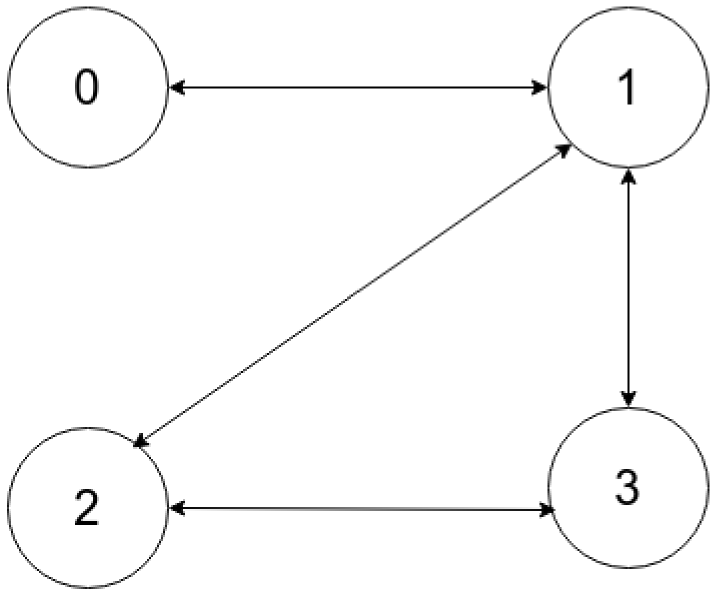
GCN Propagation Rule

In [1]:
```
import numpy as np
import scipy.sparse as sp
import networkx as nx
```

We will be showing how the propagation rule works on this graph:



Adjacency matrix $A \in \mathbb{R}^{NxN}$ where N is the number of nodes in a graph

In [2]:
```
A = np.matrix([
    [0,1,0,0],
    [1,0,1,1],
    [0,1,0,1],
    [0,1,1,0]],
    dtype=float
)

print(A)
```

```
[[0. 1. 0. 0.]
 [1. 0. 1. 1.]
 [0. 1. 0. 1.]
 [0. 1. 1. 0.]]
```

Feature matrix $X \in \mathbb{R}^{NxC}$ where C is the number of features per node

In [3]:
```
X = np.matrix([[i, -i] for i in range(A.shape[0])], dtype=float)
print(X)
```

```
[[ 0.  0.]
 [ 1. -1.]
 [ 2. -2.]
 [ 3. -3.]]
```

Multiplying A by X results in every node containing the sum of its neighbors features

In [4]:
```
Z = A * X
print(Z)
```

```
[[ 1. -1.]
 [ 5. -5.]
 [ 4. -4.]
 [ 3. -3.]]
```

Because A does not include self-loops, no node in Z contains its original data. To fix this, add an identity matrix to A, resulting in $\tilde{A}$

In [5]:
```
I = np.matrix(np.eye(A.shape[0]))
A_self = A + I
print(A_self)
```

```
[[1. 1. 0. 0.]
 [1. 1. 1. 1.]
 [0. 1. 1. 1.]
 [0. 1. 1. 1.]]
```

Now multiplying $\tilde{A}$ by X includes each node's original data

In [6]:
```
Z = A_self * X
print(Z)
```

```
[[ 1. -1.]
 [ 6. -6.]
 [ 6. -6.]
 [ 6. -6.]]
```

Instead of summing all of a node's neighbors features with its own, we want to generate an average of sorts. To do so, first we calculate the inverse square root of the diagonal degree matrix of A: $\tilde{D}^{-\frac{1}{2}}$

In [7]:
```
degrees = np.array(A_self.sum(1))
print("degree matrix")
diag_deg_mx = sp.diags(degrees.flatten()).todense()
print(diag_deg_mx)

deg_inv_sqrt = np.power(degrees, -0.5).flatten()
deg_inv_sqrt[np.isinf(deg_inv_sqrt)] = 0
diag_deg_mx_inv_sqrt = sp.diags(deg_inv_sqrt)
print("inverse square root degree matrix (D^-1/2)")
print(diag_deg_mx_inv_sqrt.todense())
```

```
degree matrix
[[2. 0. 0. 0.]
 [0. 4. 0. 0.]
 [0. 0. 3. 0.]
 [0. 0. 0. 3.]]
inverse square root degree matrix (D^-1/2)
[[0.70710678 0.         0.         0.        ]
 [0.         0.5        0.         0.        ]
 [0.         0.         0.57735027 0.        ]
 [0.         0.         0.         0.57735027]]
```

Then symmetrically normalize $\tilde{A}$:

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

In [8]:
```
sym_norm_A = diag_deg_mx_inv_sqrt * A_self * diag_deg_mx_inv_sqrt
print(sym_norm_A)
```

```
[[0.5        0.35355339 0.         0.        ]
 [0.35355339 0.25       0.28867513 0.28867513]
 [0.         0.28867513 0.33333333 0.33333333]
 [0.         0.28867513 0.33333333 0.33333333]]
```

Now multiplying $\hat{A}$ by X results in each node containing a weighted average of its and its neighbors features.

In [9]:
```
Z = sym_norm_A * X
print(Z)
```

```
[[ 0.35355339 -0.35355339]
 [ 1.69337567 -1.69337567]
 [ 1.9553418  -1.9553418 ]
 [ 1.9553418  -1.9553418 ]]
```