ECE 397
Individual Study in ECE

Independent Study Report
on
Solving PDE by Deep Learning

Ziyang Xu
Term: Spring 2019
Prof. Volodymyr, Kindratenko

Table of Content

Introduction

This semester, Enyi Jiang and I were expected to do an individual study under Prof. Volodymyr, Kindratenko with the help of his cooperative researcher, Shirui Luo, about the application on deep learning. Shirui Luo is working on deep learning for CFD (computational fluid dynamics) and he also puts forwards another potential and valuable topic which is about deep learning for PDE (partial differential equations). Since this is an individual study course, Enyi and I were expected to learn from different fields, and in this case, Enyi continued to study deep learning for CFD and I chose to learn something new about deep learning for PDE.

This is really interesting and challenging. Deep learning has shown its powerful ability in many applications recently. It has contributed to many challenging topics, say, image recognition, text recognition and some other transfer learnings. For example, using deep learning methods, we train a neural network model which is able to recognize images or objects. We can in turn estimate the price of the objects through the results of an additional estimator which accepts the outputs of the network as its inputs. This kind of transfer learning makes the deep learning more practical, potential and developable.

Also, shortly after the formation of calculus theory, people began to use partial differential equations to describe, explain or foresee various natural phenomena, and apply the obtained research methods and research results to various sciences and engineering techniques. Significant results have been achieved, showing the importance of partial differential equations for humans to understand the fundamental laws of nature. Gradually, the study of partial differential equations based on practical problems in various sciences such as physics and mechanics has become one of the most important contents in traditional applied mathematics. It is directly related to many natural phenomena and practical problems, and constantly generate new topics and new methods that need to be addressed.

Among many practical topics, many topics (especially national defense issues) are impossible or difficult to study by engineering test methods, say, the risk factor is large and it is expensive. So it is necessary to reduce the number of trials as much as possible or give a more accurate estimate before the test. With the advent of electronic computers and the development of computing technology, electronic computers have become an important tool for solving these practical problems.

Analysis of the Problem

To attempt to apply deep learning method to solving partial differential equations, we have to figure out several challenging points that we will confront. Based on these points, we may find some ideas about this problem.

The main challenging point is that the partial differential equation is quite diverse from one to the others. What kind of PDE we should firstly concentrate on needs think twice because the entry point of a problem always critical and decisive. For example, for a (string vibration equation) wave equation:

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \cdot \frac{\partial^2 u}{\partial t^2}$$

the solution varies and depends on the initial conditions: $[u(x,0)]$ and $[\partial u(x,0)/\partial t]$.

Another challenging point is that the algorithm of the deep learning for partial differential equations. Here I was recommended to use tensorflow to implement the algorithm due to its strong and convenient functions in solving deep learning problems. Previously, I used to write code in python with numpy to establish neural network and now I need to learn a new method to establish neural network. I think the new method will be more effective and efficient.

The third challenging point is the data set production. For deep learning, one of the most important components is the data set. We need a lot of data to training the network parameters (weights and bias, etc). But obviously, we do not have such data set as support, thus, we need to create some data sets by ourselves.

The last challenging point is that we should clarify in advance what kind of data or how many parameters we are supposed to input to the neural network and what kind of data we expect to get from the network. How to make the results visualizable also needs to be taken into consideration.

As for the optimization part, I don't want to involve it up till now.

Approach

In order to have a basic concept of PDEs, tensorflow which is the google library for machine learning, and how to use tensorflow to build a neural network for solving some types of PDEs, I acquired some sources from the websites.

I have learned the basic usage and syntax of the google library, tensorflow from a series of video lectures [2] and the way to apply it to establish neural networks, say, CNN, RNN. It also teaches me how to use dropout functions to deal with the overfitting problems and transfer learning. This strengthens my ability to understand others codes and do some exercises by myself.

Additionally, during this individual study, I mainly study the topic by reading related papers to acquire ideas. Some papers are provided by my instructor and some papers are found from the websites by myself. Following my instructor's suggestion, I can begin with a, relatively speaking, simple paper to begin with. I also read some other papers to understand the topic comprehensively and the methods to deal with the topic diversely. Below is a general thought that I conclude after I look through some papers, combined with the knowledge about the PDEs and the technical methods I have.

Thought:

We can notice that a neural network can accept some parameters (we call them features) as inputs, and afterwards, the network will use these features to target a corresponding set of outputs and output them. According to the suggestion from the instructor, we expected to train a neural network that can accept different values of the parameters (this parameters can even be functions or something else). As for the output, we do not expect to get an accurate solution of the specified partial differential equation, such as each term and its corresponding coefficient of the solution. We can expect to get a set of numerical outputs.

Review

Following is review of some of papers that I have read.

- In *Deep Learning for Partial Differential Equations*, Xu (2018) and his teammates mainly focus on the Laplace PDEs, expressed as:

$$\Delta u(x,y) = f(x,y)$$
$$u(x,y) = g_D(x,y)$$

and the first equation holds Dirichlet boundary condition.

They consider several groups of expressions as examples:

1. A well-behaved Solutions

$$u(x,y) = sin\pi x \cdot sin\pi y, \quad (x,y) \in [0,1]^2$$
$$f(x,y) = \Delta u(x,y) = -2\pi^2 sin\pi x \cdot sin\pi y$$

2. Solutions with a Peak

$$u(x,y) = exp[-1000(x-0.5)^2 - 1000(y-0.5)^2] + v(x,y)$$

where $v(x,y) = sin(\pi x)$ for illustration

3. Less Regular Solutions

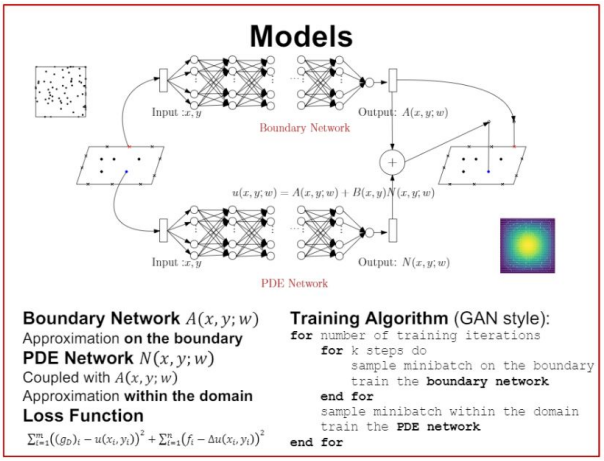
$$u(x,y) = y^{0.6}$$

4. High dimensions

This paper is easy to begin with. It introduces the method of how they generate the dataset and the algorithm for approximate u with a neural network. They generate data randomly on both the boundary of a specified range and the inner domain. They use the equation:

$$u(x,y;\omega_1,\omega_2) = A(x,y;\omega_1) + B(x,y) \cdot N(x,y;\omega_2)$$

to establish the neural network and solve the PDE. The loss function involves the boundary part and inner domain parts, as is shown below.

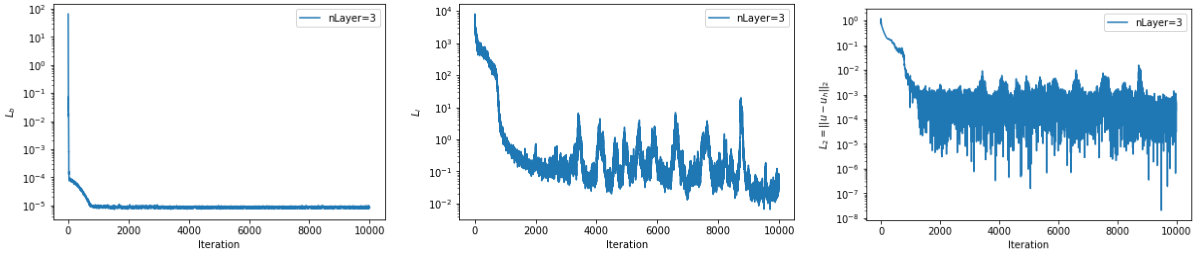


- In *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations,* E (2017) and his teammates discussed some nonlinear partial differential equations and nonlinear backward stochastic differential equations (BSDEs). They talked about the probability of solving different types of partial differential equations with high dimension. The following types of high-dimensional equations can be solved: Allen-Cahn equation, HJB equation, Derivative Pricing PDE and Burger-type PDE. But from the code we can find that it just use a selector to choose the neural network by the parameter.

- In *DGM: A deep learning algorithm for solving partial differential equations,* Sirignano (2018) used a meshfree deep learning algorithm to solve high-dimensional PDEs. They tested the deep learning algorithm on Hamilton-Jacobi-Bellman PDE with accurate results.

Simulation

- I refer the code provided on the website to simulate the project Xu did. He used tensorflow to construct their neural networks. Here is the source code: https://github.com/kailaix/nnpde



They generally use tanh as activation function. The network works well if the solution is well-behaved. If the solution has a peak, then the network works well on the boundary of the mesh while the inner domain shows a divergent trend and cannot fit the exact values. If the derivative of the solution approaches infinity as $y \to \infty$, the loss will oscillate across the iterations. They also tried higher dimensions by varying the number of layers. Though they are limited by the computer power which cannot support a larger number of iteration, they still found that the higher dimension is, the more iteration the neural network should do because their network generally reduces the loss while iterating.

- I refer the code on the website to solve High Dimensional PDE using BSDE and Deep Neural Network. The PDE to be solved could be Allen-Cahn equation, HJB equation, Derivative Pricing PDE and Burger-type PDE. To run this code, we firstly open the command line and then input “python BSDE.py --equation=PDE” where PDE could take value 'Allen-Cahn', 'HJB', 'Pricing' or 'Burger'. Here is the source code: https://github.com/sunny721330/PDE-Solver

Run “HJB”

```
step :     0 , loss : 1.7935e+01 ,  Y0 : 3.5068e-01 , runtime :   20
step :   100 , loss : 4.9026e+00 ,  Y0 : 1.2493e+00 , runtime :   35
step :   200 , loss : 3.2428e+00 ,  Y0 : 1.5438e+00 , runtime :   43
step :   300 , loss : 2.9701e+00 ,  Y0 : 1.7567e+00 , runtime :   50
step :   400 , loss : 2.6832e+00 ,  Y0 : 2.0169e+00 , runtime :   58
step :   500 , loss : 2.3467e+00 ,  Y0 : 2.3128e+00 , runtime :   66
step :   600 , loss : 1.9855e+00 ,  Y0 : 2.6221e+00 , runtime :   73
step :   700 , loss : 1.5984e+00 ,  Y0 : 2.9585e+00 , runtime :   81
step :   800 , loss : 1.1995e+00 ,  Y0 : 3.3153e+00 , runtime :   89
step :   900 , loss : 7.9714e-01 ,  Y0 : 3.6753e+00 , runtime :   96
step :  1000 , loss : 4.2444e-01 ,  Y0 : 4.0114e+00 , runtime :  104
step :  1100 , loss : 1.8318e-01 ,  Y0 : 4.2887e+00 , runtime :  112
step :  1200 , loss : 6.2140e-02 ,  Y0 : 4.4734e+00 , runtime :  119
step :  1300 , loss : 2.9726e-02 ,  Y0 : 4.5634e+00 , runtime :  127
step :  1400 , loss : 2.4835e-02 ,  Y0 : 4.5875e+00 , runtime :  135
step :  1500 , loss : 2.3544e-02 ,  Y0 : 4.5957e+00 , runtime :  143
step :  1600 , loss : 2.3562e-02 ,  Y0 : 4.5970e+00 , runtime :  151
step :  1700 , loss : 2.3134e-02 ,  Y0 : 4.5987e+00 , runtime :  158
step :  1800 , loss : 2.3456e-02 ,  Y0 : 4.5988e+00 , runtime :  165
step :  1900 , loss : 2.4242e-02 ,  Y0 : 4.5972e+00 , runtime :  173
step :  2000 , loss : 2.3709e-02 ,  Y0 : 4.5978e+00 , runtime :  181
running time : 181.477s
```

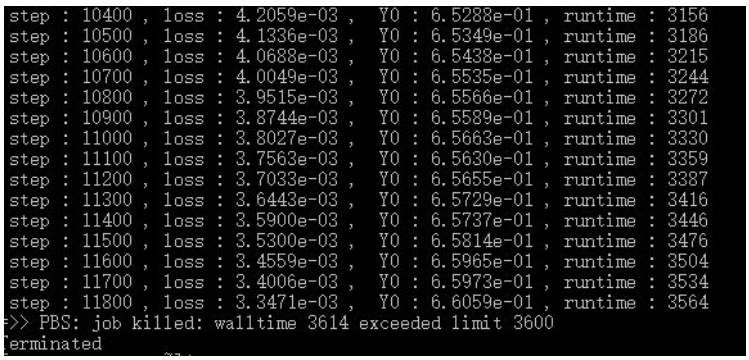
Run “Allen-Cahn”

```
step :  1900 , loss : 2.1362e-04 ,  Y0 : 5.3233e-02 , runtime :  175
step :  2000 , loss : 2.0955e-04 ,  Y0 : 5.3074e-02 , runtime :  182
step :  2100 , loss : 2.1177e-04 ,  Y0 : 5.2759e-02 , runtime :  190
step :  2200 , loss : 2.1202e-04 ,  Y0 : 5.3032e-02 , runtime :  198
step :  2300 , loss : 2.0939e-04 ,  Y0 : 5.2853e-02 , runtime :  206
step :  2400 , loss : 2.0413e-04 ,  Y0 : 5.2690e-02 , runtime :  214
step :  2500 , loss : 2.0400e-04 ,  Y0 : 5.2719e-02 , runtime :  221
step :  2600 , loss : 1.9421e-04 ,  Y0 : 5.2857e-02 , runtime :  229
step :  2700 , loss : 1.9333e-04 ,  Y0 : 5.3046e-02 , runtime :  237
step :  2800 , loss : 1.8908e-04 ,  Y0 : 5.2808e-02 , runtime :  245
step :  2900 , loss : 1.8609e-04 ,  Y0 : 5.2786e-02 , runtime :  253
step :  3000 , loss : 1.8257e-04 ,  Y0 : 5.2832e-02 , runtime :  260
step :  3100 , loss : 1.7855e-04 ,  Y0 : 5.2591e-02 , runtime :  268
step :  3200 , loss : 1.7290e-04 ,  Y0 : 5.2951e-02 , runtime :  276
step :  3300 , loss : 1.6593e-04 ,  Y0 : 5.2988e-02 , runtime :  284
step :  3400 , loss : 1.6468e-04 ,  Y0 : 5.2959e-02 , runtime :  291
step :  3500 , loss : 1.5871e-04 ,  Y0 : 5.2641e-02 , runtime :  299
step :  3600 , loss : 1.5572e-04 ,  Y0 : 5.2696e-02 , runtime :  307
step :  3700 , loss : 1.5163e-04 ,  Y0 : 5.2859e-02 , runtime :  314
step :  3800 , loss : 1.4587e-04 ,  Y0 : 5.2942e-02 , runtime :  322
step :  3900 , loss : 1.4199e-04 ,  Y0 : 5.2982e-02 , runtime :  330
step :  4000 , loss : 1.4012e-04 ,  Y0 : 5.3355e-02 , runtime :  338
running time : 338.540s
```

Run “Pricing”

```
step :  8000 , loss : 3.8158e-02 ,  Y0 : 1.9534e-01 , runtime :  716
step :  8100 , loss : 2.5020e-02 ,  Y0 : 1.5818e-01 , runtime :  724
step :  8200 , loss : 1.6047e-02 ,  Y0 : 1.2668e-01 , runtime :  732
step :  8300 , loss : 1.0056e-02 ,  Y0 : 1.0028e-01 , runtime :  741
step :  8400 , loss : 6.1492e-03 ,  Y0 : 7.8417e-02 , runtime :  749
step :  8500 , loss : 3.6642e-03 ,  Y0 : 6.0533e-02 , runtime :  757
step :  8600 , loss : 2.1248e-03 ,  Y0 : 4.6095e-02 , runtime :  766
step :  8700 , loss : 1.1972e-03 ,  Y0 : 3.4600e-02 , runtime :  774
step :  8800 , loss : 6.5438e-04 ,  Y0 : 2.5581e-02 , runtime :  783
step :  8900 , loss : 3.4644e-04 ,  Y0 : 1.8613e-02 , runtime :  791
step :  9000 , loss : 1.7733e-04 ,  Y0 : 1.3316e-02 , runtime :  799
step :  9100 , loss : 8.7594e-05 ,  Y0 : 9.3592e-03 , runtime :  808
step :  9200 , loss : 4.1674e-05 ,  Y0 : 6.4555e-03 , runtime :  816
step :  9300 , loss : 1.9057e-05 ,  Y0 : 4.3654e-03 , runtime :  824
step :  9400 , loss : 8.3572e-06 ,  Y0 : 2.8909e-03 , runtime :  833
step :  9500 , loss : 3.5067e-06 ,  Y0 : 1.8726e-03 , runtime :  841
step :  9600 , loss : 1.4043e-06 ,  Y0 : 1.1850e-03 , runtime :  849
step :  9700 , loss : 5.3533e-07 ,  Y0 : 7.3166e-04 , runtime :  858
step :  9800 , loss : 1.9371e-07 ,  Y0 : 4.4013e-04 , runtime :  866
step :  9900 , loss : 6.6340e-08 ,  Y0 : 2.5757e-04 , runtime :  874
step : 10000 , loss : 2.1434e-08 ,  Y0 : 1.4640e-04 , runtime :  883
running time : 883.369s
```

Run “Burger”



This code source can only support four types of equations.

- Here is another code source that refers the paper, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations.* These codes can add new problems very easily by inheriting the class equation in equation.py and define the new problem.
  https://github.com/frankhan91/DeepBSDE
  They added another two types of equations, quadratic gradients and reaction diffusion, which are also mentioned in the paper, showing that their codes can accept more equations to solve. They use fully connected neural network model.

Conclusion

Firstly, let me talk about what I have learned. During this semester’s individual study, I gradually acquainted myself with the process of the entire individual study and doing research. I find that the most difficult thing is not the knowledge itself but that we do not know where we should begin.

I am still confused about the potential of the neural networks and its limitations. From above, we can find that what we have done is provide a lot of input data to the NN and minimize the difference between the output values generated from the NN and the corresponding values from the given solution of PDEs. After we have trained the NN, we can define the value of the weights and biases. We can certainly use this model to get a rather perfect prediction of a similar equation.

Next, a critical problem occurs: we want use deep learning to build a NN that can solve partial differential equations. We at least want to solve a small range types of partial differential

equations by deep learning. However, up till now, we just approximate the solution of a partial differential equations. We even cannot guarantee whether applying the NN to some other similar partial differential equations can also make sense or not. What we are doing actually does not solve a partial differential equation but fit a large number of numerical values to the given solution. This is absolutely a challenging topic, for example, from [1] we can find that even a common partial differential equation (string vibration equation) has various solutions as long as the given initial conditions vary.

As is mentioned before, the thought that inputting some parameters as features to the neural network sounds effective. We can suppose that the parameters or features represent the initial conditions of the string vibration equation. If possible, the trained network will output a set of data that simulate the solution of the partial differential equation with given initial conditions.

Reference

[1] different solutions of the string vibration equation due to the different initial conditions:
http://msvlab.hre.ntou.edu.tw/grades/PDE(2008)/%E5%81%8F%E5%BE%AE%E5%88%86%E6%96%B9%E7%A8%8B_%E4%B8%80%E7%B6%AD%E5%BD%88%E6%80%A7%E6%B3%A2.pdf
[2] A Series of video lectures about applying tensorflow on build advanced neural network.
https://morvanzhou.github.io/tutorials/machine-learning/tensorflow/
[3] Deep learning for Partial Differential Equations CS230
http://stanford.edu/~kailaix/files/Deep_Learning_for_Partial_Differential_Equations.pdf
[4] W. E, J. Han, A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations,* Communications in Mathematics and Statistics, Springer, 2017.
https://arxiv.org/pdf/1706.04702.pdf
[5] PDE-solver using BSDE
https://github.com/sunny721330/PDE-Solver
[6] Using BSDE to solve PDE
https://github.com/frankhan91/DeepBSDE
[7] Sirignano JA, Spiliopoulos K. *DGM: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics.* 2018 Dec 15
https://doi.org/10.1016/j.jcp.2018.08.029

Following sources are valuable and explorable, but I cannot understand them from my point of view. [9] is the corresponding source code for [8].

[8] Long, Zichao and Lu, Yiping and Ma, Xianzhong and Dong, Bin. *PDE-Net: Learning PDEs from Data,* Proceedings of the 35th International Conference on Machine Learning (ICML 2018) https://arxiv.org/abs/1710.09668
[9] source code of the paper: *PDE-Net: Learning PDEs from Data* https://github.com/ZichaoLong/PDE-Net/tree/PDE-Net