SPARSE DEEP NEURAL NETWORK GRAPH CHALLENGE

by:
Satish kumar Oraon(12041320)
Md Arsad (12040880)
Raunak Kumar (12041190)

INTRODUCTION

- Deep neural network are the heart of ML and AI.
- We cannot imagine as it is helpful in playing with big data sets to get wonderful results.
- Previous research has shown that sparse Deep Neural Networks outperform dense DNNs in terms of memory optimization.
- Bigger the size of dataset, we get better and efficient result.
- Today's computer hardware and architecture make it difficult to play with large data sets.

MATH CONVENTION

$$Y_{l+1} = h(Y_l W_l + b_l)$$

- Y_0 is (no of inputs) , x (no of features); each row is a feature vector
- No of features = no of neurons/layer (constant for all layers)
- W_l is (no of neurons), x (# neurons); $W_l(i,j) > 0$ therefore , a connection from i to j

- `bl` is a bias row vector applied to each input
- $h(x)$ is the ReLU function with a max cutoff

  When $0 < x < 32$, x is unchanged

  $x < 0$, x is changed to 0

  And last $x > 32$, x is changed to 32

Sparse DNN Challenge uses standard graph community terminology,Standard AI definitions is used here.

Challenges in problem Solving

- Large size of neural network.
- For traversal of neural network , requires massive parallelism
- Due to limited size of GPU memory, large size neural network do not fit into the GPU memory

APPROACH

- **First approach**: every image is multiplied by different layer according to

$$Y_{l+1} = h(Y_l W_l + b_l)$$

- We have to call the kernel many times. For every image we have to call the kernel as 120 times as the number of layers.
- It takes more execution time as well as large storage

- **second approach:**,we have copied the 60000 image into kernel and then multiplied parallely with neuron layer.
- In y-direction 1875 block and there are 32 blocks in the x direction for 1024 neuron image
- This approach is more parallel than the previous one as it multiplied 60000 image parallely with neuron layer
- Requires more space in GPU as we have to pass all the 60000 image to the kernel for every layer.

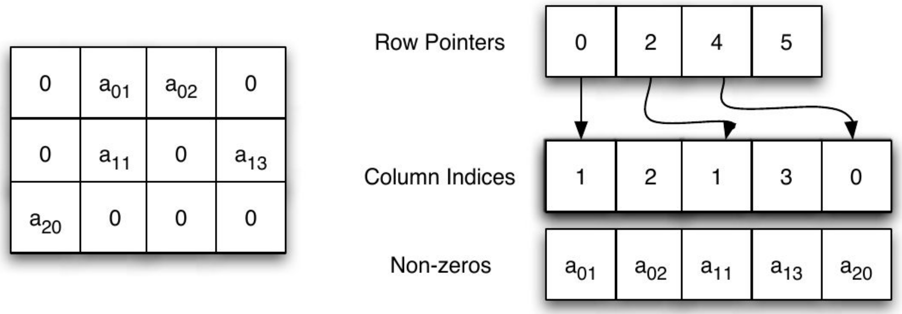
```

__global__ void Kernel(float * M,float* N,float * P){
    int Row=blockIdx.y*blockDim.y+threadIdx.y;
    int Col=blockIdx.x*blockDim.x+threadIdx.x;

    if(Row<1 && Col < 1024){
        float product=0;
        for(int k=0;k<1024;k++)
            product+=M[Row*1024+k]*N[k*1024+Col];
        product = product - 0.3;
        if((product)<0)
        {
            product = 0;
        }
        if((product) >32)
        {
            product = 32;
        }
        P[Row*1024+Col]=product;
    }
    __syncthreads();
}

```

- **3rd approach:** For efficient memory storage we have converted each layer into CSR format.
- CSR Representation



ALGORITHM

```
__global__ void Kernel(float * IMG,int* Rptr,int* Cptr,float * value,float* D_P){
      int id=threadIdx.x;
      int ind=blockIdx.x;
        float prdt=0;
        for(int i=Rptr[id]; i<Rptr[id+1]; i++){
            prdt+=IMG[ind*1024+Cptr[i]]*value[i];
        }
        prdt-=0.3;
        if(prdt<0)prdt=0;
        if(prdt>32)prdt=32;
        D_P[ind*1024+id]=prdt;
        __syncthreads();

}
```

Inference time

Layer/Time	Inference Time	Verification Time	Total TIme
120	33.832 sec	0.006846 sec	33.838846 sec
480	136.911 sec	0.006807 sec	136.917807 sec
1920	576.231 sec	0.006856 sec	576.237856 sec

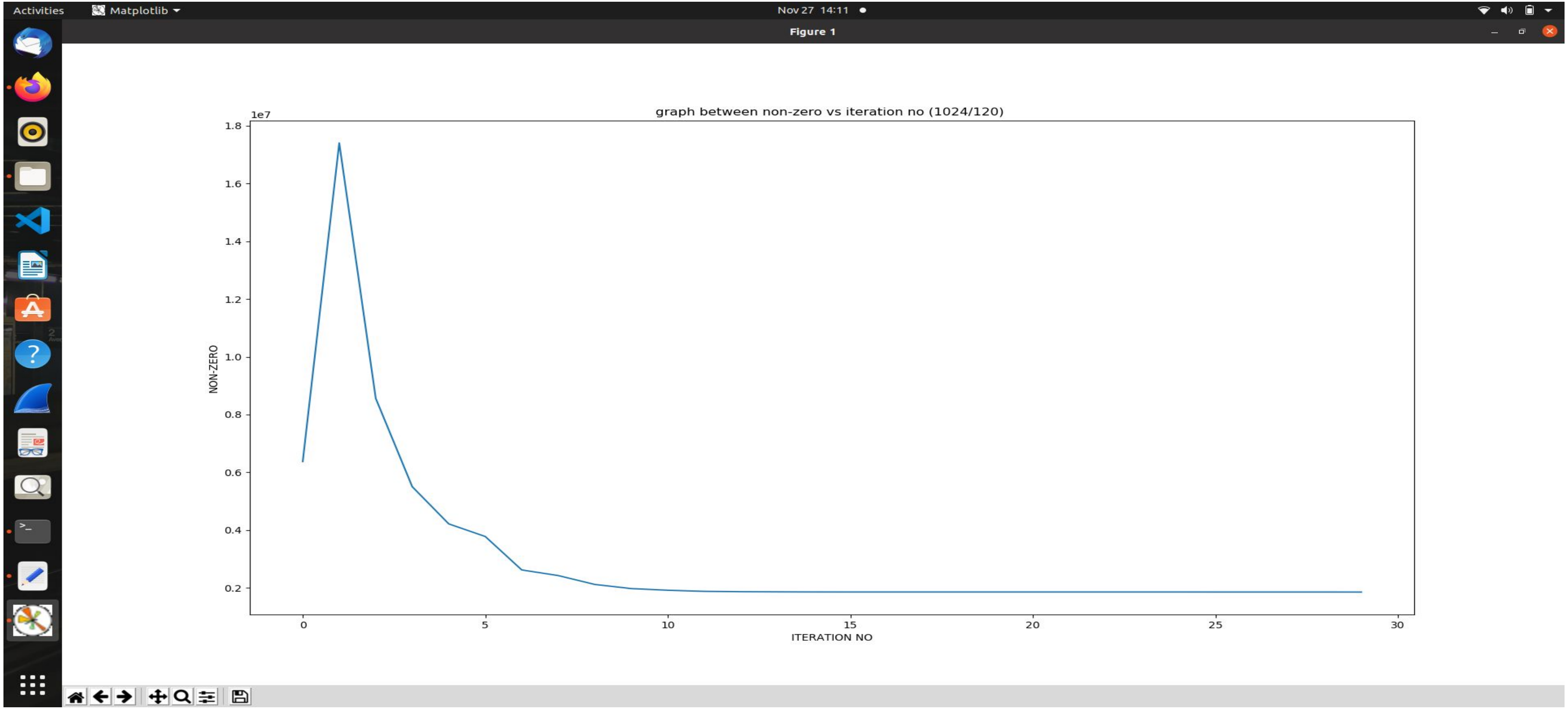
Inference rate

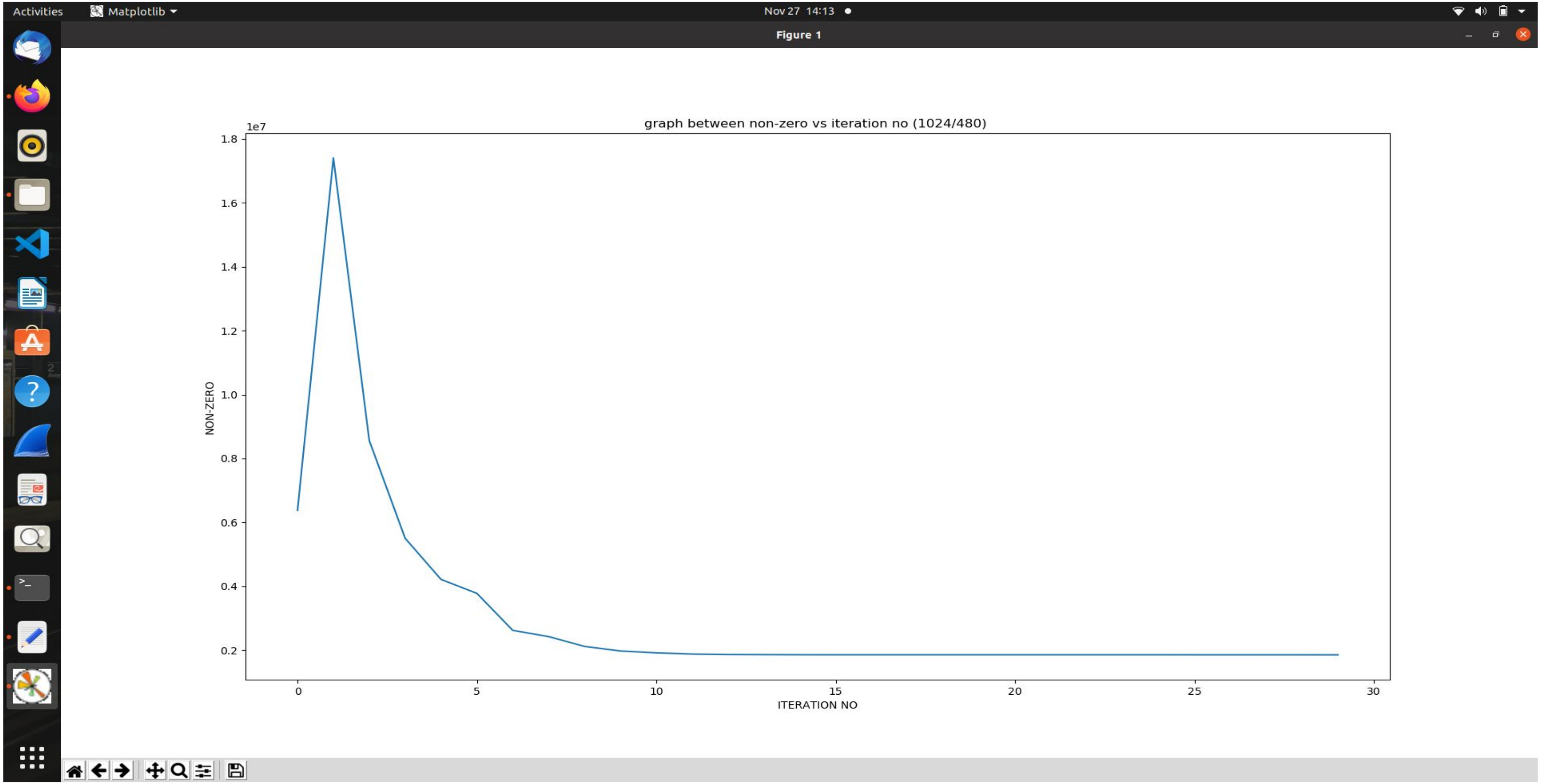
Inference Time result

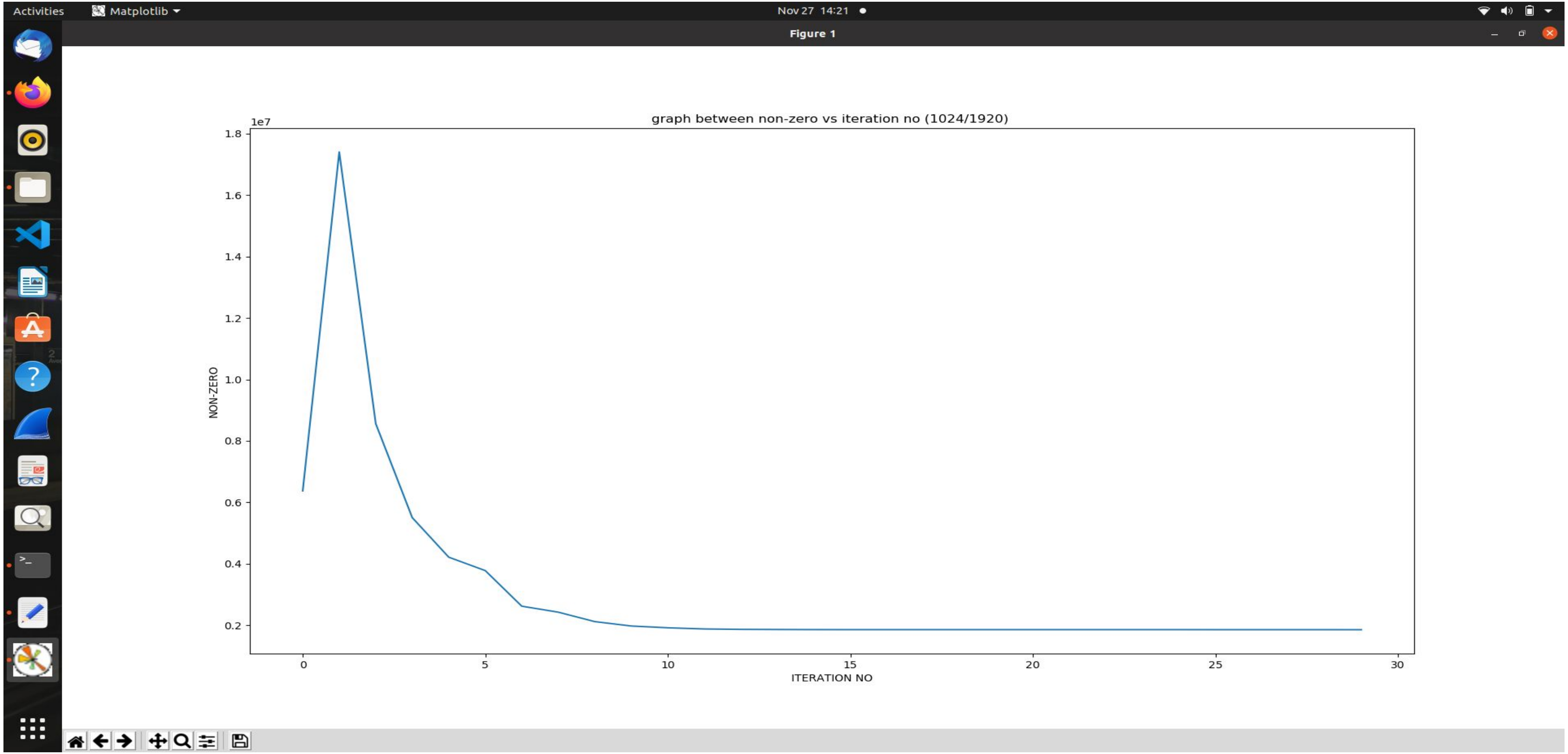
Number of Neurons per layer	layer	Time
1024	120	33.838846 s
1024	480	136.917807
1024	1920	576.237856
4096	120	480 approx seconds
4096	480	-

Rate = 32*Number of image *number_of_layers*number_neuron/inference time

Number of Neurons per layer	layer	Edged/second
1024	120	6.97 e^9
1024	480	6.89 e^9
1024	1920	6.5 e^9
4096	120	1.9 e^9
4096	480	-


Activities
Matplotlib
Nov 27 14:11
Figure 1
graph between non-zero vs iteration no (1024/120)
1e7
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
NON-ZERO
0
5
10
15
20
25
30
ITERATION NO


Activities
Matplotlib
Nov 27 14:13
Figure 1
graph between non-zero vs iteration no (1024/480)
1e7
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
NON-ZERO
0
5
10
15
20
25
30
ITERATION NO



Activities
Matplotlib
Nov 27 14:21
Figure 1
graph between non-zero vs iteration no (1024/1920)
1e7
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
NON-ZERO
0
5
10
15
20
25
30
ITERATION NO

Conclusion

- 60000*65536 could not be done due to the resources constraints in the google colab
- The problem can be solved more efficiently using multiple GPUs.
- Deep neural networks have significantly advanced the state-of-the-art across a number of domains, revolutionising the field of machine learning.
- However, the hardware required to implement deep neural network topologies is becoming increasingly taxed by their size.
- The sparsification of these neural networks has been the subject of extensive research over the past ten years in an effort to reduce storage and runtime costs.