Northeastern University

Neural Net Based Boundary Value Problem Solver

Chichen Huang

June 2020

Executive Summary

Boundary Value Problem (BVP) is a common problem in math and engineering field. A steady state BVP in 3D domain $\Omega(x, y, z)$ has a general form:

$$L(u,v,w) = f \quad \text{in} \quad \Omega$$

L(.) is differential operator. Dependent variable *u, v, w* is only a function of the spatial dimensions such that *u=u* (*x*, *y*, *z*). $f=f(x, y, z)$ is a function that represents the internal effects that act on the system.

Two kind of boundaries are applied on boundaries of domain Ω. *Essential boundary condition* (EBC) represents a prescribed value for a dependent variable; *natural boundary condition* (NBC) typically describes the external effects that cause a change in the system. They have the form:

EBC: $u = u_b(t) \text{ on } \Gamma_d$

NBC: $B(u,v,w) = g(t) \text{ on } \Gamma_t$

B(.) is differential operator. Γ is boundary of Ω.

In part 1, I designed a Neural Network (NN) based solver to solve BVP. In part 2, the NN is modified to estimate coefficient of BVP. The solver is written in Python with **TensorFlow 2.2.0** framework. The class **BVPSolver ()** receives boundaries and BCs as input tensors and a layer vector to specify the architecture of NN. The outputs are 2 tensors containing calculated dependent variables and residual.

Example Problem

The example problem is from FEM Lecture Notes written by *Professor Sinan Müftü.*

It can be interpreted as solving the deformation of a loaded elastic bar.

Example 3.9: Solve the following BVP by using Galerkin's method,

BVP: $-\frac{d}{dx}\left(a(x)\frac{du}{dx}\right) - u = q(x) \quad \text{in } 0 < x < L$

BCs: $u(0) = 0 \ , \ a\frac{du}{dx}\Big|_{x=1} = Q_0$

Consider this problem for $L = a(x) = Q_0 = 1$; $q(x) = -x^2$ and use a two term polynomial approximation (i.e. $N = 2$).

Then the approximate solution with the Galerkin method becomes:

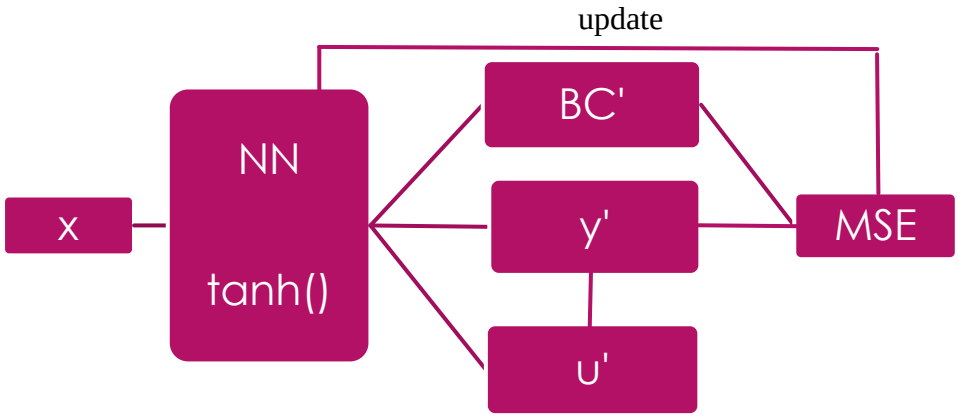
$$u_N = -\frac{x}{2153}\left(-2776+301x+7x^2\right)$$

Analytical Solution: Show that the analytical solution of this problem is:

$$u = \frac{2\cos(1-x)-\sin x}{\cos 1}+x^2-2$$

Part 1- BVP solver

In this 1-D steady state problem, a NN with 2 fully connected hidden layers is used. Each layer has 20 neurons. Input and output are 1-D vectors.



x is 100 collocations points $\in [0,1]$. y is output tensor representing the residual such that $y = -\frac{du}{dx}\left(a(x)\frac{du}{dx}\right) - u - q(x)$. For exact solution, y=0.

$MSE = MSE_y + MSE_{NBC} + MSE_{EBC}$

Where,

$MSE_y = \frac{1}{N}\sum y'^2$

and

$MSE_{NBC} = \frac{1}{N_{NBC}}\sum(NBC - NBC')^2$

and

$MSE_{EBC} = \frac{1}{N_{EBC}}\sum(EBC - EBC')^2$

y', BC' are calculated from NN and used to calculate the loss function MSE. The first term MSE_y makes sure the residual goes to zero. The other 2 terms ensure all BCs are satisfied.

The network is optimized with Adam optimizer.

The approximate solution is considered obtained when MSE is under the threshold.

The optimization is considered failed when it reaches max iteration while MSE is still high.

Test Result

Training data x = linspace (1,1,100).

Max number of iterations is 100. Hidden layers= [20 20].

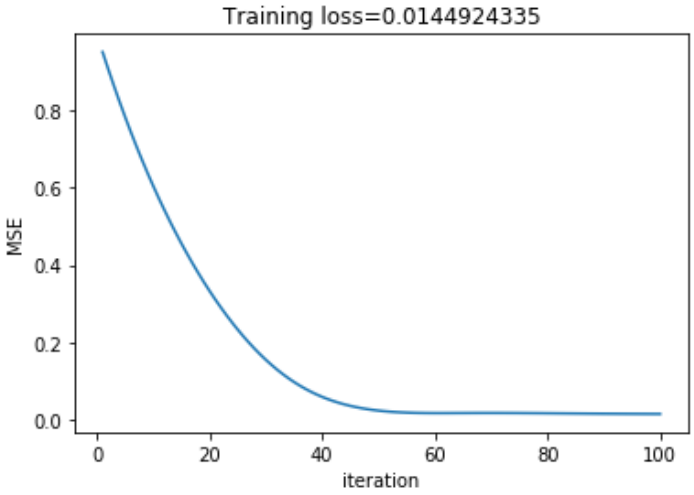
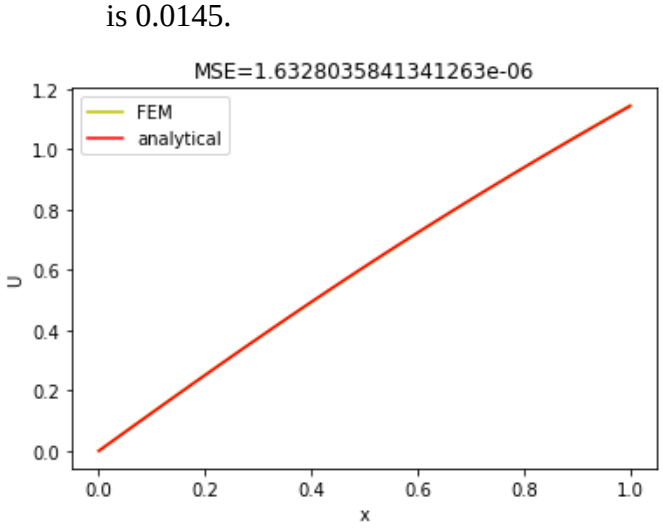


Fig. Training loss over iteration

The network converges around 60 iteration with loss equals to 0.0167. The final loss is 0.0145.



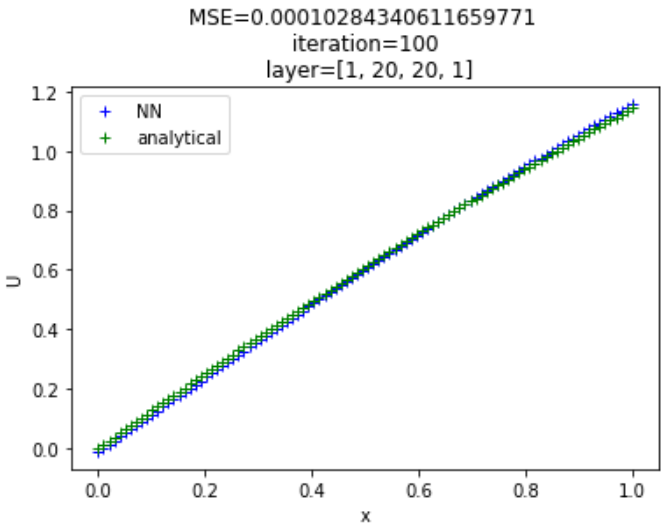
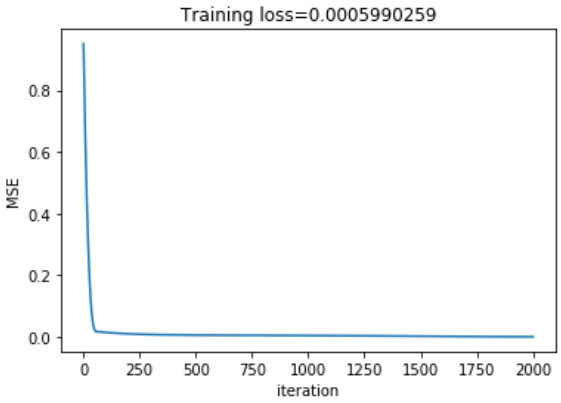


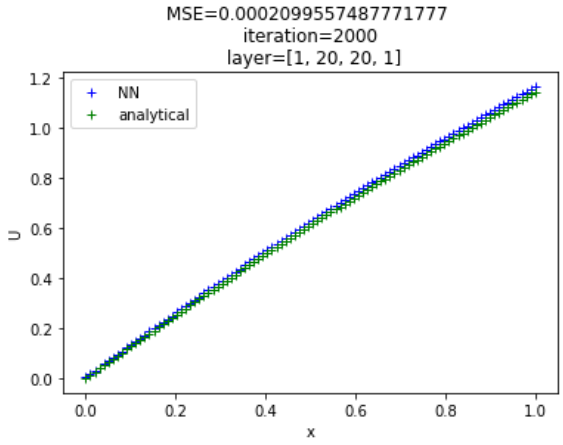
Fig. Comparsion between NN and FEM

$MSE_u = \frac{1}{N}\sum(u - u')^2$ is used to evaluate the preformance.

The above figure shows the result of NN has a MSE=1e-4, while the FEM give a result with MSE=1.6e-6. FEM give more acurracy solution.

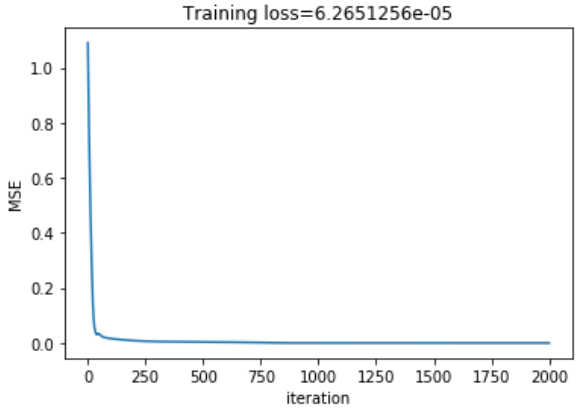
Max number of iterations is 2000. Hidden layers= [20 20].

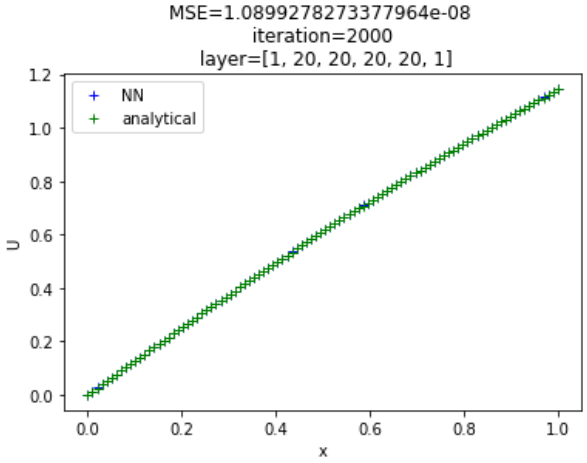




The result is slightly better in terms of training loss. However, MSE_u is higher than previous test. It implies a better approximation of NBC.

Max number of iterations is 2000. Hidden layers= [20 20 20 20].





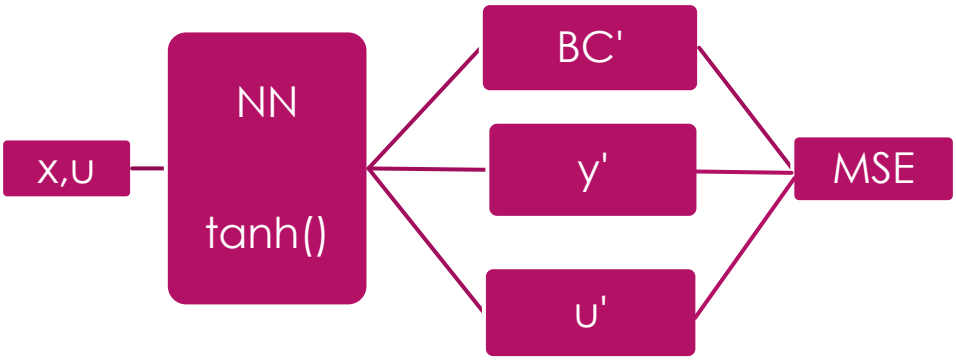
By adding two more layers, the resulting MSE_u is now lower than that of FEM solution.

Part 2- Coeff. Estimator

In this part, I assumed that the coeff. in the example problem is unknown:

$$y = -\frac{du}{dx}\left(a(x)\frac{du}{dx}\right) - b * u - c * q(x).$$

a, b are unknown and need to be estimated; c =1



The estimator problem looks alike the previous solver one. There are only two differences. First, the training set is now (x, u) instead of x. And the MSE becomes

$MSE=MSE_y+ MSE_u +MSE_{NBC}+MSE_{EBC}$

Where

$$MSE_u = \frac{1}{N}\sum(u - u')^2$$

Second, residual y' in this problem is computed given true u, while in previous problem, it is computed with u'.

The NN is optimized with Adam optimizer. Then, Lbfgs minimizer is used to estimate the coeff.

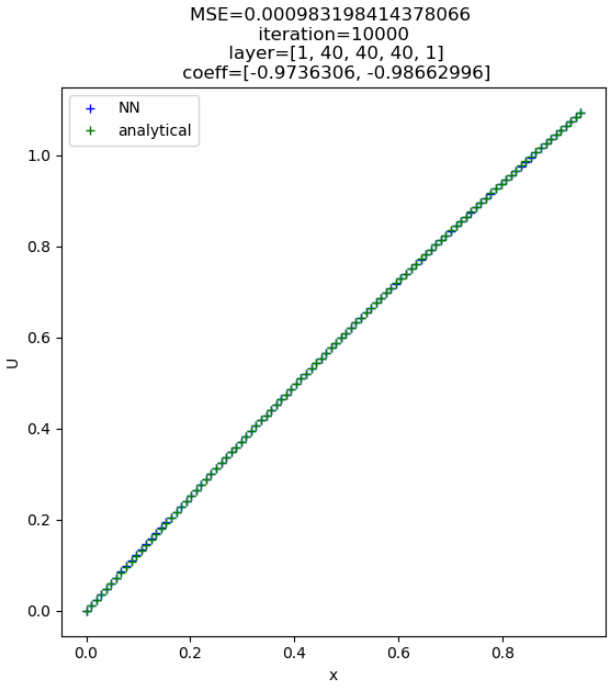
Test Result

Training set (x,u) contains 100 collocations points $x \in [0,1]$ and u which is corresponding solution to x. Boundary conditions are also given. The exact coeff. is (-1, -1). The initial guess is (-1.3, -0.5).

HIDDEN LAYER	ESTIMATED COEFF.
6*40	-1.230, -1.089
4*40	-1.195, -1.075
3*40	-1.154, -1.059
2*40	-1.346, -1.133

The above table shows the result of experiment under same optimizer. The experiment with 3 hidden layers gives best result. As the number of layers increase further, the model starts to overfit. However, the error = (-15% ,- 5%) is high than expected.

Another test is run under different optimizer setup, where learning rate is 0.0005 (was 0.001 in previous tests) and apply AMSGrad variant to this algorithm.



The above figure shows a better result with lower learning rate. error= (2.6%, 1.4%).

To Be Done In the Future

The example problem is 1-D and has small nonlinearity. It can be easily modified to solve partial differential equation. In addition, more complicated problems should be tested to verify the robustness.