Industrial Training
on

Neural Network-based Approximation for HPC Applications

SUBMITTED BY

Surya Teja Yadavilli
170905324
46

CSE-B
yst1303@gmail.com
9182823182

Under the Guidance of:

Mr. Dharmendra
Section Manager
University Relation Program Lead
HPE India R&D

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Department of Computer Science & Engineering

August 2020

Certificate

Congratulations!

Certificate of Participation

This certificate is awarded to

Surya Teja Yadavilli

Has been recognized for participating in the HPE In-Semester Project held at MIT Manipal from Feb to Jun 2020.

Domain: HPC – Neural Network

Dharmendra, Section Manager
University Relation Program Lead
HPE India R&D

Date: 19th August 2020

Acknowledgement

My efforts in undertaking this project, would not have been possible without the special help and guidance of some people.
I wish to acknowledge the help provided by Mr. Dharmendra, Section Manager HPE, for his guidance and supervision as well as for providing necessary information regarding the project.
I would like to express my gratitude towards my parents for their kind co-operation and encouragement which motivated me to complete this project. I would like to express my special gratitude and thanks to the people of HPE for giving me such attention and time.
My thanks and appreciation also goes to my team member Meena Sirisha Uppala, who took an active part in developing the project alongside me.

Abstract

High Performance Computing can be used to reduce the time taken to compute calculations of certain applications by a large factor. By making use of large scale parallel structures, HPC helps improve the computational complexity of the system.
This is primarily implemented through Neural Networks. The Neural Networks used in this process must be thoroughly tested with different model sizes and different granules to gain sufficient knowledge of the computed system and to obtain the speedup as compared to traditional methods.

Contents

1 Details of the organisation

The Hewlett Packard Enterprise Company (HPE) is an American information technology company based in San Jose, California. It was formed on November 1, 2015 after a split of the former Hewlett-Packard company. HPE is a business-focused organization with two divisions: Enterprise Group, which works in servers, storage, networking, consulting and support, and Financial Services. The current ceo-president of HPE is Antonio Neri.

2 Information Acquired

2.1 General

Machine Learning can be used to model a non linear relation between the features which are given to it as an input and can predict an output parameter based on a set of "weights" and "biases" as tools, which are "learnt" while training and help form the appropriate connections to "learn" the model. Machine Learning has been shown to be applicable in many fields such as Image Recognition, Speech Recognition, Recommendation Systems,Language Translation and even in Medicine where it is generally used as an abnormality detection tool.

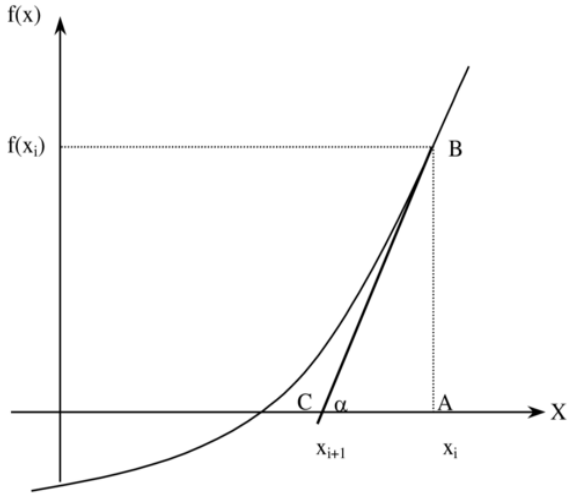
Code Region is the part of an algorithm that can be a specific function or a loop control structure that is defined as the part of an application that is intensive or time consuming. The neural network that we chose to replace the code region with will take the same variables for input and output. We do this replacement to decrease the total time of execution of the program, but keeping in mind that the accuracy of the output is not disturbed [1].

Approximation Methods is a field where Neural Networks can be used to predict a parameter with a trade-off between computational complexity or running time and accuracy. Desirable Neural Networks can be used to model a system with not much loss in accuracy. Scientific Experiments include, the medicine prediction for cancer at Argonne National Lab [2] and Experiments on the Bose-Einstein Condensate in which neural networks have been used to reduce the number of experiments by a factor of 10 [3].

2.2 Newton Raphson Method

Newton Raphson Method is a root finding algorithm to approximate the roots of a quadratic equation through differentiation of a continuous function and initial approximation of a root as shown in Figure 1.The target equation in Newton Raphson Method is f(x)=$ax^2 + bx + c$. The variable a,b,c, are input variables for the equation and x_0 is the initial guess of the solution, x_n is the output variable, which is the final solution. The main principle used in Newton Raphson method is to fit a straight line tangent to the curve. The x-intercept of this line (the value of x which makes y = 0) is taken as the next approximation, $x_n + 1$, to the root. Newton's Method may fail in case there are points of inflection, local maxima or minima around x_0 for the root.

In the Newton Raphson Method, a threshold is used. If the difference between the current solution and the immediately last solution is smaller than a threshold, then this value is the approximation of the root, the algorithm for which is shown in Figure 2 . In this project, predicting roots of an equation using Newton Raphson Method is identified as a code region. An approximation to this using HPC would require 3 inputs which are the coefficients of the quadratic equation, the output would be the predicted root.



$$\tan(\alpha) = \frac{AB}{AC}$$

$$f'(x_i) = \frac{f(x_i)}{x_i - x_{i+1}}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Figure 1. Derivation of the Newton Raphson Method Adapted [reprinted] from "Solving Transcending Equations," by Dr Ali Kashmar, Jan 2018, Chapter 1 from "Numerical Analysis"

The algorithm for the Newton Raphson Method is given below[3]:

Algorithm: Newton Raphson Method
Require : Function f; Truncate error E; Initial assumption x0; default number of iterations N; Ensure: Final solution F

1: f(x) = 0;
2: x=x0;
3: if abs($x_i - x_{i-1} < E$) and $i < N$ then
4: for each assumption xi do
5: if(xi)is differentiable then
6: $x_{i+1} = x_i - f(x_i)/\tilde{f}(x_i)$
7: i=i+ 1
8: end if
9: end for
10: end if
11: return x_n

In this project, High Performance Computing has been used to explore alternate execution of the code region using Neural Networks. The specific application is Newton Raphson Method. Different models of Neural Networks with different number of layers

to explore the prediction accuracy of the root of a quadratic equation.

The Keras API deep learning framework has been used in this study. Python has been used as the main programming language. Numpy package has been used to make the computation linear algebra easier. The specific machine learning libraries including the activation functions, specific optimisations like Root Mean Square Propagation(RMSprop) and the loss functions have been imported from the scikit-learn library. Since a supervised learning algorithm is required to be implemented, the different cross validation methods have been used. The matplotlib library has been used for visualisation of the performance of the model over diferent 'epochs' i.e the number of iterations.

Given the three coefficients (a,b,c) of a quadratic equation, f(x)= ax^2 + bx + c, q root x_n for the equation has to be predicted.

An equal distribution of roots and their coefficients has been extracted in the range [-1,1]. 105472 such samples have been extracted. The numpy.random.uniform function has been used for this purpose (Figure 2).

```
def generate(105472):
    y = np.random.uniform(-1, 1, (n_samples,2))
    y.sort(axis=1)
    X = np.array([np.poly(_) for _ in y])
```

Figure 2. Generating the roots for 105472 samples

The samples generated thus have been used for training,validation and testing in the ratio of 80% : 20% : 20%.

Initially, a 4 layered Neural Network Model has been used of the number of neurons in 3x5x3x1. The input layer has 3 neurons for each of the coefficents of the quadratic equation. The hidden layer 1 has 5 neurons and the hidden layer 2 has 3 neurons. Output Layer has 1 neuron (Predicted root). Linear and Rectified Linear Unit(reLu) activation functions have been used with the linear function being used for the input and output layer(Figure 3). reLu function has been used to generalise a non linear relationship to the model(Figure 4). Since the number of training examples is huge, the sigmoid and the tanh activation functions have not been used because they suffer from the vanishing gradient problem. In the vanishing gradient problem, for a small change in the training variable 'X' causes very less change in the prediction variable 'Y'. The relu activation function provides for backpropagation and is computationally efficient. The algorithm is given below:

Relu activation function(X):
A(X) = max(0,x)

```
model = Sequential()
model.add(Dense(3, input_dim=3,kernel_initializer='normal', activation='linear'))
model.add(Dense(5, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Figure 3. 3x5x3x1 model of the Neural Network

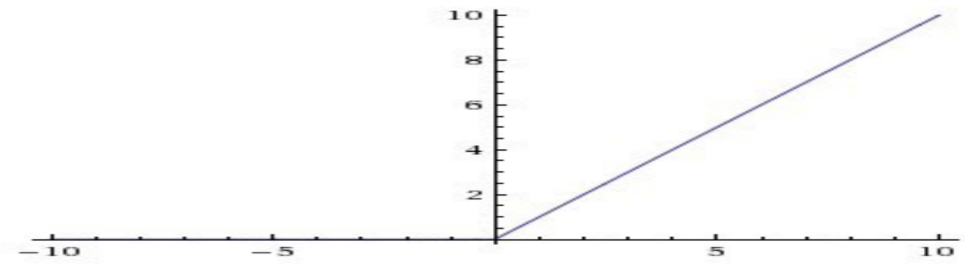


Figure 4.An example of the Relu Activation function

Gradient Descent with Momentum-Backpropagation approach, which uses the concept of the exponentially weighted average of the gradients, has been used to evaluate the model(Figure 4). The algorithm for this is given below.

Algorithm: Root Mean Square Prop (RMSprop) -

VdW = β * VdW + (1 – β) * dW2

Vdb = β * Vdb + (1 – β) * db2
W = W – learning rate *dW sqrt(VdW)
b = b – learning rate * db sqrt(Vdb)

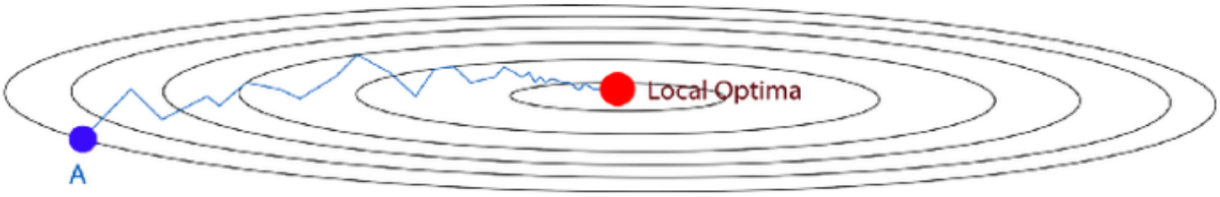


Figure 4. RMSprop, Shweta,Shaw. "RMSprop : A Better Way to Optimize Your Model", medium, "medium.com/@shwetaka1988/rmsprop-a-better-way-to-optimize-your-model-bc4eaca33090"

As shown in the figure, the 'bias' b is responsible for the vertical oscillations whereas 'weight' is responsible for movement horizontally . If bias is slowed down, then we can still move fast towards the local optima.

The learning rate has been set to 0.01, with Batch Size = 32. 500 iterations of the same have been used. This approach has been used for the reason that iterating one example at a time, which is the case in Stochastic Gradient Descent and would a lot of computational time, while Batch Gradient descent may generalise the average value of the dataset. Hence Mini Batch Gradient has been used. The Loss function used is Mean Absolute Error as it is the standard loss function used in Regression (Figure 5).

```
opt = RMSprop(lr=0.001, rho=0.9)
model.compile(loss='mean_absolute_error',
              optimizer='RMSprop',
              metrics=['mae'])
```

```
BATCH_SIZE = 32
history=model.fit(X_train,
          Y_train,
          batch_size=BATCH_SIZE,
          epochs=500,
          verbose=1,
          validation_data=(X_test,Y_test))
```

Figure 5. Evaluation of the Neural Network

A graph has been plotted for the loss function(Figure 6) and it is observed that the training loss decreases sharply initially and gradually thereafter. The accuracy for this model was computed to be 83%.

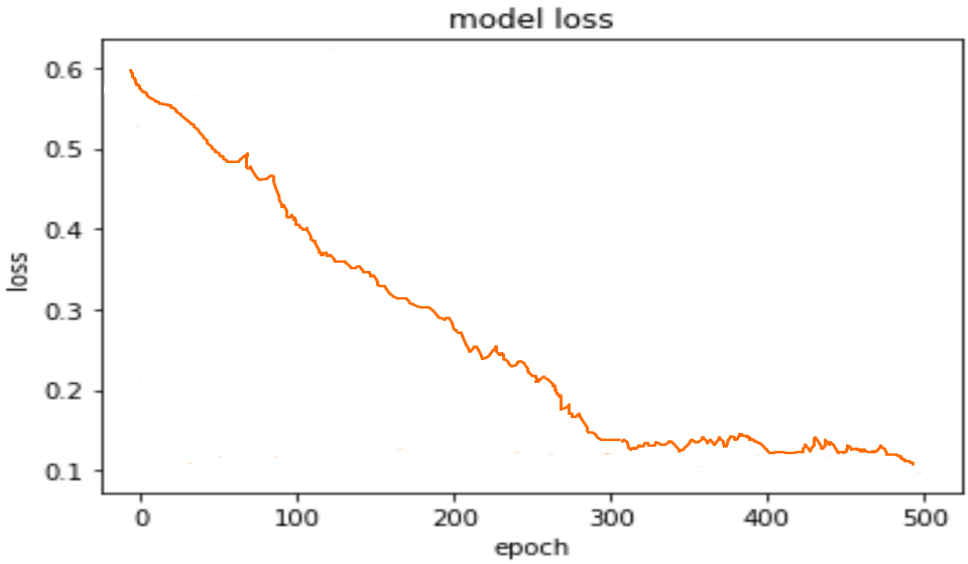


Figure 6. Plot of the value of Loss function with iterations

The model has been extended for multiple configuration of layers 3x3x1,3x5x1,3x3x2x1,3x8x5x1, 3x11x8x5x1 to measure the predictive accuracy when the number of layers and the neurons are varied.Similar study has been made for models of different number of neurons in the hidden layers and varying number of layers, by plotting the value of loss function over iterations(Table 1). In the table we see that the accuracy of the neural network with a given configuration of layers improves with the increase in the number of neurons. The accuracy of the system also improves if the the number of hidden layers is increased(4-5 layers), however the training time required for the process also increases.

Table 1. gives the final value of the loss function and the predicted accuracy of each model

Topology	Loss	Accuracy	training time(seconds)
3x3x1	0.063	50%	3603
3x5x1	0.059	57%	4001
3x3x2x1	0.042	64%	5164
3x5x3x1	0.037	79%	5298
3x8x5x1	0.028	77%	7751
3x11x8x5x1	0.031	83%	7328

2.3 LJ Potential in LAMMPS

Leonard-James Potential (LJ Potential) is a model that approximates potential energy of a system of particles. The potential is the summation of the forces of neighbours of each particle. The algorithm is give below:[3]

Algorithm : Lammps Potential
Require: Computing range start,end; cutoff distance r_{cut}; location array atom.
Ensure:Force array F.
1: for i ranges from start to end do
2: for each neighbors j of i do
3: $d_{ij}^2 \leftarrow abs(atomLocation(i) - atomLocation(j))^2$;
4: if $d_{ij}^2 < r^2$ cut then
5: calculate Forceij;
6: $TotalForce_i \leftarrow TotalForce_i + Forceij$;
7: end if
8: end for
9: $F(i) \leftarrow TotalForce_i$; 10: end for
11: return F

The dataset for the Lammps model has been obtained from the code below(in.lj.5)
:
3d Lennard-Jones melt

```
variable        x index 1
variable        y index 1
variable        z index 1
variable        t index 100
variable        xx equal 20*$x
variable        yy equal 20*$y
variable        zz equal 20*$z
units           lj
atom_style      atomic
lattice         fcc 0.8442
region          box block 0 xxyyzz
create_box      1 box
create_atoms    1 box
mass            1 1.0
velocity        all create 1.44 87287 loop geom
pair_style      lj/cut 5.0
pair_coeff      1 1 1.0 1.0
neighbor        0.3 bin
neigh_modify    delay 0 every 20 check no
fix             1 all nve
thermo          100
run             $t
```

Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is a dynamic prgram from Sandia National Laboratories. OpenMPI is used in Lammps for parallel computation. In this task, in.lj.5 is the input problem that runs on the lammps code and executes the above algorithm to find the LJ Potential of a system (Figure 7). The figure shows a sample plot of the LJ potential vs distance (conditions 3 and 4) in the algorithm.

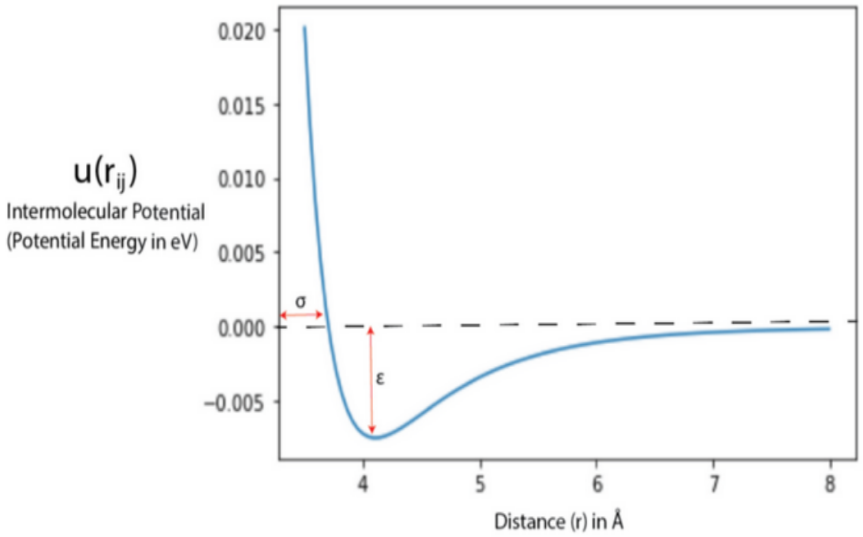


Figure 7. LJ Potential calculated for strength of the interaction=0.0103 and distance at zero intermolecular potential= 3.3 , "Zammataro Luca","The Leonnard James Potential", towardsdatascience.com/the-lennard-jones-potential-35b2bae9446c

For the calculation of LJ potential with a neural network initially a neural network with the 1x3x1 has been used. Absolute Error is employed as the cost function. The plot of the training and validation loss is plotted(Figure 8). Gradient Descent has been used and trained for 100 iterations.

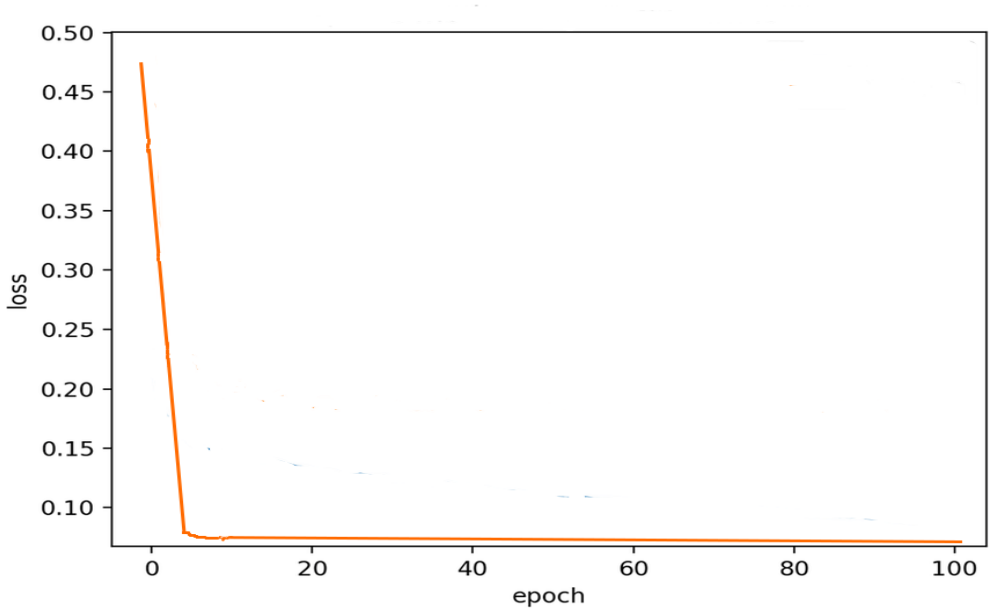


Figure 8. Plot of the value of Loss function with iterations

Table 2. gives the final value of the loss function and the predicted accuracy of each model

Topology	Loss	training time(seconds)
1x3x1	0.23	1800
1x5x1	0.29	1844
1x8x1	0.142	2103
1x3x2x1	0.137	3320
1x3x5x1	0.0082	3310
1x5x8x1	0.0044	6280

The model has been extended for multiple configuration of layers 1x3x1,1x5x1,1x8x1,1x3x2x1, 1x5x8x1 to measure the predictive accuracy when the number of layers and the neurons are varied.Similar study has been made for models of different number of neurons in the hidden layers and varying number of layers, by plotting the value of loss function over iterations(Table 2).

In the table we see that the accuracy of the neural network with a given configuration of layers improves with the increase in the number of neurons. The accuracy of the system also improves if the the number of hidden layers is increased to 4, however the training time required for the process also increases.

The model with absolute error as 0.0044 qualifies as the better model and has predictive accuracy of about 79%.

3 Conclusion

The field of Machine Learning and Deep Learning is improving as Neural Networks can approximate and compute the required parameters in an efficient way without inducing much deviation in the predicted results. In this study, a code region has been identified in the calculation of roots of a quadratic equation and LJ potential. The code region can be replaced with a Neural Network which predicts accurately the required parameter. With the help of large parallel computing resources, the neural network can be further optimised to achieve a speedup.

References

[1] Zhaoyong Liu and Yiting Zhang. Research on matrix factorization algorithm based on multiple kernel learning in collaborative filtering. 01 2018.

[2] Evan Racah, Christopher Beckham, Tegan Maharaj, Mr Prabhat, and Christopher Pal. Semi-supervised detection of extreme weather events in large climate datasets. 12 2016.

[3] P. Wigley, P. Everitt, Anton Hengel, John Bastian, M. Sooriyabandara, Gordon McDonald, Kyle Hardman, C. Quinlivan, Manju Perumbil, Carlos claiton Noschang kuhn, I. Petersen, Andre Luiten, J. Hope, N. Robins, and Michael Hush. Fast machine-learning online optimization of ultra-cold-atom experiments. *Scientific Reports*, 6, 07 2015.