ON DEEP LEARNING FOR OPTION PRICING IN LOCAL VOLATILITY MODELS

Shorokhov Sergey

"Distributed Computing and Grid-technologies in Science and Education"

July 5-9 Dubna

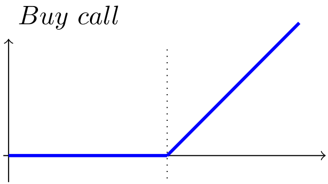
CONTENTS

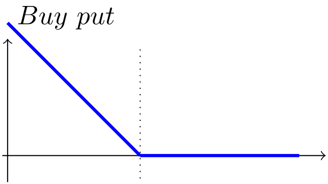
Options in Finance

Option is a financial instrument (contract) with the right, but not the obligation, of the holder (buyer) to buy or sell an underlying asset at a specified strike price on a specified date in the future.

Payoff functions of European call and put options at time $\mathbf{T}$ are equal to

$$\mathbf{c_E}\left(\mathbf{T}\right) = \max\left(\mathbf{S_T} - \mathbf{K}, \mathbf{0}\right), \; \mathbf{p_E}\left(\mathbf{T}\right) = \max\left(\mathbf{K} - \mathbf{S_T}, \mathbf{0}\right)$$





The problem of determination of the fair price of options at $\mathbf{t} < \mathbf{T}$ remained unsolved for a long time.

Black-Scholes Stochastic Model

In Black-Scholes model (BS model) the price of the underlying asset is driven by SDE

$$\frac{\mathbf{dS}}{\mathbf{S}} = \mu\,\mathbf{dt} + \sigma\,\mathbf{dW},\ \mathbf{S}\left(\mathbf{t_0}\right) = \mathbf{S_0} > \mathbf{0},$$

where

- $\mathbf{S}\left(\mathbf{t}\right)$ is the underlying asset price at time $\mathbf{t}$
- μ is constant instanteneous return of the asset
- σ is constant volatility
- $\mathbf{W}$ is a standard Wiener process

BS model is a continuous time constant volatility model.

Black-Scholes-Merton PDE

In 1973 F. Black, M. Scholes and independently R. Merton proved, that in lognormal (BS) model the fair price $\mathbf{u} = \mathbf{u}(\mathbf{S}, \mathbf{t})$ of any derivative is a solution to partial differential equation (PDE)

$$\frac{\partial \mathbf{u}}{\partial \mathbf{t}} + \mathbf{r}\,\mathbf{S}\,\frac{\partial \mathbf{u}}{\partial \mathbf{S}} + \frac{\mathbf{1}}{\mathbf{2}}\sigma^{\mathbf{2}}\,\mathbf{S}^{\mathbf{2}}\,\frac{\partial^{\mathbf{2}} \mathbf{u}}{\partial \mathbf{S}^{\mathbf{2}}} - \mathbf{r}\,\mathbf{u} = \mathbf{0},\ \mathbf{S} > \mathbf{0},\ \mathbf{t} \in [\mathbf{0},\ \mathbf{T}]$$

with appropriate boundary conditions, $\mathbf{r}$ is a risk free rate.

For a European call option the boundary (terminal) condition is

$$\mathbf{u}(\mathbf{S}, \mathbf{T}) = \max(\mathbf{S} - \mathbf{K},\ \mathbf{0}),$$

where $\mathbf{K}$ is the strike-price, $\mathbf{T}$ is the time to maturity.

Black-Scholes Formula

F. Black and M. Scholes obtained the following exact formula for the price of a European call option with strike-price **K** and maturity **T**:

$$\mathbf{u}(\mathbf{S},\mathbf{t},\mathbf{K},\mathbf{T}) = \mathbf{S}\,\mathbf{\Phi}(\mathbf{d}_+) - \mathbf{K}\,\mathbf{e}^{-\mathbf{r}\,(\mathbf{T}-\mathbf{t})}\,\mathbf{\Phi}(\mathbf{d}_-),$$

$$\mathbf{d}_\pm = \frac{\ln\left(\frac{\mathbf{S}}{\mathbf{K}}\right) + \left(\mathbf{r} \pm \frac{\sigma^2}{2}\right)(\mathbf{T}-\mathbf{t})}{\sigma\sqrt{\mathbf{T}-\mathbf{t}}},$$

$$\mathbf{\Phi}(\mathbf{d}) = \frac{\mathbf{1}}{\sqrt{\mathbf{2}\pi}} \int_{-\infty}^{\mathbf{d}} \mathbf{e}^{-\frac{\mathbf{x}^2}{2}}\,\mathbf{dx},$$

$$\mathbf{S} > \mathbf{0},\ \mathbf{t} \in [\mathbf{0},\mathbf{T}].$$

Local Volatility Model and PDE for Derivative Price

Under risk-neutral pricing SDE of a local volatility model is

$$\frac{\mathbf{dS}}{\mathbf{S}} = \mathbf{r}\,\mathbf{dt} + \sigma\left(\mathbf{S},\mathbf{t}\right)\,\mathbf{dW},\ \mathbf{S}\left(\mathbf{t_0}\right) = \mathbf{S_0} > \mathbf{0},$$

where

- $\mathbf{S}\left(\mathbf{t}\right)$ is the asset price at time $\mathbf{t}$
- $\mathbf{r} > \mathbf{0}$ is risk free interest rate
- $\sigma\left(\mathbf{S},\mathbf{t}\right)$ is a volatility function
- $\mathbf{W}$ is a standard Wiener process

Black-Scholes-Merton PDE (for European call option price $\mathbf{c}\left(\mathbf{S},\mathbf{t},\mathbf{K},\mathbf{T}\right)$) is

$$\frac{\partial\mathbf{c}}{\partial\mathbf{t}} + \mathbf{r}\,\mathbf{S}\,\frac{\partial\mathbf{c}}{\partial\mathbf{S}} + \frac{\mathbf{1}}{\mathbf{2}}\sigma^{\mathbf{2}}\left(\mathbf{S},\ \mathbf{t}\right)\,\mathbf{S}^{\mathbf{2}}\,\frac{\partial^{\mathbf{2}}\mathbf{c}}{\partial\mathbf{S}^{\mathbf{2}}} - \mathbf{r}\,\mathbf{c} = \mathbf{0}$$

with boundary (terminal) condition

$$\mathbf{c}\left(\mathbf{S},\mathbf{T},\mathbf{K},\mathbf{T}\right) = \max(\mathbf{S} - \mathbf{K},\ \mathbf{0}).$$

Shifted Lognormal Model

- Shifted lognormal model (D. Brigo, F. Mercurio, 2000) is quite close to Black-Scholes model:

$$\mathbf{dS} = \mathbf{r}\,\mathbf{S}\,\mathbf{dt} + \sigma\,\left(\mathbf{S} - \alpha\,\mathbf{e}^{\mathbf{r}\,\mathbf{t}}\right)\,\mathbf{dW},\ \mathbf{S}\,(\mathbf{t_0}) = \mathbf{S_0} > \alpha\,\mathbf{e}^{\mathbf{r}\,\mathbf{t_0}}$$

- In shifted lognormal model the asset price is may be negative for $\alpha < \mathbf{0}$.
- In shifted lognormal model European call option price is equal to

$$\mathbf{c}\,(\mathbf{S}, \mathbf{t}, \mathbf{K}, \mathbf{T}) = \left(\mathbf{S} - \alpha\,\mathbf{e}^{\mathbf{r}\,\mathbf{t}}\right)\boldsymbol{\Phi}(\mathbf{d}_+) - \left(\mathbf{K} - \alpha\,\mathbf{e}^{\mathbf{r}\,\mathbf{T}}\right)\mathbf{e}^{-\mathbf{r}(\mathbf{T}-\mathbf{t})}\boldsymbol{\Phi}(\mathbf{d}_-),$$

$$\mathbf{d}_\pm = \frac{\ln\left(\frac{\mathbf{S}-\alpha\,\mathbf{e}^{\mathbf{r}\,\mathbf{t}}}{\mathbf{K}-\alpha\,\mathbf{e}^{\mathbf{r}\,\mathbf{T}}}\right) + \left(\mathbf{r} \pm \frac{\sigma^{\mathbf{2}}}{\mathbf{2}}\right)(\mathbf{T} - \mathbf{t})}{\sigma\sqrt{\mathbf{T} - \mathbf{t}}}$$

Normal (Ornstein–Uhlenbeck) Model

- Generally, Ornstein–Uhlenbeck model (G.Uhlenbeck, L.Ornstein, 1930) is the model with mean reversion behaviour

$$\mathbf{dx} = \theta\,(\mathbf{m} - \mathbf{x})\,\mathbf{dt} + \sigma\,\mathbf{dW}.$$

- Normal model can be derived from Ornstein–Uhlenbeck model:

$$\mathbf{dS} = \mathbf{r}\,\mathbf{S}\,\mathbf{dt} + \sigma\,\mathbf{dW},\ \mathbf{S}\,(\mathbf{t_0}) = \mathbf{S_0} > \mathbf{0}$$

- In normal model the price of underlying asset $\mathbf{S}\,(\mathbf{t})$ can be negative.
- In normal model European call option price is equal to

$$\mathbf{c}\,(\mathbf{S},\mathbf{t},\mathbf{K},\mathbf{T}) = \left(\mathbf{S} - \mathbf{K}\mathbf{e}^{-\mathbf{r}(\mathbf{T}-\mathbf{t})}\right)\boldsymbol{\Phi}\left(\mathbf{d}_+^*\right) + \frac{\sigma\sqrt{\mathbf{e}^{\mathbf{2r}(\mathbf{T}-\mathbf{t})} - \mathbf{1}}}{\mathbf{2}\sqrt{\pi\mathbf{r}}}\mathbf{e}^{-\mathbf{r}(\mathbf{T}-\mathbf{t})}\mathbf{e}^{-\frac{\left(\mathbf{d}_+^*\right)^\mathbf{2}}{\mathbf{2}}},$$

$$\mathbf{d}_+^* = \sqrt{\mathbf{2r}}\,\frac{\mathbf{e}^{\mathbf{r}(\mathbf{T}-\mathbf{t})}\mathbf{S} - \mathbf{K}}{\sigma\sqrt{\mathbf{e}^{\mathbf{2r}(\mathbf{T}-\mathbf{t})} - \mathbf{1}}}$$

CEV (Constant Elasticity of Variance) Model

- CEV model was introduced by J. Cox, S. Ross (1976)

$$\mathbf{dS = r\,S\,dt} + \sigma\,\mathbf{S}^{\beta/\mathbf{2}}\,\mathbf{dW},\ \beta \neq \mathbf{2}.$$

- In CEV model asset price is positive and noncentral chi-square distributed.
- In CEV model ($\beta > \mathbf{2}$) European call option price is equal to

$$\mathbf{c\,(S,t,K,T) = S\,Q}\left(\mathbf{2x};\frac{\mathbf{2}}{\beta-\mathbf{2}},\mathbf{2y}\right) - \mathbf{K}\mathbf{e}^{-\mathbf{r(T-t)}}\left(\mathbf{1} - \mathbf{Q}\left(\mathbf{2y};\mathbf{2}+\frac{\mathbf{2}}{\beta-\mathbf{2}},\mathbf{2x}\right)\right),$$

$$\mathbf{x} = \mathbf{k}^*\mathbf{S}^{\mathbf{2}-\beta}\mathbf{e}^{\mathbf{r}(\mathbf{2}-\beta)(\mathbf{T-t})},\ \mathbf{y} = \mathbf{k}^*\mathbf{K}^{\mathbf{2}-\beta},\ \mathbf{k}^* = \frac{\mathbf{2r}}{\sigma^{\mathbf{2}}\,(\mathbf{2}-\beta)\left(\mathbf{e}^{\mathbf{r}(\mathbf{2}-\beta)(\mathbf{T-t})} - \mathbf{1}\right)},$$

$\mathbf{Q}$ is a complementary distribution function of noncentral chi-square distribution.

Hyperbolic Sine Model

- In hyperbolic sine model (S. Shorokhov, M. Fomin, 2020) asset price is driven by SDE

$$\mathbf{dS = r\,S\,dt + \sqrt{2\,r\,S^2 + \lambda^2}\,dW,\ r > 0,\ \lambda > 0.}$$

- In hyperbolic sine model asset price may be negative.
- In hyperbolic sine model European call option price is equal to

$$\mathbf{c\,(S,t,K,T) = \frac{1}{2}S\left(\Phi\left(\sqrt{2r\,(T-t)} - K^*\right) + \Phi\left(-\sqrt{2r\,(T-t)} - K^*\right)\right) +}$$

$$\mathbf{+\frac{1}{2}\sqrt{\frac{\lambda^2}{2r} + S^2}\left(\Phi\left(\sqrt{2r\,(T-t)} - K^*\right) - \Phi\left(-\sqrt{2r\,(T-t)} - K^*\right)\right) -}$$

$$\mathbf{-e^{-r(T-t)}K\Phi\,(-K^*)\,,\ K^* = \frac{\operatorname{arsinh}\left(\frac{\sqrt{2r}}{\lambda}K\right) - \operatorname{arsinh}\left(\frac{\sqrt{2r}}{\lambda}S\right)}{\sqrt{2r\,(T-t)}}}$$

Deep Learning and PDE

Deep learning is a family of machine learning methods based on artificial neural networks.

Mathematically, an artificial neural network is a directed graph with vertices representing neurons and edges representing links and with input to each neuron being a function of a weighted sum of the output of all neurons that are connected to its incoming edges

$$\mathbf{f}\left(\mathbf{x};\theta\right)=\psi_{\mathbf{d}}\left(\ldots\psi_{\mathbf{2}}\left(\psi_{\mathbf{1}}\left(\mathbf{x}\right)\right)\right),\ \psi_{\mathbf{i}}\left(\mathbf{x}\right)=\sigma\left(\mathbf{w}^{(\mathbf{i})}\,\mathbf{x}+\mathbf{b}^{(\mathbf{i})}\right)$$

Here, each layer of the network is represented by a function $\psi_{\mathbf{i}}$, incorporating the weighted sums of previous inputs and activations to connected outputs. The number of layers $\mathbf{d}$ is referred to as the depth of the neural network and the number of neurons in a layer represents the width of that particular layer.

The goal of deep learning is to find the parameter set $\theta=\left\{\mathbf{w}^{(\mathbf{i})},\mathbf{b}^{(\mathbf{i})}\right\}_{\mathbf{i=1}}^{\mathbf{d}}$ that minimizes the loss function $\mathbf{L}(\theta)$, which determines the performance of a given parameter set θ.

Deep Galerkin Method (DGM)

DGM algorithm was introduced by J.Sirignano, K.Spiliopoulos (2018) in article «DGM: A deep learning algorithm for solving partial differential equations» in Journal of Computational Physics, vol.375, pp.1339-1364.

In DGM algorithm PDE solution is approximated with a deep neural network which is trained to satisfy the differential operator and initial/boundary conditions. The following PDE were considered:

- high-dimensional free boundary PDE (American option pricing)
- High-dimensional Hamilton-Jacobi-Bellman PDE
- Burgers' PDE

Computations were performed using the Blue Waters supercomputer at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (https://bluewaters.ncsa.illinois.edu).

The unknown European call option price $\mathbf{u}(\mathbf{t}, \mathbf{S})$, determined in the domain $[\mathbf{0}, \mathbf{T}] \times \boldsymbol{\Omega}$, $\boldsymbol{\Omega} = [\mathbf{s_l}, \mathbf{s_h}] \subset \mathbb{R}$, is a solution to the following boundary problem

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial \mathbf{t}} + \mathbf{rS}\frac{\partial \mathbf{u}}{\partial \mathbf{S}} + \frac{\mathbf{1}}{\mathbf{2}}\sigma^{\mathbf{2}}(\mathbf{t}, \mathbf{S})\,\mathbf{S}^{\mathbf{2}}\frac{\partial^{\mathbf{2}} \mathbf{u}}{\partial \mathbf{S}^{\mathbf{2}}} - \mathbf{ru} = \mathbf{0}, & (\mathbf{t}, \mathbf{S}) \in [\mathbf{0}, \mathbf{T}] \times \boldsymbol{\Omega} \\ \mathbf{u}(\mathbf{T}, \mathbf{S}) = \max(\mathbf{S} - \mathbf{K}, \mathbf{0}), & \mathbf{S} \in \boldsymbol{\Omega} \end{cases}$$

The function $\mathbf{u}(\mathbf{t}, \mathbf{S})$ is approximated with a deep neural network $\mathbf{f}(\mathbf{t}, \mathbf{S}; \theta)$, where θ are the neural network's parameters. The loss function (error) is

$$\mathbf{L}(\mathbf{f}) = \left\| \frac{\partial \mathbf{f}}{\partial \mathbf{t}} + \mathbf{rS}\frac{\partial \mathbf{f}}{\partial \mathbf{S}} + \frac{\mathbf{1}}{\mathbf{2}}\sigma^{\mathbf{2}}(\mathbf{t}, \mathbf{S})\,\mathbf{S}^{\mathbf{2}}\frac{\partial^{\mathbf{2}} \mathbf{f}}{\partial \mathbf{S}^{\mathbf{2}}} - \mathbf{rf} \right\|^{\mathbf{2}}_{[\mathbf{0}, \mathbf{T}] \times \boldsymbol{\Omega}, \nu_{\mathbf{1}}} +$$

$$+ \left\| \mathbf{f} - \max(\mathbf{S} - \mathbf{K}, \mathbf{0}) \right\|^{\mathbf{2}}_{\boldsymbol{\Omega}, \nu_{\mathbf{2}}},$$

where $\|\mathbf{f}\|^{\mathbf{2}}_{\mathbf{X}, \rho} = \int_{\mathbf{X}} |\mathbf{f}(\mathbf{y})|^{\mathbf{2}} \rho(\mathbf{y})\, \mathbf{dy}$, $\rho(\mathbf{y})$ is a positive probability density on $\mathbf{X}$.

The goal is to find a set of parameters θ such that the function $\mathbf{f}(\mathbf{t}, \mathbf{S}; \theta)$ minimizes the loss function (error) $\mathbf{L}(\mathbf{f})$.

DGM Algorithm for Option Pricing in LVM

DGM is using a neural network of special architecture and stochastic gradient descent on a sequence of time and price points drawn at random from $[\mathbf{0}, \mathbf{T}] \times \boldsymbol{\Omega}$.

1. Choose initial parameter set $\theta_{\mathbf{0}}$ and learning rate $\alpha_{\mathbf{0}}$
2. Generate random points $(\mathbf{t_n}, \mathbf{S_n})$ from $[\mathbf{0}, \mathbf{T}] \times \boldsymbol{\Omega}$ with distribution $\nu_{\mathbf{1}}$ and $\mathbf{S}_{\mathbf{n}}^{'}$ from Ω with distribution $\nu_{\mathbf{2}}$.
3. Calculate the loss function (error) $\mathbf{L}\left(\theta_{\mathbf{n}}, \xi_{\mathbf{n}}\right)$ at the randomly sampled points $\xi_{\mathbf{n}} = \left\{(\mathbf{t_n}, \mathbf{S_n}), \mathbf{S}_{\mathbf{n}}^{'}\right\}$, where:

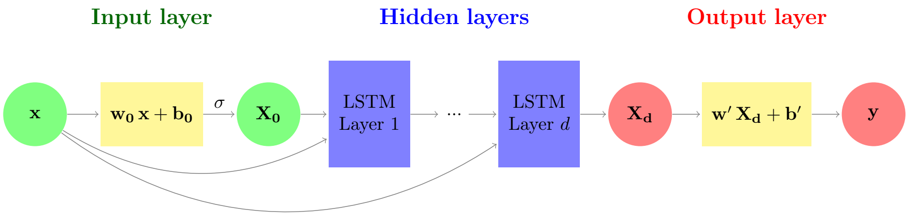
$$\mathbf{L}\left(\theta_{\mathbf{n}}, \xi_{\mathbf{n}}\right) = \left(\frac{\partial \mathbf{f}(\mathbf{t_n}, \mathbf{S_n}; \theta_{\mathbf{n}})}{\partial \mathbf{t}} + \mathbf{r}\mathbf{S_n}\frac{\partial \mathbf{f}(\mathbf{t_n}, \mathbf{S_n}; \theta_{\mathbf{n}})}{\partial \mathbf{S}} + \frac{\mathbf{1}}{\mathbf{2}}\sigma^{\mathbf{2}}\left(\mathbf{t_n}, \mathbf{S_n}\right)\mathbf{S}_{\mathbf{n}}^{\mathbf{2}}\frac{\partial^{\mathbf{2}} \mathbf{f}(\mathbf{t_n}, \mathbf{S_n}; \theta_{\mathbf{n}})}{\partial \mathbf{S}^{\mathbf{2}}} - \right.$$
$$\left. - \mathbf{r}\,\mathbf{f}\left(\mathbf{t_n}, \mathbf{S_n}; \theta_{\mathbf{n}}\right)\right)^{\mathbf{2}} + \left(\mathbf{f}\left(\mathbf{t_n}, \mathbf{S_n}; \theta_{\mathbf{n}}\right) - \max\left(\mathbf{S_n} - \mathbf{K}, \mathbf{0}\right)\right)^{\mathbf{2}}$$

4. Take a gradient descent step at the random point $\xi_{\mathbf{n}}$ with adaptive algorythm Adam for the learning rate $\alpha_{\mathbf{n}}$:

$$\theta_{\mathbf{n+1}} = \theta_{\mathbf{n}} - \alpha_{\mathbf{n}} \nabla_{\theta} \mathbf{L}\left(\theta_{\mathbf{n}}, \xi_{\mathbf{n}}\right)$$

5. Repeat steps 2-4 until convergence criterion $\|\theta_{\mathbf{n+1}} - \theta_{\mathbf{n}}\| < \epsilon$ is satisfied.

The architecture of DGM neural network is similar to LSTM and Highway Networks. It consists of three types of layers: an input layer, hidden (LSTM) layers and an output layer.



Each LSTM layer takes as an input the original mini-batch inputs $\mathbf{x} = (\mathbf{t}, \mathbf{S})$ (the set of randomly sampled time-price points) and the output of the previous LSTM layer. The process results in an output $\mathbf{y}$ which consists of the neural network approximation of the desired option price $\mathbf{u}$ evaluated at the mini-batch points.

Calculations in DGM Network Layers

In the input layer, the original inputs $\mathbf{x}$ are transformed into the output $\mathbf{X_0}$

$$\mathbf{X_0} = \sigma\left(\mathbf{w_0}\,\mathbf{x} + \mathbf{b_0}\right)$$

with a nonlinear activation function σ and input layer parameters $\mathbf{w_0}$ and $\mathbf{b_0}$.

In LSTM layers, the original inputs $\mathbf{x}$ along with the output of the previous layer $\mathbf{X_{i-1}}$ are transformed through a series of operations:

$$\begin{aligned}
\mathbf{Z_i} &= \sigma\left(\mathbf{u_i^z}\,\mathbf{x} + \mathbf{w_i^z}\,\mathbf{X_{i-1}} + \mathbf{b_i^z}\right), \quad & \mathbf{G_i} &= \sigma\left(\mathbf{u_i^g}\,\mathbf{x} + \mathbf{w_i^g}\,\mathbf{X_{i-1}} + \mathbf{b_i^g}\right), \\
\mathbf{R_i} &= \sigma\left(\mathbf{u_i^r}\,\mathbf{x} + \mathbf{w_i^r}\,\mathbf{X_{i-1}} + \mathbf{b_i^r}\right), \quad & \mathbf{H_i} &= \sigma\left(\mathbf{u_i^h}\,\mathbf{x} + \mathbf{w_i^h}\,\left(\mathbf{X_{i-1}} \odot \mathbf{R_i}\right) + \mathbf{b_i^h}\right),
\end{aligned}$$

where $\odot$ denotes element-wise multiplication, $\mathbf{u_i^z}, \mathbf{w_i^z}, \mathbf{b_i^z}, \mathbf{u_i^g}, \mathbf{w_i^g}, \mathbf{b_i^g}, \mathbf{u_i^r}, \mathbf{w_i^r}, \mathbf{b_i^r}$, $\mathbf{u_i^h}, \mathbf{w_i^h}, \mathbf{b_i^h}$ are LSTM layer parameters, and the outputs of LSTM layer are

$$\mathbf{X_i} = \left(\mathbf{1} - \mathbf{G_i}\right) \odot \mathbf{H_i} + \mathbf{Z_i} \odot \mathbf{X_{i-1}}.$$

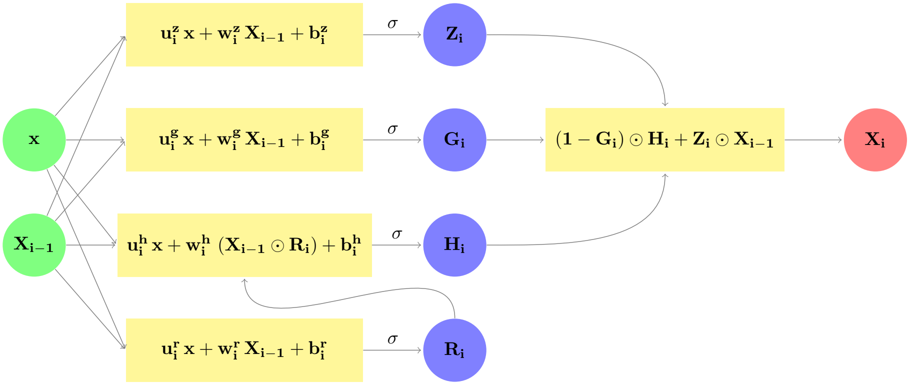
In the output layer, the outputs of the last LSTM layer $\mathbf{X_d}$ are transformed into the neural network outputs $\mathbf{y}$ via a linear transform

$$\mathbf{y} = f\left(\mathbf{x};\,\theta\right) = \mathbf{w}'\,\mathbf{X_d} + \mathbf{b}',$$

where $\mathbf{w}'$ and $\mathbf{b}'$ are the output layer parameters.

Architecture of LSTM layer

Each LSTM layer contains 8 weight matrices and 4 bias vectors:



Parameter Set of DGM Neural Network

Let $\mathbf{d}$ be the number of hidden layers and $\mathbf{N}$ be the number of neurons (nodes) in each hidden layer of DGM Network.

In the input layer, the weight parameter $\mathbf{w_0}$ is of shape $\mathbf{2} \times \mathbf{N}$ and the bias parameter $\mathbf{b_0}$ is of shape $\mathbf{1} \times \mathbf{N}$.

In LSTM layers, the weight parameters $\mathbf{u_i^z}$,$\mathbf{u_i^g}$,$\mathbf{u_i^r}$,$\mathbf{u_i^h}$ are of shape $\mathbf{2} \times \mathbf{N}$, the weight parameters $\mathbf{w_i^z}$,$\mathbf{w_i^g}$, $\mathbf{w_i^r}$, $\mathbf{w_i^h}$ are of shape $\mathbf{N} \times \mathbf{N}$, the bias parameters $\mathbf{b_i^z}$, $\mathbf{b_i^g}$, $\mathbf{b_i^r}$, $\mathbf{b_i^h}$ are of shape $\mathbf{1} \times \mathbf{N}$.

In the output layer, the parameter $\mathbf{w'}$ is of shape $\mathbf{N} \times \mathbf{1}$ and $\mathbf{b'}$ is a scalar parameter.

So the total number of parameters in DGM network is equal to

$$|\theta| = \mathbf{3\,N} + \mathbf{d}\,\left(\mathbf{8\,N} + \mathbf{4\,N^2} + \mathbf{4\,N}\right) + \mathbf{N} + \mathbf{1} = \mathbf{4\,d}\,\left(\mathbf{N} + \mathbf{1}\right)^{\mathbf{2}} + \mathbf{4\,N} + \mathbf{1}$$

In experiments we will use **3** hidden layers and **50** neurons per hidden layer and the number of parameters will be **31 413**.

NN Approximation of Option Price in HS Model

Let the underlying asset price be driven by SDE of HS model:

$$\mathbf{dS = r\,S\,dt + \sqrt{2\,r\,S^2 + \lambda^2}\,dW,\ r > 0,\ \lambda > 0.}$$

BSM PDE for HS model is

$$\mathcal{L}(\mathbf{u}) \equiv \frac{\partial \mathbf{u}}{\partial \mathbf{t}} + \mathbf{r\,S}\,\frac{\partial \mathbf{u}}{\partial \mathbf{S}} + \frac{\mathbf{1}}{\mathbf{2}}\left(\mathbf{2\,r\,S^2} + \lambda^{\mathbf{2}}\right)\frac{\partial^{\mathbf{2}} \mathbf{u}}{\partial \mathbf{S^2}} - \mathbf{r\,u} = \mathbf{0}.$$

The goal is to find a NN estimate of a European option price and compare it with known exact closed form price formula.

The computational experiments are performed with the following parameters:

- The number of hidden layers is 3 ($\mathbf{d = 3}$)
- The number of nodes (neurons) per hidden layer is 50 ($\mathbf{N = 50}$)
- Number of training stages is 100 with 10 SGD steps in each stage
- $\mathbf{r = 0.05}$, $\lambda = \mathbf{0.25}$

NN approximation for HS model is implemented with TensorFlow framework.

Computations are performed using MacBook Pro with Intel Core i9 processor (I9-9880H, 8 cores) and discrete graphics card AMD Radeon Pro 5500M (1536 shader processors).

Case #1

- Inputs are price-time points $(\mathbf{S}, \mathbf{t})$, $\mathbf{S} \in (\mathbf{0}, \mathbf{2K})$, $\mathbf{t} \in (\mathbf{0}, \mathbf{T})$
- Strike price $\mathbf{K}$ and expiration time $\mathbf{T}$ are fixed $(\mathbf{K} = \mathbf{50},\ \mathbf{T} = \mathbf{1})$
- Loss function:

$$\mathbf{L}\left(\mathbf{u}\right) = \|\mathcal{L}(\mathbf{u})\|^{\mathbf{2}}_{[\mathbf{0},\mathbf{T}]\times\boldsymbol{\Omega},\nu_\mathbf{1}} + \|\mathbf{u} - \max\left(\mathbf{S} - \mathbf{K}, \mathbf{0}\right)\|^{\mathbf{2}}_{\boldsymbol{\Omega},\nu_\mathbf{2}}$$

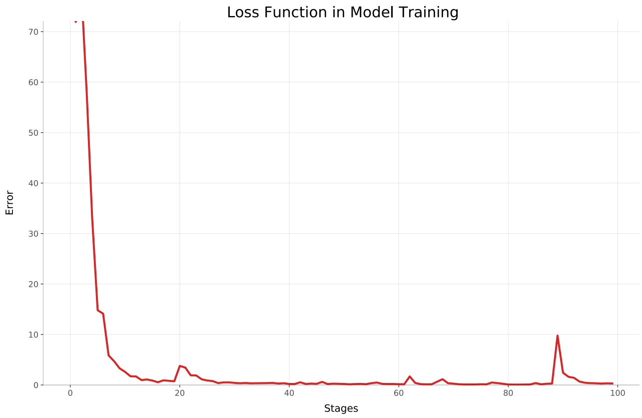
- Implementation: TensorFlow 1.15
-

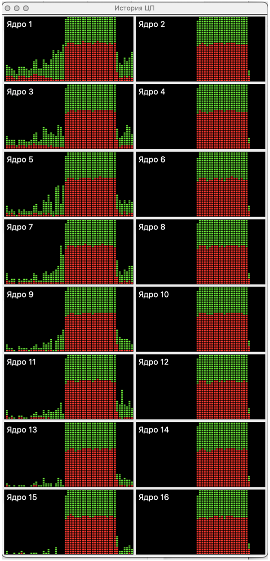
Training time	131 s
MSE	0.1858
MAE	0.2310
R^2	99.93%

Case #1 – Loss Function Implementation

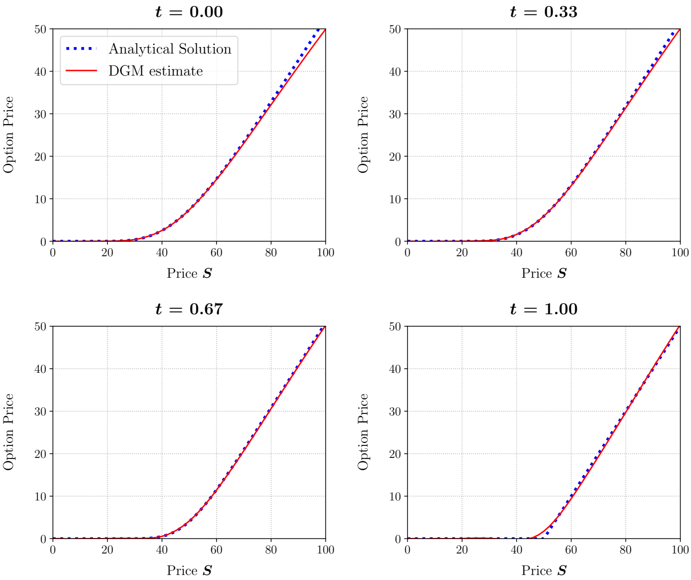
```python
# Loss function for BSM PDE in HS model
def loss(model, t_int, S_int, t_term, S_term):
    ''' Compute total loss for training.

    Args:
        model:  DGM model object
        t_int:  sampled time points in the interior of the option price domain
        S_int:  sampled price points in the interior of the option price domain
        t_term: sampled time points at terminal point (vector of terminal times)
        S_term: sampled price points at terminal time
    '''
    # Loss term #1: PDE
    # option price value and derivatives at sampled points
    V    = model(t_int, S_int)
    V_t  = tf.gradients(V, t_int)[0]
    V_s  = tf.gradients(V, S_int)[0]
    V_ss = tf.gradients(V_s, S_int)[0]
    # LVM model dependent code
    diff_V = V_t + 0.5*(lamb**2+2.*r*S_int**2)*V_ss + r*S_int*V_s - r*V

    # average L2-norm of differential operator
    L1 = tf.reduce_mean(tf.square(diff_V))

    # Loss term #2: terminal condition
    target_payoff = tf.nn.relu(S_term - K)
    fitted_payoff = model(t_term, S_term)

    L2 = tf.reduce_mean( tf.square(fitted_payoff - target_payoff) )

    return L1 + L2
```

Case #1 – Neural Network Training

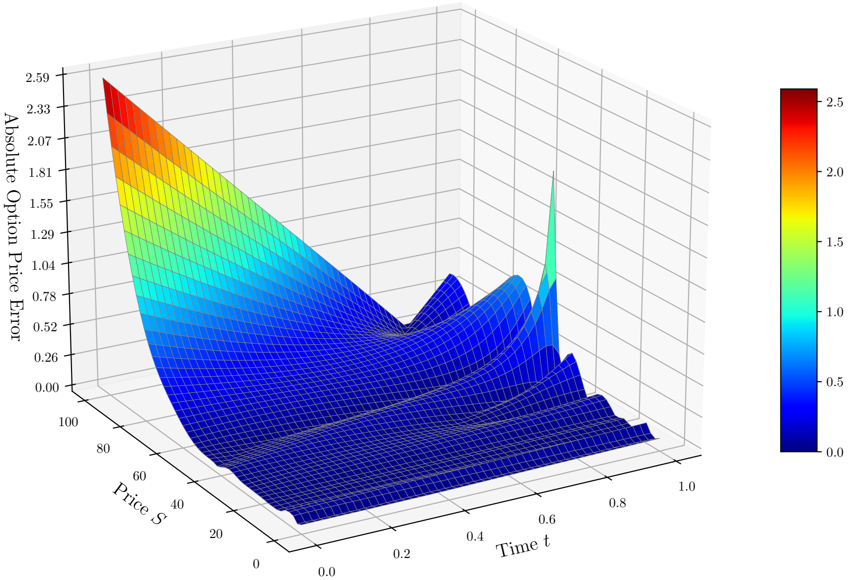




Case #1 – Exact and Predicted Option Prices



Case #1 – 3-d Surface of Absolute Error



Case #2

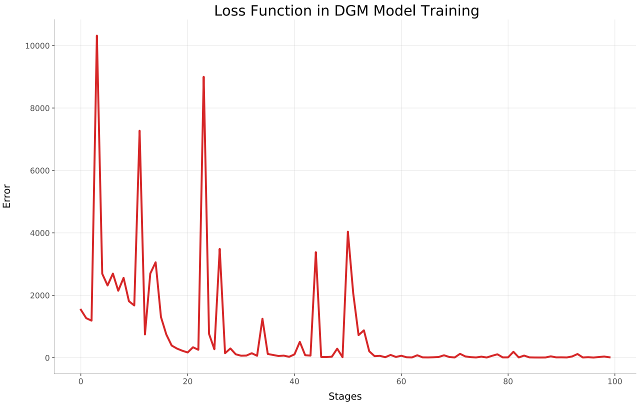
- Inputs are price-time-strike price-expiration time points $(\mathbf{S}, \mathbf{t}, \mathbf{K}, \mathbf{T})$, $\mathbf{S} \in (\mathbf{0}, \mathbf{2K}_{\max})$, $\mathbf{t} \in (\mathbf{0}, \mathbf{T}_{\max})$, $\mathbf{K} \in (\mathbf{0}, \mathbf{K}_{\max})$, $\mathbf{T} \in (\mathbf{t}, \mathbf{T}_{\max})$
- $\mathbf{K}_{\max} = \mathbf{50}$, $\mathbf{T}_{\max} = \mathbf{1}$
- Loss function:

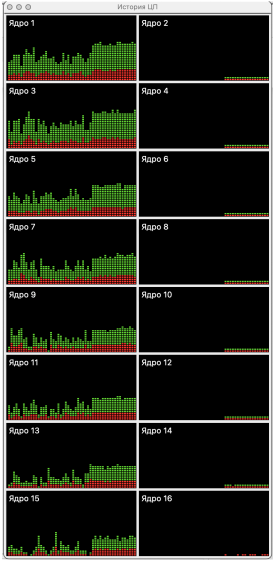
$$\mathbf{L}(\mathbf{u}) = \|\mathcal{L}(\mathbf{u})\|^{\mathbf{2}} + \|\mathbf{u} - \max(\mathbf{S} - \mathbf{K}, \mathbf{0})\|^{\mathbf{2}}$$

- Implementation: TensorFlow 2.4.1
-

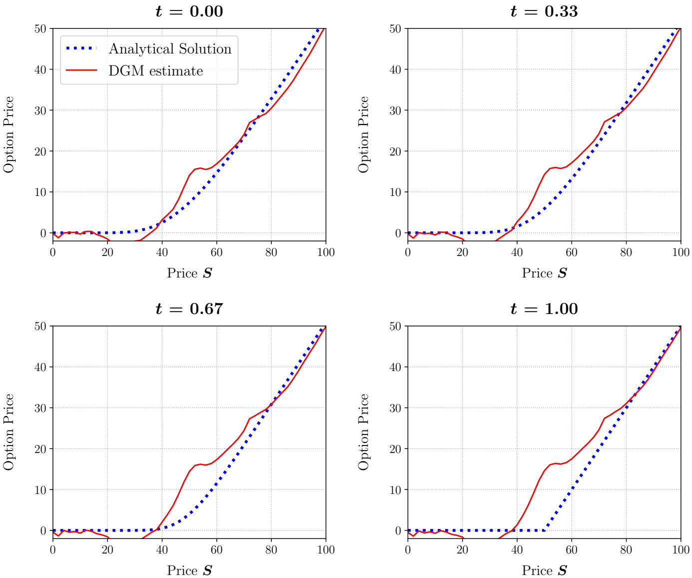
Training time	167 s
MSE	14.2007
MAE	02.7177
R^2	94.90%

Case #2 – Neural Network Training

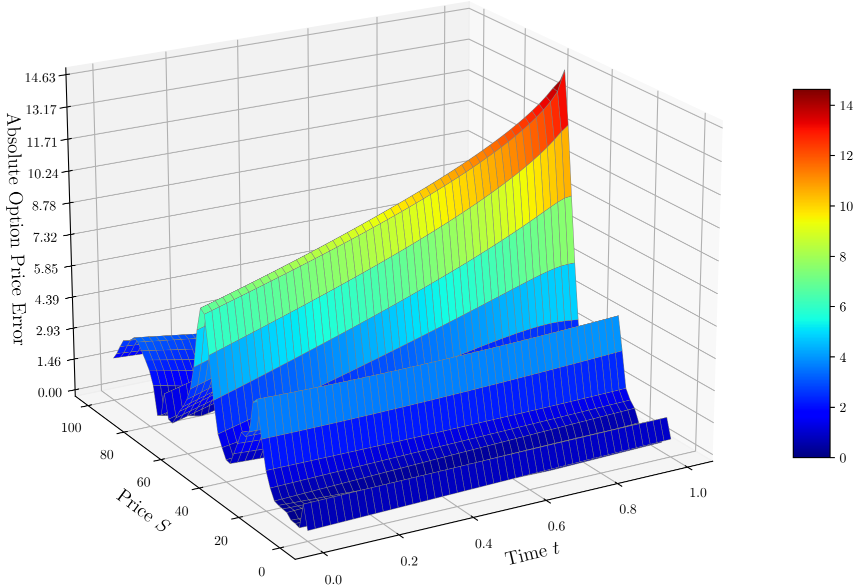




Case #2 – Exact and Predicted Option Prices



Case #2 – 3-d Surface of Absolute Error



Case #3

- Inputs are price-time-strike price-expiration time points $(\mathbf{S}, \mathbf{t}, \mathbf{K}, \mathbf{T})$, $\mathbf{S} \in (\mathbf{0}, \mathbf{2K}_{\max})$, $\mathbf{t} \in (\mathbf{0}, \mathbf{T}_{\max})$, $\mathbf{K} \in (\mathbf{0}, \mathbf{K}_{\max})$, $\mathbf{T} \in (\mathbf{t}, \mathbf{T}_{\max})$
- $\mathbf{K}_{\max} = \mathbf{50}$, $\mathbf{T}_{\max} = \mathbf{1}$
- Loss function:

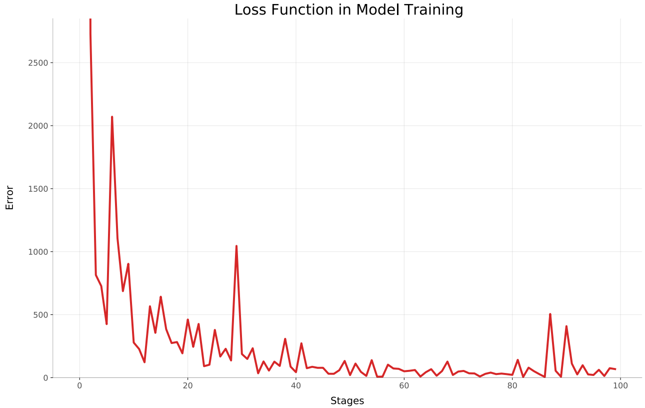
$$\mathbf{L}\left(\mathbf{u}\right) = \left\|\mathcal{L}(\mathbf{u})\right\|^{\mathbf{2}} + \left\|\mathcal{L}^{*}\left(\mathbf{u}\right)\right\|^{\mathbf{2}} + \left\|\mathbf{u} - \max\left(\mathbf{S} - \mathbf{K}, \mathbf{0}\right)\right\|^{\mathbf{2}},$$

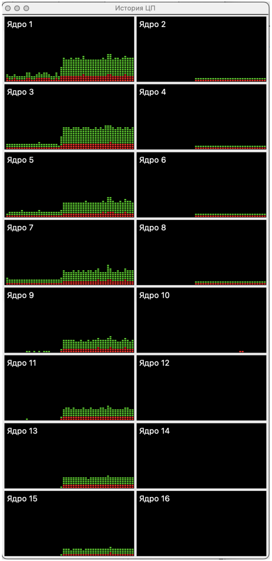
$$\mathcal{L}^{*}\left(\mathbf{u}\right) \equiv \frac{\partial \mathbf{u}}{\partial \mathbf{T}} + \mathbf{r}\,\mathbf{K}\,\frac{\partial \mathbf{u}}{\partial \mathbf{K}} - \frac{\mathbf{1}}{\mathbf{2}}\left(\mathbf{2}\,\mathbf{r}\,\mathbf{K}^{\mathbf{2}} + \lambda^{\mathbf{2}}\right)\frac{\partial^{\mathbf{2}} \mathbf{u}}{\partial \mathbf{K}^{\mathbf{2}}}.$$

- Implementation: TensorFlow 2.4.1
-

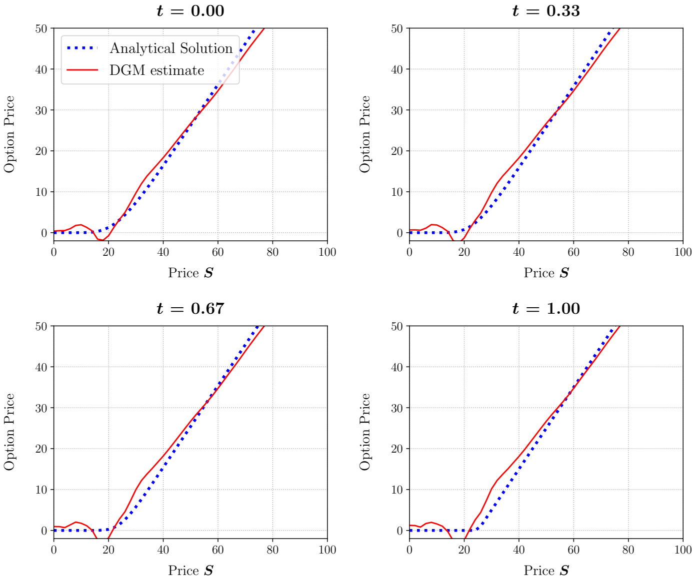
Training time	310 s
MSE	35.8328
MAE	5.3346
R^2	86.06%

Case – #3 Neural Network Training





Case #3 – Exact and Predicted Option Prices



Case #3 – 3-d Surface of Absolute Error