
ENFORCING ASYMPTOTIC BEHAVIOR WITH DNNs FOR APPROXIMATION AND REGRESSION IN FINANCE

Hardik Routray

Wells Fargo Bank, N.A.

hardik.routray@wellsfargo.com

Bernhard Hientzsch

Wells Fargo Bank, N.A.

bernhard.hientzsch@wellsfargo.com

July 8, 2025

ABSTRACT

We propose a simple methodology to approximate functions with given asymptotic behavior by specifically constructed terms and an unconstrained deep neural network (DNN). The methodology we describe extends to various asymptotic behaviors and multiple dimensions and is easy to implement. In this work we demonstrate it for linear asymptotic behavior in one-dimensional examples. We apply it to function approximation and regression problems where we measure approximation of only function values (“Vanilla Machine Learning”-VML) or also approximation of function and derivative values (“Differential Machine Learning”-DML) on several examples. We see that enforcing given asymptotic behavior leads to better approximation and faster convergence.

Keywords Asymptotic behavior and forms · Differential Machine Learning · Function Approximation · Regression · Approximation of conditional expectations

1 Introduction

Often solutions to scientific problems and financial problems have known asymptotic behavior. In particular pricing functions in quantitative finance often have or are assumed to have linear asymptotes [HS20, Section 2 in Appendix 4]. If one represents pricing functions with a deep neural network (DNN) with activation functions with certain properties, these pricing functions will be asymptotically linear, but training the asymptotic part implicitly as part of the DNN presents challenges and requires special treatment [HS20, Section 2 in Appendix 4]. Often, unless specifically addressed in the training and setup, convergence of the DNN to the true asymptotic behavior can be slow or not even assured. Thus, one is interested in ways to use DNN to represent solutions while ensuring a given asymptotic form.

[AKP20, AKP20a] capture asymptotic behavior by a spline interpolator while in the interior, the solution is approximated by a linear combination of Gaussian kernels. The centers and widths of these kernels are given by artificial neural networks ([AKP20] calls this construction “Constrained Radial Layer” or “Gaussian Radial Layer”). They are constrained such that the resulting linear combination has zero asymptotes and is numerically zero in the asymptotic region beyond a certain distance.

However, a simpler construction without a special layer and without constraints might be preferred.

One such construction is the following: one represents the solution inside the nonasymptotic domain as the sum of a function which matches the asymptotic behaviors and of a product of a polynomial term that enforces zero asymptotes (“*zasymptotic*”) and a DNN:

$$f(x) = \text{asymptotic}(x) + \text{DNN}(x) * \text{zasymptotic}(x). \quad (1)$$

Outside the domain, $f(x)$ is just $\text{asymptotic}(x)$ and the DNN only influences the solution within the domain.

Here, we test this construction in one dimension for function approximation and regression, with two kinds of loss functions, one only involving value approximation (VML - vanilla machine learning) and the other one involving both value and derivative approximation (DML - differential machine learning).

We will show on examples that adding asymptotic behavior with fixed or trainable coefficients improves the approximation and convergence for both function approximation and regression.

The paper is organized as follows - we first introduce the approach and setting(s) in section 2, then consider some first simple function approximation in section 3 to demonstrate the methodology. In section 4, we progress to approximating a Black-Scholes function with its asymptotic behavior. In section 5, we learn a Black-Scholes function with asymptotic form from payoff and potentially payoff derivative samples generated by simulation. We conclude in section 6.

2 Approach and Set-Up

We are trying to learn a function f from function values (and possibly additional derivative values), or to learn a conditional expectation function f with $f(x) = E(Y|X = x)$ from samples of the target random variate Y and the conditioning X (and possibly sample-wise derivatives of the target random variate with respect to the conditioning). In addition, through observation, limiting behavior, or asymptotic methods, we know the asymptotic behavior or form of f , either with free parameters or with already given fixed parameters. Standard deep learning approaches with generic DNN do not take advantage of that form and thus, standard DNN approximations resulting from such approaches do not satisfy or approximate this asymptotic behavior well.

In general, this availability of asymptotic forms splits the domain of f into an asymptotic domain where the asymptotic form is exact or a good approximation of the function to be sought and the non-asymptotic domain where the function needs to be learned otherwise. One possible approach is to extend, if possible, the asymptotic form(s) from the asymptotic domain to a function defined on the entire domain and to only learn the difference to the asymptotic form. For general domains, finding such extensions is not trivial and might require both mathematical and computational effort. Often, asymptotic domains are given by the complement of tensor products of intervals (“boxes”) or by the complement of a hypersphere. For the box case, spline interpolation (in particular univariate or multivariate cubic interpolation) is a natural starting point.

In symbols, $f(x) = \text{extension(asymptotic)}(x) + \text{nonasymptotic}(x)$. $\text{nonasymptotic}(x)$ has to be such that it is zero in the asymptotic domain and smoothly blends into zero so as to not affect the asymptotic form. We will denote the extension of the asymptotic form by $\text{asymptotic}(x)$ as well and thus write:

$$f(x) = \begin{cases} \text{asymptotic}(x) & \text{in asymptotic domain} \\ \text{asymptotic}(x) + \text{DNN}(x) \times \text{zasymptotic}(x) & \text{elsewhere} \end{cases} \quad (2)$$

where $\text{zasymptotic}(x)$ ensures that its product with $\text{DNN}(x)$ does smoothly paste into the asymptotic form in the asymptotic domain.

It is sometimes the case that the derivatives of the function to be approximated are given or can be approximated well. In such cases, approaches that take derivatives into account might be of interest, in particular if the improvement in efficiency from using derivatives outweighs the effort to compute or approximate the derivatives needed, or where the derivatives of the function are to be approximated also. Similarly, for the regression case, the process that generates the samples of the target variate and gives the values of the conditioning might allow one to efficiently generate sample-wise derivatives as well, making sample-wise derivatives easily accessible for regression as well. In these cases, the extension of the asymptotic form and the approximation in the nonasymptotic domain have to respect the derivatives as well and have to lead to an approximation where derivatives match as well.

Here, we consider the one-dimensional case. Then,

$$f(x) = \begin{cases} \text{asymptotic}(x) & x \leq LL \\ \text{asymptotic}(x) + \text{DNN}(x) \times \text{zasymptotic}(x) & LL \leq x \leq UL \\ \text{asymptotic}(x) & UL \leq x \end{cases} \quad (3)$$

We will concentrate on the case of linear asymptotic forms.

$$\text{asymptotic}(x) = \begin{cases} LS(x - LL) + LI & x \leq LL \\ US(x - UL) + UI & UL \leq x \end{cases}. \quad (4)$$

We use cubic interpolation to extend this to a continuous function with continuous derivatives:

$$\text{asymptotic}(x) = \begin{cases} LS(x - LL) + LI & x \leq LL \\ (x - LL)(UL - x)(a_0(x - LL) + \tilde{LS}) + (s_0(x - LL) + LI) & LL \leq x \leq UL, \\ US(x - UL) + UI & UL \leq x \end{cases} \quad (5)$$

with $s_0 := \frac{UI-LI}{UL-LL}$, $\tilde{LS} := \frac{LS-s_0}{UL-LL}$, $\tilde{US} := \frac{s_0-US}{UL-LL}$, and $a_0 := \frac{\tilde{US}-\tilde{LS}}{UL-LL}$. ([AKP20] present a more general version, but our form is enough for our purposes.) For alternative asymptotic forms, we would use these forms below LL and above UL , and set LS and LI respective US and UI to the value and derivative of these forms at LL and UL and otherwise proceed in the same way. [AKP20] present an extension by multivariate splines for the multivariate box case and we could use their construction or alternative constructions for multivariate extensions of our approach as well.

[AKP20] ensure smooth pasting to the asymptotic forms by representing $zasympot(x)$ as a linear combination of Gaussian kernels, with a more involved domain decomposition (asymptotic domain, "no-mans land", and inner domain). Here, we will use simpler forms.

For the $zasympot(x)$, we use a scaled polynomial:

$$zasympot(x) = \begin{cases} 0 & x \leq LL \\ (x - UL)^2(x - LL)^2(\frac{1}{LL \cdot UL}) & LL \leq x \leq UL \\ 0 & UL \leq x \end{cases} \quad (6)$$

Appropriate scaling is important for normalization and easier learning, in particular when several functions and/or sets of parameters are to be learned simultaneously.

The parameter are as follows: UL - upper level or limit (beyond which the asymptotic form is used), with US and UI being the upper slope and intercept of the linear asymptotic form or the derivative and value of the asymptotic form in general, LL - lower level or limit (beyond which the asymptotic form is used), with LS and LI being the lower slope and intercept of the linear asymptotic form or the derivative and value of the asymptotic form in general.

We consider applications in function approximation and in regression problems. For function approximation based on only function values (VML - vanilla machine learning), we are given samples x_i and y_i and want to learn a function f such that $y_i = f(x_i)$. We are using the VML loss function

$$vmlloss = \sum_i (y_i - f(x_i))^2 \quad (7)$$

and are looking for a function f that minimizes that loss function. In standard deep learning, we would represent the entire function f by a deep neural network DNN, whereas for given asymptotic forms, we use the form of f proposed above in (3) with (5) and (6) to enforce the given asymptotic behavior. In the function approximation case, the VML loss function would have a minimum of zero.

For function approximation based on function and derivative values (DML - differential machine learning), we are given samples $x_i, y_i, \frac{dy_i}{dx}$ and want to learn a function f such that $y_i = f(x_i)$ and $\frac{dy_i}{dx} = f'(x_i)$. We are using the DML loss function

$$dmlloss = \sum_i (y_i - f(x_i))^2 + \lambda \sum_i \left(\frac{dy_i}{dx} - f'(x_i) \right)^2 \quad (8)$$

and are looking for a function f that minimizes that loss function.¹ In standard deep learning, we would represent the entire function f by a deep neural network DNN, whereas for given asymptotic forms, we use the form of f proposed above in (3) with (5) and (6) to enforce the given asymptotic behavior. In the function approximation case, the DML loss function would have a minimum of zero, but will be larger than the VML loss function since it measures both approximation of values and derivatives.

For the regression setting where we are trying to approximate conditional expectations from samples, we are given sample $x_i \sim X$ and $y_i \sim Y$, and we are looking to learn a function that is equals to or approximates the conditional expectation function $f(x) = E(Y|X=x)$. We are using the same VML loss function as for function approximation. The minimum loss now in general will be larger than zero since in general there will be irreducible randomness, Y is still random even conditioned on X .

In many situations, Y can be represented by a function of X and some additional randomness (independent of X) which we will call Ω , $Y = h(X, \Omega)$. For instance, X could be random initial values of an SDE or random inputs to some function. The function h is often either explicitly given or given as some program or procedure. Appropriate derivatives of the function can then be computed by appropriate algorithmic differentiation of the program. We are given samples $x_i \sim X$, $y_i = h(x_i, \omega_i)$, and can efficiently compute or approximate sample-wise derivatives $\frac{dy_i}{dx} = \frac{dh}{dx}(x_i, \omega_i)$. We can then use DML and the DML loss function as above, but the minimum loss will be not zero and larger than the VML

¹ λ allows the weighting of fitting of derivatives relative to fitting of function values as well as addressing different scales of function values and derivatives. In this paper and on our examples, we assume $\lambda = 1$, but other values of λ might be appropriate for other settings. Our methods allow for general λ .

loss since we also are approximating the conditional expectations of the derivatives. The approach also covers the case where X are intermediate values in some computation that are generated from some fixed initial values and additional randomness Ω^X as in $X = g(\Omega^X)$.

For testing examples where we know the function or the conditional expectation to be approximated, we can compute the error by comparing against that function and approximate the minimum loss by evaluating that loss function against generated samples. For the regression and conditional expectation setting, [CG21, CG23] offers ways to compute estimates and guarantees.

In DML without asymptotic forms, standard deep learning frameworks implement computation of derivatives by back propagation which is a special form of adjoint algorithm differentiation a.k.a. reverse mode so that f' for the DNN function f is easily and efficiently available. Including asymptotic forms, the f is now defined piece-wise and somewhat more complicated. It can be differentiated either through the framework or explicitly as

$$f'(x) = \begin{cases} \text{asymptotic}'(x) & x \leq LL \\ \text{asymptotic}'(x) + DNN'(x) \times z\text{asymptotic}(x) + DNN(x) \times z\text{asymptotic}'(x) & LL \leq x \leq UL \\ \text{asymptotic}'(x) & UL \leq x \end{cases} \quad (9)$$

where $DNN'(x)$ is given by adjoint algorithmic differentiation or back propagation.

3 A first function approximation example

Here, we study approximation of an example function

$$f(x) = \begin{cases} \text{asymptotic}(x) & x \leq LL \\ \text{asymptotic}(x) + x \times z\text{asymptotic}(x) & LL \leq x \leq UL \\ \text{asymptotic}(x) & UL \leq x \end{cases} \quad (10)$$

with $LL = -5$, $LI = 0$, $LS = 50$, $UL = 5$, $UI = 0$, and $US = 50$.

Fig. 1 shows the example function with associated derivative and its components while Fig. 2 shows the non-asymptotic component of the example function with associated derivative and its components.

Now that we have defined the function, we generate input data for training using 50,000 randomly selected x from a uniform distribution between -10 and 10 along with associated true function and derivative values y and $\frac{dy}{dx}$. In a first approach, we learn the parameters in the asymptotic form through stochastic gradient descent or Adam [KB14] at the same time the parameters of the neural network DNN are learned (“trainable parameters”). In a second approach, we use fixed given values for the parameters (“fixed parameters”). We tested using the true values of the asymptotic parameters (here, LS, LI, US, and UI) used to generate the samples as well as estimating the asymptotic parameters (LS, LI, US, and UI) through least squares in the asymptotic regions separately, possibly constraining the intercept to ensure good fitting of function values at LL and UL. We found that for our examples, the difference between using true values and using values determined with least square estimation was negligible. Thus, we present the results with parameters determined by least square estimation in the asymptotic regions for all our examples, since this would be also applicable if true parameters are not known.

The hyperparameters for the neural network used for training on normalized inputs are: input dimension = 1, output dimension = 1, hidden layers with nodes = [20, 20], activation function = softplus, optimizer = Adam, epochs = 200, and learning rate = 0.05.

We evaluate the performance of the trained DNNs from different methods on a test dataset. Fig. 3 shows the results of the trained deep neural network using VML without asymptotic treatment, Fig. 4 shows the results of the trained deep neural network using VML with asymptotic treatment (trainable parameters) and Fig. 5 shows the results of the trained deep neural network using VML with asymptotic treatment (fixed parameters). When looking at the difference to the true values, it is evident that VML in conjunction with asymptotic treatment leads to faster and accurate convergence, compared to VML without asymptotic treatment. This conclusion can also be obtained on examining the corresponding difference and loss graphs as shown in Fig. 6. Among asymptotic methods, we observe that fixed asymptotic parameters lead to better results than trainable asymptotic parameters as is natural to expect. Comparing DML without and with asymptotic treatment, we are led to the same conclusion. Fig. 7 shows the results of the trained deep neural network using DML without asymptotic treatment, Fig. 8 shows the results of the trained deep neural network using DML with asymptotic treatment (trainable parameters), Fig. 9 shows the results of the trained deep neural network using DML with asymptotic treatment (fixed parameters) and Fig. 10 shows the corresponding difference and loss graphs.

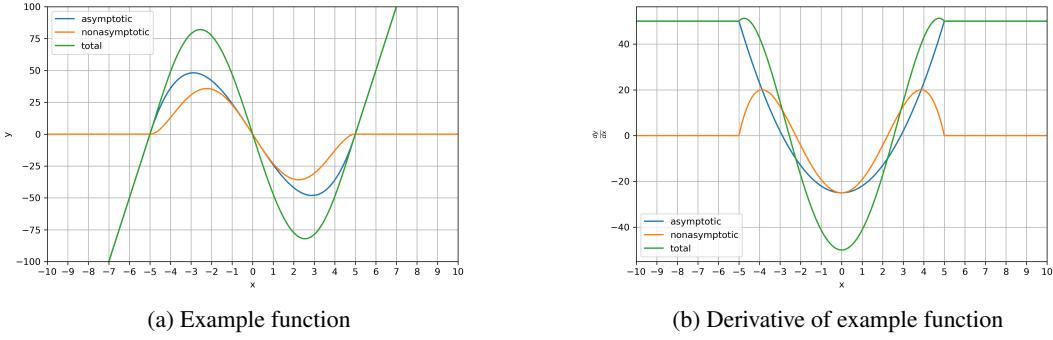


Figure 1: First function approximation: The example function with associated derivative and its two components - asymptotic and non-asymptotic part.

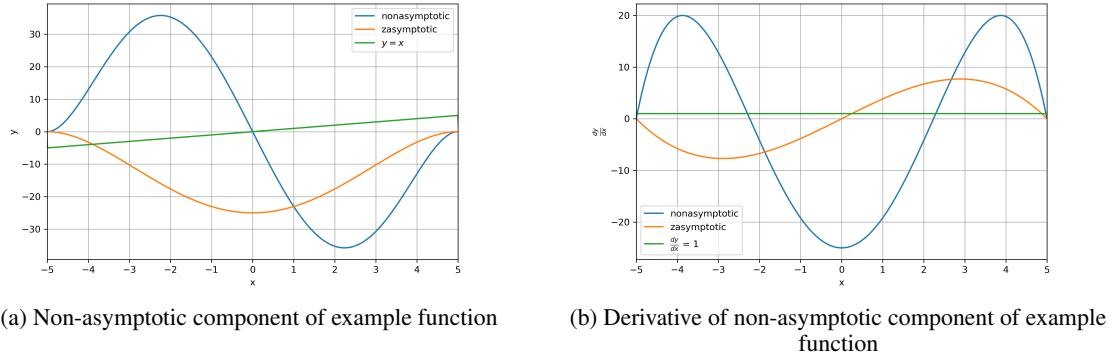


Figure 2: First function approximation: The non-asymptotic component of the example function with associated derivative and its two components - NN and non-NN (zasymptotic)

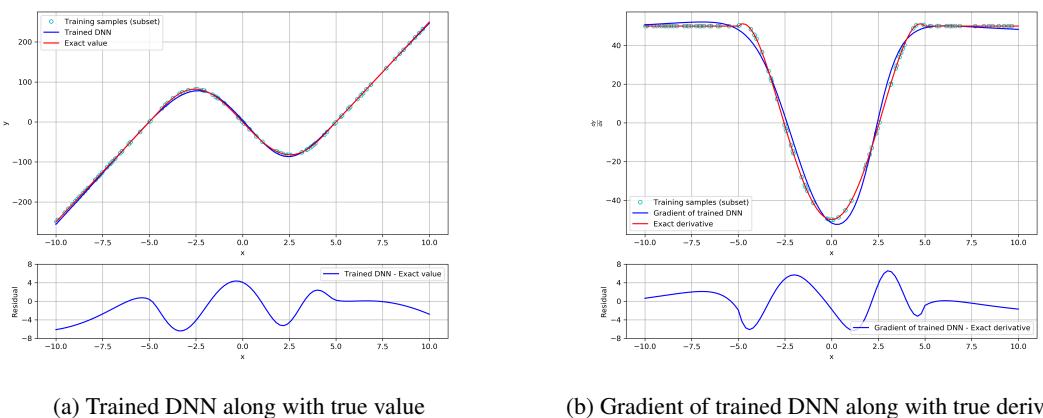
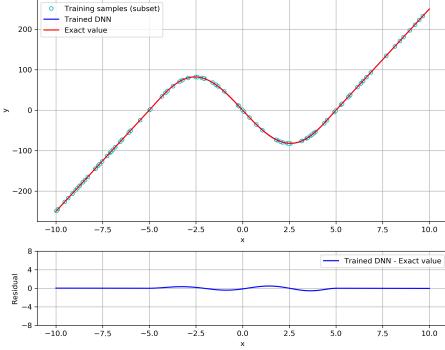
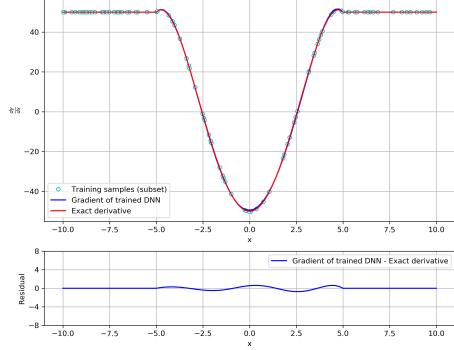


Figure 3: First function approximation: Vanilla Machine Learning without asymptotic treatment where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

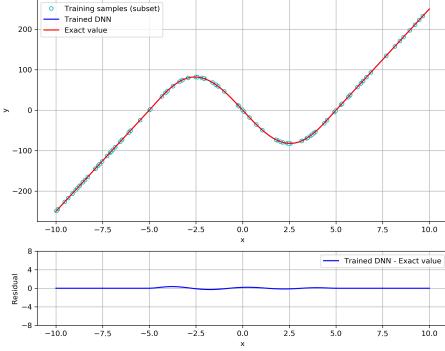


(a) Trained DNN along with true value

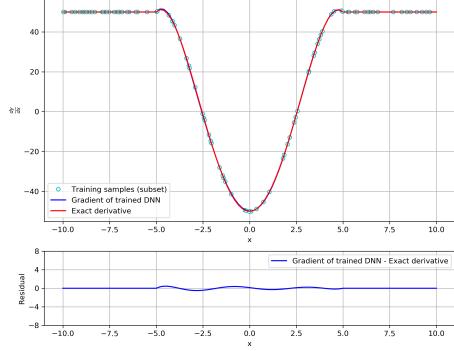


(b) Gradient of trained DNN along with true derivative

Figure 4: First function approximation: Vanilla Machine Learning with asymptotic treatment (trainable parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.



(a) Trained DNN along with true value



(b) Gradient of trained DNN along with true derivative

Figure 5: First function approximation: Vanilla Machine Learning with asymptotic treatment (fixed parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

We also note that DML with asymptotic treatment performs better than just DML without any asymptotic treatment, demonstrating the effectiveness of our methodology. The scales for the graphs are kept fixed across different methods to facilitate easier comparison. Fig. 11 shows the comparison of DML and VML in difference and loss graphs.

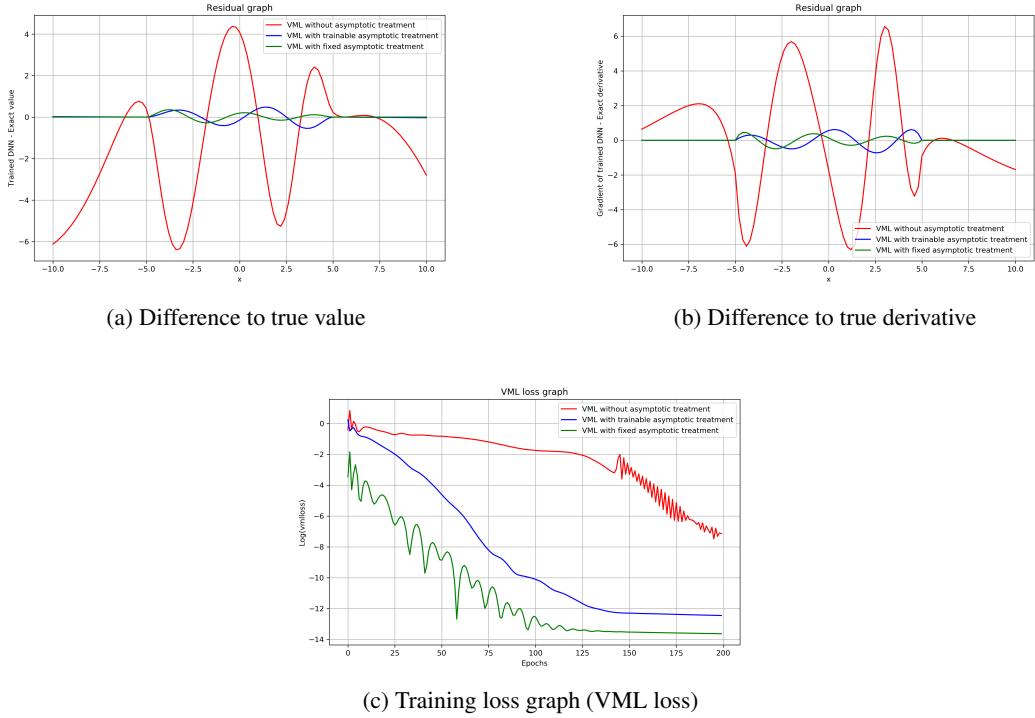


Figure 6: First function approximation: Difference and loss graphs for VML without and with asymptotic treatment

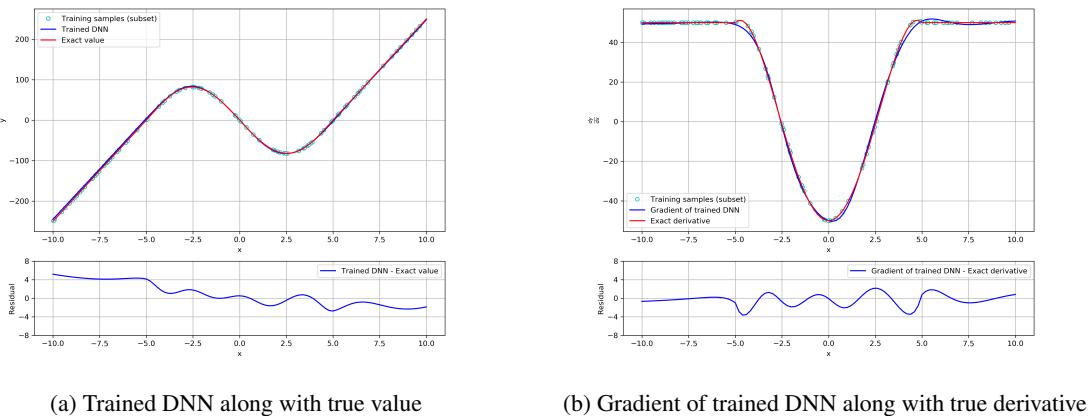


Figure 7: First function approximation: Differential Machine Learning without asymptotic treatment where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

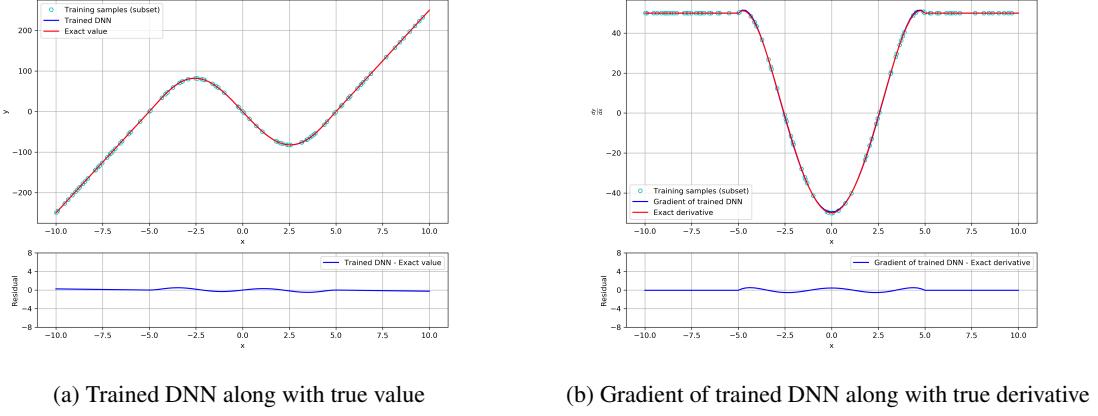


Figure 8: First function approximation: Differential Machine Learning with asymptotic treatment (trainable parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training.

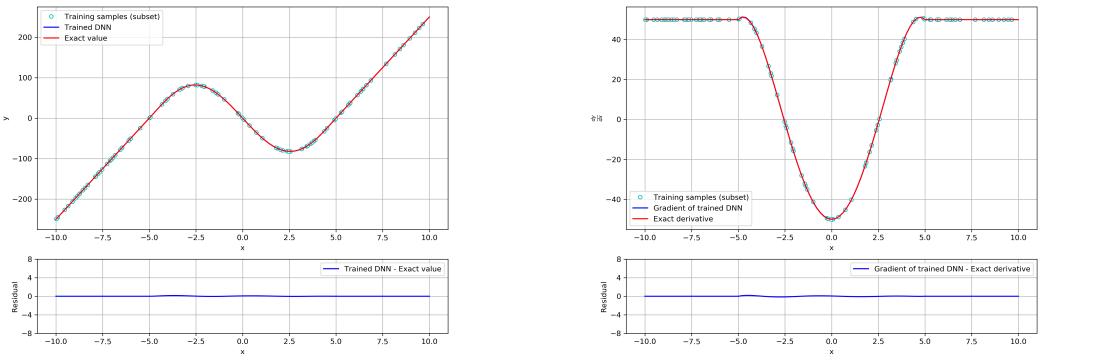


Figure 9: First function approximation: Differential Machine Learning with asymptotic treatment (fixed parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

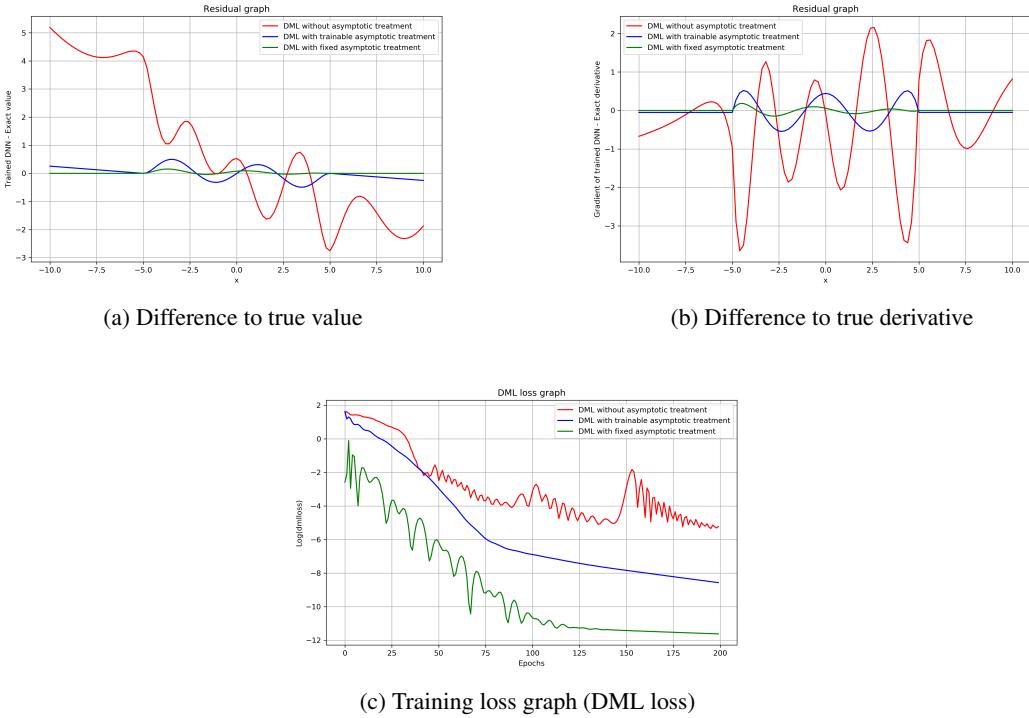


Figure 10: First function approximation: Difference and loss graphs for DML without and with asymptotic treatment

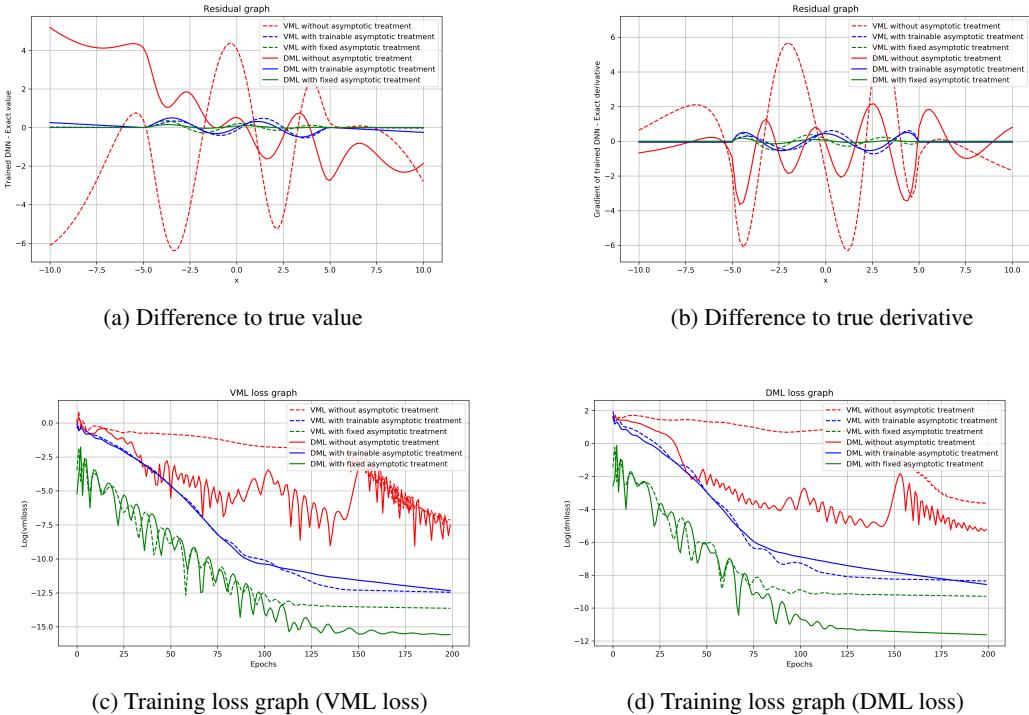


Figure 11: First function approximation: Difference and loss graphs for DML and VML (without and with asymptotic treatment)

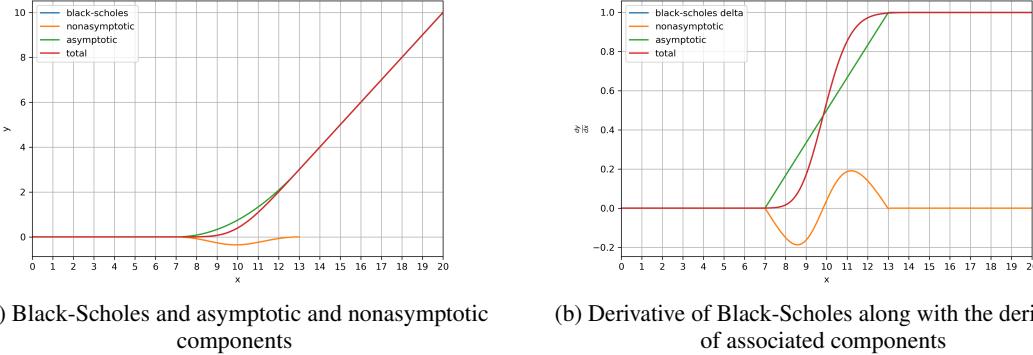


Figure 12: Black-Scholes function approximation: Function and its asymptotic and nonasymptotic components - values and derivatives.

4 Learning Black-Scholes formula with function approximation

In this section, we apply our methodology in function approximation to approximate the Black-Scholes function. The price of a vanilla option on a non-dividend paying stock $V(t) = e^{-r(T-t)} \mathbb{E}_t[(\phi(S(T) - K))^+]$ with Black-Scholes model is

$$\begin{aligned} V(t; S(t), K, r, \sigma, T) &= \phi(N(\phi d_+) \cdot S(t) - N(\phi d_-) \cdot K \cdot e^{-r(T-t)}) \\ d_+ &= \frac{1}{\sigma \sqrt{T-t}} (\ln \frac{S(t)}{K} + (r + \frac{\sigma^2}{2})(T-t)) \\ d_- &= d_+ - \sigma \sqrt{T-t} \end{aligned} \quad (11)$$

where ϕ is +1 (-1) for call (put), S is the (stochastic) price of equity, r is the (here constant) short rate, K is the strike, and T is the maturity.

We test on a call option with intermediate time $t = 1$ year, maturity $T = 2$ years, volatility $\sigma = 0.1$, short rate $r = 0\%$ and strike $K = 10$. We choose to vary $X = S(t)$ and denote the function to be approximated $Black-Scholes(x)$.

The Black-Scholes function² along with its asymptotic and non-asymptotic components is shown in Fig. 12. Nominal parameters for LL and UL are chosen to be 7 and 13 respectively. The asymptotic parameters (LS, LI) and (US, UI) are either kept trainable or estimated via least squares. We generate the input data for training using 50,000 randomly selected x from a uniform distribution between 0 and 20 along with associated true function and derivatives values y and $\frac{dy}{dx}$. The hyperparameters for the neural network used for training on normalized inputs are: input dimension = 1, output dimension = 1, hidden layers with nodes = [20, 20], activation function = softplus, optimizer = Adam, epochs = 200, and learning rate = 0.05.

Proceeding similarly as in the previous section, we evaluate the performance of the trained DNNs from different methods on a test dataset. Fig. 13 shows the results of the trained deep neural network using VML without asymptotic treatment, Fig. 14 shows the results of the trained deep neural network using VML with asymptotic treatment (trainable parameters) and Fig. 15 shows the results of the trained deep neural network using VML with asymptotic treatment (fixed parameters). Inspecting the difference as well as the loss graphs in Fig. 16, it is again evident that using VML in conjunction with asymptotic treatment leads to faster and accurate convergence as compared to VML without any asymptotic treatment. Fig. 17 shows the results of the trained deep neural network using DML without asymptotic treatment, Fig. 18 shows the results of the trained deep neural network using DML with asymptotic treatment (trainable parameters) and Fig. 19 shows the results of the trained deep neural network using DML with asymptotic treatment (fixed parameters). Observing the difference as well as the loss graphs in Fig. 20, we come to the same conclusion that DML with asymptotic treatment performs better than just DML without any asymptotic treatment, again demonstrating the effectiveness of our methodology. The scales for the graphs are kept fixed across different methods to facilitate easier comparison. Fig. 21 shows the comparison of DML and VML using difference and loss graphs.

²Or, more accurately, its representation within some nonasymptotic domain together with asymptotic forms.

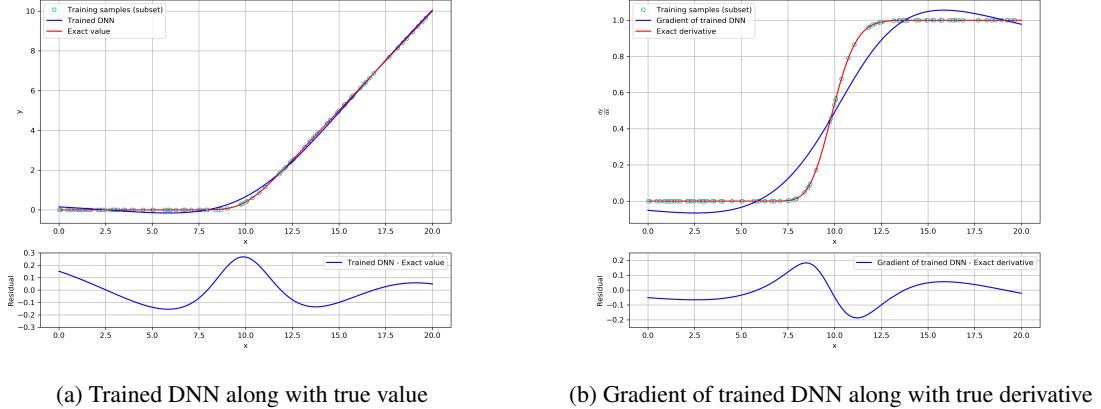


Figure 13: Black-Scholes function approximation: Vanilla Machine Learning without asymptotic treatment where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

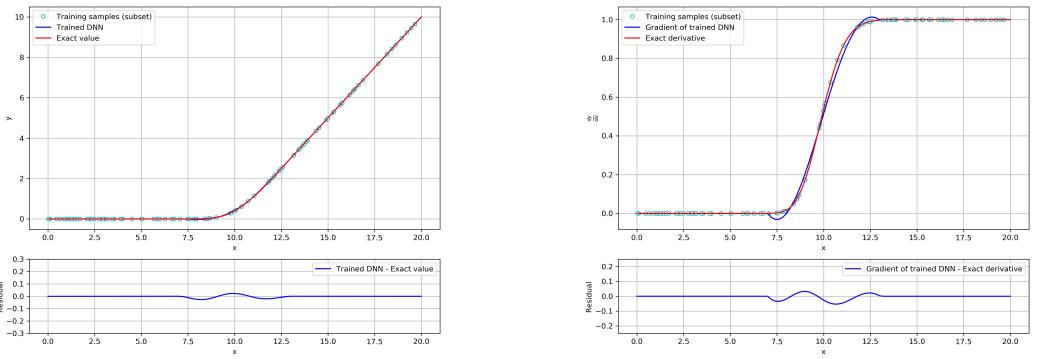


Figure 14: Black-Scholes function approximation: Vanilla Machine Learning with asymptotic treatment (trainable parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

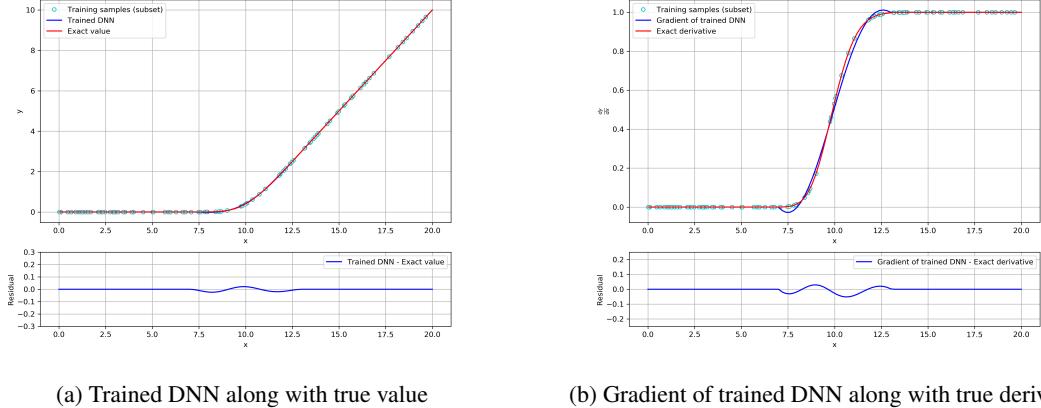


Figure 15: Black-Scholes function approximation: Vanilla Machine Learning with asymptotic treatment (fixed parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

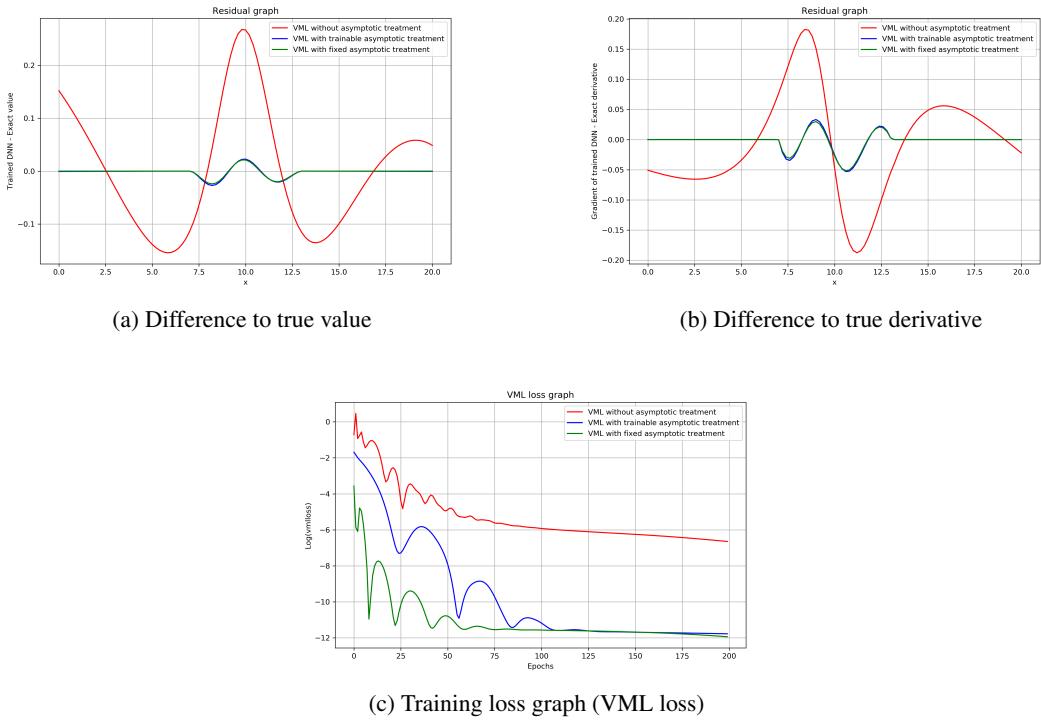


Figure 16: Black-Scholes function approximation: Difference and loss graphs for VML without and with asymptotic treatment

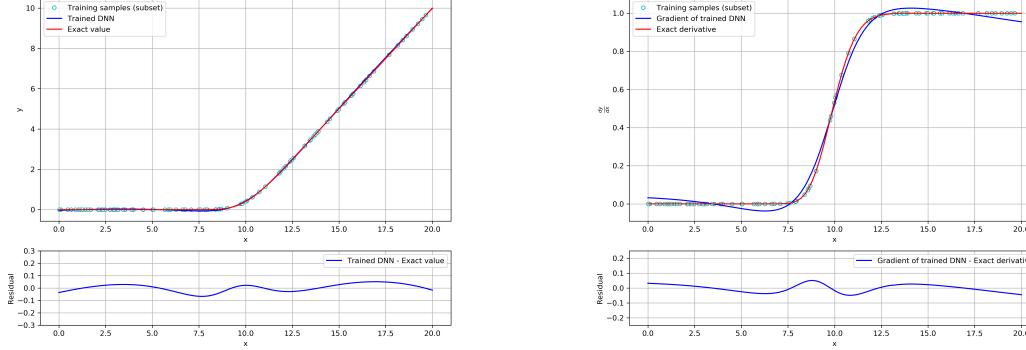


Figure 17: Black-Scholes function approximation: Differential Machine Learning without asymptotic treatment where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

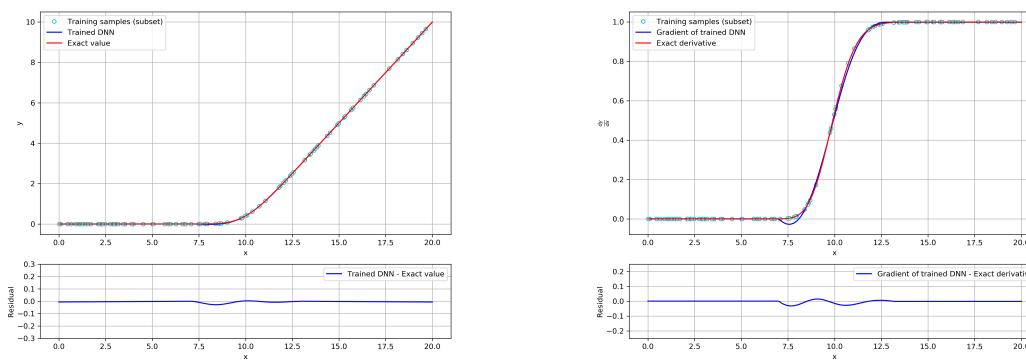


Figure 18: Black-Scholes function approximation: Differential Machine Learning with asymptotic treatment (trainable parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

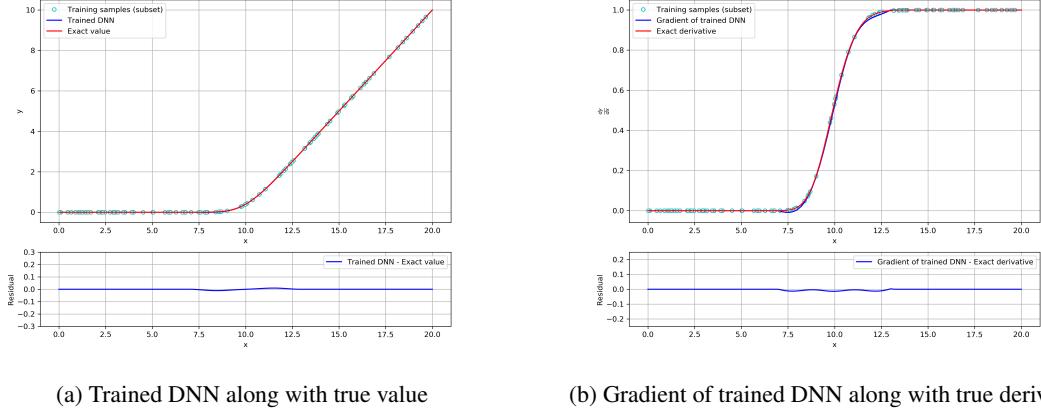


Figure 19: Black-Scholes function approximation: Differential Machine Learning with asymptotic treatment (fixed parameters) where results from DNN are in solid blue, true values are in solid red, and the blue scatter points represent a subset of samples used in training chosen randomly for visualization purposes only.

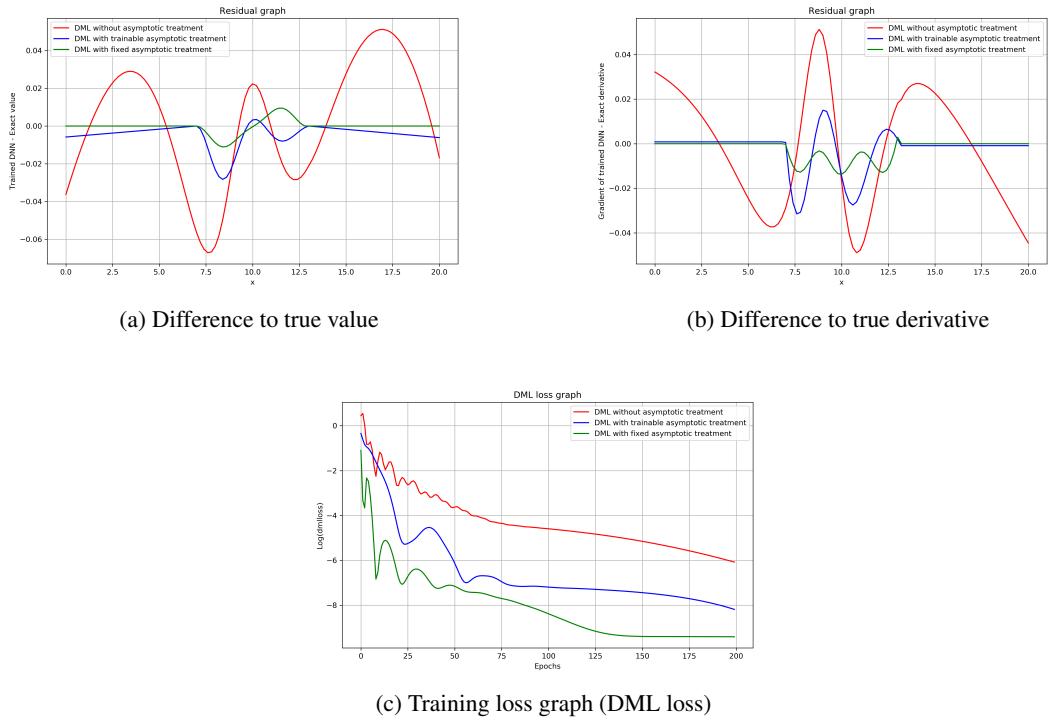


Figure 20: Black-Scholes function approximation: Difference and loss graphs for DML without and with asymptotic treatment

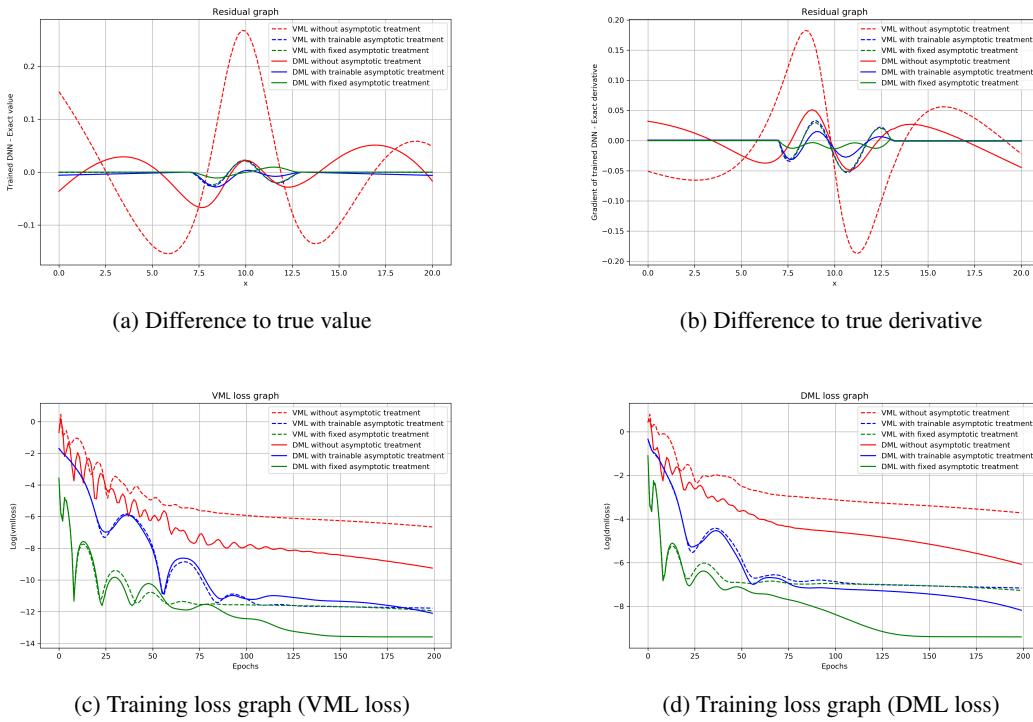


Figure 21: Black-Scholes function approximation: Difference and loss graphs for DML and VML (without and with asymptotic treatment)

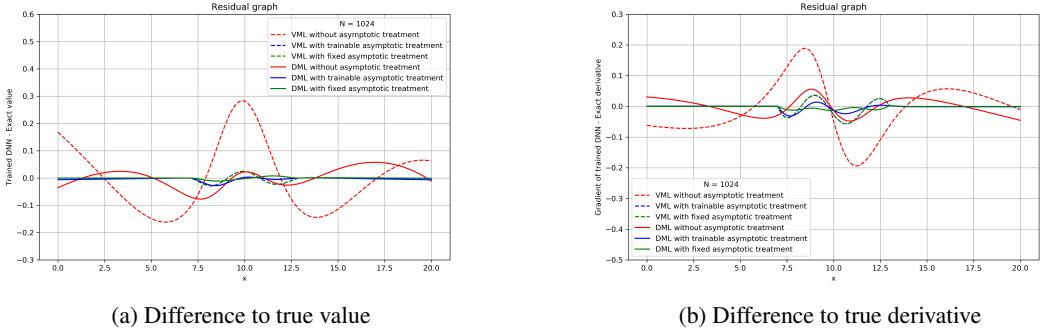


Figure 22: Black-Scholes function approximation: Difference graphs for DML and VML (without and with asymptotic treatment) for sample size = 2^{10}

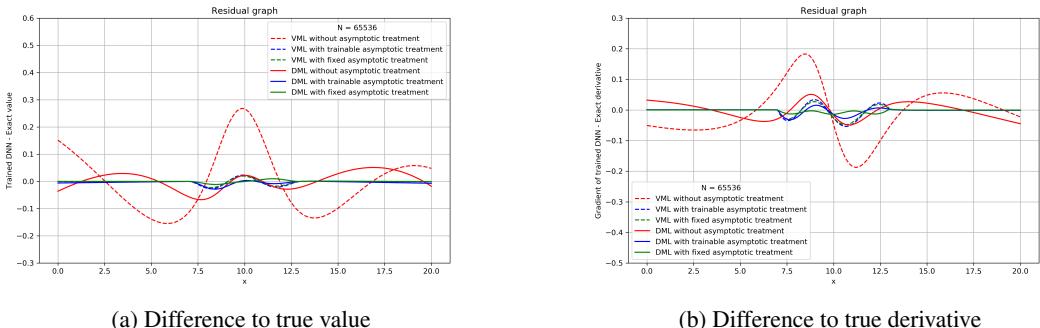


Figure 23: Black-Scholes function approximation: Difference graphs for DML and VML (without and with asymptotic treatment) for sample size = 2^{16}

We also performed tests on effects of sample size on results using VML and DML (without and with asymptotic treatment). Fig. 22 and Fig. 23 shows the results for a sample size of 2^{10} and 2^{16} respectively. It can be observed that the impact of sample size is minimal on the results of function approximation. There is some very slight improvement with increase in sample size, somewhat more pronounced in the loss function values.

Lastly, we also performed tests with non-zero interest rates r and observed very similar results. Adding r as a drift and in the discounting will change the shape and the coefficients slightly but otherwise does not seem to impact the results and we thus omit the plots.

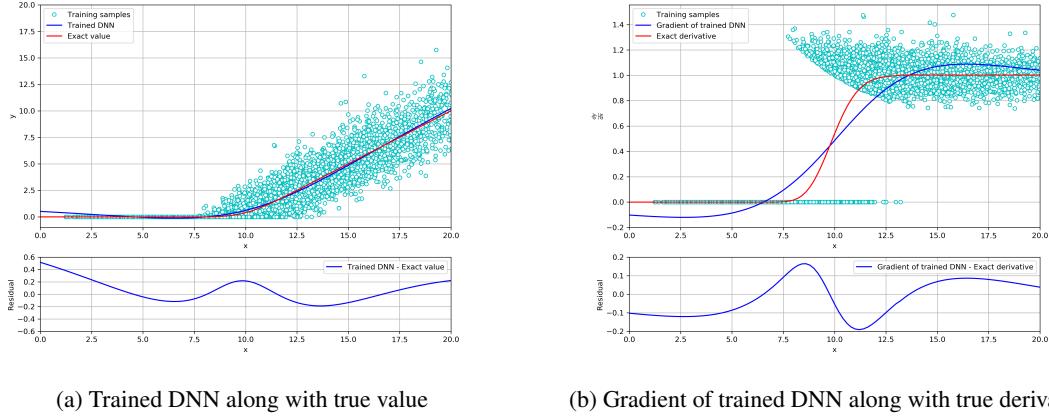


Figure 24: Black-Scholes model regression: Vanilla Machine Learning without asymptotic treatment where the scatter points represent the samples used in training.

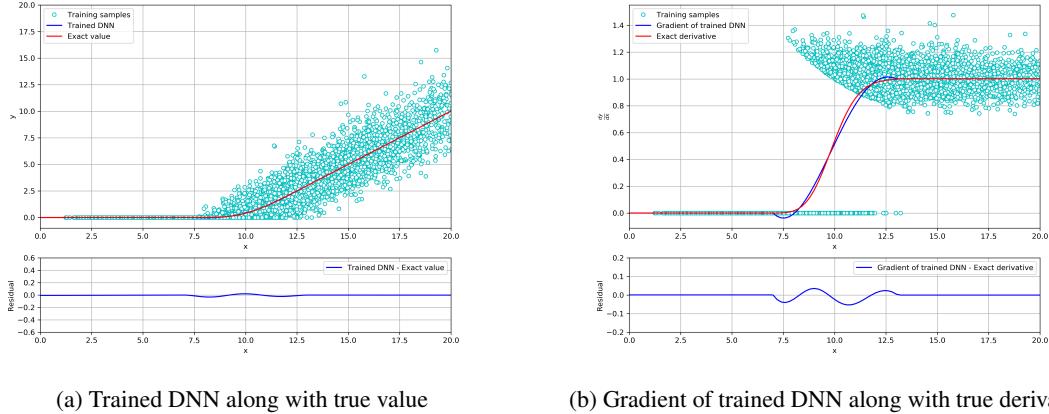


Figure 25: Black-Scholes model regression: Vanilla Machine Learning with asymptotic treatment (trainable parameters) where the scatter points represent the samples used in training.

5 Learning Black-Scholes formula through regression from simulated discounted payoff samples

Now that we have established the effectiveness of asymptotic treatment for approximating functions including Black-Scholes with faster and accurate convergence, we move on to the next natural step of learning the conditional expectation $\mathbb{E}[Y|X]$ given samples of initial or intermediate states x , values y , and sample-wise derivatives $\frac{dy}{dx}$. In the particular case of call option considered in the previous section, we use the spot price as conditioning X and the discounted samples of the payoff as Y . The conditional expectation function that we are after will represent the net present value (NPV) and will be approximated by the output of our neural network respective function. We simulate 10,000 stock price trajectories - here only prices S_t and S_T at intermediate time and maturity - under the Black-Scholes model, with log-Forward Euler (which leads to exact simulation) with one time-step from t to T , and record the discounted call option payoff $\tilde{C}_T = e^{-(T-t)}(S_T - K)^+$. We use $X = S_t$ and $Y = \tilde{C}_T$. In this test, we use the same setting as in the last section: intermediate time $t = 1$ year, maturity $T = 2$ years, volatility $\sigma = 0.1$, short rate $r = 0\%$ and strike $K = 10$. We follow the same procedure as in the preceding section for training the neural network on the simulated dataset and evaluate their performance on a test set.

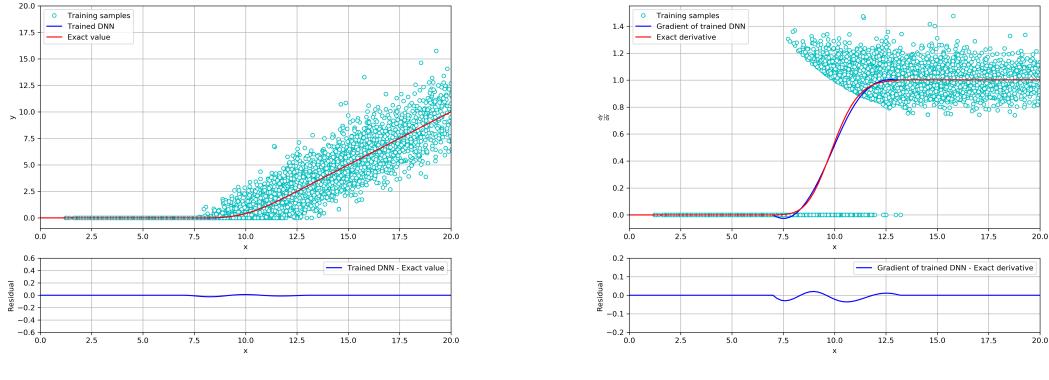


Figure 26: Black-Scholes model regression: Vanilla Machine Learning with asymptotic treatment (fixed parameters) where the scatter points represent the samples used in training.

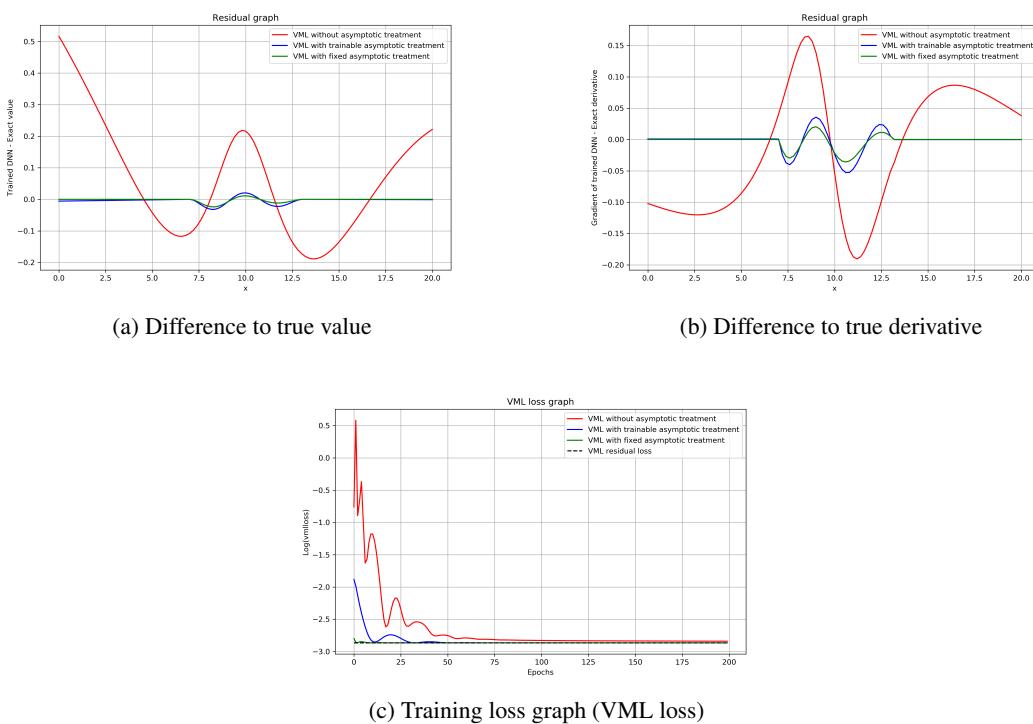


Figure 27: Black-Scholes model regression: Difference and loss graphs for VML without and with asymptotic treatment

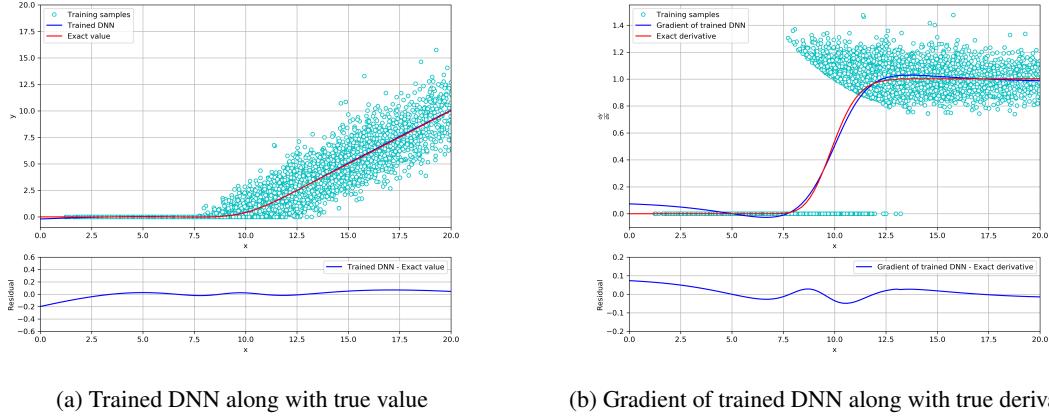


Figure 28: Black-Scholes model regression: Differential Machine Learning without asymptotic treatment where the scatter points represent the samples used in training.

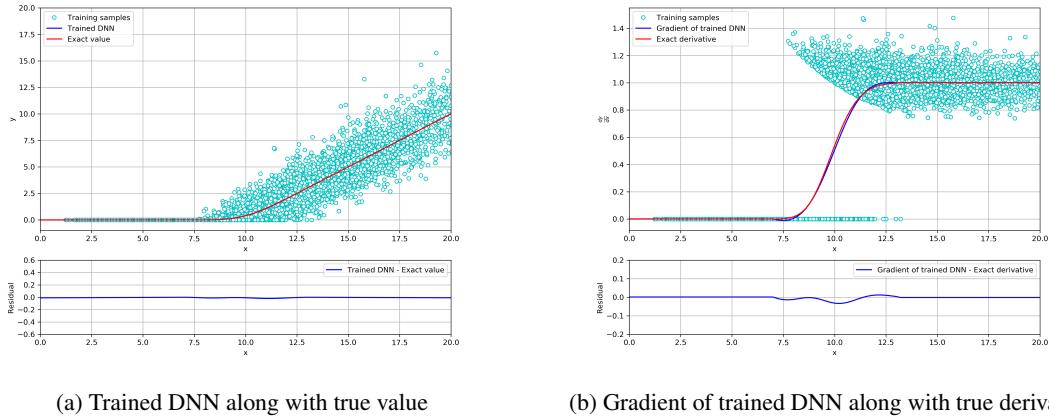
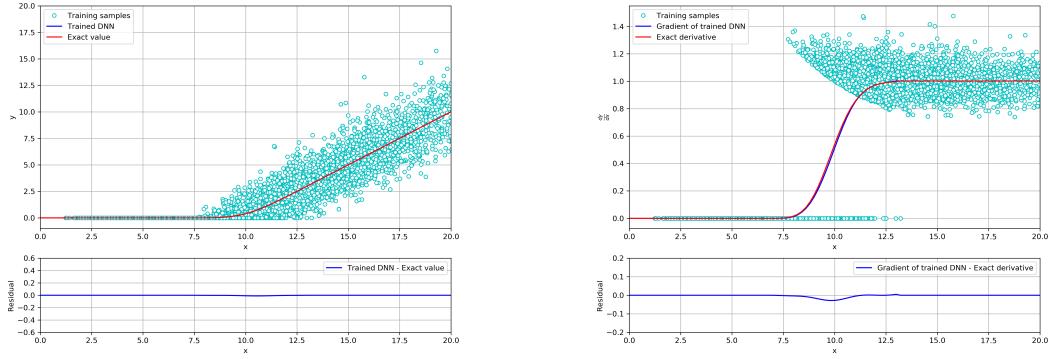


Figure 29: Black-Scholes model regression: Differential Machine Learning with asymptotic treatment (trainable parameters) where the scatter points represent the samples used in training.

Fig. 24 shows the results of the trained deep neural network using VML without asymptotic treatment, Fig. 25 shows the results of the trained deep neural network using VML with asymptotic treatment (trainable parameters) and Fig. 26 shows the results of the trained deep neural network using VML with asymptotic treatment (fixed parameters). Fig. 28 shows the results of the trained deep neural network using DML without asymptotic treatment, Fig. 29 shows the results of the trained deep neural network using DML with asymptotic treatment (trainable parameters) and Fig. 30 shows the results of the trained deep neural network using DML with asymptotic treatment (fixed parameters). The scales for the graphs are kept fixed across different methods to facilitate easier comparison. Inspecting the difference and loss graphs, we conclude that the proposed asymptotic treatment also provides better performance in regression problems. The associated difference and loss graphs are also shown in Fig. 27, Fig. 31 and Fig. 32.



(a) Trained DNN along with true value

(b) Gradient of trained DNN along with true derivative

Figure 30: Black-Scholes model regression: Differential Machine Learning with asymptotic treatment (fixed parameters) where the scatter points represent the samples used in training.

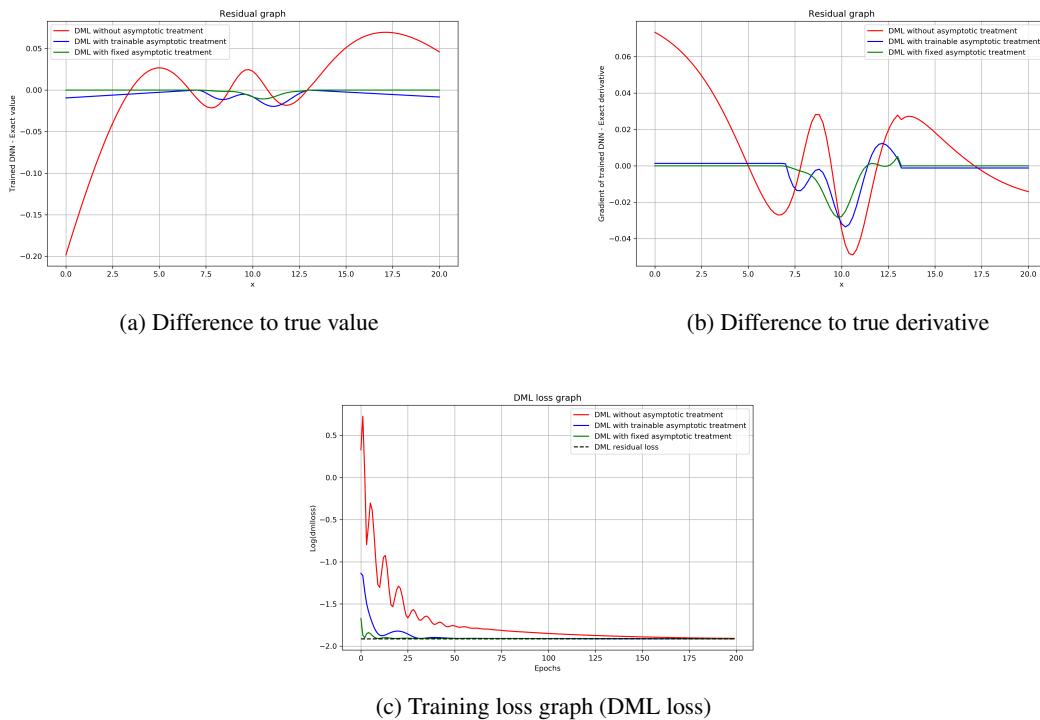


Figure 31: Black-Scholes model regression: Difference and loss graphs for DML without and with asymptotic treatment

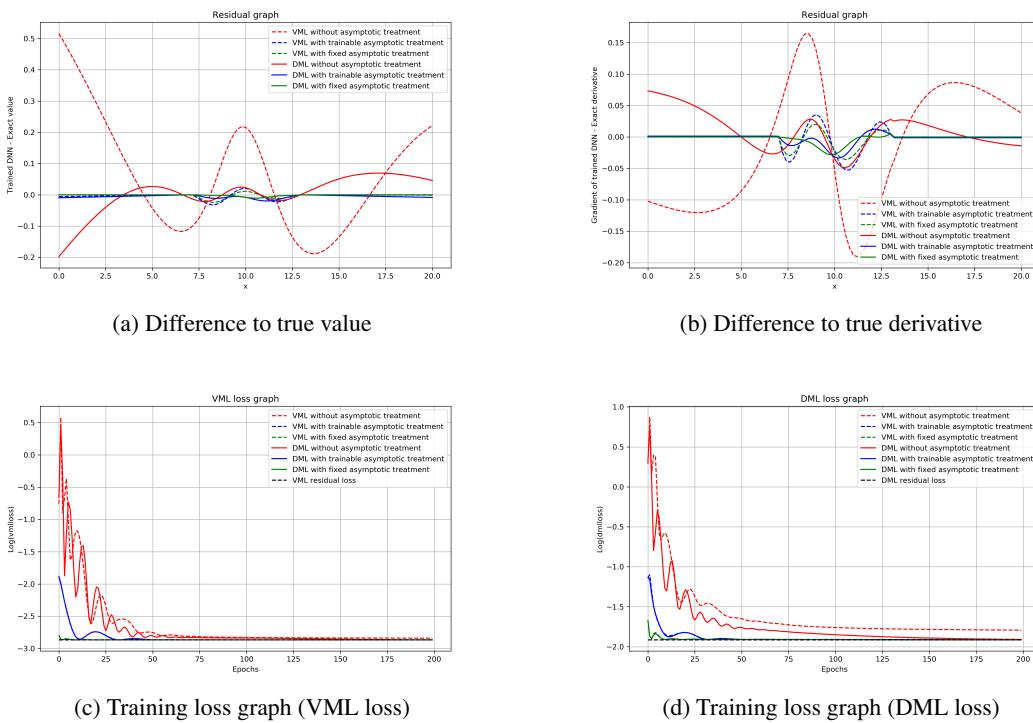


Figure 32: Black-Scholes model regression: Difference and loss graphs for VML and DML (without and with asymptotic treatment)

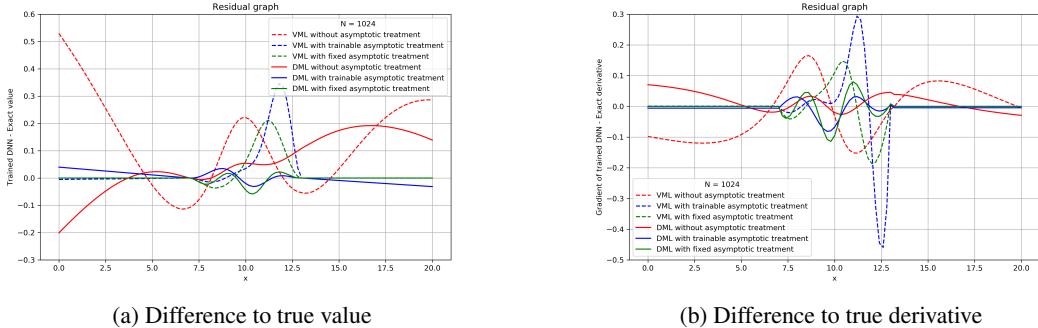


Figure 33: Black-Scholes model regression: Difference graphs for DML and VML (without and with asymptotic treatment) for sample size = 2^{10}

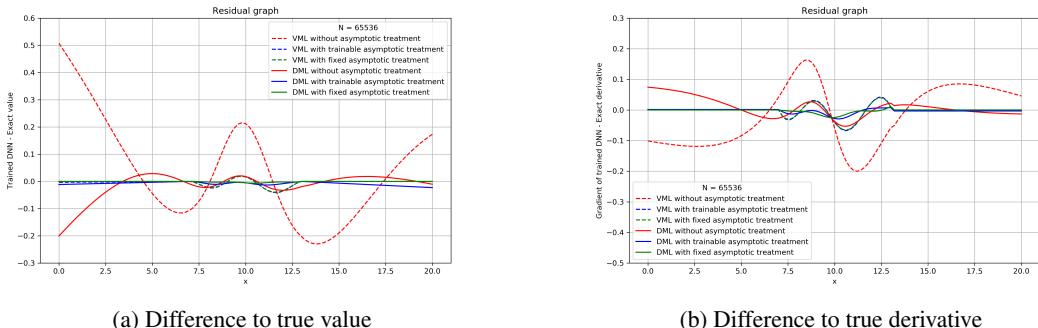


Figure 34: Black-Scholes model regression: Difference graphs for DML and VML (without and with asymptotic treatment) for sample size = 2^{16}

We also performed tests on effects of sample size on results using VML and DML (without and with asymptotic treatment). Fig. 33 and Fig. 34 shows the results for a sample size of 2^{10} and 2^{16} respectively. It can be observed that the impact of sample size is most prominent when asymptotic treatment is used in both VML and DML with higher sample size resulting in improved approximation. The impact is more clearly visible and sizable for the VML approaches (dashed lines) and smaller for the DML approaches (solid lines).

Lastly, we also performed tests with non-zero interest rates r and observed very similar results. Adding r as a drift and in the discounting will change the shape and the coefficients slightly but otherwise does not seem to impact the results and we thus omit the plots.

6 Conclusion

Often, true or approximate asymptotic behavior or forms are known for pricing functions in quantitative finance in asymptotic regimes and regions, given by known functional forms either with given parameters or with free parameters. Approximating such pricing functions with standard DNN through deep learning approaches will often not lead to good approximations of asymptotic behavior.

While some approaches in the literature enforce asymptotic forms [AKP20, AKP20a], they represent functions in the non-asymptotic domain by a linear combination of Gaussian kernels (on top of the extended asymptotic form), requiring computation of many exponentials during training and evaluation. In this paper, we proposed and tested a simpler form where instead of Gaussian kernels, the product of a simple polynomial (which assures proper matching of asymptotic behavior) and an unconstrained DNN is used. This approach leads to a simple implementation and good results in our tests, on two example function approximation and one example regression problem. It thus seems to be a promising choice and candidate for implementation in cases where asymptotic behavior is known.

In future work, we plan to test further cases with known asymptotic behaviors, including the also encountered exponential forms. We intend to work on extensions to multiple dimensions. We plan to apply such asymptotic treatment in the regression setting in our PDML approach for parametric pricers and calibration [PH23, PH24], and to the applications of the DML approach for regression to XVA [HS20, HS20a]. We will also pursue the application of such constructions in further areas beyond the function approximation and regression setting, such as in stochastic control approaches [FH23, H23], reinforcement learning [FH23, H23], and deepBSDE approaches [H19, H21, YHG20, YGH23].

7 Acknowledgements and Disclaimer

The authors thank Vijayan Nair for his support and suggestions regarding research, this work, and its presentation. We thank Orcan Ogetbil for his close reading and feedback improving the presentation in this paper.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Wells Fargo Bank, N.A., its parent company, affiliates and subsidiaries.

References

- [HS20] Brian Huge and Antoine Savine. Differential machine learning, 2020. URL <https://arxiv.org/abs/2005.02347>.
- [AKP20] Alexandre Antonov, Michael Konikov, and Vladimir Piterbarg. Neural networks with asymptotics control. *SSRN*, 2020. Also available at SSRN: <https://ssrn.com/abstract=3544698>.
- [AKP20a] Alexandre Antonov, Michael Konikov, and Vladimir Piterbarg. Deep asymptotics. *Risk Magazine*, January 2020. Preprint version available as SSRN 3544698.
- [CG21] Patrick Cheridito and Balint Gersey. Computation of conditional expectations with guarantees, 2023. URL <https://arxiv.org/abs/2112.01804>.
- [CG23] Patrick Cheridito and Balint Gersey. Computation of conditional expectations with guarantees. *Journal of Scientific Computing*, 95(1):12, 2023. Open access, preprint version available as arXiv preprint arXiv:2112.01804.
- [KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL <https://arxiv.org/abs/1412.6980>.
- [PH23] Arun Kumar Polala and Bernhard Hientzsch. Parametric differential machine learning for pricing and calibration, 2023. URL <https://arxiv.org/abs/2302.06682>.
- [PH24] Arun Kumar Polala and Bernhard Hientzsch. A case study on different one-factor Cheyette models for short maturity caplet calibration, 2024. URL <https://arxiv.org/abs/2408.11257>.
- [HS20a] Brian Huge and Antoine Savine. Differential machine learning: The shape of things to come. *Risk Magazine*, 2020. Preprint version available as arXiv preprint arXiv:2005.02347.
- [FH23] Ali Fathi and Bernhard Hientzsch. A comparison of reinforcement learning and deep trajectory based stochastic control agents for stepwise mean-variance hedging, 2023. URL <https://arxiv.org/abs/2302.07996>.
- [H23] Bernhard Hientzsch. Reinforcement learning and deep stochastic optimal control for final quadratic hedging, 2023. URL <https://arxiv.org/abs/2401.08600>.

- [H19] Bernhard Hientzsch. Introduction to solving quant finance problems with time-stepped FBSDE and deep learning, 2019. URL <https://arxiv.org/abs/1911.12231>. Also available at SSRN: <https://ssrn.com/abstract=3494359>.
- [H21] Bernhard Hientzsch. Deep learning to solve forward-backward stochastic differential equations. *Risk Magazine*, February 2021. Preprint version available as arXiv preprint arXiv:1911.12231.
- [YHG20] Yajie Yu, Bernhard Hientzsch, and Narayan Ganesan. Backward deepBSDE methods: Applications for nonlinear problems, 2020. URL <https://arxiv.org/abs/2006.07635>. Also available at SSRN: <https://ssrn.com/abstract=3626208>.
- [YGH23] Yajie Yu, Narayan Ganesan, and Bernhard Hientzsch. Backward deep BSDE methods and applications to nonlinear problems. *Risks*, 11(3):61, 2023. Open access, preprint version available as arXiv preprint arXiv:2006.07635.