Bilkent University
Department of Computer Science

CS442 - Distributed Systems and Algorithms

Final Project Report

Distributed Feed-Forward Neural Network
SerdarNet

Group Members

Aytekin İsmail - 22003988

Berkan Şahin - 22003211

Serdar Özata - 22003113

Tuna Okçu - 22002940

Introduction

We have implemented a feed-forward neural network that will run on multiple machines using MPI and the C++ programming language. The network will include ReLu and dropout layers, back-propagation, softmax, and cross-entropy loss.

We utilize low-powered machines to train a neural network using distributed and parallel processing techniques. Our approach should scale well across multiple CPU cores and multiple networked machines since the individual components of the neural network are parallelizable.

We have tested the implementation with a simple binary classification task using the airplane satisfaction dataset [1].

Project Design

MPI functions are used to communicate between processes that are running on different processors.

- **MPI_Init(int* argc, char*** arv)**
  The initialization of the MPI system takes the main method's parameters as pointers [2].
- **MPI_Finalize()**
  It closes the MPI execution environment [2].

These two method calls encapsulate all of our operations. This is required to properly execute an MPI program.

- **MPI_Comm_rank(MPI_Comm comm, int* rank)**
  We used this method with the value MPI_COMM_WORLD, which contains all processes. This method will return the rank of the process assigned by MPI [2]. All processes will have a unique rank as a result of this method.

- ## Layers

All the provided layers are descendants of the Layer class where the forward() and backward() functions are defined. We have the following layer structure in the system:

1. Fully Connected Layer

   This class will implement a fully connected layer in a neural network with overwriting forward() and backward() functions it inherited from the Layer class.
   The critical part of this class is that it implements MPI_Bcast and MPI_Reduce functions.

- **int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)**
  This function will enable the process whose rank is root to broadcast a message to all processes in the communicator specified with comm [2]. We used MPI_COMM_WORLD again to reach all the processes in the MPI system. We specified the rank as zero, which means the process with rank zero will be the source of broadcasting.

- **int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)**
  This function will perform an operation specified with op, combine the values from all processes, and distribute the result to all processes [2]. We used the MPI_IN_PLACE keyword with this function to denote that the send buffer and the receive buffer are the same buffer because we will take values from all processes to perform the operation and distribute the result back to all these processes. We used this function to sum up the values of the weight gradient and bias gradient.

2. Activation Layer
   Aside from the inherited functions from the Layer class, it only has an alloc output function, which assigns the output to a newly created matrix.

2.1 Dropout
This class implements the Dropout functionality where there is a probability that an input will not be considered while computing the output. We set this drop_out rate as 0.3, and the result and if only the random number generated with the rand() function returns a value bigger than the drop_out rate (in this case 0.3), then the input will be reflected in the output. Otherwise, the output will be zero.

2.2 ReLu
This class provides rectified linear unit activation functionality. During a forward pass, it will take the maximum of zero and the input value at that index to put it to the output at that index, essentially only passing nonnegative values from the input to the output.
This layer has a vanishing gradient problem where neurons will be stuck in an inactivated state. Therefore, we also provided LReLu.

2.3 LReLu (Leaky ReLu)

Instead of zero, we will use an alpha value (0.01) in the forward pass, which will take the maximum input value multiplied by the alpha value and the input value for the output at that index.

- ## Loss Functions

There is a Loss class that implements the checkDims() function that checks the dimensions of the label and the output, and if they do not match, it will output an error message. get_error_prime() and eval() functions are declared in the Loss class as virtual functions.

1. SoftmaxCEntropyClassification Class

   SoftmaxCEntropyClassification provides the cross entropy loss for eval() and get_error_prime() functions. In get_error_prime(), it checks the dimension of the output, then it calculates the error gradient with respect to the output of the network and returns the error matrix.

   The eval() function is used to evaluate the network output for accuracy. It uses the softmax function to calculate the probabilities and compare network output with labels to find the number of true positives.

   - **int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,MPI_Op op, int root, MPI_Comm comm)**
     This function will perform an operation specified with op for values specified in sendbuf in all processes and outputs it with recvbuf to process with rank root [2]. This function is used to sum the number of true positives from each process to get a global true positive number and send this global value back to the processes with rank root; in the case of our project, the process with rank 0. The total number of predictions by all these processes is also summed, and this global sum is sent to the process with rank zero and is used to calculate the accuracy of the network output as a percentage. The difference between MPI_Reduce and MPI_Allreduce is that in MPI_Allreduce, the result is broadcasted to all processes in the communicator, while MPI_Reduce will send the result to the process specified with its rank in the parameter.

2. LogLoss Class
   This class uses the sigmoid function instead of softmax cross entropy loss calculation and overrides eval() and get_error_prime() functions defined in the Loss class.

We use the sigmoid function defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ and we define $\varepsilon = 10^{-10}$ to prevent log(0) case.

In **eval()**, similar to the cross-entropy class, we have used **MPI_Reduce.**

- **int MPI_Comm_size(MPI_Comm comm, int *size)**
  This function is used to determine the size of the group associated with the communicator, which is specified as comm [2]. We used MPI_COMM_WORLD to refer to all processes in the MPI; therefore, this function returns the total number of processes. We used the result of this function to compare the output of each result to get the accuracy.

In **get_error_prime()**, we must take the derivative of both sigmoid and log loss. Instead of doing separate the derivation, we have preprocessed it to make a simple calculation as follows:

Because $\sigma(x) = \frac{1}{1+e^{-x}}$ the derivative of the sigmoid function is
$\sigma'(x) = \sigma(x)(1 - \sigma(x))$ and $L = (y)\, ln(\sigma(x)) + (1 - y)\,(1 - \sigma(x))$.

$$\frac{dL}{d\sigma(x)} * \frac{d\sigma(x)}{dx} = \frac{dL}{dx} = y - \sigma(x)$$

We directly used the value $y - \sigma(x)$ in the code for get_error_prime().

- ## Neural Network

The **neural network constructor** takes an input matrix pointer and a loss object pointer.
In the main.cpp file, we read two CSV files, “data/airline_passenger_satisfaction_x.csv,” whose data will be the input, and “data/airline_passenger_satisfaction_y.csv,” whose data will be the labels. We create a softmax cross-entropy classification object by processing labels. This loss object will be used to create a neural network with the input matrix.

We created seven layers (3 fully connected, two dropouts, and 2 ReLu layers) using **get_final_output()**. At first, because there are no layers before adding these newly created layers, this function will return the input matrix.

We store the layers as a vector, also called **layers,** and the add_layer function, therefore, calls push_back() that gets the layer it gets as a parameter and appends it to the **layer's** vector. We added these three layers to the neural network using the **add_layer()** function.

After adding seven layers, we trained this model with the **train()** function, which takes the epoch number, which is the number of times a forward pass needs to be done through all layers. After performing forward passes, an error matrix will be created and used in back-propagation to update weights.

The **test** function is also defined in the neural network class, which performs forward passes and then evaluates the final output.

Results

We used an airplane satisfaction dataset to verify the results we got from the neural network for a simple classification problem.

When the system is run with four, two processes and one process for tracking the average time for epoch, loss, and accuracy percentage, the results are the following:

Number of processes (n)	Average time for an epoch	Loss	Accuracy
4	0.030617	39,566.48	%88.8
2	0.054434	39,892.01	%88.8
1	0.105814	39,878.11	%88.8

Table 1: Number of processes and their effect on average time, loss, and accuracy

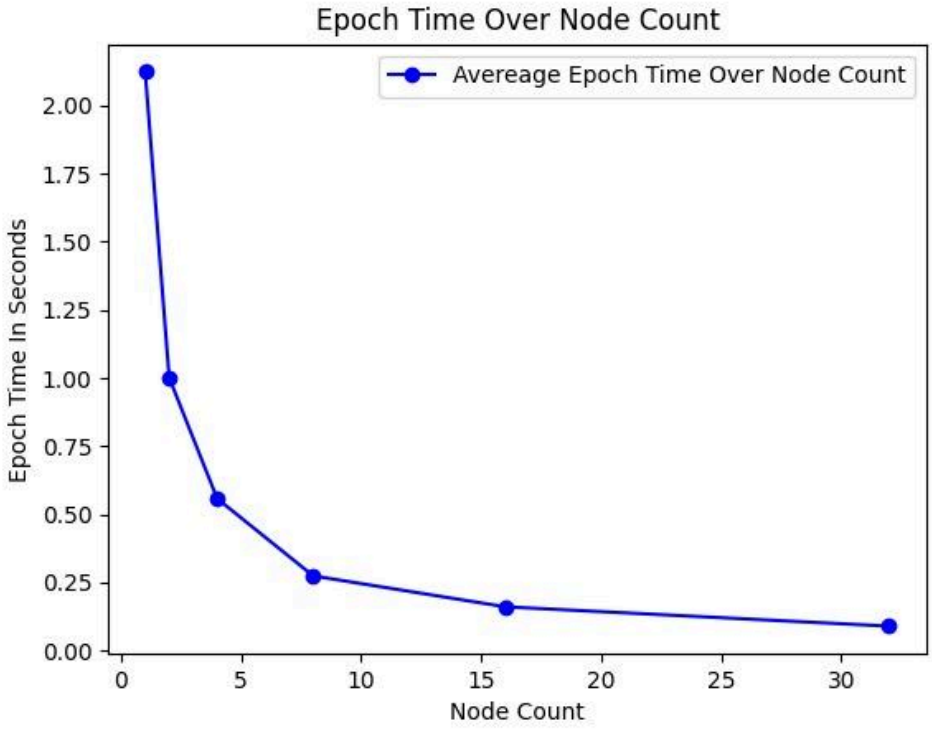

Figure 1: Average Epoch Time vs. Node Count

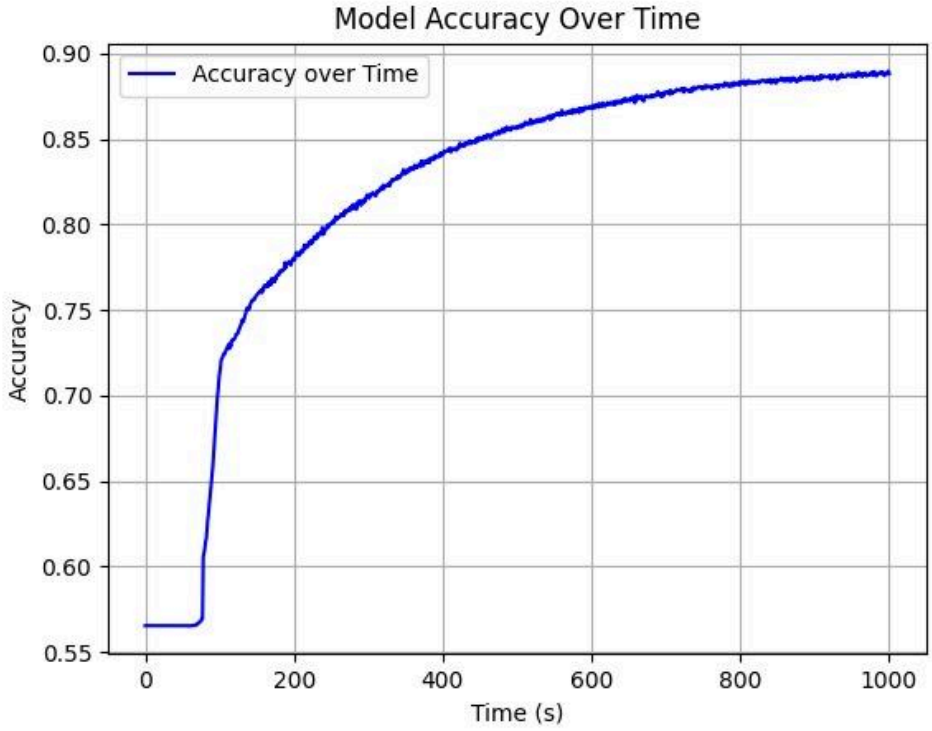

Figure 2: Accuracy vs. Time

Team Member Contributions

Serdar has implemented the neural network, layers, and loss functions using MPI functions.
Berkan has configured the I/O to take the airplane satisfaction dataset into the project to test the neural network.
Aytekin has written the final project report.
Tuna has generated the plots of the neural network results.

The code files that each member has developed can be tracked better on the project GitHub page, which is public https://github.com/serdar-ozata/distNet :

References

[1]“Airline Passenger Satisfaction,” www.kaggle.com. https://www.kaggle.com/datasets/mysarahmadbhat/airline-passenger-satisfaction

[2]“Index of /static/docs/latest/www3,” Mpich.org, 2024. https://www.mpich.org/static/docs/latest/www3/