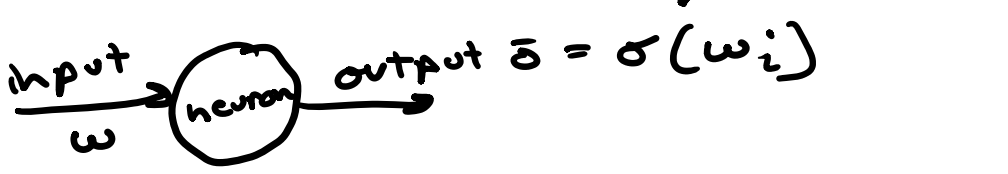
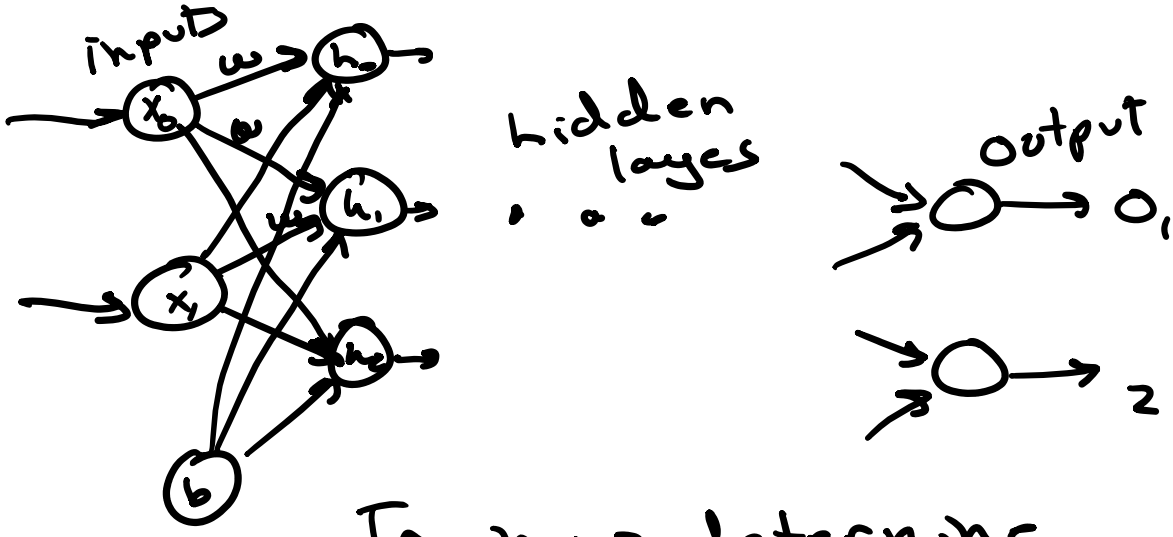
Neural networks

basic perceptron



Hooking them up into a networks



Train → determine some loss and minimize it

$$loss = \sum_{i=0} (O_i - t_i)$$

- i: over all training values
- O_i: output
- t_i: training value

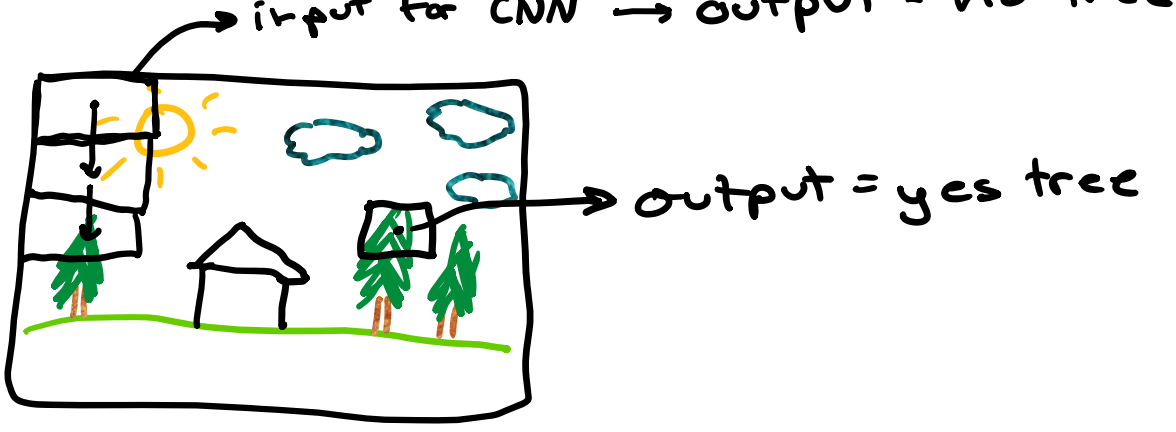
overall output value
training
values

To actually train:
gradient descent
of some kind

→ updating weights and biases
(back propagation)

Convolutional neural networks

↳ used for images



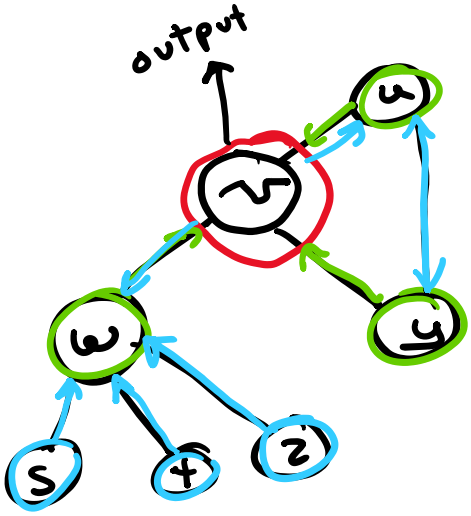
GCN → graph convolutional
networks

→ we move "inputs" across
the graph centered on vertices

the graph centered on vertices

Q1: How do we determine inputs/output for training?

Q2: How do we incorporate graph's topology?



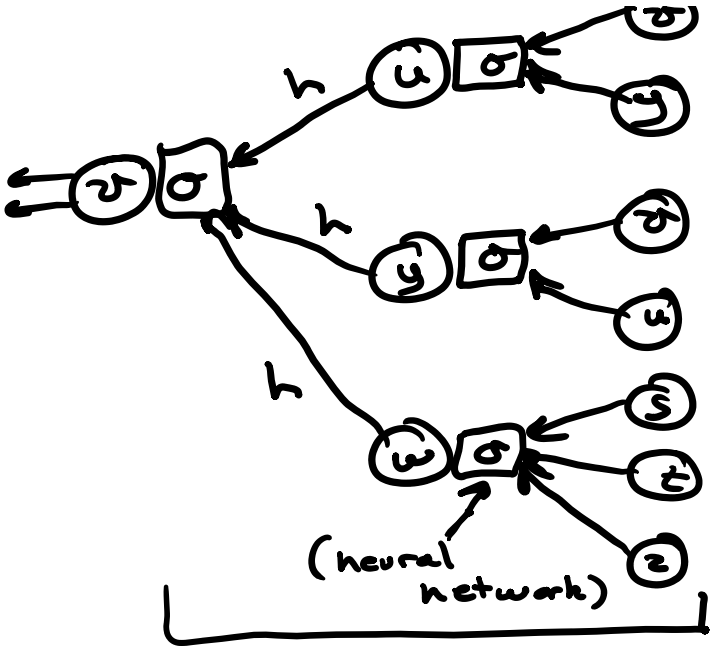
From our prior discussion

→ we can consider the "state" of a vertex as a function of its neighborhood

GCN idea: we propagate states/values and outputs along edges in the graph

→ consider the broader k-hop neighborhood around some vertex v

Formulation



k = # hops

Formulation

$\bar{O}_v$ = output

$\bar{h}_v$ = hidden states

$\bar{x}_v$ = features

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

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

f: transition function (update states)

$\bar{h}_{ne}, \bar{x}_{ne}$: neighbor states and features

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

g: output function (updates output)

General approach

we have global H, X

→ Iteratively update H

→ Iteratively update H

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

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

t_i = ith data in training set

o_i = ith output from GCN

→ Update weights / biases W, B via gradient descent

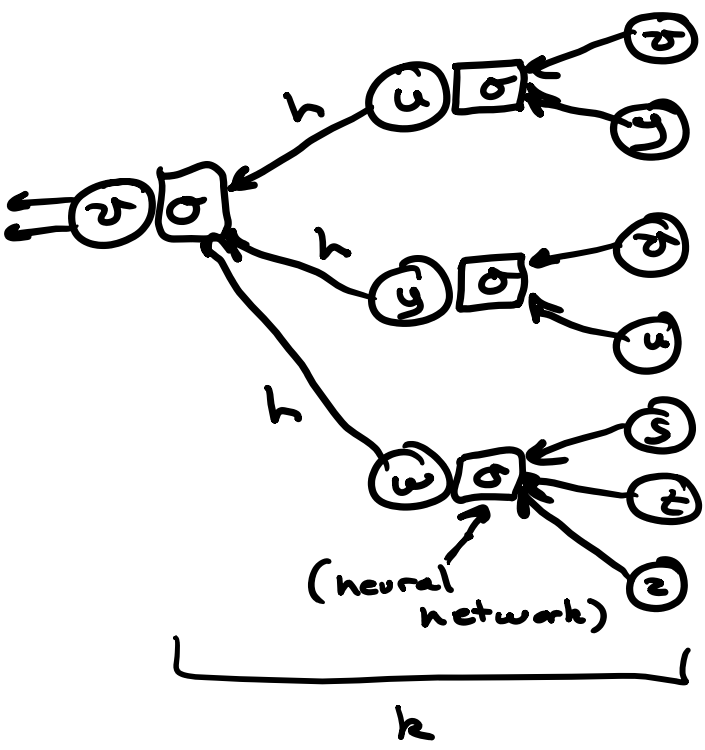
Note: $H^{t+1} = F(H^t, X)$

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

To account for varying degrees

→ aggregration via averaging or otherwise "reducing" our neighbor inputs

→ we need to do this at every level



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

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

$$\bar{o} = \bar{h}_k$$

↑ "avg." reduction

Notes:

- W and B are consistent across the whole network
- Aggregation can be done in any number of ways
- Learning for W, B is similar to training a regular NN

Overall: similar generalization and "learning power" of NNs, but we capture graph topology

---

Code Mode

We want to predict clusters on same graph

We have 4 class labels, to create 4 clusters ↑

Only 1 vertex labeled per class (semi-supervised)

Network: Zachary Karate Club

vertices: Karate students

edges: friendships