Image Convolution/Cross-correlation

In[47]:=
$$X = \begin{pmatrix} a & b & c \\ d & p & f \\ g & h & i \end{pmatrix}; \quad G = \begin{pmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{pmatrix}; \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Out[47]= {{0, 0, 1}, {0, 1, 0}, {1, 0, 0}}

Connection between cross-correlation and convolution

Cross-correlation $(X \star G)[1, 1] := \text{tr}(XG^{\intercal}) = a\, w_0 + b\, w_1 + c\, w_2 + d\, w_3 + p\, w_4 + f\, w_5 + g\, w_6 + h\, w_7 + i\, w_8$

Convolution : $(X * G)[1, 1] := (PXP \star G)[1, 1] =$
$i\, w_0 + h\, w_1 + g\, w_2 + f\, w_3 + p\, w_4 + d\, w_5 + c\, w_6 + b\, w_7 + a\, w_8$

In[49]:=
```
MatrixForm[P.X.P]
```

Out[49]//MatrixForm=
$$\begin{pmatrix} i & h & g \\ f & p & d \\ c & b & a \end{pmatrix}$$

Question for conv gnns: given X as the node features/embeddings and G as a learnable filter, how to calculate $X * G$?

Analogy between image Conv and graph Conv
*assume mix cross-correlation with convolution

Image

5x5x3 Image Im = $\begin{pmatrix} u_0 & u_1 & u_2 & \text{pixel} & \text{pixel} \\ u_3 & v_R & u_4 & \text{pixel} & \text{pixel} \\ u_5 & u_6 & u_7 & \text{pixel} & \text{pixel} \\ \text{pixel} & \text{pixel} & \text{pixel} & \text{pixel} & \text{pixel} \\ \text{pixel} & \text{pixel} & \text{pixel} & \text{pixel} & \text{pixel} \end{pmatrix}$, showing only first channel, second and third channel: $\begin{pmatrix} u_8 & u_9 & u_{10} \\ u_{11} & v_G & u_{12} \\ u_{13} & u_{14} & u_{15} \end{pmatrix} \begin{pmatrix} u_{16} & u_{17} & u_{18} \\ u_{19} & v_B & u_{20} \\ u_{21} & u_{22} & u_{23} \end{pmatrix}$, in total 27 pixel values

3x3x3 Filter G = $\begin{pmatrix} w_0 & w_1 & w_2 \\ w_3 & w_R & w_4 \\ w_5 & w_6 & w_7 \end{pmatrix}$, showing just first channel. Second channel and third channel:

$\begin{pmatrix} w_8 & w_9 & w_{10} \\ w_{11} & w_G & w_{12} \\ w_{13} & w_{14} & w_{15} \end{pmatrix} \begin{pmatrix} w_{16} & w_{17} & w_{18} \\ w_{19} & w_B & w_{20} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$, in total 27 weights

$(Im * G)[v]$ = inner product of SubTensor(0, 7) with G

$= u_0 w_0 + u_1 w_1 + u_2 w_2 + \ldots + u_{24} w_{26}$

im2col trick: If we write SubTensor(0, 7) as a vector $\vec{u}$, write G as a vector $\vec{w}$,

$= \vec{u} \cdot \vec{w}$

$\vec{u}$: all pixel values across channels of v's adjacent locations, adjacency is defined by the image layout and filter size

$\vec{w}$: all weights across channels of the filter

$$(\text{Im} * \text{G})[\text{v}] = \sum_{u_i \in \text{Neighbor}(v),\ \text{across channels}} u_i\, w_i + \sum_{c \in \{R,G,B\}} v_c\, w_c$$

Graph

A four-node graph

Embedding matrix H = $\begin{pmatrix} \overrightarrow{h_0} \\ \overrightarrow{h_1} \\ \overrightarrow{h_2} \\ \overrightarrow{h_3} \end{pmatrix}$, Nx1, each row is an embedding for a node, initially they equal to the input node features. Adjacency matrix A = $\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$, Filter matrix W = $\begin{pmatrix} \overrightarrow{w_0} \\ \overrightarrow{w_1} \\ \overrightarrow{w_2} \\ \overrightarrow{w_3} \end{pmatrix}$,Nx1, for syntax sugar, redefine W^A = $\begin{pmatrix} \overrightarrow{w_0} & \overrightarrow{w_1} & \overrightarrow{w_2} & \overrightarrow{w_3} \\ \overrightarrow{w_0} & \overrightarrow{w_1} & \overrightarrow{w_2} & \overrightarrow{w_3} \\ \overrightarrow{w_0} & \overrightarrow{w_1} & \overrightarrow{w_2} & \overrightarrow{w_3} \\ \overrightarrow{w_0} & \overrightarrow{w_1} & \overrightarrow{w_2} & \overrightarrow{w_3} \end{pmatrix}$, NxN,

What we want is filtered embedding at node v, how?

Let's say for node v, its idx is k

Vector form: $\overrightarrow{h_k}\grave{}$ = $\overrightarrow{h_k} \cdot \overrightarrow{w_{\text{center}}} + \sum_{i \in \text{Neighbor}(v)} \overrightarrow{h_i} \cdot \overrightarrow{w_i}$

Matrix form:

$A \odot W^A$ = $\begin{pmatrix} 0 & 0 & \overrightarrow{w_2} & 0 \\ 0 & 0 & 0 & \overrightarrow{w_3} \\ \overrightarrow{w_0} & \overrightarrow{w_1} & 0 & 0 \\ 0 & 0 & \overrightarrow{w_2} & 0 \end{pmatrix}$ NxN

$A \odot W^A$ H = $\begin{pmatrix} \overrightarrow{h_2}\, \overrightarrow{w_2} \\ \overrightarrow{h_3}\, \overrightarrow{w_3} \\ \overrightarrow{h_0}\, \overrightarrow{w_0} + \overrightarrow{h_1}\, \overrightarrow{w_1} \\ \overrightarrow{h_2}\, \overrightarrow{w_2} \end{pmatrix}$ = $\begin{pmatrix} \sum_{i \in \text{Neighbor}(v_0)} \overrightarrow{h_i} \cdot \overrightarrow{w_i} \\ \sum_{i \in \text{Neighbor}(v_1)} \overrightarrow{h_i} \cdot \overrightarrow{w_i} \\ \sum_{i \in \text{Neighbor}(v_2)} \overrightarrow{h_i} \cdot \overrightarrow{w_i} \\ \sum_{i \in \text{Neighbor}(v_2)} \overrightarrow{h_i} \cdot \overrightarrow{w_i} \end{pmatrix}$, Nx1, NxD

H` = activation(Θ H + $A \odot W^A$H) is what we finally want

Note 1: Obviously in the end there needs to be activation functions, skip connections can be added, residual blocks, etc.

Note 2: Embedding of first layer only contains the first-order neighbors of nodes, embedding of second layer will add information from the second-order neighbors (neighbors of neighbors).

Benefit: each row of calculation only depends on the local neighbors, so possible to parallelize and do mini-batch-style training via random walking, sampling, etc.

Alternatively, if we write H^D = $\begin{pmatrix} \overrightarrow{h_0} & 0 & \vec{0} & 0 \\ 0 & \overrightarrow{h_1} & 0 & 0 \\ 0 & 0 & \overrightarrow{h_2} & 0 \\ 0 & 0 & 0 & \overrightarrow{h_3} \end{pmatrix}$

then A$\odot W^A$H = A H^D W

Comparison to spectral based

UΛU^T = L = I - $D^{1/2}$A$D^{-1/2}$

H = $\begin{pmatrix} \overrightarrow{h_0} \\ \overrightarrow{h_1} \\ \overrightarrow{h_2} \\ \overrightarrow{h_3} \end{pmatrix}$, U = $\begin{pmatrix} u_i & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix}$, U^T, W = $\begin{pmatrix} \overrightarrow{w_0} \\ \overrightarrow{w_1} \\ \overrightarrow{w_2} \\ \overrightarrow{w_3} \end{pmatrix}$

H $*$ W = $F^{-1}(F(H * W))$

= $F^{-1}(F(H) \odot F(W))$

F(H) = U^T H, F^{-1}(H) = UH

=> H $*$ W := U($(U^T$ H) $\odot$ (U^T W)), this definition of convolution in graph, whereas in the spatial-based method we were defining the convolution with A, in analogy to image conv.

$$= \begin{pmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix} \left(\begin{pmatrix} \sum_{\text{weighted sum}} u_i \overrightarrow{h_i} \\ \sum_{\text{weighted sum}} u_i \overrightarrow{h_i} \\ \sum_{\text{weighted sum}} u_i \overrightarrow{h_i} \\ \sum_{\text{weighted sum}} u_i \overrightarrow{h_i} \end{pmatrix} \odot \begin{pmatrix} \sum_{\text{weighted sum}} u_i \overrightarrow{w_i} \\ \sum_{\text{weighted sum}} u_i \overrightarrow{w_i} \\ \sum_{\text{weighted sum}} u_i \overrightarrow{w_i} \\ \sum_{\text{weighted sum}} u_i \overrightarrow{w_i} \end{pmatrix} \right)$$

Calculations not limited to the first-order neighbors, it can figure out the hidden structure of the graph
As shown in GCN paper, if we take the first-order approximation of spectral-based method, we can recover the same spatial-based form above.
Problem: multiplication with U is inevitable and expensive, getting U in the first place may not be feasible, O(N^3) , and filter weights cannot be reused if new nodes introduced. No way to look only locally.