Neural Networks with Asymptotics Control

Alexandre Antonov
Danske Bank

Michael Konikov
Numerix

Vladimir Piterbarg
NatWest Markets

August 27, 2020

Abstract

Artificial Neural Networks (ANNs) have recently been proposed as accurate and fast approximators in various derivatives pricing applications. ANNs typically excel in fitting functions they approximate at the input parameters they are trained on, and often are quite good in interpolating between them. However, for standard ANNs, their extrapolation behavior – an important aspect for financial applications – cannot be controlled due to complex functional forms typically involved. We overcome this significant limitation and develop a new type of neural networks that incorporate large-value asymptotics, when known, allowing explicit control over extrapolation. This new type of asymptotics-controlled ANNs is based on two novel technical constructs, a multi-dimensional spline interpolator with prescribed asymptotic behavior, and a custom ANN layer that guarantees zero asymptotics in chosen directions. Asymptotics control brings a number of important benefits to ANN applications in finance such as well-controlled behavior under stress scenarios, graceful handling of regime switching, and improved interpretability.

1 Introduction

Significant advances in machine learning (ML), deep learning (DL), and artificial neural networks (ANN or NN) in image and speech recognition fueled a rush of investigations as to how these techniques could be applied in finance in general, and in derivatives pricing in particular. Typical examples of this genre include [AK], [MG], [HMT], [FG]. The main idea of these papers is to use NNs to speed up slow function calculations. A typical procedure involves training the NN offline on a sample of learning points calculated from the "true" model (or, in pre-NN language, fitting a functional form defined by an NN to a sample of function values over a collection of function arguments, often multi-dimensional), and then using the NN as an approximation to the true model during on-line pricing and risk management calculations.

It has generally been observed that NNs, once trained, do a good job interpolating between the points they were trained on (fitted to). However, extrapolation behaviour beyond the range of training points is not controllable in a typical NN, due to their complex non-parametric nature.

Electronic copy available at: https://ssrn.com/abstract=3544698

The lack of control over extrapolation is a significant limitation of the ANN approximation approach in financial applications. Financial models often need to be evaluated at the values of input variables that are significantly different from the current market conditions. Changes of regime are common in financial markets. Input values in stress scenarios, required for sound risk management, routinely fall outside of the range of the training set, with unpredictable extrapolation. Interpretability of NNs, an important issue for financial regulation of ML models, suffers if the behavior of models is undefined in some regions of the parameter space. These are some of the many reasons to strive for better control of NN extrapolation in financial applications.

One of the possible solutions to this problem is, of course, sampling the input variable space widely enough so that any possible future value of input variables falls within the sample range (interpolation) and never outside (extrapolation). It is not hard to see that this is not a fully satisfactory solution as one does not know a-priori what future values will be required. Additionally, large ranges of input values need a large number of learning points to cover, slowing down learning. More importantly, using large ranges for input variables would likely make the fit for moderate, i.e. non-extreme, values of inputs worse, as the NN would try to balance the quality of fit between all the training points.

Fortunately, for many financial applications, in addition to being able to calculate function values for moderate values of inputs, asymptotics of these functions for large values of parameters are also known. This is true for e.g. the SABR volatility function ([MG], [HMT]) and the values of many types of derivatives in various models ([FG]). The aim of this paper is to demonstrate how the knowledge of asymptotics can be effectively translated into control over the extrapolating behaviour of NNs.

We propose the following two-step approach to approximating a multi - dimensional function while preserving its asymptotics. On the first step we find a control variate function that has the same asymptotics as the initial function. On the second step we approximate the residual, or the difference between the original function and the control variate, with a special ANN that has vanishing asymptotics in all, or some, directions.

The apparent simplicity of the plan hides a number of complications that we overcome in this paper. Specifically, we make two critical contributions – our main technical results – that make this program work. For step one, we show how to construct a universal control variate, a multi-dimensional spline that has the same asymptotics as the initial function. For step two, we design a custom NN layer that guarantees zero asymptotics in required directions, with a fine control over the regions where the NN interpolation is used and where the asymptotics kick in. We note that while the idea of using a control variate has also been explored by others, see. e.g. [KALN], building a universal control variate with defined asymptotics, and an NN that does not destroy the asymptotics by indiscriminate extrapolation, are the novel contributions of this paper.

In the technical appendix we provide extensive intuition on the Kolmogorov-Arnold representation theorem, a fundamental result underlying the construc-

Electronic copy available at: https://ssrn.com/abstract=3544698

tion of the standard feed-forward multi-layer NN. The interested reader will find a comprehensive constructive proof of this complicated theorem and a description of how the asymptotics of the initial function are obscured as it is transformed through the Kolmogorov-Arnold representation.

Multi-dimensional interpolation is not the only application of NNs in quantitative finance; papers [KS], [BGTW], [GR], [PHL] explore other applications that are beyond the scope of this paper. Still, they may benefit from some of the ideas presented here.

2 Neural Networks

There are countless introductions to Neural Networks so we keep ours brief. An NN is essentially a multi-dimensional function parametrization

$$\begin{aligned} x \to x^{(1)} &= \sigma\left(W^{(1)}x + b^{(1)}\right) \\ \to x^{(2)} &= \sigma\left(W^{(2)}x^{(1)} + b^{(2)}\right) \\ &\dots \\ F_\theta(x) = x^{(N)} &= W^{(N)}x^{(N-1)} \end{aligned}$$

where

- Each line in the scheme above is called a layer;
- Nonlinear functions σ are called activation functions and are normally fixed;
- The adjustable parameters for an NN are the weight matrices $W^{(i)}$ and bias vectors $b^{(i)}$;

We denote the NN parametrization as $F_\theta(x)$, where θ is the collection of all adjustable parameters over the layers, $\theta = \left\{W^{(n)}, b^{(n)}\right\}_{n=1}^{N}$. The parameters θ are determined by the so-called learning process (non-linear optimization in the pre-NN language):

> Given input points $\left\{x^{(p)}, y^{(p)}\right\}_{p=1}^{P}$ and a loss function, e.g.
>
> $$L(\theta) = \sum_{p=1}^{P}\left(F_\theta\left(x^{(p)}\right) - y^{(p)}\right)^2, \tag{1}$$
>
> find optimal parameters θ^* that minimize the loss function.

The non-linear optimization is performed numerically. The NN parametrization is well-suited for this purpose – the derivatives of the loss function with respect to the adjustable parameters, necessary for the optimization, can be

Electronic copy available at: https://ssrn.com/abstract=3544698

rapidly calculated using back-propagation (aka adjoint algorithmic differentiation). The resulting function $F^*_\theta(x)$ approximates the "conditional expectation" $\mathbb{E}(Y \mid X)$. If the points $y^{(p)}$ are the values of some function f at $x^{(p)}$, i.e. $y^{(p)} = f\left(x^{(p)}\right)$, the NN approximates the function itself, $F_\theta \simeq f$.

Formulating an interpolation/fitting problem as a Machine Learning/NN problem brings certain advantages, to a large extend owning to highly advanced tools for ML/NN developed for image recognition and similar problems. In particular, the NN toolkit is highly efficient and customizable, with fine-grained control over the search space for adjustable parameters, ability to define bespoke layers, easy availability of hardware (GPU) acceleration and massive parallelization in the cloud, and so on. Apart from advanced tools, popularity of NNs led to extensive theoretical research, and a number of desirable properties such as the universality of the approximation and guaranteed convergence given sufficient data have been established.

At the same time, there are certain drawbacks to using NN machinery for fitting. Numerically solving a non-linear highly multi-dimensional problem could be unstable with local minima, over-fitting, etc. Interpolation between fitted points in standard NNs could exhibit un-intuitive behavior even if the training points are well fit, especially for multi-layer NNs. Last but not least, there is no asymptotic (extrapolation) control in standard NNs. Addressing the last point is the main focus of this paper.

Theoretical basis for NN construction and practical usability can be traced to two theorems. One is Cybenko and Hornik (see [GC], [HSW]) that establishes the universality of approximation of a very general class of NNs. The other is the solution of Hilbert's 13th problem and its extensions by Kolmogorov and Arnold, see [BG]. In Appendix A we give intuitive but sufficient for practical application description of these theorems including a comprehensive proof of the Kolmogorov - Arnold result (see appendix A.3).

While both theorems are powerful theoretical results, we show in the Appendix A that information about the asymptotics is very hardly extractable from these theorems and even less from the underlying standard NNs.

3 Plan for Controlling Asymptotics

In this section, we outline our overall plan. Broadly, the approach consists of two steps.

We start with a function $f(x)$ that we want to approximate, while preserving its asymptotics. (It goes without saying that the asymptotics need to be known.) As the first step, we find a control variate function $S(x)$ that has the same asymptotics as $f(x)$. For multi-dimensional inputs – the most interesting case – these asymptotics could either be known in all directions, or only in some; naturally, as much information about the asymptotics as is available should be incorporated into the control variate function $S(x)$.

On step two, we approximate the residual function $R(x) = S(x) - f(x)$ with a special NN that has vanishing (zero) asymptotics in all, or some, directions.

Electronic copy available at: https://ssrn.com/abstract=3544698

The apparent simplicity of the plan hides a number of complications that we overcome in this paper. Specifically, we make two critical contributions that make this program work. For step one, we show how to construct a universal control variate, a multi-dimensional spline $S(x)$ that has the same asymptotics as $f(x)$, and that can be used in all situations. In specific applications, of course, if more is known about the function f, a better choice of the control variate could be available. For step two, we design a custom NN layer that guarantees zero asymptotics in all directions, with a fine control over the regions where the NN interpolation is used and where the asymptotics kick in.

4 Fixed Asymptotics Splines

In this section, we construct splines with given asymptotics in all directions. We start with one-dimensional splines.

4.1 One-Dimensional Cubic Spline

A spline or, more specifically, a cubic spline $S(x)$ is a piece-wise cubic polynomial between its nodes $\{h_i\}_{i=1}^N$, i.e.

$$S(x) = a_i + b_i x + c_i x^2 + d_i x^3 \quad \text{for } x \in [h_i,\, h_{i+1}).$$

For a C^2 cubic spline, the kind we are interested in using, the values, derivatives and the second derivatives are matched at the nodes, i.e.

$$\begin{aligned} S(h_i-) &= S(h_i+), \\ S'(h_i-) &= S'(h_i+), \\ S''(h_i-) &= S''(h_i+). \end{aligned}$$

These conditions are almost sufficient to fix all the spline coefficients. We only need two extra conditions — at the first node and the last one — to fully specify the functional form. The classical C^2 spline, the so-called natural spline, has zero second derivatives at the boundaries, i.e. at the first node and the last one. See a detailed description in Appendix B.1.

In Figure 1, we approximate the Black-Scholes option value, as a function of the log-spot (all other parameters fixed), using the natural spline. Clearly, the asymptotics are not captured.

Electronic copy available at: https://ssrn.com/abstract=3544698

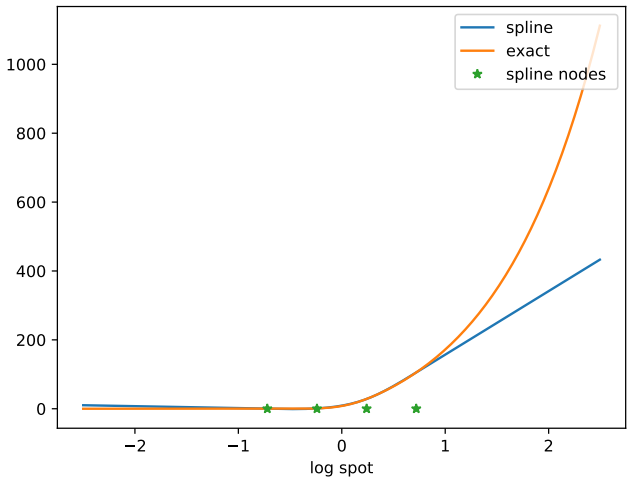

Figure 1: Fit of the natural spline to the Black-Scholes price as a function of log-spot, fitted at green dots.

4.2 One-Dimensional Spline with Asymptotics Control

Instead of using the natural boundary conditions, i.e. setting second derivatives to zero at the boundaries, we can fix the first derivatives at boundary points to arbitrary values. Clearly, this paves the way to controlling asymptotics.

Suppose that we know the behavior of the original, calculation-heavy function in its tails, i.e. $f(x) \simeq f_-(x)$ for $x < h_0$ and $f(x) \simeq f_+(x)$ for $x > h_{N+1}$ for large negative h_0 and large positive h_{N+1}. The recipe is simple. First, we expand the set of spline nodes to include points h_0 and h_{N+1} and make sure the spline passes through them, $S(h_0) = f_-(h_0)$ and $S(h_{N+1}) = f_+(h_{N+1})$. To finish, we specify first order derivatives at these new boundary points

$$S'(h_0) = f'_-(h_0), \quad S'(h_{N+1}) = f'_+(h_{N+1}).$$

This fully specifies the approximating function $\tilde{S}(x)$,

$$\tilde{S}(x) = \begin{cases} f_-(x), & x \le h_0, \\ S(x), & h_0 < x < h_{N+1}, \\ f_+(x), & h_{N+1} \le x. \end{cases}$$

(see also Appendix B.1). The function $\tilde{S}(x)$ is C^1 but not C^2 at h_0, h_{N+1}.

We will refer to the extended function $\tilde{S}(x)$ as a spline for brevity, even though technically it is not; we will also drop the tilde going forward. To emphasize the dependence of the spline on the nodes and values we sometimes use an extended notation

$$S(x \,|\, y, h) \tag{2}$$

Electronic copy available at: https://ssrn.com/abstract=3544698

for a spline passing through y_i at node h_i for $i = 1, \ldots, N$, and some asymptotic conditions.

4.3 One-Dimensional Spline with Asymptotics Control: Continuous Second Derivatives

It is also possible to construct a control variate function based on the spline as in the previous section, but with the continuous second derivative at the end points. In order to do that, we pick a point between h_0 and h_1, say $h_{\frac{1}{2}}$, and find, analytically, the value $S\left(h_{\frac{1}{2}}\right)$ such that the second derivative at h_0, $S''(h_0) = f''_-(h_0)$ is matched. The same is done for the right end point. According to our experiments, such control of second derivatives helps approximate "well behaved" functions better. See detailed description in Appendix B.1.

Figure 2 demonstrates the improvements in the fit to the Black-Scholes function when we control the asymptotics and second derivatives at the end points.

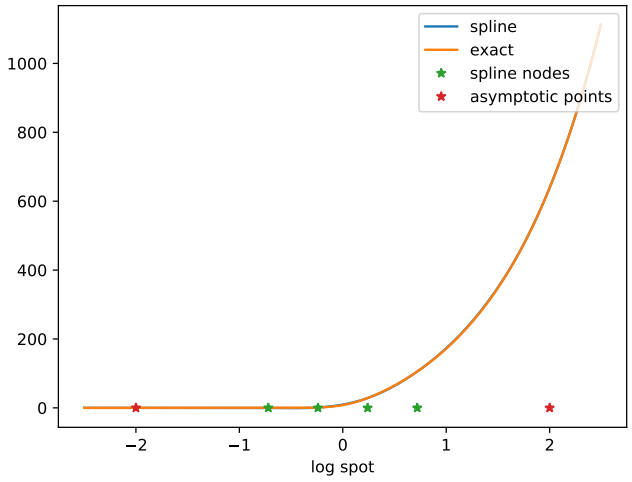

Figure 2: Fit of the asymptotics-controlled spline to the Black-Scholes price as a function of log-spot, fitted at green dots.

We note that in contrast to the natural spline, a general asymptotics - controlled spline is not linear in its values at the nodes $S(h_i)$ for $i = 1, \ldots, N$. However, in the special case of zero asymptotics, i.e. $y_0 = y'_0 = y''_0 = 0$ and $y_{N+1} = y'_{N+1} = y''_{N+1} = 0$, it becomes linear. Indeed, a set of splines $S_k(x)$ for $i = 1, \ldots, N$ with values $S_k(h_i) = \delta_{i,k}$ and zero asymptotics form a linear basis

Electronic copy available at: https://ssrn.com/abstract=3544698

for an arbitrary spline with zero asymptotics with some values y at the nodes

$$S(x) = \sum_{k=1}^{N} y_k S_k(x). \tag{3}$$

To prove this, it is sufficient to notice that the spline above takes values y_i at h_i for $i = 1, \ldots, N$, has continuous second derivatives and zero asymptotics.

Here and below in determining complexity of various operations for multi-dimensional spline calculations in Section 4.4 and Appendix B.2, we assume/use the following complexity for basic operations.

- Asymptotic function calculation time is

$$T_A \sim O(1).$$

- One-dimensional spline build/preparation consisting of calculating coefficients $\{a_i, b_i, c_i, d_i\}_{i=1}^{N}$ is linear in the number of the nodes, i.e.

$$T_B\, N \sim O(N),$$

  as the spline coefficients come from solving a tri-diagonal linear system (40) with N equations.

- One-dimensional spline value calculation consist of a search of the nearest spline node ($\sim O(\log N)$) and a cubic polynomial evaluation. In general, for a moderate N the search time is tiny w.r.t. to the cubic spline calculations, thus, we assume the value calculation time being

$$T_C \sim O(1)$$

4.4 Multi-Dimensional Splines with Asymptotics

For the multi-dimensional case we use a tensor product of one-dimensional splines with asymptotics control in each direction where they are available.

Denote by

$$h^{(d)} = \left\{h_1^{(d)}, \ldots, h_{N_d}^{(d)}\right\}$$

a one-dimensional grid for dimension d, and define a D-dimensional grid

$$H^{(D)} = h^{(1)} \otimes \cdots \otimes h^{(D)}$$

with points $H^{(D)}_{i_1,\ldots,i_D} = \left(h^{(1)}_{i_1}, \ldots, h^{(D)}_{i_D}\right)$.

Let the set of values to interpolate be given. We denote it by a D-dimensional tensor $Y^{(D)}_{i_1,\ldots,i_D}$. A D-dimensional spline

$$S^{(D)}(x) = S^{(D)}(x_1, \ldots, x_D)$$

Electronic copy available at: https://ssrn.com/abstract=3544698

is defined by the condition that the spline values at the corresponding grid points match the set of interpolated values

$$S^{(D)}\left(h^{(1)}_{i_1},\ldots,h^{(D)}_{i_D}\right) = Y^{(D)}_{i_1,\ldots,i_D}.$$

The grid $H^{(D)}$ and the values tensor $Y^{(D)}$ are parameters of the spline. When convenient we indicate that by using the full notation

$$S^{(D)}(x_1,\ldots,x_D) = S^{(D)}\left(x_1,\ldots,x_D \mid\ Y^{(D)},\, H^{(D)}\right).$$

Calculating the value of the spline at an arbitrary point proceeds inductively. We associate the initial spline $S^{(D)}$ with a series of $D-1$ dimensional splines

$$S^{(D-1)}_{n_D}(x) = S^{(D-1)}\left(x_1,\ldots,x_{D-1} \mid\ Y^{(D-1)}_{n_D},\, H^{(D-1)}\right) \tag{4}$$

with the grid $H^{(D-1)}_{i_1,\ldots,i_{D-1}}$ and values tensor $Y^{(D-1)}_{n_D}$ with elements

$$\left\{Y^{(D-1)}_{n_D}\right\}_{i_1,\ldots,i_{D-1}} = Y^{(D)}_{i_1,\ldots,i_{D-1},n_D},$$

with the running $D-1$ dimensional index $i_1,\ldots,i_{D-1}$ and fixed index n_D.

To calculate the spline value $S^{(D)}(x'_1,\ldots,x'_D)$ at a given point x' we execute the following algorithm:

A1. Calculate the values of the splines $S^{(D-1)}_{n_D}$ at the truncated point $(x'^{(D-1)} = x'_1,\ldots,x'_{D-1})$,

$$y^{D-1}_{n_D} = S^{(D-1)}_{n_D}(x'^{(D-1)}).$$

for all $n_D = 1,\ldots N_D$.

A2. Fit a one-dimensional spline $S^{(1)}\left(z \,|\, y^{(D-1)}, h^{(D)}\right)$ on the grid $h^{(D)}_{n_D}$ to values $y^{(D-1)}_{n_D}$ for $n_D = 1,\ldots N_D$.

A3. Attach asymptotics to the one-dimensional spline $S^{(1)}\left(z \,|\, y^{(D-1)}, h^{(D)}\right)$ if available, and calculate its coefficients.

A4. Set the value of the spline $S^{(D)}(x'_D)$ to be the value of the one-dimensional spline from the previous step,

$$S^{(D)}(x'_1,\ldots,x'_D) = S^{(1)}\left(x'_D \,|\, y^{(D-1)}, h^{(D)}\right).$$

A5. Reduced spline values from **A1** are calculated inductively: we repeat the same steps for each spline $S^{(D-1)}_{n_D}$ reducing them by one dimension, i.e. creating a set of $D-2$ splines

$$\left\{S^{(D-2)}_{i_{D-1},n_D}\right\}^{N_{D-1}}_{i_{D-1}=1}$$

for all n_D, build a one-dimensional spline etc.

Electronic copy available at: https://ssrn.com/abstract=3544698

The induction terminates when we reach one-dimensional splines $S^{(1)}_{n_2,\dots,n_D}$; the total number of them is $N_2 \times \cdots \times N_D$. Each of these splines is based on the same grid $H^{(1)} = h^{(1)}$ but, obviously, interpolates different values

$$\left\{ Y^{(1)}_{n_2,\dots,n_D} \right\}_{i_1} = Y^{(D)}_{i_1,n_2,n_3\dots,n_D}$$

for the running index i_1 and a fixed multi-index $(n_2, \dots, n_D)$. These one-dimensional splines

$$S^{(1)}_{n_2,\dots,n_D}(z) = S^{(1)}\left(z \,|\, Y^{(1)}_{n_2,\dots,n_D}, h^{(1)}\right) \tag{5}$$

are equipped with the corresponding asymptotics and can be prepared for repeated calculations up front by computing and storing coefficients $\{a_i, b_i, c_i, d_i\}_{i=1}^{N_1}$ for each multi-index $n_2, \dots, n_D$.

To estimate the overall calculation effort we will use the one-dimensional spline costs presented in Section 4.3. As shown in Appendix B.2, the total multi-dimensional spline calculation effort is

$$T_B\, N_1 \cdots N_D + (T_B + T_C)\, M_p\, N_2 \dots N_D + 2\, T_A\, M_p\, N_3 \cdots N_D \tag{6}$$

Spline calculations can be relatively slow for a large number of nodes and dimensions. However, the purpose of this multi-dimensional spline is to be a *control variate* to control asymptotics and thus only a moderate number of nodes[1] is required, resulting in a fast scheme.

It is easy to see that the spline values are *not* linear in the node values. However, for a special case of zero asymptotics, the spline becomes linear. Indeed, from the one-dimensional basis representation (3), we can deduce that our initial multi-dimensional spline can be presented as

$$\begin{aligned} S^{(D)}(x) &= S^{(D)}\left(x_1, \dots, x_D \,|\, Y^{(D)}, H^{(D)}\right) \\ &= \sum_{i_1,\dots,i_N} Y^{(D)}_{i_1,\dots,i_N} S\left(x_1 \,|\, \delta^{(i_1)}, h^{(1)}\right) \cdots S\left(x_D \,|\, \delta^{(i_D)}, h^{(D)}\right), \end{aligned} \tag{7}$$

where the vector $\delta^{(i)}$ has Kronecker elements $\delta_{i,k}$. In other words, the basis functions is a product of splines $S\left(x_d \,|\, \delta^{(i_d)}, h^{(d)}\right)$ which satisfy $S\left(h^{(d)}_i \,|\, \delta^{(i_d)}, h^{(d)}\right) = \delta_{i,k}$. To prove that this formulation is coherent with the algorithm above, let us represent the reduced splines (4) using (7) ,

$$\begin{aligned} S^{(D-1)}_{n_D}(x_1, \dots, x_{D-1}) = &\sum_{i_1,\dots,i_{N-1}} Y^{(D)}_{i_1,\dots,i_{D-1},n_N} \\ &\times S\left(x_1 \,|\, \delta^{(i_1)}, h^{(1)}\right) \cdots S\left(x_{D-1} \,|\, \delta^{(i_{D-1})}, h^{(D-1)}\right). \end{aligned} \tag{8}$$

[1] We use three nodes per dimension in four dimensions for our test results below.

Electronic copy available at: https://ssrn.com/abstract=3544698

Then, the one-dimensional spline from **A4** will have the values in its nodes $h_{n_D}^{(D)}$ from (8). Using the one-dimensional decomposition (3) we obtain

$$S(x) = \sum_{i_D=1}^{N_d} S_{n_D}^{(D-1)}(x_1, \ldots, x_{D-1}) S\left(x_D \,|\, \delta^{(n_D)}, h^{(D)}\right). \tag{9}$$

Substituting the expression (8) into the above decomposition and rearranging the summation, we obtain our initial conjecture (7).

Figures 3, 4, 5, 6 demonstrate the asymptotics control effect with true function being the Black-Scholes price for $T = 1$, $K = 100$ as a function of log-spot and log-volatility.

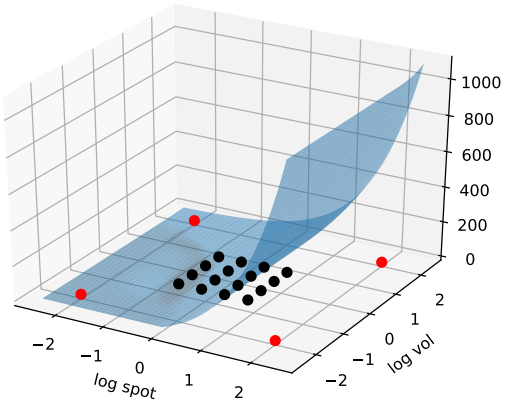

Figure 3: Blue surface: 2D asymptotic spline approximation; red dots: corners of the asymptotic region; dark blue dots: spline nodes (a 4×4 grid).

Electronic copy available at: https://ssrn.com/abstract=3544698

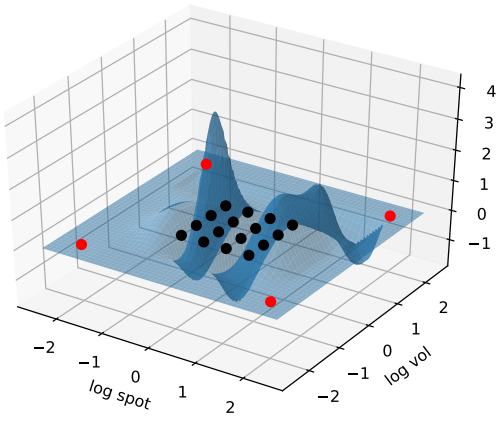

Figure 4: Blue surface: Difference between the 2D asymptotic spline and the true function; red dots: corners of the asymptotic region; dark blue dots: spline nodes.

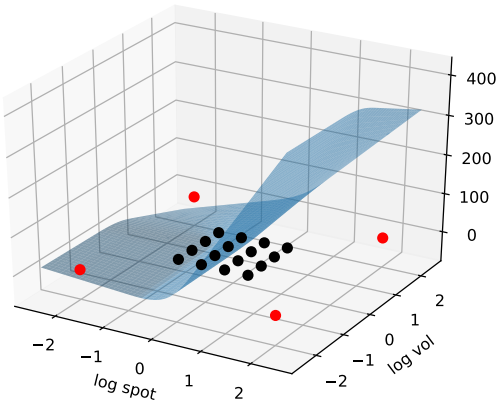

Figure 5: Blue surface: 2D natural spline approximation; red dots: corners of the asymptotic region; dark blue dots: spline nodes.

Electronic copy available at: https://ssrn.com/abstract=3544698

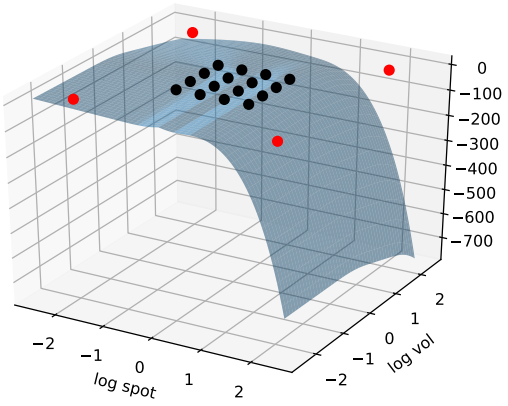

Figure 6: Blue surface: Difference between the 2D natural spline and the true function; red dots: corners of the asymptotic region; dark blue dots: spline nodes.

Electronic copy available at: https://ssrn.com/abstract=3544698

5 Constrained Radial Layer (CRL)

5.1 Gaussian Radial Layer

Using a control variate function, be that a multi-dimensional asymptotics-controlled spline from the previous section or a bespoke one tailored to a particular problem at hand, reduces the problem of fitting an arbitrary function to that of fitting a function with vanishing (zero) asymptotics. In this section we develop a NN that respects vanishing asymptotics outside a given domain, while fitting a set of function values for input values within a smaller domain.

To achieve this we propose to use the following NN representation of the function with vanishing/zero asymptotics,

$$f_G(x) = \sum_{i=1}^{N} \lambda_i \, G\left(x \,|\, c^{(i)}, w^{(i)}\right), \tag{10}$$

where $G\,(x \,|\, c, w)$ is a Gaussian "bell", or kernel, with a centre c (D-dim vector) and a width w ($D \times D$-dim matrix),

$$G\,(x \,|\, c, w) = \exp\left(-\frac{1}{2}(x-c)^T (w^{-1})^T \, w^{-1}(x-c)\right). \tag{11}$$

Sometimes these functions are called radial functions, so we call the NN layer with Gaussian kernels the (Gaussian) Radial Layer.

It is worth noting that in our numerical experiments we use a diagonal matrix

$$w = \mathrm{diag}\,(w_1, \ldots, w_D),$$

so that (11) becomes

$$G\,(x \,|\, c, w) = \exp\left(-\frac{1}{2}\sum_{d=1}^{D}(x_d - c_d)^2 / w_d^2\right). \tag{12}$$

The decomposition (10) is local, i.e. each Gaussian bell is limited in space. Gaussian kernels are quite similar to splines with zero asymptotics (7) but bells have adjustable irregular centres while the splines must have nodes on a regular rectangular grid.

A numerical solver, such as the standard ADAM solver from Keras in our experiments, changes the parameters of the function $f_G(x)$ to match the target values at the learning points until convergence.

To achieve zero asymptotics outside a certain domain, or, equivalently, (near) zero values and first-order derivatives on the boundary of that domain for the fitting function, it is important to control the centres and widths of Gaussian kernels. In simple terms, they should be deep inside the asymptotic region so that all Gaussian kernels have small values at the boundary.

Such control can be achieved in a number of different ways. For example, the loss function (1) can be complemented with terms that penalize bells that

Electronic copy available at: https://ssrn.com/abstract=3544698

get close or over the asymptotic bounds. We, however, find that a superior approach is to use a mapping function to make sure kernels' centres and widths are within the region of our choosing, thus defining what we call a Constrained Radial Layer (CRL). Let us elaborate.

5.2 Constraints on Centres

For brevity we denote $c = c^{(1)}$ to be the center of the first kernel in the sum (10); all the others are handled identically. Here $c = (c_1, \ldots, c_D)$ with D being the dimension of the problem, i.e. the number of input arguments to the function we fit. Suppose our goal to make sure c_1, the first coordinate of the centre, is always in the interval $[l, u]$, for $l < u$ (all the other coordinates are handled similarly). For that we introduce the following mapping function

$$c_1 = \tau_{l,u}(\bar{c}_1),$$

where

$$\tau_{l,u}(x) = \frac{ue^x + le^{-x}}{e^x + e^{-x}}. \tag{13}$$

The function $\tau_{l,u}(x)$ is monotonically increasing and maps $\mathbb{R}$ to $[l, u]$,

$$\tau_{l,u}(\cdot) : \mathbb{R} \to [l, u].$$

We apply similar transformations to all coordinates of all centres $c^{(i)}$, $i = 1, \ldots, N$, and replace the parametrization (10) with

$$f_G(x) = \sum_{i=1}^{N} \lambda_i \, G\left(x \,|\, \tau\left(\bar{c}^{(i)}\right), w^{(i)}\right), \tag{14}$$

where we have abused the notations somewhat. The optimizer will now iterate over variables $\bar{c}^{(i)}$, $i = 1, \ldots, N$, with our mapping ensuring that Gaussian kernel centres $c^{(i)} = \tau\left(\bar{c}^{(i)}\right)$, $i = 1, \ldots, N$, are always within the specified bounds.

5.3 Interpolation and Minimum Width

Consider a hypothetical case where there are as many terms in (10) as there are training, or fitting, points x. During non-linear optimization/fitting, there is nothing in what we described so far that stops the optimizer from placing each kernel precisely over each learning point, with as narrow a width as it wants. While the fitted functions will reproduce the values of the objective at the learning points exactly, the resulting "spiky" fitted function is unlikely to interpolate faithfully between the learning points. This is of course a classical case of over-fitting, but even if we have fewer kernels than the learning points, the danger of getting an overly spiky fit, and poor interpolation, remains.

To discourage such behavior and, ultimately, to control interpolation between the learning points, we introduce the minimum width $w_{\min} > 0$ and require that

$$w_d^{(i)} \geq w_{\min}, \quad i = 1, \ldots, N, \quad d = 1, \ldots, D. \tag{15}$$

Electronic copy available at: https://ssrn.com/abstract=3544698

5.4 Constraints on Widths

Constraining centres of Gaussian kernels to a certain domain is not sufficient to guarantee small values of kernels on a given asymptotic boundary. This is so because widths, if left unchecked during optimization, can grow large enough for kernels to have non-vanishing values at the asymptotic boundary.

We apply similar mapping technique to control the widths in the context of the diagonal specification (12).

Again we demonstrate the technique on one coordinate of the parameters of one of the kernels. Consider $c_1 = c_1^{(1)}$ and $w_1 = w_1^{(1)}$, where $c^{(1)} = (c_1^{(1)}, \dots, c_D^{(1)})$ and $w^{(1)} = (w_1^{(1)}, \dots, w_D^{(1)})$. Let l^{asm} and u^{asm} be the asymptotic bounds in this coordinate such that we want the kernel to be close to zero at those bounds (in the chosen coordinate). Let us introduce a coefficient α; the value of α is given by the requirement that the value of the kernel at x such that $(x - c_1)/w_1 > \alpha$ is negligible. We typically use $\alpha = 3$ as in "the probability of a Gaussian being three standard deviations away from its mean is small".

We further define the distance to the nearest asymptotic bound, and a scaled upper bound, as

$$\Delta = \min(c_1 - l^{asm}, u^{asm} - c_1), \quad u^{scaled} = \Delta/\alpha. \tag{16}$$

Then we apply the mapping function (15) to make sure w_1 is always in the range $[w_{\min}, u^{scaled}]$, where $w_{\min}$ is the minimum width as introduced in (15) and u is given by (16):

$$w_1 = \tau_{w_{\min}, u^{scaled}}(\bar{w}_1).$$

As mentioned, the same transformation is applied to all coordinates of widths of all kernels. The final functional form of the fitting function is then given by

$$f_G(x) = \sum_{i=1}^{N} \lambda_i \, G\left(x \,|\, \tau\left(\bar{c}^{(i)}\right), \tau(\bar{w}^{(i)})\right). \tag{17}$$

The formula (17) defines the Constrained Radial Layer (CRL).

Practical applications of our method that consists of using a fixed asymptotics spline and CRL are best demonstrated with examples. Before proceeding, let us briefly discuss the usage of CRL NNs for calculating conditional expectations.

6 Calculating Conditional Expectations

The utility of asymptotically-controlled CRL NNs is not limited to approximating explicitly-calculable functions. It is relatively straightforward to apply the same method to the problem of calculating conditional expectations, a calculation that is at the heart of the American Monte-Carlo and, more generally, Monte-Carlo based methods for solving high-dimensional PDEs and optimal control problems in finance and elsewhere, see e.g. [PHL].

Electronic copy available at: https://ssrn.com/abstract=3544698

In all of these methods the objective is to calculate

$$f(x) = \mathbb{E}\,(Y\,|X = x\,) \tag{18}$$

given simulated values of random variables $\{X_p\}_{p=1}^{M_p}$ and $\{Y_p\}_{p=1}^{M_p}$. It is not uncommon to know asymptotics of the solution function $f(x)$. If this is the case, we can use this knowledge to our advantage in a way similar to our general recipe in Section 3, with small adjustments. Specifically, the knots for the control variate spline $S(x)$ cannot be inferred directly from the simulated values $\{Y_p\}_{p=1}^{M_p}$. This obstacle is easily overcome. First, we calculate an approximation to $f(\cdot)$ in (18) using simple, standard regression techniques. The key point is that this regression need not to be very accurate. Once we have an approximation to $f(x)$, we can calculate its values on some representative grid and fit the control variate spline $S(x)$ extended with the (known) asymptotics. The learning set is then defined as

$$\{Y_p - S(X_p)\}_{p=1}^{M_p}\,, \tag{19}$$

and is used to train the CRL layer using the mean-squared error as the target to minimize.

As an alternative to the last step, we can least-square regress the learning set (19) against the value of the collection of multi-dimensional splines with zero asymptotics (7) at the X-points. Regressing against known basis functions is, of course, a much faster operation than fitting a non-linear CRL layer. What is gained in speed, however, is often lost in flexibility, and the merits of the two approaches should be judged against the specific requirements of a particular problem.

7 Experiments with Black-Scholes

We start fleshing out our ideas on the problem of fitting the Black-Scholes function of spot and volatility with other parameters being fixed, expiry $T = 1$, and strike $K = 100$. We translate the problem into the normalized log-spot, log-vol space

$$x_1 = \ln \frac{S}{K}, \quad y_1 = \ln \frac{\sigma}{\sigma_{\text{ref}}},$$

where we set $\sigma_{\text{ref}} = 20\%$ as a normalizing value.

For the first step, we use the asymptotics-controlled spline from Section 4 as the control variate, and use the NN to fit the difference. We define three regions:

- $[-1, 1] \times [-1, 1]$ – where the NN is trained;
- $[-3, 3] \times [-3, 3]$ – where the errors that we report later are measured;
- $[-2, 2] \times [-2, 2]$ – where the approximation is allowed to have non-zero values: outside of this box the function values are nearly zero.

Electronic copy available at: https://ssrn.com/abstract=3544698

The area $[-2,2] \times [-2,2] \setminus [-1,1] \times [-1,1]$ is the "no man's land" – there is no direct control as to what happens there but we expect our NN to behave well in this region. Figure 7 shows the three regions.

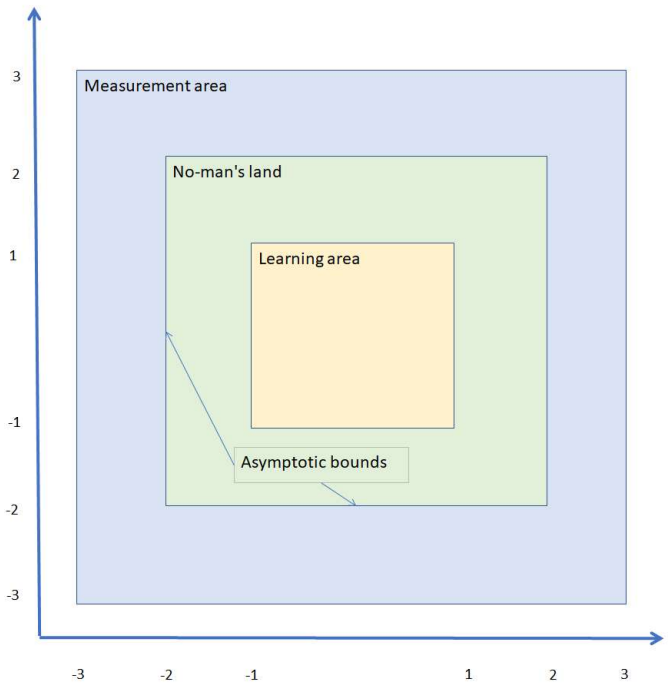

Figure 7: The three areas of interest

7.1 Constrained Radial Layer (CRL) for Black-Scholes

To fit the (de-asymptotized) Black-Scholes function, we use a two-layer NN. The first layer is a custom Constrained Radial Layer (CRL) with the activation function being bell-shaped Gaussian kernels (radial functions) $G(x \mid c, w)$ such that

- $x = (x_1, x_2) = (\ln(S/K), \ln(\sigma/\sigma_{\text{ref}}))$ is the input;
- c's are centres of the Gaussian kernels – related to biases of a NN layer;
- w's are widths of the Gaussian kernels – related to weights of a NN layer.

The second layer is a linear layer with no biases, to preserve zero asymptotics. The NN is then given by (it is in fact (17) as discussed in Section 5 but we

Electronic copy available at: https://ssrn.com/abstract=3544698

simplify the exposition for brevity)

$$f_G(x) = \sum_{i=1}^{N} \lambda_i \, G\left(x \,|\, c^{(i)}, w^{(i)}\right)$$

such that the parameters to calibrate are ($D = 2$ here):

- First layer: $D \times N$ centres c's and $D \times N$ widths w's;
- Second layer: N of λ's.

Following Section 5 we keep the centres of bells in the $[-1, 1] \times [-1, 1]$ area and control the widths such that the bells with a *triple* width are inside the $[-2, 2] \times [-2, 2]$ area with the help of the mapping functions $\tau(\cdot)$ as defined in (13). Again as in Section 5 we introduce the minimum width parameter to control the quality of interpolation; in the numerical experiments we set it to 0.05.

7.2 Test Results for CRL

We fit the function $BS(x) - S(x)$, with zero asymptotics for large $\|x\|$ by construction, in the learning area $[-1, 1] \times [-1, 1]$ over 40×40 points. We call this grid the learning grid. The test grid is the grid 300×300 over $[-3, 3] \times [-3, 3]$ box, which has roughly $\times 2$ density of the learning grid. We calculate MSE ($\|\cdot\|_2$) and MAE ($\|\cdot\|_1$) metrics over various grids as we increase the number of nodes in powers of 2: 2^n for $n = 0, \ldots, 10$. The number of iterations of the optimizer is proportional to the number of nodes of the NN we use for approximating. We calculate error metrics on the following grids:

- Learning points we use in the fit in $[-1, 1] \times [-1, 1]$, compared to the test grid, which is finer, in the same area. This measures interpolation error.
- Test grid in the whole area $[-3, 3] \times [-3, 3]$.

Figure 8 shows how the interpolation error in the learning box behaves as we increase the number of nodes.

Evidently, as we increase the number of nodes, the NN starts locating Gaussian kernels with small width over learning points. The fit over learning points is near-perfect but the interpolation between them suffers as the number of nodes increases – a classical example of over-fitting. It is clear that $2^6 = 64$ nodes gives a good fit and avoids over-fitting. Apart from choosing the number of nodes carefully, we can also control interpolation by increasing the minimum kernel width, i.e. using less spiky basis functions.

Electronic copy available at: https://ssrn.com/abstract=3544698

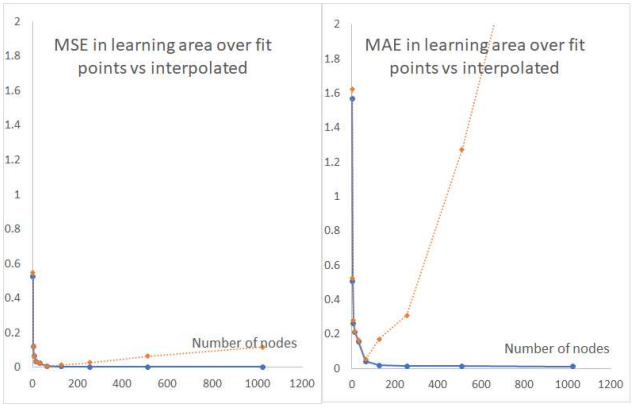

Figure 8: Errors of the fit over the learning points (blue solid line) vs. the testing points (yellow dotted line) in $[-1,1] \times [-1,1]$.

Figure 9 shows overall fit errors over the measurement area $[-3,3] \times [-3,3]$. They are the weighted average of fit errors in the learning area, no man's land area, and asymptotics area. One area where we do not directly control the quality of the fit is the no man's land area and this what drives the overall error. We observe that in this area the MSE is quite small while MAE is not. The MAE up to 250 nodes is driven by the extrapolation errors; incidentally, Gaussian kernels have a spiky form as we show later. For a large number of nodes, the interpolation spikes become dominant in the MAE. On the other hand, outside of the asymptotics box $[-2,2] \times [-2,2]$ the errors are tiny by construction.

Electronic copy available at: https://ssrn.com/abstract=3544698

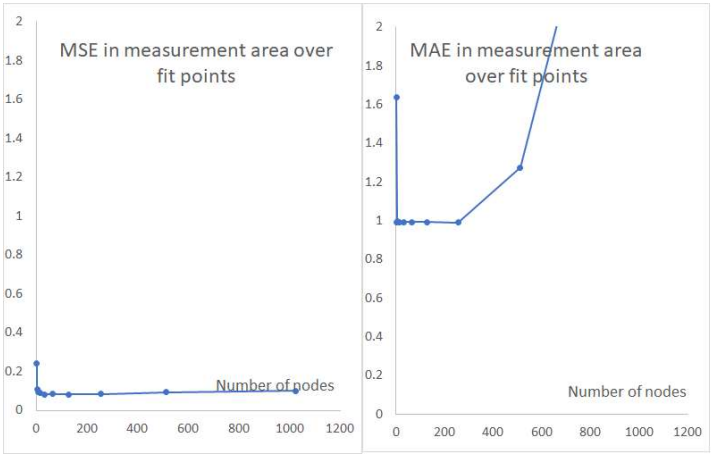

Figure 9: Errors of the fit over the measurement area $[-3, 3] \times [-3, 3]$.

7.3 Example Fits

Let us look at a number of graphs to understand the behavior of our NN with CRL better. Figure 10 shows the function we are trying to fit, the Black-Scholes value minus the control variate (asymptotic spline), with vanishing (zero) asymptotics in all directions. Clearly it is not an easy to fit function.

Electronic copy available at: https://ssrn.com/abstract=3544698

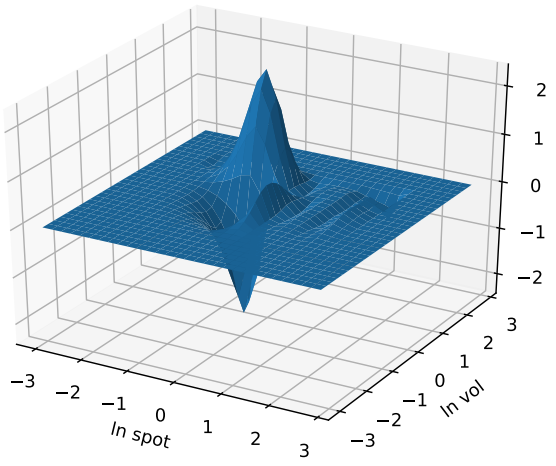

Figure 10: Black-Scholes formula minus the asymptotic spline.

The next few figures show the quality of the fit. We use 250 nodes in this example. Figure 11 shows the output of the NN fitted to the function of interest. Figure 12 shows the difference between the function and the fit. Additionally, Figure 13 shows the contours of the bells of the Gaussian kernels, to give an idea what the optimizer came up with for the fit.

We note the following features of our approximation procedure. We obtain a very good fit in the learning area, with differences at the learning points plotted in orange color. We observe relatively regular, but not controlled, behavior in the "no man's land". Finally, the fit has the correct vanishing, or zero, asymptotics outside the asymptotic bounds.

Electronic copy available at: https://ssrn.com/abstract=3544698

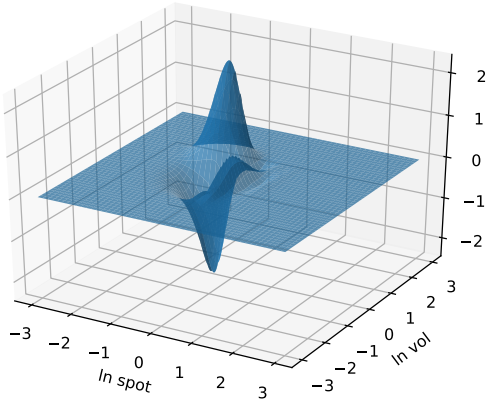

Figure 11: Replicating: Fit to the CRL.

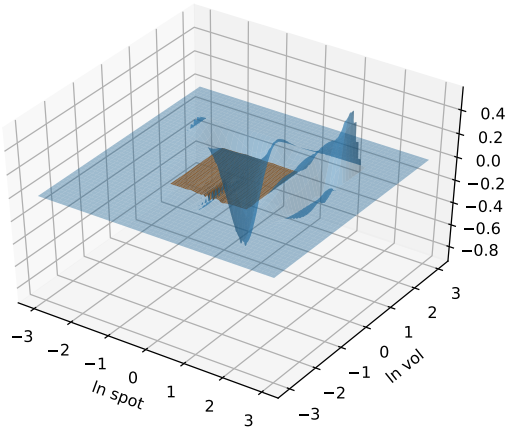

Figure 12: Difference: Actual minus fit to the CRL.

Electronic copy available at: https://ssrn.com/abstract=3544698

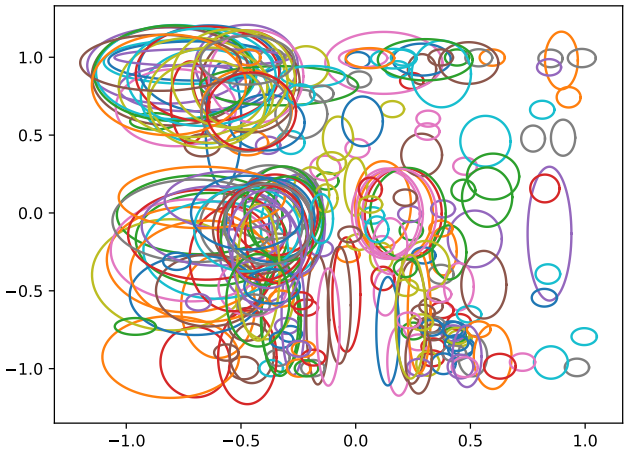

Figure 13: Bells and their widths – a sample fit.

7.4 Standard Layers

To demonstrate the superiority of the custom Constrained Radial Layer over the standard layers for our application, we perform similar fits using a standard NN architecture. We consider the following standard (non-custom) NN with 3 layers:

$$\begin{aligned} x \to x^{(1)} &= \sigma\left(W^{(1)}x + b^{(1)}\right), \quad 250 \text{ nodes} \\ \to x^{(2)} &= \sigma\left(W^{(2)}x^{(1)} + b^{(2)}\right), \quad 2 \text{ nodes} \\ F(x) &= W^{(3)}x^{(2)} \end{aligned}$$

We use either a Gaussian or sigmoid activation functions. The total number of parameters is comparable with the CRL NN. Figures 14 and 15 demonstrate the shape of the fitted function and the difference from the objective for Gaussian activation. Figures 16 and 17 show the same for sigmoid activation. It is not hard to observe that both choices are significantly inferior to the CRL NN and provide no control over extrapolation.

Below, we will observe a poor fit outside of the training area and relatively good one in the learning area (plotted in orange color).

Electronic copy available at: https://ssrn.com/abstract=3544698

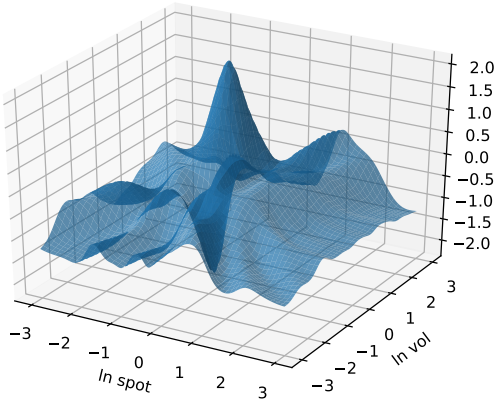

Figure 14: Replicating: Gaussian activation standard layer.

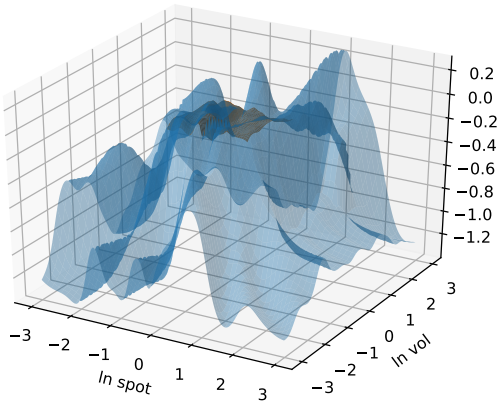

Figure 15: Difference: Gaussian activation standard layer. The learning region is plotted in orange color.

Electronic copy available at: https://ssrn.com/abstract=3544698

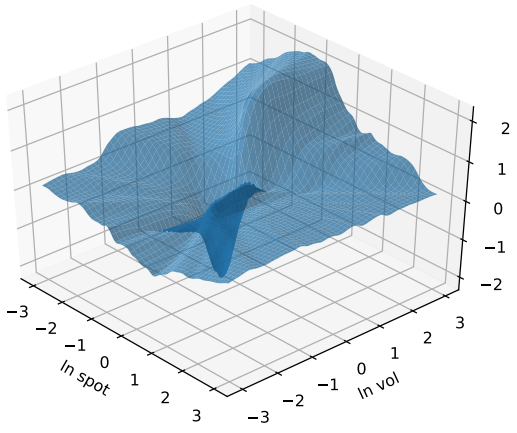

Figure 16: Replicating: sigmoid activation standard layer.

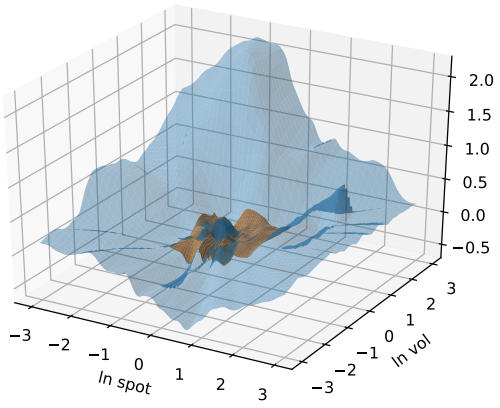

Figure 17: Difference: sigmoid activation standard layer. The learning region is plotted in orange color.

Electronic copy available at: https://ssrn.com/abstract=3544698

8 Experiments with SABR

To further demonstrate the merits of our approach, we consider the benchmark problem of fitting the implied volatility function of the SABR model, see [MG], [HMT]. We show that incorporating asymptotics into the approximation, per our program, dramatically improves the fit in the extrapolation region.

The standard parametrization of the SABR model is given by

$$dS(t) = \alpha(t)S(t)^{\beta}\, dW_1(t), \tag{20}$$

$$d\alpha(t) = \gamma\alpha(t)\, dW_2(t), \tag{21}$$

$$S(0) = S_0, \quad \alpha(0) = \alpha_0, \quad dW_1\, dW_2 = \rho\, dt, \tag{22}$$

where S_0 is the spot, α_0 is the initial level of volatility, β is the skew, ρ is the correlation between the spot process and the volatility process, and γ is the volatility of volatility. We denote by $\sigma_{\mathrm{N}}(S_0, \alpha_0, \beta, \rho, \gamma, K, T)$ the implied normal volatility for a European call (or put) option with strike K and expiry T in the SABR model (20)–(22) with the parameters $(S_0, \alpha_0, \beta, \rho, \gamma)$. Our goal is to fit the function $\sigma_{\mathrm{N}}(\cdot)$ with an ANN incorporating known asymptotics.

The focus of this paper is on how to incorporate known behavior of the approximated function for large values of (some of) the parameters into the fitting procedure, and not on deriving/revisiting asymptotics for the SABR model (see [AKS] for the latter). Hence, for brevity, we consider the case that can be treated exactly without approximations. This also helps us rigorously quantify errors of the NN approximations. Specifically, we set

$$\rho = 0$$

and use the exact analytical formulas for European option values in the SABR model with zero correlation from [AKS].

We apply a number of transformations to the original SABR parameters to reduce the dimensionality of the problem and normalize parameter ranges.

8.1 Re-Normalization

The initial volatility α_0 is not orthogonal to β in the sense that changing β would require a change in α_0 to keep the ATM implied volatility constant. Therefore, we re-parametrize (20)–(22) in terms of the approximate ATM log-normal volatility v. To this effect, we define

$$v = \alpha_0\, S_0^{\beta-1},$$

and introduce normalized processes

$$\bar{S}(t) = \frac{S(t)}{S_0}, \quad \bar{\alpha}(t) = \frac{\alpha(t)}{\alpha_0}.$$

Then

$$\begin{aligned} d\bar{S}(t) &= S_0^{-1} dS(t) = S_0^{-1}\alpha(t)S(t)^{\beta}\ W_1(t) = S_0^{-1}\alpha_0\bar{\alpha}(t)S(t)^{\beta}\ W_1(t) \\ &= v\,\bar{\alpha}(t)\,(S(t)/S_0)^{\beta}\ \ W_1(t), \end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3544698

leading to the re-normalized model

$$d\bar{S}(t) = v\,\bar{\alpha}(t)\bar{S}(t)^{\beta}\;dW_1(t), \tag{23}$$

$$d\bar{\alpha}(t) = \gamma\,\bar{\alpha}(t)\;dW_2(t), \tag{24}$$

$$\bar{S}(0) = \bar{S}_0 = 1, \quad \bar{\alpha}(0) = \bar{\alpha}_0 = 1. \tag{25}$$

European option prices in the original model are recovered from the normalized one via

$$\mathbb{E}\left[(S(T)-K)^{+}\right] = S_0\,\mathbb{E}\left[\left(\bar{S}(T)-\bar{K}\right)^{+}\right],$$

where

$$\bar{K} = K/S_0,$$

so the two are equivalent for our purposes. Given this normalization we use (23)–(25) instead of (20)–(22) from now on, and drop bars for brevity.

As we explain later, we will approximate SABR implied volatility for one fixed T at a time. It then follows that, for a fixed T (and with ρ fixed at 0) the implied normal volatility is a function of four arguments (v, β, γ, K),

$$\sigma_{\mathrm{N}}(\cdot) = \sigma_{\mathrm{N}}(v, \beta, \gamma, K).$$

8.2 Parameters

For testing, we set time to expiry to one, so $T = 1$. To demonstrate the flexibility of our approach, we control asymptotics in some but not all dimensions. We focus our testing on a configuration where we control asymptotics in the dimensions of the (approximate) at-the-money volatility v and the strike K only, and let the approximating NN extrapolate in β and γ dimensions as it may. This configuration corresponds loosely to the typical requirements of being able to apply spot and volatility stress scenarios while keeping the skew and volatility of volatility (slope and curvature of the volatility smile) near the "normal" levels.

For a couple of tests, we also include a configuration where we additionally control asymptotics in the γ direction to show that, unsurprisingly, controlling more asymptotics leads to a better fit.

As a reminder, we use the exact values of the SABR implied volatility function in the asymptotic region (available in the case of $\rho = 0$ that we consider) to simplify our exposition.

As dimension of our space grows (it was 2 for Black-Scholes, and 4 for SABR), it becomes less feasible to define training and validation samples as Cartesian product of grids in each variable, and we resort to sampling our space using random numbers. We sample four-dimensional vector of uncorrelated standard Gaussian random numbers, $(z_1, \ldots, z_4)$, and map those into some reasonable ranges for our variables. We simulate the (approximate) at-the-money volatility as

$$v = v_{\mathrm{ref}} \times \exp\left(v_{\mathrm{width}} z_1 - \frac{1}{2} v_{\mathrm{width}}^2\right),$$

Electronic copy available at: https://ssrn.com/abstract=3544698

with $v_{\text{ref}} = 20\%$ and $v_{\text{width}} = 0.75$, which for $z_1 = (-4, -3, -2, -1, 0, 1, 2, 3, 4)$ gives us $v = (1\%, 2\%, 3\%, 7\%, 15\%, 32\%, 68\%, 143\%, 303\%)$.

For the skew β, we simply map the standard Gaussian variate into a uniform one on $[0, 1]$ using the normal cumulative distribution function (CDF) $\Phi(\cdot)$,

$$\beta = \Phi(z_2)$$

that ensures that $\beta \in (0, 1)$.

For the volatility-of-volatility parameter γ mapping, we define the variance of $\bar{S}(T)$ to be a multiple (SABR, being a stochastic volatility process, has fat tails) of its approximate Gaussian variance, with the multiplier driven by γ:

$$V = v^2 T \left(1 + (V_{\text{mult}} - 1)\, \Phi(z_3)\right).$$

Here we set the maximum allowed value of the multiplier $V_{\text{mult}} = 4$, and use the approximation

$$V \approx \hat{V} = v^2 T \left(1 + \frac{\gamma^2 T}{2}\right)$$

to solve for γ:

$$\gamma = \sqrt{\frac{V - v^2 T}{\frac{1}{2} T^2 v^2}} = \sqrt{\frac{2\,(V_{\text{mult}} - 1)\, \Phi(z_3)}{T}}.$$

For the strike K, we sample from the Gamma distribution with the same mean and standard deviation as $\bar{S}(T)$, i.e. with the mean 1 and the standard deviation $\sqrt{\hat{V}}$, scaled by the factor $K_{\text{width}} = 0.2$ to concentrate the strikes around the region of interest. We note that the strike samples depend on realizations of the v and γ, and are given as follows,

$$K = \Gamma^{-1}\left(\Phi(z_4), g\right)/g,$$

where $\Gamma(z)$ is the Gamma CDF and

$$g = \left(K_{\text{width}}^2\, v^2 T \left(1 + \frac{\gamma^2 T}{2}\right)\right)^{-1}.$$

8.3 Bounds

As in Section 7, we define a number of regions. Since all the parameters are normalized to be of the same magnitude (all z_j are standard Gaussian), we use the same bounds in all dimensions, and set

- The learning bound $L_l = 1.0$ that defines the region where we train the NN on non-asymptotic values;
- The asymptotic bound $L_a = 1.5$ that defines the region outside of which we use asymptotics;

Electronic copy available at: https://ssrn.com/abstract=3544698

- The measurement bound $L_m = 2.0$ that defines the region where we measure performance/errors.

We then use $[-L_l, L_l]^4$, $[-L_a, L_a]^4$, $[-L_m, L_m]^4$ as the learning, asymptotic and measurement regions. Asymptotic bounds are only relevant for asymptotics-controlled dimensions which, for most of the tests, are the volatility v and the strike K.

Tables 1–3 show the corresponding standard, i.e. non-normalized, SABR parameter bounds. Note that the bounds for the strike correspond to the largest variance span.

	lower bound	upper bound
v	0.034	0.677
β	0.023	0.977
γ	0.369	2.421
K	0.538	1.605

Table 1: Learning bounds for standard (non-normalized) SABR parameters.

	lower bound	upper bound
v	0.016	1.432
β	0.001	0.999
γ	0.09	2.448
K	0.073	3.595

Table 2: Asymptotic bounds for standard (non-normalized) SABR parameters.

	lower bound	upper bound
v	0.008	3.032
β	0	1
γ	0.01	2.5
K	0	13.723

Table 3: Measurement bounds for standard (non-normalized) SABR parameters.

8.4 Asymptotic Splines and Constrained Radial Layers

The construction of asymptotic splines follows closely the program outlined in Sections 4.4 and 7. We use a rather sparse grid of 3 nodes in each of the four dimensions, at the normalized values of $\{-1, 0, 1\}^4$; we remind the reader that the point here is to control the asymptotics and not necessarily have a close fit inside the learning region, as that will be taken care of by the NN.

Electronic copy available at: https://ssrn.com/abstract=3544698

Having removed the asymptotics in the v and K dimension, we train the Constrained Radial Layer (CRL) NN, of the same structure as described in Section 7, to the residual.

8.5 Testing: Configuration

The training set is comprised of randomly generated (with a fixed seed) $10,000$ standard Gaussian 4-dimensional samples of $(z_1, \ldots, z_4)$ that are then converted to (v, β, γ, K) using mappings from Section 8.2. To make sure that the training and validation points stay within their prescribed regions, which are the learning and the measurements regions, we sample enough points to get $10,000$ of them inside the appropriate region while rejecting those that are outside. This procedure ensures that the learning region is sampled at sufficient granularity. Random sampling, in contrast to the regular 4-dimensional grid sampling, also helps avoid the curse of dimensionality.

The Constrained Radial Layer (CRL) NN we use to fit the de-asymptotized values has 300 nodes (Gaussian kernels) with $2,700$ parameters to train. For the dimensions where we aim to control asymptotics, we make sure the centers and widths of the Gaussian kernels are within the learning bounds as described in Section 5. For uncontrolled dimensions the bounds for the centers and widths are set very wide.

We compare our results with the method that does not employ asymptotics control at all. For that we use a NN with two hidden sigmoid[2] activation layers, one with 300 nodes and another with 4. The number of training parameters, 2708, is very close to the CRL network.

We use $10,000$ learning points in the learning region to train our models, and $10,000$ validation points over the (larger) measurement region. The points are sampled from a fixed-seed Gaussian distribution with appropriate standard deviation to ensure good coverage (we report the split of validation points over various regions in Table 4). In reported tests we use Adam optimizer with $1,000$ epochs and 32 batches. The Adam optimizer uses default parameters as specified in Tensorflow library, specifically with the learning rate of 0.001. The model with no CRL uses the standard mean squared error loss function, whereas for the CRL layer we modify it by adding a penalty to ensure that Gaussian "bells" stay within required bounds as described in Section 5. We do not explicitly introduce any overfitting measures except by using models with the number of parameters significantly smaller than the number of learning points.

Once the approximation to the implied volatility function is learned, we generate a validation sample that covers the larger measurement region so that we can calculate various statistics, focusing on the deviation of the fit from the known exact results in various regions. We describe the quality of the fit in the next section.

Our NN architecture resembles [MG] more than [HMT] as we train the network to fit each learning point individually rather than a grid of them together.

[2] According to our experiments, the sigmoid activation functions are the most efficient for our application.

Electronic copy available at: https://ssrn.com/abstract=3544698

The latter was demonstrated to be superior but, since the focus of our paper is on comparing our asymptotics controlled architecture versus a non-asymptotics controlled one, we use a simpler framework of [MG] for clarity of exposition. We expect similar improvements in the [HMT] architecture.

8.6 Testing: Errors

Our tests show that controlling asymptotics significantly improves the NN fit to the SABR implied volatilities. Table 4 summarizes the results. Errors reported are mean-squared absolute errors against the exact results that are in units of log-normal volatilities so of the order $\sim$20% around the center of the learning region, but could be as high as 800% at the edges.

"Learning error" is the mean-squared error over all learning points (that are all in the learning region). "Validation error" is the mean-squared error over all validation points; they cover the measurement region which is the largest of the three. Learning, asymptotic, and measurement region errors are mean-squared absolute errors over validation points in the relevant parts of the respective regions, i.e. in $[-L_l, L_l]^4$, $[-L_a, L_a]^4 \setminus [-L_l, L_l]^4$, and $[-L_m, L_m]^4 \setminus [-L_a, L_a]^4$ respectively. We report the number of validation points in each set in Table 4.

"Model 1" in Table 4 refers to an asymptotics controlled CRL NN where in addition to the volatility and the strike dimensions, we also control the asymptotics in the volatility of volatility (γ) dimension. We include these results to show the improvements achieved by controlling asymptotics in more dimensions.

"Model 2" refers to our main subject of experiments, an asymptotics controlled CLN NN where only the volatility and the strike dimensions are controlled.

"Model 3" refers to our baseline NN without asymptotics control as described in Section 8.5, which is a deep NN with two hidden layers (specifically, 300 nodes on the first one and 4 nodes on the second one) and sigmoid activators.

All three models fit the inputs equally well when measured at all the learning points ("Learning error" row) and over validation points in the learning region ("Learning region error" row). Model 1 and Model 2 outperform Model 3 significantly (in some cases by an order of magnitude) outside the learning region as seen from other rows. Also, not surprisingly, Model 1 outperforms Model 2 measurably outside of the learning region as it has more dimensions under asymptotic control.

8.7 Testing: Plots

For further insight, we visualize test results in the following figures, although, admittedly, it is difficult to plot four-dimensional functions. We present our results as a grid of surface subplots. Different surface subplots correspond to various values of fixed parameters. For example, for Figure 18, each subplot corresponds to a specific pair of values for β and γ (normalized) and displays the value of interest (fit error) against the other two dimensions, in this case ATM volatility v and strike K (also normalized).

Electronic copy available at: https://ssrn.com/abstract=3544698

Absolute error	Model 1	Model 2	Model 3	Num pts
Learning error	0.026%	0.029%	0.029%	N/A
Validation error	0.75%	1.56%	15.16%	10,000
Learning region error	0.09%	0.08%	0.07%	2,600
Asymptotic region error	0.49%	1.25%	2.69%	4,200
Measurement region error	1.2%	2.3%	26.7%	3,200

Table 4: Summary results for SABR fitting.

Note: Absolute mean-squared error of volatility fitting. "Model 1" is a CRL NN with asymptotics control in 3 dimensions. "Model 2" is a CRL NN with asymptotics control in 2 dimensions. "Model 3" is a standard deep NN with no asymptotics control. More details in Section 8.6.

Figure 18 plots SABR volatilities that we are trying to fit. Note quite extreme values towards the edges of the region of interest.

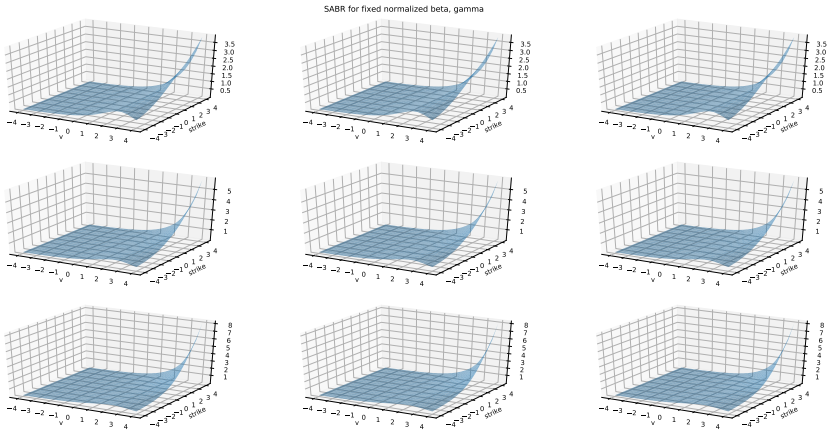

Figure 18: SABR implied volatility for different β, γ.

In Figure 19 we plot the fitting error of the NN vs. exact SABR values using our method of asymptotics control.

Electronic copy available at: https://ssrn.com/abstract=3544698

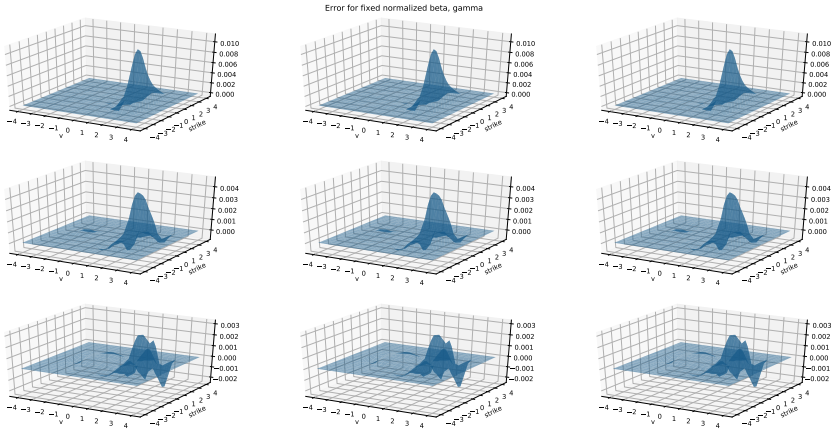

Figure 19: Volatility fitting error for NN with asymptotics control for β, γ.

The same in the non-asymptotics-controlled case is plotted in Figure 20.

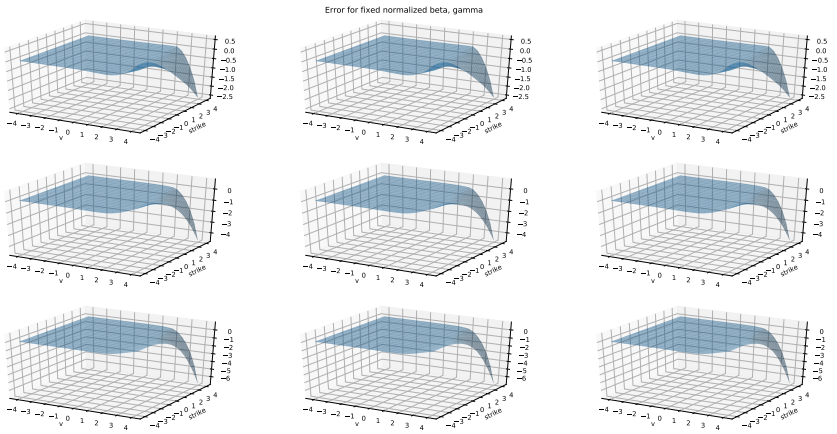

Figure 20: Volatility fitting error for NN without asymptotics control for different β, γ.

Comparing the two plots, Figure 19 and 20, we note that the error is much smaller in the asymptotics controlled case for all values of parameters, with the highest error coming from the non-controlled behavior in the "no-man's land" $[-L_a, L_a]^4 \setminus [-L_l, L_l]^4$, with the expected behavior within the learning region $[-L_l, L_l]^4$ and in the asymptotics regime $[-L_m, L_m]^4 \setminus [-L_a, L_a]^4$. Yet, it is still dwarfed by the error in the non-controlled case outside of $[-L_l, L_l]^4$.

To gain further insight, we focus our attention on what we call the "central" case of setting the fixed normalized parameters to 0, i.e. $z_2 = 0$ and $z_3 = 0$. This corresponds to their de-normalized values $\beta = 0.5$ and $\gamma = \sqrt{3} \approx 1.73$. In

Electronic copy available at: https://ssrn.com/abstract=3544698

Figure 21, we plot SABR volatilities we try to approximate.

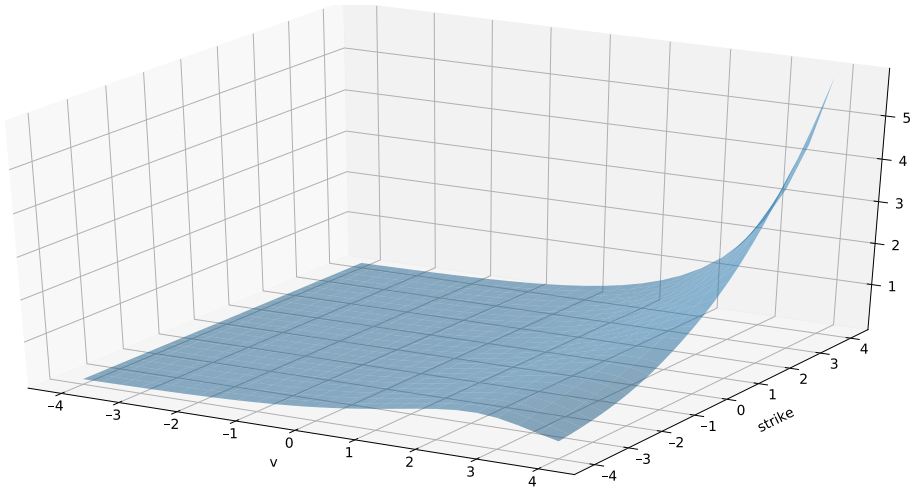

Figure 21: SABR implied volatility for fixed $\beta = 0.5$ and $\gamma = \sqrt{3}$.

Figure 22 shows the approximate values obtained with the NN where the strike and volatility asymptotics are controlled.

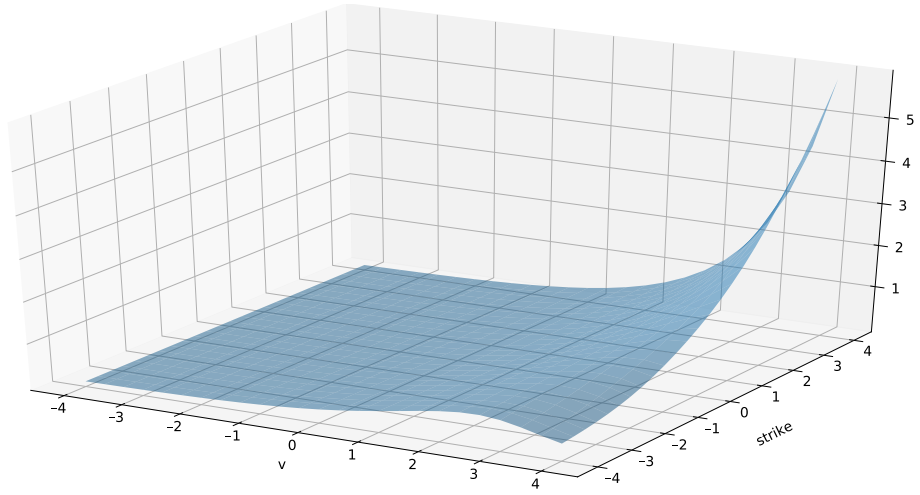

Figure 22: Implied volatility approximated with NN with strike and volatility asymptotics control for fixed $\beta = 0.5$ and $\gamma = \sqrt{3}$.

Figure 23 shows the approximate values obtained with the NN with no asymptotics control.

Electronic copy available at: https://ssrn.com/abstract=3544698

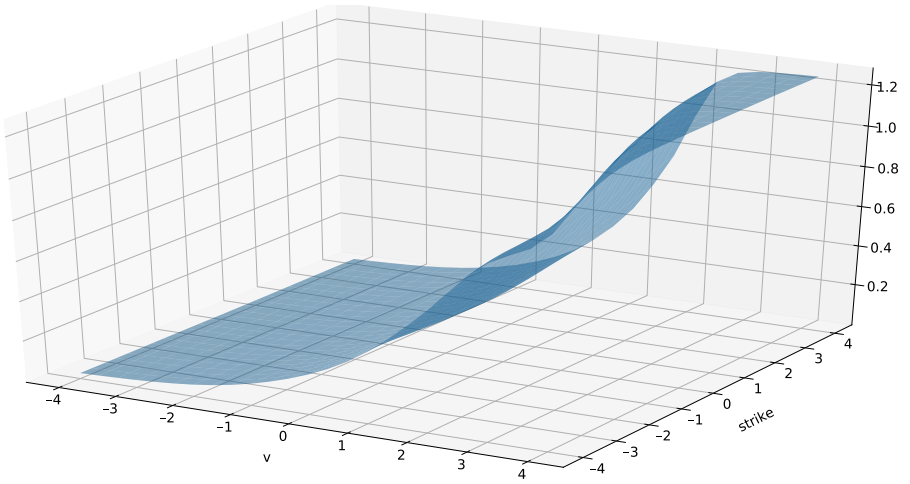

Figure 23: Implied volatility approximated with NN with no asymptotics control for fixed $\beta = 0.5$ and $\gamma = \sqrt{3}$.

Let us now look at the errors. Figure 24 shows the approximation error for the NN where strike and volatility asymptotics are controlled.

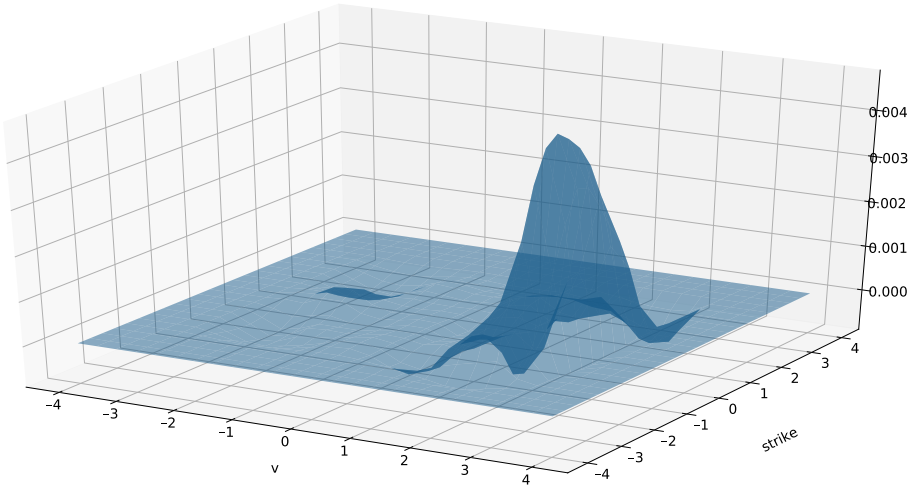

Figure 24: Volatility fitting error for NN with strike and volatility asymptotics control for fixed $\beta = 0.5$ and $\gamma = \sqrt{3}$.

Likewise, Figure 25 shows the approximation error for the NN with no asymptotics control. As already demonstrated the fit outside of the learning region is significantly worse.

Electronic copy available at: https://ssrn.com/abstract=3544698

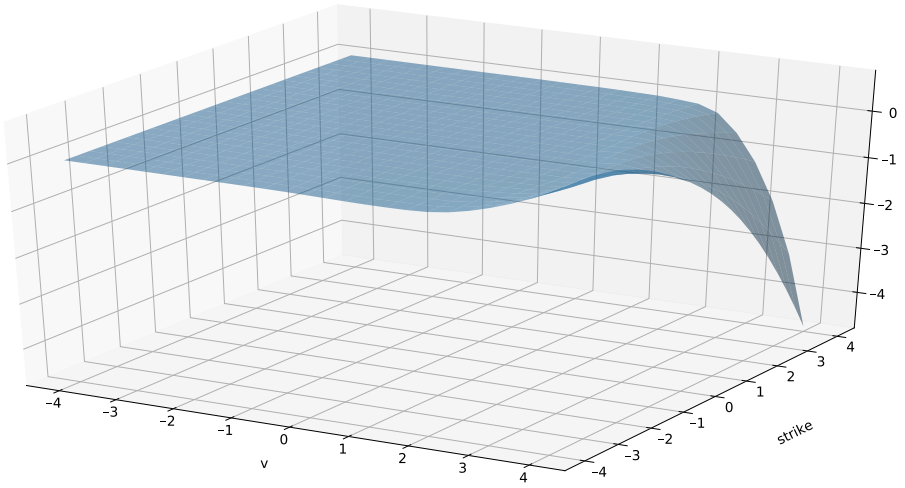

Figure 25: Volatility fitting error for NN with no asymptotics control for fixed $\beta = 0.5$ and $\gamma = \sqrt{3}$.

Next, let us look more closely at the effect of controlling asymptotics in more dimensions. In Figure 26, we plot the fitting error for the NN where the strike and volatility dimensions are asymptotics-controlled but the skew and volatility-of-volatility are not – i.e. our main test case. These results are for the same model as in Figure 24 but we plot them for fixed volatility and skew so that we can explore the behavior of the error for large strikes and volatility-of-volatilities.

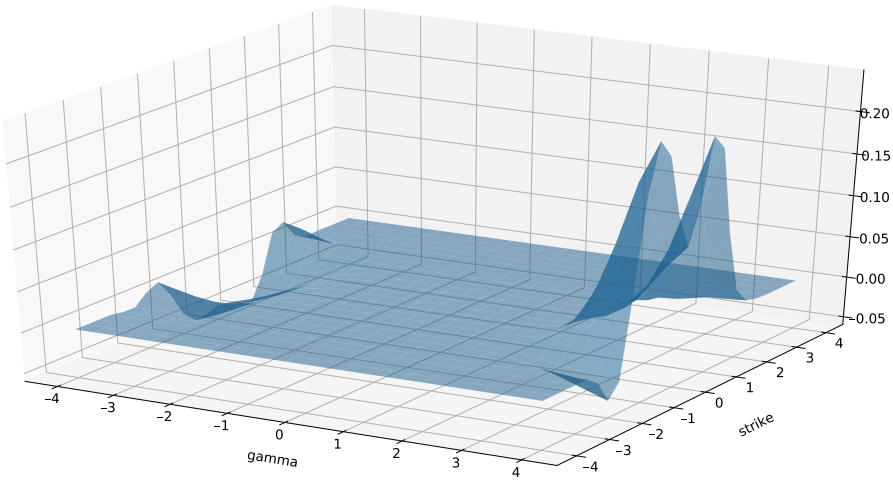

Figure 26: Volatility fitting error for NN with strike and volatility asymptotics control for fixed $v = 0.2$ and $\beta = 0.5$.

Electronic copy available at: https://ssrn.com/abstract=3544698

Figure 27 demonstrates what happens to this error if we add volatility-of-volatility γ to the dimensions we control. Comparing to Figure 26 we see much smaller error approaching 0 for now-controlled large γ values.

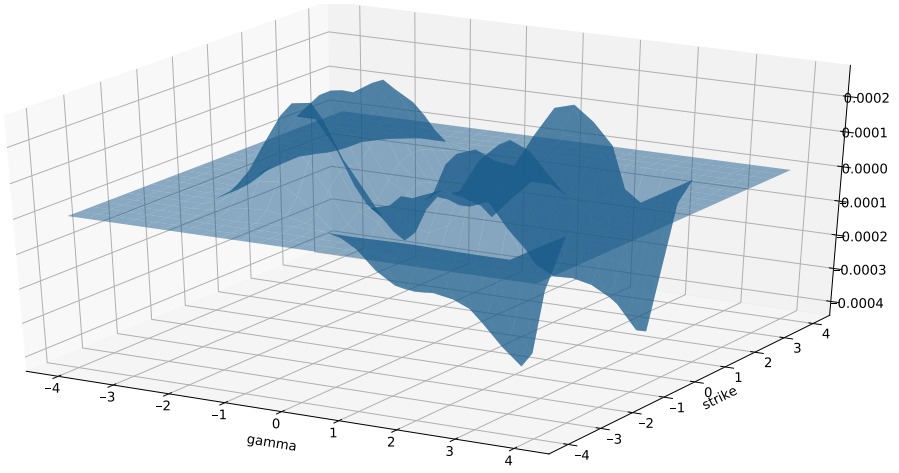

Figure 27: Volatility fitting error for NN with strike, volatility and volatility-of-volatility asymptotics control for fixed $v = 0.2$ and $\beta = 0.5$.

9 Conclusions

Recently ANNs, and specifically deep ANNs, have been proposed as an effective approximation mechanism for various multi-dimensional functions arising in financial applications. These include values of various derivatives as functions of multi-dimensional pricing parameters, volatility surface parameterizations, and the like. It is safe to say that it has been conclusively confirmed that ANNs are good in matching function values at the learning points and, with some care, in interpolating between the training points. Extrapolating beyond the training set, however, is widely acknowledged as a serious weakness of traditional ANNs. The problem is particularly acute in financial applications given that financial models need to deal with rapid regime changes, stress tests, and regulatory requirements on explainability, to name only a few areas of concern.

In this paper, we develop a method to control ANN extrapolation by incorporating extra knowledge of the asymptotics of financial functions into the ANN construction. Such asymptotics are often already available for common financial functions, such as implied volatilities in the SABR model for large values of parameters. Alternatively, they could be derived using an extensive and ever-growing tool-set of asymptotic analysis techniques. Initial results look promising but more work remains to test our approach for a wide variety of ANN approximation problems.

Electronic copy available at: https://ssrn.com/abstract=3544698

10 References

References

[AP] A. Antonov and V. Piterbarg, "Neural Networks with Asymptotics Control", 2019, The 15th WBS Fixed Income Conference

[AKS] A. Antonov, M. Konikov and M. Spector, "Modern SABR Analytics", 2019, Springer

[BG] J. Braun and M. Griebel "'On a constructive proof of Kolmogorov's superposition theorem"

[BGTW] H. Buehler, L. Gonon, J. Teichmann, B. Wood. "Deep Hedging", 2018, arxiv

[GC] G. Cybenko "Approximation by superpositions of a sigmoidal function" Math. Control Signal Systems (1989) 2: 303.

[FG] R. Ferguson and A.D. Green, "Deeply Learning Derivatives", 2018, SSRN working paper, https://ssrn.com/abstract=3244821

[PHL] P. Henry-Labordere. "CVA and IM: Welcome to the Machine", March 2019, Risk Magazine

[HSW] K. Hornik, M. Stinchcombe and H. White "Multilayer feedforward networks are universal approximators", Neural Networks Volume 2, Issue 5, 1989, Pages 359-366

[HMT] B. Horvath, A. Muguruza and M. Tomas, "Deep Learning Volatility" 2019, SSRN working paper: https://ssrn.com/abstract=3322085

[KALN] L. Kienitz, S. K. Acar, Q. Liang and N. Nowaczyk, "The CV Makes the Difference – Control Variates for Neural Networks", 2020, SSRN working paper: https://ssrn.com/abstract=3527314

[AK] A. Kondratyev, "Curve dynamics with artificial neural networks", 2018, Risk

[KS] A. Kondratyev, C. Schwarz. "The Market Generator", 2019, SSRN working paper, https://ssrn.com/abstract=3384948

[MG] W. McGhee, "An Artificial Neural Network Representation of the SABR Stochastic Volatility Model" (2018), SSRN working paper https://ssrn.com/abstract=3288882

[NR] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, "Numerical Recipes 3rd Edition: The Art of Scientific Computing", 2007

[GR] G. Ritter. "Machine learning for trading". October 2017, Risk Magazine, pp. 84-89

Electronic copy available at: https://ssrn.com/abstract=3544698

A Theoretical Basis

Two theorems form the theoretical basis for the NN construction, Cybenko-Hornik [HSW], [GC] and Kolmogorov-Arnold, or Hilbert's 13th problem extension, see [BG].

A.1 Cybenko-Hornik

The Cybenko-Hornik theorem states that one can approximate any continuous function f as a sum of

$$f(x) \simeq \sum_{j=1}^{N} a_j \sigma \left(x \cdot w_j + b_j \right),$$

where $\sigma(x)$ is the so-called sigmoid function [GC]. Later this result was generalized to any continuous bounded (and non-constant) function [HSW]. In other words, sums $\sum_{j=1}^{N} a_j \sigma \left(x \cdot w_j + b_j \right)$ are dense in $C(I^D)$.

This theorem forms a basis for a construction of a two-layer Neural Network. Namely, the first layer does a linear transformation to the input and applies the non-linear function σ, i.e. $x \to \sigma \left(x \cdot w_j + b_j \right)$. The second layer sums up the obtained values. The "trainable" coefficients $\{a_j, w_j, b_j\}_{j=1}^{N}$ are found by a numerical procedure minimizing a difference between the Cybenko-Hornik form and the input function values.

For a special case of a multi-dimensional Fourier series, i.e. $\sigma(x) = e^x$, fixed (imaginary) weights w's and zero b's, one can identify the function asymptotics with a behaviour of coefficients a. However, while working with a typical NN, the weights are variable and the function $\sigma(x)$ can be complicated, so the asymptotic control is hardly possible.

A.2 Hilbert: A Bit of History

In 1900 David Hilbert posed 23 problems that shaped the course of mathematics for the next century. Among them was Problem 13: A solution to a 7th degree polynomial equation can be reduced to a function of three parameters. Can the solution be represented as a combination of functions of two arguments? Hilbert did not think so and Hilbert's conjecture states that "A solution of the general equation of degree 7 cannot be represented as a superposition of continuous functions of two variables".

The problem can be generalized to whether every continuous function of three variables can be expressed as a composition of finitely many continuous functions of two variables?

In 1956, Andrey Kolmogorov showed that any function of several variables can be constructed as a superposition of finite number of three-variable functions. Kolmogorov's student Vladimir Arnold, 19 at the time, proved that only two-variable functions are in fact required, thus showing that Hilbert's intuition was wrong and solving the 13th problem.

Electronic copy available at: https://ssrn.com/abstract=3544698

Kolmogorov and Arnold later refined it and showed that every multivariate continuous function can be represented as a superposition of continuous functions of one variable.

A.3 Kolmogorov-Arnold Theorem

The Kolmogorov-Arnold theorem (see e.g. [BG]) states that a D-dimensional function (on a D-dim unit cube I^D) can be represented as a sum of $M \geq 2D+1$ elements

$$f(x_1, \ldots, x_D) = \sum_{q=0}^{M} g \left(\sum_{p=1}^{D} \lambda_p \psi(x_p + q\, a) + B\, q \right),$$

where an integer $\gamma \geq M + 2$ and constants $a = \frac{1}{\gamma(\gamma-1)}$, $\lambda_1 = 1$, $\lambda_p = \sum_{r=1}^{\infty} \gamma^{-(p-1)\,\beta(r)}$ for $p = 2, \ldots, D$ for $\beta(r) = \frac{D^r - 1}{D-1}$. Finally, the constant B is proportional to a sum of λ's.

The Kolmogorov-Arnold representation is based on two one-dimensional functions: *outer* function $g(x)$ and the *inner* one ψ. The outer function depends on the initial function $f(x_1, \ldots, x_D)$ while the inner one does not. It maps I to $\mathbb{R}$ and has a very irregular structure. In the graph below we give its Spencer form

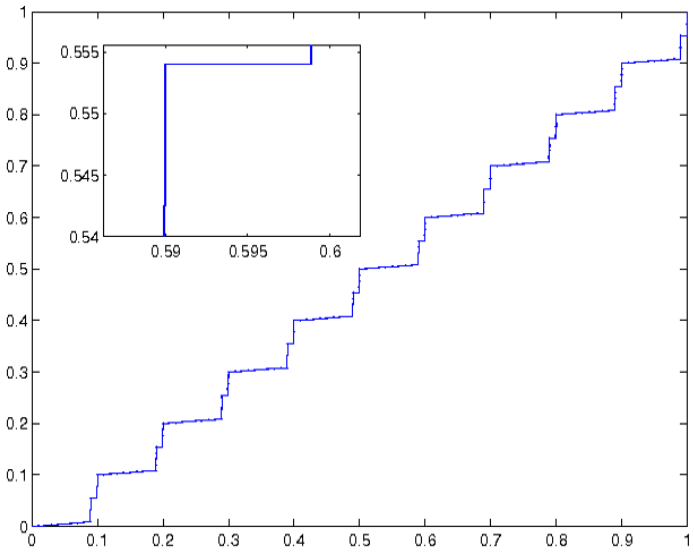

Figure 28: Spencer ψ-function– the plot is taken from Braun & Griebel.

A *constructive* proof of the Kolmogorov-Arnold theorem describes a construction of the function g given the initial function f. The logic is based on

Electronic copy available at: https://ssrn.com/abstract=3544698

the so called K-basis functions, a map from $I^D \to \mathbb{R}$

$$\xi(x_1, \dots, x_D) = \sum_{p=1}^{D} \lambda_p \, \psi(x_p)$$

such that

$$f(x_1, \dots, x_D) = \sum_{q=0}^{M} g\left(\xi(x_1 + q\,a, \dots, x_D + q\,a) + B\,q\right). \tag{26}$$

The function ξ maps $I^D \to \mathbb{R}$ similar to how one can map a matrix (or a tensor) into a vector. The map is "almost" a bijection: it sends D-dimensional non-overlapping hypercubes to disjoint intervals on $\mathbb{R}$. Obviously, it has some imperfections – little holes between the hypercubes – otherwise, the D-dimensional space would be equivalent to one-dimensional one!

A striking property of the function ξ is that even if we decrease the size of non-overlapping hypercubes, their images under ξ will still be disjoint.

As an illustration let us construct the function $\xi(x)$ for a *fixed* spacing L of hypercubes. Taking $\psi(x) = x$ and $\lambda_p = \text{const}\ L^{-p+1}$, choose the function

$$\xi^{(L)}(x) = \frac{\sum_{p=1}^{D} L^{-p+1}\, x_p}{\sum_{p=1}^{D} L^{-p+1}} = \frac{x_1 + x_2 L^{-1} + x_3 L^{-2} + \dots + x_D\, L^{-D+1}}{\sum_{p=1}^{D} L^{-p+1}}. \tag{27}$$

Its property

$$\xi^{(L)}(\dots, x_k, x_{k+1} + L, \dots) = \xi^{(L)}(\dots, x_k + 1, x_{k+1}, \dots) \tag{28}$$

makes it similar to a vector/matrix indexation: $(i, j) \to i + N\,j$.

It is easy to see that the "toy" function $\xi^{(L)}$ maps different L-sized hypercubes $\mathcal{Q}$ into non-overlapping intervals on $\mathbb{R}$ except maybe for one point. There is no guarantee that *smaller* cubes will be mapped into non-overlapping intervals.

As a variant of $\xi^{(L)}$ we can "flatten" the function (27) to be constant on each interval L. This brings us to a "stairs" representation of $\psi(x)$-function resembling that of Spencer at Figure 28. For our toy example the function g can be constructed as

$$g\left(\xi^{(L)}\left(x_1, \dots, x_D\right)\right) \simeq f\left(x_1, \dots, x_D\right). \tag{29}$$

Let us illustrate a form of the toy $g(x)$ corresponding to a two-dimensional function $f(x_1, x_2) = x_2/(1 - x_1)$

Electronic copy available at: https://ssrn.com/abstract=3544698

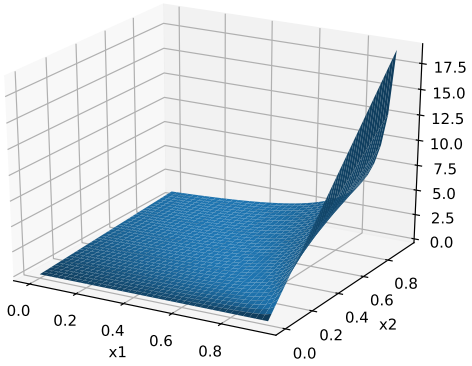

Figure 29: Function $f(x_1, x_2) = x_2/(1 - x_1)$.

The function g represents a "matrix" f in terms of a vector on a grid with the spacing $L = 0.01$.

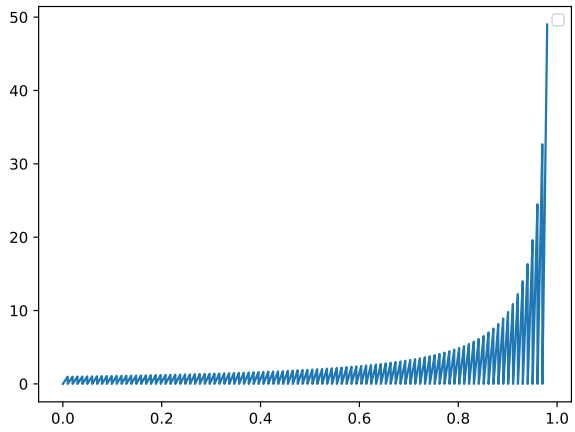

Figure 30: The toy function g for a spacing $L = 0.01$.

We observe its irregular and quasi-periodic behavior (due to (28)). With some effort, one can see in the graph 30 some "traces" of the asymptotics of the function f.

Coming back to the proof of the theorem, we can set the function $g(x)$ following (29) and somehow interpolate between the hypercube images. However,

Electronic copy available at: https://ssrn.com/abstract=3544698

the imperfections of the map ξ *between* the hypercubes are such that the convergence $g\left(\xi\left(x_1,\ldots,x_D\right)\right)$ to $f\left(x_1,\ldots,x_D\right)$, as cube size tends to zero, cannot be guaranteed.

The solution to the convergence issue is to shift the cubes to cover potential holes in such a way that

$$g\left(\xi\left(x_1+q\,a,\ldots,x_D+q\,a\right)+B\,q\right)\simeq\frac{f\left(x_1+q\,a,\ldots,x_D+q\,a\right)}{M+1}.$$

The shift $B\,q$ extrapolates the function g outside of the unit interval. The number of shifts $M+1$ is chosen to assure the convergence of the sum in (26)

$$\begin{aligned}\sum_{q=0}^{M}g\left(\xi(x_1+q\,a,\ldots,x_D+q\,a)+B\,q\,\right)&\simeq\sum_{q=0}^{M}\frac{f\left(x_1+q\,a,\ldots,x_D+q\,a\right)}{M+1}\\&\to f(x_1,\ldots,x_D),\end{aligned}$$

while the hypercube sizes tend to zero.

A.4 The Kolmogorov-Arnold theorem and the NN

We have mentioned that the Cybenko-Hornik theorem gives rise to a two-layer NN. Let us see how the Kolmogorov-Arnold theorem can transform the initial multi-dimensional function f into a feedforward NN.

First, represent the one-dimensional function g using Cybenko-Hornik theorem

$$g(y)\simeq\sum_{j=1}^{N}\alpha_j\sigma\left(yw_j+\beta_j\right). \tag{30}$$

Then, the Kolmogorov-Arnold representation transforms this into

$$\begin{aligned}f(x)&=\sum_{q=0}^{M}g\left(\sum_{p=1}^{D}\lambda_p\psi(x_p+q\,a)+B\,q\right)\\&\simeq\sum_{j=1}^{N}\sum_{q=0}^{M}\alpha_j\sigma\left(\left(\sum_{p=1}^{D}\lambda_p\psi(x_p+q\,a)+B\,q\right)w_j+\beta_j\right),\end{aligned}$$

which corresponds to the following layers:

- The first hidden layer transforms the input variables x_i into

  $$x_{p,q}^{(1)}=\psi(x_p+qa),$$

  where the function ψ is serves at an activation function and all the parameters are non-trainable.

Electronic copy available at: https://ssrn.com/abstract=3544698

- The second hidden layer

$$x^{(2)}_{j,q} = \sigma\left(w_j \sum_{p=1}^{D} \lambda_p\, x^{(1)}_{p,q} + B\, q\, w_j + \beta_j\right)$$

  contains w_i and β_i as trainable parameters.

- The output layer

$$f(x) = x^{(3)} = \sum_{j=1}^{N} \sum_{q=0}^{M} \alpha_j\, x^{(2)}_{j,q}$$

  trains the linear parameters α_j.

Let us analyze the obtained NN from the asymptotic control point of view. As we have mentioned in the previous section the function g blurs the information from the initial function f, see Figure 30, but still there are some "traces" (of course, they are visible only for low dimensional problems, e.g. 2-3D). However, the procedure (30) underlying the NN construction completely destroys any asymptotic information of the function f. So, a direct control of asymptotics inside standard NNs is hardly possible.

B Splines

B.1 One-Dimensional Cubic Spline

In this appendix we use notations and natural (i.e. without asymptotic control) spline description from [NR].

A simple linear interpolation between x_j and x_{j+1} for the function $y(x)$ with values y_j and y_{j+1} is given by

$$y(x) = Ay_j + By_{j+1}, \tag{31}$$

where

$$A = A(x) = \frac{x_{j+1} - x}{x_{j+1} - x_j} \quad \text{and} \quad B = B(x) = 1 - A = \frac{x - x_j}{x_{j+1} - x_j}. \tag{32}$$

Let us consider the problem of cubic interpolation assuming that we already know the second derivatives at the knots y''_j and y''_{j+1}. First we interpolate second order derivatives linearly,

$$y''(x) = Ay''_j + By''_{j+1}. \tag{33}$$

Then we integrate this expression two times and, for the interval $[y_j,y_{j+1}]$ we obtain a nice and symmetric expression for the function and its derivative,

$$y(x) = A(x)y_j + B(x)y_{j+1} + \Delta^2 x_j \frac{A^3(x) - A(x)}{6}\, y''_j + \Delta^2 x_j \frac{B^3(x) - B(x)}{6}\, y''_{j+1},$$

$$y'(x) = \frac{\Delta y_j}{\Delta x_j} - \Delta x_j \frac{3A^2(x) - 1}{6}\, y''_j + \Delta x_j \frac{3B^2(x) - 1}{6}\, y''_{j+1},$$

Electronic copy available at: https://ssrn.com/abstract=3544698

with the following derivatives at the knots

$$y'_j = \frac{\Delta y_j}{\Delta x_j} - \frac{1}{3}\Delta x_j\, y''_j - \frac{1}{6}\Delta x_j\, y''_{j+1}, \tag{34}$$

$$y'_{j+1} = \frac{\Delta y_j}{\Delta x_j} + \frac{1}{6}\Delta x_j\, y''_j + \frac{1}{3}\Delta x_j\, y''_{j+1}. \tag{35}$$

To calculate the second derivatives at the knots given the input $\{x_i, y_i\}_{i=0}^N$, we impose the condition that the first-order derivative is continuous at the knots, so that.

$$y'(x_j-) = y'(x_j+). \tag{36}$$

The second derivative is continuous at the knots due to (33). The condition (36) implies

$$\frac{1}{6}\Delta x_{j-1}\, y''_{j-1} + \frac{1}{3}(\Delta x_{j-1} + \Delta x_j)\, y''_j + \frac{1}{6}\Delta x_j\, y''_{j+1} = \frac{\Delta y_j}{\Delta x_j} - \frac{\Delta y_{j-1}}{\Delta x_{j-1}} \tag{37}$$

for $j = 2, \cdots, N-1$ or, simply,

$$\mu_{j-1}\, y''_{j-1} + 2\, y''_j + (1-\mu_{j-1})\, y''_{j+1} = d_j \quad \text{for} \quad j = 2, \cdots, N-1$$

where

$$\mu_j = \frac{\Delta x_j}{\Delta x_j + \Delta x_{j+1}} \quad \text{and} \quad d_j = 3\frac{\frac{\Delta y_j}{\Delta x_j} - \frac{\Delta y_{j-1}}{\Delta x_{j-1}}}{\frac{\Delta x_{j-1}+\Delta x_j}{2}}.$$

The value of d_j is 3 times a finite difference approximation to the second derivative.

As we have $N-2$ equations for N unknowns (derivatives y''_i for $j = 1, \cdots, N$), we can/should impose two extra conditions. The general conditions

$$2\, y''_0 + (1-\mu_{-1})\, y''_1 = d_0 \quad \text{and} \quad \mu_{N-1}\, y''_{N-1} + 2\, y''_N = d_N$$

encapsulate two standard approaches:

- a natural spline: zero second derivatives at the boundaries $y''_0 = y''_N = 0$ (linear extrapolation); or
- fixed first derivatives at the boundaries y'_0 and y'_N
  From (34)–(35) we have

$$y''_0 + 2y''_1 = \frac{6}{\Delta x_0}\left(\frac{\Delta y_0}{\Delta x_0} - y'_0\right),$$

$$y''_{N-1} + 2\, y''_N = \frac{6}{\Delta x_{N-1}}\left(y'_N - \frac{\Delta y_{N-1}}{\Delta x_{N-1}}\right). \tag{38}$$

The second case is of interest to us as the control of the values and first derivatives of the function at the edges (y_0, y'_0) and (y_N, y'_N) allows us to attach

Electronic copy available at: https://ssrn.com/abstract=3544698

(linear) asymptotics to the spline. All we need to do is to combine the system of equations (37) with the two extra edge conditions (38).

The approach above allows us to have C^1 smoothness at the edge points where the spline switches to asymptotic functions. For C^2 smoothness we need to control the second derivative at the edge knots, and for that we need to introduce extra degrees of freedom. As mentioned in the paper, we can achieve that by adding two extra points to the spline nodes, $x_{\frac{1}{2}}$ situated between x_0 and x_1 and $x_{N-\frac{1}{2}}$ between x_{N-1} and x_N. According to our experiments, these extra points can be placed in the middle of the corresponding intervals. The values on these new points give a handle to control the second derivatives at the boundaries. For better readability of formulas we rename $x_{\frac{1}{2}}$ to x_1 and $x_{N-\frac{1}{2}}$ to x_N increasing the number of nodes by 2.

Formally, our input consists on the derivatives at the boundaries, (y_0, y'_0, y''_0) and $(y_{N+1}, y'_{N+1}, y''_{N+1})$, and values at the initial "internal" knots $y_2, \cdots, y_{N-1}$. Our goal is to find values of the added points y_1 and y_N coherent with the input and the spline smoothness.

Using (34)–(35) we can express the added values through the boundary derivatives as well as the second derivatives at the added points,

$$\begin{aligned} y_1 &= y_1^* + \frac{1}{6}\Delta x_0^2\, y_1'' \quad \text{for} \quad y_1^* = y_0 + \Delta x_0\, y'_0 + \frac{1}{3}\,\Delta x_0^2\, y''_0, \\ y_N &= y_N^* + \frac{1}{6}\Delta x_N^2\, y_N'' \quad \text{for} \quad y_N^* = y_{N+1} - \Delta x_N\, y'_{N+1} + \frac{1}{3}\,\Delta x_N^2\, y''_{N+1}, \end{aligned} \tag{39}$$

The added values depend on the fixed ones, y_1^* and y_N^*, as well as yet unknown second derivatives y_1'' and y_N''. Thus, we should substitute the values (39) into the r.h.s of (37) and transfer the dependence on the second derivatives in the l.h.s. The final system of equations reads:

$$\begin{aligned} \mu_0\, y''_0 + &\left(2 + \frac{1}{6}\Delta x_0^2\,\xi_0\right) y''_1 + &&(1-\mu_0)\, y''_2 = R_0 \\ (\mu_1 - \varepsilon_1)\, y''_1 + &\quad 2\, y''_2 + &&(1-\mu_1)\, y''_3 = R_1 \\ &\cdots \\ \mu_j\, y''_j + &\quad 2\, y''_{j+1} + &&(1-\mu_j)\, y''_{j+2} = R_j \\ &\cdots \\ \mu_{N-2}\, y''_{N-2} + &\quad 2\, y''_{N-1} + &&\left(1-\mu_{N-2}-\varepsilon_{N-2}\right) y''_N = R_{N-2} \\ \mu_{N-1}\, y''_{N-1} + &\left(2 + \frac{1}{6}\Delta x_N^2\,\xi_{N-1}\right) y''_N + &&(1-\mu_{N-1})\, y''_{N+1} = R_{N-1} \end{aligned} \tag{40}$$

where we have introduced the following notations

$$\begin{aligned} \xi_j &= \frac{6}{\Delta x_j\, \Delta x_{j+1}} \quad \text{and} \quad \beta_j = (1-\mu_j)\,\xi_j, \\ \varepsilon_1 &= \frac{1}{6}\Delta x_0^2\,\beta_1 \quad \text{and} \quad \varepsilon_{N-2} = \frac{1}{6}\Delta x_N^2(\xi_{N-2} - \beta_{N-2}) \end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3544698

The r.h.s. and the diagonal are denoted, respectively,

$$R_j = \beta_j\,\hat{y}_j - \xi_j\,\hat{y}_{j+1} + (\xi_j - \beta_j)\,\hat{y}_{j+2} \quad \text{and} \quad \hat{y}_j = y_j + (y_1^* - y_1)\delta_{j,1} + (y_N^* - y_N)\delta_{j,N}$$

for $j = 0, \cdots, N-1$.

The tri-diagonal linear system (40) determines $y_1'', \cdots, y_N''$. The second derivatives at the boundaries y_0'' and y_{N+1}'' are transferred to the l.h.s. for symmetry. Finally, the added values (39) can be obtained from the second derivatives.

B.2 Multi-Dimensional Spline Calculation Complexity

We will use the one-dimensional spline costs detailed in Section 4.3 to estimate the overall multi-dimensional spline calculation. For this we should add the timings of the following operations:

1. Prepare the elementary splines (5).
   It is easy to see, by induction, that the preparation time is
   $$T_B\; N_1 \cdots N_D. \tag{41}$$
   Note that this operation is done once and does not depend on the number of calculation points x'.

2. Prepare and calculate the *bespoke* one-dimensional splines in algorithm steps **A3**–**A4** from Section 4.4.
   We suppose that the calculation point lies inside the spline grid – an asymptotic point calculation time is simply T_A.
   Denote by $T^{(D)}$ a calculation time for one point in the spline $S^{(D)}$. By induction step **A5** (from Section 4.4), the time $T^{(D)}$ can be presented as a time of N_D calculations of the reduced splines $S_{n_D}^{(D-1)}$ and the bespoke spline time (**A4**). The latter consists of its preparation $N_D\,T_B$ with two asymptotics calculations $2T_A$ and the point evaluation T_C. In sum, we have
   $$T^{(D)} = N_D\,T^{(D-1)} + 2\,T_A + N_D\,T_B + T_C \tag{42}$$
   To start the induction we need 1D spline timing which is simply $T^{(1)} = T_C$. Elementary calculations give
   $$\begin{aligned} T^{(D)} &= T_B\;(N_D + N_{D-1}\,N_D + \cdots + N_2 \ldots N_D) \\ &+ T_C\,N_2 \ldots N_D \\ &+ (2T_A + T_C)\,(1 + N_D + N_{D-1}\,N_D + \cdots + N_3 \cdots N_D) \end{aligned}$$
   For sufficiently large number of nodes it is reduced to
   $$T^{(D)} \simeq (T_B + T_C)\;N_2 \ldots N_D + 2T_A\;N_3 \cdots N_D$$

Finally, the calculation effort estimate is given by

$$T_B\;N_1 \cdots N_D + (T_B + T_C)\;M_p\;N_2 \ldots N_D + 2T_A\;M_p\;N_3 \cdots N_D \tag{43}$$

where M_p is the number of calculation points x'.

Electronic copy available at: https://ssrn.com/abstract=3544698