/j.physd.2019.132306.
Sirignano, J., and K. Spiliopoulos. 2018. "DGM: a Deep Learning Algorithm for Solving Partial Differential Equations." *Journal of Computational Physics* 375:1339–1364. https://doi.org/10.1016/j.jcp.2018.08.029.
Srivastava, R. K., K. Greff, and J. Schmidhuber. 2015. https://arxiv.org/abs/1505.00387.
Telgarsky, M. 2016. "Benefits of Depth in Neural Networks." In *29th Annual Conference on Learning Theory*, edited by V. Feldman, A. Rakhlin, and O. Shamir, 1517–1539. PMLR.
Van Mieghem, L. 2022. "Option Pricing Techniques Using Neural Networks." Master's Thesis, TU Delft.
Zhang, Z., S. Zohren, and S. Roberts. 2022. "Deep Learning for Portfolio Optimization." *The Journal of Financial Data Science* 2 (4): 8–20. https://doi.org/10.3905/jfds.2020.1.042.

Appendix. Summary of network architectures

This appendix presents a comprehensive summary of all neural network configurations employed in our experiments. For each architecture – the MLP, residual, highway, generalized highway, deep DGM, and no-recurrence DGM models – we report:

- the number of layers and nodes per layer,
- the activation functions,
- the total number of trainable parameters,
- the corresponding references to the relevant tables in the main text.

For all the architectures we used:

- loss = MSE,
- learning rate = 10^{-5},
- batch size = 64,
- optimizer = Adam,
- epochs = 200.

Table A1. Overview of feedforward and highway neural network architectures and corresponding hyperparameter configurations.

Architecture	Layers	Nodes	Activation	Initializer	Parameters	References
MLP	2, 3	50–500	ReLU	Glorot	2851–504,001	6, 7, 8, 9
Residual	3	50	*tanh*	Glorot	7951	11, 12, 20
Highway	3	50	*tanh*	Glorot	15,601	11, 12, 20
Generalized Highway	3	50	*tanh*	Glorot	23,251	11, 12, 20

Table A2. Overview of DGM variants and corresponding hyperparameter configurations.

Architecture	Layers	Nodes	Activation	Initializer	Parameters	References
DGM	3	50	*tanh*	Glorot Normal	33,459	14, 15, 16
Deep DGM	5	50	*tanh*	Glorot Normal	49,959	14, 15, 16
No-Recurrence DGM	3	50	*tanh*	Glorot Normal	31,059	14, 15, 16