Contents

Abstract

We propose to solve high-dimensional PDEs using a deep learning algorithm which uses multi-layer neural networks. Rather than feeding large amounts data for training, the only select the boundary data based on periodic to be our training data. We use the Poisson equation to understand how the NN approximate the hidden pattern. By giving the specific boundary data, NN can predict the whole picture correctly. This paper provides the loss function which is a method for training using Sobolev space. The goal is to find the solutions the solution approximated by a neural network instead of a linear combination of basis functions.

1 Introduction

In our paper, we consider a class of linear partial differential equations(PDEs) where u is defined over $\Omega \subset \mathbb{R}^M \times \mathbb{R}^N$

$$\begin{aligned} \mathcal{L}(u) &= f,\ x \in \Omega \\ \partial u(t,x) &= g(t,x),,\ x \in \partial\Omega \end{aligned} \tag{1}$$

We focus on the problems where $\mathcal{L}$ is the differential operator. The goal is to find the functions $q \in \Omega$ by deep NN that satisfies the given linear equation $\mathcal{L}(u) = f$.

We develop deep neural network approximations for stationary PDEs. The domain for the input dataset is $x \subset \Omega \subset \mathbb{R}^M \times \mathbb{R}^N$. The solution is in the Banach space with the norm

$$||u||_{q,\Omega} = \left(\int_\Omega |u(x)|^p dx \right)^{\frac{1}{p}} \tag{2}$$

and $u_0 \in L^p(\Omega)$. In our paper, we aim to implement multi-layer neural networks to approximate the solutions of PDEs in $L^p(\mu)$. $L^p(\mu)$ is the Banach space consisting of all measurable functions on μ that are p-power summable on Ω

$$||q(\sigma,x)||_{p,\mu} = \left[\int_\mu |q(\psi,x,t)|^p d\mu(x) \right]^{\frac{1}{p}} < \infty \tag{3}$$

so that

$$J_{p,\mu}(f) \in L^p(\mu). \tag{4}$$

Let the input environment measure μ be a Borel measure on Ω and $C^d(\mathbb{R}^k)$ denotes the space of all continuous and differentiable functions f with partial derivative $D^\alpha f$ of order $|\alpha| < d$ being continuous on $\mathbb{R}^k$. For $q \in C^d(\mathbb{R}^k)$ and $1 < p < \infty$, $C^{d,p}(\mu)$ is the usual Sobolev space of order d with the norm.

$$||q(\sigma,x)||_{p,\mu} = \left[\sum_{\alpha \le d} \int_{\mathbb{R}^k} |D^\alpha q(\sigma,x)|^p d\mu(x) \right]^{\frac{1}{p}} \tag{5}$$

The space of continuous functions on I_n is denoted by $C(I_n)$.

2 Literature Review

Using deep neural network to find the solutions for PDEs has rising in the computational mathematics field in these years. To the best of our knowledge, most of the works trained the deep neural network with supervised learning to approximate the solutions of PDEs [[1] , [2]].

Some research works use special tool to correct the initialization of weights and biases. They found that the initialization of weights and biases determine the level of accuracy of the solver. Weinan E et al. [3] initialized the weights via transfering the weight from similar problem to speed up the convergence rate and learning time. Yuehaw Khoo et al. [4] solve the committor function which is the central object of transition path theory by NN. They suggest that the usage of sampling scheme can prevent the output goes too disperse.

In fact, the prototypes of these algorithms were based on Kolmogorov's dimension-reducing superpositions. The pristine architecture consists of nonlinear and linear layers, the first constitute the hidden layer and the second constitutes the output layer. The mathematical analysis of neural network and deep neural network remain the computational black box. Cybenko [5] provided the mathematical proof that the finite linear combination form like the q function in (**??**) can approximate any continuous function with support in the unit hypercube. Moreover, it proves that the space for the output of neural network is dense in $C(I_n)$ which share the same properties with surface-filling curve.

Therefore, we examine underlying topological properties of the superposition function for neural network. The topology help us understand the underlying approach used to approximate any continuous function. The analysis of topology is derives from Sprecher's [[6], [7]] work that each of the function in 6 determines a surface-filling curve with the specific properties.

3 Neural networks

The Kolmogorov's superposition theorem presented in 1957 and solved the Hilbert's thirteenth problem [1]about analytic function of three variables that can be expressed as a superposition of bivariate ones. Kolmogorov superposition theorem found attention in neural network computation by Hecht-Nielsen[8]. The representation of continuous functions defined on an n-dimensional cube by sums and superpositions of continuous functions of one variable shows the potential applicability in neural networks[9]. Due to many scientists devoted to transform Kolmogorov's superposition theorem to numerical algorithm in many years, Kolmogorov's generalization becomes the current neural network approximate theorem.

Theorem 3.1. *(A. Kolmogorov, 1956; V. Arnold, 1957) Let $f : \mathbb{I}^n := [0,1]^n \to \mathbb{R}$ be an arbitrary multivariate continuous function. Then, it has the representation*

$$f(x_1, x_2, \ldots, x_n) = \sum_{j=0}^{2m} \Phi_q \left(\sum_{i=1}^{n} \phi_{i,j}(x_i) \right) \tag{6}$$

with continuous one-dimensional outer and inner function Φ_j and $\phi_{j,i} \in L^p(\mu)$, $0 \le j \le 2m, m \in \mathbb{R}$. Moreover, functions $\phi_{i,j}$ are universal for the given dimension n; they are independent of f.

3.1 Framework

3.2 Network Architecture

In our paper, we consider multilayer NN. The deep NN is consisted of $L+1$ layers where layer 0 is the input layer and layer L is the output layer. The layers $0 < \ell < L$ are the hidden layers. The activation in the hidden layer can be any bounded, nonconstant function such as hyperbolic tangents, sigmoids and rectified linear units[10]. The activation function for output layer will be the linear activation. Each neuron in the NN is supplied with a bias and the weight corresponding to the each neurons of subsequent layer, including the output neurons but excluding the input neuron. Figure 1 show the schematic representation of a Deep NN with $L = 4$ layers, $M = 4$ hidden units.

3.2.1 Notations for parameters in neural network

- We take input as 2 dimensional vector. Let $x_{i,j}$ denote $(x_i, y_j), i, j = \{1, 2, \ldots, n\}, n \in \mathbb{N}$.
- We set each layer has K neurons. We use k to indicate the single neuron in each layer.
- The bias in neuron k in layer ℓ is denoted by $c_k^{(\ell)}$.

[1]Hilbert considered the seventh-degree equation: $x^7 + ax^3 + bx^2 + cx + 1 = 0$ and asked whether its solution, can be the composition of a finite number of two-variable functions.

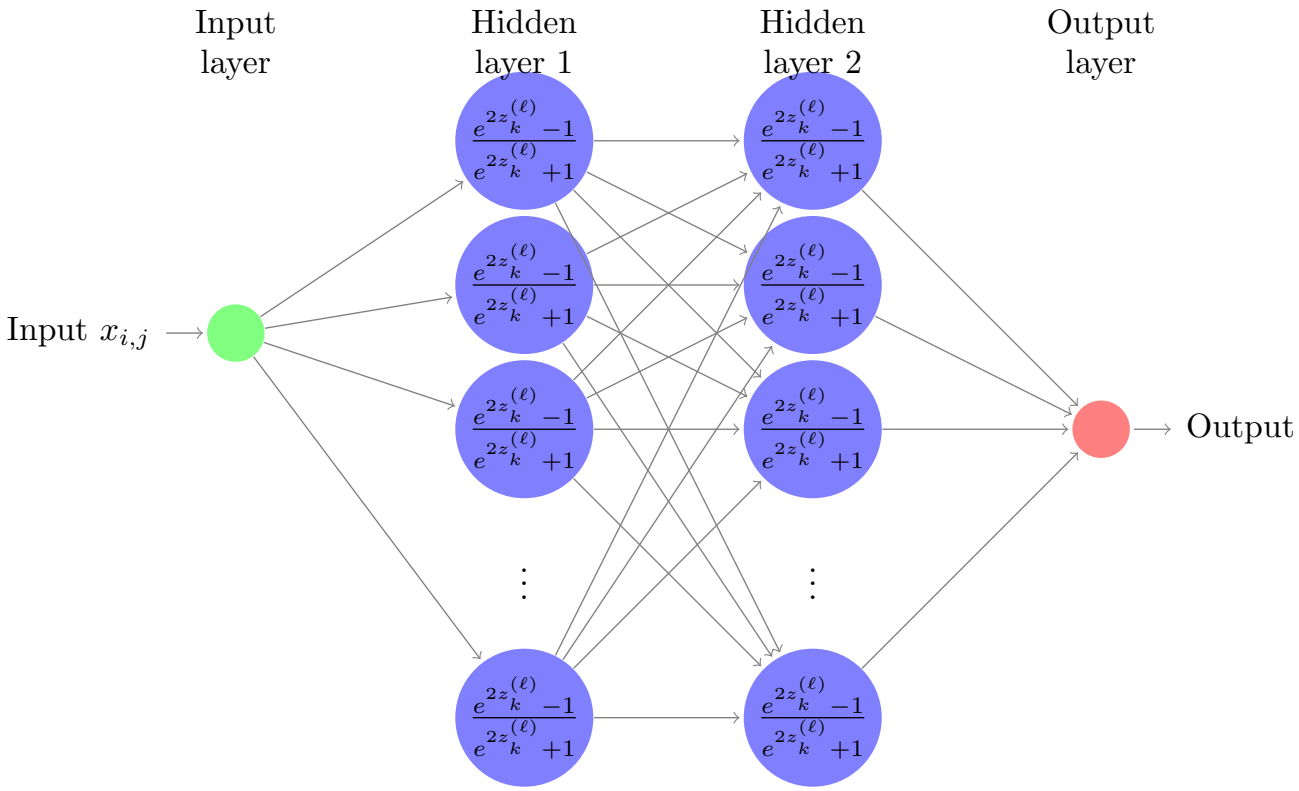


Figure 1: Schematic representation of multilayers neural network

- $w_{k,k'}^{(\ell)}$: The weight between k neuron in layer $\ell-1$ and $k_i, i=1,\ldots,K$ neuron in ℓ layer. For notational convenience, in the sums of the form of NN in (7), we will replace $k_i^{\ell-1}$ with k and let k' denote $k_i^{(\ell)}$.
- The activation function in layer ℓ is denoted by $\sigma^{(\ell)}$. We will take hyperbolic tangents in the hidden layer. The activation is hyperbolic tangent which takes the form

$$\tanh(z_k^{(\ell)}) = \frac{e^{2z_k^{(\ell)}}-1}{e^{2z_k^{(\ell)}}+1}$$

- The output from neuron k in layer ℓ is denoted by $q(x_{i,j},\sigma)_k^{(\ell)}$.

3.2.2 Feed forward procedure

The input of neurons for layer ℓ is given by

$$z(\sigma,x_{i,j})_m^{(\ell)} = \sum_{k^{\ell-1}=1}^{M} w_{k^{\ell-1},k_i^\ell}^{(\ell)} \sigma_{\ell-1}(z_k^{\ell-1}) + c_m^{(\ell)} \tag{7}$$

where the sum is taken over all inputs to neuron m in layer ℓ. The quantity will be used extensively in feedforward algorithm for computing $q(x_{i,j},\sigma)^L$. The feedforward algorithm of NN for each layer is implemented subsequently:

1. When $\ell=0$, $z_k^{(0)}=\mathbf{x}, x\subset\Omega\subset\mathbb{R}\times\mathbb{R}^N$. We consider $x\in[0,N]\times\mathbb{R}^N$ case to illustrate the algorithm, so we denote the elements in $\mathbf{x}$ as $x_{i,j} i=1\ldots N, j=1\ldots J$.
2. When $\ell=1$, the input of neuron k', where $k'=k_i^\ell, i=1,\ldots,K$ is:

$$z(\sigma,x_{i,j})_{k'}^{(1)} = \sum_{k=1}^{K} w_{k,k'}^{(1)} \sigma(w_{0,k}x_{i,j}+c_{k'}^{(1)}) + c_{k'}^{(\ell)} \tag{8}$$

3. When $1 < \ell < L$The input for neuron m in layer ℓ is:

$$\begin{aligned} z(\sigma, x_{i,j})_{k'}^{(\ell)} &= \sum_{k=1}^{K} w_{k,k'}^{(\ell)} \sigma(z_k^{(\ell-1)}) + c_m^{(\ell)} \\ &= \sum_{k=1}^{K} w_{k,k'}^{\ell} q_k^{(\ell-1)} + c_{k'}^{(\ell)} \end{aligned} \tag{9}$$

4. When $\ell = L$, assume there are K neurons in the output layer. The output for neuron $k = K$ is:

$$q(\sigma, x_{i,j})^{(L)} = \sum_{k=1}^{K} w_{k,k'}^{(L)} \sigma(z_k^{(L-1)}) + c_{k'}^{L} \tag{10}$$

For simplicity, the form of multilayer neural network can be given by a composition of activation and affine functions [11]

$$q = \sigma^{(L)} \circ z^{(L)} \circ \sigma^{(L-1)} \circ z^{(L-1)} \cdots \circ \sigma^{(L)} \circ z^{(1)} \tag{11}$$

The preactivation function ϕ is affine transformation $\psi : \Omega_T \to I$.

The affine function $z^{(\ell)} : \mathbb{R}^{n_{(\ell-1)}} \to \mathbb{R}(\ell)^{n_{(\ell)}}$ is given by a weight matrix $A^{(\ell)} \in \mathbb{Z}^{n_{(\ell)} \times n_{(\ell+1)}}$ and a bias vector $b^{(\ell)} \in \mathbb{R}^{n_{(\ell)}}$:

$$z^{(\ell)}(\sigma^{(\ell-1)}) := A^{(\ell)} \sigma^{(\ell-1)} + c^{(\ell)} \tag{12}$$

3.2.3 Backforward procedure

Given the input $x_{i,j} \in \mathbb{R}^{nxn}$ and the target outputs $u = [u_1, u_2, \ldots, u_n]^T$, we wish to choose our weights and biases such that $\hat{u}$ is the best approximation of u. After initializing the computation for $u(x_i)$, $i = \{0, \ldots, N\}$ in the iteration, we update the weights in NN by minimizing the residual E for the next iteration. We seek to find the weights and bias to minimize the empirical error in relation to f according to some loss function L:

$$\sum_{j=1}^{N} \sum_{n=1}^{N} L(q(x_{i,j}), f(x_{i,j})) \tag{13}$$

When learning in Sobolev space, all the derivatives of q belong to L^d.

$$J(x_{i,j}, f, q) = ||L(\partial^d f(x_{i,j}) - \partial^d q(x_{i,j}))||^2 + ||\mathcal{L} f|| + \sum_{\alpha=1}^{d} \left\| D_x^i q(\sigma, x_{i,j}) \right\|^2 \tag{14}$$

where f is the PDE function.

Assume we have j units in one hidden layer, so we need to update the weights from inner layer to hidden layer$w_{k,m}^{(\ell)} = (w_{k,1}, \ldots, w_{k,m})$. We use stochastic gradient descent to optimize the weights for the next iteration.

$$\frac{\partial J}{\partial w_n^{(\ell)}}(x_{i,j}, f, q) = \sum_{k=1}^{N} \frac{\partial J}{\partial q_k}(x_{i,j}, f, q) \frac{\partial q}{\partial w_{k,m}^{(\ell)}} \tag{15}$$

$$w_{k,m}^{(\ell)} = w_{k,m}^{(1)} - r * \frac{\partial J}{\partial w_{k,m}^{(1)}}(x_{i,j}, f, q) \tag{16}$$

where r is the learning rate for NN.

We use the new weight getting from (16) and repeat the procedure until finishing 15000 iterations or attaining the convergence condition.

3.3 The function space of neural network

Cybenko provided the proof in [5] that the output space of neural network is dense. Let I_n denote the n-dimensional unit cube, $[0, 1]^n$. The space of continuous functions on I_n is denoted by $C(I_n)$.

(From G.Cybenco)

Theorem 3.2. *Let σ be any continuous function and inputs $\boldsymbol{v}, \boldsymbol{c} \in \mathbb{R}^N, v_i \in \boldsymbol{v}, i = 1, \ldots, N$. If there is a one hidden layer with M hidden units and one output unit, then the finite sums of the form implemented by such a network*

$$q(\sigma, \boldsymbol{v}) = W^T \boldsymbol{v} + \boldsymbol{c} = \sum_{i=1}^{M} \sigma(W_k v_i + c_k) = \sum_{k=1}^{M} \sigma(z_k) \tag{17}$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum $q(\sigma, x_{i,j})$, of the above form, for which

$$|q(\sigma, \boldsymbol{v}) - f(\boldsymbol{v})| \text{ for all, } \boldsymbol{v} \in I_n \tag{18}$$

The elements at layer ℓ of the weight matrix $W^\ell, W^\ell \in \mathbb{R}^{NxN}$ are given by $w^\ell_{k,m}, 0 \leq m \leq N$. In this case, we only consider $\ell = 1$.

Proof. **Claim:** Let $S \subset C(I_n)$ be the set of the function of the form

$$q(\sigma, \mathbf{v}) = \sum_{k=1}^{N} \sigma(W_k \mathbf{v} + c_k)$$

We claim that the closure of S is all of $C(I_n)$.

Assume that the closure of S is not all in $C(I_n)$. Let $R \subset S$ and $g : R \to C(I_n)$. R is a closed proper subspace of $C(I_n)$ By Zorn's lemma, all of extension of g is bounded by p such that

$$g(\mathbf{v}) \leq p(\mathbf{v})$$

By the Hahn-Banach theorem, there is a bounded linear extension L on $C(I_n)$. with the property that $L \neq 0$ but $L(R) = L(S) = 0$. By the Riesz Representation Theorem, the bounded function L has the form

$$L(R) = \int_{I_n} h(\mathbf{v})\phi(\mathbf{v})dx = \int_{I_n} h(\mathbf{v})d\mu(\mathbf{v}) \tag{19}$$

for all $h \in C(I_n)$. where ϕ is the orthonormal basis and μ is the space of finite , signed regular Borel measures on I_n which is denoted by $M(I_n)$.

Since $\sigma(W^T \mathbf{v} + \mathbf{c})$ is in R for all w and b, we must have that

$$L(R) = \int_{I_n} \sigma(W^T \mathbf{v} + \mathbf{c})d\mu(\mathbf{v}) = 0 \tag{20}$$

, implying $\mu = 0$. However, $L \neq 0$. Thus, $\mu = 0$ implies S is not dense which contradicts the assumption. Hence, the subspace S must be dense in $C(I_n)$.

□

3.4 Convergence of the loss function

The approach is inspired by the work of Hornik which proved the universal approximation theorems for neural networks in Sobolev spaces Measuring closeness of functions requires that the activation funcction is nonconstant and derivatives of the approximate function up to order d are bounded. [10]. Let $C^d(\mathbb{R}^N)$ denote the space of all functions f and their derivatives $D^\alpha f$ of order $|\alpha| \leq d$ are continuous on $\mathbb{R}^N$.

For functions $f \in C^d(\Omega), \Omega \subset \mathbb{R}^N$

$$||f||_{d,p,\Omega} = \left[\sum_{|\alpha| \leq d} \int_\Omega |D^\alpha f|^p d\mu \right]^{1/p} \tag{21}$$

Then, $q(\sigma, x)$ is d-dense in weighted Sobolev space which is defined as [10]

$$C^{d,p}(\mu) = \{u \in C^d([0, N] \times \mathbb{R}^N) : ||u||_{d,p,\mu} < \infty\} \tag{22}$$

This means that $u \in C^{d,p}([0, N]x\mathbb{R}^N)$. Observe that $C^{d,p}(\mu) = C^d(\mathbb{R}^m \times \mathbb{R}^N)$, if μ has compact support. In this paper, the loss function is designed for $C^{1,2}(\mu)$.

Then, we use the characteristic to design the solver so that it can minimize the loss function.

Theorem 3.3. *Let $q(\sigma, x_{i,j})$ be approaching solutions of neural network which minimizes loss function $J(f(q, \bigtriangledown_x q, \bigtriangledown_{xx} q, \dots), q)$. Using the results from the proof of Theorem 3 [10], for all $u \in C^d(R^n))$ and $\epsilon > 0$, such that*

$$q(\sigma, x_{i,j}) \in C^{1,2}(\mu) \text{ such that } q(\sigma, x_{i,j}) \to u(x_{i,j}) \text{ as } J(f, q) \to 0$$

The optimizing process minimize the output value and values of its d-th order derivatives with respect to the input $D_x^\alpha q(\sigma, x_{i,j})$ where $\alpha \leq d$.

Therefore, The loss function is constructed as,

$$\begin{aligned} J(x_{i,j}, f, q) = ||L(\partial^d f(x_{i,j}) - \partial^d q(x_{i,j}))||^2 + ||\mathcal{L} f|| \\ + \sum_{\alpha=1}^{d} \|D_x^\alpha q(\sigma, x_{i,j})\|^2 \end{aligned} \tag{23}$$

where d is the highest order of derivative in the equation. This causes the neural network to encode derivatives of the target function in its own derivatives.

Proof. Hornik in Theorem 3 [10] presented that if the activation function $\sigma \in C^d(I^n)$ is nonconstant and bounded, a subset Ω of $C^2(\mathbb{R}^N)$ is uniformly 2-dense on the compacts of $C^2(\mathbb{R}^N)$ and dense in $C^{1,2}(q(\sigma, x_{i,j}))$ with the topology of uniform convergence.

$$\sup_{(t,x)\in\Omega_T} |\partial_t u - \partial_t q(\sigma, x_{i,j})| + \max_{|d|\leq 2} \left|\partial_x^{(d)} u - \partial_x^{(d)} q(\psi, x, t)\right| < \epsilon \tag{24}$$

Assume first that $(u, \bigtriangledown_x u) \to f(t, x, u, \bigtriangledown_x u)$ is locally Lipschitz continuous[2]in $(u, \bigtriangledown_x u)$ with Lipschiz constant. f is the loss function and also the PDE function we aim to solve. This means that

$$\begin{aligned} |f(x_{i,j}, u, \bigtriangledown_x u) - f(t, x, q, \bigtriangledown_x q)| \leq \Big(|u|^{p_1/2} + |\bigtriangledown_x u|^{p_2/2} + |q(\sigma, x_{i,j})|^{p_3/2} + |\bigtriangledown_x q(\sigma, x_{i,j}, t)|^{p_4/2}\Big) \\ (|u - q(\sigma, x_{i,j})| + |\bigtriangledown_x u - \bigtriangledown_x q(\sigma, x_{i,j})|) \end{aligned} \tag{25}$$

for some constants $0 < p_1, p_2, p_3, p_4 < \infty$. We integrate (25) and using Hölder inequality with exponents r_1, r_2:

$$\begin{aligned} &\int_\Omega |f(t, x_{i,j}, u, \bigtriangledown_x u) - f(t, x_{i,j}, q, \bigtriangledown_x q(\sigma, x_{i,j}, t))|^2 \, dv_1(x_{i,j}) \\ &\leq \int_\Omega (|u|^{p_1} + |\bigtriangledown_x u|^{p_2} + |q(\sigma, x_{i,j})|^{p_3} + |\bigtriangledown_x q(\sigma, x_{i,j})|^{p_4}) \\ &\times \left(|u - q|^2 + |\bigtriangledown_x u - \bigtriangledown_x q(\sigma, x_{i,j})|^2\right) dv_1(x_{i,j}) \\ &\leq \left(\int_\Omega (|u|^{p_1} + |\bigtriangledown_x u|^{p_2} + |q(\sigma, x_{i,j})|^{p_3} + |\bigtriangledown_x q(\sigma, x_{i,j})|^{p_4})^{r_1} \, dv_1(x_{i,j})\right)^{1/r_1} \\ &\times \left(\int_\Omega \left(|u - q|^2 + |\bigtriangledown_x u - \bigtriangledown_x q(\sigma, x_{i,j})|^2\right)^{r_2} dv_1(x_{i,j})\right)^{1/r_2} \\ &\leq K \left(\int_\Omega \left(|u - q|^2 + |\bigtriangledown_x u - \bigtriangledown_x q(\sigma, x_{i,j}, t)|^2\right)^{r_2} dv_1(x_{i,j})\right)^{1/r_2} \\ &\leq K \left(\int_\Omega \left(|u|^{p_1} + |\bigtriangledown_x u|^{p_2} + |q(\sigma, x_{i,j})|^{\max\{p_1 p_3\}} + |\bigtriangledown_x q(\sigma, x_{i,j})|^{\max\{p_2, p_4\}}\right)^{r_1} dv_1(x_{i,j})\right)^{1/r_1} \\ &\times \left(\int_\Omega \left(|u - q(\sigma, x_{i,j})|^2 + |\bigtriangledown_x u - \bigtriangledown_x q(\sigma, x_{i,j})|^2\right)^{r_2} dv_1(x_{i,j})\right)^{1/r_2} \\ &\leq K(\epsilon^{p_1} + \epsilon^{p_2} + sup|u|^{\max\{p_1 p_3\}} + sup|\bigtriangledown_x u|^{\max\{p_2 p_4\}}) \leq K\epsilon^2 \end{aligned} \tag{26}$$

□

4 Experiments and Results

We use tensorflow to implement our multilayered model. The algorithm is tested on a class of high-dimensional PDEs. The results shows that the solution calculated from our model satisfies the f in nonlinear PDEs (1) to zero.

The results shows that NN is good at solving the problem which boundary condition determine the whole shape.

[2]Suppose $A \subset \mathbb{R}^n$ is open and $f : A \to \mathbb{R}^n$ is differentiable. A function is locally Lipschitz continuous if there exists a contant $K > 0$ and $\delta > 0$such that $|x_1 - x_2| > \delta$ implies $|q(x_1) - q(x_2)| \leq K|x_1 - x_2|$.

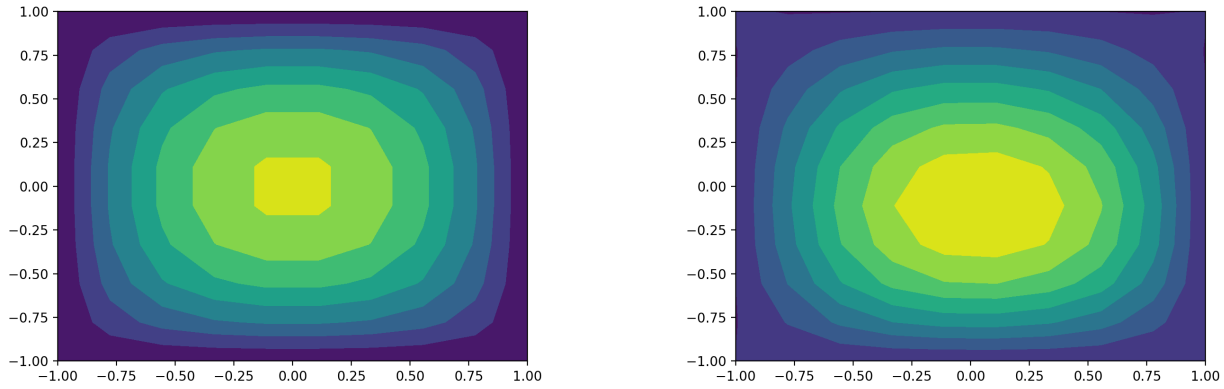
4.1 Example 1:

We take a poission equation for example.

$$-\bigtriangledown^2 u = 1, x, y \in \Omega$$
$$u(x, y) = 0, x, y \in \partial\Omega$$

By training the deep learning model with the boundary domain, it can forecast the inner domain of u values.

The predicting result is shown in Figure 2. The interesting in the experiment is that if we train the neural network for the whole dataset, it failed to approximate result.



(a) The true solutions of Example 1.

(b) The NN solutions from Example 1.

Figure 2: The solutions generated from NN compared with the ground true solutions in Example 1.

4.2 Example 2:

Let's take a another poisson equation for example. However, unlike the first example, we set the second order derivative of u $-\bigtriangledown^2 u$ to a constant:

$$-\bigtriangledown^2 u = 30\pi^2 \sin(5\pi x)\sin(6\pi y), x, y \in \Omega$$
$$u(x, y) = 0, x, y \in \partial\Omega$$

The predicting result is shown in Figure 3. In this example, since it involves sine and cosine, it has the periodic property. Therefore, the whole picture cannot develop based on the boundary domain. If we trained NN is with boundary domain and collected the points which value of x axis $< 1/5$ and value of y axis $< 1/6$ the result is presented as Figure 3b.

However, if we get rid of the boundary domain and only trained with the value of x axis $< 1/5$ and value of y axis $< 1/6$ the result is presented as Figure 3c. The computed result is better than training with boundary domain.

Therefore, we conclude that when the differential operator of Poisson equation is not constant, the boundary domain is not enough for training. In addition, we need to concern the periodic in equation. In example 2, we have two sine terms in the differential operator. When $x \leq 2/5$, the $\sin(5\pi x)$ can goes from $\sin(0)$ to $\sin(2\pi)$ so that it can finish a period. In the same way, when we start from the end point of the domain $(1, 1)$, the data points which $x \geq 4/5$, the $\sin(5\pi x)$ can goes from $\sin(0)$ to $\sin(\pi)$ so that it can finish a period.

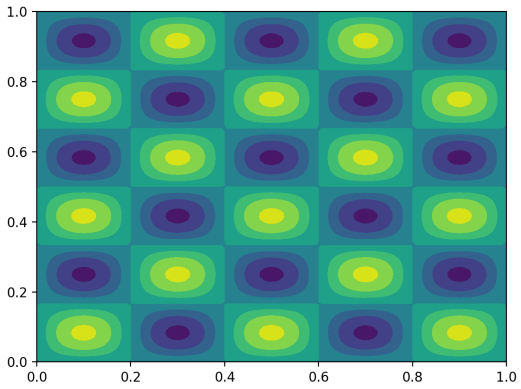
5 Conclusion

References

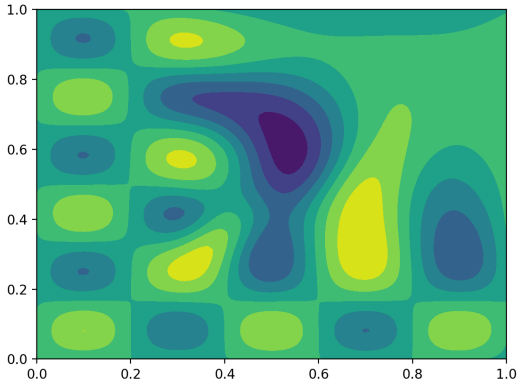
[1] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *J. Mach. Learn. Res.*, 19(1):932–955, January 2018.

[2] Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural PDE solvers with convergence guarantees. *CoRR*, abs/1906.01200, 2019.

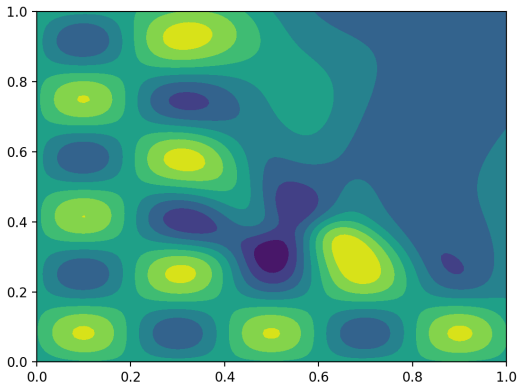
[3] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1), 3 2018.



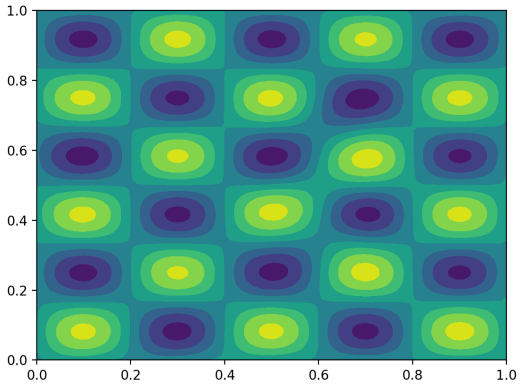
(a) The true solution of Example 2.



(b) The NN solutions from Example 2 is trained with boundary values and the cooridinates value of x axis $< 1/5$ and value of y axis $< 1/6$.



(c) The NN solutions from Example 2 is trained with the cooridinates value of x axis $< 1/5$ and value of y axis $< 1/6$.



(d) The NN solutions from Example 2 is trained with the cooridinates value of x axis $\leq 2/5$ or value of y axis $\leq 1/6$ and cooridinates value of x axis $\geq 4/5$ or value of y axis $\geq 5/6$.

Figure 3: The solutions generated from NN compared with the ground true solutions in Example 2.

[4] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving for high dimensional committor functions using artificial neural networks. *CoRR*, abs/1802.10275, 2018.

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[6] David A. Sprecher and Sorin Draghici. Space-filling curves and kolmogorov superposition-based neural networks. *Neural networks : the official journal of the International Neural Network Society*, 15 1:57–67, 2002.

[7] David Sprecher. On computational algorithms for real-valued continuous functions of several variables. *Neural Networks*, 59:16 – 22, 2014.

[8] Robert H. Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the IEEE First International Conference on Neural Networks* (San Diego, CA), volume III, pages 11–13. Piscataway, NJ: IEEE, 1987.

[9] V. Kurkova. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5(3):501 – 506, 1992.

[10] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[11] Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. *CoRR*, abs/1805.07091, 2018.