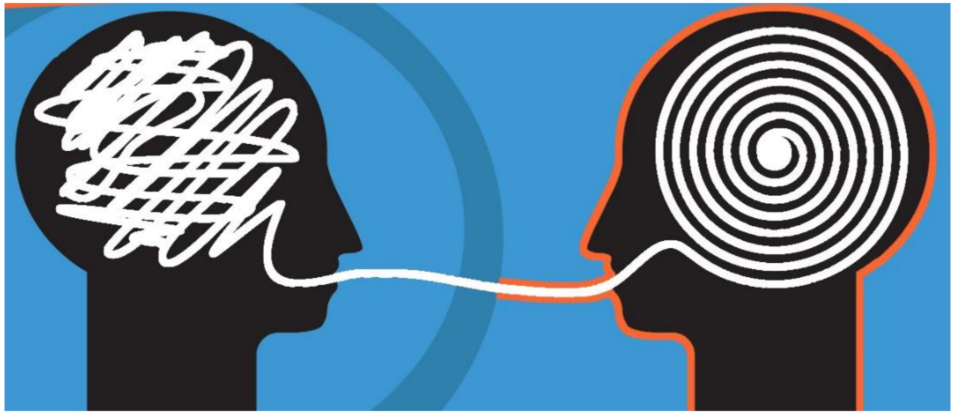

Evaluating Occurrence of a *Ramanujan Lakshmana Super Magic Square* via Deep Learning techniques with Keras

What is a *Ramanujan Lakshmana Super Magic Square*?

Ramanujan Square:

Let us consider a square with its cells as given alongside.

R_1C_1	**R_1C_2**	**R_1C_3**	**R_1C_4**
R_2C_1	**R_2C_2**	**R_2C_3**	**R_2C_4**
R_3C_1	**R_3C_2**	**R_3C_3**	**R_3C_4**
R_4C_1	**R_4C_2**	**R_4C_3**	**R_4C_4**

For this square to be a ***Ramanujan Magic Square*** for the birth date *'dd/mm/ccyy'*, we assign R_1C_1=*dd*, R_1C_2=*mm*, R_1C_3=*cc*, R_1C_4=*yy* and fill the remaining cells satisfying the following rules/properties:

> ***<u>Property i:</u>*** The sum of the numbers of every cell in *each row*, *column* and *diagonal* as well as that of the *four corners* should be a *constant*
>
> *i.e.,* $[R_nC_1 + R_nC_2 + R_nC_3 + R_nC_4] = [R_1C_n + R_2C_n + R_3C_n + R_4C_n] = [R_1C_1 + R_2C_2 + R_3C_3 + R_4C_4] = [R_1C_4 + R_2C_3 + R_3C_2 + R_4C_1]$
>
> $= [R_1C_1 + R_4C_1 + R_1C_4 + R_4C_4] = constant$

Property ii: The sum of the numbers in each *2x2 boxes* should be equal to same *constant* except the *[R_1C_2, R_1C_3, R_2C_2, R_2C_3]* and *[R_3C_2, R_3C_3, R_4C_2, R_4C_3]* boxes

For example, in the squares shown below, the cells highlighted with the same colour correspond to a box

R_1C_1	**R_1C_2**	**R_1C_3**	**R_1C_4**
R_2C_1	**R_2C_2**	**R_2C_3**	**R_2C_4**
R_3C_1	**R_3C_2**	**R_3C_3**	**R_3C_4**
R_4C_1	**R_4C_2**	**R_4C_3**	**R_4C_4**

R_1C_1	R_1C_2	R_1C_3	R_1C_4
R_2C_1	**R_2C_2**	**R_2C_3**	R_2C_4
R_3C_1	**R_3C_2**	**R_3C_3**	R_3C_4
R_4C_1	R_4C_2	R_4C_3	R_4C_4

R_1C_1	R_1C_2	R_1C_3	R_1C_4
R_2C_1	**R_2C_2**	**R_2C_3**	**R_2C_4**
R_3C_1	**R_3C_2**	**R_3C_3**	**R_3C_4**
R_4C_1	R_4C_2	R_4C_3	R_4C_4

Property iii: All the numbers of the square must belong to the set of natural numbers N and should be unique with the exception of the first row (in case of repetition in the birth date itself)

On solving for such a square in accordance with the above properties using *linear algebra*, we obtain the values of each cell of the *Ramanujan Square* as given below:

dd	*mm*	*cc*	*yy*
aa	*cc + yy - aa*	*-mm - yy + aa + 2 * bb*	*dd + 2 * mm + yy - aa - 2 * bb*
bb	*dd + mm - bb*	*mm + yy - bb*	*-mm + cc + bb*
mm + cc + yy - aa - bb	*-mm + aa + bb*	*dd + mm + yy - aa - bb*	*-yy + aa + bb*

where '*aa*' and '*bb*' are variants to be adjusted to fulfil the above *property iii*

Note:

i. *aa* cannot exceed *cc + yy*; since $R_2C_2 = cc + yy - aa$

ii. *bb* cannot exceed *dd + mm* and *mm + yy*; since $R_3C_2 = dd + mm - bb$ and $R_3C_3 = mm + yy - bb$

Ramanujan Lakshmana Magical Square:

If ***bb = (dd + mm - cc + yy) / 2*** then all adjacent *2x2 boxes* will satisfy *property ii* including the *[R_1C_2, R_1C_3, R_2C_2, R_2C_3]* and *[R_3C_2, R_3C_3, R_4C_2, R_4C_3]* boxes.

This square is a '***Ramanujan Lakshmana Super Magic Square'***.

R_1C_1	**R_1C_2**	**R_1C_3**	R_1C_4
R_2C_1	**R_2C_2**	**R_2C_3**	R_2C_4
R_3C_1	**R_3C_2**	**R_3C_3**	R_3C_4
R_4C_1	**R_4C_2**	**R_4C_3**	R_4C_4

Who will have a Super Magic Square?

For a *Super Magic Square*, *bb = (dd + mm - cc + yy) / 2*. Consequently, the third row becomes

(dd + mm - cc + yy) / 2	(dd + mm + cc - yy) / 2	(-dd + mm + cc + yy) / 2	(dd - mm + cc + yy) / 2

Corollary: In order for the third row to comply with *property iii*, *dd + mm + cc + yy* must be **even number** and *(dd + mm + cc + yy)/2* must be greater than *dd*, *mm*, *cc* and *yy*.

Let *dd*, *mm*, *cc* take the highest possible values for this century i.e. *dd=31*, *mm=12*, *cc=20*. Now we will find yy to suit the corollary.

=> 31.5 +(yy/2) > {31, 12, 20, yy}

=> 31.5 > (yy/2)

=> yy < 63.

Hence to obtain *Super Magic Square* for this century, it is necessary to have ***yy < 63***.

Now let us know, how to apply deep learning to predict whether magic square and super magic square using Keras.

Keras:

Keras is a powerful high-level *neural network API* and an *easy-to-use, free, open source* Python library for developing and evaluating *deep learning models* in just a few lines of code. It wraps the efficient numerical computation libraries – Theano, TensorFlow, and CNTK and is designed to run on the CPU as well as the GPU.

'Deep Learning' steps:

Python Program to illustrate the use of Deep Learning to predict the Occurrence of a Ramanujan Lakshmana Super Magic Square for given dates:

1. **Load historical data**

```
# for loading data Let us take first week of 2019
start_date = date(2019, 1, 1)
end_date = date(2019, 1, 7)

# load magic square data for weeks
dataset = find_squares_dates(start_date, end_date)

# range of the input columns in the dataset
xColumns = numpy.r_[0:5, 8:9]
```

```
# range of the output columns in the dataset
# let us only evaluate magic or special for testing purpose.
yColumns = numpy.r_[16:18]
# otherwise un comment the following
# yColumns =  numpy.r_[5:8, 9:21]

# split into input and output variables
X_input = dataset[:, xColumns]
Y_output = dataset[:, yColumns]
```

2. **Create training and testing data**

```
# seed is initialized to get same results
seed = 5
numpy.random.seed(seed)

# split the data into training (70%) and testing (30%)
(X_train, X_test, Y_train, Y_test) = train_test_split(X_input, Y_output, test_size=0.30, random_state=seed)
```

3. **Define Keras model**

```
# create the model
model = Sequential()
# add the first hidden layer of input columns with 8 nodes, also called a Neuron or Perceptron
model.add(Dense(8, input_dim=xColumns.size,  activation='relu', kernel_initializer='uniform'))
# add the second hidden layer with 6 nodes
model.add(Dense(6,  activation='relu', kernel_initializer='uniform'))
# finally add output columns
model.add(Dense(yColumns.size,  activation='sigmoid', kernel_initializer='uniform'))
```

4. **Compile Keras model**

```
# compile the model
# loss function : maximum likelihood, error between 2 probability distributions measured using cross-entropy
# optimizer: The Adam optimization algorithm, an extension to stochastic gradient descent to increase the learning rate
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

5. **Train model on training data**

```
# fit the model
# training occurs over epochs(One pass through all rows in training dataset) and each epoch is split into batches
```

```
# verbose is to display the training sessions in console
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=5, verbose=1)
```

6. **Evaluate model on test data**

```
# finally evaluate the model
scores = model.evaluate(X_test, Y_test)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

7. **Guess predictions via trained model**

```
# initalize to randomly generate 100 dates
predict_date_variables: numpy.ndarray = numpy.ndarray((0, 6), dtype=int)
# randomly generate 100 dates
for myrandom in range(99):
    myRandArray = numpy.array([1+get_random_number(27), 1+get_random_number(11), 18+get_random_number(2),
                    1+get_random_number(98), 1+get_random_number(98), 1+get_random_number(98)],dtype=int)
    predict_date_variables = numpy.concatenate((predict_date_variables, [myRandArray]), axis=0)
# guess predictions for given days at a time
predictions = model.predict(predict_date_variables)
pointer = 0
for each_day in predict_date_variables:
    print("date: %d / %d / %d%d " %(each_day[0], each_day[1], each_day[2], each_day[3]))
    print("Magic Availability: %.2f%% Super Magic Availability: %.2f%%"
          %(predictions[pointer][0]*100, predictions[pointer][0]*100) )
```

Full Source Code is copied below for the reference:

```
# organize imports
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from datetime import timedelta, date
import numpy

#Square basic class
class Square:

    def __init__(self):
        self.dd=0
        self.mm=0
```

```
        self.cc=0
        self.yy=0
        self.aa=0
        self.bb=0
        self.n1=0
        self.n2 = 0
        self.n3 = 0
        self.n4 = 0
        self.n5 = 0
        self.n6 = 0
        self.n7 = 0
        self.n8 = 0
        self.n9 = 0
        self.n10 = 0

# Collection of Squares Class
class SquareCollection:
    def __init__(self):
        self.magic: bool = False
        self.special:bool = False
        self.negatives:int = 0
        self.zeros:int = 0
        self.repetition:int = 0
        self.numbers = []

    # funtion to print the square
    def __str__(self):
        return "{ repetition: " + self.repetition.__str__() + "; zeros: " + self.zeros.__str__()  + "; magic: " \
               + self.magic.__str__() +  "; special: "  + self.special.__str__()+  "; numbers: "  + \
               self.numbers.__str__() + " }"

    # function to convert elements to array
    def toArray(self,oneSquare:Square):
        self.numbers = [oneSquare.dd, oneSquare.mm, oneSquare.cc, oneSquare.yy, oneSquare.aa, oneSquare.n1,
                        oneSquare.n2, oneSquare.n3, oneSquare.bb, oneSquare.n4, oneSquare.n5, oneSquare.n6,
                        oneSquare.n7, oneSquare.n8, oneSquare.n9, oneSquare.n10]

        self.negatives = 0
        self.zeros = 0
        self.repetition = 0
        #compare each element whether negative or zero or repetition
        for num in range(4, 16, 1):
            if (self.numbers[num] < 0):
                self.negatives += 1
```

```
            if (self.numbers[num] == 0):
                self.zeros += 1
            for comp in range(0, 4, 1):
                if (self.numbers[num] == self.numbers[comp]):
                    self.repetition += 1
            for comp in range(num + 1, 16, 1):
                if (self.numbers[num] == self.numbers[comp]):
                    self.repetition += 1
        self.magic = (self.repetition == 0) and (self.zeros <= 1) and (self.negatives==0)
        self.special = (2 * oneSquare.bb) == (oneSquare.dd + oneSquare.mm - oneSquare.cc + oneSquare.yy) \
                       and (self.magic)


        myspecial = numpy.array([self.magic, self.special, self.repetition, self.zeros, self.negatives], dtype=int)
        mynumbers = numpy.array(self.numbers, dtype=int)
        myarray = numpy.concatenate([mynumbers, myspecial])
        return myarray


# finding square algorithm
def find_squares(oneSquare:Square, SquareCollections):
    oneSquare.aa=0
    oneSquare.bb=0
    for oneSquare.aa in range(120):
        oneSquare.bb = 0
        for oneSquare.bb in range(50):
            oneSquare.n1 = oneSquare.cc + oneSquare.yy - oneSquare.aa
            oneSquare.n2 = -oneSquare.mm - oneSquare.yy + oneSquare.aa + 2 * oneSquare.bb
            oneSquare.n3 = oneSquare.dd + 2 * oneSquare.mm + oneSquare.yy - oneSquare.aa - 2 * oneSquare.bb
            oneSquare.n4 = oneSquare.dd + oneSquare.mm - oneSquare.bb
            oneSquare.n5 = oneSquare.mm + oneSquare.yy - oneSquare.bb
            oneSquare.n6 = -oneSquare.mm + oneSquare.cc + oneSquare.bb
            oneSquare.n7 = oneSquare.mm + oneSquare.cc + oneSquare.yy - oneSquare.aa - oneSquare.bb
            oneSquare.n8 = -oneSquare.mm + oneSquare.aa + oneSquare.bb
            oneSquare.n9 = oneSquare.dd + oneSquare.mm + oneSquare.yy - oneSquare.aa - oneSquare.bb
            oneSquare.n10 = -oneSquare.yy + oneSquare.aa + oneSquare.bb
            square = SquareCollection()
            myarray = square.toArray(oneSquare)

            SquareCollections = numpy.concatenate((SquareCollections, [myarray] ), axis=0)
    return SquareCollections

# unit testing routine
def test_squares(oneDate):
```

```python
    #initialize square
    myDate: Square = Square()
    # intialize the data set array
    data_store: numpy.ndarray = numpy.ndarray((0, 21), dtype=int)
    myDate.dd = oneDate.day
    myDate.mm = oneDate.month
    # trunc is to avoid decimals
    myDate.cc = numpy.trunc(oneDate.year / 100)
    myDate.yy = oneDate.year % 100
    data_store = find_squares(myDate,data_store)
    for sq in data_store:
        #17th location is the magic square found.
        if sq[16] == True:
            print(sq)
    print("Unit test passed!")

def daterange(start_date, end_date):
    for n in range(int ((end_date - start_date).days)):
        yield start_date + timedelta(n)

#Finding possible squares with various combinations
def find_squares_dates(start_date, end_date):
    #intialize the data set array
    data_store:numpy.ndarray = numpy.ndarray((0,21),dtype=int)

    for each_date in daterange(start_date, end_date):

        myDate:Square =Square()
        myDate.dd = each_date.day
        myDate.mm = each_date.month
        #trunc is to avoid decimals
        myDate.cc = numpy.trunc(each_date.year / 100)
        myDate.yy = each_date.year % 100

        data_store = find_squares(myDate, data_store)

    return data_store
# get random number
def get_random_number(high):
    return numpy.random.randint(0,high)

# unit test the data load for a date
test_date = date(2005, 10, 17)
test_squares( test_date )
```

```python
# for loading data Let us take first week of 2019
start_date = date(2019, 1, 1)
end_date = date(2019, 1, 7)

# load magic square data for weeks
dataset = find_squares_dates(start_date, end_date)

# range of the input columns in the dataset
xColumns = numpy.r_[0:5, 8:9]

# range of the output columns in the dataset
# let us only evaluate magic or special for testing pupose.
yColumns = numpy.r_[16:18]
# otherwise un comment the following
# yColumns =  numpy.r_[5:8, 9:21]

# split into input and output variables
X_input = dataset[:, xColumns]
Y_output = dataset[:, yColumns]


# seed is initialized to get same results
seed = 5
numpy.random.seed(seed)

# split the data into training (67%) and testing (33%)
(X_train, X_test, Y_train, Y_test) = train_test_split(X_input, Y_output, test_size=0.33, random_state=seed)

# create the model
model = Sequential()
# add the first hidden layer of input columns with 8 nodes, also called a Neuron or Perceptron
model.add(Dense(8, input_dim=xColumns.size,  activation='relu', kernel_initializer='uniform'))
# add the second hidden layer with 6 nodes
model.add(Dense(6,  activation='relu', kernel_initializer='uniform'))
# finally add output columns
model.add(Dense(yColumns.size,  activation='sigmoid', kernel_initializer='uniform'))

# compile the model
# loss function : maximum likelihood, error between 2 probability distributions measured using cross-entropy
# optimizer: The Adam optimization algorithm, an extension to stochastic gradient descent to increase the learning
rate
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```python
# fit the model
# training occurs over epochs(One pass through all rows in training dataset) and each epoch is split into batches
# verbose is to display the training sessions in console
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=5, verbose=1)

# finally evaluate the model
scores = model.evaluate(X_test, Y_test)
print("Accuracy: %.2f%%" % (scores[1]*100))

# initalize to randomly generate 100 dates
predict_date_variables: numpy.ndarray = numpy.ndarray((0, 6), dtype=int)
# randomly generate 100 dates
for myrandom in range(99):
    myRandArray = numpy.array([1+get_random_number(27), 1+get_random_number(11), 18+get_random_number(2),
                   1+get_random_number(98), 1+get_random_number(98), 1+get_random_number(98)],dtype=int)
    predict_date_variables = numpy.concatenate((predict_date_variables, [myRandArray]), axis=0)
# guess predictions for given days at a time
predictions = model.predict(predict_date_variables)
pointer = 0
for each_day in predict_date_variables:
    print("date: %d / %d / %d%d " %(each_day[0], each_day[1], each_day[2], each_day[3]))
    print("Magic Availability: %.2f%% Super Magic Availability: %.2f%%"
          %(predictions[pointer][0]*100, predictions[pointer][0]*100) )
```

The same principle can be extended to speedup:

1) Chat bots to find suitable answers from given questions' phrases
2) Prescription and medication based on patients' demographics, vitals and chief complaints
3) Prediction of agriculture yields, weather reports, stock markets, prices and so on…

I have also hosted an angular program at https://lksmangai.github.io/AngularBirthDate. You can find anyone's birth date's magic square over there.

My LinkedIn profile is available at https://www.linkedin.com/in/lakshmanarajsankaralingam/

For further information, please mail me at lksmangai@yahoo.com