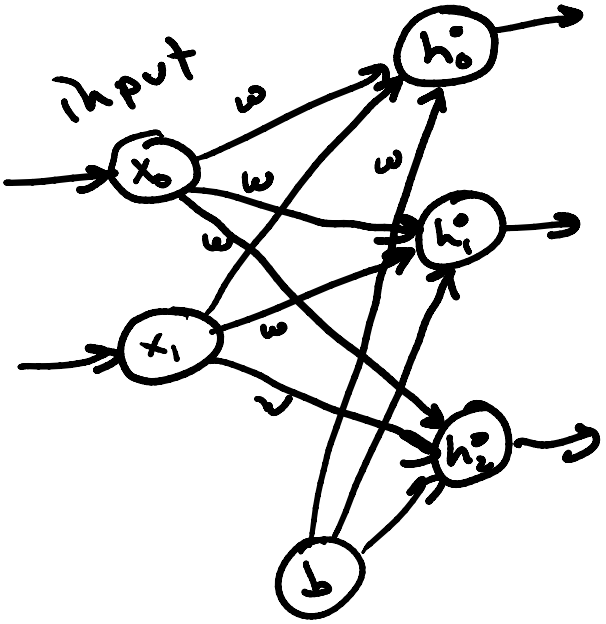
Neural networks

basic perceptron

nonlinear activation function

$$\vec{O} = \sigma(w\vec{i})$$

Hooking them up



$$h_n^k = \sigma\left(\sum \omega h^{k-1} + b^{k-1}\right)$$

Train → determine some loss

$$\text{loss} = \sum_{i \in \vec{O}} (O_i - t_i)$$

← over all training data

O_i: output from NN

t_i: expected training value

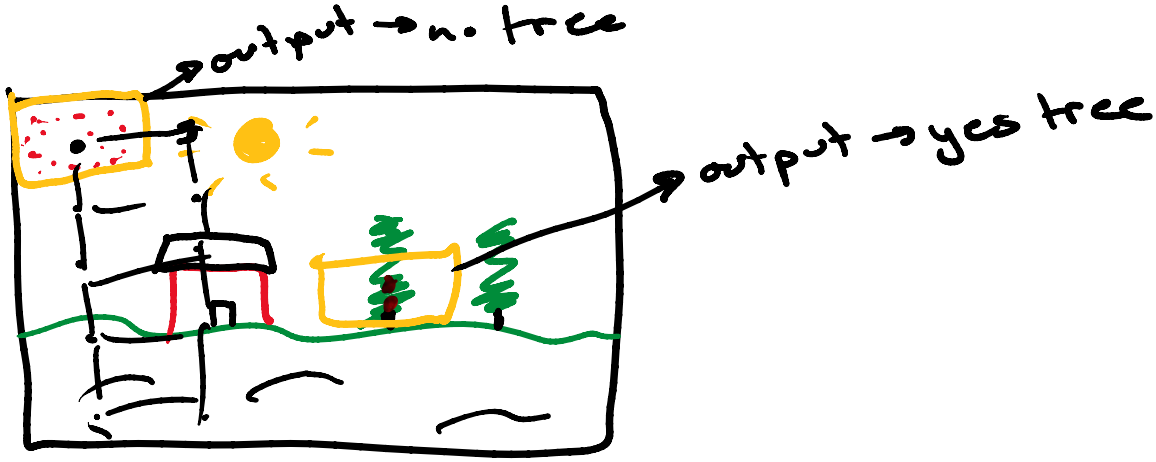
to actually train: gradient descent

=> updating weights and biases

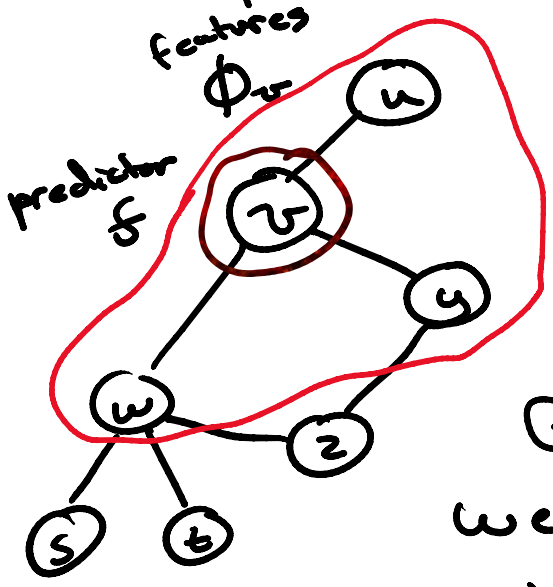
(backpropagation)

Convolutional neural networks

→ used for image processing



Graph Neural networks (GNN)



From a few classes ago:

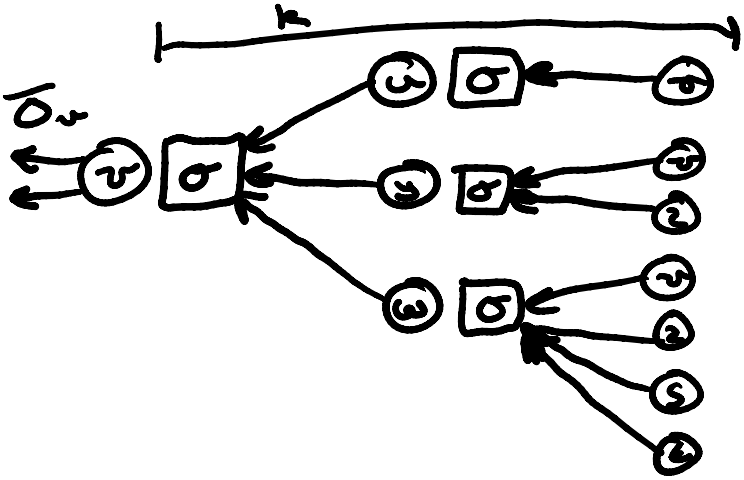
- we predicted some output on v, which was dependent on $N(v)$

GNN basic idea:

we propagate outputs/states along edges with vertices acting as neurons/NN.

acting as neurons/NNs

Note: we can consider the broader network around v to some k-hops



Formulation

$\bar{O}_v$ = output

$\bar{h}_v$ = hidden states

$\bar{x}_v$ = features

$\bar{x}_{co[v]}$ = edge features

$$\bar{h}_v = f(\bar{x}_v, \bar{x}_{co}, \bar{h}_{ne}, \bar{x}_{ne})$$

$$\bar{O}_v = g(\bar{h}_v, \bar{x}_v)$$

$\bar{h}_{ne[v]}$ = neighbor state

$\bar{x}_{ne[v]}$ = neighbor features

f = transition function
→ updates states

g = output function
→ updating output

General approach:

- consider global H, X matrices
- we iteratively update H

$$H^{t+1} = F(H^t, X)$$

$$H^{t+1} = F(H^t, X)$$

- supervised: $loss = \sum_{i=0}^{|\bar{v}|} (t_i - o_i)$

t_i = ith data in training set for v

o_i = ith output from GNN on v

To actually update:

we have some weight matrix W and possibly some bias matrix B

W, B are updated via gradient descent or similar

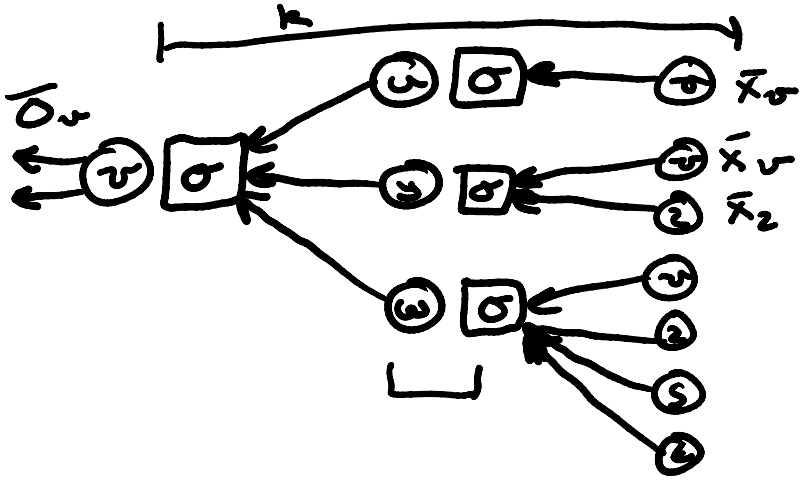
Note: $H^{t+1} = F(H^t, X)$

$F(H^t, X) \sim \sigma(H^t, X, W^t, B^t)$

To account for varying degrees

→ we can aggregate by averaging neighbor inputs (avg. degree)

→ Need to potentially account for this at every level



$\bar{h}_0 = \bar{x}_0$

$\bar{h}_k = \sigma(w_k \frac{\sum_{u \in N} h_u}{|N|} + B_k h_{k-1})$

$\bar{o} = \bar{h}_k$ (actual k)

Note: we train these

Notes:

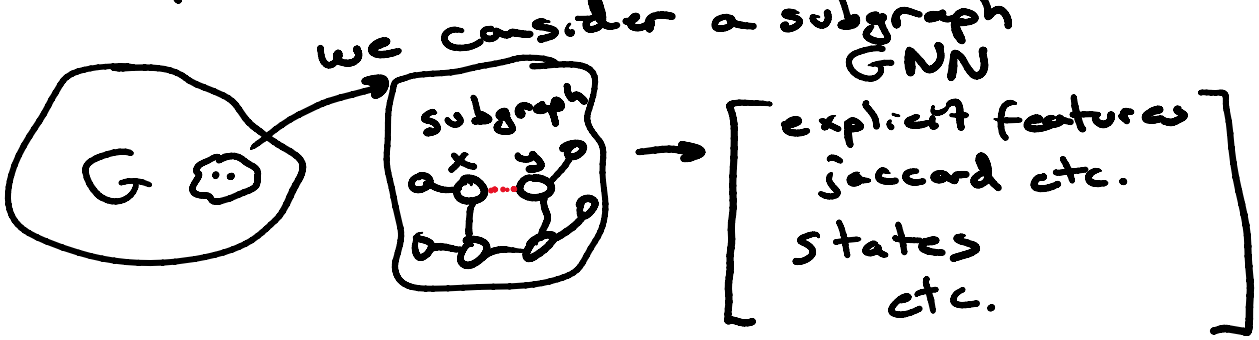
- W and B are consistent across the networks
- Aggregation can be done in any number of ways
- Learning for W, B is similar to B.P. on regular old NNs

Overall: similar power to NNs but we include the additional per-vertex edge-based

per-vertex edge-based computation graph

Applications
(almost everything we've talked about)

- Link prediction

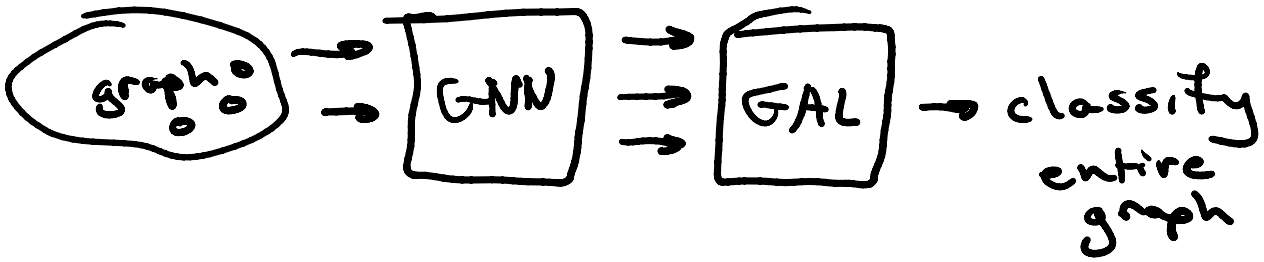


- Centrality
  → we can use (semi) supervised along with regression

- Clustering and C.D.
  → we can do classification with same set of "seed" vertices
  (more soon)

- graph measures
  (subgraph mining measures for classification)

→ Note: we can include an addition layer: aggregation layer



- graph alignment

  Use our aggregation layer to construct some "embedding"

  Recall: theoretically fastest isomorphism algorithms convert a graph topology to a string

  → similar idea

Code mode

Data: Karate network with 4 class labels

2 labeled, unlabeled

Problem: classification of unlabeled set of vertices

Approach: a GNN