Today (11/15/15) I finally conceived of a versatile method for describing graphs mathematically, which is outlined below. While it seems simple, getting to this vantage point took a lot of traveling.

Originally I was operating with the layer paradigm but realized it was flawed. Instead, I create the following new paradigm. N is the set of all neurons and N' is the adjacency matrix in which are encoded the weights, if they exist, between each neuron in N. The weight of the connection between N_i and N_{i+n} can be represented by the function $Z(i,n)$. In this way, any graph can be generated and described mathematically.

For example,

The adjacency matrix for a square:

Let X be a directed graph of size l
X' is the adjacency matrix for graph X
$Z(i,n)$ is the weight of the connection from node X_i to node X_{i+n} in X.

$$0 \le i+n < l$$
$$Z(i,n) = \begin{cases} 1, & \text{if } n = 1 \lor n = l-1 \\ 0, & \text{otherwise} \end{cases}$$

$$X' = \begin{bmatrix} Z(0,0) & Z(0,1) & Z(0,2) & Z(0,3) \\ Z(1,-1) & Z(1,0) & Z(1,1) & Z(1,2) \\ Z(2,-2) & Z(2,-1) & Z(2,0) & Z(2,1) \\ Z(3,-3) & Z(3,-2) & Z(3,-1) & Z(3,0) \end{bmatrix}$$

Python code:

```
In[1]:

import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

def ngen(z, size):
    a = np.arange(size).reshape([1,size])
    b = np.transpose(a)
    Z = np.vectorize(lambda i,j: z(i, j-i))
    return Z(b,a)

def n_sides_matrix(l):
    z = lambda i,n: 1 if (n==1 or n==(l-1)) else 0
    return ngen(z, l)

G=nx.from_numpy_matrix(n_sides_matrix(4))
nx.draw(G)
```

```
plt.show()
```

Out[1]: