MANAGEMENT SCIENCE
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0025-1909 | EISSN 1526-5501 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

Online Appendices for Deep Learning of Transition Probability Densities for Stochastic Asset Models with Applications in Option Pricing

Haozhe Su
Nottingham Business School, Nottingham Trent University, Nottingham NG1 4FQ, UK, Haozhe.Su@ntu.ac.uk

M.V. Tretyakov
School of Mathematical Sciences, University of Nottingham, Nottingham NG7 2RD, UK, Michael.Tretyakov@nottingham.ac.uk

David P. Newton
School of Management, University of Bath, Bath BA2 7AY, UK, dpn25@bath.ac.uk

This is a supplementary material to the paper "Deep Learning of Transition Probability Densities for Stochastic Asset Models with Applications in Option Pricing" published in Management Science journal.

Appendices

A. Training and Calculation Times

Training neural networks (NNs) to replace other numerical methods can be expensive but, once trained, the NNs can be used, e.g. in option pricing, with no further major computational effort to evaluate transition probability density functions (TPDFs), as illustrated in our paper. The speed of training depends on the GPU and the deep learning implementation library used. Multiple GPUs can, of course, accelerate the training and it is also possible to use server training for asynchronous training. Theoretically speaking, using such an asynchronous training strategy, n GPUs can increase the speed by n times. For our work, we did not have access to computers with multiple high-end GPUs but, nevertheless, our paper shows that training can still be achieved using a single GPU, either via the Google Colab Pro server or simply using a gaming computer (since gaming computers come with relatively powerful GPUs). More powerful computers could considerably accelerate the training but this is unimportant since a trained network can be used without further computational effort and becomes an ultra-fast generator of TPDFs, insertable into various numerical techniques (we illustrate using QUAD for option pricing) and, also, is portable to less powerful computers.

In Appendix A.1, we compare training speeds using 10 GPUs. Of these, 5 GPUs are from the gaming GPU category and the other 5 are from the data centre and high performance computing category. We show in Appendix A.2 that the online accessing time is very quick and independent of the number of points on which TPDFs need to be evaluated.

A.1. Training time using various GPUs

In this test, we select 5 NVIDIA gaming GPUs (RTX 3090, RTX 3080, RTX 2080 Ti, RTX 2070, RTX 1080 Ti) and 5 NVIDIA data centre GPUs (A100 SXM4, A40, V100, P100, T4). We refer readers to the NVIDIA official web page for their detailed specifications. The GPU memory size limits the amount of data which can be used in NN training as well as the complexity of NN architecture. For all the applications in this paper, GPU memory size was not a limiting factor; but to tackle more complex problems (e.g., with more parameters to train or training for higher dimensional models of underliers) GPUs with more memory might be required. At the same time, memory size is not a deciding factor for training speed which is determined by the number of GPU cores and the GPU architecture. It can be expected that more cores can accelerate parallel computation while newer generation GPUs deliver faster computations. In particular, RTX 3090, RTX 3080, A100 SXM4 and A40 are equipped with the latest NVIDIA Ampere GPU architecture (at the time of the writing) and all the other GPUs have previous generation GPU architectures. The older generation of the TensorFlow library (TensorFlow 2.3 or lower) cannot be used on Ampere architecture GPUs. However, we find that for GPUs with previous generation architectures, TensorFlow 2.2 and TensorFlow 2.3 work best, while newer or older versions perform worse. Thus, we use TensorFlow 2.3 on GPUs with previous generation architectures and TensorFlow 2.5 on GPUs with Ampere GPU architecture. The speed of training depends solely on the GPUs where thousands of NVIDIA's Compute Unified Device Architecture (Tensor or CUDA) cores perform highly efficient parallel computing. It might be expected that newer GPUs should perform much better than older ones and that higher end models are better, e.g., RTX 3090 better than RTX 3080 and RTX 3080 better than RTX 2080 Ti. This turns out not to be entirely the case, as can be seen from the test results below.

Figure 1 shows the training times using various GPUs. One can see that the most powerful data centre GPU A100 (at the time of the writing) tops this comparison, followed by another data centre GPU A40. Among gaming GPUs, the RTX 2080 Ti performs the best, even better than its successors RTX 3080 and 3090. This could be because TensorFlow 2.2 and 2.3 are more optimised for training the NN architecture used in this paper. If we use the same TensorFlow 2.5 on RTX 2080 Ti as on the Ampere GPUs, the computing time increases to 3.001, 4.9469, 4.8508, 3.7418, 5.6915 seconds on the five different models of underliers, respectively – a 50% to 100% increase

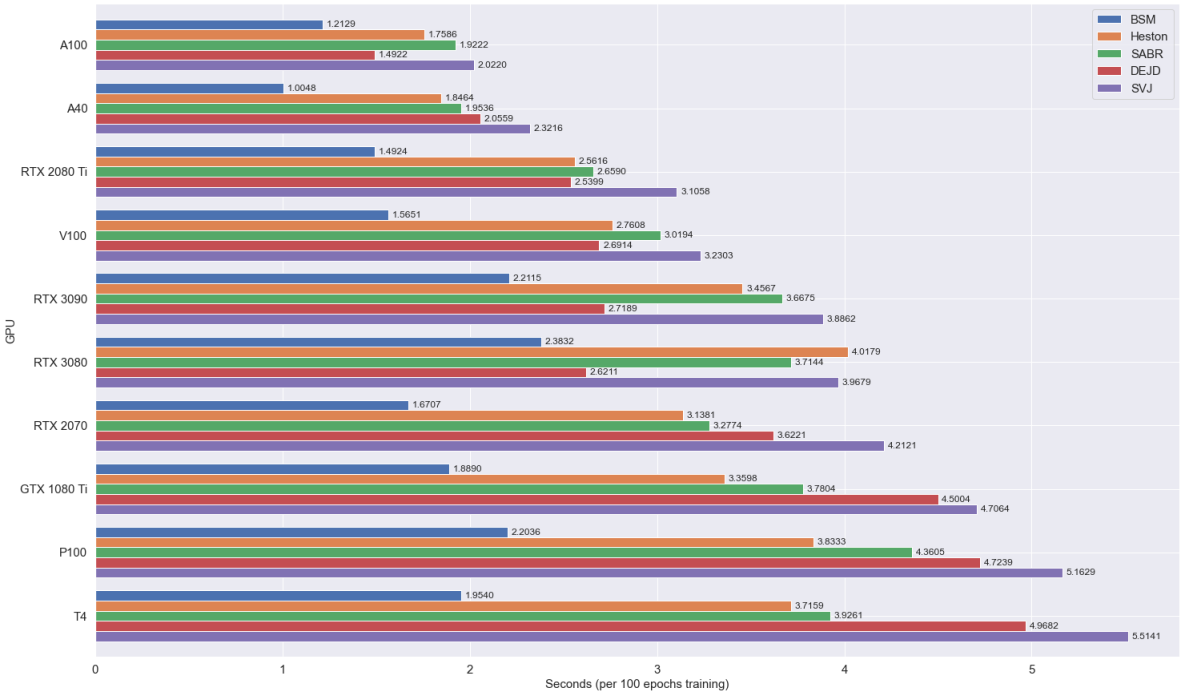

Figure 1 Training time of the 5 models of underliers considered in this paper per 100 epochs using various GPUs. Ranked by the speed of the stochastic volatility jump diffusion model (SVJ).

compared with using TensorFlow 2.3, making it worse than RTX 3080 and 3090. However, since RTX 3080 and 3090 cannot use TensorFlow 2.3 or below, the older GPUs such as RTX 2080 Ti are still better than the newer ones. Additionally, we find that the previous generation top gaming GPU RTX 2080 Ti performs a little bit better than the same generation data centre server GPU V100. This demonstrates that the more accessible and cheaper gaming GPUs are already sufficient for the purposes of this paper.

To conclude, we have shown that the NNs exploited in this paper can be trained using a single gaming GPU. As computing power advances, the training speed will certainly be improved. It is worth emphasising that the results in this paper come from training using Google Colab Pro servers (the two types of GPUs available in Google Colab Pro servers are P100 and T4). These two older GPUs can still be used to train NNs, although that may take much more time than using the latest GPUs. However, training is done offline (i.e., before the NN is used in computational finance), and in finance applications the offline training speed is not important in comparison with the online speed of accessing TPDFs. Next, we will show how fast online accessing speed is.

A.2. Calculation Time

Once a NN is trained and ready to use, the application calculation ("online") time, delivering densities, is very fast. Similar to the training, the calculations are also done in parallel with thousands of CUDA cores, meaning that the calculation speed does not depend on the number of points

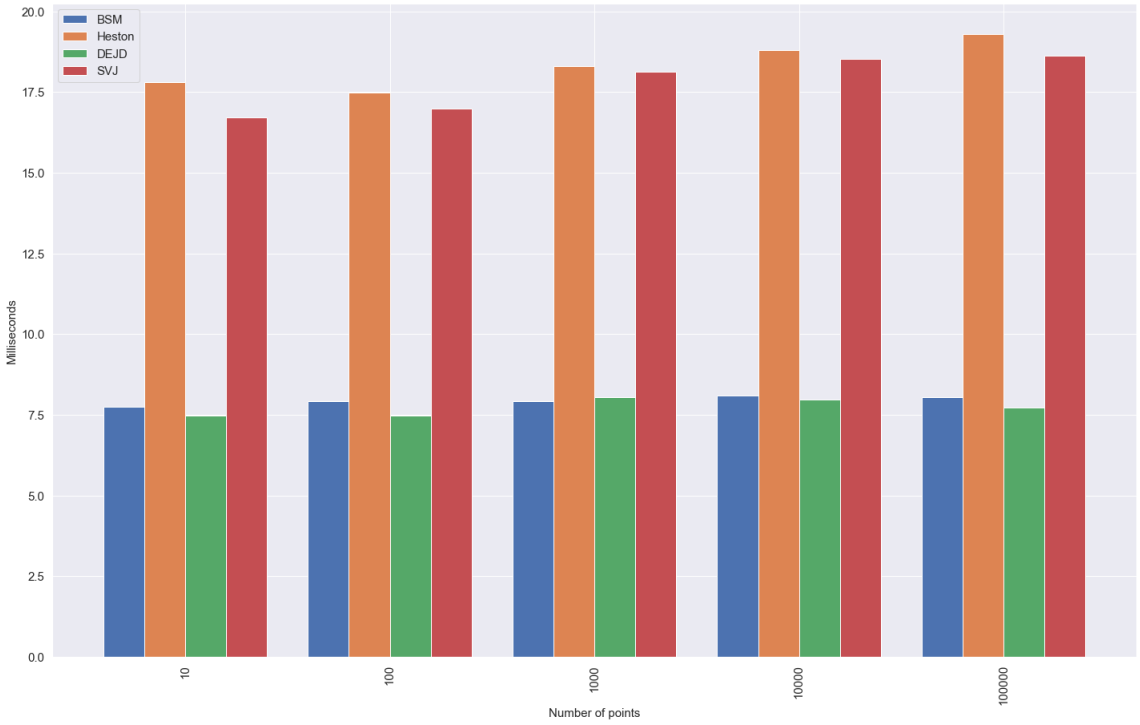

Figure 2 **Computing time of TPDFs for various models of underliers on a single GPU RTX 2070. The number of points the NN calculates at the same time are 10, 100, 1000, 10000, 100000.**

calculated. In Figure 2, we calculate the NN approximated densities using a single GPU RTX 2070. The NN computes the densities in a matter of a few milliseconds; the calculation time remains roughly the same regardless of the number of points calculated. Here, we note that the maximum number of points a GPU can handle depends on its memory size. The fast online usage time is another useful property of the NN approach shown here. The difference in computing time between one dimensional and two dimensional models of underliers is mainly due to the differentiation of the cumulative distribution function (CDF) to obtain the TPDF.

B. The performance of using Multilayer Perceptrons

In this appendix, we briefly discuss the performance of an alternative NN setup. In Sections 3-4 of the main paper, we utilised the DGM NN, which was introduced alongside the DGM algorithm by Sirignano and Spiliopoulos (2018). In this appendix we explore whether simpler network architectures can also be effective for training in the context of approximating TPDFs. Specifically, we examine one of the simplest NN structures, known as Multilayer Perceptrons (MLPs). MLPs are widely used in deep learning and are a type of feedforward NN (see, e.g., Goodfellow et al. (2016)).

MLPs consist of multiple layers of nodes, where each node in a layer is connected to every node in the subsequent layer. Figure 3 provides a visualisation of the MLP neural network architecture. The initial layer is referred to as the input layer, while the final layer is known as the output layer.

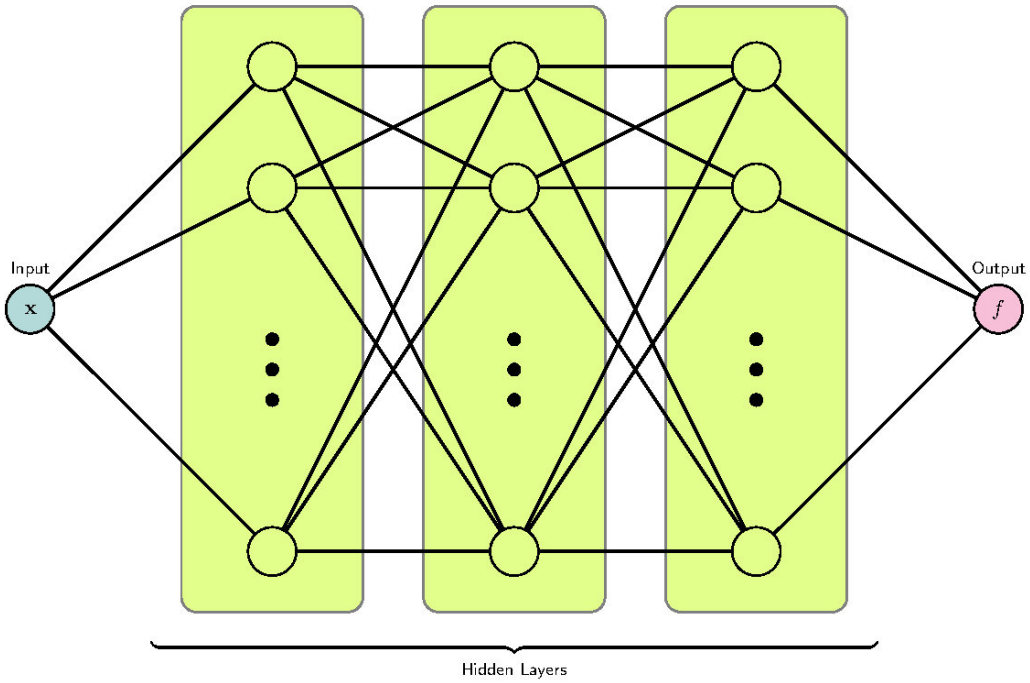

Figure 3 Illustration of the Multilayer Perceptrons NN architecture.

Any layers positioned between these two are referred to as hidden layers. The number of nodes in the input and output layers is determined by the specific problem, while the number of nodes in the hidden layers is a hyperparameter that must be selected.

Each node within a layer is connected to every node in the subsequent layer through weighted connections. During training, these weights are adjusted to enable the neural network to learn how to perform the desired task, such as classification or regression. MLPs derive their name from "perceptrons" because each node calculates a weighted sum of its inputs and applies an activation function to the result. The activation function introduces nonlinearity to the network, allowing it to model complex relationships between inputs and outputs.

Using the same notations as in Section 2.3.3 of the main paper, the MLP neural network can be expressed as follows:

$$\begin{aligned}
\mathbf{S}^1 &= \vartheta(\mathbf{W}^0\mathbf{x} + \mathbf{b}^\mathbf{0}) \\
\mathbf{S}^{\ell+1} &= \vartheta(\mathbf{W}^\ell\mathbf{S}^\ell + \mathbf{b}^\ell),\ \ell = 1, \ldots, L-1, \\
f(\mathbf{x};\boldsymbol{\theta}) &= \mathbf{W}\mathbf{S}^L + \mathbf{b},
\end{aligned}$$

where $\mathbf{x}$ represents the input layer, $\vartheta(\cdot)$ denotes the activation function, L signifies the number of hidden layers, and $\mathbf{W}, \mathbf{b}$ are the network parameters, i.e., weights and biases. The input $\mathbf{x}$ and the output f are described in Section 2.3.3. From the above expressions, we can observe that the MLP structure is simpler compared to the DGM network.

Table 1 Under the MLP network, the pricing errors of the NN approximated density of Kou's double exponential jump diffusion model, compared with Kou's semi-closed form pricing solutions. The other table information remains consistent with Table 9 of the main paper.

Maturity	Error Type	DOTM	OTM	ATM	ITM	DITM
$t = 0.25$	PrPCTE	0.565540	0.227200	0.047374	0.020492	0.011626
	PrRMSE	0.000309	0.001625	0.004176	0.003275	0.004250
	IVPCTE	0.053758	0.036993	0.046664	0.078632	0.195837
	IVRMSE	0.026923	0.019267	0.021644	0.024183	0.076983
$t = 0.5$	PrPCTE	0.183634	0.054303	0.032206	0.013901	0.008919
	PrRMSE	0.000357	0.004457	0.002425	0.002385	0.003170
	IVPCTE	0.024350	0.019254	0.030406	0.037600	0.072384
	IVRMSE	0.010135	0.020723	0.008940	0.010598	0.023723
$t = 0.75$	PrPCTE	0.152235	0.088177	0.022879	0.011666	0.007300
	PrRMSE	0.000435	0.001322	0.002684	0.002234	0.002597
	IVPCTE	0.018415	0.018745	0.023755	0.026259	0.042757
	IVRMSE	0.006708	0.007255	0.007833	0.007467	0.013185
$t = 1.0$	PrPCTE	0.128098	0.028252	0.023524	0.011519	0.006743
	PrRMSE	0.001195	0.001718	0.003825	0.003100	0.002656
	IVPCTE	0.020543	0.012923	0.025150	0.023839	0.030379
	IVRMSE	0.008059	0.005255	0.009625	0.009151	0.009602

While making direct comparisons between two different network structures is challenging, we opted to use a MLP network with 3 hidden layers and 300 nodes per layer. This choice aimed to minimise the computational time difference with the DGM NN used in Sections 3-4[1].

Tables 1 and 2 present the pricing errors of two models: Kou's double exponential jump diffusion and stochastic volatility jump diffusion (SVJ). The results indicate that the MLP network performs slightly better on Kou's model, while the DGM NN excels on SVJ. This intriguing discovery challenges the assumption that a more complex neural network structure always yields superior performance. One possible explanation for this phenomenon is that the DGM NN was specifically designed to tackle high-dimensional problems, thereby achieving better results on SVJ (a 12-dimensional parametric problem) than Kou's model (an 8-dimensional parametric problem).

However, determining the best NN for solving parametric PDEs extends beyond the scope of this paper.

References

Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning* (MIT press).

Sirignano J, Spiliopoulos K (2018) DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics* 375:1339–1364.

[1] During our testing, we found that GPUs such as the NVIDIA RTX 2070 and NVIDIA Tesla T4 yielded identical computational times for both networks. However, on more powerful server GPUs like the NVIDIA RTX A6000, the MLP network demonstrated a computational time approximately 30% faster than that of the DGM NN.

Table 2 Under the MLP network, the pricing errors of the NN approximated density of stochastic volatility jump diffusion model, compared with SVJ's semi-closed form pricing solutions. The other table information remains consistent with Table 11 of the main paper.

Maturity	Error Type	DOTM	OTM	ATM	ITM	DITM
$t = 0.25$	PrPCTE	0.355213	0.053095	0.020794	0.012279	0.008007
	PrRMSE	0.002246	0.002587	0.002647	0.002708	0.003373
	IVPCTE	0.056589	0.027055	0.020781	0.021206	0.045405
	IVRMSE	0.017434	0.007637	0.006148	0.006497	0.020049
$t = 0.5$	PrPCTE	0.066232	0.018693	0.010047	0.007895	0.006452
	PrRMSE	0.001825	0.001859	0.001935	0.002292	0.003204
	IVPCTE	0.020812	0.012015	0.010197	0.011691	0.017303
	IVRMSE	0.008963	0.005024	0.004530	0.005312	0.008535
$t = 0.75$	PrPCTE	0.029266	0.010795	0.007516	0.006662	0.006045
	PrRMSE	0.001542	0.001628	0.001888	0.002377	0.003339
	IVPCTE	0.011648	0.007788	0.007708	0.009246	0.013340
	IVRMSE	0.006291	0.004281	0.004482	0.005495	0.008182
$t = 1.0$	PrPCTE	0.017908	0.008097	0.006694	0.006325	0.006134
	PrRMSE	0.001418	0.001638	0.002033	0.002582	0.003563
	IVPCTE	0.008275	0.006303	0.006951	0.008483	0.012112
	IVRMSE	0.005223	0.004297	0.004902	0.006005	0.008426