Gebze Technical University
Department of Computer Engineering

GRADUATION PROJECT - II

INTEGER LINEAR PROGRAMMING

Student:
Ahmet Melih YANALAK
Advisor:
Zafeirakis ZAFEIRAKOPOULOS

12-09-2021

This study was accepted as an Undergraduate Graduation Project in the Department of Computer Engineering by the following jury on ../..../20..

Graduation Project Jury

Supervisor Name	
University	
Faculty	

Jury Name	
University	
Faculty	

Jury Name	
University	
Faculty	

Preface

I would like to thanks to my advisor Zafeirakis Zafeirakopoulos and Gebze Technical University for the support on this project.I also thanks to my family and my friends for support I felt everyday.

September, 2021 Ahmet Melih Yanalak

Contents

Summary

There are many different algorithms to solve the Integer Linear Programming problems and each of them has different behaviour on different examples.One algorithm may be the fastest when solving a problem with small numbers, but slowest when it comes to an example with big numbers.The decision should be done according to the values of the input.

As a Graduation Project in Gebze Technical University, a study was conducted in order to decide the optimal method to use when solving an integer linear programming example by using a deep learning model.

1 Introduction

Integer Linear Programming Examples are quite common and tried to be solved by many people.However it is not the best way to solve all the problems by one algorithm.

Optimization is one of the most import concepts in Computer Science and it helps us to save time and memory.Deciding on which algorithm to solve the ILP problem will make the problem easier and it will save time.

In this report, generating a large data set that is consisting of ILP Problems and training a model to predict optimum solution will be explained.

2 Integer Linear Programming

2.1 Definition

$$\max z = 8x_1 + 11x_2 + 6x_3 + 4x_4$$

$$\text{s.t.}$$

$$\begin{aligned} 5x_1 + 7x_2 + 0x_3 + 3x_4 &\leq 14 \\ 8x_1 + 0x_2 + 4x_3 + 4x_4 &\leq 12 \\ 2x_1 + 10x_2 + 6x_3 + 4x_4 &\leq 15 \end{aligned}$$

Figure 2.1: Integer Linear Programming Example

An integer programming problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. In many settings the term refers to integer linear programming (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.

In the example above, the aim is to maximize the function **8x1 + 11x2 + 6x3 + 4x4**.While maximizing the function, constraints needs to be considered.In addition to these constraints, values of the variables (x1,x2,x3,x4) must take an integer value.

2.2 Flow of the Project

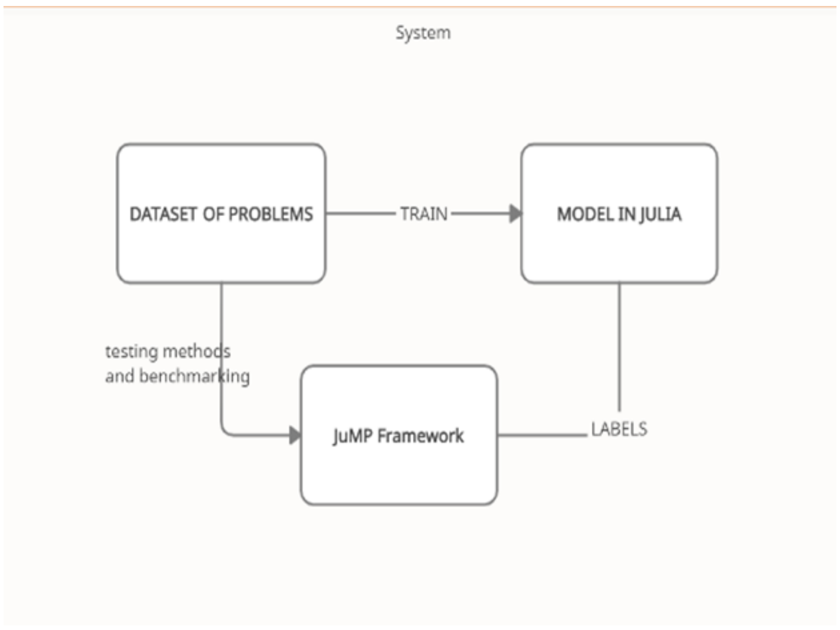


Figure 2.2: Flow of the project

Firstly, data set should be generated by using JuMP in Julia.Then model will be created in Knet Framework of Julia and the data will be passed to this model in order to train and get the predictions.

3 JuMP in Julia

Figure 3.1: JuMP

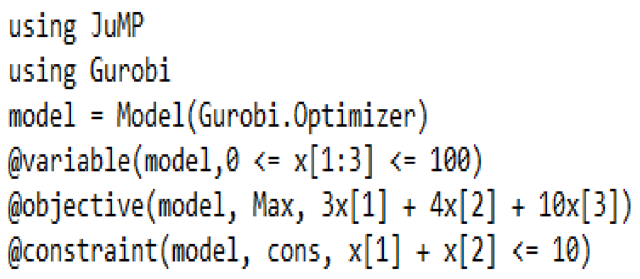
JuMP is a domain-specific modeling language for mathematical optimization embedded in Julia. It currently supports a number of open-source and commercial solvers for a variety of problem classes, including linear, mixed-integer, second-order conic, semidefinite, and nonlinear programming.

In order to optimize the Integer Linear Programming Problem and get the solving time to train,JuMP is used in this project.

There are plenty of optimizers that compatible with JuMP and each of them is useful to solve different types of problem.In order to work with the Integer Linear Programming Examples, **Gurobi Optimizer** is used in JuMP.

3.1 Creating an ILP Model

In order to create model of a problem in Julia, first of all JuMP and Gurobi optimizer are needed to be installed.Then, a model can be created by defining optimizer,objective function and constraints.



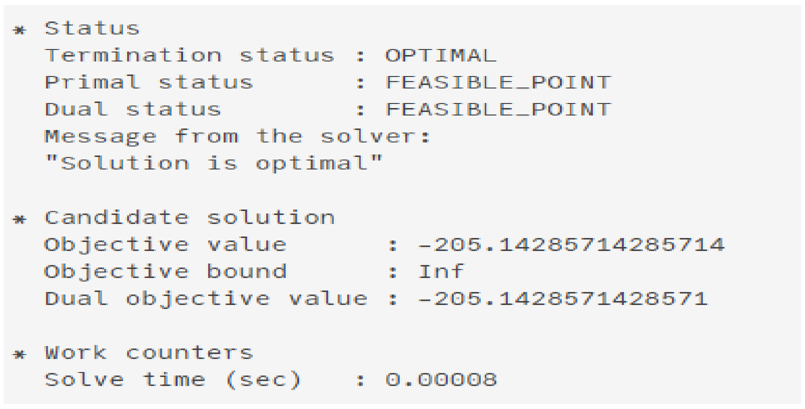
```
using JuMP
using Gurobi
model = Model(Gurobi.Optimizer)
@variable(model,0 <= x[1:3] <= 100)
@objective(model, Max, 3x[1] + 4x[2] + 10x[3])
@constraint(model, cons, x[1] + x[2] <= 10)
```

Figure 3.2: Creating a model in Julia

3.2 Optimizing

After creating the model, the next step is to optimize the model and get the solving time to use as a label.

optimize!(model)
solve_time(model)
solution_summary(model)



```
* Status
  Termination status : OPTIMAL
  Primal status      : FEASIBLE_POINT
  Dual status        : FEASIBLE_POINT
  Message from the solver:
  "Solution is optimal"

* Candidate solution
  Objective value      : -205.14285714285714
  Objective bound      : Inf
  Dual objective value : -205.1428571428571

* Work counters
  Solve time (sec)   : 0.00008
```

Figure 3.3: Output of the solution

4 Preparing the Data Set

Before creating the data set, format of that data should be specified.Since the data will be passed to a neural network in order to make a prediction from a given input, it will be wise choice to represent a ILP problem in matrix form.

4.1 Matrix Representation

Consider Example:

maximize:

3x1 + 4x2 - 2x3

subject to:

x1 + 0.5x2 - 5x3 $\leq$ 2
2x1 - x2 + 3x3 $\leq$ 3
x1, x2, x3 $\geq$ 0

In order to get rid of inequalities,slack variables are used as additional variables.

Add slacks (using x's for slack variables):

x1 + 0.5x2 - 5x3 + x4 = 2
2x1 - x2 + 3x3 + x5 = 3.

Cast constraints into matrix notation:

$$\begin{bmatrix} 1 & 0.5 & -5 & 1 & 0 \\ 2 & -1 & 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

Figure 4.1: Matrix representation of Example

4.2 Generating the Data Set in Julia

In order to generate a data set of ILP Problems in Matrix format in Julia firstly a fixed sized format is decided:

- **Each sample will contain 5 variables in objective function: x1,x2,x3,x4,x5**

- **There will be 3 constraints added to model:**

  1.First constraint will contain all the 5 variables
  2.Second constraint will contain first 4 variables
  3.Third constraint will contain first 3 variables.

- **All the values of the coefficients are randomly generated in range 0-100**

- **Right Hand Side values of the constraints are randomly generated in range 0-1000**

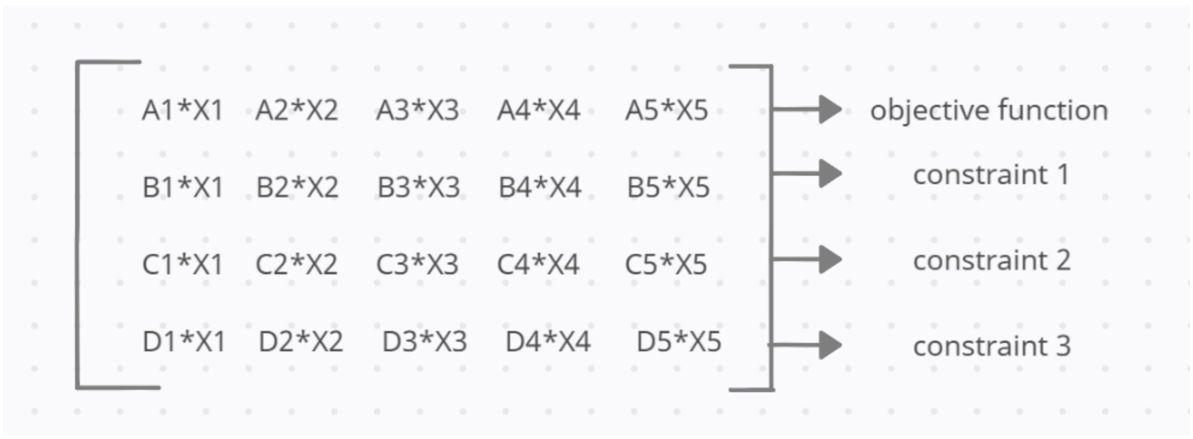

A1*X1 A2*X2 A3*X3 A4*X4 A5*X5
B1*X1 B2*X2 B3*X3 B4*X4 B5*X5
C1*X1 C2*X2 C3*X3 C4*X4 C5*X5
D1*X1 D2*X2 D3*X3 D4*X4 D5*X5
objective function
constraint 1
constraint 2
constraint 3

Figure 4.2: Format of The Matrix

The Figure above show how the values are kept in matrix.

However, in order to train the neural network,matrix should contain additionally the slack variables and the right hand side values which will make the ultimate sizes of the matrix 4x9.

In order to see how the optimizer performs on an Integer Linear Programming example with larger number of variable and constraints, an additional set of data is created.Which will be helpful in order to analyze the difference of the solving time of the optimizer and the difference in the accuracy while performing a training on a neural network.

• Each sample will have 10 variables in objective function: x1,x2,x3,x4,x5,x6,x7,x8,x9,x10

•There will be 3 constraints added to model:

1.First constraint will contain all the 10 variables
2.Second constraint will contain first 9 variables
3.Third constraint will contain first 8 variables.

•All the values of the coefficients will be in range 0-100

•Right Hand Side value of the constraints will be in range 0-1000

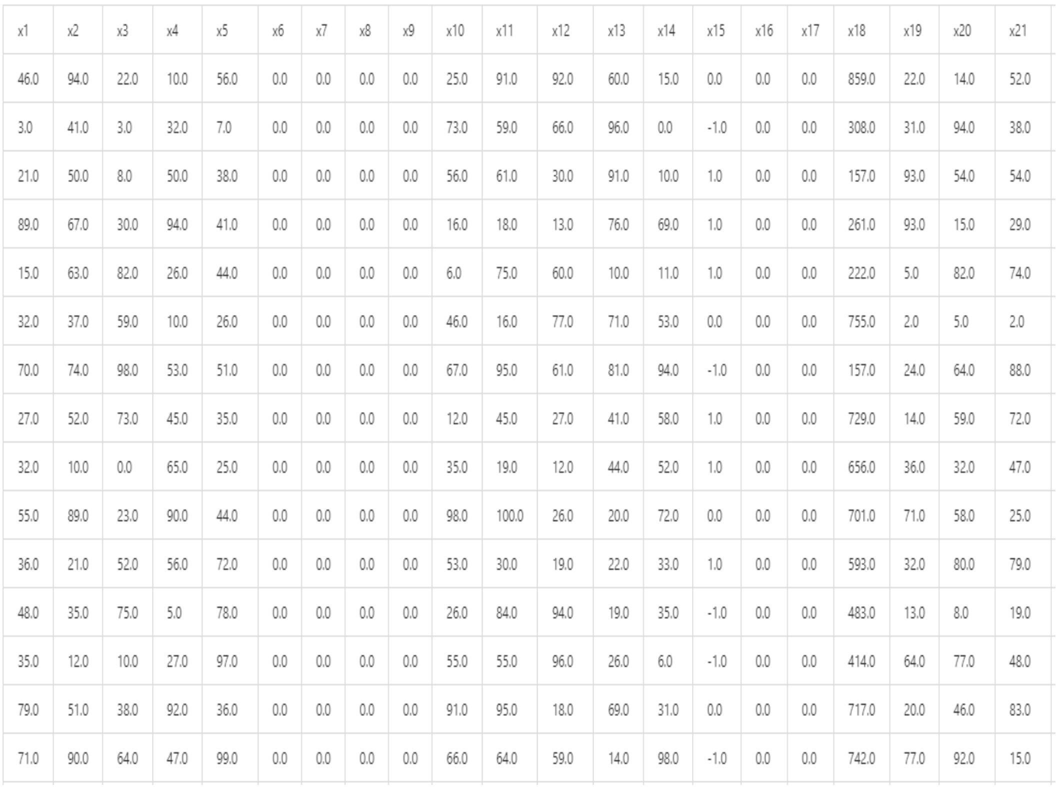
The Figure above show how the values are kept in matrix.

However, in order to train the neural network,matrix should include additionally the slack variables and the right hand side values which will make the ultimate sizes of the matrix 4x14.In the next section, how a matrix is kept in CSV file will be explained.

4.3 Representation in a CSV File

After generating random ILP examples and getting the solving time by using Gurobi Solver in JuMP,examples and their corresponding labels should have been kept in a data set to feed the neural network that will be created in next section.4x9 matrix is written as a regular 36 element array to CSV file and the 37th element is the label of that example.

With the help of the Gurobi Solver in JuMP,after generating each example, feasibility of that example is checked and only feasible solutions added to data set. Each example is optimized with 3 different algorithm 100 times for each.And the algorithm with minimum average solving time is assigned as label to 37th index.



x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21
46.0	94.0	22.0	10.0	56.0	0.0	0.0	0.0	0.0	25.0	91.0	92.0	60.0	15.0	0.0	0.0	0.0	859.0	22.0	14.0	52.0
3.0	41.0	3.0	32.0	7.0	0.0	0.0	0.0	0.0	73.0	59.0	66.0	96.0	0.0	-1.0	0.0	0.0	308.0	31.0	94.0	38.0
21.0	50.0	8.0	50.0	38.0	0.0	0.0	0.0	0.0	56.0	61.0	30.0	91.0	10.0	1.0	0.0	0.0	157.0	93.0	54.0	54.0
89.0	67.0	30.0	94.0	41.0	0.0	0.0	0.0	0.0	16.0	18.0	13.0	76.0	69.0	1.0	0.0	0.0	261.0	93.0	15.0	29.0
15.0	63.0	82.0	26.0	44.0	0.0	0.0	0.0	0.0	6.0	75.0	60.0	10.0	11.0	1.0	0.0	0.0	222.0	5.0	82.0	74.0
32.0	37.0	59.0	10.0	26.0	0.0	0.0	0.0	0.0	46.0	16.0	77.0	71.0	53.0	0.0	0.0	0.0	755.0	2.0	5.0	2.0
70.0	74.0	98.0	53.0	51.0	0.0	0.0	0.0	0.0	67.0	95.0	61.0	81.0	94.0	-1.0	0.0	0.0	157.0	24.0	64.0	88.0
27.0	52.0	73.0	45.0	35.0	0.0	0.0	0.0	0.0	12.0	45.0	27.0	41.0	58.0	1.0	0.0	0.0	729.0	14.0	59.0	72.0
32.0	10.0	0.0	65.0	25.0	0.0	0.0	0.0	0.0	35.0	19.0	12.0	44.0	52.0	1.0	0.0	0.0	656.0	36.0	32.0	47.0
55.0	89.0	23.0	90.0	44.0	0.0	0.0	0.0	0.0	98.0	100.0	26.0	20.0	72.0	0.0	0.0	0.0	701.0	71.0	58.0	25.0
36.0	21.0	52.0	56.0	72.0	0.0	0.0	0.0	0.0	53.0	30.0	19.0	22.0	33.0	1.0	0.0	0.0	593.0	32.0	80.0	79.0
48.0	35.0	75.0	5.0	78.0	0.0	0.0	0.0	0.0	26.0	84.0	94.0	19.0	35.0	-1.0	0.0	0.0	483.0	13.0	8.0	19.0
35.0	12.0	10.0	27.0	97.0	0.0	0.0	0.0	0.0	55.0	55.0	96.0	26.0	6.0	-1.0	0.0	0.0	414.0	64.0	77.0	48.0
79.0	51.0	38.0	92.0	36.0	0.0	0.0	0.0	0.0	91.0	95.0	18.0	69.0	31.0	0.0	0.0	0.0	717.0	20.0	46.0	83.0
71.0	90.0	64.0	47.0	99.0	0.0	0.0	0.0	0.0	66.0	64.0	59.0	14.0	98.0	-1.0	0.0	0.0	742.0	77.0	92.0	15.0

Figure 4.3: CSV File Format

5 Creating Neural Network

In order to create a neural network model and train with the data set that has been created in the previous sections,Knet framework is used.

Knet is the Koç University deep learning framework implemented in Julia by Deniz Yuret and collaborators.

5.1 Creating a Knet Model

The training is done on 2 different types of neural network model in Knet for the comparison:

1.Multi-layer Perceptron

A multi-layer perceptron, a fully connected feed-forward neural network, is basically a bunch of linear regression models stuck together with non-linearities in between.

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function.

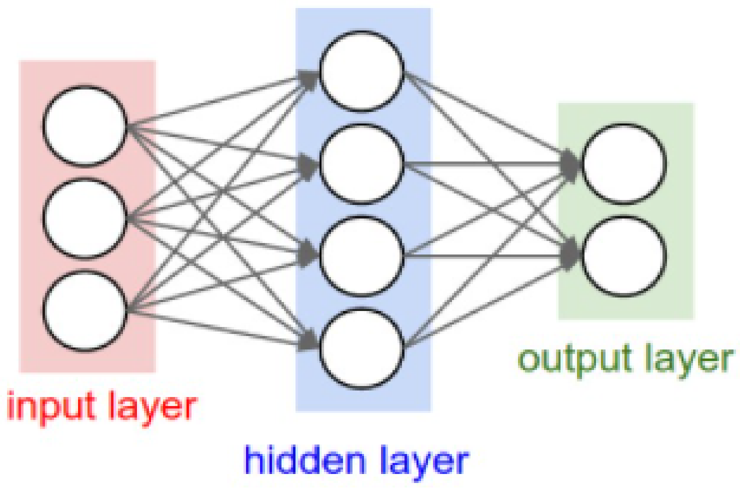


Figure 5.1: Multi-layer Perceptron Model

2.Convolutional Neural Network(CNN)

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer.

CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

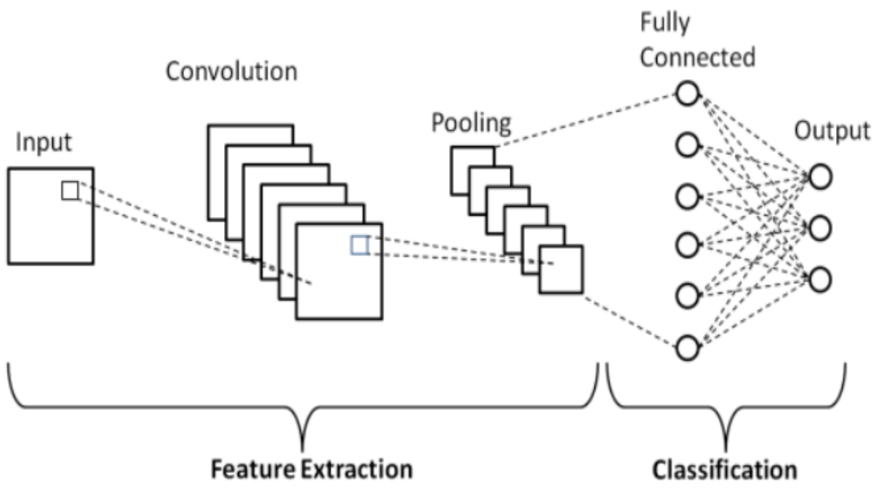


Figure 5.2: Convolutional Neural Network

5.2 Preparing the Data For Training

Training data is kept in CSV format as 37 and 57 element array including last value as label.
First of all,it needs to be converted to 4x9 and 4x14 matrix and labels should be kept in a single dimension list.After making operations in Julia,data is converted to 4x9 and 4x14 Matrix.

However model expects a Matrix with the same size in height and weight.For this reason, zero-padding is applied to matrix before passing to the model.

6 Results and Observations

6.1 Accuracy

The data set is consisting of 60000 example for training and 10000 for test data for 5-variable and 10-variable problems for each.

Problems are solved with 3 different method, 100 times by each and average solve time is calculated before assigning labels.

1.Dual Simplex Algorithm
2.Barrier Algorithm
3.Concurrent

1.Data Set with 5-Variable Examples

a.Multi-layer Perceptron

Training the model with epoch number 10 give the result of maximum **0.73** accuracy. Below,Figure 6.1 shows the changing of the accuracy on each epoch from 1 to 10.

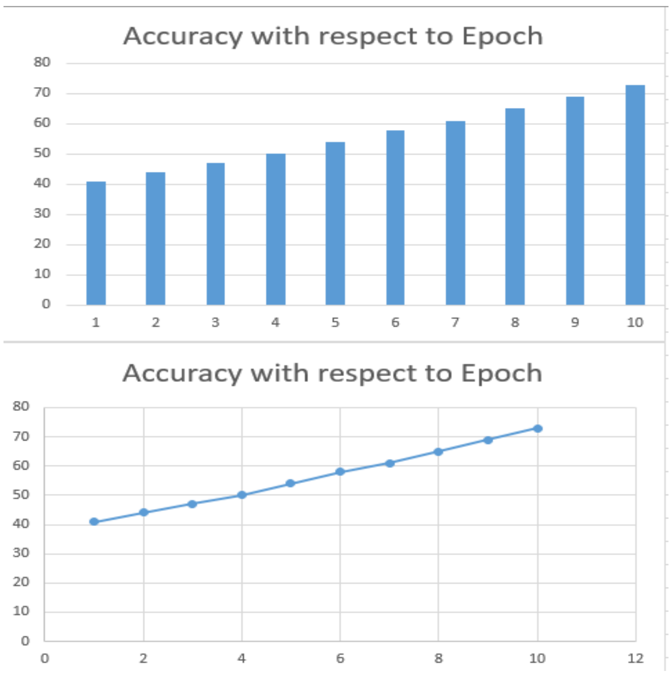


Figure 6.1: Accuracy Graph of MLP with 5-Variable Examples

b.Convolutional Neural Network(CNN)

Training the model with epoch number 10 give the result of maximum **0.7448** accuracy. Below,Figure 6.2 shows the changing of the accuracy on each epoch from 1 to 10.

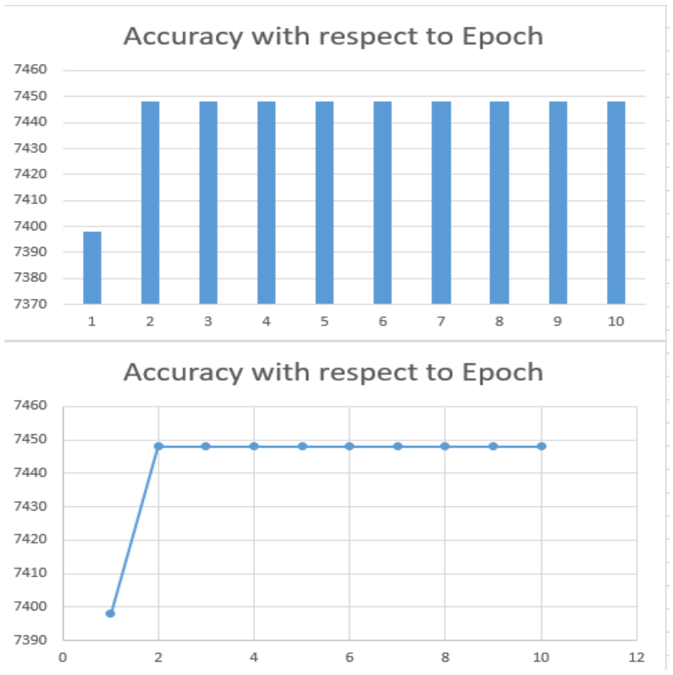


Figure 6.2: Accuracy Graph of CNN with 5-Variable Examples

2.Data Set with 10-Variable Examples

a.Multi-layer Perceptron

Training the model with epoch number 10 give the result of maximum **0.810** accuracy. Below,Figure 6.3 shows the changing of the accuracy on each epoch from 1 to 10.

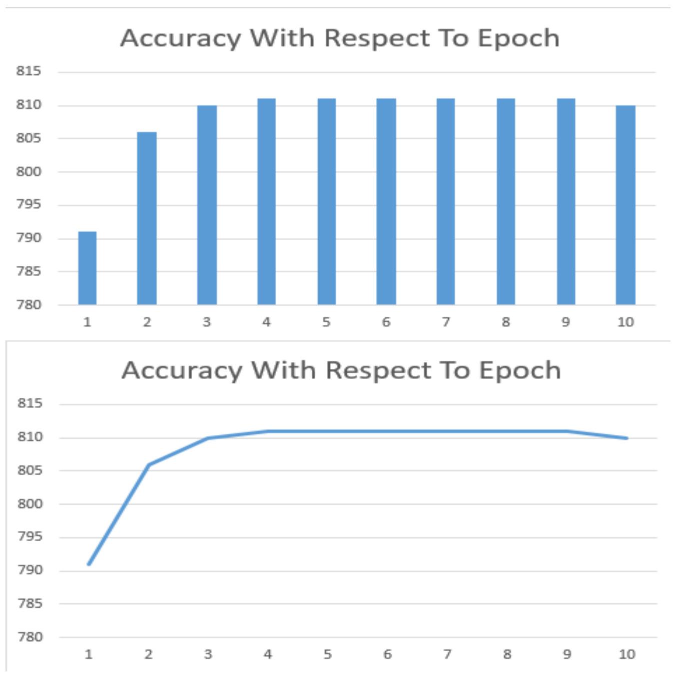


Figure 6.3: Accuracy Graph of MLP with 10-Variable Examples

b.Convolutional Neural Network(CNN)

Training the model with epoch number 10 give the result of maximum **0.810** accuracy. Below,Figure 6.4 shows the changing of the accuracy on each epoch from 1 to 10.

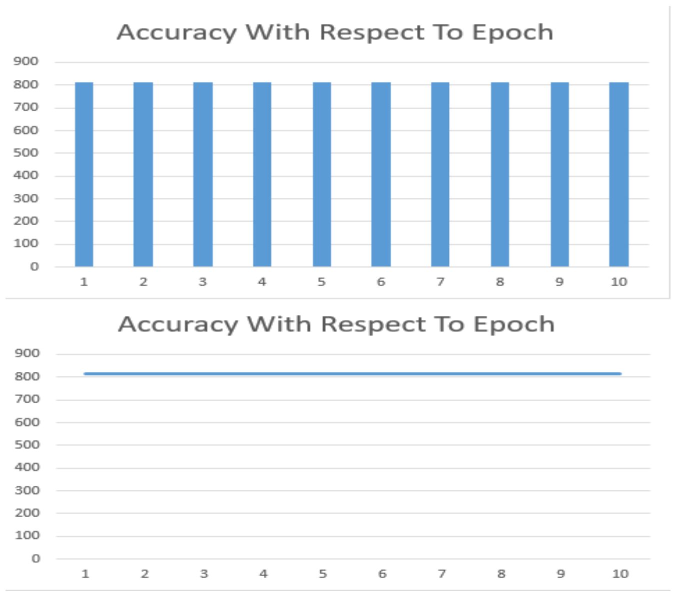


Figure 6.4: Accuracy Graph of CNN with 10-Variable Examples

6.2 Observations

The barrier algorithm is mainly used for large and complex examples,but it is more numerically sensitive. If a numerical issue happens, when the barrier algorithm converges, the simplex algorithm that usually follows can wait due to numerical issues.Since the data set in this project is created with relatively large numbers and all the data is checked before added to data set,the barrier algorithm is picked by the Gurobi optimizer mostly.

The dual simplex algorithm is a better choice when the data of problems may have feasibility problems in acceptable time, since it is more stable to numerical issues. Usually, for the optimization examples, problem samples don't have considerable issues numerically.For this reason, deciding on the dual simplex algorithm can cause prevention from taking advantage of the performance of the barrier algorithm on data with no numerical issues. In such cases, concurrent optimizer should be used, which uses multiple algorithms simultaneously and returns the solution from the fastest one to solve.

The test results shows that the accuracy on test data with 10-variable examples are slightly higher than the examples with 5-variable.

7 References

• [1] Philip I. Thomas, Video — Optimization in Julia, Medium.com, June 27,2016

• [2] Gurobi Optimizer, gurobi.com

• [3] JuMP Modeling language and supporting packages for mathematical optimization in Julia, jump.dev

• [4] Knet Framework for Julia ,Deniz Yuret, github.com/denizyuret/Knet.jl

• [5] Wikipedia, Integer Programming, en.wikipedia.org/wiki/Integer_programming,