LuaNet

Lua Artificial Neural Network by João P. Maia

Introduction

(In the following file you will find a description of all functions as well as usage examples and solutions to problems)

LuaNet, a module for using neural networks in Lua! This module is a tool easy to use for beginners and at the same time powerful enough to be used in small projects

Inspired by the Toy Neural Network, this module aims to reach all people who are entering the world of machine learning and want to implement it in their projects and games.

In view of this, this module comes with functions for creating neural networks, as well as manipulating matrices. All of this is compatible with LOVE2Das well as with all other platforms and projects that use lua.

As mentioned earlier, the objectives of this module are:

- Be friendly and easy to use for beginners
- Be compatible with the maximum of platforms, projects, systems and etc. that use lua

It is worth mentioning that this project **is not a library for high-level projects**, that is: that it has all the capacities and technologies for application in professional and academic environments, for that I recommend using tools such as pytorch.

Getting Started

Installing

1. Download and extract to anywhere you wish. It is important that the files “NeuralLua.lua”, “Matrix.lua” and “csvHandler.lua” stay together in the same folder
2. Sometimes you may have to adjust the location of the files in the code, if any of these files cannot find the other open the code and follow the instructions commented above the “require” of each file

Module Issue

The first time the following error can happen (in one or more files):

```
1.  Exception has occurred: .../Desktop/Programacao/LUA/LuaANN/LuaNet/Lib/NeuralLu
    a.lua:5: module './LuaNet.Lb.Matrix' not found:
```

```
2.      no field package.preload['./LuaNet.Lb.Matrix']
3.      no file path/LuaNet\Lb\Matrix.lua'
4.      no file path/\/LuaNet\Lb\Matrix.dll'
5.      no file 'path\LuaANN/.dll'
```

If this error occurs, open the file in which it is appearing. Look for the comment "--module issue fix". Soon after, you will have how to solve it.

In short you must change the require path so that it points to the right file

Round issue

In LUA there is a limitation of decimal places so the whole program is configured to round to 10 decimal places to avoid errors. You can change this by modifying this function:

```
1.  function round(number, decimals)
2.      decimals = 10 ←change this or decimals
3.      local power = 10^decimals
4.      return math.floor(number * power) / power
5.  end
```

Examples

Xor Problem

Let's see how our neural network can solve a problem Xor, if you never heard about I recommend reading this link: [mlopt.com]

In summary, the truth table for an Xor problem is as follows:

Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

The purpose of our neural network will be to "predict" the following outputs based on the inputs. Note that these points are not linearly separable, that is, you cannot draw a line to divide the data.

First let's import the modules:

```
1.  local nn = require("LuaNet.Lib.NeuralLua")
2.  local matrix = require("LuaNet.Lib.Matrix")
```

The "nn" module will take care of the neural networks, while the "matrix" module will take care of matrix operations. Now we are going to create a neural network called "brain" with two inputs, a hidden layer with 3 neurons and 1 input:

```
1.  brain = nn.new({2,3,1})
```

Now let's create a training loop and pass the information from the table. Our neural network will be trained 200 times with the information set in the table:

```
1.  while i<200 do
2.    brain = nn.train(brain,{1,1},{0})
3.    brain = nn.train(brain,{0,1},{1})
4.    brain = nn.train(brain,{1,0},{1})
5.    brain = nn.train(brain,{1,1},{0})
6.    i= i+1
7.    print("Epoch"..i)
8.  end
```

Note that line *2.* represents the first line of the table: 1 and 1 results in 0. The other lines follow the same pattern.Now our neural network is trained, let's test it!

To test it we will pass a variable *pr* that will keep the matrix resulting from this prediction

```
1.  pr = nn.preddict(brain,{0,1});
2.  matrix.print(pr)
```

To test it we will pass a variable pr that will keep the matrix resulting from this prediction. We are using the second row of the table, so the output must be 1.

Using matrix.print (pr) to display the result matrix, we see that the value approaches 1. Testing more we will see that the neural network works.

Full code bellow:

```
local nn = require("LuaNet.Lib.NeuralLua")
local matrix = require("LuaNet.Lib.Matrix")

local brain= nn.new({2,3,1})

local i = 0
while i<200 do
  brain = nn.train(brain,{1,1},{0})
  brain = nn.train(brain,{0,1},{1})
  brain = nn.train(brain,{1,0},{1})
  brain = nn.train(brain,{1,1},{0})
  i= i+1
  print("Epoch"..i)
end

pr = nn.preddict(brain,{0,1});
matrix.print(pr)
```

In the Demo folder there is a complete version of this code

Iris dataset

Now we are going to use our neural network in a very famous problem: Classification of iris flowers. For more information about I recommend the link: [Iris Dataset]

We will import this data from a .csv file. I got this file from the Kaggle website. In the file we have the following information:

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Iris Selosa
...	...	...	...	...

From the table we can see that there are 3 types of iris and 4 characteristics and define it.

We will import the modules and create a neural network with 4 inputs and 3 outputs

```
nn = require('LuaNet.Lib.NeuralLua')
mat = require('LuaNet.Lib.Matrix')

Brain = nn.new({4,5,3})
```

Now let's go to the import part of the csv data. First let's remember that this is a classification problem in case we want to classify the fifth column of the table. Soon we will transform the *strings* into *int* when creating the dataset.

We are passing a table with the columns that have strings, the column that will be the output, the location of the file and the separator that is: how the information in the csv file is separated. In this case with ",":

```
dataset = nn.newdataset({5},5,'Demos/Iris/iris_dataset.csv',",")
```

Now let's make a table with the training options of dataset.lr refers to the learning rate, Epochs the number of times the table will be trained, validation is how many percent of the table will be used to check the neural network's accuracy, debug will display information during training and label indicates that this is a classification problem.

```
options = {lr = 0.01,
   epochs = 150,
   validation = 0.20,
   debug = true,
   label = true}
```

Now we are going to train the neural network using the options:

```
Brain = nn.traindataset(Brain,dataset,options)
```

We will be able to see the training because we activate the debug true option. At the end we will see that the final precision will be around 1. As weights are started randomly there are times that this can change but trainees again and you should get results tending to 1.

Now let's move the forecast matrix to the pr variable. Finally we will transform the matrix into a label, using the *nn.getlabel ()* function that receives the dataset and the prediction matrix.

```
pr = nn.preddict(Brain,{5.1,3.5,1.4,0.2})

lbl = nn.getlabel(dataset,pr)

print(lbl)
```

If we use the above values we will see that the program will print "Iris Selosa".

Full code bellow:

```
nn = require('LuaNet.Lib.NeuralLua')
mat = require('LuaNet.Lib.Matrix')

Brain = nn.new({4,5,3})

dataset = nn.newdataset({5},5,'Demos/Iris/iris_dataset.csv',",")

  options = {lr = 0.01,
     epochs = 150,
     validation = 0.20,
     debug = true,
     label = true}

pr = nn.preddict(Brain,{5.1,3.5,1.4,0.2})

lbl = nn.getlabel(dataset,pr)

print(lbl)
```

In the Demo folder there is a complete version of this code

Dataset:

https://www.kaggle.com/arshid/iris-flower-dataset

Saving/Loading

Now we are going to use our Iris Dataset from the last example to demonstrate how to save and load neural networks.

```
1.  --using iris example
2.  nn.save(Brain,"./test/","IRIS")
```

When using the "*nn.save*" function we pass neural network as first parameter, a path where all the files will be saved. I recommend using a folder for each NN. We give a name for our model, in this case "IRIS", if any name is given a random one is chosen.

Now let´s load our saved NN:

```
1.  --Loading
2.  Brain_loaded = nn.load("./test/","NNIRIS")")
```

Now we have the variable "*Brain_Loaded*" will receive a complete trained neural network. Note that the first parameter is the location of the neural network and the second is its name with NN before

A little bit of genetic algorithms

Now we are going to talk a little bit about genetic algorithms, if you never heard about, I recommend you read the articles at the end of this section before reading this. First, let's load a Random NN. For these examples, we aren't going to train it. Just predict random values and see how to use Genetic Functions.

```
1.  local Brain2 = nn.load("./test/","NNGen")
2.  pr = nn.preddict(Brain2,{1,2,3,4})
3.  mat.print(pr);
```

Because we are loading a NN we are going to see that the prediction never changes, in my case it's the following matrix:

```
1.  0.4779791131 |
2.  0.5291821749 |
3.  0.7972476588 |
```

Now let's apply the mutation function. By default, the mutation occurs by shuffling 50% of the weights. so, we are going to see that the predictions have changed:

```
1.  0.6444973491 |
```

```
0.3524559467 |
0.5227299021 |
```

Let's do it again but this time we are going to add some parameters: First, we are going to change de mutation type to function, this is: every weight value will pass through a function, in this case, the hyperbolic tangent.

We are going to set the mutation rate to 0.8, that means 80% of the NN will be mutated

```
mutation_type = "function"
function_mutation = math.tanh;
mutation_rate = 1;

Brain2 = nn.mutate(Brain,mutation_type,mutation_rate,function_mutation)
pr = nn.preddict(Brain2,{1,2,3,4})

mat.print(pr);
```

Finally, let's use the crossover function. We are going to load two different NN, one of them we will mutate and then we will join both of then again creating a "child".

```
local Brain1 = nn.load("./test/","NNIRIS")
local Brain2 = nn.load("./test/","NNIRIS")

Brain2 = nn.mutate(Brain)

--Child
Brain3 = nn.crossover(Brain1,Brain2)

--Preddict using the Child
pr = nn.preddict(Brain3,{1,2,3,4})

mat.print(pr);
```

Now we have the prediction of two combined NN.

For more info:

https://www.kindsonthegenius.com/blog/2018/11/what-is-genetic-algorithm-a-simple-and-detailed-explanation.html

Functions by file

NeuralLua.lua

Basic

nn.new(neurons,activation?)

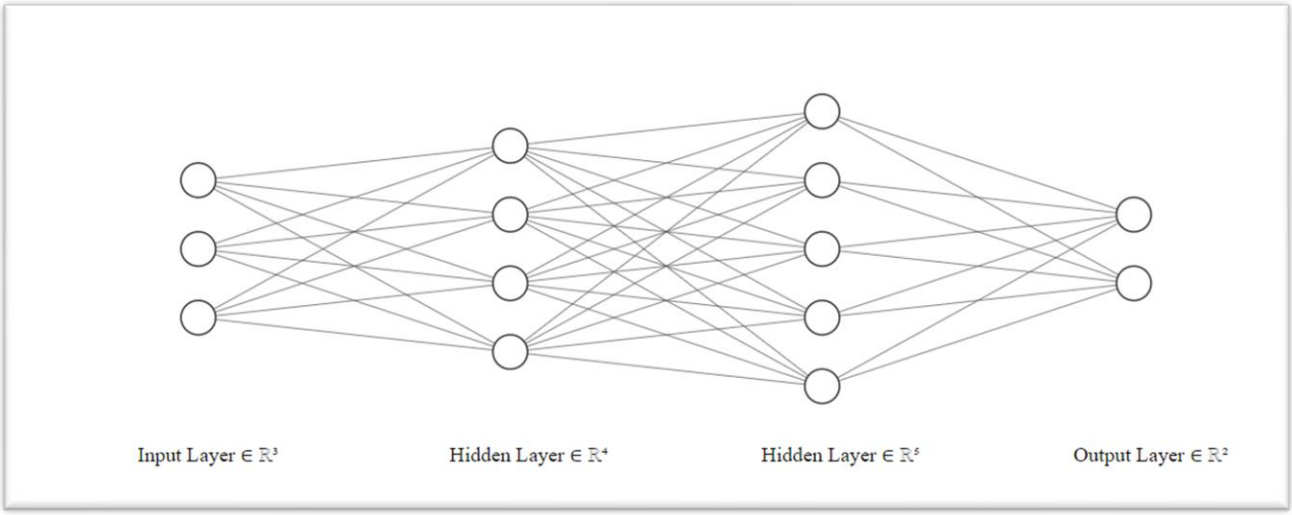
Neurons: receives a table of integers values where each index represent a layer.

Activation: Activation function, by default Sigmoid but you can also use hyperbolic tangent (see activation functions)

Example:

```
1.  neuralNet = nn.new({3,4,5,2})
```

Will create the following neural network (the Bias will be created automatically):



nn.preddict(neural,x,training?)

Neural: Receives the NN that will do the prediction

X: Table with inputs

Trainning: is never goind to be used by the user. It's used only inside *traindataset* code

Returns a prediction Matrix, you can convert to a normal table by using *mat.toarray(m)*

nn.train(neural,input_x,target_y,lr?)

Neural: *Receives the nn that will be trained*

Input_x: Table with the inputs

Input_y: Table with the desired outputs for the given inputs

Lr: Learning rate, by default is 0.01

Returns a trained NN

Dataset Functions

nn.newdataset(labels,target,file,sep?)

Labels: A table with the columns number that are labels (not numerical values)
Target: The number of input column
Sep: How the data is separated in csv file, by default is ","
Example:

Animal	Has Feathers	Number of Legs	Species
Chicken	1	2	Bird
Salmon	0	0	Fish

The output its going to be the species so if we load from csv dataset will use the following code:

```
1.  data = nn.newdataset({1,4},4,"/location/data.csv")
```

Inside the table we put columns 1 and 4, the non-numerical columns, they will be converted to an integer.
Returns a dataset.

nn.traindataset(neural,dataset,options?)

Neural: Receives the NN that will be trained in the dadataset
Dataset: Receives a dataset table from the *newDataset()* function
Options: A table with the following options

- lr: Learning rate, default 0.1
- epochs: default 150
- debug: Boolean default is true. Show the process of training and the estimated accuracy
- label: Boolean, true if the output is a label. Default is False
- Validation: The percentage of data that will be used for validation, by default is 0.20 (20%)

Example:

```
1.  options={
2.  lr = 0.01,
3.  epochs = 100,
4.  validation =0.10,
5.  debug = false}
6.  neural = nn.traindataset(neural,dataset,options)
```

Returns a trained NN

nn.getlabel(dataset,mat)

This function is used to transform a result matrix (only NN that uses labels) into a label
dataset: The dataset with a string's outputs
mat: Receives the result matrix
Returns a string, the label with the highest "score"

Save and Load Functions

nn.save(neural,path,personal_name)

Neural: Receives a NN table

Path: Receives a path to a folder where the NN will be saved

Personal_name: The name of the NN, by default is a random number. If you give the name "ruby" its going to be saved as "NNruby".

nn.load(path,name)

path: Receives a string with the folder path

name: The name of the NN in this format: "NN(namehere)"

Neuroevolutionary Functions

nn.copy(neural)

This functions simply receives a NN and returns its copy

nn.mutate(neural,mutation_type?,mutation_rate?,func_mutation?)

If you never heard about mutations I recommend reading this article: TutorialsPoint-Mutation.

Neural: Receives a NN table.

Mutation: Receives a string with the type of the mutation:

- Shuffle: Shuffle the weights (default)
- Random: randomize the weights
- Function: apply a function in each weight (in mutation_rate range)

Mutation_rate: How much of the weights will change. By default, is 0.5 (50%)

Func_mutation: If mutation_type is "function" then this will receive the function that will be applied in each weight (in mutation_rate range).

nn.crossover(neuralA,neuralB)

If you never heard about crossover I recommend reading this article: GeeksForGeeks-Crossover.

Receives two NN and crossover then. Returns the "child" with 50% of each NN.

Activation Functions

Using activation functions

Its very simple to use different activation function. You can configure a NN to use it when you initialize it:

```
1.  --Using sigmoid
2.  Neural = nn.new({1,2,3},"sigmoid")
3.  --Using hyperbolic tangent
4.  Neural = nn.new({1,2,3},"tanh")
```

Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

This is the default activation function used in the NN. If you don't know what is a activation function I recommend reading this: Medium-Understanding Activation Functions

Derivative:

$$\frac{d}{dx} f(x) = f(x) * (1 - f(x))$$

Hyperbolic Tangent

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

More about hyperbolic tangent: Wolfram-Tanh

Derivative:

$$\frac{d}{dx} \tanh(x) = 1 - \tanh(x)^2$$

Matrix.lua

Creating matrices

mat.new(rows,cols,values)

Receive a number of rows, columns and the values to fill the matrix
Rows: receives an integer >0
Cols: receives an integer >0
Values: receives a table of values: {*Val1*,*Val2*..*Valn*}
Returns a matrix table
The example bellow creates de following matrix:

```
m = mat.new(3,2,{1,2,3,4,5,6})
```

$$M = \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix}$$

mat.newfromarray(array)

Receive a array/table and transform in a Row(N)xCol(1) matrix
Returns a matrix table
The example bellow creates de following matrix:

```
m = mat.newfromarray({1,2,3})
```

$$M = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

mat.newzero(rows,cols)

Receive a number of rows, columns
Rows: receives an integer >0
Cols: receives an integer >0
Fills all positions with zeros
Returns a matrix table

mat.newrdn(rows,cols,positive?)

Receive a number of rows, columns and an optional Boolean parameter
Rows: receives an integer >0
Cols: receives an integer >0
Positive: it is disabled by default, that is, the random values assigned will be between -1 and. If the True parameter is passed to positive then only values from 0 to 1 will be generated
Returns a matrix table filled with random values

mat.newfromlabel(rows,label_index)

This function is specific to the neural network.
Transform a label into an array

Matrix Modifications

mat.innersum(m1)

Receives an m1 matrix and returns the sum of all its elements
Returns the inner sum value

mat.sum(m1,m2)

Receives two matrices and returns their sum
For more matrix sum information: Matrix Addition

mat.sub(m1,m2)

Subtract the two matrices it receives and return its result: M1 -M2
For more matrix sum information: Matrix Subtraction

mat.mult(m1,m2)

Calculate Hadamard product from the two matrices it receives
Returns the product
More information about Hadamard product: Hardman product

mat.dot(m1,m2)

Receives two matrices m1 and m2
Returns the product m1m2
For more information: Matrix dot product

mat.mult1D(m1,scalar)

Take an m1 matrix and a scalar value and multiply each value in the matrix by the scalar value
Returns a matrix

Matrix Modifications

mat.transpose(m1)

Receives an M matrix and returns it transposed
For more information: Matrix Transpose

mat.map(m1,func)

Receives an M matrix and a function
Apply the function to each element of the matrix
returns the new matrix with each element applied to the function

mat.toarray(m1)

Receives an M1 Matrix
Transform the array into a simple array / table

returns the array

Uncategorized functions

mat.print(m,debug?,separator?)

Receives an *M matrix*
Receives *Boolean debug* value. Default is false
Receives 'separator' character. Default is '|'
print the matrix 2x2 in the following format:

1|2|
3|4|

csvHandler.lua

shuffle(array)

Fisher-yates shuffle, receives an array and return it shuffled
Bibliography: https://scriptinghelpers.org/questions/17917/is-there-a-built-in-function-to-randomize-liststables

csv.readfile(arq)

csv.parseCSVLine(line,sep)

Receives a line and a separator and return a table of the given line
Bibliography: http://lua-users.org/wiki/LuaCsv

csv.totrain(arq,sep)

arq: Receives a string path for the csv file
sep: Receives a separator character, this is the separator of the csv file data. By default is ","

Bibliography

Books:

1. Tariq Rashid “Make you Own Neural Network”: https://www.amazon.com/Make-Your-Own-Neural-Network-ebook/dp/B01EER4Z4G

Sites:

1. Neural Networks: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464
2. Neural Networks: https://playground.tensorflow.org/
3. Activation Functions: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
4. Activation Functions: https://mlcheatsheet.readthedocs.io/en/latest/activation_functions.html
5. Activation Functions: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
6. Perceptron: https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/
7. Multilayer Perceptron: https://machinelearningmastery.com/neural-networks-crash-course/
8. Multilayer Perceptron: http://deeplearning.net/tutorial/mlp.html

You tube:

1. 3Blue1Brown: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
2. Coding Train: https://www.youtube.com/playlist?list=PLRqwX-V7Uu6aCibgK1PTWWu9by6XFdCfh

Version 1 by JPMaia