Lecture 10 - GNNs

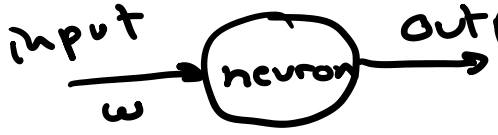
Tuesday, February 20, 2024 11:56 AM

Neural networks

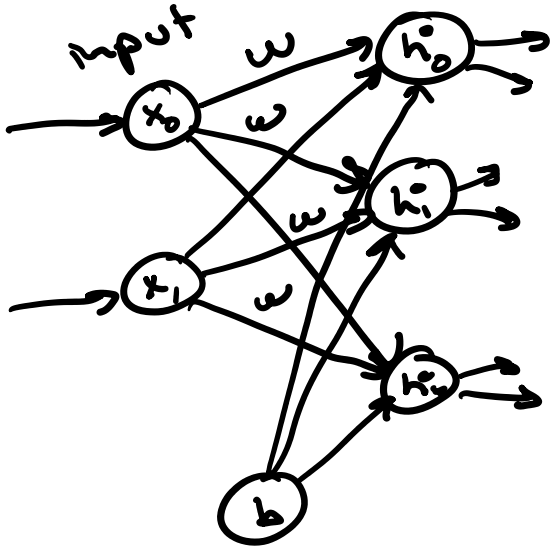
basic perceptron

nonlinear activation

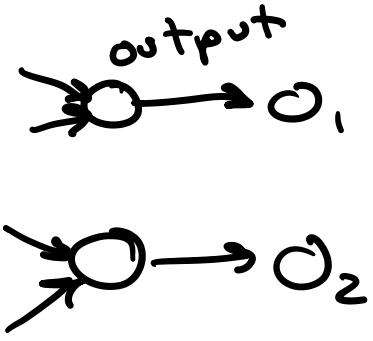
$$\text{output } \bar{o} = \sigma(w\bar{i})$$



Hooking them up



hidden layer ...



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

Train → determine some loss and minimize it

$$\text{loss} = \sum_{i \in \bar{O}} (o_i - t_i)$$

over all training data

o_i: output from NN

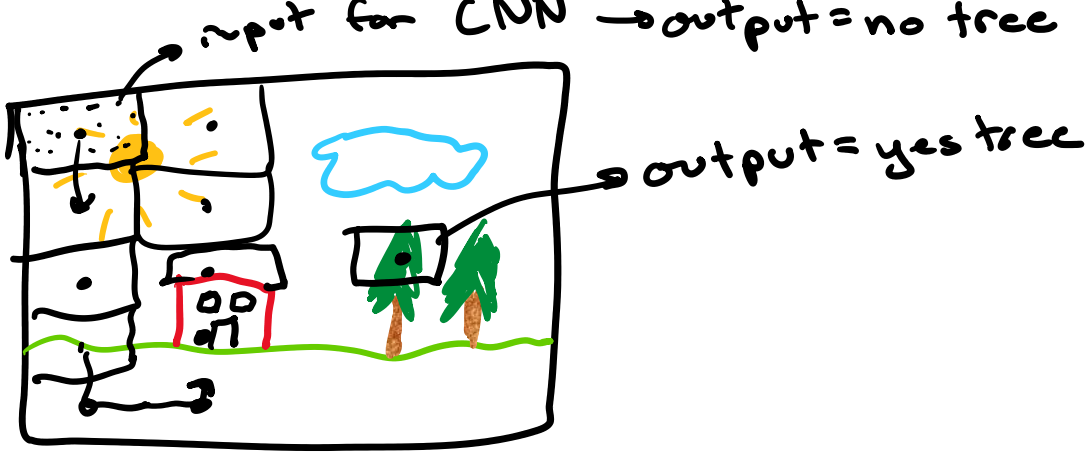
t_i: expected training value

To actually train: gradient descent

↳ updating weights and biases (back propagation)

Convolutional neural networks

→ used for image processing

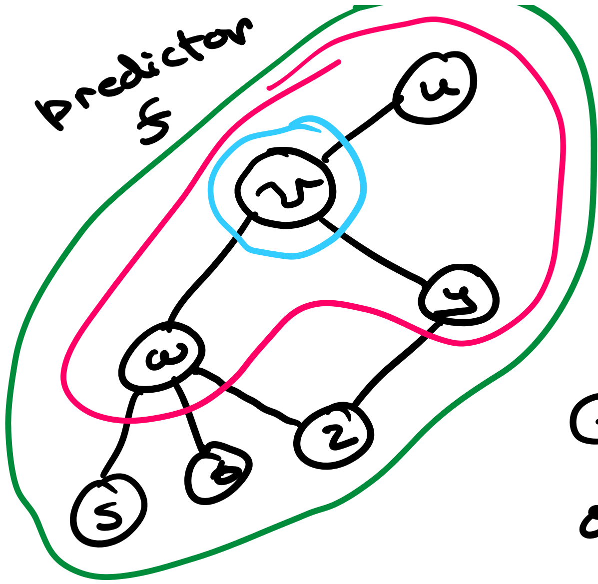


For GNNs → we move "inputs" across the graph centered on vertices

---

Graph Neural Networks (GNNs)

From our prior examples → we considered the

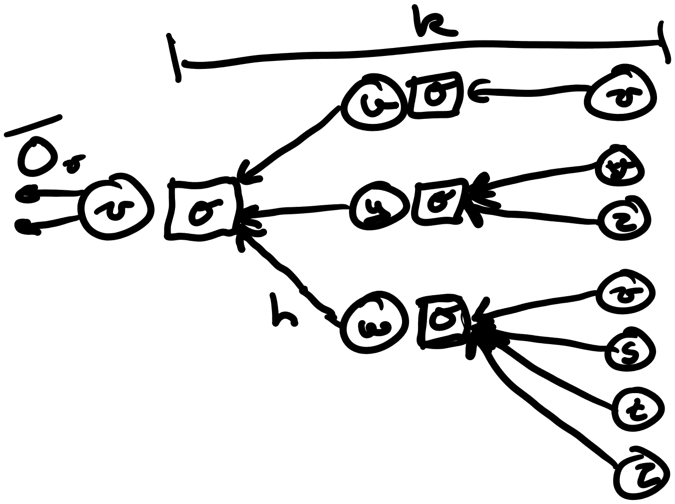


From our prior ...

→ we considered the state of some v as a function of its neighborhood

GNN idea: we propagate output/states along edges with vertices acting as neurons

→ we can consider the broader network around v to some k number of hops



Formulation

$\bar{O}_v$ = output

$\bar{h}_v$ = hidden states

$\bar{x}_v$ = features

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

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

↑ neighbor state ↑ neighbor features

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

f = transition function
→ updates states
g = output function
→ updates output

General Approach

Consider global H, X matrices

Iteratively update H

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

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

t_i = i^{th} data in training set

o_i = i^{th} output from the GNN

Update weights and biases W, B
→ via gradient descent

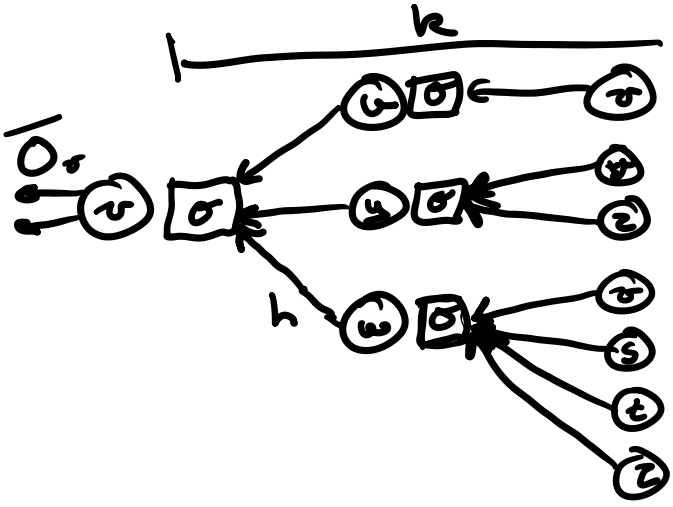
Note: $H^{t+1} = F(H^t, X)$

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

To account for varying degrees

We can aggregate by averaging or otherwise "reducing" our neighbor inputs

→ we'd need to account for this at every level



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

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

$\bar{O} = \bar{h}_K$

Note: we train these

Notes:

- W and B are consistent across the whole network
- Aggregation can be done in many different ways
- Learning for W, B is similar

- Learning for W, B is similar to training for a regular old NN

Overall: similar generalization and learning power NNs, but we also inherently capture the network topology

---

Code Mode

We want to predict clusters on some graph

We have 4 class labels, to predict 4 cluster

only one vertex labeled per class (semi-supervised)

Network: Zachary Karate Club

vertices = students in Karate class

edges = interactions outside of class

At some point in time => the class split in two

this split followed the graph topology