RNG_solution

August 22, 2019

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

From Wikipedia https://en.wikipedia.org/wiki/Linear_congruential_generator

A linear congruential generator (LCG) is an algorithm that yields a sequence of pseudo-randomized numbers calculated with a discontinuous piecewise linear equation. The method represents one of the oldest and best-known pseudorandom number generator algorithms. The theory behind them is relatively easy to understand, and they are easily implemented and fast, especially on computer hardware which can provide modulo arithmetic by storage-bit truncation.

The generator is defined by recurrence relation:

$$X_{n+1} = (aX_n + c) \; mod \; m \tag{1}$$

where X is the sequence of pseudorandom values, and

$$\begin{aligned} m, 0 < m - the\ modulus & \quad (2) \\ a, 0 < a < m - the\ multiplier & \quad (3) \\ c, 0 \leqslant c < m - the\ increment & \quad (4) \\ X_0, 0 \leqslant X_0 < m - the\ seed & \quad (5) \end{aligned}$$

are integer constants that specify the generator.

```
In [2]: # helper function
        def int_to_bin_seq(x, num_digit):
            '''
            converts integer to binary sequence list
            of specified length by num_digit

            e.g. x=4, num_digit=3  to [1,0,0]
            e.g. x=4, num_digit=4  to [0,1,0,0]
            '''
            # convert int to binary string
            # e.g. 4 to '100'
            x_bin = '{0:b}'.format(x)
            # convert binary string to sequence of integers
            # e.g. '100' to [1, 0 ,0]
```

```
    x_bin_seq = list(map(int,x_bin))

    # add zeros in front if needed
    x_bin_seq = [0]*(num_digit-len(x_bin_seq)) + x_bin_seq

    return x_bin_seq
```

In [3]:
```
def lcg_generator(m, a, c, seq_len, seed = 'random'):
    '''
    For a given m and coeffs a and c,
    generates binary numbers of length (seq_len),
    based on linear congruential generator algorithm.
    '''
    # calculate the num of binary digits
    # necessary to represent nums.
    num_digit = int(np.ceil(np.log2(m)))

    # generate random seed
    if seed == 'random':
        x_init = np.random.randint(0,m)
    else:
        x_init = seed

    # convert int to binary sequence
    x_init_bin_seq = int_to_bin_seq(x_init, num_digit)

    # initialize sequence
    sequence = []
    sequence += x_init_bin_seq

    # initialize recurrence relation
    x = x_init

    while len(sequence) < seq_len:
        # recurrence relation
        x_next = (a*x + c) % m
        # convert int to binary sequence
        x_next_bin_seq = int_to_bin_seq(x_next, num_digit)
        # add this to sequence
        sequence += x_next_bin_seq
        # prepare for the following loop
        x = x_next

    # crop to fixed size
    sequence = sequence[0:seq_len]

    return sequence
```

In [4]:
```
RNs_training = lcg_generator(m=15, a=5, c=10, seq_len=2000, seed = 'random')
```

```
RNs_test = lcg_generator(m=15, a=5, c=10, seq_len=500, seed = 'random')
```

```
In [5]: def dataset_generator(RNs, inp_seq_len, step):
            X = []
            y = []
            i = 0
            while i+inp_seq_len+1<=len(RNs):
                X.append(RNs[i:i+inp_seq_len])
                y.append(RNs[i+inp_seq_len])
                i+=step
            return np.array(X), np.array(y)
```

```
In [6]: inp_seq_len = 50
        X_train, y_train = dataset_generator(RNs_training, inp_seq_len, step=1)
        X_test, y_test = dataset_generator(RNs_test, inp_seq_len, step=1)
```

```
In [7]: from keras.models import Sequential
        from keras.layers import Dense, Activation, Dropout
```

```
Using TensorFlow backend.
```

```
In [8]: model = Sequential()
        model.add(Dense(20, input_dim=inp_seq_len, activation='relu'))
        model.add(Dense(20, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                1020
_________________________________________________________________
dense_2 (Dense)              (None, 20)                420
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 21
=================================================================
Total params: 1,461
Trainable params: 1,461
Non-trainable params: 0
_________________________________________________________________
```

```
In [9]: from keras.optimizers import Adam
        opt = Adam()
        model.compile(optimizer=opt, loss='mse', metrics=['accuracy'])
```

```
In [10]: H = model.fit(X_train, y_train, batch_size=32, epochs=5, validation_data=(X_test, y_te
```

```
Train on 1950 samples, validate on 450 samples
Epoch 1/5
1950/1950 [==============================] - 0s 179us/step - loss: 0.1556 - acc: 0.8128 - val_l
Epoch 2/5
1950/1950 [==============================] - 0s 52us/step - loss: 0.0149 - acc: 1.0000 - val_lo
Epoch 3/5
1950/1950 [==============================] - 0s 55us/step - loss: 0.0019 - acc: 1.0000 - val_lo
Epoch 4/5
1950/1950 [==============================] - 0s 67us/step - loss: 5.4251e-04 - acc: 1.0000 - va
Epoch 5/5
1950/1950 [==============================] - 0s 80us/step - loss: 2.6720e-04 - acc: 1.0000 - va
```

In [11]:
```
f = open("JRNG.txt", "r") # open
JRNG = list(f.read()) # read
JRNG = list(map(int, JRNG)) # convert to integers
JRNG = np.array(JRNG) # convert list to np array
```

In [12]:
```
# check if it is only 0's and 1's
np.unique(JRNG)
```

Out[12]:
```
array([0, 1, 2, 4])
```

In [13]:
```
# delete non-0's and non-1's
JRNG = JRNG[JRNG != 2]
JRNG = JRNG[JRNG != 4]
np.unique(JRNG)
```

Out[13]:
```
array([0, 1])
```

In [14]:
```
inp_seq_len = 50
X_train, y_train = dataset_generator(JRNG[0:1500], inp_seq_len, step=1)
X_test, y_test = dataset_generator(JRNG[1500:], inp_seq_len, step=1)
```

In [15]:
```
model = Sequential()
model.add(Dense(20, input_dim=inp_seq_len, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 20)                1020
_________________________________________________________________
dense_5 (Dense)              (None, 20)                420
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 21
=================================================================
```

```
Total params: 1,461
Trainable params: 1,461
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```python
opt = Adam()
model.compile(optimizer=opt, loss='mse', metrics=['accuracy'])
```

In [17]:
```python
H = model.fit(X_train, y_train, batch_size=32, epochs=20, validation_data=(X_test, y_t
```

```
Train on 1450 samples, validate on 446 samples
Epoch 1/20
1450/1450 [==============================] - 0s 235us/step - loss: 0.2394 - acc: 0.5910 - val_l
Epoch 2/20
1450/1450 [==============================] - 0s 56us/step - loss: 0.2173 - acc: 0.6876 - val_lo
Epoch 3/20
1450/1450 [==============================] - 0s 76us/step - loss: 0.1977 - acc: 0.7297 - val_lo
Epoch 4/20
1450/1450 [==============================] - 0s 82us/step - loss: 0.1826 - acc: 0.7393 - val_lo
Epoch 5/20
1450/1450 [==============================] - 0s 60us/step - loss: 0.1719 - acc: 0.7510 - val_lo
Epoch 6/20
1450/1450 [==============================] - 0s 92us/step - loss: 0.1667 - acc: 0.7634 - val_lo
Epoch 7/20
1450/1450 [==============================] - 0s 92us/step - loss: 0.1604 - acc: 0.7697 - val_lo
Epoch 8/20
1450/1450 [==============================] - 0s 79us/step - loss: 0.1573 - acc: 0.7772 - val_lo
Epoch 9/20
1450/1450 [==============================] - 0s 69us/step - loss: 0.1564 - acc: 0.7793 - val_lo
Epoch 10/20
1450/1450 [==============================] - 0s 80us/step - loss: 0.1531 - acc: 0.7807 - val_lo
Epoch 11/20
1450/1450 [==============================] - 0s 110us/step - loss: 0.1506 - acc: 0.7903 - val_l
Epoch 12/20
1450/1450 [==============================] - 0s 88us/step - loss: 0.1489 - acc: 0.7869 - val_lo
Epoch 13/20
1450/1450 [==============================] - 0s 59us/step - loss: 0.1483 - acc: 0.7945 - val_lo
Epoch 14/20
1450/1450 [==============================] - 0s 71us/step - loss: 0.1466 - acc: 0.7959 - val_lo
Epoch 15/20
1450/1450 [==============================] - 0s 73us/step - loss: 0.1444 - acc: 0.8041 - val_lo
Epoch 16/20
1450/1450 [==============================] - 0s 68us/step - loss: 0.1430 - acc: 0.8014 - val_lo
Epoch 17/20
1450/1450 [==============================] - 0s 94us/step - loss: 0.1403 - acc: 0.8041 - val_lo
Epoch 18/20
1450/1450 [==============================] - 0s 85us/step - loss: 0.1400 - acc: 0.8097 - val_lo
```

```
Epoch 19/20
1450/1450 [==============================] - 0s 105us/step - loss: 0.1375 - acc: 0.8138 - val_l
Epoch 20/20
1450/1450 [==============================] - 0s 96us/step - loss: 0.1371 - acc: 0.8138 - val_lo
```