Masters Project Presentation

Deep Galerkin Method for Solving European-Option Pricing Model

Akash Debnath
Roll No:- 212123004

Department of Mathematics
Indian Institute of Technology Guwahati

April 28, 2023

Table of Contents

Introduction

- In Artificial intelligence (AI) one of the subfield is Deep Learning and also Machine learning. Machine learning focuses on enabling machines to learn from data and improve their performance on a specific task over time.

Introduction

- In Artificial intelligence (AI) one of the subfield is Deep Learning and also Machine learning. Machine learning focuses on enabling machines to learn from data and improve their performance on a specific task over time.
- The field of machine learning has grown rapidly in recent years, thanks to advances in computing power, data storage, and the availability of vast amounts of data.

Introduction

- In Artificial intelligence (AI) one of the subfield is Deep Learning and also Machine learning. Machine learning focuses on enabling machines to learn from data and improve their performance on a specific task over time.
- The field of machine learning has grown rapidly in recent years, thanks to advances in computing power, data storage, and the availability of vast amounts of data.
- It has applications in a wide range of fields, from natural language processing and computer vision to robotics and self-driving cars.

Introduction

- In Artificial intelligence (AI) one of the subfield is Deep Learning and also Machine learning. Machine learning focuses on enabling machines to learn from data and improve their performance on a specific task over time.
- The field of machine learning has grown rapidly in recent years, thanks to advances in computing power, data storage, and the availability of vast amounts of data.
- It has applications in a wide range of fields, from natural language processing and computer vision to robotics and self-driving cars.

Introduction

- In Artificial intelligence (AI) one of the subfield is Deep Learning and also Machine learning. Machine learning focuses on enabling machines to learn from data and improve their performance on a specific task over time.
- The field of machine learning has grown rapidly in recent years, thanks to advances in computing power, data storage, and the availability of vast amounts of data.
- It has applications in a wide range of fields, from natural language processing and computer vision to robotics and self-driving cars.
- **Deep Learning** is a subfield of Machine Learning and AI, that is concerned with developing algorithms that can learn from large amounts of data. It involves building and training neural networks, which are computational models that are inspired by the structure and function of the human brain. And deep learning we used various fields, like natural language processing, speech recognition, image processing and many more.

Introduction

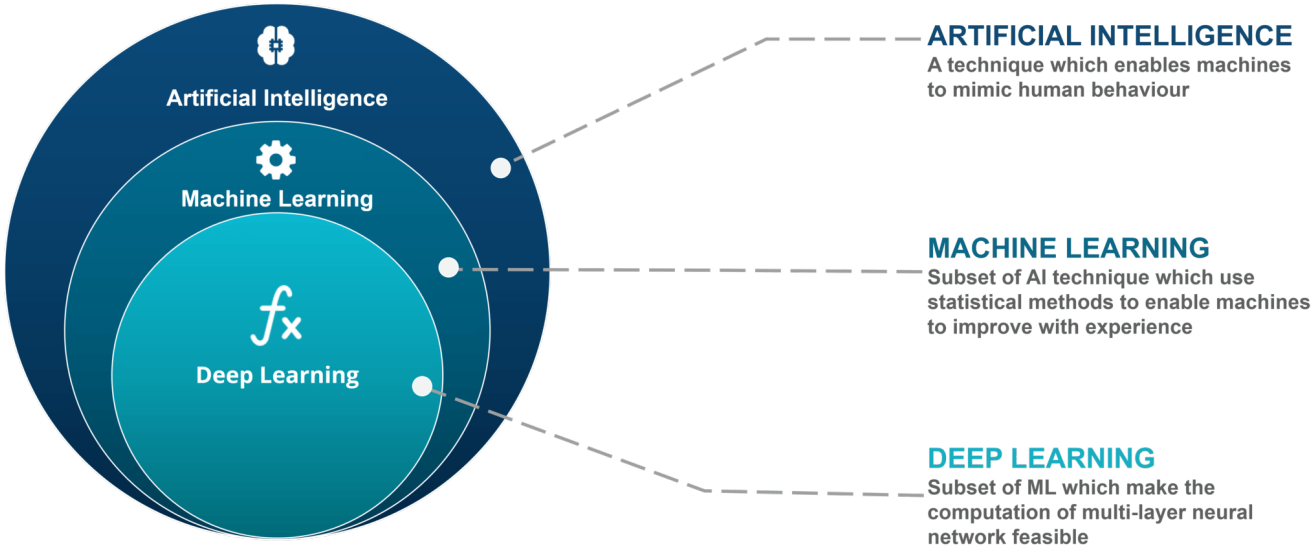


Figure: AI vs ML vs Deep-Learning

Introduction

Applications

- Image and Speech Recognition: Machine learning algorithms can be used to identify and recognize objects in images and videos, as well as transcribe and translate speech.

Introduction

Applications

- Image and Speech Recognition: Machine learning algorithms can be used to identify and recognize objects in images and videos, as well as transcribe and translate speech.
- Natural Language Processing: Machine learning algorithms can be used to analyze and understand natural language, allowing for the development of chatbots and virtual assistants.

Introduction

Applications

- Image and Speech Recognition: Machine learning algorithms can be used to identify and recognize objects in images and videos, as well as transcribe and translate speech.
- Natural Language Processing: Machine learning algorithms can be used to analyze and understand natural language, allowing for the development of chatbots and virtual assistants.
- Fraud Detection: Machine learning algorithms can be used to identify fraudulent activities in real-time, such as credit card fraud or identity theft.

Introduction

Applications

- Image and Speech Recognition: Machine learning algorithms can be used to identify and recognize objects in images and videos, as well as transcribe and translate speech.
- Natural Language Processing: Machine learning algorithms can be used to analyze and understand natural language, allowing for the development of chatbots and virtual assistants.
- Fraud Detection: Machine learning algorithms can be used to identify fraudulent activities in real-time, such as credit card fraud or identity theft.
- Predictive Analytics: Machine learning algorithms can be used to analyze historical data and predict future outcomes, such as sales forecasting, customer churn, or demand forecasting.

Introduction

- The **Deep Galerkin Method** (DGM) is a Deep learning-based approach for solving PDEs.The DGM combines the strengths of deep learning and traditional numerical methods.

Introduction

- The **Deep Galerkin Method** (DGM) is a Deep learning-based approach for solving PDEs.The DGM combines the strengths of deep learning and traditional numerical methods.
- In the DGM, a neural network is trained to approximate the solution to a differential equation on a set of discretized points. The neural network takes in the input variables (e.g., time and space) and returns an approximation of the solution.

Introduction

- The **Deep Galerkin Method** (DGM) is a Deep learning-based approach for solving PDEs.The DGM combines the strengths of deep learning and traditional numerical methods.
- In the DGM, a neural network is trained to approximate the solution to a differential equation on a set of discretized points. The neural network takes in the input variables (e.g., time and space) and returns an approximation of the solution.
- The **Black-Scholes model** is a mathematical model used to estimate the price of financial instruments, such as stock options. It was developed by Fischer Black and Myron Scholes in 1973 and this model is widely used by traders, investors, and financial institutions to price options and manage risk.

Table of Contents

What is a Neural Network

Neural Networks is a computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form. The concept of the *Artificial Neural Network (ANN)* was inspired by human biology and the way neurons of the human brain function together to understand inputs from human senses. In simple words, Neural Networks are a set of algorithms that tries to recognize the patterns, relationships, and information from the data through a process that is inspired by and works like the human brain/biology.

Neural Network

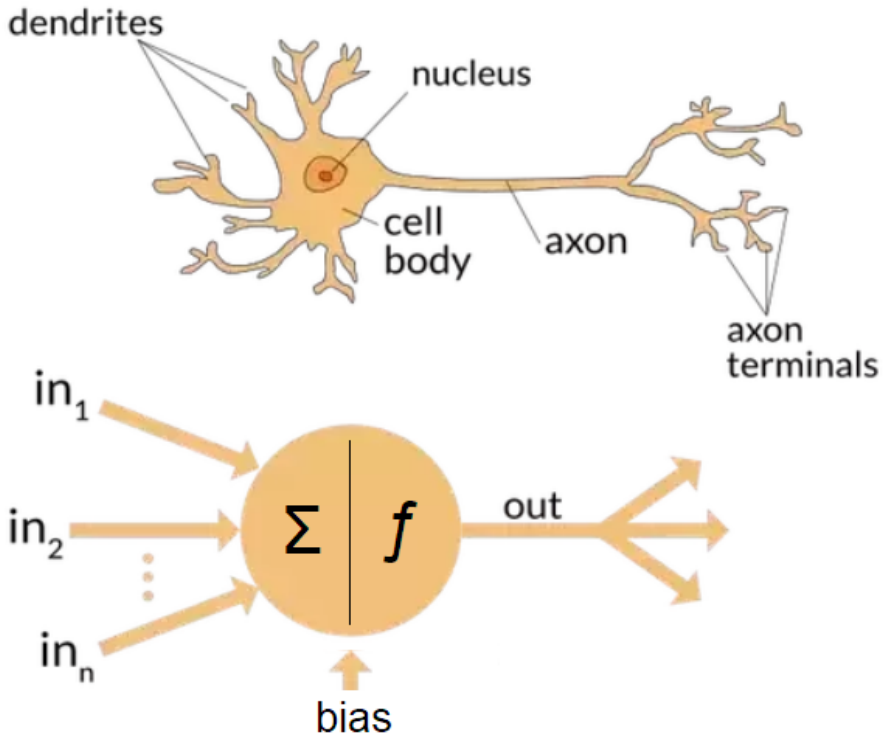


Figure: Human brain vs Neural Network

Components of a Neural Network

A simple Neural Network consists of three components :

1. Input Layer
2. Hidden Layer
3. Output Layer

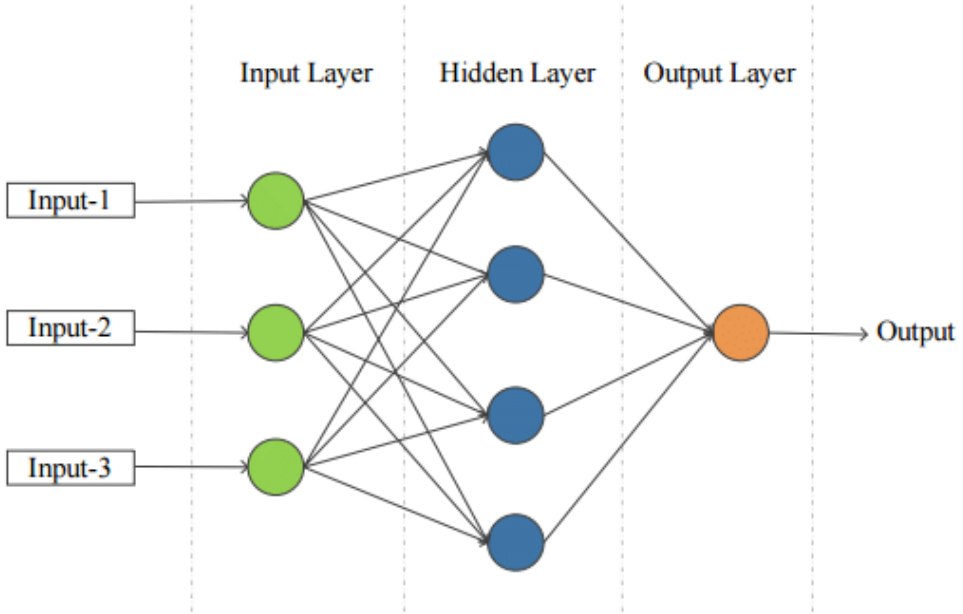


Figure: A Basic Neural Network

Components of a Neural Network

- **Input Layer:** Also known as Input nodes are the inputs/information from the outside world which is provided to the model to learn and derive conclusions from.
- **Hidden Layer:** Hidden layer is the set of neurons where all the computations are performed on the input data.
- **Output layer:** The output layer is the output/conclusions of the model derived from all the computations performed. There can be single or multiple nodes in the output layer.

Components of a Neural Network

- **Inputs** to a neuron can either be features from a training set or outputs from a previous layer's neurons.

Components of a Neural Network

- **Inputs** to a neuron can either be features from a training set or outputs from a previous layer's neurons.Weights are applied to the inputs as they travel along synapses to reach the neuron.

Components of a Neural Network

- **Inputs** to a neuron can either be features from a training set or outputs from a previous layer's neurons.Weights are applied to the inputs as they travel along synapses to reach the neuron.
- **Weights** are values that control the strength of the connection between two neurons.

Components of a Neural Network

- **Inputs** to a neuron can either be features from a training set or outputs from a previous layer's neurons.Weights are applied to the inputs as they travel along synapses to reach the neuron.
- **Weights** are values that control the strength of the connection between two neurons.That is, inputs are typically multiplied by weights, and that defines how much influence the input will have on the output.

Components of a Neural Network

- **Inputs** to a neuron can either be features from a training set or outputs from a previous layer's neurons.Weights are applied to the inputs as they travel along synapses to reach the neuron.
- **Weights** are values that control the strength of the connection between two neurons.That is, inputs are typically multiplied by weights, and that defines how much influence the input will have on the output.
- **Bias** terms are additional constants attached to neurons and added to the weighted input before the activation function is applied.

Components of a Neural Network

- **Inputs** to a neuron can either be features from a training set or outputs from a previous layer's neurons.Weights are applied to the inputs as they travel along synapses to reach the neuron.
- **Weights** are values that control the strength of the connection between two neurons.That is, inputs are typically multiplied by weights, and that defines how much influence the input will have on the output.
- **Bias** terms are additional constants attached to neurons and added to the weighted input before the activation function is applied.Bias terms help models represent patterns that do not necessarily pass through the origin.

Components of a Neural Network

- **Activation functions** modify the data they receive before passing it to the next layer. It give neural networks their power allowing them to model complex non-linear relationships. By modifying inputs with non-linear functions neural networks can model highly complex relationships between features. Popular activation functions include ReLU and Sigmoid.

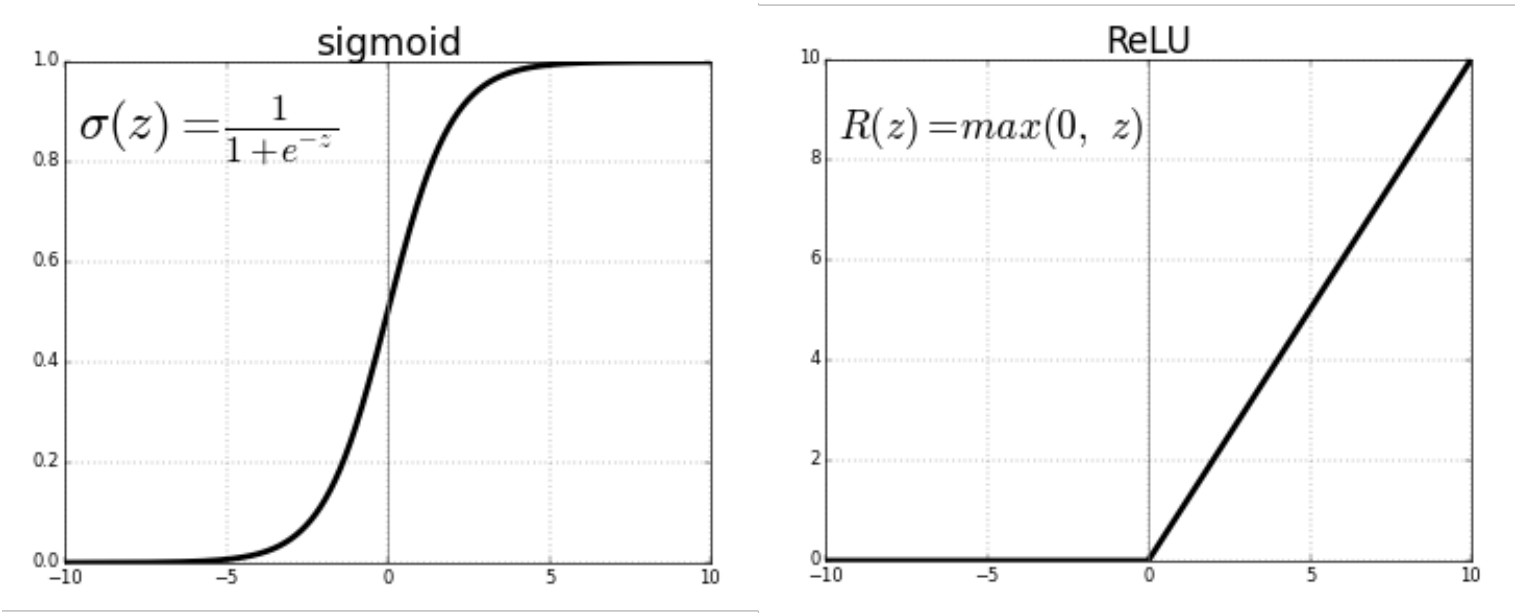


Figure: Sigmoid (left) and ReLU (right)

LSTM

- **LSTM (Long Short-Term Memory)** is a type of recurrent neural network (RNN) that is designed to handle the vanishing gradient problem in traditional RNNs. The vanishing gradient problem occurs when the gradients that are used to update the weights of the neural network during training become too small, making it difficult for the network to learn long-term dependencies.

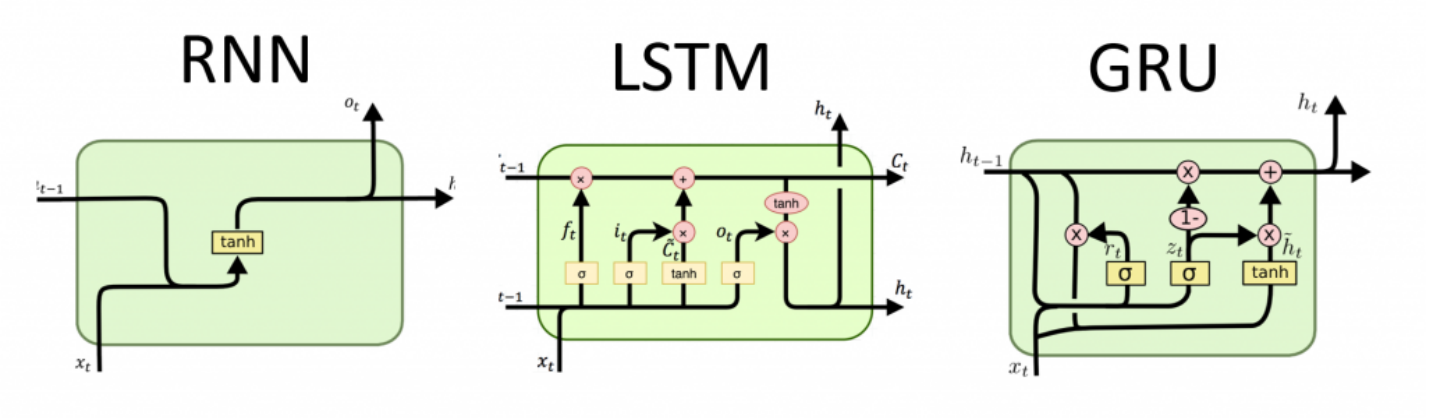


Figure: RNN vs LSTM vs GRU

Table of Contents

Deep Galerkin Method

The Deep Galerkin Method (DGM) is a machine learning approach for solving differential equations.

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:

Deep Galerkin Method

The Deep Galerkin Method (DGM) is a machine learning approach for solving differential equations.

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
-

$$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$

Deep Galerkin Method

The Deep Galerkin Method (DGM) is a machine learning approach for solving differential equations.

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
- $$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$
- $$u(0, x) = u_0(x) \quad x \in \Omega \quad \text{(Intial condition)}$$

Deep Galerkin Method

The Deep Galerkin Method (DGM) is a machine learning approach for solving differential equations.

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
- $$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$
- $$u(0, x) = u_0(x) \quad x \in \Omega \quad \text{(Intial condition)}$$
- $$u(t, x) = g(t, x) \quad (t, x) \in [0, T] \times \partial\Omega \quad \text{(Boundary condition)}$$

Deep Galerkin Method

The Deep Galerkin Method (DGM) is a machine learning approach for solving differential equations.

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
- $$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$
- $$u(0, x) = u_0(x) \quad x \in \Omega \quad \text{(Intial condition)}$$
- $$u(t, x) = g(t, x) \quad (t, x) \in [0, T] \times \partial\Omega \quad \text{(Boundary condition)}$$
- So now the main goal is approximate u with an approximating function $f(t, x, \theta)$, given by deep neural network with parameter set θ. The loss function for the associate training problem consists of three parts:

Deep Galerkin Method

- Measure that how well the approximation satisfies the differential operator,

$$\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t, x; \theta) \|^2_{[0,T] \times \Omega, v_1}$$

Deep Galerkin Method

- Measure that how well the approximation satisfies the differential operator,

$$\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t, x; \theta) \|^2_{[0,T] \times \Omega, v_1}$$

- measure of how well the approximation satisfies the boundary condition,

$$\| f(t, x; \theta) - g(t, x) \|^2_{[0,T] \times \partial\Omega, v_2}$$

Deep Galerkin Method

- Measure that how well the approximation satisfies the differential operator,

$$\left\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t, x; \theta) \right\|^2_{[0,T] \times \Omega, v_1}$$

- measure of how well the approximation satisfies the boundary condition,

$$\| f(t, x; \theta) - g(t, x) \|^2_{[0,T] \times \partial\Omega, v_2}$$

- measure of how well the approximation satisfies the initial condition,

$$\| f(t, x; \theta) - u_0(x) \|^2_{\Omega, v_3}$$

Deep Galerkin Method

- In all three terms above the error is measured in terms of L^2 -norm, And combining these three terms gives the total loss and we minimize this loss function.

Deep Galerkin Method

- In all three terms above the error is measured in terms of L^2 -norm, And combining these three terms gives the total loss and we minimize this loss function.
-

$$L(\theta) = \left\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t,x;\theta) \right\|^2_{[0,T]\times\Omega,v_1} + \| f(t,x;\theta) - g(t,x) \|^2_{[0,T]\times\partial\Omega,v_2}$$

$$+ \| f(t,x;\theta) - u_0(x) \|^2_{\Omega,v_3}$$

Table of Contents

Definition(Option)

Definition(Option)

An option is a financial instrument that allows its owner to buy or sell(but not the obligation) an underlying asset at a pre-specified fixed price within a specific period.

- **Call Option**: An option that gives its holder the right time buy the underlying asset at an agreed price by the maturity data.

Definition(Option)

Definition(Option)

An option is a financial instrument that allows its owner to buy or sell(but not the obligation) an underlying asset at a pre-specified fixed price within a specific period.

- **Call Option**: An option that gives its holder the right time buy the underlying asset at an agreed price by the maturity data.
- **Put Option**: An option that gives its holder the right to sell the underlying asset at an agreed price by the maturity data.

Definition

- **Europen Option**: Europen Option which gives the holder the right to buy or sell an underlying asset at a predetermined price and time in the future

Definition

- **Europen Option**: Europen Option which gives the holder the right to buy or sell an underlying asset at a predetermined price and time in the future
- **American Option**: American-style option allows the holder to exercise the option at any time prior to the expiration date.

Definition(Pay-off)

The pay-off of an option is its value at the time of exercise T. It is basically gain or loss that can be described in the following figure.

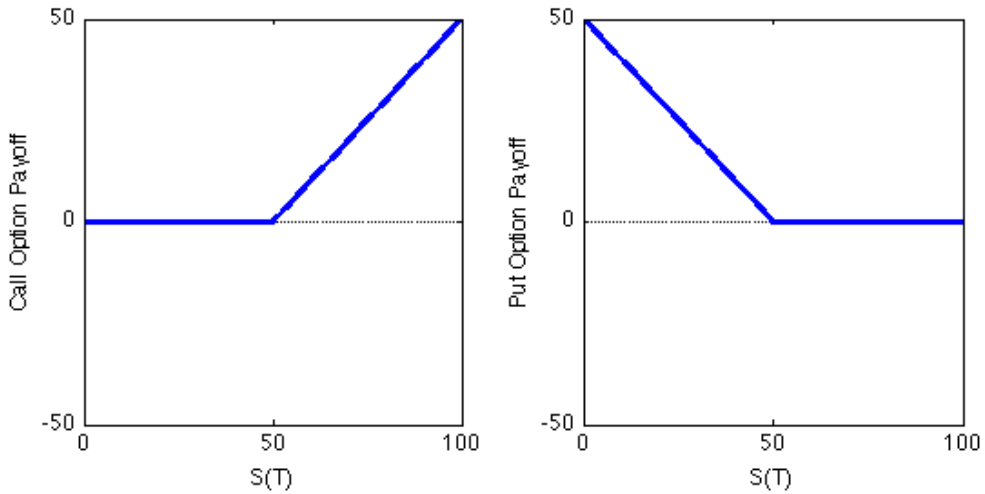


Figure: Payoff diagram of European Call and European Put.

$$V(S_T, T) = \max\{S_T - K, 0\}.$$

$$V(S_T, T) = \max(K - S_T, 0).$$

Table of Contents

Black-Scholes Formula

Now we have boundary conditions and final condition. So when the interest rate r and volatility σ are constant.The exact solution for European call is

$$C(S,t) = SN\left(d_1\right) - Ke^{-r(T-t)}N\left(d_2\right), \tag{1}$$

where $N(\cdot)$ is the cumulative distribution function for a standardised normal random variable, given by

$$N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}y^2}dy. \tag{2}$$

Here

$$\begin{cases} d_1 = \dfrac{\log(\frac{S}{K}) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, \\ d_2 = \dfrac{\log(\frac{S}{K}) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}. \end{cases} \tag{3}$$

For a European put, the exact solution is

$$P(S,t) = Ke^{-r(T-t)}N\left(-d_2\right) - SN\left(-d_1\right). \tag{4}$$

Considering the values of parameters as $T = 1, K = 10, r = 0.2$, $\sigma = 0.25$ and $S_{max} = 20$. We get the following plots for $V(S, t)$

Plot of Exact Solution of Black-Scholes Equation

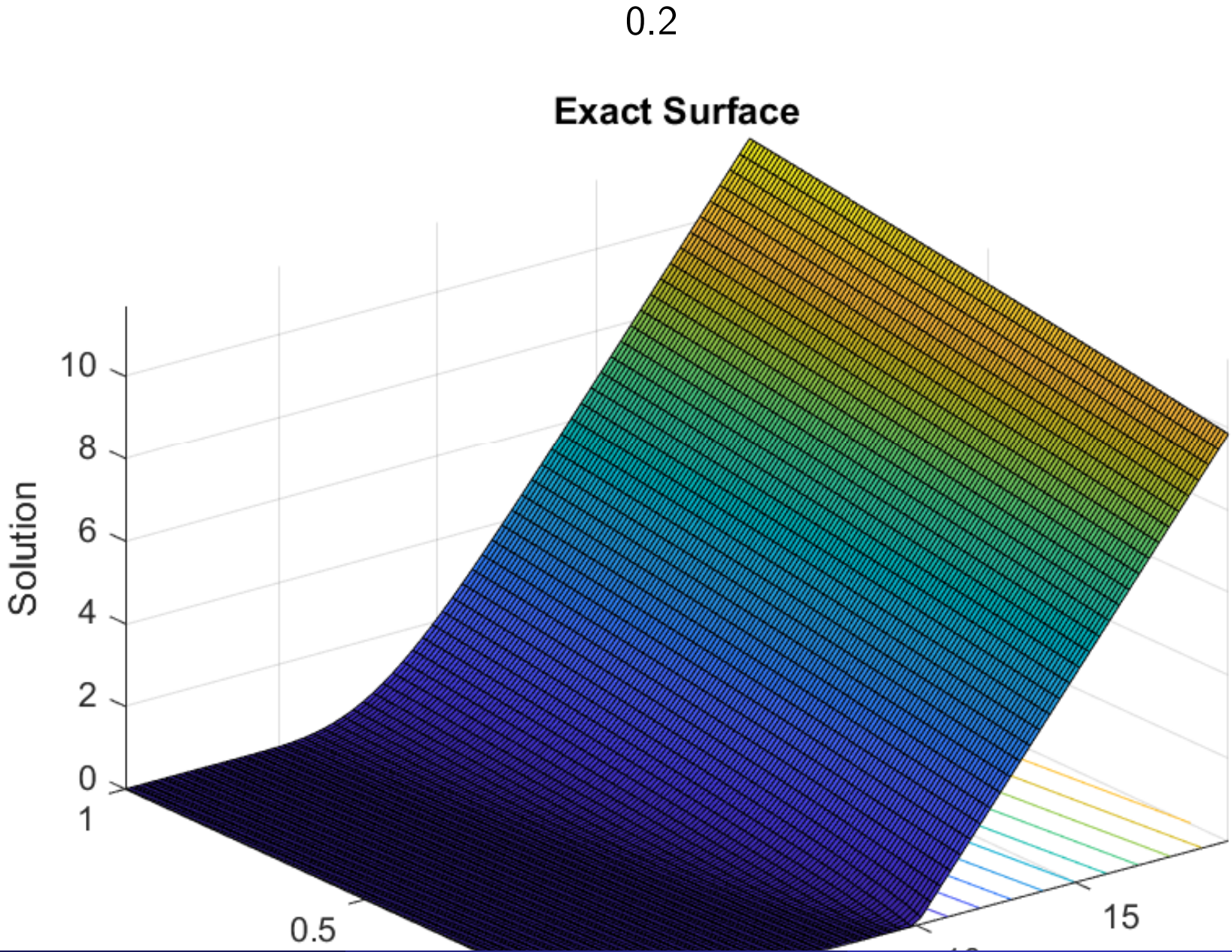


Table of Contents

Work done

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:

Work done

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
-

$$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$

Work done

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
- $$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$
- $$u(0, x) = u_0(x) \quad x \in \Omega \quad \text{(Intial condition)}$$

Work done

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
- $$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$
- $$u(0, x) = u_0(x) \quad x \in \Omega \quad \text{(Intial condition)}$$
- $$u(t, x) = g(t, x) \quad (t, x) \in [0, T] \times \partial\Omega \quad \text{(Boundary condition)}$$

Work done

- Suppose the form of the PDEs of interest are generally described as follows: let u be an unknown function of the time and space defined on the region $[0, T] \times \Omega$ where $\Omega \subset \mathbf{R}$ and u satisfies the PDEs:
- $$\left(\frac{\partial}{\partial t} + \mathcal{L}\right) u(t, x) = 0 \quad (t, x) \in [0, T] \times \Omega$$
- $$u(0, x) = u_0(x) \quad x \in \Omega \quad \text{(Intial condition)}$$
- $$u(t, x) = g(t, x) \quad (t, x) \in [0, T] \times \partial\Omega \quad \text{(Boundary condition)}$$
- So now the main goal is approximate u with an approximating function $f(t, x, \theta)$, given by deep neural network with parameter set θ. The loss function for the associate training problem consists of three parts:

Work done

- Measure that how well the approximation satisfies the differential operator,

$$\left\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t, x; \theta) \right\|^2_{[0,T] \times \Omega, v_1}$$

Work done

- Measure that how well the approximation satisfies the differential operator,

$$\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t, x; \theta) \|^2_{[0,T] \times \Omega, v_1}$$

- measure of how well the approximation satisfies the boundary condition,

$$\| f(t, x; \theta) - g(t, x) \|^2_{[0,T] \times \partial\Omega, v_2}$$

Work done

- Measure that how well the approximation satisfies the differential operator,

$$\left\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t, x; \theta) \right\|^2_{[0,T] \times \Omega, v_1}$$

- measure of how well the approximation satisfies the boundary condition,

$$\| f(t, x; \theta) - g(t, x) \|^2_{[0,T] \times \partial\Omega, v_2}$$

- measure of how well the approximation satisfies the initial condition,

$$\| f(t, x; \theta) - u_0(x) \|^2_{\Omega, v_3}$$

Work done

- In all three terms above the error is measured in terms of L^2 -norm, And combining these three terms gives the total loss and we minimize this loss function.

Work done

- In all three terms above the error is measured in terms of L^2 -norm, And combining these three terms gives the total loss and we minimize this loss function.
-

$$L(\theta) = \left\| \left(\frac{\partial}{\partial t} + \mathcal{L} \right) f(t, x; \theta) \right\|^2_{[0,T]\times\Omega, v_1} + \| f(t, x; \theta) - g(t, x) \|^2_{[0,T]\times\partial\Omega, v_2}$$

$$+ \| f(t, x; \theta) - u_0(x) \|^2_{\Omega, v_3}$$

Work done

So now we minimize the loss function using Stochastic Gradient Descent(SGD) and more specifically we apply the algorithm that is defined below.

1. Initialize the parameter set θ_0 and the learning rate α_n

Work done

So now we minimize the loss function using Stochastic Gradient Descent(SGD) and more specifically we apply the algorithm that is defined below.

1. Initialize the parameter set θ_0 and the learning rate α_n
2. Generate random samples from the domain interior and time, spatial boundaries
   - Generate (t_n, x_n) from $[0, T] \times \Omega$ according to v_1
   - Generate (τ_n, z_n) from $[0, T] \times \partial\Omega$ according to v_2

Work done

So now we minimize the loss function using Stochastic Gradient Descent(SGD) and more specifically we apply the algorithm that is defined below.

1. Initialize the parameter set θ_0 and the learning rate α_n
2. Generate random samples from the domain interior and time, spatial boundaries
   - Generate (t_n, x_n) from $[0, T] \times \Omega$ according to v_1
   - Generate (τ_n, z_n) from $[0, T] \times \partial\Omega$ according to v_2
   - Generate w_n from Ω, according to v_3
3. Calculate the loss function for the current mini-batch (the randomly sampled points $s_n = ((t_n, x_n), (\tau_n, z_n), w_n)$
   - Compute $L_1(\theta_n; , t_n, x_n) = \left(\left(\frac{\partial}{\partial t} + \mathcal{L}\right) f(\theta_n; t_n, x_n)\right)^2$
   - Compute $L_2(\theta_n; , \tau_n, z_n) = \left(f(\tau_n, z_n) - g(\tau_n, z_n)\right)^2$
   - Compute $L_3(\theta_n; w_n) = \left(f(0, w_n) - u_0(w_n)\right)^2$

Work done

So now we minimize the loss function using Stochastic Gradient Descent(SGD) and more specifically we apply the algorithm that is defined below.

1. Initialize the parameter set θ_0 and the learning rate α_n
2. Generate random samples from the domain interior and time, spatial boundaries
   - Generate (t_n, x_n) from $[0, T] \times \Omega$ according to v_1
   - Generate (τ_n, z_n) from $[0, T] \times \partial\Omega$ according to v_2
   - Generate w_n from Ω, according to v_3
3. Calculate the loss function for the current mini-batch (the randomly sampled points $s_n = ((t_n, x_n), (\tau_n, z_n), w_n)$
   - Compute $L_1(\theta_n; , t_n, x_n) = \left(\left(\frac{\partial}{\partial t} + \mathcal{L}\right) f(\theta_n; t_n, x_n)\right)^2$
   - Compute $L_2(\theta_n; , \tau_n, z_n) = \left(f(\tau_n, z_n) - g(\tau_n, z_n)\right)^2$
   - Compute $L_3(\theta_n; w_n) = \left(f(0, w_n) - u_0(w_n)\right)^2$
4. Take a descent step at the random point s_n with Adam-based learning rates

$$\theta_{n+1} = \theta_n - \alpha_n . \nabla_\theta L(\theta_n; s_n)$$

Work done

So now we minimize the loss function using Stochastic Gradient Descent(SGD) and more specifically we apply the algorithm that is defined below.

1. Initialize the parameter set θ_0 and the learning rate α_n
2. Generate random samples from the domain interior and time, spatial boundaries
   - Generate (t_n, x_n) from $[0, T] \times \Omega$ according to v_1
   - Generate (τ_n, z_n) from $[0, T] \times \partial\Omega$ according to v_2
   - Generate w_n from Ω, according to v_3
3. Calculate the loss function for the current mini-batch (the randomly sampled points $s_n = ((t_n, x_n), (\tau_n, z_n), w_n)$
   - Compute $L_1(\theta_n; , t_n, x_n) = \left(\left(\frac{\partial}{\partial t} + \mathcal{L}\right) f(\theta_n; t_n, x_n)\right)^2$
   - Compute $L_2(\theta_n; , \tau_n, z_n) = \left(f(\tau_n, z_n) - g(\tau_n, z_n)\right)^2$
   - Compute $L_3(\theta_n; w_n) = \left(f(0, w_n) - u_0(w_n)\right)^2$
4. Take a descent step at the random point s_n with Adam-based learning rates

$$\theta_{n+1} = \theta_n - \alpha_n . \nabla_\theta L(\theta_n; s_n)$$

5. Repeat the steps (2) and (4) until $\|(\theta_{n+1} - \theta_n)\|$ is small.

Implementation

- We used Long-Short Term Memory(LSTM) model for the implementation of DGM Layer. In this layer has an input layer, hidden layer, and output layer.

Implementation

- We used Long-Short Term Memory(LSTM) model for the implementation of DGM Layer. In this layer has an input layer, hidden layer, and output layer.

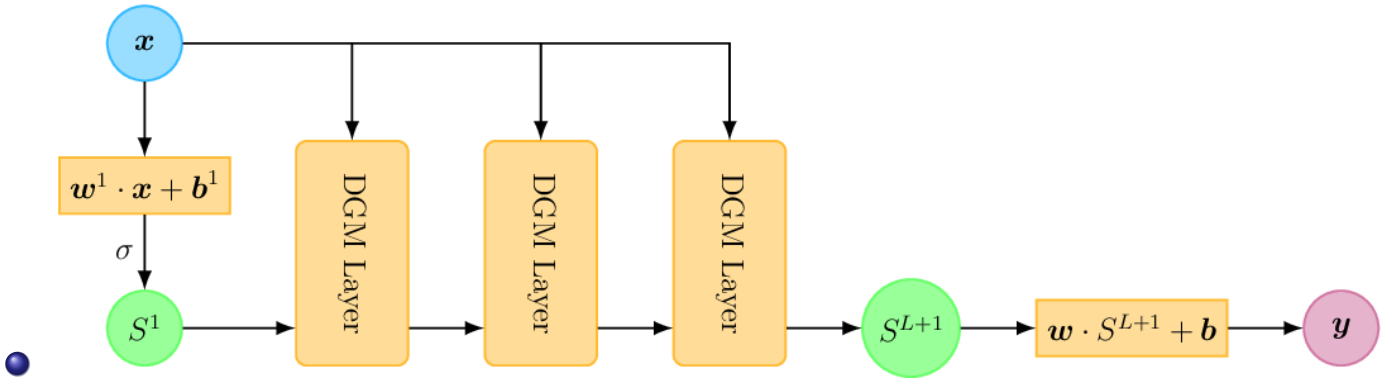


Figure: DGM Architecture

Implementation

- We used Long-Short Term Memory(LSTM) model for the implementation of DGM Layer. In this layer has an input layer, hidden layer, and output layer.

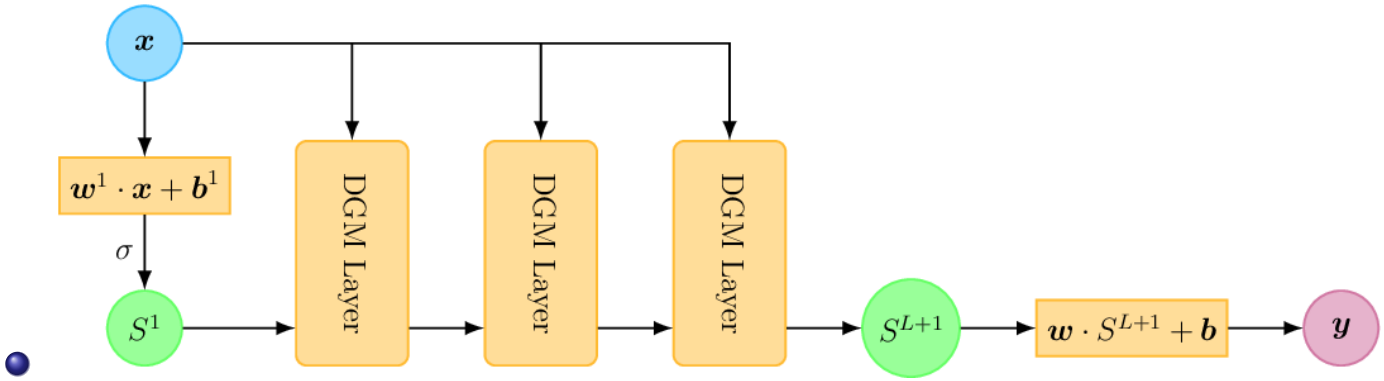


Figure: DGM Architecture

- So now we take the input k mini-batch point on the domain (i.e x in our case and also we chose random normal distribution to take the input on the domain) and also an output of the previous DGM layer. So in this process, we get a vector value output y which consists of the Neural Network and approximates the function of u.

Implementation

Now we give the equation that we used in Implementation and also give the visual representation.

- $$S_1 = \sigma(w_1.x + b_1)$$

Implementation

Now we give the equation that we used in Implementation and also give the visual representation.

- $$S_1 = \sigma(w_1.x + b_1)$$
- $$Z_l = \sigma(u_{z,l}.x + w_{z,l}.S_l + b_{z,l})$$

Implementation

Now we give the equation that we used in Implementation and also give the visual representation.

- $$S_1 = \sigma(w_1.x + b_1)$$
- $$Z_l = \sigma(u_{z,l}.x + w_{z,l}.S_l + b_{z,l})$$
- $$G_l = \sigma(u_{g,l}.x + w_{g,l}.S_l + b_{g,l})$$

Implementation

Now we give the equation that we used in Implementation and also give the visual representation.

- $$S_1 = \sigma(w_1.x + b_1)$$
- $$Z_l = \sigma(u_{z,l}.x + w_{z,l}.S_l + b_{z,l})$$
- $$G_l = \sigma(u_{g,l}.x + w_{g,l}.S_l + b_{g,l})$$
- $$R_l = \sigma(u_{r,l}.x + w_{r,l}.S_l + b_{r,l})$$

Implementation

Now we give the equation that we used in Implementation and also give the visual representation.

- $$S_1 = \sigma(w_1.x + b_1)$$
- $$Z_l = \sigma(u_{z,l}.x + w_{z,l}.S_l + b_{z,l})$$
- $$G_l = \sigma(u_{g,l}.x + w_{g,l}.S_l + b_{g,l})$$
- $$R_l = \sigma(u_{r,l}.x + w_{r,l}.S_l + b_{r,l})$$
- $$H_l = \sigma(u_{h,l}.x + w_{h,l}.(S_l \odot R_l) + b_{h,l})$$

Implementation

- $$S_{l+1} = (1 - G_l) \odot H_l + Z_l \odot S_l$$

Implementation

- $$S_{l+1} = (1 - G_l) \odot H_l + Z_l \odot S_l$$
- $$f(t, x; \theta) = w.S_{L+1} + b$$

Implementation

- $$S_{l+1} = (1 - G_l) \odot H_l + Z_l \odot S_l$$
- $$f(t, x; \theta) = w.S_{L+1} + b$$
- where $\odot$ denote element-wise multiplication and L denotes the total number of layer, σ denotes the activation function and x denotes the initial input, w denotes the weight and b is bias terms.

Implementation

- $$S_{l+1} = (1 - G_l) \odot H_l + Z_l \odot S_l$$
- $$f(t, x; \theta) = w.S_{L+1} + b$$
- where $\odot$ denote element-wise multiplication and L denotes the total number of layer, σ denotes the activation function and x denotes the initial input, w denotes the weight and b is bias terms.

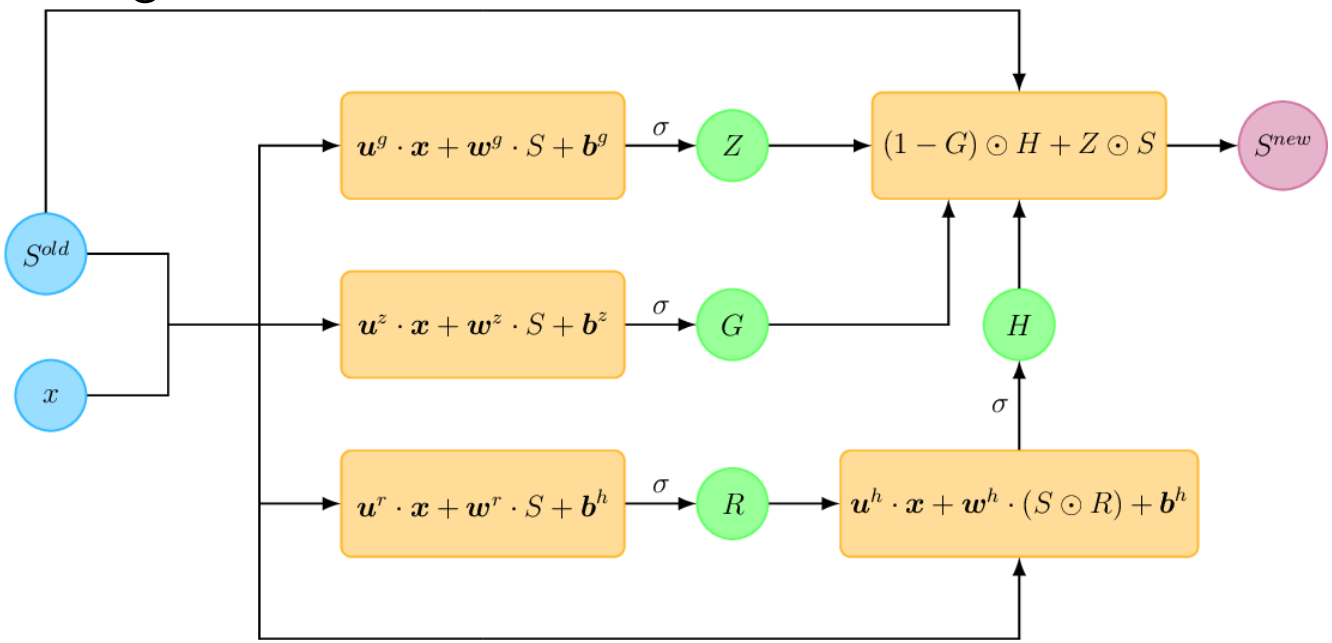


Figure: Notation each single DGM Layer

Experimentation and Results

European Call Option
One-Dimension Black-Scholes PDE

- $$\frac{\partial s(t,x)}{\partial t} + rs.\frac{\partial s(t,x)}{\partial x} + 1/2\sigma^2 x^2.\frac{\partial^2 s(t,x)}{dx^2} = r.s(t,x)$$

Experimentation and Results

European Call Option
One-Dimension Black-Scholes PDE

- $$\frac{\partial s(t,x)}{\partial t} + rs.\frac{\partial s(t,x)}{\partial x} + 1/2\sigma^2 x^2.\frac{\partial^2 s(t,x)}{dx^2} = r.s(t,x)$$
- $$s(T,x) = F(x)$$

Experimentation and Results

European Call Option
One-Dimension Black-Scholes PDE

- $$\frac{\partial s(t,x)}{\partial t} + rs.\frac{\partial s(t,x)}{\partial x} + 1/2\sigma^2 x^2.\frac{\partial^2 s(t,x)}{dx^2} = r.s(t,x)$$
- $$s(T,x) = F(x)$$
- **Solution**
- $$s(t,x) = x.N(v_+) - K.e^{-r(T-t)}N(v_-$$
- where

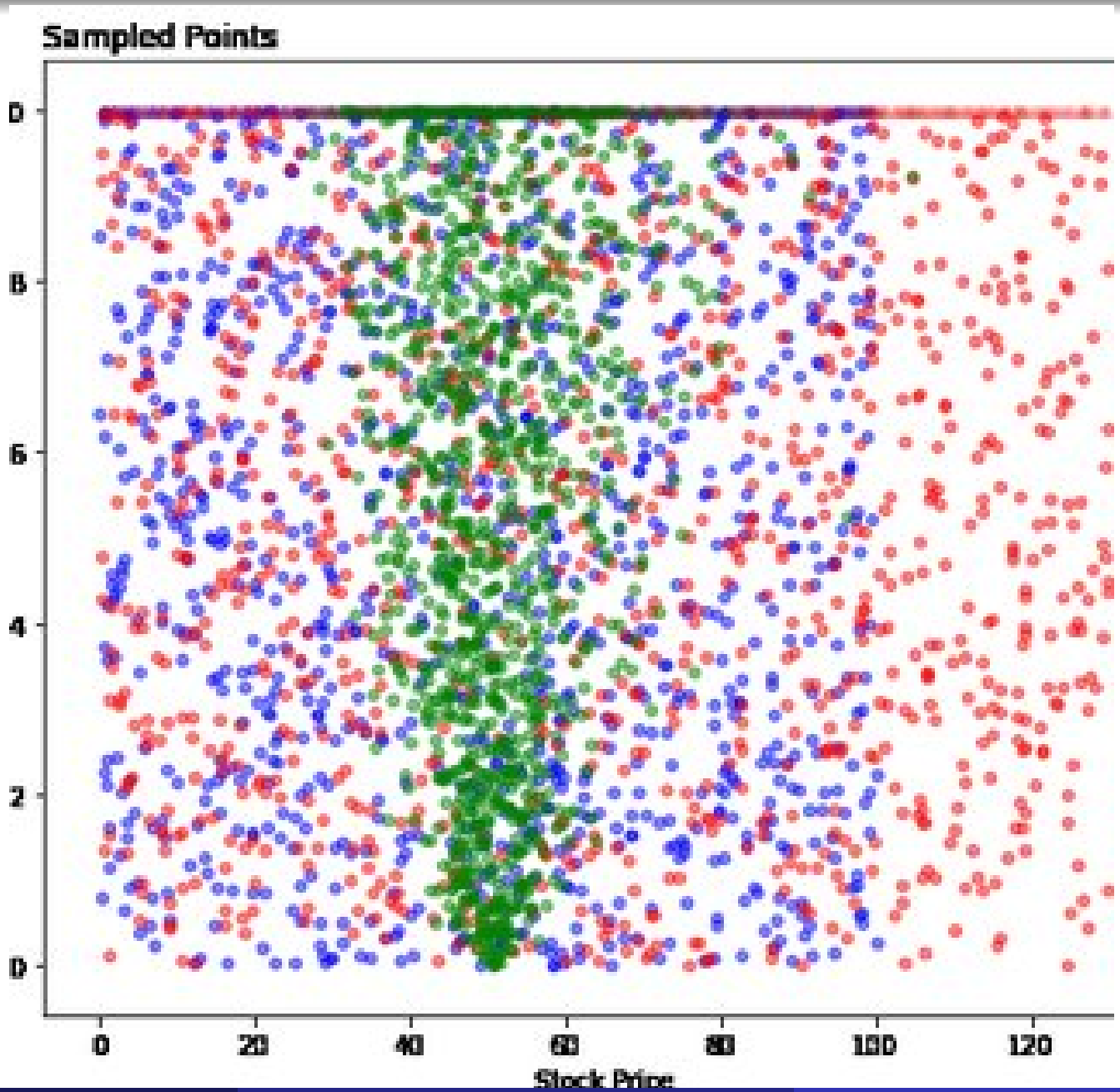
$$v_+ = \frac{\ln(x/K) + (r + 1/2\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

$$v_- = v_+ - \sigma\sqrt{T-t}$$

Experimentation and Results

Suppose you want to price a European call option on a non-dividend paying stock with a strike price of $K = 50$, a current stock price of $S = 50$, a time to maturity of $T = 1$, a continuously compounded risk-free interest rate of $r = 5\%$, and a volatility of $\sigma = 20\%$.

Experimentation and Results

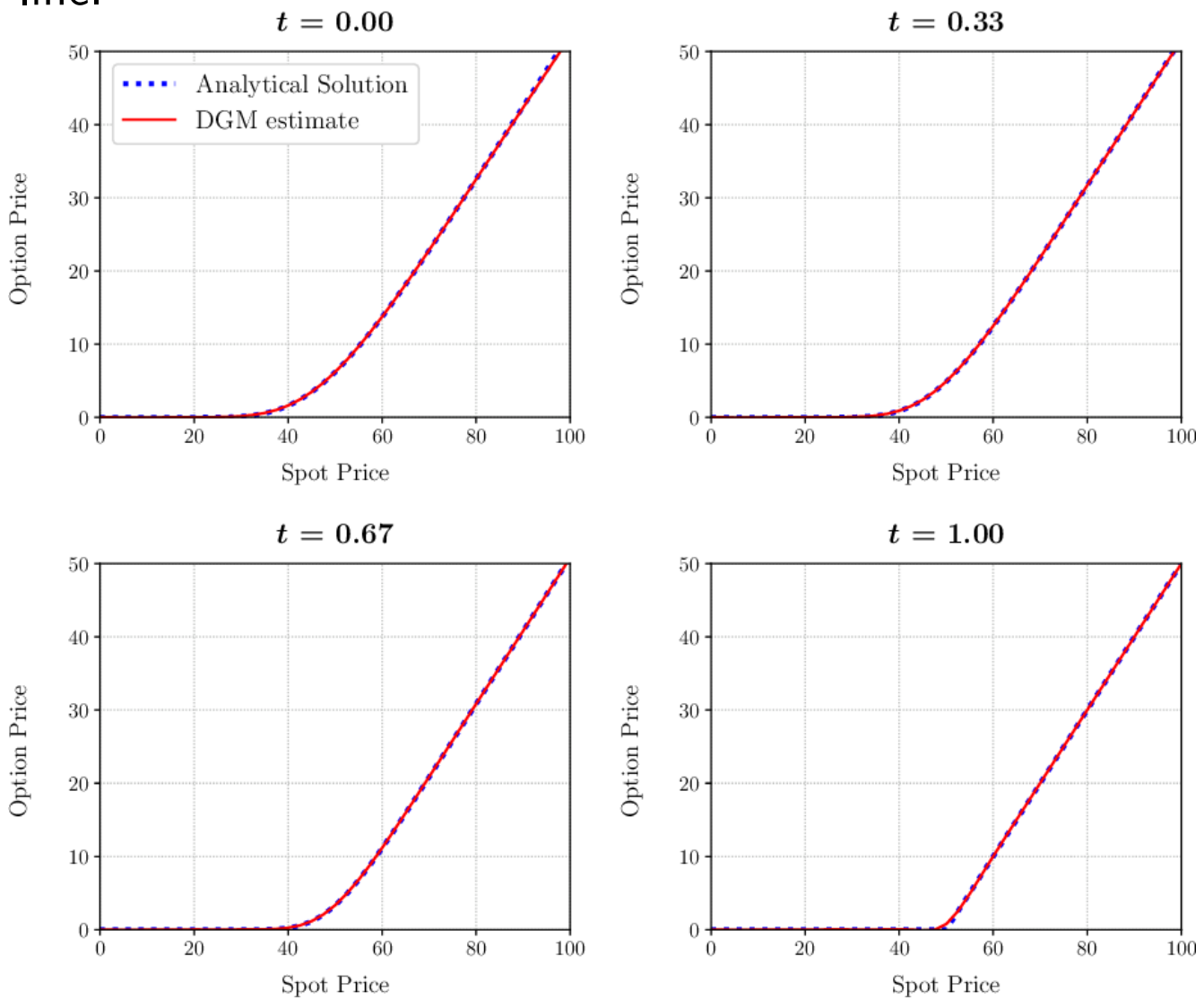


Experimentation and Results

- So Finally the DGM network points that create a best-fit line that we see in the below line.

Experimentation and Results

- So Finally the DGM network points that create a best-fit line that we see in the below line.



Conclusions

- We have shown that the Deep Galerkin Method(DGM) is a promising approach for solving the European-option pricing model.Our results demonstrate that the DGM can accurately and efficiently approximate the solution to the Black-Scholes equation, but in traditional numerical methods struggle to converge.

Bibliography

V. Müller, and Konstantinos Spiliopoulos
A deep learning algorithm for solving partial differential equations. Journal of computational
Studia physics, 375:1339–1364, 2018.

Jiequn Han, Arnulf Jentzen, and Weinan E.
Solving high-dimensional partial differential equa- tions using deep learning. Proceedings of the National Academy of Sciences
115(34):8505–8510,2018.