Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Deeper network methods for solving 3D American Heston-CIR model

Chinonso Nwankwo

chinonso.nwankwo@ucalgary.ca

University of Calgary https://orcid.org/0000-0001-5526-1337

Weizhong Dai

Louisiana Tech University

Research Article

Keywords: American Heston-CIR model, SoftNet, LinNet, AddNet, Multiplicative ensembling, Softmax block, Early exercise boundary, Value function, Greeks

Posted Date: October 22nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11446743/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Deeper network methods for solving 3D American Heston-CIR model

Chinonso Nwankwo *[1] and Weizhong Dai[2]

[1]Department of Mathematics and Statistics, University of Calgary, Calgary T2N 1N4, Canada
[2]Mathematics and Statistics, Louisiana Tech University, Ruston, LA 71272, USA

Abstract

Deeper network methods often encounter challenges such as overfitting, vanishing or exploding gradient effect, and performance degradation. Existing network models proposed to tackle these challenges have focused mostly on classification tasks. In this research, we propose three deeper network methods for solving 3D American Heston-CIR model. These three network methods denoted as SoftNet, LinNet, and AddNet, respectively are designed via multiplicative ensembling of sub-network architectures. Moreover, SoftNet, LinNet, and AddNet involve feature reuse, aggregation of linear units, and early exits, respectively. Importantly, the first sub-network architecture within SoftNet, LinNet, and AddNet are designed with deeper structure via softmax blocks. The remaining sub-network architectures are supportive networks with shallow configurations. Compared with existing data, the predictions from SoftNet, LinNet, and AddNet are reasonable. Compared with conventional network models with 13 hidden layers which give poor performance, the predictive performances of SoftNet, LinNet, and AddNet are reasonable beyond 13 hidden layers. We envisage that the trio circumvent the vanishing gradient effect to a reasonable extent. In particular, LinNet presents the most efficiency and we use up to 100 hidden layers within the LinNet.

Keywords: American Heston-CIR model, SoftNet, LinNet, AddNet, Multiplicative ensembling, Softmax block, Early exercise boundary, Value function, Greeks

1 Introduction

Neural network models involve many hidden layers (with interconnected nodes) between the input and output layers which are sequentially stacked in compositional form to perform range of tasks with the hope of more complex feature extraction during inference. Some of the common tasks include regression and classification problems. Conventional neural network methods have the tendency of presenting good predictive performance in general. When very high-dimensional or strongly nonlinear model is considered, there could be a need to design a more efficient neural network algorithm that will capture required features and extract necessary information during inference. It is expected that deep neural network will present better predictive performance than shallow networks in more complex task. However, deep learning methods still suffer challenges such as overfitting, vanishing or exploding gradient effect, and performance degradation [3, 7, 8, 10]. Several remedies to these challenges has been addressed in existing literature which include dropout [25], batch normalization [13], residual connection [8], feature reuse via concatenation [11], and layer aggregation [29]. It is interesting to see in existing literature how some of these remedies have led to the designed of network architecture with a very deep structure mostly in convolutional neural network (CNN) for image classification [8, 11].

It is important to note that DenseNet [11] and ResNet [8] are two notable CNN architectures that can train with very deep structures involving thousands of layers. Moreover, DenseNet architecture implemented in convolution neural network (CNN) by Huang et al. [11] has its main application in image processing where the complexity of the input feature is very high and involves redundant information [12]. The convolution kernel enhances performance by reducing repetition among variable and

*Corresponding author: chinonso.nwankwo@ucalgary.ca

extracting important information [12]. However, as mentioned in the Jiang et. al [12], CNN may lose essential information from feature vector due to the local convolution kernels and thus may not be suitable for non-linear regression. It should be pointed out that the first sub-network architecture within SoftNet is closely related to the work of Jiang et al. [12] where the authors reformulated DenseNet [11] architecture and designed a neural network model for solving a non-linear regression problem. The authors recorded the best performance on depth with 19 hidden layers and recommended depth of 13-19 hidden layers.

The problem that we focus for this research is a three-dimensional American Heston-CIR model and we solve it as a free boundary problem. Here, the free boundary known as the early exercise boundary is simultaneously approximated or predicted (based on the solution method) with the value function and Greeks. The Greeks are the derivatives of the value function. Our objective is to solve three-dimensional American Heston-CIR model using a network architecture that trains with a deeper structure. Based on the comments made by Jiang et al. [12] and Chen et al. [5] in their literature and our best knowledge, the direct implementation of CNN based on DenseNet [11] or ResNet [8] architectures might not be suitable. We hope to show in the experiment section that the predictive performance of the conventional deep learning framework with softmax units is very poor beyond thirteen hidden layers. We envisage that this poor predictive performance could be as a result of vanishing gradient effect. The implication is that the conventional DNN architecture with softmax units might not be suitable for a more complex task that requires deeper network models beyond thirteen hidden layers. To circumvent this poor predictive performance, we design a suite of multiplicative ensembled network models which are formulated via feature reuse, aggregation of linear units, and early exits, respectively. We hope that the design of SoftNet, LinNet, and AddNet, respectively. Hence, the design of SoftNet, LinNet, and AddNet might have mitigate vanishing gradient effect which allowed us to train with a deeper structure within the first sub-network architecture.

The rest of the manuscript is organized as follows: In Section 2, we describe the three-dimensional American Heston-CIR model which we consider to be solved. In Section 3, we present the design of SoftNet and its variant M-SoftNet. The design of the network model AddNet is described in Section 4. In Section 5, we present of the network model LinNet. We describe sampling, grid refinement, and optimization strategies in Section 6. Experiments are conducted in Section 7 and we conclude in Section 8.

2 Three-Factor American Heston-CIR model

In this study, we consider a three-factor American Heston-CIR model as follows:

$$\begin{aligned}\frac{\partial V(t,S,\upsilon,r)}{\partial t}+\frac{\upsilon S^2}{2}\frac{\partial^2 V(t,S,\upsilon,r)}{\partial S^2}+\frac{\upsilon\sigma_1^2}{2}\frac{\partial^2 V(t,S,\upsilon,r)}{\partial \upsilon^2}+\frac{r\sigma_2^2}{2}\frac{\partial^2 V(t,S,\upsilon,r)}{\partial r^2}\\ +\rho_{1,2}\sigma_1\upsilon S\frac{\partial^2 V(t,S,\upsilon,r)}{\partial S\partial\upsilon}+\rho_{1,3}\sigma_2\sqrt{\upsilon}\sqrt{r}S\frac{\partial^2 V(t,S,\upsilon,r)}{\partial S\partial r}+\rho_{2,3}\sigma_1\sigma_2\sqrt{\upsilon}\sqrt{r}\frac{\partial^2 V(t,S,\upsilon,r)}{\partial\upsilon\partial r}\\ +(r-\delta)S\frac{\partial V(t,S,\upsilon,r)}{\partial S}+\kappa(\theta-\upsilon)\frac{\partial V(t,S,\upsilon,r)}{\partial\upsilon}+b(a-r)\frac{\partial V(t,S,\upsilon,r)}{\partial r}\\ =rV(t,S,\upsilon,r),\qquad S>\hat{s}_f(t,\upsilon,r),\ t\in[0,T),\ \upsilon,r>0.\end{aligned}\tag{2.1}$$

We denote the value function and the early exercise boundary as $V(t,S,\upsilon,r)$ and $\hat{s}_f(t,\upsilon,r)$, respectively. The parameters υ_t and r_t are the variance and interest rate. The variable S is the underlying asset price. The parameters κ and b represent the speeds of mean reversion for both the variance and interest rate, respectively. The parameter θ and a are the long time means for the variance and interest rate, respectively.

To solve the above three-factor American Heston-CIR model, we consider a fixed-free boundary problem and fix the early exercise boundary using the logarithm Landau transformation. For efficient

implementation of our deep learning solution framework, we further normalize the value function and time domain. Below, we give the descriptions of the considered transformation and normalization:

$$x := \ln\left(\frac{S}{\hat{s}_f(t,\upsilon,r)}\right), \qquad \tau := \frac{T-t}{T}, \qquad U(\tau,x,\upsilon,r) := \frac{1}{K}V(t,S,\upsilon,r); \tag{2.2}$$

$$s_f(\tau,\upsilon,r) := \frac{\hat{s}_f(t,\upsilon,r)}{K}, \qquad \chi(\tau,\upsilon,r) := \ln\frac{1}{s_f(t,\upsilon,r)}. \tag{2.3}$$

The transformed fixed-free boundary model is represented below:

$$\frac{1}{T}\frac{\partial U}{\partial \tau} + rU = \beta_1\frac{\partial^2 U}{\partial x^2} + \beta_2\frac{\partial^2 U}{\partial \upsilon^2} + \beta_3\frac{\partial^2 U}{\partial r^2} + \beta_4\frac{\partial U}{\partial x\partial \upsilon} + \beta_5\frac{\partial U}{\partial x\partial r} + \beta_6\frac{\partial U}{\partial \upsilon\partial r} + \beta_7\frac{\partial U}{\partial x} + \beta_8\frac{\partial U}{\partial r} + \beta_9\frac{\partial U}{\partial \upsilon}, \tag{2.4}$$

$$\tau \in (0,T],\ x>0,\ \upsilon>0,\ r>0, \tag{2.5}$$

where, for $i = 1,\ldots,9$, $\beta_i = \beta_i(\tau,\upsilon,r)$ is defined as follows:

$$\beta_1(\tau,\upsilon,r) = \rho_{1,2}\chi_\upsilon\sigma_1\upsilon + \rho_{1,3}\chi_r\sigma_2\sqrt{\upsilon r} + \frac{\upsilon}{2} + \frac{\upsilon}{2}\chi_\upsilon^2\sigma_1^2 + \frac{\chi_r^2\sigma_2^2 r}{2} + \chi_\upsilon\chi_r\rho_{2,3}\sigma_1\sigma_2\sqrt{\upsilon r}; \tag{2.6}$$

$$\beta_2(\tau,\upsilon,r) = -\frac{\sigma_2^2\upsilon}{2}, \quad \beta_3(\tau,\upsilon,r) = -\frac{\sigma_1^2 r}{2}, \quad \beta_4(\tau,\upsilon,r) = \chi_r\rho_{2,3}\sigma_1\sigma_2\sqrt{\upsilon r} + \rho_{1,2}\sigma_1\upsilon + \chi_\upsilon\sigma_1^2\upsilon; \tag{2.7}$$

$$\beta_5(\tau,\upsilon,r) = \chi_\upsilon\rho_{2,3}\sigma_1\sigma_2\sqrt{\upsilon r} + \rho_{1,3}\sigma_2\sqrt{\upsilon r} + \chi_r\sigma_2^2 r, \quad \beta_6(\tau,\upsilon,r) = \rho_{2,3}\sigma_1\sigma_2\sqrt{\upsilon r}; \tag{2.8}$$

$$\beta_7(\tau,\upsilon,r) = -\frac{\upsilon}{2} + \frac{\sigma_1^2\upsilon}{2}\chi_{\upsilon\upsilon} + \frac{\sigma_2^2}{2}r\chi_{rr} - \rho_{2,3}\sigma_1\sigma_2\sqrt{\upsilon r}\chi_{\upsilon r} - \frac{\chi_\tau}{T} + r - \delta + \chi_\upsilon\kappa(\theta-\upsilon) + \chi_r b(a-r); \tag{2.9}$$

$$\beta_8(\tau,\upsilon,r) = b(a-r), \qquad \beta_9(\tau,\upsilon,r) = \kappa(\theta-\upsilon). \tag{2.10}$$

The subscripts on χ denote partial derivatives with respect to the corresponding variables. In addition, we have

$$U(\tau,0,\upsilon,r) = 1 - e^{-\chi(\tau,\upsilon,r)}, \quad \frac{\partial U}{\partial x}(\tau,0,\upsilon,r) = -e^{-\chi(\tau,\upsilon,r)}, \qquad \tau>0,\ \upsilon>0,\ r>0, \tag{2.11}$$

together with

$$U(0,x,\upsilon,r) = \max(1-e^{x-\chi(\tau,\upsilon,r)},0) = 0, \qquad s_f(0,\upsilon,r) = 1. \tag{2.12}$$

3 Design of SoftNet and M-SoftNet

In this section, we present SoftNet and M-SoftNet, two network models designed via multiplicative ensembling of two sub-network models and feature reuse and trained with deeper hidden layers. The first sub-network model is designed with a deeper hidden layer while the second sub-network model is strictly designed with two hidden layers. The outputs of these two sub-network models are then ensembled multiplicatively from which SoftNet or M-SoftNet is formulated. To design the first sub-network model, we borrow a reasonable insight from DenseNet [11]. However, unlike DenseNet [11] that established connections between layers within a block, the first sub-architecture involves concatenated shortcut connections (feature reuse and couplings) between softmax blocks instead of layers. In the work of Hess [9], the author asked an important question: To what extent is it necessary to connect all the previous layers in order to reap the benefit of feature reuse? Here, we answer the question to some extent by considering feature reuse and coupling after each softmax block. Softmax block is a n hidden layer sub-architecture with softmax units (softmax activation function). If $n = 1$, shortcut connection will be between layers.

One can view the first sub-architecture for SoftNet as a train with rail vehicles and couplers. The first sub-architecture involves softmax blocks and the input of each block (from the third block) involves concatenation of the output of the previous softmax blocks. Softmax blocks represent rail vehicles and coupling units of these vehicles are similar to the concatenating shortcut units where the initial feature vector and the features obtained from softmax blocks are coupled. Feature reuse is achieved between two softmax blocks and not within blocks. Moreover each individual softmax block might have the tendency to present good predictive performance. However, by coupling these softmax blocks from the lens of feature reuse, we intend to design an efficient and advanced DNN architecture that has the capacity to train with deeper hidden layers. We note that M-SoftNet is a modularized SoftNet and the configuration of its first sub-architecture differs from SoftNet.

3.1 Mixture of deep learning solution

SoftNet consists of sub-architectures which are ensembled multiplicatively. The hidden layers in the first sub-architecture (main architecture) is deeper and varies across versions of SoftNet. The second sub-architecture strictly involves two hidden layers and ensures that some of the terminal conditions are satisfied when both sub-architectures are ensembled. In the following paragraphs, we present the description of the DNN solution frameworks within SoftNet

Feature vector: Since the pricing model for performance validation is 3D American Heston-CIR model as described in Section 2, the considered models is in three-dimensional space with time and space of model parameter as

$$\vartheta = (\tau, x, \upsilon, r), \tag{3.1}$$

where the domain for the parameter is

$$\vartheta \in ((0,1], [0, x_{\max}], [0, \upsilon_{\max}], [0, r_{\max}]). \tag{3.2}$$

The design of SoftNet involves two sub-architectures with two different input features. For first sub-architecture, we consider

$$\mathbf{z}_1 = (\tau, x, \upsilon, r). \tag{3.3}$$

For the second sub-architecture, we consider

$$\mathbf{z}_2 = \tau. \tag{3.4}$$

DNN solution framework: The solution framework for SoftNet involving mixture of deep learning solution is expressed as follows:

$$\mathcal{M}(\mathbf{y}; \nu_M) = \sum_{i=1}^{n} g_{i,o}(\mathbf{y}; \nu_{i,o}) e^{-\bar{q}_i(\mathbf{y};\gamma)}, \qquad \mathbf{y} = \mathbf{z}_1 = (\tau, x, \upsilon, r), \qquad n = 18; \tag{3.5}$$

under the condition that

$$\bar{s}_f(\tau, \upsilon, r; \nu_M) = 1 - \mathcal{M}(\tau, 0, \upsilon, r; \nu_M). \tag{3.6}$$

Here, we define

$$\mathbf{g}_o(\mathbf{y}; \nu_o) = [g_{1,o}(\mathbf{y}; \nu_{1,o}), g_{2,o}(\mathbf{y}; \nu_{2,o}), \cdots g_{n,o}(\mathbf{y}; \nu_{n,o})]. \tag{3.7}$$

Moreover, we enable optionality in the choice of $q_i(\mathbf{x})$ in the form of mixture of functions such that

$$\bar{q}_1(\mathbf{y};\gamma) = \gamma_{opt} \frac{x^2(\upsilon_{\max} - \upsilon)^2}{\tau + \epsilon_\tau}, \qquad \bar{q}_1(\mathbf{y};\gamma) = \gamma_{opt} \frac{x^2(\upsilon_{\max} - \upsilon)^2}{\upsilon\tau + \epsilon_\tau}; \tag{3.8}$$

$$\bar{q}_3(\mathbf{y};\gamma) = \frac{x^2(\gamma_{opt}\upsilon_{\max} - \upsilon)^2}{\tau + \epsilon_\tau}, \qquad \bar{q}_4(\mathbf{y};\gamma) = \frac{x^2(\gamma_{opt}\upsilon_{\max} - \upsilon)^2}{\upsilon\tau + \epsilon_\tau}; \tag{3.9}$$

$$\bar{q}_5(\mathbf{y};\gamma) = \gamma_{opt} \frac{x^2(r + \upsilon_{\max} - \upsilon)^2}{\tau + \epsilon_\tau}, \qquad \bar{q}_6(\mathbf{y};\gamma) = \gamma_{opt} \frac{x^2(r + \upsilon_{\max} - \upsilon)^2}{\upsilon\tau + \epsilon_\tau}; \tag{3.10}$$

$$\bar{q}_7(\mathbf{y};\gamma) = \gamma_{opt}\frac{x^2(\upsilon_{\max}-\upsilon)}{\tau+\epsilon_\tau}, \qquad \bar{q}_8(\mathbf{y};\gamma) = \gamma_{opt}\frac{x^2(\upsilon_{\max}-\upsilon)}{\upsilon\tau+\epsilon_\tau}; \tag{3.11}$$

$$\bar{q}_9(\mathbf{y};\gamma) = \gamma_{opt}\frac{x^2}{\tau+\epsilon_\tau}, \qquad \bar{q}_{10}(\mathbf{y};\gamma) = \gamma_{opt}\frac{x^2}{\upsilon\tau+\epsilon_\tau}; \tag{3.12}$$

$$\bar{q}_{11}(\mathbf{y};\gamma) = \gamma_{opt}\frac{x^2r^2}{\tau+\epsilon_\tau}, \qquad \bar{q}_{12}(\mathbf{y};\gamma) = \gamma_{opt}\frac{x^2r^2}{\upsilon\tau+\epsilon_\tau}; \tag{3.13}$$

$$\bar{q}_{13}(\mathbf{y};\gamma) = \frac{x^2(\gamma_{opt}\upsilon_{\max}-\upsilon)}{\tau+\epsilon_\tau}, \qquad \bar{q}_{14}(\mathbf{y};\gamma) = \frac{x^2(\gamma_{opt}\upsilon_{\max}-\upsilon)}{\upsilon\tau+\epsilon_\tau}; \tag{3.14}$$

$$\bar{q}_{15}(\mathbf{y};\gamma) = \gamma_{opt}\frac{(x+r)^2}{\tau+\epsilon_\tau}, \qquad \bar{q}_{16}(\mathbf{y};\gamma) = \gamma_{opt}\frac{(x+r)^2}{\upsilon\tau+\epsilon_\tau}; \tag{3.15}$$

$$\bar{q}_{17}(\mathbf{y};\gamma) = \gamma_{opt}\frac{(x+r)^2(\upsilon_{\max}-\upsilon)^2}{\tau+\epsilon_\tau}, \qquad \bar{q}_{18}(\mathbf{y};\gamma) = \gamma_{opt}\frac{(x^2+r)^2(\upsilon_{\max}-\upsilon)^2}{\upsilon\tau+\epsilon_\tau}; \tag{3.16}$$

$$\epsilon_\tau \in [10^{-13},\ 10^{-4}]. \tag{3.17}$$

We refer the reader to our previous work [18, 19] on the discussions regarding $\bar{q}_i(\mathbf{y};\gamma)$.

Multiplicative ensembled layer: The multiplicative ensembled layer is involved in all the network architectures presented in this study (with or without feature reuse). In this layer, the output of the first sub-architecture $\mathcal{R}(\mathbf{y};\nu_R)$ and second sub-architecture $\mathcal{T}(\tau;\nu_\mathcal{T})$ are multiplied as follows:

$$\mathbf{s}(\mathbf{y};\nu_s) = \mathcal{T}(\tau;\nu_\mathcal{T})\mathcal{R}(\mathbf{y};\nu_R), \qquad \mathcal{R}(\mathbf{y};\nu_R) \in \mathbb{R}^{1\times n}, \qquad \mathcal{T}(\tau;\nu_\mathcal{T}) \in \mathbb{R}. \tag{3.18}$$

We note that all the hidden and output layers are formulated without bias term. Moreover, the second sub-architecture is designed with adjusted softmax activation function to ensure $\mathcal{T}(0;\nu_\mathcal{T}) = 0$. Moreover, the main purpose of this layer is to ensure that some of the terminal conditions are satisfied in exact form (after removing the bias term from the hidden and output layers). Here, we express the layer's output as follows:

$$g_{i,o} = \sum_{j=1}^{m_f}\left(\mathcal{A}_1(\mathbf{s}(\mathbf{y};\nu_s))_j - \frac{1}{m_f}\right)w^o_{j,i}, \tag{3.19}$$

$$\mathcal{A}_1(\mathbf{s}) = \texttt{softmax}(\mathbf{s}) = \left[\frac{e^{s_1}}{\sum_{i=1}^{m_f}e^{s_i}}, \cdots, \frac{e^{s_{m_f}}}{\sum_{i=1}^{m_f}e^{s_i}}\right], \qquad \mathcal{A}_1(\mathbf{s})_j = \texttt{softmax}(\mathbf{s})_j = \frac{e^{s_j}}{\sum_{i=1}^{m_f}e^{s_i}} \tag{3.20}$$

$$j = 1, 2, \cdots, m_f. \tag{3.21}$$

Output of the first sub-architecture: Here, we describe two network outputs for the first sub-architecture. The first description involves feature reuse and coupling (SoftNet). The second description involves no feature reuse and coupling (standard DNN architecture). For the network involving feature reuse and coupling (SoftNet), the output is as follows:

$$\mathcal{R}(\mathbf{y};\nu_\mathcal{R}) = \mathcal{A}_1(\mathbf{g}_2\left(\mathbf{z}_1^n;\nu_{2,n}\right)), \qquad \mathcal{A}_1(\mathbf{g}_2)_j = \frac{e^{(g_2)_j}}{\sum_{i=1}^{m_{2,n}}e^{(g_2)_i}}. \tag{3.22}$$

For the conventional DNN architecture without feature reuse and coupling, the neural network output is expressed as follows:

$$\mathcal{R}(\mathbf{y};\nu_\mathcal{R}) = \mathcal{A}_1(\mathbf{g}_{\hat{n}}\left(\mathbf{y};\nu_{\hat{n}}\right)), \qquad \mathcal{A}_1(\mathbf{g}_{\hat{n}})_j = \frac{e^{(g_{\hat{n}})_j}}{\sum_{i=1}^{m_{\hat{n}}}e^{(g_{\hat{n}})_i}}. \tag{3.23}$$

The descriptions of $\mathbf{g}_2\left(\mathbf{z}_1^n;\nu_{2,n}\right)$ and $\mathbf{g}_{\hat{n}}\left(\mathbf{y};\nu_{\hat{n}}\right)$ are given in the coming subsections below when presenting the design for network sub-architecture with or without feature reuse, respectively.

Output of the second sub-architecture: The second sub-architecture is strictly designed with two hidden layers. The output of the second sub-architecture is expressed as follows:

$$\mathcal{T}(\tau;\nu_{\mathcal{T}}) = \sum_{i=1}^{\hat{m}_f} \left(\mathcal{A}_1(\mathbf{h}_f(\tau;\hat{\nu}_f))_j - \frac{1}{\hat{m}_f} \right) \hat{w}^o_{j,i}. \tag{3.24}$$

The design is to ensure that some of the terminal conditions are satisfied in exact form. To further illustrate this, when $\tau = 0$, we formulate

$$\mathbf{h}_f(0;\hat{\nu}_f) = \mathbf{0}. \tag{3.25}$$

The detailed description is given in Subsection 3.2. The implication is that

$$\mathcal{A}_1(\mathbf{h}_f(\tau;\hat{\nu}_f)) - \frac{1}{\hat{m}_f} = \mathbf{0}, \qquad \mathcal{T}(0;\nu_{\mathcal{T}}) = 0; \tag{3.26}$$

$$\mathbf{s}(0,x,\upsilon,r;\nu_s) = 0, \qquad g_{i,o}(0,x,\upsilon,r;\nu_{1,o}) = 0, \qquad \mathbf{g}_o(0,x,\upsilon,r;\nu_o) = \mathbf{0}; \tag{3.27}$$

$$\mathcal{M}(0,x,\upsilon,r;\nu_M) = 0, \qquad \bar{s}_f(0,\upsilon,r;\nu_M) = 1. \tag{3.28}$$

Note that in exact form as presented in (2.12),

$$s_f(0,\upsilon,r) = 1, \qquad U(0,x,\upsilon,r) = 0. \tag{3.29}$$

Hence,

$$\mathcal{M}(0,x,\upsilon,r;\nu_M) = U(0,x,\upsilon,r), \qquad \bar{s}_f(0,\upsilon,r;\nu_M) = s_f(0,\upsilon,r). \tag{3.30}$$

This is the main reason for the multiplicative ensembling layer. The multiplication of the second sub-architectures $\mathcal{T}(\tau;\nu_{\mathcal{T}})$ to the first sub-architecture $\mathcal{R}(\mathbf{y};\nu_{\mathcal{R}})$ ensures exact imposition of the terminal condition on SoftNet.

3.2 Design of the second sub-architecture

Here, we further elaborate on $\mathbf{h}(\tau;\hat{\nu}_f)$ which we used in the design of the network output for the second sub-architecture $\mathcal{T}(\tau;\nu_{\mathcal{T}})$. Here, $f = 2$ because of the considered two hidden layers. The description of the architecture based on $\mathbf{h}(\tau;\hat{\nu}_f)$ is given below.

Second hidden layer: The second hidden layer is expressed as follows:

$$\mathcal{A}_2(\mathbf{h}_f) = \mathcal{A}_1(\mathbf{h}_f) - \frac{1}{\hat{m}_f}, \qquad \mathbf{h}_f(\tau;\hat{\nu}_f) = [h_{1,f}(\tau;\hat{\nu}_{1,f}), h_{2,f}(\tau;\hat{\nu}_{2,f}), \cdots, h_{\hat{m}_f,f}(\tau;\hat{\nu}_{\hat{m}_f,f})], \tag{3.31}$$

with

$$h_{j,f} = \sum_{k=1}^{\hat{m}_{f-1}} \mathcal{A}_2(\mathbf{h}_{f-1}(\tau;\hat{\nu}_{f-1}))_k \hat{w}^f_{k,j}. \tag{3.32}$$

First hidden layer: The first hidden layer is described below

$$\mathcal{A}_2(\mathbf{h}_{f-1}) = \mathcal{A}_1(\mathbf{h}_{f-1}) - \frac{1}{\hat{m}_{f-1}}, \qquad \mathbf{h}_{f-1} = [h_{1,f-1}, h_{2,f-1}, \cdots, h_{\hat{m}_{f-1},f-1}], \tag{3.33}$$

with

$$h_{j,f-1} = \tau \hat{w}^1_j. \tag{3.34}$$

The description of the second sub-architecture is given in Figure 1.

Remark. *The designed second sub-architecture in this section is the same in all the considered DNN architectures across this research. In the following subsection, we focus on the design of the first sub-architecture and the multiplicative ensembling of these two sub-architectures.*

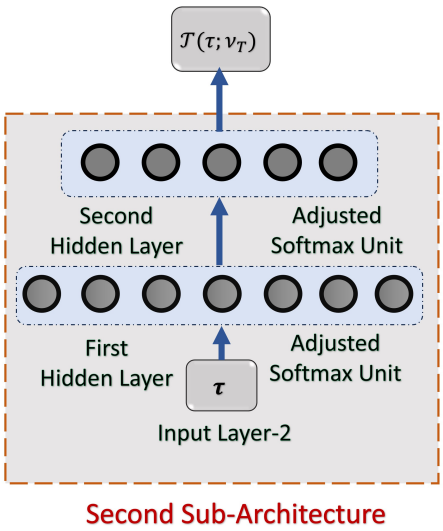


Figure 1: DNN for the second sub-architecture. The second sub-architecture involves units that decreases across hidden layers

3.3 Design of DNN architecture without feature reuse

Here, we present and describe the design of the first sub-architecture without feature reuse and coupling and the multiplicative ensembling of these two sub-architectures. The feature vector for the first sub-architecture is as follows:

$$\mathbf{z}_1 = \mathbf{y} = (\tau, x, \upsilon, r). \tag{3.35}$$

The network output for the hidden layers is described below.

First hidden layer: Let $\hat{k}$ represent the $\hat{k}^{th}$ hidden layer and $\hat{n}$ represent the number of hidden layers before the multiplicative ensembled layer. For the first hidden layer with $\hat{k} = 1$, the network is expressed as follows:

$$\mathbf{g}_1(\mathbf{y};\nu_1) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_1,1}]; \tag{3.36}$$

$$g_{j,1} = -0.5\sum_{k=1}^{m_0} y_k (w_{k,j}^1)^2, \qquad \mathcal{A}_1(\mathbf{g}_1(\mathbf{y};\nu_1)) = \texttt{softmax}(\mathbf{g}_1(\mathbf{y};\nu_1)); \tag{3.37}$$

where m_0 is the number of units.

Remaining hidden layers before multiplicative ensembled layer: For $\hat{k} = 2,\ 3,\ \cdots \hat{n}$, the description of the hidden layers is as follows:

$$\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}}) = [g_{1,\hat{k}},\ g_{2,\hat{k}},\ \cdots,\ g_{m_{\hat{k}},\hat{k}}], \qquad \hat{k} = 2,\ 3,\ \cdots, \hat{n}; \tag{3.38}$$

$$g_{j,\hat{k}} = -0.5\sum_{k=1}^{m_{\hat{k}-1}} \mathcal{A}_1(\mathbf{g}_{\hat{k}-1})_k (w_{k,j}^{\hat{k}})^2, \qquad \mathcal{A}_1(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})) = \texttt{softmax}(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})), \quad \hat{k} = 2,\ 3,\ \cdots, \hat{n}, \tag{3.39}$$

where $m_{\hat{k}-1}$ is the number of units. Moreover,

$$\mathcal{R}(\mathbf{y};\nu_R) = \left[\sum_{j=1}^{m_{\hat{n}}} \mathcal{A}_1(\mathbf{g}_{\hat{n}}(\mathbf{y};\nu_{\hat{n}}))_j w_{1,j}, \sum_{j=1}^{m_{\hat{n}}} \mathcal{A}_1(\mathbf{g}_{\hat{n}}(\mathbf{y};\nu_{\hat{n}}))_j w_{2,j},\ \cdots \sum_{j=1}^{m_{\hat{n}}} \mathcal{A}_1(\mathbf{g}_{\hat{n}}(\mathbf{y};\nu_{\hat{n}}))_j w_{m_e,j}\right], \tag{3.40}$$

for the conventional DNN architecture.

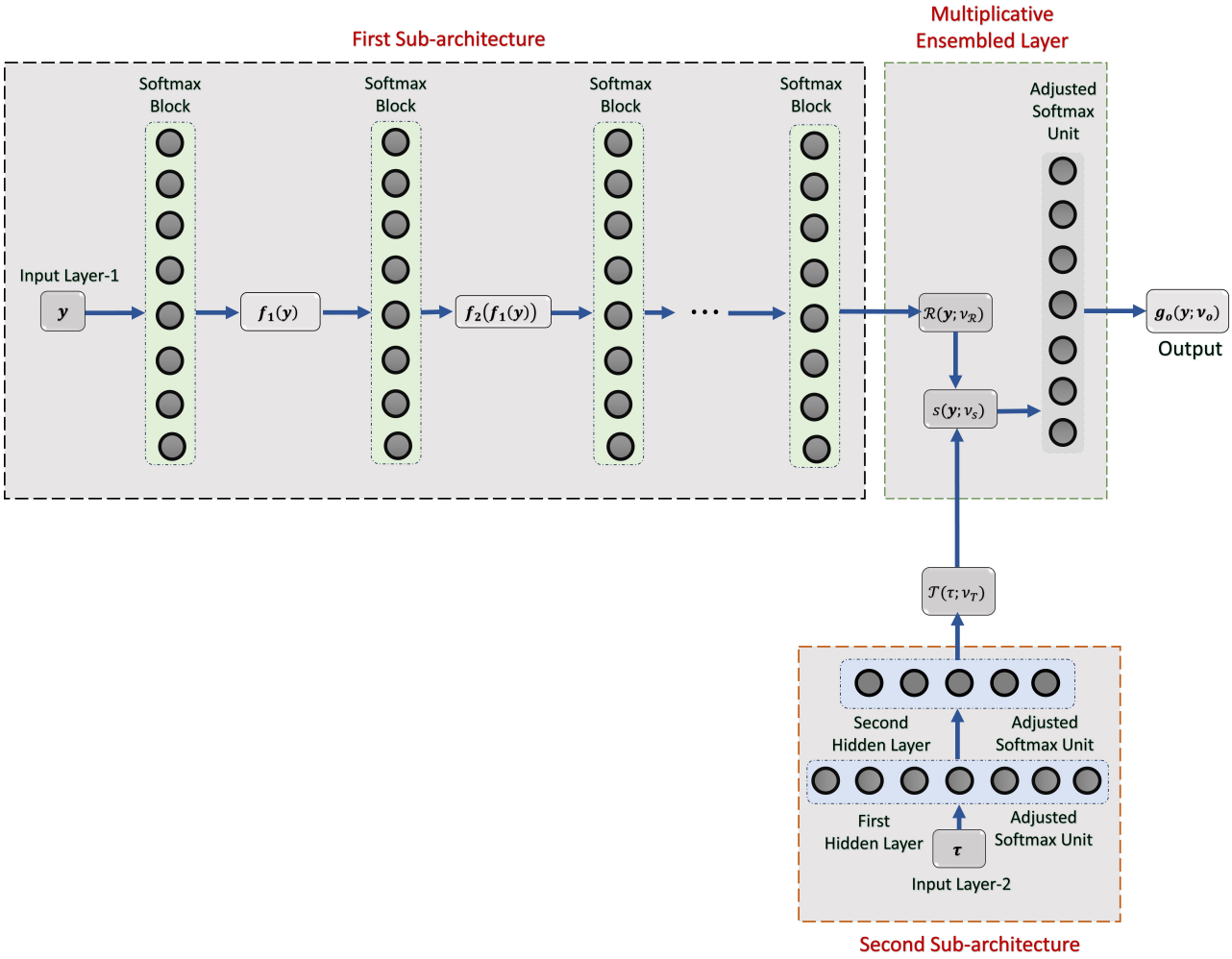


Figure 2: DNN architecture without feature reuse and coupling. The first sub-architecture could involve units that decrease or increase across softmax blocks

3.4 Design of the first sub-architecture within SoftNet

We describe the formulation of feature coupling and softmax blocks as follows:

$$\mathbf{z}_1^1 = \mathbf{y} = [\tau, x, \upsilon, r], \qquad \mathbf{g}_1(\mathbf{z}_1^1; \nu_{1,1}) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_{1,1},1}]; \tag{3.41}$$

$$g_{j,1} = -0.5 \sum_{k=1}^{m_{0,1}} (z_1^1)^k (w_{k,j}^{1,1})^2, \qquad \mathcal{A}_1(\mathbf{g}_1(\mathbf{z}_1^1; \nu_{1,1})) = \texttt{softmax}(\mathbf{g}_1(\mathbf{z}_1^1; \nu_{1,1})); \tag{3.42}$$

$$\mathbf{g}_i(\mathbf{z}_1^1; \nu_{i,1}) = [g_{1,i},\ g_{2,i},\ \cdots,\ g_{m_{i,1},i}]; \tag{3.43}$$

$$g_{j,i} = -0.5 \sum_{k=1}^{m_{i-1,1}} \mathcal{A}_1(\mathbf{g}_{i-1})_k (w_{k,j}^{i,1})^2, \qquad \mathcal{A}_1(\mathbf{g}_i(\mathbf{z}_1^1; \nu_{i,1})) = \texttt{softmax}(\mathbf{g}_i(\mathbf{z}_1^1; \nu_{i,1})), \qquad i = 2. \tag{3.44}$$

The description above represents the first feature vector and first softmax block. The second feature vector is the coupling (concatenating) of the first feature vector and the output of the first softmax block as given below:

$$\mathbf{z}_1^2 = \mathcal{A}_1(\mathbf{g}_2(\mathbf{z}_1^1; \nu_{2,1})). \tag{3.45}$$

With the new feature vector, we formulate another softmax block as given in (3.41)-(3.44). The process of coupling features and generating softmax block continue until we reach $n-1$ number of softmax blocks such that

$$\mathbf{z}_1^n = \texttt{concat}[\mathcal{A}_1(\mathbf{g}_2(\mathbf{z}_1^1; \nu_{2,1})), \cdots, \mathcal{A}_1(\mathbf{g}_2(\mathbf{z}_1^{n-1}; \nu_{2,n-1}))] = \texttt{concat}[\mathbf{z}_1^{n-1}, \mathcal{A}_1(\mathbf{g}_2(\mathbf{z}_1^{n-1}; \nu_{2,n-1}))], \tag{3.46}$$

$$\mathbf{g}_1(\mathbf{z}_1^n;\nu_{1,n}) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_{1,n},1}]; \tag{3.47}$$

$$g_{j,1} = -0.5\sum_{k=1}^{m_{0,n}}(z_1^n)^k(w_{k,j}^{1,n})^2, \quad \mathcal{A}_1(\mathbf{g}_1(\mathbf{z}_1^n;\nu_{1,n})) = \texttt{softmax}(\mathbf{g}_1(\mathbf{z}_1^n;\nu_{1,n})); \tag{3.48}$$

$$\mathbf{g}_i(\mathbf{z}_1^n;\nu_{i,n}) = [g_{1,i},\ g_{2,i},\ \cdots,\ g_{m_{i,n},i}]; \tag{3.49}$$

$$g_{j,i} = -0.5\sum_{k=1}^{m_{i-1,n}}\mathcal{A}_1(\mathbf{g}_{i-1})_k(w_{k,j}^{i,n})^2, \qquad \mathcal{A}_1(\mathbf{g}_i(\mathbf{z}_1^n;\nu_{i,n})) = \texttt{softmax}(\mathbf{g}_i(\mathbf{z}_1^n;\nu_{i,n})), \qquad i=2; \tag{3.50}$$

for which we implement the multiplicative ensembling layer and formulate the DNN solution framework as described in (3.5)-(3.19) and Figure 3. Here

$$\mathcal{R}(\mathbf{y};\nu_R) = \left[\sum_{j=1}^{m_{2,n}}\mathcal{A}_1(\mathbf{g}_2(\mathbf{z}_1^n;\nu_{2,n}))_j w_{1,j}, \sum_{j=1}^{m_{2,n}}\mathcal{A}_1(\mathbf{g}_2(\mathbf{z}_1^n;\nu_{2,n}))_j w_{2,j}, \cdots \sum_{j=1}^{m_{2,n}}\mathcal{A}_1(\mathbf{g}_2(\mathbf{z}_1^n;\nu_{2,n}))_j w_{m_e,j}\right]. \tag{3.51}$$

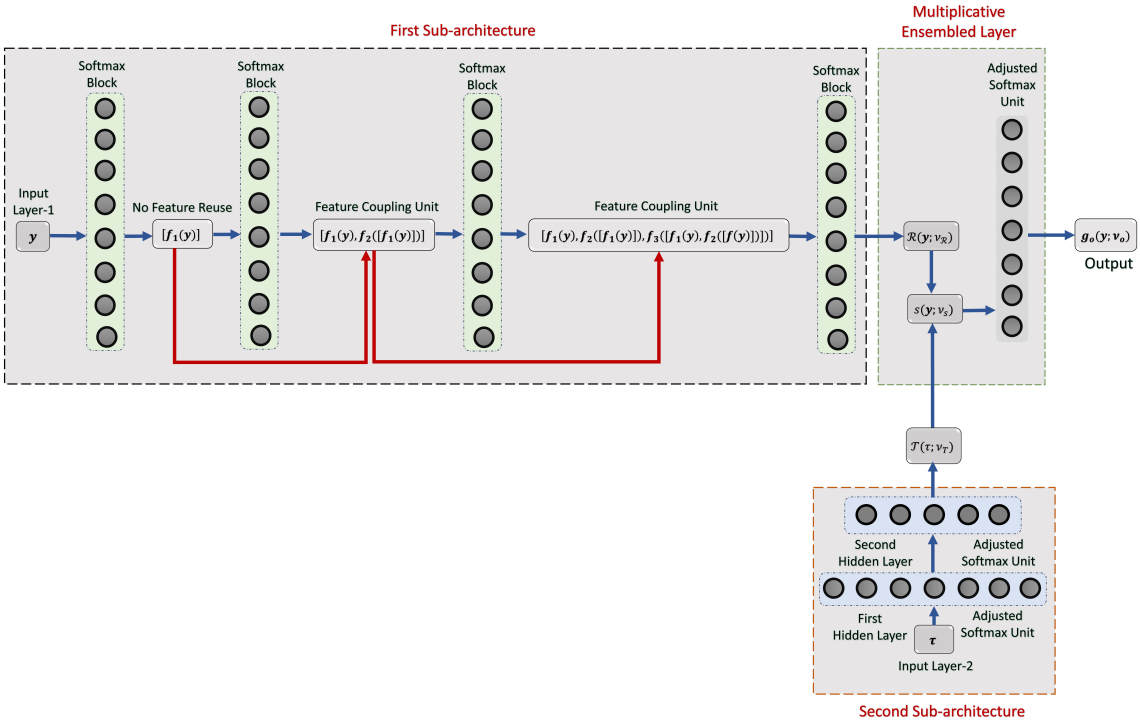


Figure 3: SoftNet architecture. For brevity, only four softmax blocks are considered in the diagram for the first sub-architecture.

3.5 Short note on the design of feature coupling unit via concatenation shortcut

It is important to observe that we avoid feature reuse after the first softmax block in the SoftNet architecture as described in Figure 3. This is intentional as we observe stable predictive performance when there is no feature reuse via concatenation shortcut after the first softmax block.

In this research, we are very judicious in terms of the considered number of sample points as we focus on training with small training dataset and concentrating most of the samples in the locality of the left corner point where irregularities and spikes resides. We will describe this further in one of the following sections. Moreover, we strictly design each softmax block with two hidden layers that involves small units and that decreases across blocks as described in the experiment section. We think that it makes the network more efficient in terms of training runtime while achieving reasonable predictive performance.

Our configuration entails that hidden layers far away from the input layer will have very small units (we consider up to 4 hidden units). Including the main input feature during feature concatenation might reduce the predictive performance of SoftNet if we consider feature concatenation as an ensemble of input feature and shallow and deep layers. Hence, we drop the main input feature and consider feature reuse via concatenation shortcut after two softmax blocks with reasonable number of units in the first softmax blocks. Finally, we mention that only the output of the final softmax block is used in the multiplicative ensembled layer as the input feature for that layer.

3.6 Modular SoftNet

In this section, we present M-SoftNet-10 and express it as SoftNet with modular configuration based on different response in terms of gradient flow. It has three pathways that lead to three different layers outputs. These three layer outputs are concatenated as one layer before the multiplicative ensembled layer as follows:

$$\mathcal{R}(\mathbf{y}, \nu_{\mathcal{R}}) = \texttt{concat}\left[\mathcal{R}_1(\mathbf{y}, \nu_{\mathcal{R}_1}), \mathcal{R}_2(\mathbf{y}, \nu_{\mathcal{R}_2}), \mathcal{R}_3(\mathbf{y}, \nu_{\mathcal{R}_3})\right]. \tag{3.52}$$

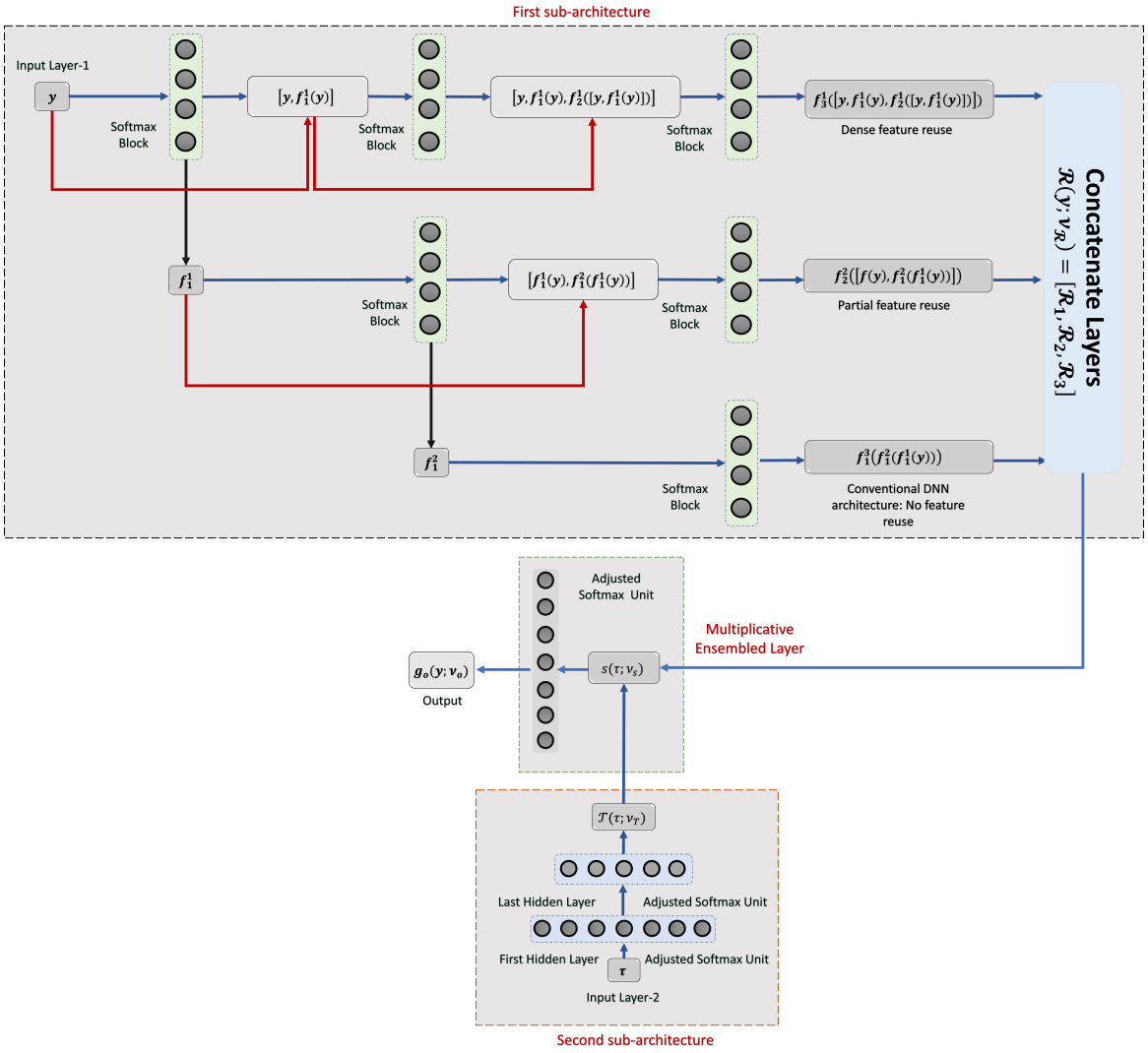


Figure 4: M-SoftNet-10 with six softmax blocks. Each softmax block involves three hidden layers with softmax units. All the softmax blocks have the same representation in terms of number of units and hidden layers.

For M-SoftNet-10, we consider six softmax blocks for this configuration as described in Figure 4. Each softmax block involves three hidden layers with softmax units. The M-SoftNet-10 has some flexibility due to the presence of three different pathways. Here, we elaborate on this flexibility. The

first pathway represents the SoftNet with complete (dense) feature reuse with the output $\mathcal{R}_1(\mathbf{y}, \nu_{\mathcal{R}_1})$ which is the same as (3.51) except that we did not skip feature reuse after the first softmax block. The configuration is as follows:

$$\mathbf{z}_1^1 = \mathbf{y} = [\tau, x, \upsilon, r], \qquad \mathbf{g}_1(\mathbf{z}_1^1; \nu_{1,1}) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_{1,1},1}]; \tag{3.53}$$

$$g_{j,1} = -0.5 \sum_{k=1}^{m_{0,1}} (z_1^1)^k (w_{k,j}^{1,1})^2, \quad \mathcal{A}_1(\mathbf{g}_1(\mathbf{z}_1^1; \nu_{1,1})) = \texttt{softmax}(\mathbf{g}_1(\mathbf{z}_1^1; \nu_{1,1})); \tag{3.54}$$

$$\mathbf{g}_i(\mathbf{z}_1^1; \nu_{i,1}) = [g_{1,i},\ g_{2,i},\ \cdots,\ g_{m_{i,1},i}]; \tag{3.55}$$

$$g_{j,i} = -0.5 \sum_{k=1}^{m_{i-1,1}} (z_1^1)^k (w_{k,j}^{i,1})^2, \quad \mathcal{A}_1(\mathbf{g}_i(\mathbf{z}_1^1; \nu_{i,1})) = \texttt{softmax}(\mathbf{g}_i(\mathbf{z}_1^1; \nu_{i,1})), \qquad i = 2, 3. \tag{3.56}$$

The description above represents the first feature vector and first softmax block. The second feature vector is the coupling (concatenating) of the first feature vector and the output of the first softmax block as given below:

$$\mathbf{z}_1^2 = \texttt{concat}[\mathbf{z}_1^1, \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^1; \nu_{3,1}))] = \texttt{concat}[\tau, x, \upsilon, r, \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^1; \nu_{3,1}))]. \tag{3.57}$$

With the new feature vector, we formulate another softmax block as given in (3.53)-(3.56). We generate the final input feature and network output as follows:

$$\mathbf{z}_1^3 = \texttt{concat}[\mathbf{z}_1^1, \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^1; \nu_{3,1})), \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^2; \nu_{3,2}))] = \texttt{concat}[\mathbf{z}_1^2, \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^2; \nu_{3,2}))], \tag{3.58}$$

$$\mathbf{g}_1(\mathbf{z}_1^3; \nu_{1,3}) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_{1,3},1}]; \tag{3.59}$$

$$g_{j,1} = -0.5 \sum_{k=1}^{m_{0,3}} (z_1^3)^k (w_{k,j}^{1,3})^2, \quad \mathcal{A}_1(\mathbf{g}_1(\mathbf{z}_1^3; \nu_{1,3})) = \texttt{softmax}(\mathbf{g}_1(\mathbf{z}_1^3; \nu_{1,3})); \tag{3.60}$$

$$\mathbf{g}_i(\mathbf{z}_1^3; \nu_{i,3}) = [g_{1,i},\ g_{2,i},\ \cdots,\ g_{m_{i,3},i}], \qquad \mathcal{A}_1(\mathbf{g}_i(\mathbf{z}_1^3; \nu_{i,3})) = \texttt{softmax}(\mathbf{g}_i(\mathbf{z}_1^3; \nu_{i,3})), \qquad i = 2, 3, \tag{3.61}$$

$$g_{j,i} = -0.5 \sum_{k=1}^{m_{i-1,3}} \mathcal{A}_1(\mathbf{g}_i)_k (w_{k,j}^{i,3})^2, \quad \mathcal{A}_1(\mathbf{g}_i(\mathbf{z}_1^3; \nu_{i,3})) = \texttt{softmax}(\mathbf{g}_i(\mathbf{z}_1^3; \nu_{i,3})), \qquad i = 2, 3; \tag{3.62}$$

for which we obtain

$$\mathcal{R}_1(\mathbf{y}; \nu_{\mathcal{R}_1}) = \left[\sum_{j=1}^{m_{3,3}} \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^3; \nu_{i,3}))_j w_{1,j}^1 1, \sum_{j=1}^{m_{3,3}} \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^3; \nu_{i,3}))_j w_{2,j}^1,\ \cdots \sum_{j=1}^{m_{3,3}} \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^3; \nu_{i,3}))_j w_{m_e^1,j} \right]. \tag{3.63}$$

The second pathway involves partial feature reuse with the output given as

$$\mathcal{R}_2(\mathbf{y}; \nu_{\mathcal{R}_2}) = \left[\sum_{j=1}^{m_{3,3}} \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^3; \nu_{i,3}))_j w_{1,j}^2, \sum_{j=1}^{m_{3,3}} \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^3; \nu_{i,3}))_j w_{2,j}^2,\ \cdots \sum_{j=1}^{m_{3,3}} \mathcal{A}_1(\mathbf{g}_3(\mathbf{z}_1^3; \nu_{i,3}))_j w_{m_e^2,j} \right], \tag{3.64}$$

which is similar to (3.51) as we skip feature reuse after the first softmax block. The last pathway is simply a conventional deep learning architecture with softmax units as follows:

$$\mathcal{R}_3(\mathbf{y}, \nu_{\mathcal{R}_3}) = \left[\sum_{j=1}^{m_9} \mathcal{A}_1(\mathbf{g}_9(\mathbf{y}; \nu_9))_j w_{1,j}^3, \sum_{j=1}^{m_9} \mathcal{A}_1(\mathbf{g}_9(\mathbf{y}; \nu_9))_j w_{2,j}^3,\ \cdots \sum_{j=1}^{m_9} \mathcal{A}_1(\mathbf{g}_9(\mathbf{y}; \nu_9))_j w_{m_e^3,j} \right], \tag{3.65}$$

which is similar to (3.40). All the three outputs obtained from these pathways share the same initial feature vector $\mathbf{y}$ but will have different responses in terms of gradient propagation. Direct addition, weighting, or concatenation of the three layers' outputs obtained from these three pathways can be

seen as another form of feature coupling. Here, we consider it as ensembles of layer's outputs involving the same initial feature vector but with different response in terms of feature reuse and gradient flow. Our choice is concatenation of these three layers' outputs and we allow the network to make decision on the importance placed on these three layers' outputs by multiplying each component with an appropriate trainable weight. We further refer the reader to the work of Li et al. [15] where the authors implemented some sort of varying feature reuse strategies.

Finally, we note that in this study, we strictly consider only M-SoftNet-10 which involves modularized nine hidden layers in the first sub-architecture (has more softmax blocks in the first sub-architecture than the non-modularized SoftNet with nine hidden layers). Further increase in depth will increase the width and number of pathways which is nice but will impact on the last pathway involving no feature reuse due to the vanishing gradient effect. The possible way to alleviate such challenge to some extent is to consider less hidden layers in the softmax block. For instance, one can design M-SoftNet-7 that has six pathways and 21 softmax blocks and each block formulated with one hidden layer. One can also design M-SoftNet-9 that has four pathways and 10 softmax blocks and each block formulated with two hidden layers.

4 Design of AddNet

Here, the research question we seek to address is how to design a resource efficient neural network architecture that trains with a reasonable deeper hidden layer thereby reducing vanishing gradient effect and saving computational cost. For instance, the feature reuse via concatenation in DenseNet architecture can increases computational cost. In a more resources constrain devices, this sort of challenge might hamper the deployment of such deep neural network architecture [14]. In this section, we present AddNet, a neural network model with reasonable deeper hidden layers. We formulate AddNet via ensembling of single-mixture of deep learning solutions obtained from local (early) exit and final hidden layers. The network model AddNet is designed with three sub-architectures and trained with deeper hidden layers within the first sub-network architecture. Neural network methods with early exits have been considered in existing literature mostly in the design of convolution neural network for image classification problems [22, 27, 28]. They have also been applied in natural language processing [26] and edge computing [21].

In the design of AddNet, the second and third sub-architecture are configured with two hidden layers and one hidden layer, respectively. The first sub-network architectures is designed with deeper hidden layers, which is the main sub-architecture. It comprises a conventional DNN architecture with softmax units as a backbone network. From this backbone network, n network outputs are systematically obtained from intermediate layers which we denote as "local exit layers" and the final hidden layer. These n network outputs are used to formulate n single mixture of deep learning solutions. Each single mixture of deep learning solutions is formulated via multiplicative ensembling of one of the outputs of the first sub-architecture (from either local exit layers or final hidden layer), and the outputs of the second and third sub-architectures. These single mixture of deep learning solutions are then additively ensembled with systematic importance (weight) attached to each solution to obtain a multi-mixture of deep learning solution. Our formulation is also known as joint training strategy in training strategies for early-exit deep neural network method [23]

4.1 Multi-mixture of deep learning solutions

The design of AddNet involves multi mixture of deep learning solution strategy formulated via early exits and final network output from the first sub-architecture. The early exists are early outputs from the first sub-architecture, which are systematically selected from some of the intermediate hidden layers as shown in Figure 2. This is to possibly, enhance regularization [23, 24, 27] via modularization of deep learning solutions from the outputs of the early exits and final hidden layer. Another reason is to reduce vanishing gradient effect [23] which allows us to train with deeper hidden layer in the

first sub-architecture. The solution framework for AddNet with multi mixture of solution strategy is expressed as follows:

$$\mathcal{M}(\mathbf{y};\nu_M)=\sum_{j=1}^{n}\hat{\pi}_j\mathcal{M}_j(\mathbf{z}_1;\nu_{M,j})=\sum_{j=1}^{n}\hat{\pi}_j\sum_{i=1}^{m}g_{i|j,o}(\mathbf{z}_1;\nu_{i|j,o})e^{-\frac{x^2}{\tau\bar{q}_i(\mathbf{z}_3;\nu_q)+\epsilon}}, \tag{4.1}$$

$$\sum_{j=1}^{n}\hat{\pi}_j=1, \qquad \mathbf{z}_1=(\tau,x,\upsilon,r), \qquad \mathbf{z}_3=(\upsilon,r); \tag{4.2}$$

under the condition that

$$\bar{s}_f(\tau,\upsilon,r;\nu_M)=1-\mathcal{M}(\tau,0,\upsilon,r;\nu_M). \tag{4.3}$$

Here, $\mathcal{M}_j(\mathbf{y};\nu_{M,j})$ represents a single mixture of deep learning solution. By generating n components of single mixture of deep learning solution and ensembling them, we hope to further enhance the predictive performance of AddNet. Here

$$\mathbf{g}_{j|o}(\mathbf{y};\nu_{j|o})=[g_{1|j,o}(\mathbf{y};\nu_{1|j,o}),g_{2|j,o}(\mathbf{y};\nu_{2|j,o}),\cdots g_{m|j,o}(\mathbf{y};\nu_{m|j,o})], \tag{4.4}$$

which is presented in (4.36) for a given j. It is worth noting again from (4.35) that when $\tau=0$

$$\mathbf{s}_j(0,x,\upsilon,r;\nu_{s,j})=0, \qquad \forall j. \tag{4.5}$$

It implies that

$$g_{i|j,o}(0,x,\upsilon,r;\nu_{i|j,o})=0, \qquad \forall j; \tag{4.6}$$

$$\mathcal{M}(0,x,\upsilon,r;\nu_M)=0, \qquad \bar{s}_f(0,\upsilon,r;\nu_M)=1. \tag{4.7}$$

In exact form as presented in (2.12),

$$s_f(0,\upsilon,r)=1, \qquad U(0,x,\upsilon,r)=0. \tag{4.8}$$

Hence,

$$\mathcal{M}(0,x,\upsilon,r;\nu_M)=U(0,x,\upsilon,r), \qquad \bar{s}_f(0,\upsilon,r;\nu_M)=s_f(0,\upsilon,r). \tag{4.9}$$

In the following subsection, we present the designs of AddNet via multi-mixture of deep learning solutions. We further present the design of a conventional DNN architecture for comparison purpose in the experimental section.

4.2 Feature vector

The pricing model is three-factor American Heston-CIR model as described in Section 2. The considered model is in three-dimensional space with the space and time of model parameter given as

$$\vartheta=(\tau,x,\upsilon,r). \tag{4.10}$$

The domain for the parameter is

$$\vartheta\in((0,1],[0,x_{\max}],[0,\upsilon_{\max}],[0,r_{\max}]). \tag{4.11}$$

AddNet involves three network sub-architectures with two different input features. The feature vector for the first sub-architecture is as follows:

$$\mathbf{z}_1=(\tau,x,\upsilon,r). \tag{4.12}$$

For the second sub-architecture, we consider

$$\mathbf{z}_2=\tau. \tag{4.13}$$

For the third sub-architecture, we consider the following feature vector

$$\mathbf{z}_3=(\upsilon,r). \tag{4.14}$$

4.3 Design of the third sub-architecture

The output of the third architecture is as follows:

$$e^{-\frac{x^2}{\tau \mathbf{q}(\mathbf{z}_2;\nu_q)+\epsilon}}, \qquad \mathbf{z}_3 = (\upsilon, r). \tag{4.15}$$

The design of $q_i(\mathbf{y};\nu_q)$ involves only one hidden layer which is described below:

$$\mathbf{q} = [q_1(\mathbf{z}_3;\nu_{q_1}), q_2(\mathbf{z}_3;\nu_{q_2}), \cdots q_m(\mathbf{z}_3;\nu_{q_m})] = \check{\mathbf{h}}\text{softplus}(\hat{W}_o), \qquad \hat{W}_o \in \mathbb{R}^{n_f \times m}; \tag{4.16}$$

$$\check{\mathbf{h}} = [\check{h}_{1,f}, \check{h}_{2,f} \cdots \check{h}_{n_f,f}], \qquad \check{h}_{j,f} = \upsilon \frac{1}{1+(\hat{w}^f_{1,j})^2} + r\frac{1}{1+(\hat{w}^f_{2,j})^2}. \tag{4.17}$$

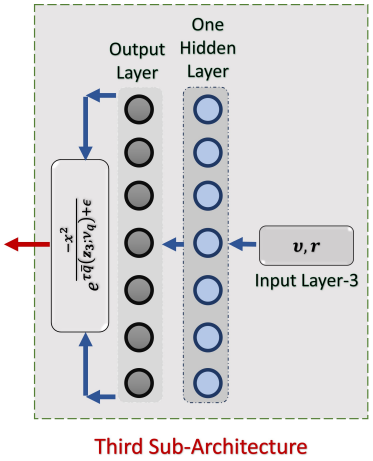


Figure 5: Shallow network for the third sub-architecture.

4.4 Design of the second sub-architecture

We design the second sub-architecture with only two hidden layers and present the network output as given below:

$$\mathcal{T}(\tau;\nu_{\mathcal{T}}) = \sum_{i=1}^{\hat{m}_f} \left(\mathcal{A}_1(\mathbf{h}_f(\tau;\hat{\nu}_f))_j - \frac{1}{\hat{m}_f} \right) \hat{w}^o_{j,i}. \tag{4.18}$$

All the hidden and output layers are formulated without bias term and the choice of the network output for the second sub-architecture is to ensure exact imposition of the terminal conditions. When $\tau = 0$,

$$\mathbf{h}_f(0;\hat{\nu}_f) = \mathbf{0}, \tag{4.19}$$

which we describe in (4.21) and (4.24). Furthermore, we define

$$\mathcal{A}_1(\mathbf{h}_f(0;\hat{\nu}_f)) - \frac{1}{\hat{m}_f} = \mathbf{0}, \qquad \mathcal{T}(0;\nu_{\mathcal{T}}) = 0. \tag{4.20}$$

We further elaborate on $\mathbf{h}(\tau;\hat{\nu}_f)$ which we use in the design of the network output for the second sub-architecture $\mathcal{T}(\tau;\nu_{\mathcal{T}})$. Here, $f = 2$ because of the considered two hidden layers. The description of the architecture based on $\mathbf{h}(\tau;\hat{\nu}_f)$ is given below.

Second hidden layer: The second hidden layer is expressed as follows:

$$\mathcal{A}_2(\mathbf{h}_f) = \mathcal{A}_1(\mathbf{h}_f) - \frac{1}{\hat{m}_f}, \qquad \mathbf{h}_f(\tau;\hat{\nu}_f) = [h_{1,f}(\tau;\hat{\nu}_{1,f}), h_{2,f}(\tau;\hat{\nu}_{2,f}), \cdots, h_{\hat{m}_f,f}(\tau;\hat{\nu}_{\hat{m}_f,f})], \tag{4.21}$$

with

$$h_{j,f} = \sum_{k=1}^{\hat{m}_{f-1}} \mathcal{A}_2(\mathbf{h}_{f-1}(\tau;\hat{\nu}_{f-1}))_k \hat{w}^f_{k,j}. \tag{4.22}$$

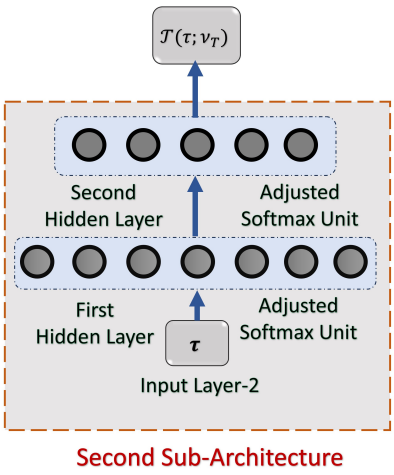


Figure 6: DNN for the second sub-architecture. The second sub-architecture involves units that decrease across hidden layers.

First hidden layer: The first hidden layer is described below:

$$\mathcal{A}_2(\mathbf{h}_{f-1}) = \mathcal{A}_1(\mathbf{h}_{f-1}) - \frac{1}{\hat{m}_{f-1}}, \qquad \mathbf{h}_{f-1} = [h_{1,f-1}, h_{2,f-1}, \cdots, h_{\hat{m}_{f-1},f-1}], \tag{4.23}$$

with

$$h_{j,f-1} = \tau \hat{w}_j^1. \tag{4.24}$$

Figure 6 shows the description of the second sub-architecture.

4.5 Design of the first sub-architecture for conventional DNN architecture and AddNet

The first sub-architecture for AddNet with deeper hidden layer and softmax blocks involves no feature reuse. First, we consider a conventional DNN architecture before the multiplicative ensembled layer as follows:

Input Layer: The input layer for AddNet is as follows:

$$\mathbf{z}_1 = \mathbf{y} = (\tau, x, \upsilon, r). \tag{4.25}$$

Hidden Layers and Local Exits: The $\hat{n}$ hidden layer for AddNet architecture is given below:

$$\mathbf{g}_1(\mathbf{y};\nu_1) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_1,1}]; \tag{4.26}$$

$$g_{j,1} = -0.5\sum_{j=1}^{m_0} y_j (w_j^1)^2, \qquad \mathcal{A}_1(\mathbf{g}_1(\mathbf{y};\nu_1)) = \texttt{softmax}(\mathbf{g}_1(\mathbf{y};\nu_1)); \tag{4.27}$$

$$\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}}) = [g_{1,\hat{k}},\ g_{2,\hat{k}},\ \cdots,\ g_{m_{\hat{k}},\hat{k}}], \qquad \hat{k} = 2,\ 3,\ \hat{n}; \tag{4.28}$$

$$g_{j,\hat{k}} = -0.5\sum_{j=1}^{m_{\hat{k}-1}} \mathcal{A}_1(\mathbf{g}_{\hat{k}-1})_j (w_j^{\hat{k}})^2, \qquad \mathcal{A}_1(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})) = \texttt{softmax}(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})), \quad \hat{k} = 2,\ 3,\ \hat{n}. \tag{4.29}$$

Instead of considering only $\mathcal{A}_1(\mathbf{g}_{\hat{n}}(\mathbf{y};\nu_{\hat{n}}))$ as the feature for the multiplicative ensembled layer as shown in Figure 7 for the conventional DNN architecture, we systematically select $n-1$ intermediate layers which we denote as "local exit layers":

$$\{\mathcal{A}_1(\mathbf{g}_{l\bar{a}}(\mathbf{y};\nu_{l\bar{a}})),\ \ l = 1, 2, \cdots n-1\} \subset \{\mathcal{A}_1(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})),\ \ \hat{k} = 1, 2, \cdots \hat{n}\}. \tag{4.30}$$

We note that

$$\mathcal{R}_l^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},l\bar{a}}) = \left[\sum_{j=1}^{m_{\hat{k}}} \mathcal{A}_1(\mathbf{g}_{l\bar{a}}(\mathbf{y};\nu_{l\bar{a}}))_j w_{1,j}^{e_l}, \sum_{j=1}^{m_{\hat{k}}} \mathcal{A}_1(\mathbf{g}_{l\bar{a}}(\mathbf{y};\nu_{l\bar{a}}))_j w_{2,j}^{e_l}, \cdots \sum_{j=1}^{m_{\hat{k}}} \mathcal{A}_1(\mathbf{g}_{l\bar{a}}(\mathbf{y};\nu_{l\bar{a}}))_j w_{m_e,j}^{e_l}\right]; \tag{4.31}$$

$$l = 1, 2, \cdots n-1;$$

$$\mathcal{R}_n^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},n\bar{a}}) = \left[\sum_{j=1}^{m_{\hat{k}}} \mathcal{A}_1(\mathbf{g}_{\hat{n}}(\mathbf{y};\nu_{\hat{n}}))_j w_{1,j}^{e_n}, \sum_{j=1}^{m_{\hat{k}}} \mathcal{A}_1(\mathbf{g}_{\hat{n}}(\mathbf{y};\nu_{\hat{n}}))_j w_{2,j}^{e_n}, \cdots, \sum_{j=1}^{m_{\hat{k}}} \mathcal{A}_1(\mathbf{g}_{\hat{n}}(\mathbf{y};\nu_{\hat{n}}))_j w_{m_e,j}^{e_n}\right]; \tag{4.32}$$

$$\bar{a} \in \{2, 3, 4\}, \qquad n\bar{a} = \hat{n}. \tag{4.33}$$

With the outputs of these $n-1$ local exit and final hidden layers from the first sub-architecture, we formulate n multiplicative ensembled layers, as shown in Figure 8 for further description.

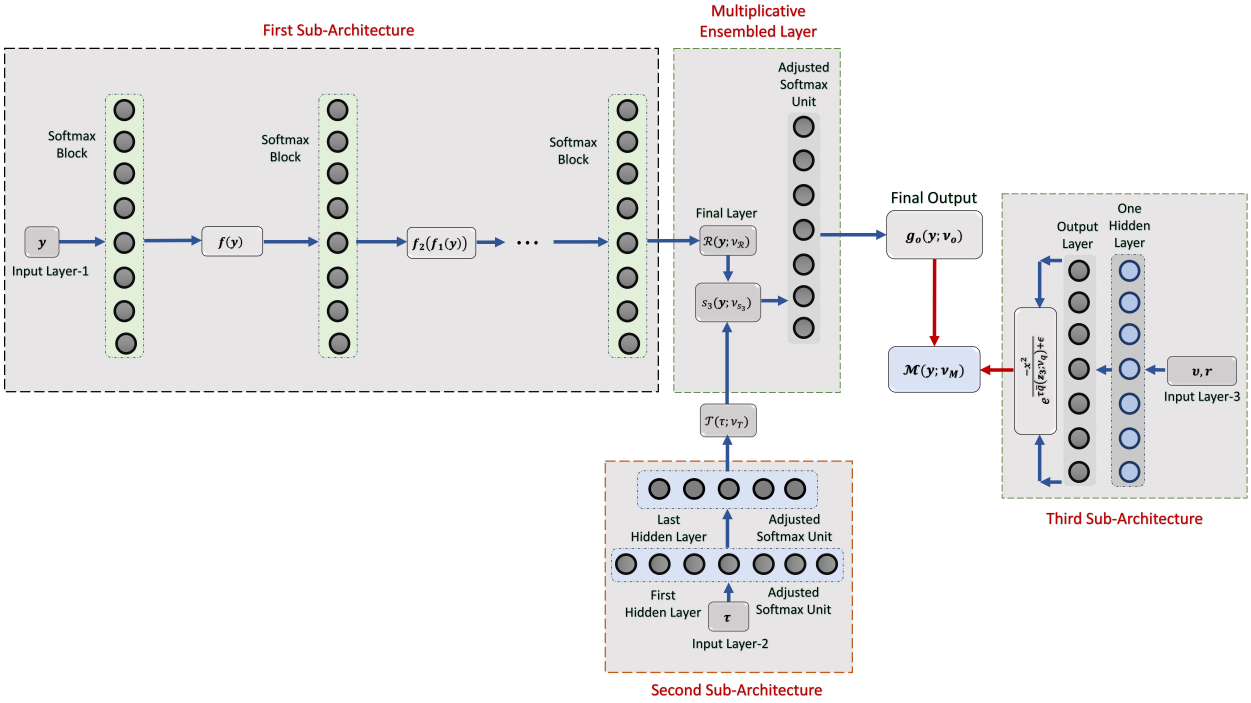


Figure 7: Conventional DNN architecture.

4.6 Design of n multiplicative ensembled layers for multi mixture of solutions strategy

AddNet involves n multiplicative ensembled layers due to the considered outputs of local exit and final hidden layers. The feature for each multiplicative ensembled layer is as follows:

$$\mathbf{s}_j(\mathbf{y};\nu_{s,j}) = \mathcal{T}(\tau;\nu_{\mathcal{T}})\mathcal{R}_j^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},j\bar{a}}), \quad \mathcal{R}_j^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},j\bar{a}}) \in \mathbb{R}^{1\times m_e}, \quad \mathcal{T}(\tau;\nu_{\mathcal{T}}) \in \mathbb{R}, \quad j = 1, 2, \cdots n-1. \tag{4.34}$$

We have mentioned earlier that all the hidden and output layers are formulated without bias term and the second sub-architecture is designed with adjusted softmax activation function to ensure $\mathcal{T}(0;\nu_{\mathcal{T}}) = 0$ which implies that

$$\mathbf{s}_j(0, x, \upsilon, r;\nu_{s,j}) = 0, \qquad \forall j. \tag{4.35}$$

Moreover, for $i|j$, the multiplicative ensembled layer is as follows:

$$g_{i|j,o} = \sum_{k=1}^{m_e} \left(\mathcal{A}_1(\mathbf{s}_j(\mathbf{y};\nu_{s,j}))_k - \frac{1}{m_e}\right) w_{k,j,i}^{o}, \tag{4.36}$$

$$\mathcal{A}_1(\mathbf{s}_j) = \texttt{softmax}(\mathbf{s}_j) = \left[\frac{e^{(s_j)_1}}{\sum_{i=1}^{m_e} e^{(s_j)_i}}, \cdots \frac{e^{(s_j)_{m_e}}}{\sum_{i=1}^{m_e} e^{(s_j)_i}} \right], \tag{4.37}$$

$$\mathcal{A}_1(\mathbf{s}_j)_k = \texttt{softmax}(\mathbf{s}_j)_k = \frac{e^{(s_j)_k}}{\sum_{i=1}^{m_e} e^{(s_j)_i}}. \tag{4.38}$$

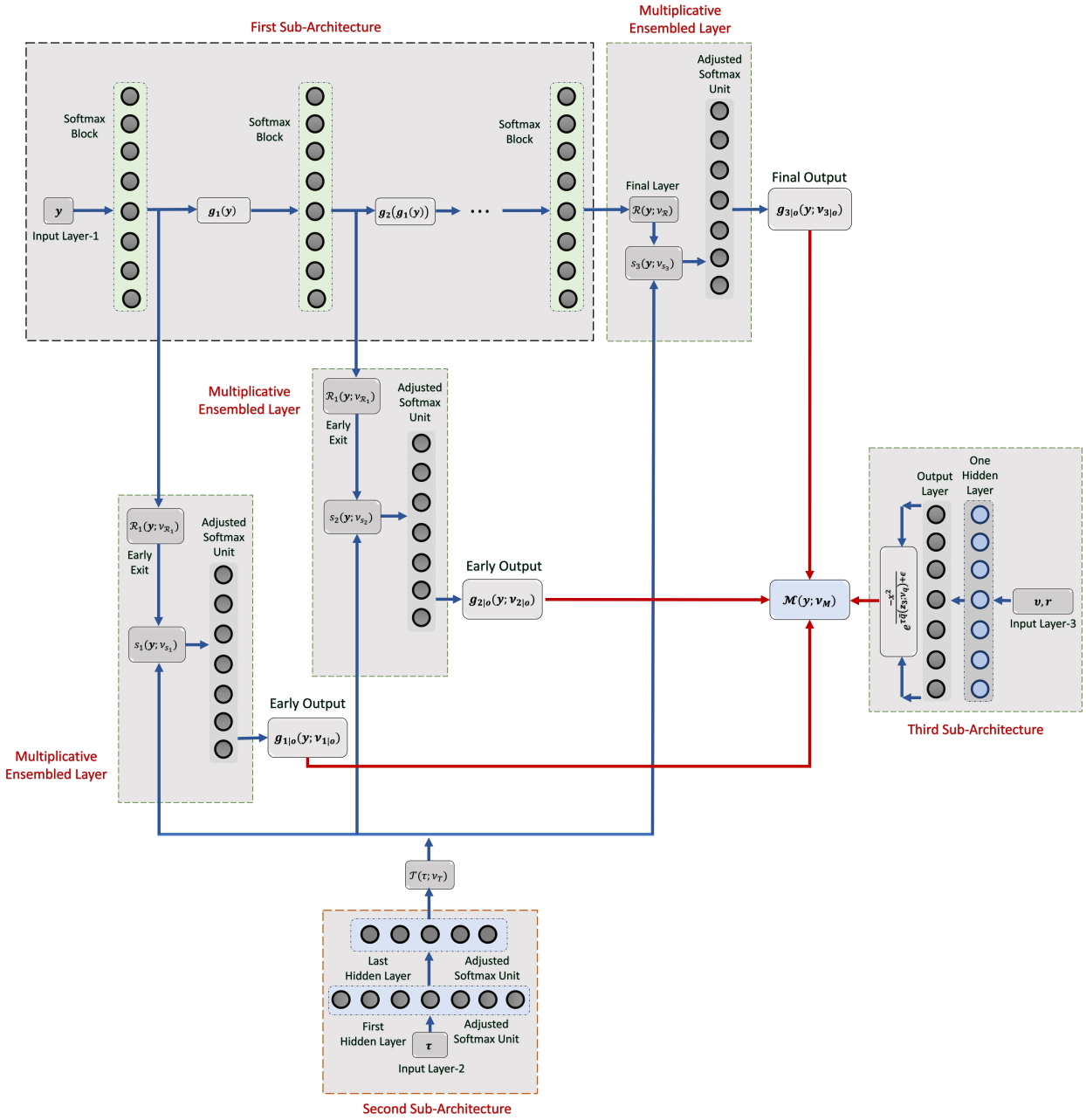


Figure 8: AddNet architecture.

4.7 Short note on the selection of local exit layers

The selection of local exit layers in this research is somewhat systematic. Staying some hidden layers away from the input layer in the choice of earlier outputs based on local exit layers might be crucial. Shallow layers might not be able to extract more complex feature. This scenario might reduce or even deteriorate predictive performance if local exit layers from shallower hidden layers are selected. In this regard, we refer the reader to the work of Sarti et al. [24] where the authors described similar scenario in the experiment section of their literature.

If we consider averaging of these single-mixture of deep learning solutions, equal contribution could have been given to each single mixture of deep learning solutions. If we have selected shallow hidden layer in this instance, AddNet might present poor predictive performance. We hope to further investigate this phenomenon in the experiment section. We mention that we will consider couple of weighting strategies in the experiment section.

We assume that an architecture with three to five hidden layers (including the multiplicative ensembling layer which is present in our network design before the output layer) is reasonably deep to extract reasonable complex feature and information. In this regard, we consider

$$\mathcal{R}_1^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},\bar{a}}) \subset \mathcal{R}_2^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},2\bar{a}}) \subset \cdots \subset \mathcal{R}_n^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},n\bar{a}}) \qquad \bar{a} \in \{2,3,4\}, \qquad n\bar{a} = \hat{n}. \tag{4.39}$$

Here, $\mathcal{R}_1^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},\bar{a}})$ is the first local exit layer and we neither want it to be too shallow to lack the capacity of extracting complex features nor too deep to be impacted by the vanishing gradient effect. It is important to note that $\mathcal{R}_j^{\bar{a}}(\mathbf{y};\nu_{\mathcal{R},\bar{a}})$ which represents the output of the j^{th} local exit layer is the $j\bar{a}$ hidden layer. For instance, if $j = 2$ and $\bar{a} = 4$, then $\mathcal{R}_2^4(\mathbf{y};\nu_{\mathcal{R},\bar{a}})$ is the output of the eight hidden layers.

5 Design of LinNet

The design of LinNet solution involves three sub-architectures which we ensemble multiplicatively. The first sub-architecture which is the main architecture is designed via softmax blocks and trained with deeper hidden layers. The second sub-architecture is designed with adjusted softmax units, strictly involves two hidden layers and is configured to ensure exact imposition of some of the terminal conditions. The third sub-network is designed with one hidden layer.

Though LinNet has three sub-architectures, its deeper structure is from the first sub-architecture which is formulated via softmax blocks and systematic aggregation of linear units before each softmax block. Furthermore, we present the design of the conventional DNN architecture without aggregation of linear units. The design of these two deep learning solution frameworks is described in the following subsections for this section.

5.1 Feature vector

The three-factor American Heston-CIR model with free boundary feature as described in Section 2 is in three-dimensional space which the model parameters denoted as

$$\vartheta_u = (\tau, x, \upsilon, r). \tag{5.1}$$

for the value function and its derivatives (Greeks). Moreover, the time and space of model parameters for the early exercise boundary are denoted as follows:

$$\vartheta_s = (\tau, \upsilon, r). \tag{5.2}$$

The domains for these parameters are given below:

$$\vartheta_u \in ((0,1],[0,x_{\max}],[0,\upsilon_{\max}],[0,r_{\max}]), \qquad \vartheta_s \in ((0,1],[0,\upsilon_{\max}],[0,r_{\max}]). \tag{5.3}$$

The network model LinNet involves three sub-architectures with different input features. The feature vector for the first sub-architecture is denoted as:

$$\mathbf{y} = (\tau, x, \upsilon, r). \tag{5.4}$$

For the second sub-architecture, the input feature we consider is simply τ. For the third sub-architecture, we consider the following feature vector as

$$\mathbf{z} = (\upsilon, r). \tag{5.5}$$

5.2 Mixture of deep learning solution

We note that both the conventional DNN architecture (without aggregation of linear units) and LinNet have the same solution framework. The difference of LinNet from the conventional DNN architecture (without aggregation of linear units) lies in the design of the first sub-architecture. The general solution framework is formulated via a mixture of deep learning solutions and is expressed as given below:

$$\mathcal{M}(\mathbf{y};\nu_M)=\sum_{i=1}^{n} g_{i,o}(\mathbf{y};\nu_{i,o})\exp\left(\frac{[x-\tau\bar{q}_i(\mathbf{z};\nu_{\bar{q},i})]^2}{\tau q_i(\mathbf{z};\nu_{q,i})+\epsilon}\right), \qquad \mathbf{z}=(\upsilon,r), \qquad \mathbf{y}=(\tau,x,\upsilon,r); \tag{5.6}$$

$$\mathbf{q}\in\mathbb{R}_{+}^{1\times n}, \qquad \bar{\mathbf{q}}\in\mathbb{R}^{1\times n}, \tag{5.7}$$

under the condition

$$\bar{s}_f(\tau,\upsilon,r;\nu_M)=1-\mathcal{M}(\tau,0,\upsilon,r;\nu_M). \tag{5.8}$$

From (5.6), n components of deep learning solutions are ensembled additively to obtain LinNet. Here, we define

$$\mathbf{g}_o(\mathbf{y};\nu_o)=[g_{1,o}(\mathbf{y};\nu_{1,o}),g_{2,o}(\mathbf{y};\nu_{2,o}),\cdots,g_{n,o}(\mathbf{y};\nu_{n,o})]. \tag{5.9}$$

We give further details on the designs of $\mathbf{q}$ and $\bar{\mathbf{q}}$ in one of the subsequent subsections. In the next subsection, we discuss the design of the multiplicative ensembled layer with the output is given as $\mathbf{g}_o(\mathbf{y};\nu_o)$.

5.3 Multiplicative ensembled layer

From the mixture of solution framework given in (5.6), we define

$$\mathbf{g}_o(\mathbf{y};\nu_o)=[g_{1,o}(\mathbf{y};\nu_{1,o}),g_{2,o}(\mathbf{y};\nu_{2,o}),\cdots g_{n,o}(\mathbf{y};\nu_{n,o})], \tag{5.10}$$

which represents the output of the multiplicative ensembled layer. We note that the multiplicative ensembled layer is involved in both LinNet and the conventional DNN architecture (without aggregation of linear units). Here,

$$g_{i,o}=\sum_{j=1}^{m_f}\left(\mathcal{A}_1(\mathbf{s}(\mathbf{y};\nu_s))_j-\frac{1}{m_f}\right)w_{j,i}^{o}; \tag{5.11}$$

$$\mathcal{A}_1(\mathbf{s})=\texttt{softmax}(\mathbf{s})=\left[\frac{e^{s_1}}{\sum_{i=1}^{m_f}e^{s_i}},\cdots,\frac{e^{s_{m_f}}}{\sum_{i=1}^{m_f}e^{s_i}}\right], \qquad \mathcal{A}_1(\mathbf{s})_j=\texttt{softmax}(\mathbf{s})_j=\frac{e^{s_j}}{\sum_{i=1}^{m_f}e^{s_i}}; \tag{5.12}$$

$$j=1,2,\cdots,m_f, \tag{5.13}$$

with

$$\mathbf{s}(\mathbf{y};\nu_s)=\mathcal{T}(\tau;\nu_{\mathcal{T}})\mathcal{R}(\mathbf{y};\nu_R), \qquad \mathcal{R}(\mathbf{y};\nu_R)\in\mathbb{R}^{1\times n}, \qquad \mathcal{T}(\tau;\nu_{\mathcal{T}})\in\mathbb{R}. \tag{5.14}$$

In this layer as given in (5.14), the output of the first sub-architecture is denoted as $\mathcal{R}(\mathbf{y};\nu_R)$ and the second sub-architecture given as $\mathcal{T}(\tau;\nu_{\mathcal{T}})$ are ensembled multiplicatively as $\mathbf{s}(\mathbf{y};\nu_s)$. We note that all the hidden and output layers are formulated without bias term. Moreover, the second sub-architecture is designed with adjusted softmax activation function to ensure that when $\tau=0$,

$$\mathcal{T}(0;\nu_{\mathcal{T}})=0, \qquad \mathbf{s}(0,x,\upsilon,r;\nu_s)=\mathbf{0}, \qquad \mathbf{g}_o(0,x,\upsilon,r;\nu_o)=\mathbf{0}; \tag{5.15}$$

$$\mathcal{M}(0,x,\upsilon,r;\nu_M)=0, \qquad \bar{s}_f(\tau,\upsilon,r;\nu_M)=1. \tag{5.16}$$

This is the main purpose of the multiplicative ensembled layer which is to ensure that some of the terminal conditions are satisfied in exact form (after removing the bias term from the hidden and output layers). It is worth noting that in the exact form as presented in (2.12),

$$s_f(0,\upsilon,r)=1, \qquad U(0,x,\upsilon,r)=0. \tag{5.17}$$

It implies that

$$\mathcal{M}(0,x,\upsilon,r;\nu_M)=U(0,x,\upsilon,r), \qquad \bar{s}_f(0,\upsilon,r;\nu_M)=s_f(0,\upsilon,r). \tag{5.18}$$

We further discuss the design of $\mathcal{R}(\mathbf{y};\nu_R)$ (for LinNet and the conventional DNN architecture) and $\mathcal{T}(\tau;\nu_{\mathcal{T}})$ in the subsequent subsections.

5.4 Suite of designs for the third sub-architecture

Here, we consider suite of designs for the third sub-architecture. In our earlier works, we have considered several formulations for the exponential terms which multiply to the network output to obtain our DNN solution framework. Moreover, in one of our previous works [20], we furnished the exponential term with a network output as given below:

$$\exp\left(-\frac{x^2}{\tau\mathbf{q}+\epsilon}\right), \qquad \mathbf{z}=(\upsilon, r). \tag{5.19}$$

In such scenario, $\bar{\mathbf{q}}=\mathbf{0}$. The design of $\mathbf{q}$ involves only one hidden layer which is described below:

$$\mathbf{q}=[q_1(\mathbf{z};\nu_{q_1}), q_2(\mathbf{z};\nu_{q_2}), \cdots q_m(\mathbf{z};\nu_{q_m})] = \check{\mathbf{h}}\text{softplus}(\hat{W}_o), \qquad \hat{W}_o\in\mathbb{R}^{n_f\times m}; \tag{5.20}$$

$$\check{\mathbf{h}}=[\check{h}_{1,f}, \check{h}_{2,f}\cdots\check{h}_{n_f,f}], \qquad \check{h}_{j,f}=\upsilon\frac{1}{1+(\hat{w}^f_{1,j})^2}+r\frac{1}{1+(\hat{w}^f_{2,j})^2}. \tag{5.21}$$

In this research, we consider couple of options. The first option is similar to the one in (5.19) except that the design of $\mathbf{q}(\mathbf{z};\nu_q)$ has n different network structure as given below:

$$\mathbf{q}=[q_1(\mathbf{z};\nu_{q_1}), q_2(\mathbf{z};\nu_{q_2}), \cdots q_m(\mathbf{z};\nu_{q_m})] = \texttt{concat}\left[\texttt{arctan}(0.3\mathbf{z}), \check{\mathbf{h}}\right]\text{softplus}(\hat{W}_o), \quad \hat{W}_o\in\mathbb{R}^{(2+n_f)\times m}. \tag{5.22}$$

Please see Figure 9 The second option differs entirely from the first option in both network output and structure as follows:

$$\exp\left(-\frac{[x-\tau\bar{\mathbf{q}}]^2}{\tau\mathbf{q}+\epsilon}\right), \qquad \mathbf{z}=(\upsilon, r), \tag{5.23}$$

with

$$\bar{\mathbf{q}}=[\bar{q}_1(\mathbf{z};\nu_{\bar{q}_1}), \bar{q}_2(\mathbf{z};\nu_{\bar{q}_2}), \cdots \bar{q}_m(\mathbf{z};\nu_{\bar{q}_m})] = \check{\mathbf{h}}+\check{\mathbf{h}}\frac{\bar{W}^1_o}{1+\left(\bar{W}^1_o\right)^2}, \qquad \bar{W}^1_o\in\mathbb{R}^{n_f\times m}; \tag{5.24}$$

$$\mathbf{q}=[q_1(\mathbf{z};\nu_{q_1}), q_2(\mathbf{z};\nu_{q_2}), \cdots q_m(\mathbf{z};\nu_{q_m})] = \check{\mathbf{h}}+\check{\mathbf{h}}\text{softplus}(\bar{W}^2_o), \qquad \bar{W}^2_o\in\mathbb{R}^{n_f\times m}; \tag{5.25}$$

$$\check{\mathbf{h}}=[\check{h}_{1,f}, \check{h}_{2,f}\cdots\check{h}_{n_f,f}], \qquad \check{h}_{j,f}=\upsilon\frac{1}{1+(\hat{w}^f_{1,j})^2}+r\frac{1}{1+(\hat{w}^f_{2,j})^2}. \tag{5.26}$$

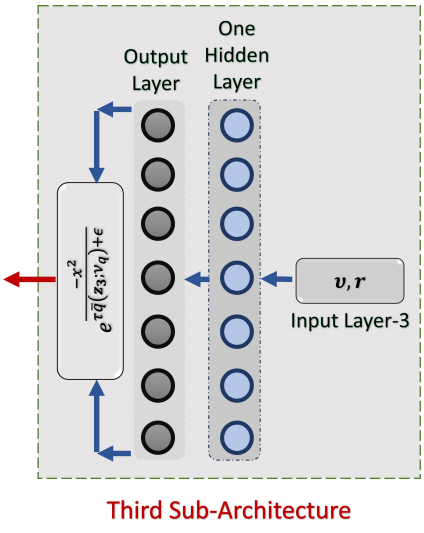


Figure 9: DNN for the third sub-architecture (first option). We note that the output of the network architecture is given in (5.19)

The second option for the third sub-architecture as presented in (5.23)-(5.26) includes a network output $\bar{\mathbf{q}}$ in the numerator of the exponential term. We envisage that it could be an important addition in the numerator. It will always act on x to either increase or decrease the value in the numerator (force the exponential term to tend to either zero or one rapidly or slowly). The consideration could be very important because of the far boundary behaviours. For instance, increasing in υ increases the value function while the value function vanishes as x increases. There is a tendency that $\bar{\mathbf{q}}$ and $\mathbf{q}$ might shift and scale the input variable x appropriately to control how the exponential term vanishes or grow. Here, we ensure that the $\bar{\mathbf{q}}$ does not grows rapidly by acting a nonlinear function to the weight values which vanishes as the weight values move away from zero value.

Remark. *It is worth noting that the considered two designs for the third sub-architecture is not final. It is still an ongoing research for us and we seek to develop a more unified framework for the exponential term that can address wider range of options pricing models especially in a very high-dimensional setting that could involve more variables and parameters. We further mention that the major criteria that the exponential term might need to satisfy is the indeterminate condition when $x = 0$ and $\tau = 0$ The introduction of ϵ is to regulate and control this response.*

5.5 Design of the second sub-architecture

The second sub-architecture for LinNet is designed with only two hidden layers. The network is given below:

$$\mathcal{T}(\tau;\nu_{\mathcal{T}}) = \sum_{i=1}^{\hat{m}_f} \left(\mathcal{A}_1(\mathbf{h}_f(\tau;\hat{\nu}_f))_j - \frac{1}{\hat{m}_f}\right) \hat{w}^o_{j,i}. \tag{5.27}$$

For the second sub-architecture, the configuration of the network design and its output ensures exact imposition of the terminal conditions such that when $\tau = 0$,

$$\mathbf{h}_f(0;\hat{\nu}_f) = \mathbf{0}, \tag{5.28}$$

which we described in (3.31) and (5.35). It implies that

$$\mathcal{A}_1(\mathbf{h}_f(0;\hat{\nu}_f)) - \frac{1}{\hat{m}_f} = \frac{\mathbf{1}}{\hat{m}_f} - \frac{1}{\hat{m}_f} = \mathbf{0}, \qquad \mathcal{T}(0;\nu_{\mathcal{T}}) = 0; \tag{5.29}$$

$$T(0;\nu_{\mathcal{T}}) = 0, \qquad \mathbf{s}(0,x,\upsilon,r;\nu_s) = \mathbf{0}, \qquad \mathbf{g}_o(0,x,\upsilon,r;\nu_o) = \mathbf{0}; \tag{5.30}$$

$$\mathcal{M}(0,x,\upsilon,r;\nu_M) = 0, \qquad \bar{s}_f(\tau,\upsilon,r;\nu_M) = 1. \tag{5.31}$$

Below, we give further description on the design of $\mathbf{h}(\tau;\hat{\nu}_f)$ used in the formulation of $\mathcal{T}(\tau;\nu_{\mathcal{T}})$. Here, $f = 2$ because of the considered two hidden layers.

Second hidden layer: The second hidden layer is expressed as follows:

$$\mathcal{A}_2(\mathbf{h}_f) = \mathcal{A}_1(\mathbf{h}_f) - \frac{1}{\hat{m}_f}, \qquad \mathbf{h}_f(\tau;\hat{\nu}_f) = [h_{1,f}(\tau;\hat{\nu}_{1,f}), h_{2,f}(\tau;\hat{\nu}_{2,f}), \cdots, h_{\hat{m}_f,f}(\tau;\hat{\nu}_{\hat{m}_f,f})], \tag{5.32}$$

with

$$h_{j,f} = \sum_{k=1}^{\hat{m}_{f-1}} \mathcal{A}_2(\mathbf{h}_{f-1}(\tau;\hat{\nu}_{f-1}))_k \hat{w}^f_{k,j}. \tag{5.33}$$

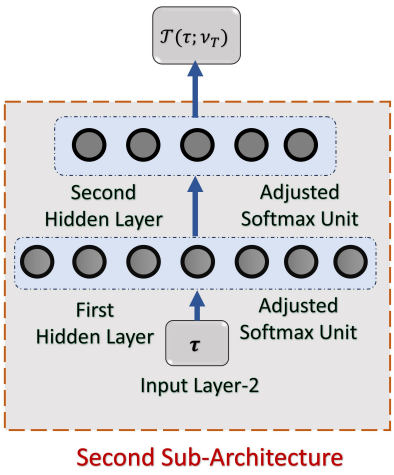


Figure 10: DNN for the second sub-architecture. The second sub-architecture involves units that decrease across hidden layers. The output "$\mathcal{T}(\tau, \nu_{\mathcal{T}})$" is given in (5.27).

First hidden layer: The first hidden layer is described below:

$$\mathcal{A}_2(\mathbf{h}_{f-1}) = \mathcal{A}_1(\mathbf{h}_{f-1}) - \frac{1}{\hat{m}_{f-1}}, \qquad \mathbf{h}_{f-1} = [h_{1,f-1}, h_{2,f-1}, \cdots, h_{\hat{m}_{f-1},f-1}], \tag{5.34}$$

with

$$h_{j,f-1} = \tau \hat{w}_j^1. \tag{5.35}$$

The description of the second sub-architecture is given in Figure 10. It is important to observe from (5.35) that when $\tau = 0$, $h_{j,f-1} = 0$. It implies that $\mathbf{h}_{f-1} = \mathbf{0}$, $\mathcal{A}_2(\mathbf{h}_{f-1}) = \mathbf{0}$, $h_{j,f} = 0$, and $\mathcal{A}_2(\mathbf{h}_f) = \mathbf{0}$.

5.6 Design of first sub-architecture without aggregation of linear units

In this subsection, we give the description of the first sub-architecture without aggregation of linear units. First, we present the feature vector for the first sub-architecture as follows:

$$\mathbf{y} = (\tau, x, \upsilon, r). \tag{5.36}$$

The configuration of the hidden layers is described below.

First hidden layer: Let $\hat{k}$ represent the $\hat{k}^{th}$ hidden layer and $\hat{n}$ represent the number of hidden layers before the multiplicative ensembled layer. For the first hidden layer with $\hat{k} = 1$, the network is expressed as follows:

$$\mathbf{g}_1(\mathbf{y}; \nu_1) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_1,1}]; \tag{5.37}$$

$$g_{j,1} = -0.5 \sum_{k=1}^{m_0} y_k (w_{k,j}^1)^2, \qquad \mathcal{A}_1(\mathbf{g}_1(\mathbf{y}; \nu_1)) = \texttt{softmax}(\mathbf{g}_1(\mathbf{y}; \nu_1)); \tag{5.38}$$

where m_0 is the number of units.

Remaining hidden layers before multiplicative ensembled layer: For $\hat{k} = 2,\ 3,\ \cdots \hat{n}$, the description of the hidden layers is as follows:

$$\mathbf{g}_{\hat{k}}(\mathbf{y}; \nu_{\hat{k}}) = [g_{1,\hat{k}},\ g_{2,\hat{k}},\ \cdots,\ g_{m_{\hat{k}},\hat{k}}], \qquad \hat{k} = 2,\ 3,\ \cdots \hat{n}; \tag{5.39}$$

$$g_{j,\hat{k}} = -0.5 \sum_{k=1}^{m_{\hat{k}-1}} \mathcal{A}_1(\mathbf{g}_{\hat{k}-1})_k (w_{k,j}^{\hat{k}})^2, \qquad \mathcal{A}_1(\mathbf{g}_{\hat{k}}(\mathbf{y}; \nu_{\hat{k}})) = \texttt{softmax}(\mathbf{g}_{\hat{k}}(\mathbf{y}; \nu_{\hat{k}})), \quad \hat{k} = 2,\ 3,\ \cdots \hat{n}, \tag{5.40}$$

where $m_{\hat{k}-1}$ is the number of units. Here,

$$\mathcal{R}_j(\mathbf{y};\nu_R) = \sum_{k=1}^{m_{\hat{n}}} \mathcal{A}_1(\mathbf{g}_{\hat{n}})_k w_{k,j}^{\hat{n}+1}, \qquad j = 1, 2, \cdots m_{\hat{n}+1}, \tag{5.41}$$

which represents the output for the final hidden layer in the first sub-architecture for the conventional DNN architecture without aggregation of linear units, as shown in Figure 11. Here, $m_{\hat{n}+1}$ is the number of units.

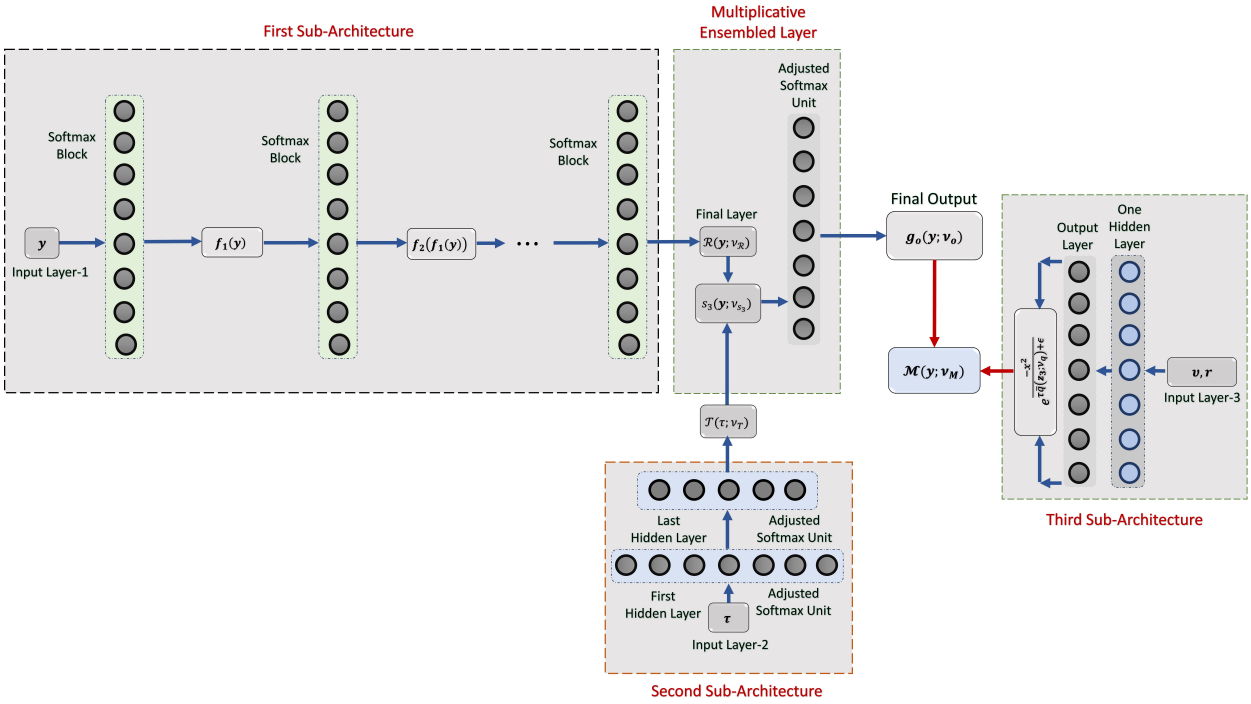


Figure 11: Conventional DNN architecture.

We note that "$\mathcal{T}(\tau, \nu_{\mathcal{T}})$" is given in (5.27), "$\mathcal{R}(\mathbf{y}, \nu_{\mathcal{R}})$" in (5.60), and "$s(\mathbf{y}, \nu_s)$" in (5.14). The final output "$\mathcal{M}(\mathbf{y}, \nu_{\mathcal{M}})$" which represents the solution framework is described in (5.6).

5.7 LinNet: Design of the first sub-architecture via aggregation of linear units

In this subsection, we present the design of the first sub-architecture for LinNet which involves softmax blocks and systematic aggregation of linear units. We mention that all softmax blocks are designed with two hidden layers and each layer is designed with softmax unit. The description for the design of the first sub-architecture within LinNet is given below.

First aggregating unit: The first aggregating unit is simply the first hidden layer which is more like an initializer and is given below:

$$\mathbf{y} = (\tau, x, \upsilon, r). \tag{5.42}$$

$$\mathbf{g}_1(\mathbf{y};\nu_1) = [g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_1,1}]; \tag{5.43}$$

$$g_{j,1} = -0.5 \sum_{k=1}^{m_0} y_k (w_{k,j}^1)^2, \qquad \mathcal{A}_1(\mathbf{g}_1(\mathbf{y};\nu_1)) = \texttt{softmax}(\mathbf{g}_1(\mathbf{y};\nu_1)). \tag{5.44}$$

First softmax block: The design of the first softmax block is as follows:

$$\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}}) = [g_{1,\hat{k}},\ g_{2,\hat{k}},\ \cdots,\ g_{m_{\hat{k}},\hat{k}}]; \tag{5.45}$$

$$g_{j,\hat{k}} = -0.5 \sum_{j=1}^{m_{\hat{k}-1}} \mathcal{A}_1(\mathbf{g}_{\hat{k}-1})_k (w_{k,j}^{\hat{k}})^2, \qquad \mathcal{A}_1(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})) = \texttt{softmax}(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})), \quad \hat{k} = 2,\ 3. \tag{5.46}$$

Second aggregating unit: The second aggregating units is as follows:

$$\mathbf{g}_4(\mathbf{y};\nu_4) = [\bar{a}_{1,2},\ \bar{a}_{2,2},\ \cdots,\ \bar{a}_{m_4,2}]; \tag{5.47}$$

$$\bar{a}_{1,2} = \bar{a}_{1,1} + g_{1,4}, \quad \bar{a}_{2,2} = \bar{a}_{2,1} + g_{2,4}, \quad \cdots, \quad \bar{a}_{m_4,2} = \bar{a}_{m_4,1} + g_{m_4,4}; \tag{5.48}$$

$$g_{j,4} = -0.5\sum_{k=1}^{m_3} \mathcal{A}_1(\mathbf{g}_3)_k (w_{k,j}^4)^2, \qquad \bar{a}_{j,1} = -0.5\sum_{k=1}^{m_1} \mathcal{A}_1(\mathbf{g}_1)_k (\bar{w}_{k,j}^2)^2; \tag{5.49}$$

$$\mathcal{A}_1(\mathbf{g}_4(\mathbf{y};\nu_4)) = \texttt{softmax}(\mathbf{g}_4(\mathbf{y};\nu_4)). \tag{5.50}$$

We note that the first and second aggregating units are slightly different from the rest of aggregating units. For the second aggregating unit, from (5.43), we could have directly considered

$$[g_{1,1},\ g_{2,1},\ \cdots,\ g_{m_1,1}] = [\bar{a}_{1,1},\ \bar{a}_{2,1},\ \cdots,\ \bar{a}_{m_1,1}]. \tag{5.51}$$

However, we want to stay a hidden layer away from the input feature as we do not want the contribution of the linear unit from the first hidden layer. This choice is with regards to our experimental observation based on predictive performance and minimization of the loss minimizes when the input feature is directly involved. Moreover, all the layers in the aggregating units must have the same output dimension and we want to have more neurons in the shallow layers. Hence, it is a need to bypass the linear units from the first hidden layer and to formulate another linear units with the output of the first hidden layer whose weight value $\bar{w}_{k,j}^2$ is different from $w_{k,j}^2$ as shown in (5.49).

Second softmax block: For the second softmax block, we obtain as follows:

$$\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}}) = [g_{1,\hat{k}},\ g_{2,\hat{k}},\ \cdots,\ g_{m_{\hat{k}},\hat{k}}]; \tag{5.52}$$

$$g_{j,\hat{k}} = -0.5\sum_{k=1}^{m_{\hat{k}-1}} \mathcal{A}_1(\mathbf{g}_{\hat{k}-1})_k (w_{k,j}^{\hat{k}})^2, \qquad \mathcal{A}_1(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})) = \texttt{softmax}(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})), \quad \hat{k} = 5,\ 6. \tag{5.53}$$

The $\bar{n}^{th}$ aggregating unit: The unit is described as follows:

$$\mathbf{g}_{3\bar{n}-2}(\mathbf{y};\nu_{3\bar{n}-2}) = [\bar{a}_{1,\bar{n}},\ \bar{a}_{2,\bar{n}},\ \cdots,\ \bar{a}_{m_{3\bar{n}-2},\bar{n}}]; \tag{5.54}$$

$$\bar{a}_{1,\bar{n}} = \bar{a}_{1,\bar{n}-1} + g_{1,3\bar{n}-2}, \quad \bar{a}_{2,\bar{n}} = \bar{a}_{2,\bar{n}-1} + g_{2,3\bar{n}-2}, \quad \cdots, \quad \bar{a}_{m_{3\bar{n}-2},\bar{n}} = \bar{a}_{m_{3(\bar{n}-1)-2},\bar{n}-1} + g_{m_{3\bar{n}-2},3\bar{n}-2}; \tag{5.55}$$

$$m_{3\bar{n}-2} = m_{3(\bar{n}-1)-2}; \tag{5.56}$$

$$g_{j,3\bar{n}-2} = -0.5\sum_{k=1}^{m_{3(\bar{n}-1)}} \mathcal{A}_1(\mathbf{g}_{3(\bar{n}-1)})_k (w_{k,j}^{3\bar{n}-2})^2, \quad \mathcal{A}_1(\mathbf{g}_{3\bar{n}-2}(\mathbf{y};\nu_{3\bar{n}-2})) = \texttt{softmax}(\mathbf{g}_{3\bar{n}-2}(\mathbf{y};\nu_{3\bar{n}-2})). \tag{5.57}$$

The $\bar{n}^{th}$ softmax block: We note that the $\bar{n}^{th}$ softmax block is the final softmax block before the multiplicative ensembled layer. The output of the $\bar{n}^{th}$ softmax block represents the feature vector in the multiplicative ensembled layer. The design of the $\bar{n}^{th}$ softmax block is as follows:

$$\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}}) = [g_{1,\hat{k}},\ g_{2,\hat{k}},\ \cdots,\ g_{m_{\hat{k}},\hat{k}}]; \tag{5.58}$$

$$g_{j,\hat{k}} = -0.5\sum_{k=1}^{m_{\hat{k}-1}} \mathcal{A}_1(\mathbf{g}_{\hat{k}-1})_k (w_{k,j}^{\hat{k}})^2, \quad \mathcal{A}_1(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})) = \texttt{softmax}(\mathbf{g}_{\hat{k}}(\mathbf{y};\nu_{\hat{k}})), \quad \hat{k} = 3\bar{n}-1,\ 3\bar{n}. \tag{5.59}$$

Here,

$$\mathcal{R}_j(\mathbf{y};\nu_{\mathcal{R}_j}) = \sum_{k=1}^{m_{\bar{m}+1}} \mathcal{A}_1(\mathbf{g}_{\bar{m}})_k w_{k,j}^{\bar{m}+1}, \qquad j = 1, 2, \cdots \bar{m}+1, \tag{5.60}$$

which represents the $\bar{m}^{th}$ hidden layer with $\bar{m} = 3\bar{n}$. We note that the $\bar{m}^{th}$ hidden layer is the final hidden layer in the first sub-architecture within LinNet.

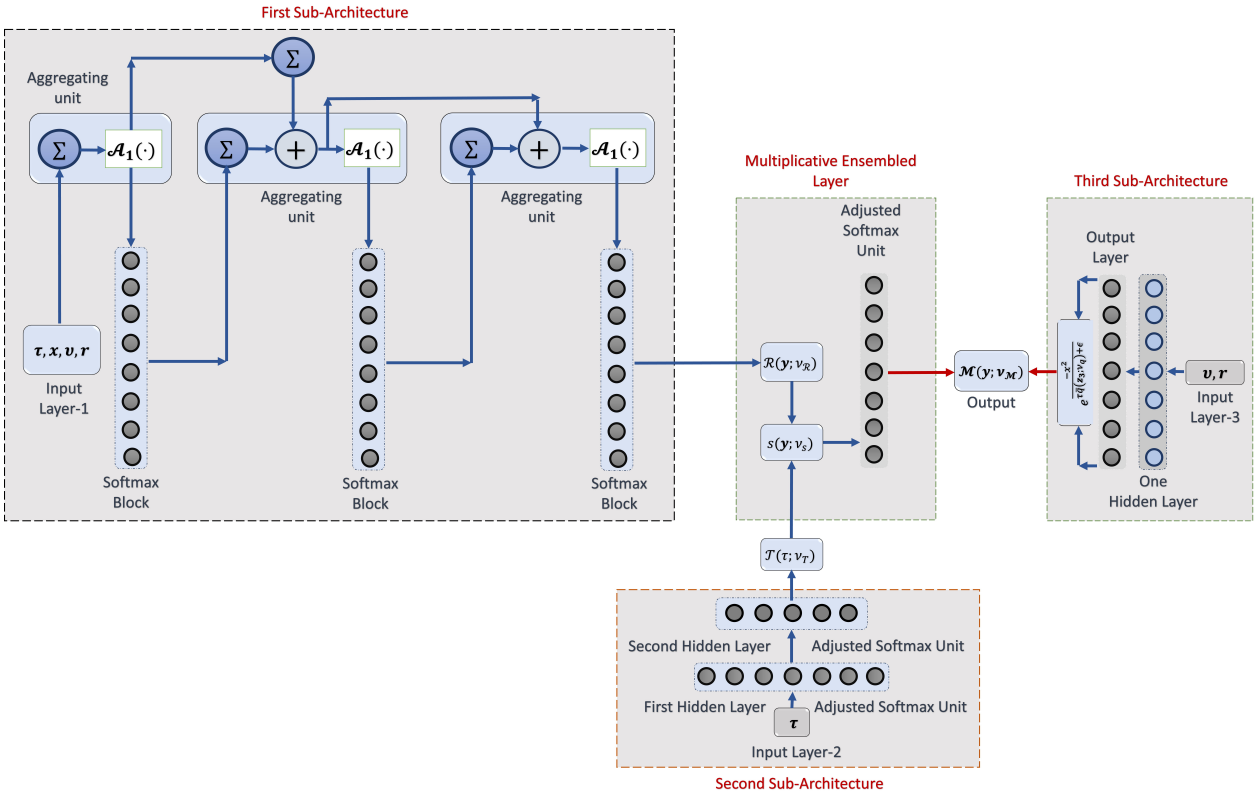


Figure 12: LinNet architecture. For brevity, only three aggregating units and softmax blocks are considered.

The LinNet architecture is shown in Figure 12. In Figure 12, the symbol "$\sum$" represents a linear unit (output of a given hidden layer with no activation function) as described in (5.44), (5.49), and (5.57). The symbol "+" represents addition of linear units as described in (5.48) and (5.55). The symbol "$\mathcal{A}_1(\cdot)$" represents the softmax unit described in (5.11). Furthermore, "$\mathcal{T}(\tau, \nu_{\mathcal{T}})$" is given in (5.27), "$\mathcal{R}(\mathbf{y}, \nu_{\mathcal{R}})$" in (5.60), and "$s(\mathbf{y}, \nu_s)$" in (5.14). The final output "$\mathcal{M}(\mathbf{y}, \nu_{\mathcal{M}})$" which represents the solution framework is described in (5.6).

5.8 Short note on the aggregation of the linear units for LinNet

Here, we elaborate further on the aggregation of linear units in the first sub-architecture within LinNet. One might ask, why aggregation of the linear units? As more linear units are combined, how does it impact on the output value (before been passed to the softmax unit) for a given aggregation unit. What if the output of the combined linear units grows? What are the chances that numerical instability might occur when this output is passed to a softmax unit in a given aggregation unit.

Before proceeding further, we mention that in this research, we did not consider any regularization method(s) in the form of lasso regression, ridge regression, combination of ridge and lasso regression, dropout, batch normalization. From observation, we understand that the implementation of softmax activation across hidden layers plays a very crucial role in ensuring the stability of LinNet design. We envisage that considering unbounded activation function, even a bounded activation function might escalate a growth of the combined linear units. To elaborate further, let us observe how these linear units are combined. The $\bar{n}^{th}$ aggregating unit is described as follows:

$$\mathbf{g}_{3\bar{n}-2}(\mathbf{y}; \nu_{3\bar{n}-2}) = [\bar{a}_{1,\bar{n}},\ \bar{a}_{2,\bar{n}},\ \cdots,\ \bar{a}_{m_{3\bar{n}-2},\bar{n}}]. \tag{5.61}$$

$$\bar{a}_{j,\bar{n}} = \bar{a}_{j,\bar{n}-1} + g_{j,3\bar{n}-2} = \bar{a}_{j,1} + \sum_{i=2}^{n} g_{j,3i-2}, \qquad j = 1, 2, \cdots m_{3\bar{n}-2}, \tag{5.62}$$

with

$$g_{j,3i-2} = -0.5 \sum_{k=1}^{m_{3(i-1)}} \mathcal{A}_1(\mathbf{g}_{3(i-1)})_k (w_{k,j}^{3i-2})^2, \tag{5.63}$$

and

$$\bar{a}_{j,1} = -0.5 \sum_{k=1}^{m_1} \mathcal{A}_1(\mathbf{g}_1)_k (\bar{w}_{k,j}^2)^2. \tag{5.64}$$

Because of the softmax activation unit $\mathcal{A}_1(\cdot)$, one can consider each sum $g_{j,3i-2}$ for a given j as classification on the given weights. For extreme classification, it is possible that most of the weight values will be deactivated by the zero or near zero values from entries of $\mathcal{A}_1(\cdot)$. Another interesting observation is that in a nice extreme scenario, high and very high weight values could be deactivated by the zero or near zero entries of $\mathcal{A}_1(\cdot)$ and weight(s) with the small value(s) could be retained for a given $g_{j,3i-2}$. The reverse could also be the case. However, in any of this scenarios, we envisage that this classification mechanism present in $g_{j,3i-2}$ could control growth of the combined linear units and ensure stability during training. Moreover, note that

$$\bar{a}_{j,1} \in \mathbb{R}_-, \qquad g_{j,3i-2} \in \mathbb{R}_-, \qquad \forall i, j. \tag{5.65}$$

Even though we square the weight values to ensure non-positivity which might reduce the numerical instability that arises due to the exponential in the softmax unit, we mention that other possibility exists. One can choose any function that enforces positivity on the weight. For instance, one can choose functions that have a linear or close to a linear growth such as:

$$f(x) = x\texttt{tan}(x), \qquad f(x) = \sqrt{x^2 + \acute{a}} - \sqrt{\acute{a}}, \qquad 0 < \acute{a} \leq 1. \tag{5.66}$$

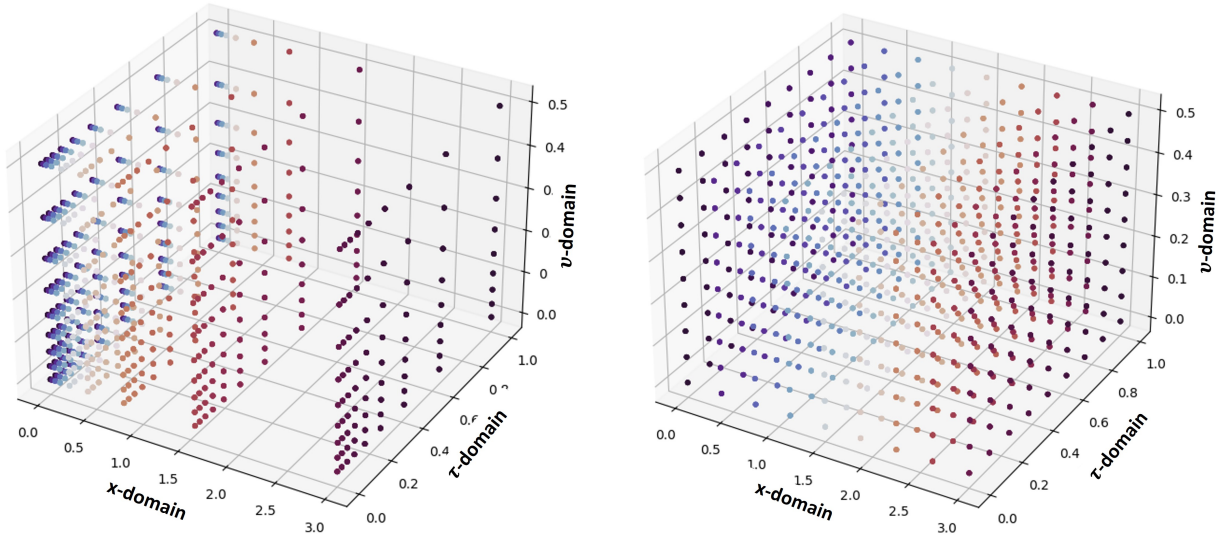
One could also choose a function that deactivates most of the negative weight values and forces the rest of the weights to be positive. Such functions include softplus [6] and squareplus [2] activation functions. We further mention that acting a nonlinear function to enforce non-negativity on the weights was considered in the work of Dugas et al. [6] where the authors applied a neural network method for solving call options.

6 Training: Sampling, Grid Refinement, and Optimization

In this section, we first present the sampling for the transformed asset price normalized time to maturity, variance, and interest rate. Our sampling strategy involves generating an equidistant sample points for the transformed asset price normalized time to maturity, variance, and interest rate with further implementation of grid stretching to concentrate most of these sampling points in the neighbourhood of the left corner point. Our grid stretching method is formulated with the sinh function as given below:

$$\tau_s = s_1 \sinh\left[\text{arcsinh}\left(\frac{\tau_{\max}}{s_1}\right)\frac{\tau}{\tau_{\max}}\right], \quad x_s = s_2 \sinh\left[\text{arcsinh}\left(\frac{x_{\max}}{s_2}\right)\frac{x}{x_{\max}}\right]; \tag{6.1}$$

$$\upsilon_s = s_3 \sinh\left[\text{arcsinh}\left(\frac{\upsilon_{\max}}{s_3}\right)\frac{\upsilon}{\upsilon_{\max}}\right], \qquad r_s = s_4 \sinh\left[\text{arcsinh}\left(\frac{r_{\max}}{s_4}\right)\frac{r}{r_{\max}}\right]. \tag{6.2}$$



(a) Stretched grid sampling in τ, x, and υ directions.

(b) Uniform grid sampling in τ, x, and υ directions.

Figure 13: Input sampling and sample strategies in 3D representation.

Here, the critical point is at $(\tau_{\min}, x_{\min}, \upsilon_{\min}, r_{\min}) = (0, 0, 0, 0)$. Figure 13 describes a 3D representation of the grid sampling based on uniform and uniformly stretched samplings.

For the loss function, we consider mean square error. Moreover, the global loss function is formulated to incorporate the Robin boundary term which couples the value matching and smooth pasting conditions and the PDE loss functions as given below:

$$\begin{aligned}\mathcal{L}_f(\nu_M) =& \frac{1}{N}\sum_{i=1}^{N}[\mathcal{L}_i\mathcal{M}(\tau_i, x_i, \upsilon_i, r_i; \nu_M)]^2 \\ &+ \frac{1}{N_b}\sum_{j=1}^{N_b}\left[\mathcal{M}(\tau_j, 0, \upsilon_j, r_j; \nu_M) - \frac{\partial\mathcal{M}(\tau_j, 0, \upsilon_j, r_j; \nu_M)}{\partial x} - 1\right]^2, \end{aligned} \tag{6.3}$$

where

$$\begin{aligned}\mathcal{L}_i =& \frac{\partial}{\partial\tau} + \beta_1(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial^2}{\partial x^2} + \beta_2(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial^2}{\partial r^2} + \beta_3(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial^2}{\partial\upsilon^2} + \beta_4(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial}{\partial x\partial r} \\ &+ \beta_5(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial}{\partial x\partial r} + \beta_6(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial}{\partial x\partial\upsilon} + \beta_7(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial}{\partial x} \\ &+ \beta_8(\tau_i, \upsilon_i, r_i; \nu_s)\frac{\partial}{\partial r} + r_i = 0, \end{aligned} \tag{6.4}$$

and the coefficients β_i are given in (2.6)-(2.10). The weights represented as ν_M are optimized by solving the minimization problem as given below:

$$arg\min_{\nu_M}\mathcal{L}_f(\nu_M) \qquad \text{subject to} \qquad \bar{s}_f(\tau_i, \upsilon_i, r_i; \nu_s) = 1 - \mathcal{M}(\tau_i, 0, \upsilon_i, r_i; \nu_M). \tag{6.5}$$

The designed three network models and their derivatives will predict the surfaces of the early exercise boundary, value function and the first and second order Greeks after training.

7 Numerical Experiment

In this experimental section, we carry out experiments to better understand the predictive performances of AddNet, SoftNet, and LinNet for solving the three-dimensional American Heston-CIR model with deeper hidden layers. The options parameters for the three-dimensional American Heston-CIR model are listed in Table 1.

Table 1: Options parameters for the 3D American Heston-CIR model.

Parameters	Values
K	90, 100, 110
S_0	100
T	$\frac{1}{12}$, $\frac{1}{4}$, $\frac{1}{2}$
$(\upsilon_{\min}, \upsilon_{\max})$, $(r_{\min}, r_{\max})$	(0.0, 0.5), (0.0, 0.2)
$(x_{\min}, x_{\max})$, $(\tau_{\min}, \tau_{\max})$	(0.0, 3.5), (0.0, 1.0)
σ_1, σ_2	0.20, 0.08
$\rho_{1,2}, \rho_{1,3}, \rho_{2,3}$	-0.26, -0.26 , 0
κ, θ	1.58, 0.03
b, a	0.26, 0.04

Our experiments were conducted on a computer with a 12th Gen Intel Core i7-12700H processor running at 2.30 GHz on a 64-bit Windows 11 operating system. Our codes were written in `Python` with `Tensorflow` [1]. For clarity, we describe the considered architectures as follows:

- SoftNet-i: SoftNet with i hidden layers (including the multiplicative ensembling layer)
- C-DNN-(s, i): Conventional DNN architecture for comparison with SoftNet.
- AddNet-(i, j): AddNet involving i hidden layers in the first sub-architecture, $j-1$ local exit layers, and $\frac{i}{j}$ softmax blocks.
- C-DNN-(a, i): Conventional DNN architecture for comparison with AddNet.
- LinNet-(i, j): LinNet with i hidden layers (including the multiplicative ensembling layer) and j^{th} options with respect to the design of the third sub-architecture.
- C-DNN-(l, i): Conventional DNN architecture for comparison with LinNet network model.

Across experiments, we hope to test the predictive performance of SoftNet, AddNet, and LinNet network models by comparing with existing results over different deeper structures. We hope to further validate the importance of the three network models in mitigating vanishing gradient effect over C-DNN-(s, i), C-DNN-(a, i), and C-DNN-(l, i).

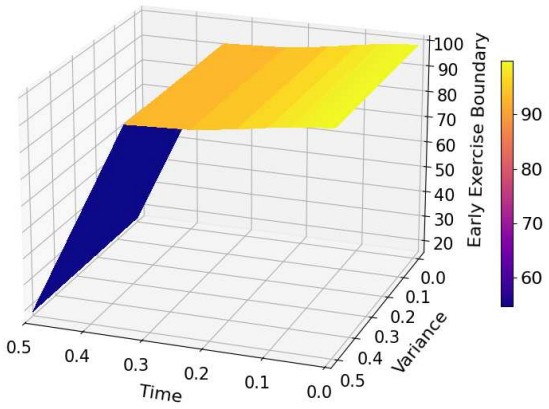
7.1 Limitation of C-DNNs

For brevity, we represent the three conventional DNN architectures as

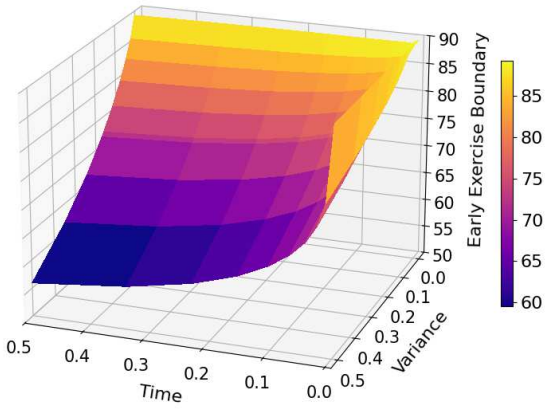
$$\text{C-DNN} - (m, i), \qquad m \in [a, s, l].$$

Here, we show the limitation of C-DNN-$(m, 6)$ by considering C-DNN-$(m, 6)$ and C-DNN-$(m, 13)$ involving six and thirteen hidden layers, respectively, as described in Figure 14 and Tables 2. For the predicted results in Tables 2, we observe good predictive performance with C-DNN-$(m, 6)$ and poor predictive performance with C-DNN-$(m, 13)$. We envisage that this poor performance with C-DNN-$(m, 13)$ could be attributed to vanishing gradient effect. Moreover, the predicted plots of the early exercise boundary with C-DNN-$(m, 13)$ are poor. However, the predicted results and plots of the early exercise boundary with AddNet-(12,6), SoftNet-13, and LinNet-(13,1) are reasonably consistent with the one obtained from C-DNN-$(m, 6)$ (with good predictive performance). Hence, we can affirm

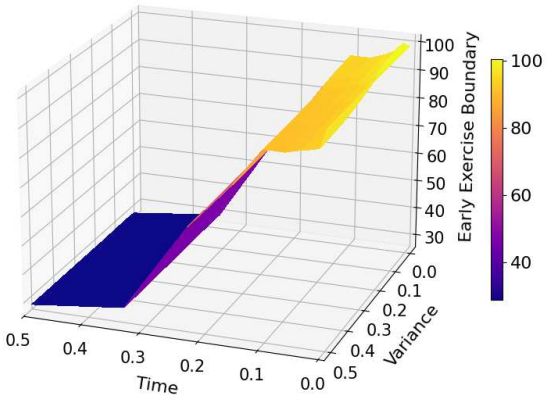
that AddNet, SoftNet, and LinNet present better predictive performance than C-DNN-(m, i) beyond twelve hidden layers in the first sub-architecture. When a more complex task is given, the indication is that C-DNN-$(m, 13)$ with deeper structure may not be suitable.



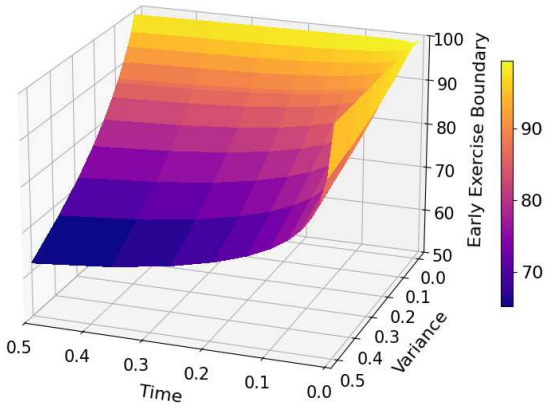
(a) Early exercise boundary with C-DNN-$(a, 13)$.



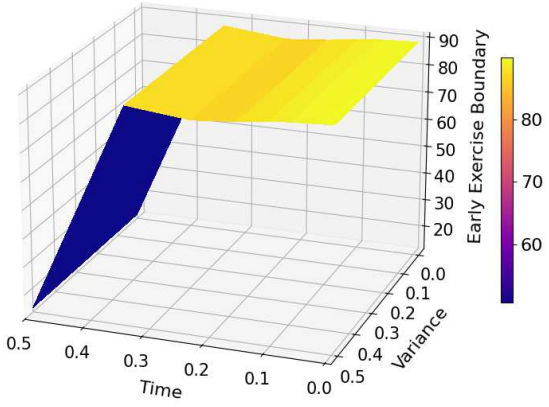
(b) Early exercise boundary with AddNet-(12,3).



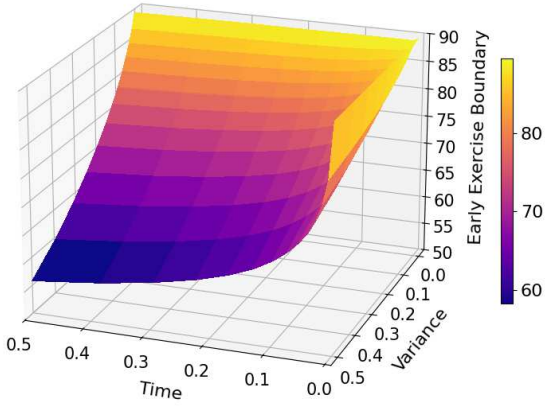
(c) Early exercise boundary with C-DNN-$(s, 13)$.



(d) Early exercise boundary with SoftNet-13.



(e) Early exercise boundary with C-DNN-$(l, 13)$.



(f) Early exercise boundary with LinNet-(13,1).

Figure 14: Early exercise boundary with different DNN architectures and depths.

Table 2: Solutions of the value function and the early exercise boundary with the DNN architecture involving no feature reuse. Here $T = \frac{1}{2}$, $\upsilon_0 = 0.04$, and $r_0 = 0.04$ are considered.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
C-DNN-$(a, 6)$			
Value function	1.407	4.590	10.940
Early exercise boundary	74.976	83.307	91.638
C-DNN-$(s, 6)$			
Value function	1.313	4.570	10.948
Early exercise boundary	74.909	83.232	91.556
C-DNN-$(l, 6)$			
Value function	1.366	4.609	10.969
Early exercise boundary	74.994	83.327	91.659
C-DNN-$(a, 13)$			
Value function	9.673	12.275	15.162
Early exercise boundary	15.238	16.931	18.624
C-DNN-$(s, 13)$			
Value function	9.584	12.971	16.852
Early exercise boundary	62.612	69.569	76.526
C-DNN-$(l, 13)$			
Value function	4.341	6.039	8.099
Early exercise boundary	12.985	14.428	15.871
SoftNet-13			
Value function	1.338	4.609	10.970
Early exercise boundary	74.883	83.203	91.523
AddNet-(12,3)			
Value function	1.363	4.624	10.963
Early exercise boundary	74.875	83.195	91.514
LinNet-(13,1)			
Value function	1.352	4.574	10.927
Early exercise boundary	75.116	83.463	91.809

7.2 Numerical experiment for AddNet

In this subsection, we conduct experiments to test and better understand the predictive performance of AddNet for solving the three-dimensional American Heston-CIR model. Across experiments, we hope to validate the predictive performance of AddNet with different number of hidden layers and number of local exit layers by comparing the predicted results with those obtained from existing methods.

7.2.1 Experiment with small number of local exit layers

In this subsection, we consider an experiment involving few local exit layers for AddNet. Here, we consider a maximum of twelve hidden layers in the first sub-architecture with fixed two-hidden layers in the second sub-architecture and one hidden layer in the third sub-architecture. In the previous experiment, we showed that C-DNN-$(m, 13)$ fails to present good predictive performance. In Table 3, we present the hyperparameters for AddNet. In Tables 4-5, we present the predicted results of the value function and the early exercise boundary obtained from AddNet-(6,2), AddNet-(9,3), and AddNet-(12,3) which involve six, nine, and twelve hidden layers in the first sub-architecture, respectively. Our observation is a reasonable predictive performance when compared with existing results.

Table 3: Hyperparameters for AddNet.

Hyperparameters	Values
Training steps/Learning rate decay	12000/ Yes (exponential)
Optimizer, β_1, β_2, ϵ_{adam}	Adam, 0.994, 0.999, 10^{-8}
$(s_1,\ s_2,\ s_3,\ s_4)$ for AddNet	(0.03, 0.02, 0.03, 0.03)
Starter learning rate, Decay step, Decay rate	0.1, 2000, 0.80
Ensemble of single mixture of solutions	Arithmetic mean
AddNet-(6,2)	
Softmax block/Number of local exit layers	3/1 hidden layers
AddNet-(9,3)	
Softmax block/Number of local exit layers	3/2 hidden layers
AddNet-(12,3)	
Softmax block/Number of local exit layers	4/2 hidden layers

Table 4: Solutions of the value function and the early exercise boundary with AddNet, 12000 training steps, and $(10\times10\times10\times10)$ stretched dataset. Here, $T=\frac{1}{2}$, $\upsilon_0=0.04$, and $r_0=0.04$ are considered.

Methods	$K=90$	$K=100$	$K=110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
AddNet-(6,2)			
Value function	1.328	4.589	10.967
Early exercise boundary	74.854	83.171	91.488
AddNet-(9,3)			
Value function	1.381	4.602	10.953
Early exercise boundary	74.946	83.273	91.600
AddNet-(12,3)			
Value function	1.363	4.624	10.963
Early exercise boundary	74.875	83.195	91.514

Table 5: Solutions of the value function and the early exercise boundary with AddNet, 12000 training steps, and $(10\times10\times10\times10)$ stretched dataset. Here, $T=\frac{1}{2}$ and $r_0=0.04$ are considered.

	$\upsilon=0.09$			$\upsilon=0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
AddNet-(6,2)						
Value function	2.842	6.693	12.776	4.611	8.870	14.816
Early exercise boundary	67.247	74.719	82.191	60.136	66.818	73.499
AddNet-(9,3)						
Value function	2.844	6.693	12.758	4.610	8.883	14.836
Early exercise boundary	67.311	74.790	82.269	60.015	66.683	73.351
AddNet-(12,3)						
Value function	2.869	6.738	12.792	4.644	8.944	14.902
Early exercise boundary	67.658	75.176	82.694	60.412	67.124	73.836

7.2.2 Effect of varying the number of mixtures m in the multi-mixture of DNN solutions for AddNet

The deep learning solution framework for AddNet is given as follows:

$$\mathcal{M}(\mathbf{y};\nu_M)=\sum_{j=1}^{n}\hat{\pi}_j\mathcal{M}_j(\mathbf{y};\nu_{M,j})=\sum_{j=1}^{n}\hat{\pi}_j\sum_{i=1}^{m}g_{i|j,o}(\mathbf{z}_1;\nu_{j|o})e^{-\frac{x^2}{\tau\bar{q}_i(\mathbf{z}_3;\nu_q)+\epsilon}}, \tag{7.1}$$

$$\sum_{j=1}^{n}\hat{\pi}_j=1, \qquad \mathbf{y}=(\tau,x,\upsilon,r), \qquad \mathbf{z}_3=(\upsilon,r) \tag{7.2}$$

Table 6: Solutions of the value function and the early exercise boundary with AddNet-(8,2), varying m parameter, and 12000 training steps. Here, $T=\frac{1}{2}$, $\upsilon_0=0.04$, and $r_0=0.04$ are considered.

Methods	$K=90$	$K=100$	$K=110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$(10\times10\times10\times10)$ stretched dataset			
$m=6$			
Value function	1.359	4.580	10.938
Early exercise boundary	75.089	83.432	91.775
$m=12$			
Value function	1.327	4.594	10.953
Early exercise boundary	74.931	83.257	91.583
$m=18$			
Value function	1.369	4.653	10.987
Early exercise boundary	74.935	83.261	91.587

Table 7: Solutions of the value function and the early exercise boundary with AddNet-(8,2), varying m parameter, and 12000 training steps. Here, $T=\frac{1}{2}$ and $r_0=0.04$.

	$\upsilon=0.09$			$\upsilon=0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$(10\times10\times10\times10)$ stretched dataset						
$m=6$						
Value function	2.885	6.751	12.813	4.624	8.916	14.888
Early exercise boundary	67.118	74.576	82.033	60.117	66.796	73.476
$m=12$						
Value function	2.835	6.700	12.762	4.583	8.863	14.831
Early exercise boundary	67.367	74.853	82.338	60.264	66.960	73.656
$m=18$						
Value function	2.885	6.738	12.784	4.638	8.918	14.885
Early exercise boundary	67.516	75.017	82.519	60.119	66.799	73.479

The parameter n represents the number of single mixtures of deep learning solution based on the local exit and final output layer. The parameter m represents the number of mixtures for each given single mixture of deep learning solution. Here, we want to investigate the effect of varying the parameter m

on the predictive performance of AddNet. We consider AddNet-(8,2) and present a solution profiles of the value function and the early exercise boundary. From the predicted results in Tables 6 and 7, $m = 12$ generalizes well across varying υ which is the number of mixtures we used across other experiments.

7.2.3 Experimenting with high number of local exit layers and adaptive weighting

In this subsection, we consider an experiment with five local exit layers in AddNet. In the previous subsection, we considered maximum of two local exit layers in AddNet and limited our experiment only to twelve hidden layers in the first sub-architecture. Here, we carry out investigation with high number of local exit layers, resulting in AddNet version with deeper hidden layers in the first sub-architecture as described in Table 8.

Table 8: Hyperparameters for AddNet.

Hyperparameters	Values
Training steps	12000
Learning rate decay	Yes (exponential)
Optimizer, β_1, β_2, ϵ_{adam}	Adam, 0.994, 0.999, 10^{-8}
Stretching	Yes
$(s_1,\ s_2,\ s_3,\ s_4)$	(0.03, 0.015, 0.03, 0.03)
Starter learning rate	0.1
Decay step, Decay rate	2000, 0.80
Ensemble of single mixture of solutions	Adaptive
AddNet-(12,6)	
Softmax block	2 hidden layers
Number of local exit layers	5
AddNet-(18,6)	
Softmax block	3 hidden layers
Number of local exit layers	5
AddNet-(24,6)	
Softmax block	4 hidden layers
Number of local exit layers	5

As presented in (4.2) and (7.2), we consider

$$\sum_{j=1}^{n} \hat{\pi}_j = 1, \tag{7.3}$$

where the parameter n represents the number of single mixtures of deep learning solution based on the local exit and final output layers. We note that the output of the j^{th} local exit layer is given the j^{th} weight value $\hat{\pi}_j$ while the output of the final hidden layer is given the last weight value $\hat{\pi}_n$. In one of the previous experiment where we implemented small number of local exit layers, we considered averaging as the weighting strategy with

$$\hat{\pi}_j = \frac{1}{n}, \qquad \forall j. \tag{7.4}$$

Because we train with deeper hidden layers in this subsection due to high number of local exist layers, we seek an adaptive alternative in terms of weighting strategy with the hope of enhancing predictive performance.

We have already demonstrated in the first experiment for this section that the conventional DNN architecture without early exits has poor predictive performance with twelve hidden layers in the first

sub-architecture and we envisage that it could be as a result of vanishing gradient effect. Here, our weighting strategy is to decrease the weight across the outputs obtained from local exit and final hidden layers in an increasing order. For some $0 < \varpi < B \in \mathbb{R}_+$, we design the weighting function as

$$\hat{\pi}_j = \frac{e^{\frac{-j}{0.5+\varpi}}}{\sum_{i=1}^{n} e^{\frac{-i}{0.5+\varpi}}} \tag{7.5}$$

such that

$$i < j \Rightarrow \hat{\pi}_i < \hat{\pi}_j, \qquad \sum_{j=1}^{n} \hat{\pi}_j = 1. \tag{7.6}$$

We note that ϖ is a single trainable weight. The reason for this weighting strategy is that we assume that the predictive performance could deteriorate as the number of hidden layers increases due to vanishing gradient effect. Hence, we hope to attach more importance to the shallower local exits while reserving the contribution of the output from the deeper local exit and final hidden layer. We also mention that if $\varpi \to \infty$,

$$\hat{\pi}_j \to \frac{1}{n}, \qquad \forall j, \tag{7.7}$$

which resembles the considered averaging in the previous experiment. Moreover for $0 < \varpi < B \in \mathbb{R}_+$, we obtain some sort of weightings that attach more importance to the shallower local exit layers.

In this experiment, we consider up to 24 hidden layers in the first sub-architecture as described in Table 8. We list the predicted results in Tables 9 and 10. We observe a reasonable prediction accuracy with AddNet-(12,6), AddNet-(18,6), and AddNet-(24,6) involving twelve, eighteen, and twenty-four hidden layers in the first sub-architecture, respectively. Moreover, we track the values of ϖ for AddNet-(12,6), AddNet-(18,6), and AddNet-(24,6) and list the weights attached to each output from local exits and final hidden layers below.

For AddNet-(12,6), $\varpi = 2.2702$ and

$$\hat{\pi}_1 = 0.3423, \qquad \hat{\pi}_2 = 0.2385, \qquad \hat{\pi}_3 = 0.1663; \tag{7.8}$$

$$\hat{\pi}_4 = 0.1159, \qquad \hat{\pi}_5 = 0.0808, \qquad \hat{\pi}_6 = 0.0563. \tag{7.9}$$

For AddNet-(18,6), $\varpi = 0.3572$ and

$$\hat{\pi}_1 = 0.6892, \qquad \hat{\pi}_2 = 0.2146, \qquad \hat{\pi}_3 = 0.0669; \tag{7.10}$$

$$\hat{\pi}_4 = 0.0208, \qquad \hat{\pi}_5 = 0.0065, \qquad \hat{\pi}_6 = 0.0020 \tag{7.11}$$

For AddNet-(24,6), $\varpi = 0.1006$ and

$$\hat{\pi}_1 = 0.8110, \qquad \hat{\pi}_2 = 0.1534, \qquad \hat{\pi}_3 = 0.0290; \tag{7.12}$$

$$\hat{\pi}_4 = 0.0055, \qquad \hat{\pi}_5 = 0.0010, \qquad \hat{\pi}_6 = 0.0002 \tag{7.13}$$

The observation is that ϖ gets smaller as the number of hidden layers increases for a fixed number of softmax blocks and local exit layers. Moreover, it is observable that as the number of hidden layers in softmax block increases, more importance is attached to the single mixture of solution from the first local exit layer and less importance is attached to the solution from the final hidden layers. For further illustration, when the solution from the first local exit layer is obtained from $\bar{a} = 2$, i.e., from AddNet-(12,6), such solution is not obtained from a very deep network (can be seen as an architecture with three hidden layers including the multiplicative ensembling layer) and might not be able to extract complex features. Hence, the system attached smaller weight ($\hat{\pi}_1 = 0.3423$) to the single-mixture of solution obtained from the first local exit layer. However, when $\bar{a} = 4$, i.e., from AddNet-(24,6), the solution from the first local exit layer is obtained from a reasonably deep neural network and it can be observed that the network attached a reasonable importance ($\hat{\pi}_1 = 0.8110$) to such solution.

Table 9: Solutions of the value function and the early exercise boundary with AddNet and 12000 training steps. Here, $T = \frac{1}{2}$, $\upsilon_0 = 0.04$, and $r_0 = 0.04$ are considered.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$(10 \times 10 \times 10 \times 10)$ stretched dataset			
AddNet-(12,6)			
Value function	1.350	4.602	10.968
Early exercise boundary	74.962	83.291	91.620
$(12 \times 12 \times 12 \times 12)$ stretched dataset			
AddNet-(18,6)			
Value function	1.317	4.537	10.914
Early exercise boundary	75.124	83.471	91.818
AddNet-(24,6)			
Value function	1.359	4.593	10.933
Early exercise boundary	75.138	83.486	91.835

Table 10: Solutions of the value function and the early exercise boundary with AddNet and 12000 training steps. Here, $T = \frac{1}{2}$ and $r_0 = 0.04$.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$(10 \times 10 \times 10 \times 10)$ stretched dataset						
AddNet-(12,6)						
Value function	2.863	6.725	12.789	4.581	8.900	14.880
Early exercise boundary	67.466	74.963	82.459	59.975	66.639	73.303
$(12 \times 12 \times 12 \times 12)$ stretched dataset						
AddNet-(18,6)						
Value function	2.781	6.609	12.688	4.582	8.860	14.841
Early exercise boundary	67.710	75.234	82.757	60.139	66.822	73.504
AddNet-(24,6)						
Value function	2.901	6.732	12.777	4.638	8.916	14.879
Early exercise boundary	67.609	75.121	82.633	60.154	66.838	73.522

Moreover, it is understandable that as the number of hidden layers increases, less importance might be attached to the solution obtained from the final hidden layer. However, it is interesting to observe that though the conventional DNN architecture has very poor predictive performance with twelve hidden layers, AddNet-(12,6) presents a reasonable predictive performance and still attaches reasonable weight $(\hat{\pi}_n)$ to the solution from the final hidden layer.

7.3 Numerical experiment for SoftNet and M-SoftNet

In this section, we conduct some experiments to test the predictive performance of SoftNet for solving three-dimensional American Heston-CIR model with deeper hidden layers. In the following subsec-

tions, we hope to test the predictive performance of SoftNet by comparing with existing methods.

7.3.1 Comparing predictions

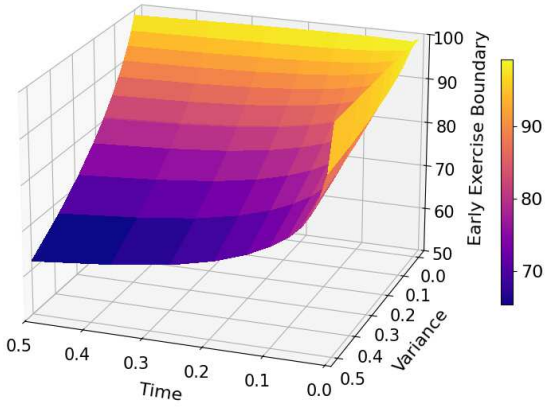
In this subsection, we compare the predictive performances of SoftNet and M-SoftNet-10 with existing results. For SoftNet, we consider SoftNet-13 and SoftNet-25. The hyperparameters for M-SoftNet-10, SoftNet-13, and SoftNet-25 are listed in Table 11.

Table 11: Hyperparameters for SoftNet and M-SoftNet-10.

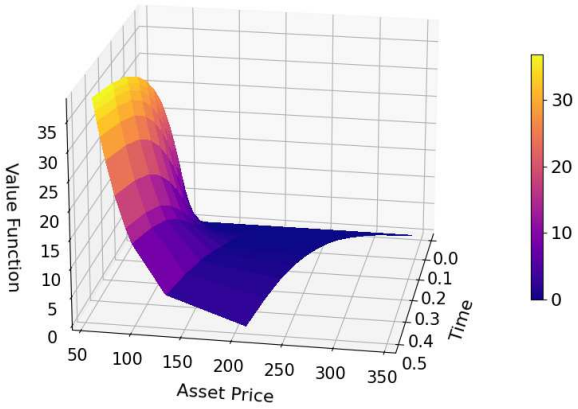
Hyperparameters	Values
Training steps	12000
Learning rate decay	Yes (exponential)
Optimizer, β_1, β_2, ϵ_{adam}	Adam, 0.994, 0.999, 10^{-8}
Stretching	Yes
$(s_1,\ s_2,\ s_3,\ s_4)$	(0.04, 0.02, 0.04, 0.04)
M-SoftNet-10	
Starter learning rate	0.04
Decay step, Decay rate	2500, 0.85
Softmax block	3 hidden layers
Number of softmax blocks	(1,2,3)
Units: Softmax blocks	[(32,16,8); (32,16,8),(32,16,8); (32,16,8),(32,16,8),(32,16,8)]
Units: Multiplicative ensembled layer	24
SoftNet-13 (see Figure 14)	
Starter learning rate	0.04
Decay step, Decay rate	2000, 0.80
Softmax block	2 hidden layers
Number of softmax block	6
Units: Softmax blocks	[(48,32), (24,16), (16,12), (12,12), (8,8), (24,12)]
Units: Multiplicative ensembled layer	8
SoftNet-25 (see Figures 15-18)	
Starter learning rate	0.03
Decay step, Decay rate	2000, 0.70
Softmax block	2 hidden layers
Number of softmax block	12
Units: Softmax blocks	[(48,32), (24,16), (12,12), (12,12), (8,8), (8,8). (8,8), (8,6), (6, 6), (6,6), (4,4), (16,8)]
Units: Multiplicative ensembled layer	12

It is interesting to observe from Tables 12-13 that even with small stretched training dataset, predicted results from M-SoftNet-10 and SoftNet-25 when compared with the one from the existing methods are very reasonable. Though not presented, we observed that finding an optimal learning rate as the number of hidden layers increases helps to stabilize minimization of the loss function and improve predictive performance. Too small or too high learning rate proves to be unhelpful. That is why the learning rate varies in some of the considered SoftNet and M-SoftNet-10 as presented in Table 11.

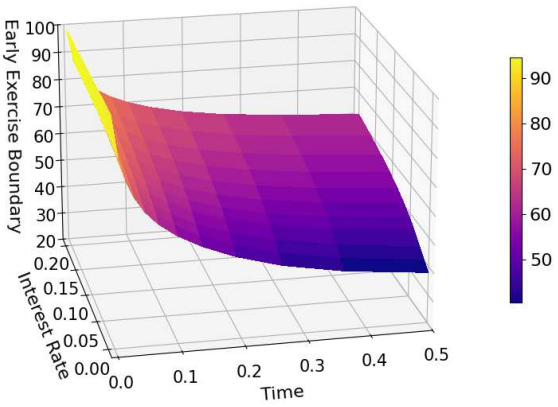
Furthermore, we present the surface plot profiles of the early exercise boundary, the value function, and some of the first and second-order Greeks by slicing their predictions. The predictions of the surfaces are done with SoftNet-25 and the plot profiles are displayed in Figures 15-18. It is interesting to see that the plots obtained from SoftNet-25 are reasonably smooth. The value function vanishes as we move out of the money, and the Delta sensitivity is reasonably in the interval of [-1,0]. Moreover, Gamma and Vega are reasonably in the positive domain and Rho is reasonably in the negative domain (for put options).



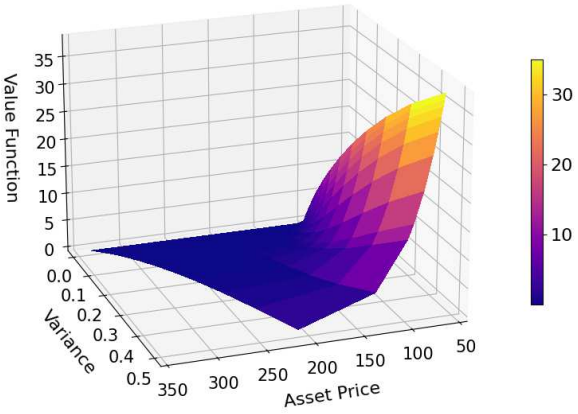
(a) Early exercise boundary for a fixed interest rate.



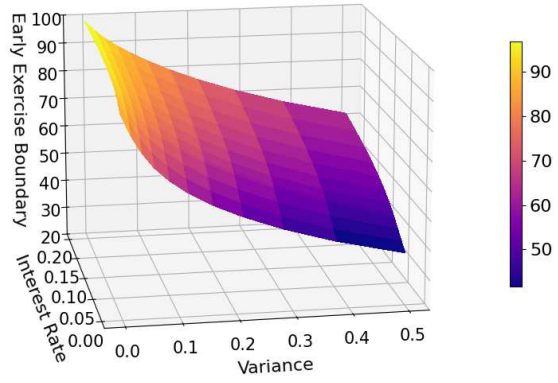
(b) Value function for fixed variance and interest rate.



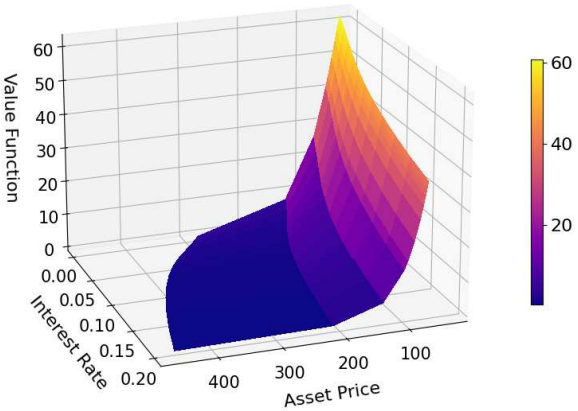
(c) Early exercise boundary for a fixed variance.



(d) Value function for fixed time and interest rate.

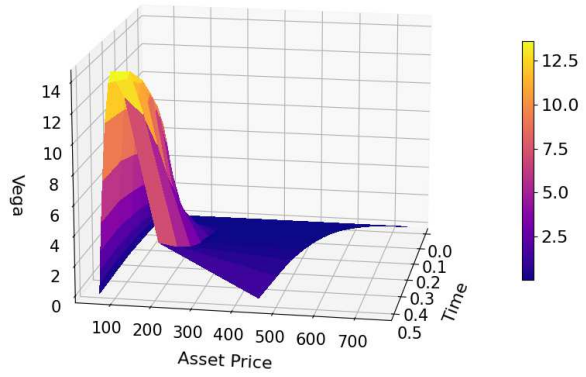


(e) Early exercise boundary for a fixed time.

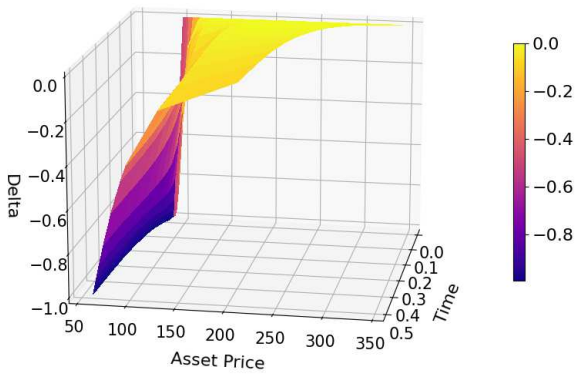


(f) Value function for a fixed time and variance.

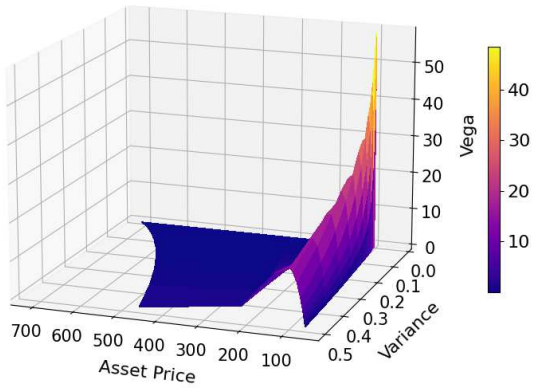
Figure 15: Early exercise boundary and value function surfaces for the Heston-CIR model with SoftNet-25, $(12 \times 12 \times 12 \times 12)$ stretched dataset ($s_1 = 0.04$, $s_2 = 0.02$, $s_3 = 0.04$, $s_4 = 0.04$), and $T = \frac{1}{2}$.



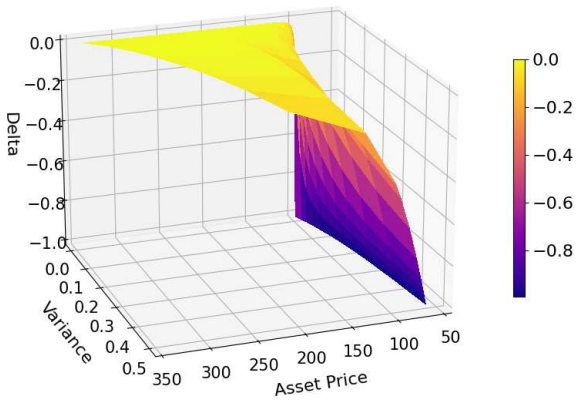
(a) Vega surface for fixed variance and interest rate.



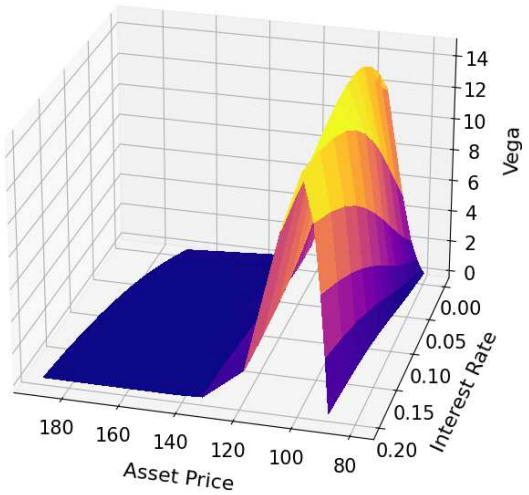
(b) Delta surface for fixed variance and interest rate.



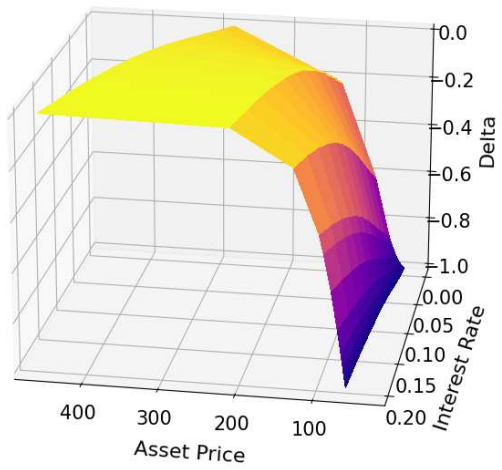
(c) Vega surface for fixed time and interest rate.



(d) Delta surface for fixed time and interest rate.

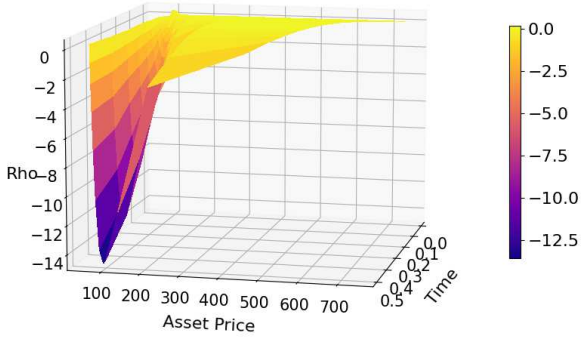


(e) Vega surface for fixed time and interest rate.

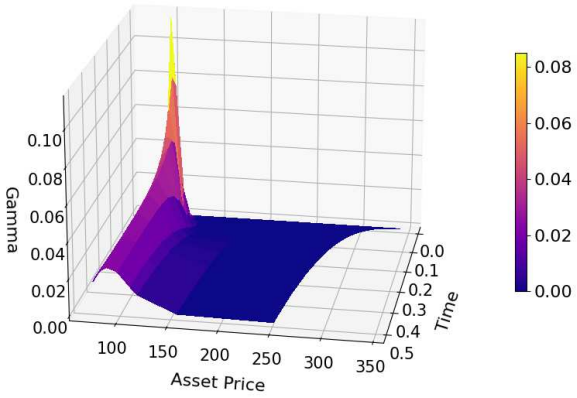


(f) Delta surface for a fixed time and variance.

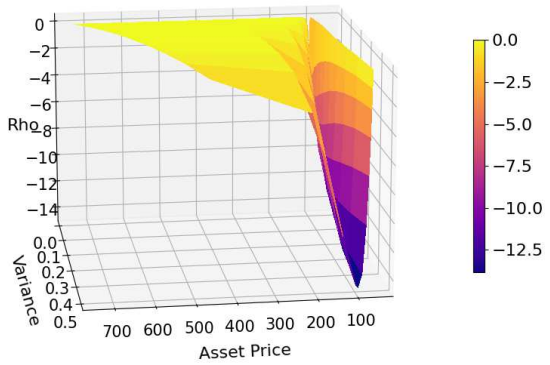
Figure 16: Vega and Delta for the Heston-CIR model with SoftNet-25, $(12 \times 12 \times 12 \times 12)$ stretched dataset $(s_1 = 0.04,\ s_2 = 0.02,\ s_3 = 0.04,\ s_4 = 0.04)$, and $T = \frac{1}{2}$.



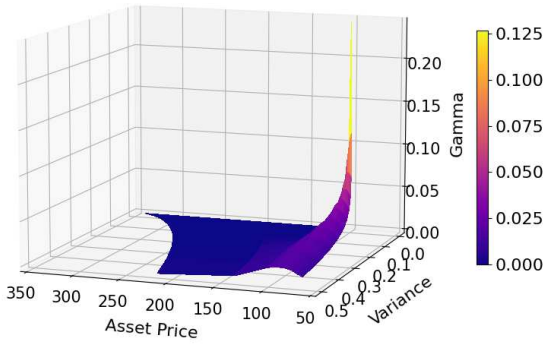
(a) Rho surface for fixed variance and interest rate.



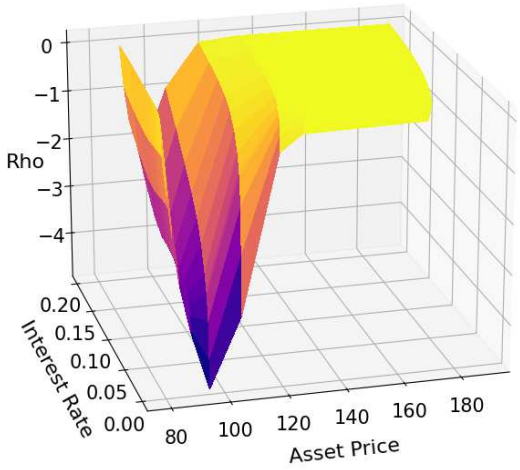
(b) Gamma surface for fixed variance and interest rate.



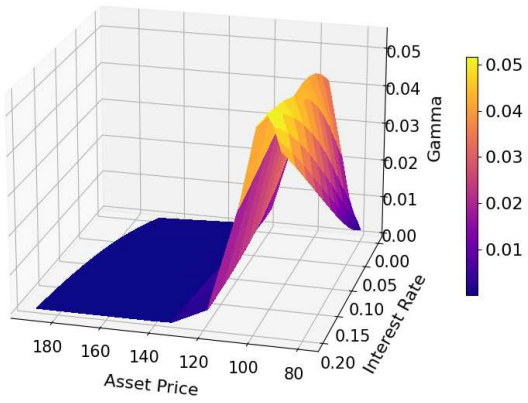
(c) Rho surface for fixed variance and interest rate.



(d) Gamma surface for fixed time and interest rate.

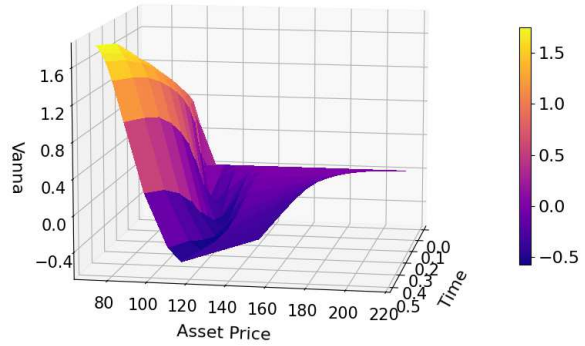


(e) Rho surface for fixed variance and interest rate.

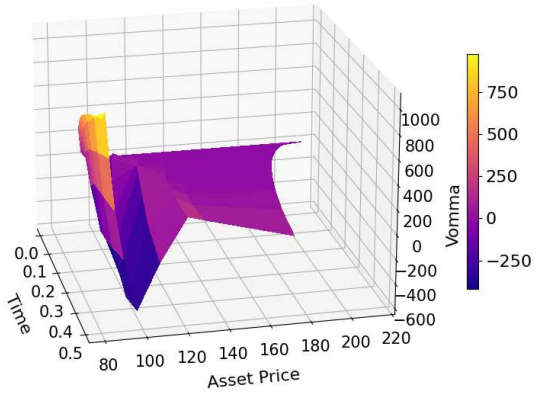


(f) Gamma surface for a fixed time and variance.

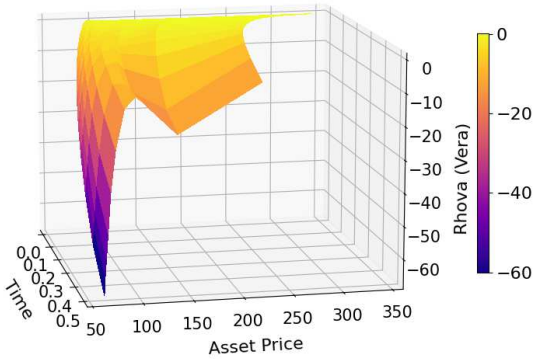
Figure 17: Rho and Gamma for the Heston-CIR model with SoftNet-25, $(12 \times 12 \times 12 \times 12)$ stretched dataset $(s_1 = 0.04,\ s_2 = 0.02,\ s_3 = 0.04,\ s_4 = 0.04)$, and $T = \frac{1}{2}$.



(a) Vanna for a fixed variance and interest rate.



(b) Vomma for a fixed variance and interest rate.



(c) Rhova (Vera) for a fixed variance and interest rate.

Figure 18: Vanna, Vomma, and Rhova for the Heston-CIR model with SoftNet-25, $(12 \times 12 \times 12 \times 12)$ stretched dataset $(s_1 = 0.04,\ s_2 = 0.02,\ s_3 = 0.04,\ s_4 = 0.04)$, and $T = \frac{1}{2}$.

Table 12: Solutions of the value function and the early exercise boundary with 12000 training steps. Here, $T = \frac{1}{2}$, $v_0 = 0.04$, and $r_0 = 0.04$ are considered.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and LevendorskiĬ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
M-SoftNet-10			
Value function	1.365	4.634	10.979
Early exercise boundary	75.069	83.441	91.784
SoftNet-25			
Value function	1.343	4.598	10.966
Early exercise boundary	74.809	83.121	91.433

Remark. *In Tables 12-13, we trained the model three different times and average predictions after each training.*

Table 13: Solutions of the value function and the early exercise boundary with $(10 \times 10 \times 10 \times 10)$ stretched dataset, and 12000 training steps. Here, $T = \frac{1}{2}$ and $r_0 = 0.04$ are considered.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
M-SoftNet-10						
Value function	2.859	6.737	12.795	4.606	8.903	14.859
Early exercise boundary	67.581	75.080	82.599	60.286	66.965	73.661
SoftNet-25						
Value function	2.835	6.699	12.766	4.548	8.845	14.821
Early exercise boundary	67.507	75.008	82.509	60.489	67.211	73.931

7.3.2 Increasing training dataset for SoftNet-25 and M-SoftNet-10

In this subsection, we consider further experiment to investigate the impact of increasing training dataset on the predictive performance of SoftNet-25 and M-SoftNet-10. We consider the hyperparameters in Table 11 and present the predicted results with varying training datasets in Tables 14-17.

Table 14: Solutions of the value function and the early exercise boundary with SoftNet-25 and 12000 training steps. Here, $T = \frac{1}{2}$, $\upsilon_0 = 0.04$, and $r_0 = 0.04$ are considered.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$(12 \times 12 \times 12 \times 12)$ stretched dataset **Average** (see Table 12)			
Value function	1.343	4.598	10.966
Early exercise boundary	74.809	83.121	91.433
$(14 \times 14 \times 14 \times 14)$ stretched dataset			
Value function	1.337	4.592	10.973
Early exercise boundary	74.831	83.146	91.460

Table 15: Solutions of the value function and the early exercise boundary with SoftNet-25 and 12000 training steps. Here, $T = \frac{1}{2}$, $\upsilon_0 = 0.04$, and $r_0 = 0.04$ are considered.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$(12 \times 12 \times 12 \times 12)$ stretched dataset **Average** (see Table 13)						
Value function	2.835	6.699	12.766	4.548	8.845	14.821
Early exercise boundary	67.507	75.008	82.509	60.489	67.211	73.931
$(14 \times 14 \times 14 \times 14)$ stretched dataset						
Value function	2.838	6.699	12.764	4.612	8.889	14.851
Early exercise boundary	67.735	75.261	82.787	60.708	67.453	74.199

Table 16: Solutions of the value function and the early exercise boundary with M-SoftNet-10 and 12000 training steps. Here, $T = \frac{1}{2}$ and $r_0 = 0.04$.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$(10 \times 10 \times 10 \times 10)$ stretched dataset **Average** (see Table 12)			
Value function	1.365	4.634	10.979
Early exercise boundary	75.069	83.441	91.784
$(12 \times 12 \times 12 \times 12)$ stretched dataset			
Value function	1.396	4.638	10.954
Early exercise boundary	74.930	83.256	91.581
$(14 \times 14 \times 14 \times 14)$ stretched dataset			
Value function	1.342	4.587	10.972
Early exercise boundary	74.911	83.235	91.558

Table 17: Solutions of the value function and the early exercise boundary with M-SoftNet-10 and 12000 training steps. Here, $T = \frac{1}{2}$ and $r_0 = 0.04$ are considered.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$(10 \times 10 \times 10 \times 10)$ stretched dataset **Average** (see Table 13)						
Value function	2.859	6.737	12.795	4.606	8.903	14.859
Early exercise boundary	67.581	75.080	82.599	60.286	66.965	73.661
$(12 \times 12 \times 12 \times 12)$ stretched dataset						
Value function	2.779	6.633	12.717	4.591	8.872	14.845
Early exercise boundary	67.589	75.099	82.609	60.011	66.679	73.347
$(14 \times 14 \times 14 \times 14)$ stretched dataset						
Value function	2.851	6.691	12.758	4.564	8.823	14.809
Early exercise boundary	67.524	75.027	82.530	60.462	67.180	73.898

For SoftNet-25, we observe from Tables 10 and 15 that there is not much changes between the average of predictions obtained with $(12 \times 12 \times 12 \times 12)$ stretched dataset when compared with the single prediction obtained from $(14 \times 14 \times 14 \times 14)$ stretched dataset. For M-SoftNet-10, we trained with $(10 \times 10 \times 10 \times 10)$, $(12 \times 12 \times 12 \times 12)$, and $(14 \times 14 \times 14 \times 14)$ stretched datasets. From Tables 16 and 17, one can observe that the predicted results from M-SoftNet-10 improve with $(14 \times 14 \times 14 \times 14)$ stretched dataset when compared with ones obtained with smaller training dataset. It signifies that there is a tendency that more training samples might improve performance.

7.3.3 SoftNet: More on depth

Here, we make some comments regarding the depth of the first sub-architecture for SoftNet. In the previous experiment, we considered only SoftNet-25 with 24 hidden layer in the first sub-architecture. Here, we want to emphasis that SoftNet with deeper hidden layer in the first sub-architecture could still be considered. We show this by considering SoftNet with 34 hidden layer in the first sub-architecture.

It is observable that the predicted results from SoftNet-35 which is listed in Tables 19 and 20 are reasonable. Even though we consider reasonable depth (up to 34 hidden layer in the first sub-architecture), an important question to ask is how deeper we could go in the first sub-architecture for SoftNet? Is it possible to train with hundreds of layers. One of the important challenge to observe is overfitting. We emphasis that making the network complex needs to be balanced with the given task. Increasing training dataset is also very important. Moreover, regularization methods could further enhance predictive performance. Here, no regularization in the form of lasso regression, ridge regression, combination of ridge and lasso regressions, dropout, batch normalization is implemented in the design of SoftNet.

Table 18: Hyperparameters for SoftNet-35.

Hyperparameters	Values
Training steps	12000
Learning rate decay	Yes (exponential)
Optimizer, β_1, β_2, ϵ_{adam}	Adam, 0.994, 0.999, 10^{-8}
Stretching	Yes
$(s_1,\ s_2,\ s_3,\ s_4)$	(0.03, 0.02, 0.03, 0.03)
SoftNet-35	
Starter learning rate	0.02
Decay step, Decay rate	2000, 0.68
Softmax block	2 hidden layers
Number of softmax block	12
Units: Softmax blocks	[(48,32), (24,16), (16,12), (12,12), (8,8), (8,8), (8,8), (8,8), (6,6), (6,6), (6,6), (6,6), (4,4), (4,4), (4,4), (4,4), (16,12)]
Units: Multiplicative ensembled layer	12

Table 19: Solutions of the value function and the early exercise boundary with SoftNet-35 and $(14 \times 14 \times 14 \times 14)$ stretched dataset. Here, $T = \frac{1}{2}$, $\upsilon_0 = 0.04$, and $r_0 = 0.04$ are considered.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
6490 training steps			
Value function	1.333	4.578	10.896
Early exercise boundary	74.930	83.737	92.110
12000 training steps			
Value function	1.341	4.589	10.963
Early exercise boundary	74.998	83.331	91.664

Moreover, we consider training dataset that is around 40000 samples. Also, there is a multiplicative ensembled layer that multiplies the first and second sub-architectures which could further increase the complexness of SoftNet. For the given task with the considered maximum training dataset which is around 40000 samples, we think that the implemented depths in this research for SoftNet is reasonable and hope to investigate on deeper network in our future work for more complex task.

Remark. *It is important to observe that we have presented several predictions from different trainings across datasets and depths. Here, we note an important finding. For the considered SoftNets, we observed no degradation in predictive performance as depth increases (up to SoftNet-35). We hope to further observe this trend in our future work when we consider more complex tasks involving SoftNet with deeper structure.*

Table 20: Solutions of the value function and the early exercise boundary with SoftNet-35 and $(14 \times 14 \times 14 \times 14)$ stretched dataset. Here, $T = \frac{1}{2}$ and $r_0 = 0.04$ are considered.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
6490 training steps						
Value function	2.816	6.707	12.773	4.538	8.903	14.802
Early exercise boundary	67.946	75.495	83.044	61.073	67.858	74.644
12000 training steps						
Value function	2.826	6.680	12.740	4.558	8.828	14.787
Early exercise boundary	67.485	74.984	82.482	60.327	67.030	73.733

7.4 Numerical experiment for LinNet

In this subsection, we test the predictive performance of LinNet for solving the three-dimensional American Heston-CIR model.

Table 21: Hyperparameters for LinNet-(13,j), LinNet-(25,j), and LinNet-(52,j).

Hyperparameters	Values
Training steps	12000
Learning rate decay	Yes (exponential)
Optimizer, β_1, β_2, ϵ_{adam}	Adam, 0.994, 0.999, 10^{-8}
Stretching	Yes
$(s_1,\ s_2,\ s_3,\ s_4)$	(0.04, 0.02, 0.04, 0.04), (0.03, 0.02, 0.03, 0.03)
Starter learning rate, Decay step, Decay rate	0.1, 2000, 0.80
LinNet-(13,j) (see Figure 14)	
Number of softmax blocks/aggregating units	4/4
Units: Softmax blocks and aggregating units	[(48)/(16),(32,24),(16),(12,8),(16), (12,8),(16),(24,12)]
Units: Second/third sub-architecture	(48,24)/24
Units: Multiplicative ensembled layer	8
LinNet-(25,j)	
Number of softmax blocks/aggregating units	8/8
Units: Softmax blocks and aggregating units	[(48)/(16),(32,24),(16),(8,6),(16), (8,6),(16),(8,6),(16),(8,6),(16) (8,6),(16),(8,6),(16),(24,12)]
Units: Second/third sub-architecture	(48,24)/24
Units: Multiplicative ensembled layer	8
LinNet-(52,j)	
Number of softmax blocks/aggregating units	17/17
Units: Softmax blocks and aggregating units	[(48)/(16),(32,24),(16),(8,6),(16), (8,6),(16),(8,6),(16),(8,6),(16) (8,6),(16),(8,6),(16),(8,6),(16) (8,6),(16),(8,6),(16),(8,6),(16) (8,6),(16),(8,6),(16),(8,6),(16) (8,6),(16),(8,6),(16),(24,16)]
Units: Second/third sub-architecture	(48,24)/24
Units: Multiplicative ensembled layer	12

We hope to carry out extensive experiments in the following subsection to better understand the predictive performance of LinNet using different depths and training steps and datasets. We also hope to validate the performance of LinNet by comparing its predicted results with the ones obtained from existing methods.

7.4.1 Testing the performance of LinNet

Here, we conduct an experiment to test the predictive performance of LinNet in training with deeper hidden layers. The considered hyperparameters are listed in Table 21.

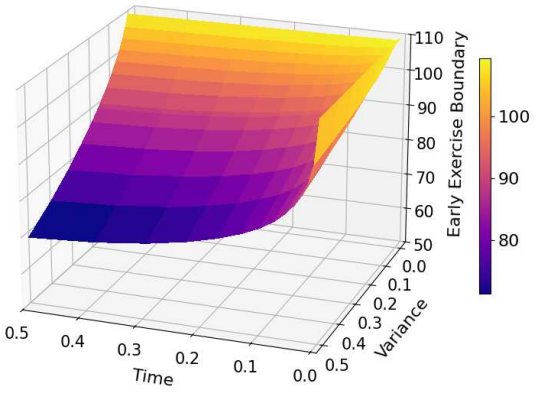
Table 22: Solutions of the value function and the early exercise boundary with LinNet-(13,1) and 12000 training steps. Here, $\upsilon_0 = 0.04$ and $r_0 = 0.04$.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$(10 \times 10 \times 10 \times 10)$ stretched dataset			
Value function	1.352	4.574	10.927
Early exercise boundary	75.116	83.463	91.809
$(14 \times 14 \times 14 \times 14)$ stretched dataset			
Value function	1.379	4.602	10.965
Early exercise boundary	74.889	83.210	91.530

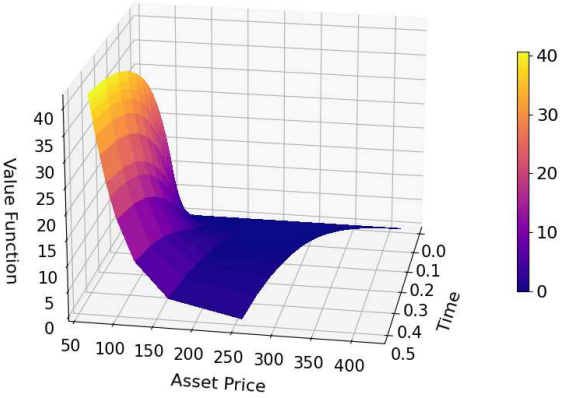
Table 23: Solutions of the value function and the early exercise boundary with LinNet-(13,1) and 12000 training steps. Here, $r_0 = 0.04$.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$(10 \times 10 \times 10 \times 10)$ stretched dataset						
Value function	2.816	6.657	12.712	4.585	8.855	14.812
Early exercise boundary	67.605	75.116	82.628	60.304	67.004	73.705
$(14 \times 14 \times 14 \times 14)$ stretched dataset						
Value function	2.843	6.706	12.786	4.592	8.870	14.850
Early exercise boundary	67.408	74.898	82.388	60.239	66.932	73.626

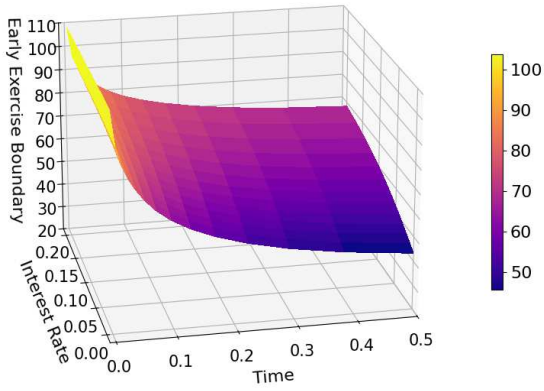
We seek to understand what extent LinNet holds importance in terms of good predictive performance when deeper hidden layer is considered in the first sub-architecture which is the main architecture. Here, we compare predicted results from LinNet with other existing results obtained from the analytical approach of Medvedev and Scaillet [17], the Wiener-Hopf factorization method of Boyarchenko and Levendorskiĭ [4], and the LSM algorithm of Longstaff and Schwartz[16]. We consider LinNet-(13,1), LinNet-(25,1), and LinNet-(52,1). The predicted results are listed in Tables 23-27. Our observation is a reasonable predictive performance in comparison with the existing methods.



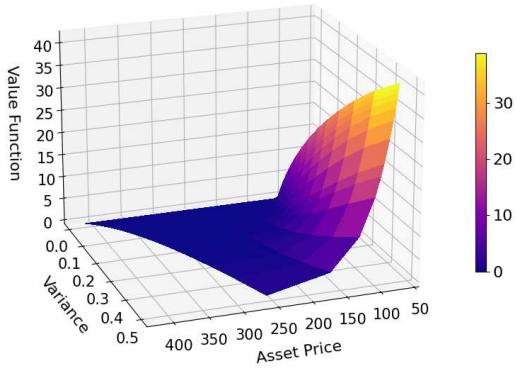
(a) Early exercise boundary for a fixed interest rate.



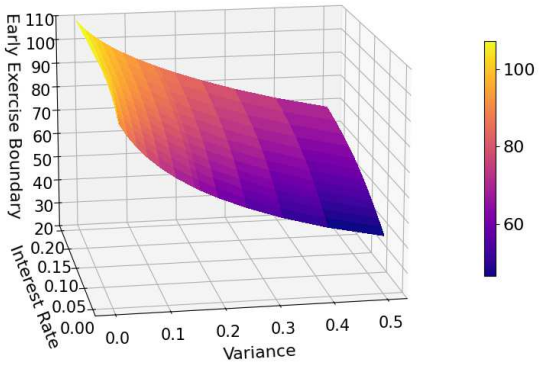
(b) Value function for fixed variance and interest rate.



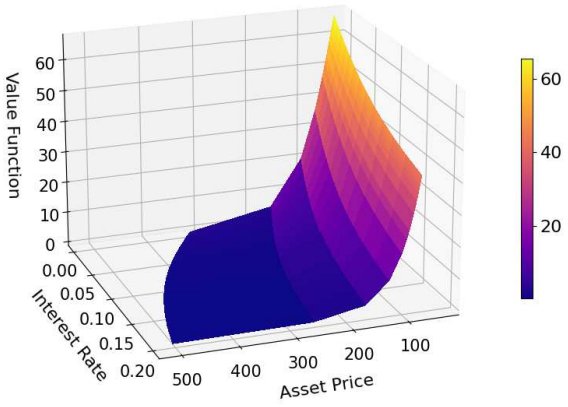
(c) Early exercise boundary for a fixed variance.



(d) Value function for fixed time and interest rate.

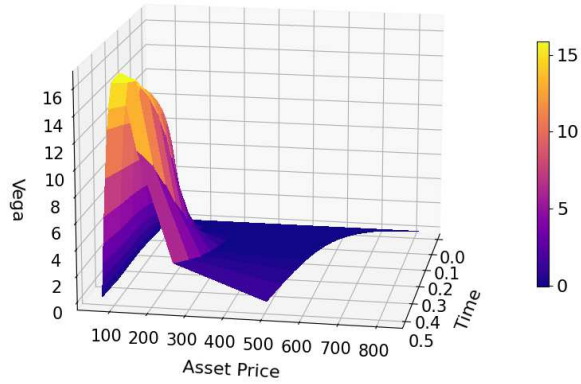


(e) Early exercise boundary for a fixed time.

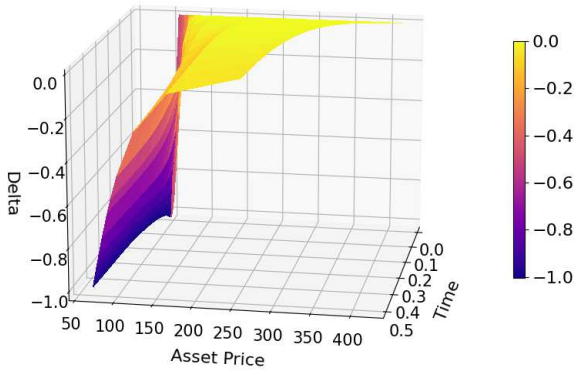


(f) Value function for fixed time and variance.

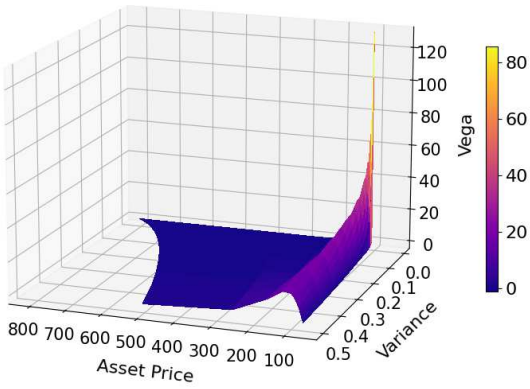
Figure 19: Early exercise boundary and value function surfaces for the Heston-CIR model with LinNet-(52,1), $(14 \times 14 \times 14 \times 14)$ stretched dataset ($s_1 = 0.03,\ s_2 = 0.02,\ s_3 = 0.03,\ s_4 = 0.03$), and $T = \frac{1}{2}$.



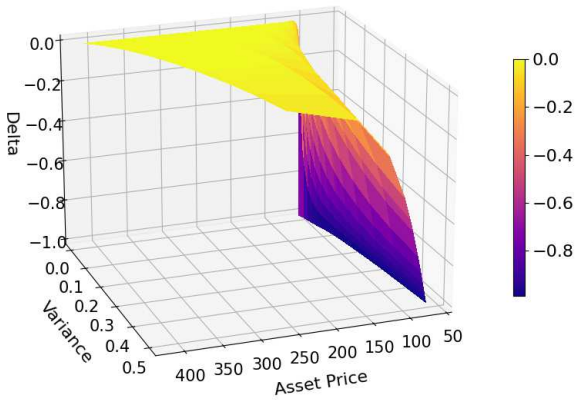
(a) Vega surface for fixed variance and interest rate.



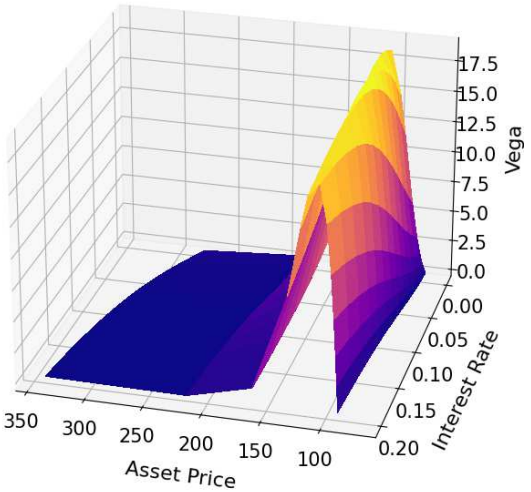
(b) Delta surface for fixed variance and interest rate.



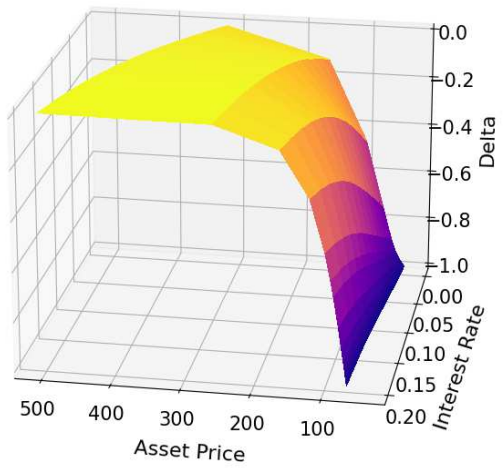
(c) Vega surface for fixed time and interest rate.



(d) Delta surface for fixed time and interest rate.

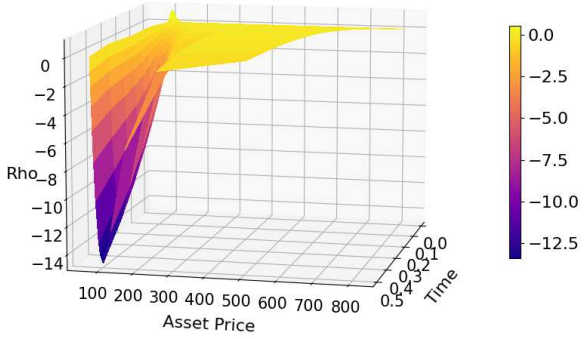


(e) Vega surface for fixed time and interest rate.

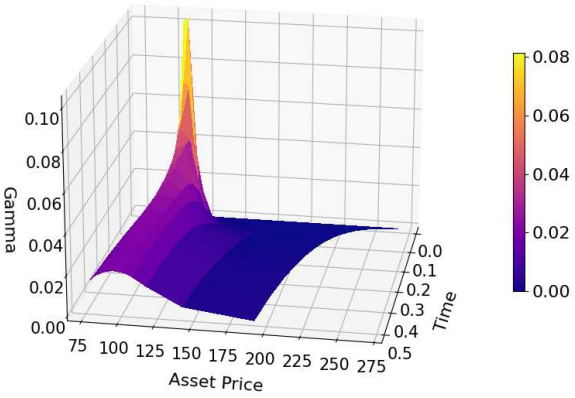


(f) Delta surface for fixed time and variance.

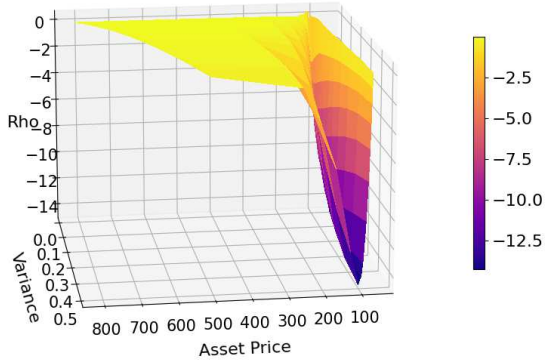
Figure 20: Vega and Delta for the Heston-CIR model with LinNet-(52,1), $(14 \times 14 \times 14 \times 14)$ stretched dataset $(s_1 = 0.03,\ s_2 = 0.02,\ s_3 = 0.03,\ s_4 = 0.03)$, and $T = \frac{1}{2}$.



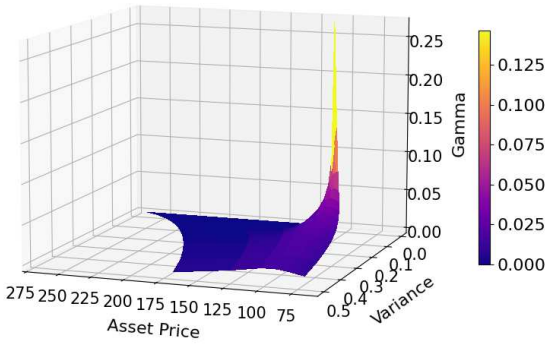
(a) Rho surface for fixed variance and interest rate.



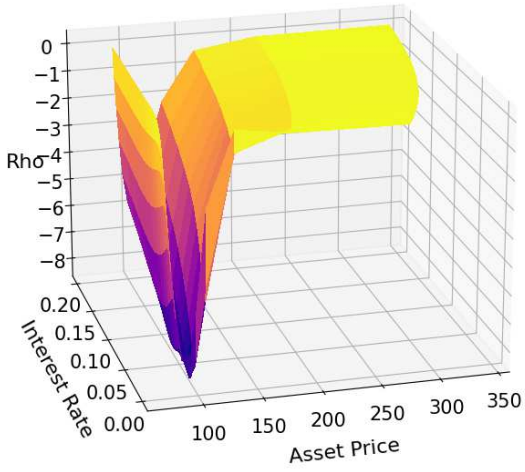
(b) Gamma surface for fixed variance and interest rate.



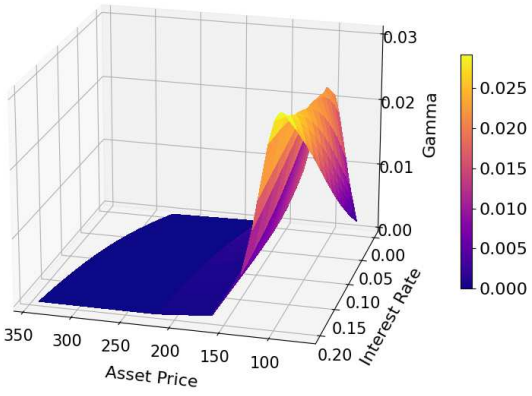
(c) Rho surface for fixed variance and interest rate.



(d) Gamma surface for fixed time and interest rate.

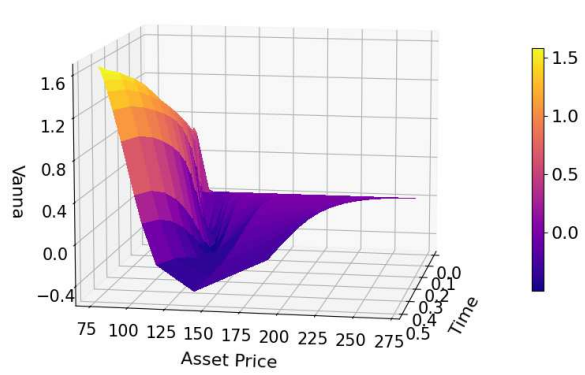


(e) Rho surface for fixed variance and interest rate.

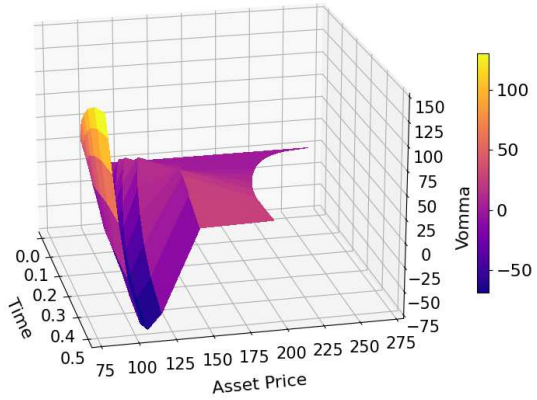


(f) Gamma surface for fixed time and variance.

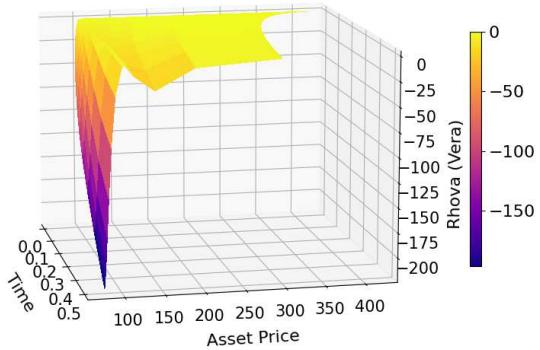
Figure 21: Rho and Gamma for the Heston-CIR model with LinNet-(52,1), $(14\times14\times14\times14)$ stretched dataset $(s_1 = 0.03,\ s_2 = 0.02,\ s_3 = 0.03,\ s_4 = 0.03)$, and $T = \frac{1}{2}$.



(a) Vanna for a fixed variance and interest rate.



(b) Vomma for fixed variance and interest rate.



(c) Rhova (Vera) for fixed variance and interest rate.

Figure 22: Vanna, Vomma, and Rhova for the Heston-CIR model with LinNet-(52,1), $(14\times14\times14\times14)$ stretched dataset ($s_1 = 0.03,\ s_2 = 0.02,\ s_3 = 0.03,\ s_4 = 0.03$), and $T = \frac{1}{2}$.

Table 24: Solutions of the value function and the early exercise boundary with LinNet-(25,1), varying training dataset, and 12000 training steps. Here, $v_0 = 0.04$ and $r_0 = 0.04$.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$(12 \times 12 \times 12 \times 12)$ stretched dataset			
Value function	1.339	4.582	10.943
Early exercise boundary	75.007	83.341	91.675
$(14 \times 14 \times 14 \times 14)$ stretched dataset			
Value function	1.387	4.604	10.949
Early exercise boundary	74.994	83.327	91.659
$(16 \times 16 \times 16 \times 16)$ stretched dataset			
Value function	1.370	4.597	10.953
Early exercise boundary	75.028	83.365	91.701

Table 25: Solutions of the value function and the early exercise boundary with LinNet-(25,1) and 12000 training steps. Here, $r_0 = 0.04$.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and LevendorskiĬ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$(12 \times 12 \times 12 \times 12)$ stretched dataset						
Value function	2.816	6.661	12.721	4.553	8.818	14.780
Early exercise boundary	67.657	75.175	82.692	60.252	66.947	73.642
$(14 \times 14 \times 14 \times 14)$ stretched dataset						
Value function	2.861	6.701	12.757	4.575	8.843	14.814
Early exercise boundary	67.552	75.057	82.563	60.055	66.727	73.400
$(16 \times 16 \times 16 \times 16)$ stretched dataset						
Value function	2.853	6.705	12.769	4.619	8.898	14.865
Early exercise boundary	67.387	74.875	82.362	59.877	66.530	73.182

Table 26: Solutions of the value function and the early exercise boundary with LinNet-(52,1) and varying training dataset and steps. Here, $\upsilon_0 = 0.04$ and $r_0 = 0.04$.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and LevendorskiĬ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$(14 \times 14 \times 14 \times 14)$ stretched dataset 7336 training steps			
Value function	1.333	4.588	10.936
Early exercise boundary	75.109	83.454	91.800
$(16 \times 16 \times 16 \times 16)$ stretched dataset 12000 training steps			
Value function	1.330	4.558	10.927
Early exercise boundary	75.070	83.411	91.752

Moreover, we present the predicted plot profiles of the early exercise boundary, the value function, and some of the first and second-order Greeks by slicing their predictions with LinNet-(52,1) as displayed in Figures 27-28. It is interesting to see that the plots obtained from LinNet-(52,1) are reasonably smooth. The value function vanishes as we move out of the money, and the Delta sensitivity is reasonably in the interval of [-1,0]. Moreover Gamma and Vega are reasonably in the positive domain and Rho is reasonably in the negative domain (for put options).

Furthermore, we increase the units across hidden layers in the first sub-architecture within LinNet-(25,1) as shown in Table 28 and observe if the increase in units across hidden layer might further enhance predictive performance for relatively deep structure. From Tables 29-30, we observe not much difference from the predicted results obtained with varying units in the first sub-architecture with LinNet-(25,1).

Table 27: Solutions of the value function and the early exercise boundary with LinNet-(52,1) and varying training datasets and steps. Here, $r_0 = 0.04$.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$(14 \times 14 \times 14 \times 14)$ stretched dataset 7336 training steps						
Value function	2.798	6.656	12.746	4.540	8.812	14.794
Early exercise boundary	67.543	75.048	82.553	60.248	66.942	73.636
$(16 \times 16 \times 16 \times 16)$ stretched dataset 12000 training steps						
Value function	2.805	6.655	12.727	4.530	8.815	14.791
Early exercise boundary	67.735	75.261	82.787	60.497	67.219	73.941

Table 28: Varying units for the first sub-architecture within LinNet-(25,1).

Case-1 (also see Table 21)	
Units: Softmax blocks and aggregating units	[(48)/(16),(32,24),(16),(8,6),(16), (8,6),(16),(8,6),(16),(8,6),(16) (8,6),(16),(8,6),(16),(24,12)]
Units: Multiplicative ensembled layer	8
Case-2	
Units: Softmax blocks and aggregating units	[(64)/(16),(48,32),(16),(12,8),(16), (12,8),(16),(12,8),(16),(12,8),(16) (12,8),(16),(12,8),(16),(24,12)]
Units: Multiplicative ensembled layer	12

Table 29: Solutions of the value function and the early exercise boundary with varying units in the first sub-architecture within LinNet-(25,1), $(14 \times 14 \times 14 \times 14)$ stretched dataset and 12000 training steps. Here, $\upsilon_0 = 0.04$ and $r_0 = 0.04$.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
Case-1			
Value function	1.370	4.597	10.953
Early exercise boundary	75.028	83.365	91.701
Case-2			
Value function	1.315	4.552	10.908
Early exercise boundary	75.291	83.657	92.023

Table 30: Solutions of the value function and the early exercise boundary with varying units in the first sub-architecture within LinNet-(25,1), $(14 \times 14 \times 14 \times 14)$ stretched dataset and 12000 training steps. Here, $r_0 = 0.04$.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
Case-1						
Value function	2.853	6.705	12.769	4.619	8.898	14.865
Early exercise boundary	67.387	74.875	82.362	59.877	66.530	73.182
Case-2						
Value function	2.793	6.653	12.717	4.565	8.850	14.827
Early exercise boundary	67.591	75.101	82.611	60.099	66.777	73.455

7.4.2 Experiment on the suite of design for the third sub-architecture

For the third sub-architecture, we present two different options in terms of its design. Here, our intention is to observe the predictive performance of LinNet with these two designs.

Table 31: Solutions of the value function and the early exercise boundary with LinNet-(13,j), $(10 \times 10 \times 10 \times 10)$ stretched dataset and 12000 training steps. Here, $\upsilon_0 = 0.04$ and $r_0 = 0.04$.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
$j = 1$			
Value function	1.352	4.574	10.927
Early exercise boundary	75.116	83.463	91.809
$j = 2$			
Value function	1.345	4.553	10.907
Early exercise boundary	75.160	83.511	91.862

Table 32: Solutions of the value function and the early exercise boundary with LinNet-(13,j) and varying training datasets and steps. Here, $r_0 = 0.04$.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
$j = 1$						
Value function	2.816	6.657	12.712	4.585	8.855	14.812
Early exercise boundary	67.605	75.116	82.628	60.304	67.004	73.705
$j = 2$						
Value function	2.850	6.698	12.759	4.580	8.863	14.835
Early exercise boundary	68.320	75.911	83.502	60.582	67.313	74.044

Though it is not main the objective of this research, it is important to understand if any of these

implementation holds further importance. We list the predicted results in Tables 31-32. The predicted results for the value function is reasonable. We hope to further observe the performance of the second option for the third sub-architecture in our future work for solving wider range of high-dimensional American options as it appears more unifying than the first options.

7.4.3 Training with 100 hidden layers

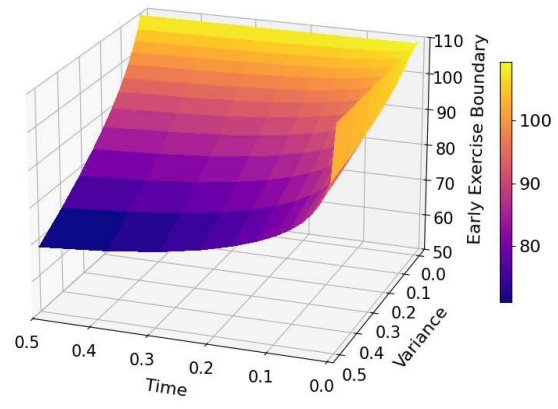
It will be important to understand if one can train with hundreds of layers in the first sub-architecture within LinNet. Even though our consideration in the previous experiment is up to 52 hidden layers, does it worth increasing the depth further? At what point is it necessary? These are important questions that will depend on several factors and challenges.

Table 33: Hyperparameters for LinNet-(100,1).

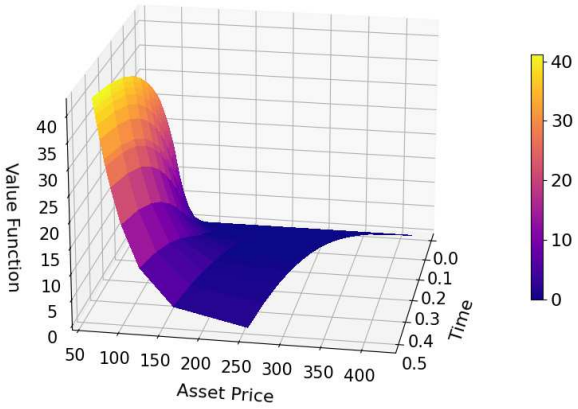
Hyperparameters	Values
Training steps	10000
Learning rate decay	Yes (exponential)
Optimizer, β_1, β_2, ϵ_{adam}	Adam, 0.994, 0.999, 10^{-8}
Stretching	Yes
(s_1, s_2, s_3, s_4)	(0.04, 0.02, 0.04, 0.04)
Starter learning rate, Decay step, Decay rate	0.1, 2000, 0.80
Number of softmax blocks/aggregating units	33/33
Units: Softmax blocks and aggregating units	[(64)/(8),(32,24),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8), (8,8),(8),(8,8),(8),(8,8),(8),(24,16)]
Units: Second/third sub-architecture	(32,16)/24
Units: Multiplicative ensembled layer	8

One of such challenge is overfitting [8, 11]. If a simple task is considered, it might not be ideal to consider a very complex and deeper structure. Regularization methods could also present a very important remedy for overfitting. We mention that no regularization in the form of lasso regression, ridge regression, combination of ridge and lasso regression, dropout, batch normalization is implemented in the design of LinNet. Moreover, it is also important to observe that we consider three sub-architectures in the design of LinNet which we ensembled multiplicatively. Such implementation could further increase the complexness of the LinNet.

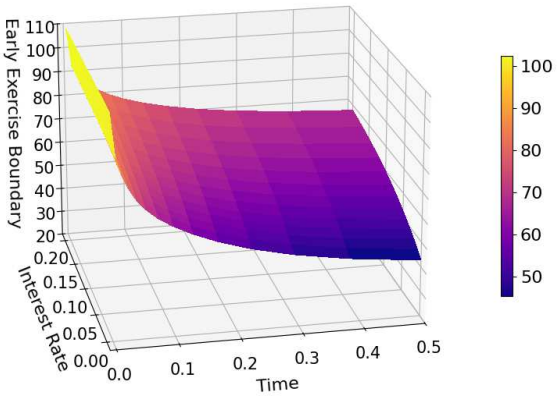
In this subsection, we experiment more with deeper structure. We consider LinNet-(100,1) which involves 100 hidden layers. Here, we stretch the performance of LinNet to observe if it has the tendency to perform well on reasonably deeper structure. Our strategy for LinNet-(100,1) is to train with deeper depth and thinner width and thereby reasonably reduce the units across hidden layers in the first and second sub-architectures as described in Table 33.



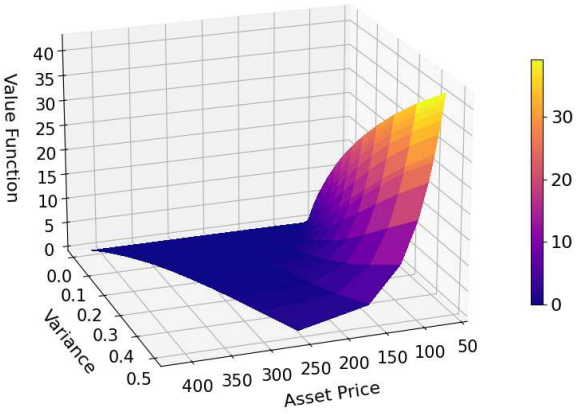
(a) Early exercise boundary for a fixed interest rate.



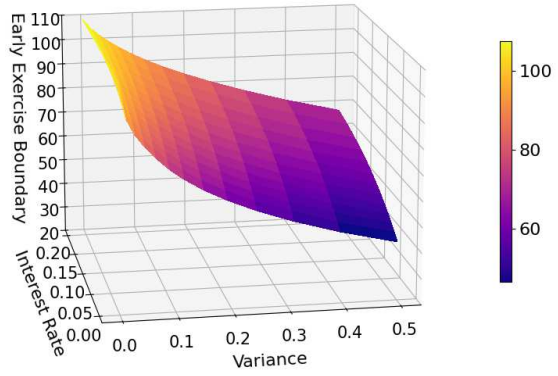
(b) Value function for fixed variance and interest rate.



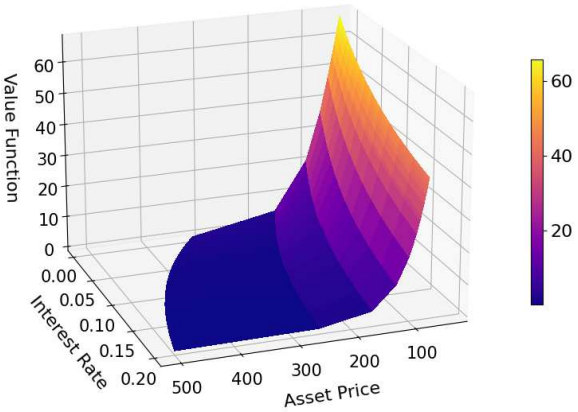
(c) Early exercise boundary for a fixed variance.



(d) Value function for fixed time and interest rate.

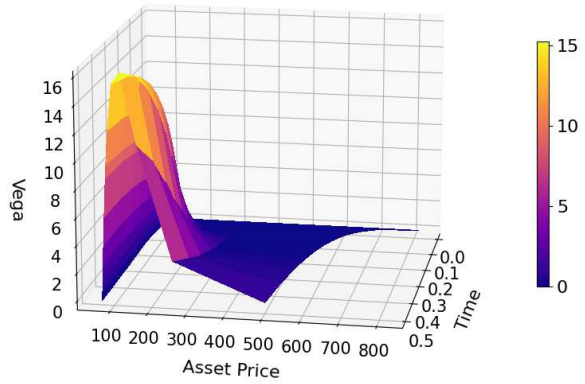


(e) Early exercise boundary for a fixed time.

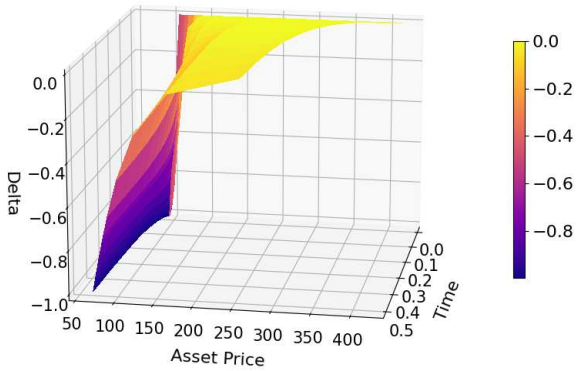


(f) Value function for a fixed time and variance.

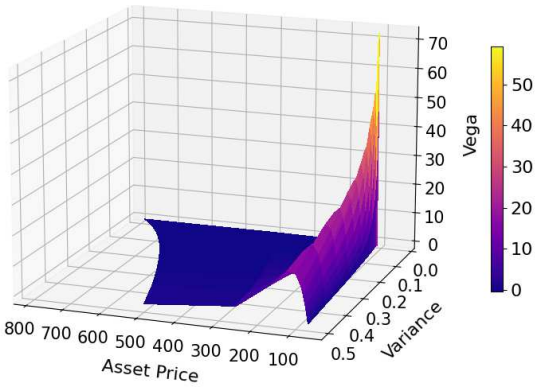
Figure 23: Early exercise boundary and value function surfaces for the Heston-CIR model with LinNet-(100,1), $(14 \times 14 \times 14 \times 14)$ stretched dataset $(s_1 = 0.04,\ s_2 = 0.02,\ s_3 = 0.04,\ s_4 = 0.04)$.



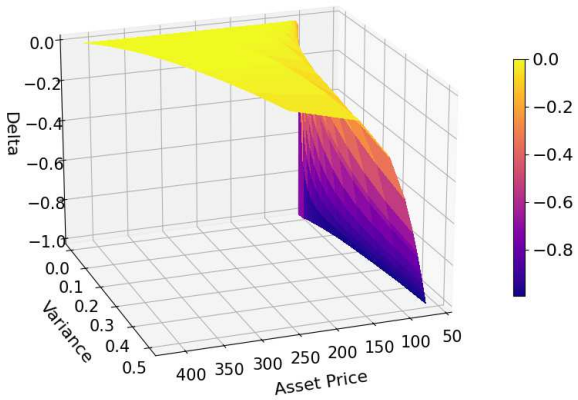
(a) Vega surface for fixed variance and interest rate.



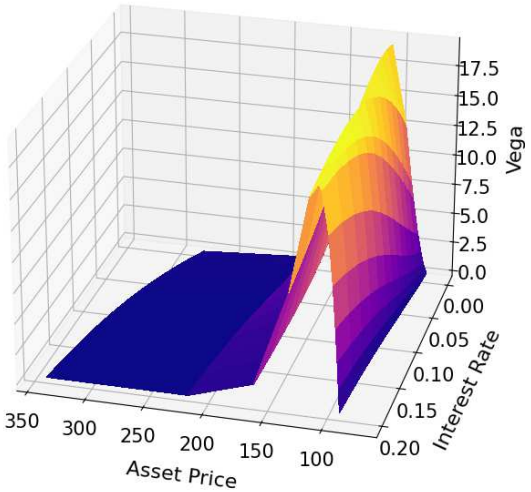
(b) Delta surface for fixed variance and interest rate.



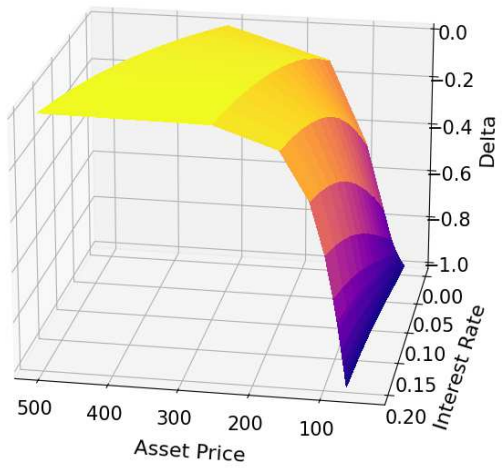
(c) Vega surface for fixed time and interest rate.



(d) Delta surface for fixed time and interest rate.

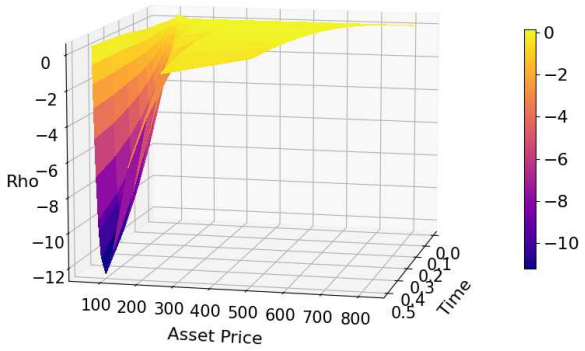


(e) Vega surface for fixed time and interest rate.

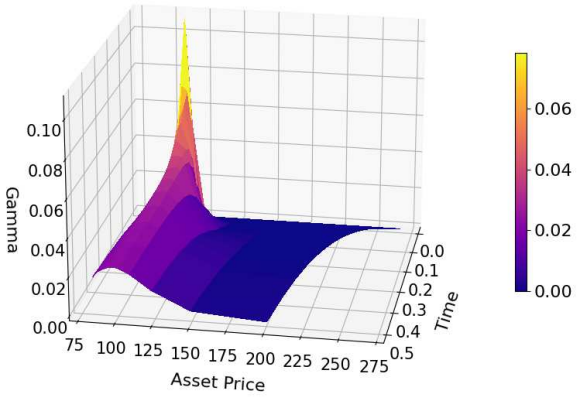


(f) Delta surface for a fixed time and variance.

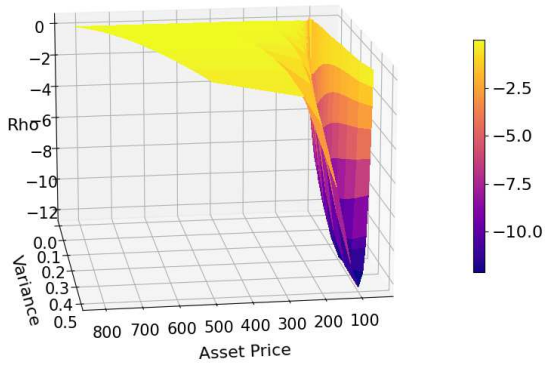
Figure 24: Vega and Delta for the Heston-CIR model with LinNet-(100,1), $(14\times14\times14\times14)$ stretched dataset $(s_1 = 0.04,\ s_2 = 0.02,\ s_3 = 0.04,\ s_4 = 0.04)$.



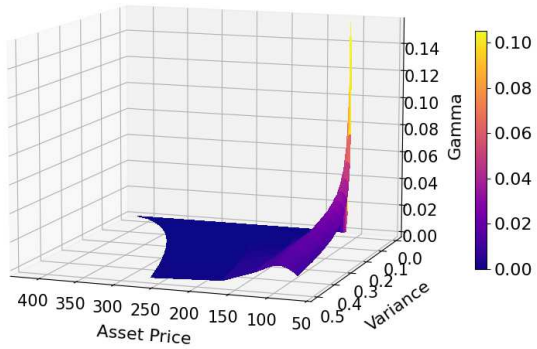
(a) Rho surface for fixed variance and interest rate.



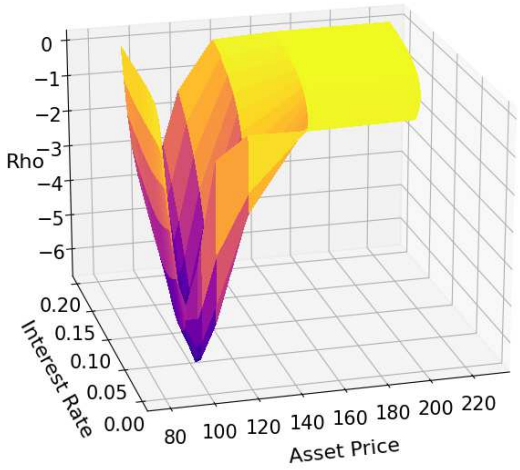
(b) Gamma surface for fixed variance and interest rate.



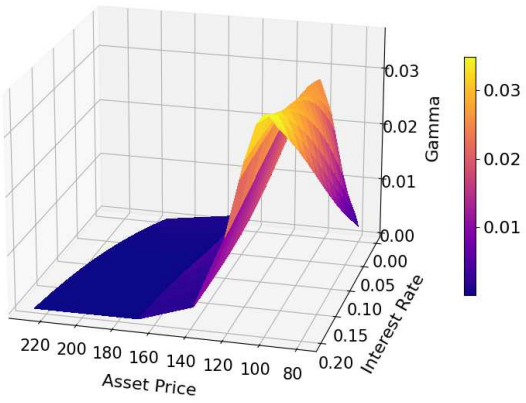
(c) Rho surface for fixed variance and interest rate.



(d) Gamma surface for fixed time and interest rate.

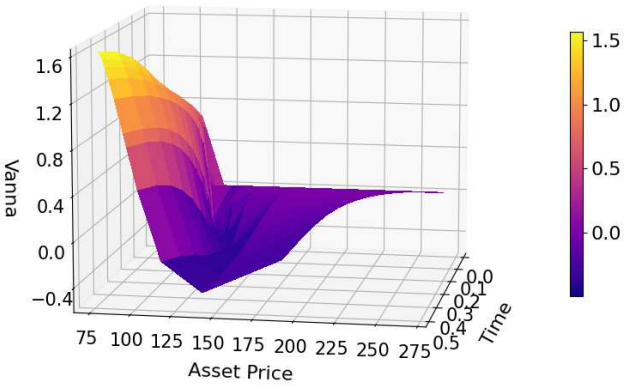


(e) Rho surface for fixed variance and interest rate.

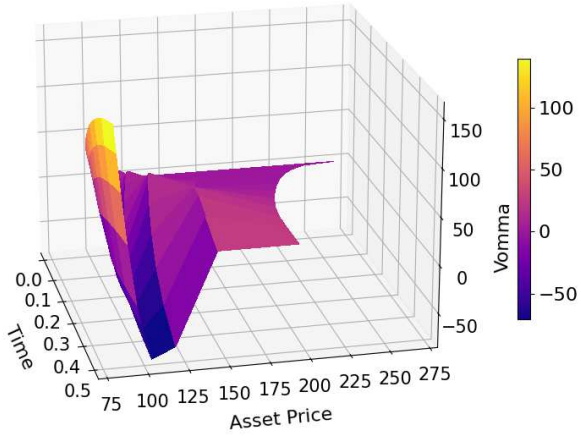


(f) Gamma surface for a fixed time and variance.

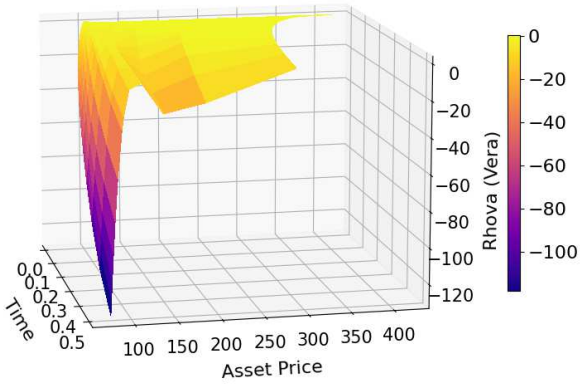
Figure 25: Rho and Gamma for the Heston-CIR model with LinNet-(100,1), $(14 \times 14 \times 14 \times 14)$ stretched dataset ($s_1 = 0.04,\ s_2 = 0.02,\ s_3 = 0.04,\ s_4 = 0.04$).



(a) Vanna for fixed variance and interest rate.



(b) Vomma for fixed variance and interest rate.



(c) Rhova for fixed variance and interest rate.

Figure 26: Vanna, Vomma, and Rhova for the Heston-CIR model with LinNet-(100,1), $(14 \times 14 \times 14 \times 14)$ stretched dataset ($s_1 = 0.04,\ s_2 = 0.02,\ s_3 = 0.04,\ s_4 = 0.04$).

Table 34: Solutions of the value function and the early exercise boundary with LinNet-(100,1), $(14 \times 14 \times 14 \times 14)$ dataset and 10000 training steps. Here, $v_0 = 0.04$ and $r_0 = 0.04$.

Methods	$K = 90$	$K = 100$	$K = 110$
Boyarchenko and Levendorskiĭ [4]	1.362	4.574	10.896
Medvedev and Scaillet [17]	1.357	4.559	10.789
Longstaff and Schwartz [16]	1.351	4.578	10.912
LinNet-(100,1)			
Value function	1.315	4.615	10.952
Early exercise boundary	75.159	83.510	91.861

In Tables 34-35, we list the predicted results obtained from LinNet-(100,1) and further observe a reasonable predictive performance for the value function when compared with the existing results. Moreover, the plot profiles of the early exercise boundary, the value function, and Greeks obtained from LinNet-(100,1) which are presented in Figures 23-26 are also reasonable. Though not presented,

we observed that further reduction in units across layers in the first sub-architecture (as listed in Table 33) might reduce performance of LinNet-(100,1). It is possible that slight increase in units across hidden layers in the first and second sub-architecture might further enhance predictive performance of LinNet-(100,1). However, drastic increase in the number of units across layers might result in overfitting because we consider reasonable small training dataset with no regularization in the form of lasso regression, ridge regression, combination of ridge and lasso regression, dropout, and batch normalization. Moreover, though not presented, the performance of LinNet-(100,1) could be enhanced further by increasing training datasets and steps.

Table 35: Solutions of the value function and the early exercise boundary with LinNet-(100,1) and varying training datasets and steps. Here, $r_0 = 0.04$.

	$\upsilon = 0.09$			$\upsilon = 0.16$		
Methods/K	90	100	110	90	100	110
Boyarchenko and Levendorskiĭ [4]	2.852	6.717	12.773	4.667	8.986	14.972
Medvedev and Scaillet [17]	2.822	6.654	12.680	4.575	8.840	14.781
Longstaff and Schwartz [16]	2.819	6.678	12.678	4.573	8.843	14.785
LinNet-(100,1)						
Value function	2.828	6.721	12.781	4.598	8.887	14.852
Early exercise boundary	67.427	74.918	82.410	60.220	66.911	73.602

We note that we are very judicious in terms of considered training datasets and units across hidden layers. To a large extent, we reduce the units across the hidden layers for LinNet-(100,1). It is interesting to see that we still obtain reasonable prediction with LinNet-(100,1) using relatively small training dataset.

Moreover, we consider several predictions from different trainings across datasets, depths, and training steps and observe no degradation in predictive performance across depths. We hope to further increase the depth of the first sub-architecture within LinNet and observe the performance in our future work for more complex tasks. We further note that making the network more complex with the necessary regularization method(s) for a very complex task might further unlock the importance of LinNet especially in training with hundreds of layers. Finally, we note that the challenging nature of our problem makes its ideal to train with LinNet-(100,1). The piecewise nature of the value function, the discontinuity in delta (though we are focusing on the positive real line), The spike and singularity in Gamma, and also in some of the second-order Greeks. Moreover, it is also important to mention the singularity in the first-order derivatives of the early exercise boundary. These irregularities reside in the lower left corner point $(\tau, x, \upsilon, r) = (0, 0, 0, 0)$. By implementing grid stretching strategy, we concentrate most of the sample points in the locality of $(\tau, x, \upsilon, r) = (0, 0, 0, 0)$. We envisage that such stretching strategy could further increase the complexness of the given task and make it reasonable to consider a network model with deeper structure. In our future work, we hope to consider deeper structure for similar or more complex task, especially in higher-dimensional setting, and when full parametrization of parameters and variables are considered which could involve the parametrization of correlation matrix or Cholesky factor.

7.5 Discussion on computational runtime and loss function

In this subsection, we compare AddNet, SoftNet, and LinNet. Already, we have some knowledge a priori that feature reuse via concatenating mechanism will impact on the computational cost for SoftNet. Though its predictive performance is very reasonable, training with very deep structure will drastically increase the number of neurons in the coupling unit. This effect will further increase the computational cost and might hamper the deployment of SoftNet in a resource efficient technology like mobile devices. AddNet presents another important choice when efficiently implemented. However, it might not be ideal for a very deep structure because of vanishing gradient effect that will impact on the single-mixture of network solutions obtained from the final hidden layers and local exit very

far from the input layer. LinNet is a very important network model for us due to its flexibility and efficiency. As the network structure gets deeper, its additive attributes avoid growth of number of neurons in the aggregating units, thereby reducing computational cost to a reasonable extent. This is why we consider up to 100 hidden layers for LinNet. Moreover, layer reuse via aggregation allows training with very deep structure similar to SoftNet. Hence, the performance of LinNet can be seen as combination of the performances of AddNet and SoftNet with respect to mitigating vanishing gradient effect and reducing computational cost.

Table 36: Comparison of the computational runtime.

Prediction time			Training time		
AddNet-(12,6)	LinNet-(13,1)	SoftNet-13	AddNet-(12,6)	LinNet-(13,1)	SoftNet-13
0.241s	0.148s	0.287s	4050s	2919s	3314s

To elucidate further, we briefly compare the training and prediction runtimes for the three proposed network models and list the values in Table 36. One can observe that LinNet has the least value for the training and prediction runtimes in seconds. AddNet has less prediction time when compared with SoftNet. Though not presented, we expect that increase in the number of early exits will increase training runtime. Here, we consider 5 early exits. In addition, we examine the loss function (training) for the three proposed network models and display their plot profiles in Figure 27. We observe less variation in the three plots of the loss function for the three proposed network models.

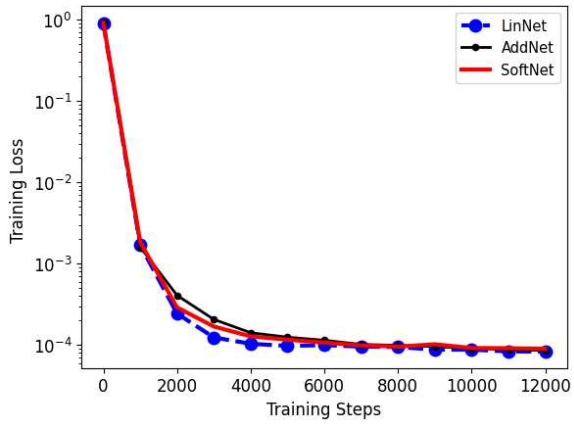


Figure 27: Loss function against training steps involving 13 hidden layers.

8 Conclusion

We have exploited the design of advanced deeper learning architectures for solving 3D American Heston-CIR model and thereby, have proposed AddNet, LinNet, and SoftNet. The proposed network models AddNet, LinNet, and SoftNet are generally formulated via softmax blocks and multiplicative ensembling of sub-network models. Moreover, network models AddNet, LinNet, and SoftNet are uniquely designed via early exits, aggregation of linear units, and feature reuse.

To better understand the importance of the proposed network models and test their performances, we have compared predictions with existing results. In addition, we compared AddNet, LinNet, and SoftNet with conventional DNN architectures. We confirm that the conventional DNN architecture has a very poor predictive performance with thirteen hidden layers. However, AddNet, LinNet, and SoftNet have a reasonable predictive performance beyond thirteen hidden layers. Furthermore, we have considered up to hundred hidden layers within LinNet and observed reasonable predictive performance. These three proposed network models predict the surfaces of the early exercise boundary (and partial derivatives), the value function, and Greeks after training.

Furthermore, no regularization in the form of lasso regression, ridge regression, combination of ridge and lasso regression, dropout, batch normalization was implemented in the design of these three proposed network models. It could be possible that considering some of these regularization methods might further enhance the predictive performance (generalization) of the proposed network models for deeper architectures. In our future work, we hope to further enhance our deeper network models for solving higher-dimensional American pricing models. We also hope to consider full parametrization of two and three-dimensional American pricing models. In such scenario, the complexity of the input feature will substantially increase due to parametrization of the correlation matrix and other parameters like volatility of volatility, volatility of interest rate, market price of risk, etc. It could be pertinent to consider an advanced architecture for training such models.

Acknowledgement

The first author acknowledge the funding in part by an NSERC Discovery Grant.

Funding

The first author is funded in part by an NSERC Discovery Grant.

Code and Data Availability

The code and data could be shared at a reasonable request.

Conflict of interest

The authors declare that they have no conflict of interest

Authors' contributions

Chinonso Nwankwo: Conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing–original draft preparation, visualization. **Weizhong Dai**: Supervision, writing–review and editing, project administration.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, J., Levenberg, M., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint*, arXiv:1603.04467.

[2] Barron, J. T. (2021). Squareplus: A softplus-like algebraic rectifier. *arXiv preprint*, arXiv:2112.11687v1.

[3] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 157-166.

[4] Boyarchenko, S., & LevendorskiĬ, S. (2013). American options in the Heston model with stochastic interest rate and its generalizations. *Applied Mathematical Finance*, 20(1), 26-49.

[5] Chen, D., Hu, F., Nian, G., & Yang, T. (2022). Deep residual learning for nonlinear regression. *Entropy*, 22, 193.

[6] Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., & Garcia, R. (2000). Incorporating second-order functional knowledge for better options pricing. *Advances in Neural Information Processing System*, 13.

[7] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 249-256).

[8] He, K., Zhang, X., Ren, S., & Sun J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 770-778).

[9] Hess, A. (2018). Exploring feature reuse in DenseNet architectures. *arXiv preprint*, arXiv:1806.01935.

[10] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 31.

[11] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *In Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).

[12] Jiang, C., Jiang, C., Chen, D., & Hu, F. (2022). Densely connected neural networks for nonlinear regression. *Entropy*, 24(7), 876.

[13] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, (pp. 448-456).

[14] Laskaridis, S., Kouris, A., & Lane, N. D. (2021). Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceeding of the 5th International Workshop on Embedded and Mobile Deep Learning* (pp. 1-6).

[15] Li, Y., Li, M., Li, Z., Xiao, C., & Li, H. (2022). EFRNet: efficient feature reuse network for real-time semantic segmentation. *Neural Processing Letters*, 54(6), 4647-4659.

[16] Longstaff, F. A., & Schwartz, E. S. (2001). Valuing American options by simulation: A simple least-squares approach. *The Review of Financial Studies*, 14(1), 113-147.

[17] Medvedev, A., & Scaillet, O. (2010). Pricing American options under stochastic volatility and stochastic interest rates. *Journal of Financial Economics*, 98(1), 145-159.

[18] Nwankwo, C., Ware, T., & Dai, W. (2025). Ensemble deep neural network method for solving free boundary American style stochastic volatility models, 55(2), 158.

[19] Nwankwo, C., Ware, T., & Dai, W. (2025). Explainable deep learning for American rate and volatility models. Research square.

[20] Nwankwo, C., & Dai, W. (2025). A deep learning method for solving American double Heston model.

[21] Pacheco, R. G., Bochie, K., Gilbert, M. S., Couto, R. S., & Campista, M. E. M. (2021). Towards edge computing using early-exit convolutional neural networks. *Information*, 12(10), 431.

[22] Panda, P., Sengupta, A., & Roy, K. (2016). Conditional deep learning for energy-efficient and enhanced pattern recognition. In *Design, Automation & Test in Europe Conference & Exhibition* (pp. 475-480).

[23] Rahmath P, H., Srivastava, V., Chaurasia, K., Pacheco, R. G., & Couto, R. S. (2024). Early-exit deep neural network-a comprehensive survey. *ACM Computing Surveys*, 57(3), 1-37.

[24] Sarti, S., Lomurno, E., & Matteucci, M. (2023). Anticipate, ensemble and prune: Improving convolutional neural networks via aggregated early exits. *Procedia Computer Science*, 222, 519-528.

[25] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(56), 1929-1958.

[26] Sun, T., Zhou, Y., Liu, X., Zhang, X., Jiang, H., Cao, Z., ... & Qiu, X. (2021). Early exiting with ensemble internal classifiers. *arXiv preprint* arXiv:2105.13792.

[27] Teerapittayanon, S., McDanel, B., & Kung, H. T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR* (pp. 2464-2469).

[28] Wołczyk, M., Wójcik, B., Bałazy, K., Podolak, I. T., Tabor, J., Śmieja, M., & Trzcinski, T. (2021). Zero time waste: Recycling predictions in early exit neural networks. *Advances in Neural Information Processing Systems*, 34, 2516-2528.

[29] Yu, F., Wang, D., Shelhamer, E., & Darrell., T. (2018). In *Proceedings of the IEEE conference on computer vision and pattern recognition.* (pp. 2403-2412)