Automated Architecture Testing

Hans C. Suganda

8^{th} November 2021

Contents

0.1 Rationale & Process

Testing the various Neural Network Architectures, renaming and removing the initial weight files is a time-consuming process. To make the process run faster and also in the process test a large variety of Neural Network architecture beyond human capability, it is important to automate Neural Network Architecture testing.

The architecture tested is a neural network architecture with a single input node and a single output node with uniformly sized hidden layers (number of nodes in hidden layers remain constant for all hidden layers). The neural network is then run to solve the 1-dimensional Helmholtz equation and the performance of the neural network is measured by the error produced by the neural network over its entire computational domain after it has been trained for a sufficiently large number of iterations. The better the performance of the Neural Network Architecture, the lesser the error along its entire computational domain.

The Neural Network is trained over a large number of iteration (50000) so that it could be assumed that the neural network has converged sufficiently and that any error produced by the neural network can be attributed only to the architecture of the network. For this case, the activation functions for all hidden layers are declared as 'tanh'

0.2 Bash Script

Bash (Bourne Again Shell) is a common shell that is used in many linux and unix servers. Therefore, writing automation in bash is scalable as it is compatible with common computer super-clusters. Although it is possible to automate the various tasks within python itself, I attempt to minimize the changes necessary to the Neural Network Script in itself. Due to these 2 reasons, it seems rational to implement the automation in bash.

The automation is implemented via bash scripts. Once provided with user input for the minimum width, depth, the steps and the corresponding number of steps, the script runs a series of process that results in the "performance" of the network described in a `NN_Architecture_Error.txt`. The Bash script is shown below,

```
#!/bin/bash

#Author: Hyahoos

#Name of the NN_Train training Algorithm
NN_Train="helm1d_locdnn_dgm_adam_unicolloc.py"
Error="Error_Analysis.py"

#Read Width Range
echo -n 'Minimum Width: '; read MinWidth
echo -n 'Width Step: '; read StepWidth
echo -n 'Number of Width Steps: '; read NumStepW

#Read Depth Range
echo -n 'Minimum Depth: '; read MinDepth
echo -n 'Depth Step: '; read StepDepth
echo -n 'Number of Depth Steps: '; read NumStepB

#Printing Mode
echo -n 'Log Level: '; read log

#Initialization of Width and Depth
```

```
declare -i width=0
declare -i depth=0

#Main Loop Body
for ((indexw = 0; indexw <= NumStepW; indexw++)) ;do
    for ((indexb = 0; indexb <= NumStepB; indexb++)) ;do

        #Updating the Width and Depth
        width=$((indexw * StepWidth + MinWidth))
        depth=$((indexb * StepDepth + MinDepth))

        #Possible Show Progress
        if (($log >= 1)); then
        echo  'Running␣for␣Width:␣' $width '␣Depth:␣' $depth
        fi

        #Run Neural Network Given the Width and Depth
        if (($log >= 2)); then
            python $NN_Train $width $depth #Show NN_Train outputs
        else
            python $NN_Train $width $depth  > /dev/null #Hide NN_Train outputs
        fi

        #Go to where data is located
        cd data/Helm1D_LocDNN_DGM_Adam_UniformColloc/

        #Rename output files
        mv helm1d_locdnn_dgm_adam_soln.dat* "Solution_"$width"_"$depth".dat"
        mv helm1d_locdnn_dgm_adam_param.dat* "Parameter_"$width"_"$depth".dat"
        mv helm1d_locdnn_dgm_adam_weights.hd5* "WeightFile_"$width"_"$depth".dat"
        rm helm1d_locdnn_dgm_adam_init_weights.hd5*

        #Run the Post-Processing Utilities
        python $Error "Solution_"$width"_"$depth".dat" $width $depth >> NN_Architecture_Error.
    txt

        #Go back to original directory
        cd ../..
    done
done

echo 'Cannon␣Ringing...'
```

0.3 Dependent Python Scripts

Minor modifications must be made the neural network script `helm1d_locdnn_dgm_adam_unicolloc.py` so that the python script is compatible with bash. The script is set to have its architecture be of constant width throughout the hidden layers. The script is modified to accept variables passed via bash. The modifications are shown below,

```
import sys

#Take Command Line Inputs (Hans)
width = int(sys.argv[1])
depth = int(sys.argv[2])

#Declare layers and activations
layers = []
activations = []
```

```
#Appending First Element
layers.append(1)
activations.append('None')

#Loop to build Architecture
for x in range(0, depth):
    layers.append(width)
    activations.append('tanh')

#Appending Last Element
layers.append(1)
activations.append('linear')
```

Post-Processing of the solution data is performed in python. The script `Error_Analysis.py` which is referred to in the bash script reads the solution files produced by `helm1d_locdnn_dgm_adam_unicolloc.py` and prints the average absolute error, maximum absolute error, the L^2 error as well as the corresponding width and depth of the Neural Network Architecture. The output of `Error_Analysis.py` is then redirected to a text file. The script `Error_Analysis.py` is shown below,

```
#Author: Hans C. Suganda
import matplotlib.pyplot as plt
from numpy import linalg as LA
import numpy as np
import sys

#Command Line Input Name
solname = sys.argv[1]

#Command Line Width Breadth of NN
width = int(sys.argv[2])
depth = int(sys.argv[3])

#Reading from Files
data = np.genfromtxt(solname, comments="variables")

#Average Error
aveError = np.mean(data[:,3])

#Maximum Error
maxError = max(data[:,3])

#L^2 Error
L2 = LA.norm(data[:,3])

print(width, depth, aveError, maxError, L2)
```

The python script

0.4 Test Case & Limited Results

The Error of the Neural Network over its domain is tabulated below,

Width	Depth	Average Error	Maximum Error	L^2
20	1	0.4767139016743369	1.21321010343055	16.43196975831925
20	2	0.36337689474248147	1.07577173547851	11.813225793596358
20	3	0.15119404313853818	0.851842367925335	6.864143199186082
20	4	0.1536894754002597	1.97415648146565	9.622198221171052
40	1	0.4637485332729497	1.26615554426419	16.148805129240056
40	2	0.18779691923494238	1.05299890813605	8.737344682662584
40	3	0.2335196696594362	1.77169708726389	11.722378501336951
40	4	3.223507104515959e-05	0.000155790739117201	0.0011565866332871985
60	1	0.4511783735343545	1.31555873396604	15.91540162087984
60	2	0.25665571306799917	0.876703969082931	10.814118782581973
60	3	0.017735567897361287	0.155393447169199	0.9151063209921008
60	4	0.0001560251953447245	0.000380070972433977	0.005241123681117718
80	1	0.44909653120942183	1.32552402137809	15.888488757909293
80	2	0.005080361087561434	0.0146219797317602	0.19318236993930002
80	3	0.00014694120421892615	0.00056067699706297	0.005469079621267272
80	4	0.00017289489075562206	0.000860454240213748	0.007260651768913759
100	1	0.44487882638948234	1.35427810144497	15.820101171207822
100	2	0.000697345414142119	0.00272186407889197	0.025952533601088467
100	3	1.481419501935395e-05	0.000178296509776743	0.0008104285525297383
100	4	8.730144114653492e-06	9.45060494710326e-05	0.0004246878711267579

0.5 Further Development

Other than just analyzing the error of as the result of the neural network, it might also be helpful to seek for convergece of the weights. An interesting question to ask is given the optimizer and the initial weight conditions, will the weights converge to some unique value? This might be an interesting topic for further discussion.