EECS 182/282A

Today: Graph Neural Nets

Prince Chapter 13

Other Resources: distill.pub/2021/gnn-intro

Packages: dgl.ai, PyG, Spektral

More info: STANFORD CS 224W

Architecture Order In Class:

MLPs → CNNs → Graph NN → RNN/state-space → Transformers

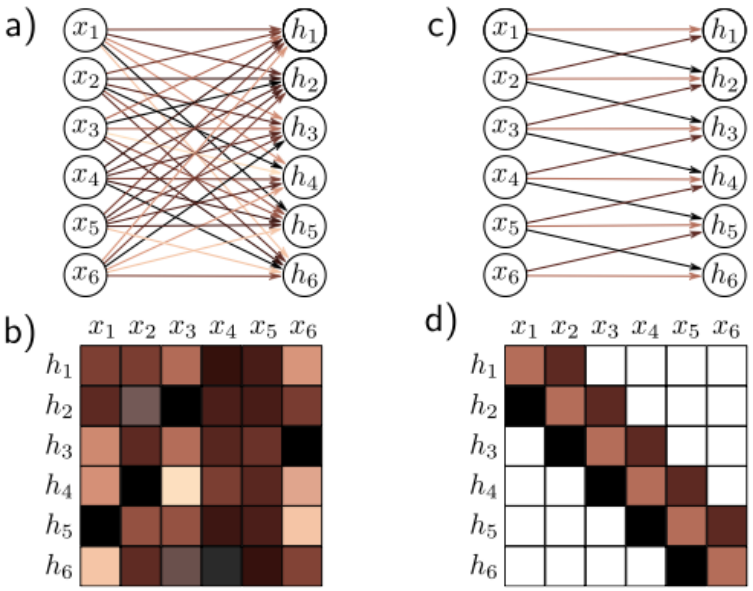
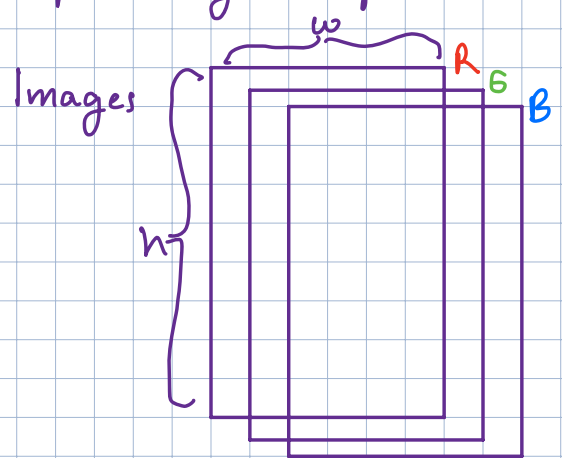


Fig 10.4 in Prince

MLP CNN

Inspired by computer vision problems: classification,



$h \times w \times 3$

channels c



semantic segmentation, etc.

Recall Concepts/Ideas from CNN module:

Weight-sharing to respect symmetries/invariances in domain

Convolutions — local processing

Combined with additional channels and depth/nonlinearities for expressivity

Data Augmentations

Dropout as an augmentation for MLP-like Elements in hidden layers

Normalizations for avoiding explosions in activations or gradients

Residual Connections for successful gradient flow to all layers

Downsampling/Pooling to get the big picture faster

Upsampling and U-Nets to restore appropriate detail at original scale

Generalizing CNNs: Images → General Graphs

Fig 13.2 in Prince

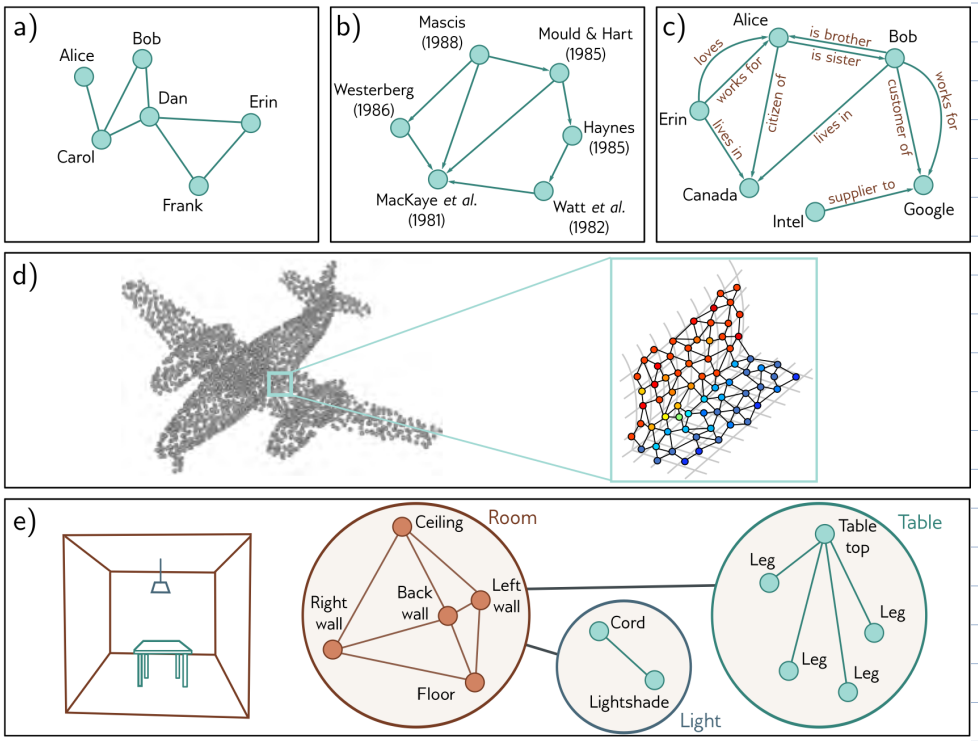
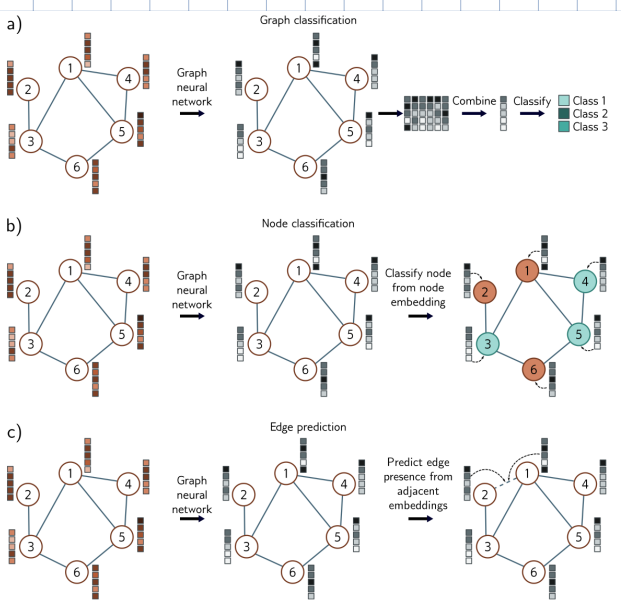


Image Classification



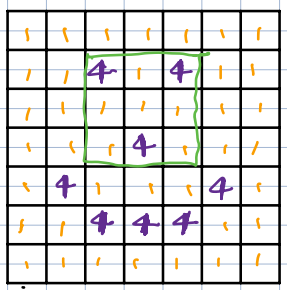
Semantic Segmentation

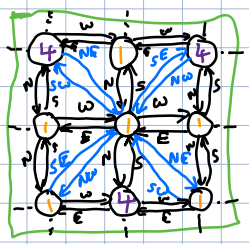
Fig 13.6 in Prince



First Conceptual Step: Image as a graph

Nodes = Pixels





Directed Graph

Edges labeled with one of $\{N, S, E, W, NE, SE, NW, SW\}$

(Always a special relation to self)

A 3×3 Conv can be viewed as receiving messages from neighbors, each of whom has their own labeled edge to me, multiplying by an edge-label-specific weight matrix, and adding those up together with a special self-specific matrix acting on this node's info and a bias $\vec{b}$.

Note: Implicit Zero-Padding and Same-Pad
Weight-sharing across nodes
1×1 Convs are local to a node

Matrices W_i and biases $\vec{b}$ are specific to layer of processing

Recall Picture:

3×3 example:

$$\begin{bmatrix} W_{NW} & W_N & W_{NE} \\ W_W & W_{mc} & W_E \\ W_{SW} & W_S & W_{SE} \end{bmatrix} \quad c_{in} \overbrace{\{ W_i, \vec{b} \}}^{} \; c_{out}$$

$$\vec{h}_{out} = \vec{b} + \sum_{i \in \text{neighbors labels}} W_i \vec{h}_{in,i}$$

Taking Stock: We have, in principle, an immediate recipe to generalize CNNs to directed graphs with the same property: Every edge labeled from a discrete finite set.

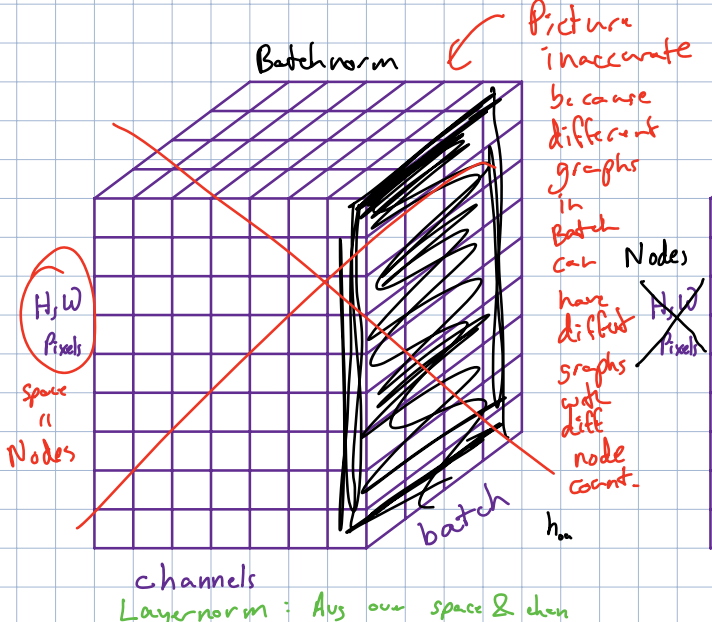
What Generalizes Immediately: Weight-sharing (Nodes do the same things)
3×3 convs (As above, Matrices for edge-labels)
Additional Channels, Residual Connections, Dropout for MLPs, Stochastic Depth

What does not generalize: Pooling / Downsampling and Upsampling

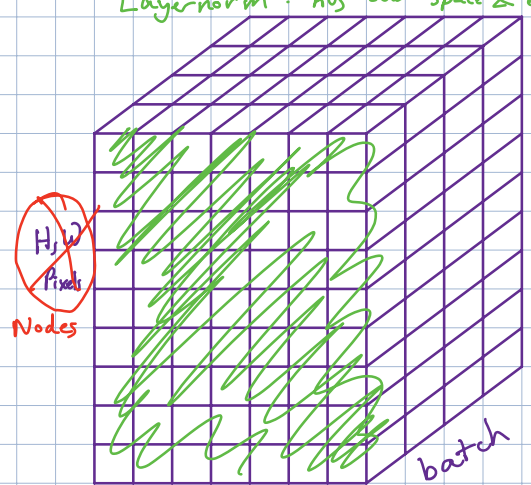
What needs discussion: Normalization.

Core Insight: The graph plays the role of the empty grid of pixels in CNNs.
The input is a graph with vectors in it

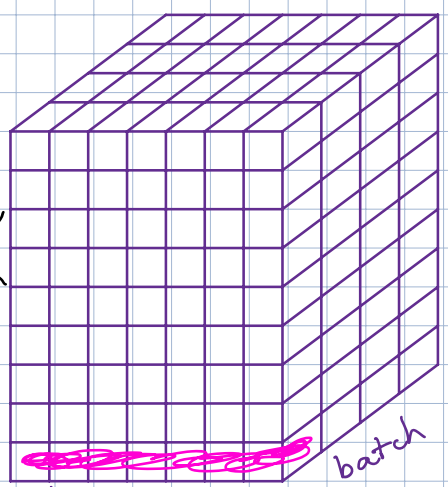
Normalization Layers in GNNs



Layernorm: Avg over space & chan



Can do this



Easy Choice: Just per-embedding normalization

Same applied at each node

Has weights that are learned γ, β

scale, shift

So are we done? No.

Kinds of Graphs:

	Edge Labels (finite)	Edge Labels (Infinite)	None
Directed	✓		
Undirected			???

First Attempt: Reduce to already understood case.

1) Treat each undirected edge as both directions ↔ replaces —
2) Pick a single dummy label for every edge.

What happens? Conv: $\vec{b} + \sum_i W \vec{h}_i = \vec{b} + W \sum_i \vec{h}_i$

Step back: What do we know?

a) Me — my own node information is special h_0
b) How many neighbors I have n
c) A set $\{h_1, h_2, \ldots, h_n\}$ of info from neighbors

Architectural Constraints: I can use n and h_0 specially
But must be permutation-invariant to $h_1, \ldots, h_n$

So for each output channel, I can compute:

e.g. $f_{W_1}(\vec{h}_0, \sum_{i=1}^{n} \vec{g}_{W_2}(\vec{h}_i))$ with learnable parameters W_1, W_2

← Can be MLP

Other choices for Σ: Average (divide by n)
RMS Norm (and other norms)
Max, Min,
(Compose with vector-wise operations)
Random Sample
Multiply

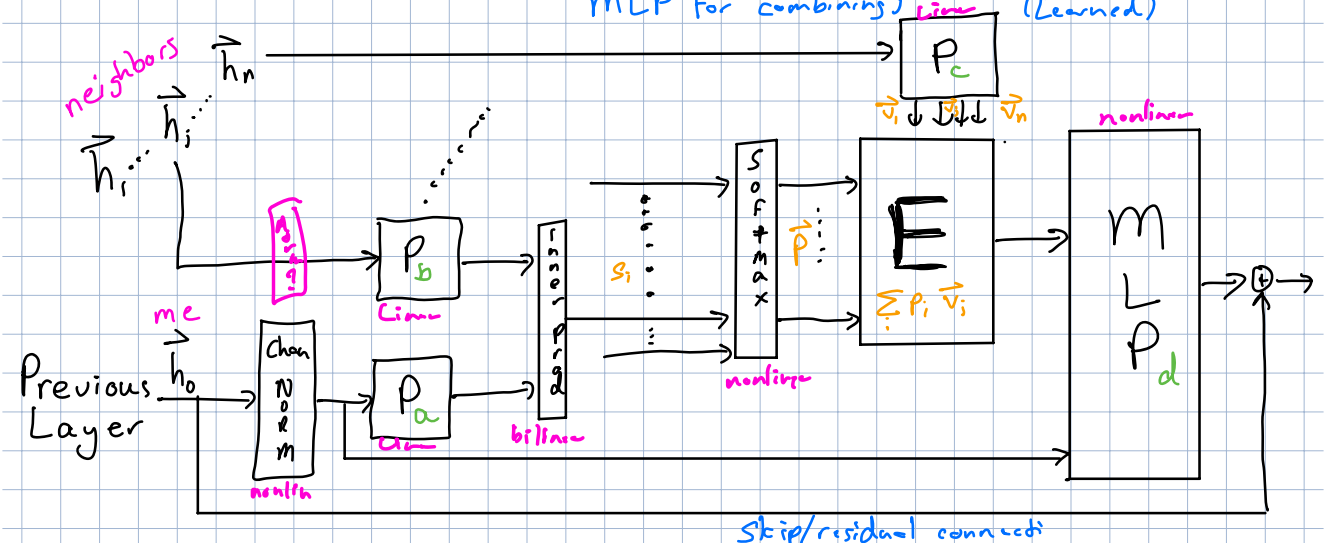
Any ML Approach: e.g. Least Squares
Nearest Neighbor
Logistic Regression
⋮

More generally: $f_{w_1}(\vec{h}_0, \sum \vec{g}_{w_2}(\vec{h}_0, \vec{h}_i))$

or with more structure $f_{w_1}(\vec{h}_0, \sum s_{w_2}(\vec{h}_c, \vec{h}_i)\, \vec{g}_{w_3}(\vec{h}_i))$

Natural Example: GNN Building Block

Design Choices: Linear for g (Learned)
Inner Product For Similarity s (Learned)
Expectation-style for "sum" (Learned Linear Function)
MLP for combining f_{linear} (Learned)



Question: What's nonlinear here?

Nonlinearity is a prereq for avoiding oversmoothing when we have no edge labels.

Note: This approach is related to attention mechanisms in Transformers but the GNN motivation is from minimal "what's possible" considerations.

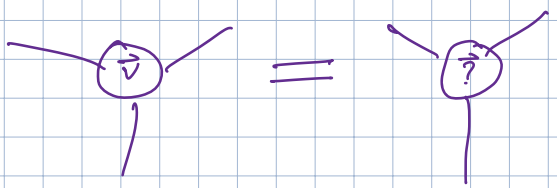
In fact Transformers can be usefully thought of as GNNs on a complete graph.

Challenge: Held-Out Data (Node-level Problems)

Approach 1: Excise entire nodes from the graph.

Challenge: Impacts connectivity

Approach 2: Remove content from the node but leave it in.



$\vec{?}$ = learnable vector usually called $\vec{\text{mask}}$

Note:

By making $\vec{?}$ learnable, we need to make sure the training examples seen during training let that happen.

1) It shouldn't impede information flow as messages are passed. ← This will happen even with the $\vec{?}$ in trails held-out positions during training.

2) It should prompt the network to correctly predict labels for the node even though there isn't input there.

To get (2) to happen, it is useful to do data augmentation during training. Randomly mask some nodes for which we have target labels, and keep the loss on that node's predictions.

What does not generalize: Pooling / Downsampling and Upsampling

↑
Key Challenge

Why do we pool anyway? — Grow Receptive Fields Faster (with layers)

Hack for GNNs: Add a fake global node connected to everything.

Aside: "oversquashing" — information from far away lost because of small embeddings.
vs "oversmoothing" — the operations of the GNN naturally blur away all fine-grained details

Is there a way to go: Many → Few based on vector contents?

Like Clustering!

Recall Classic Clustering Approach: k-means / Lloyd's Algorithm for VQ

Hyperparam Choice: k

0) Start at random in some way

1) Calculate Centers

2) Map points to centers to get clusters

3) Goto 2 & Repeat

Is this differentiable?

What's the barrier? Hard Assignments of points to clusters

DiffPool Idea: When we downsample, we get a smaller graph but it's fully connected with probabilities on each edge

[Good Docs in PyG]

Hierarchical Graph Representation Learning with Differentiable Pooling by Ying, et al. 2019

Make each point belong to every cluster, just with some probability.

"Grey-box" the problem: Inputs: Graph with Embedded Vectors

Output: k-length vector for each node with probabilities in it

i.e. $s_i^T = [p_1, \ldots, p_k]$ for node i

Need differentiable operation that does this.

Ingredients? 1) Softmax on scores will give me

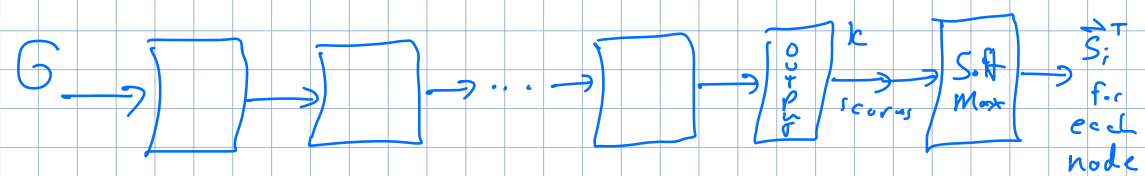
New Problem: Given a graph, do some differentiable steps to get scores.

Don't know what to do. Make it learnable.

What's a generic way to have a learnable algorithm on a graph?

USE a GNN!

Wherever we need to "downsample" a graph insert a little GNN that doesn't share weights with the rest of the GNN and has the structure:



But these $\vec{s}_i^T$ aren't a graph... Not yet.

What do we do with these soft assignments to clusters?

Need New Vectors (Embeddings) for the k clusters.

$$\vec{h}_j^{(l+1)} = \sum_i s_i[j]\, \vec{h}_i^{(l)}$$

j^{th} cluster in layer $l+1$ (After Pooling)

i: Pre-Pooling nodes

$s_i[j]$: "Probability" node i belongs to cluster j

The content of original node flows into different cluster according to membership

Need New (Soft) Adjacency Matrix

$N\{A^{(l)}$ ← original adjacency (symmetric matrix)

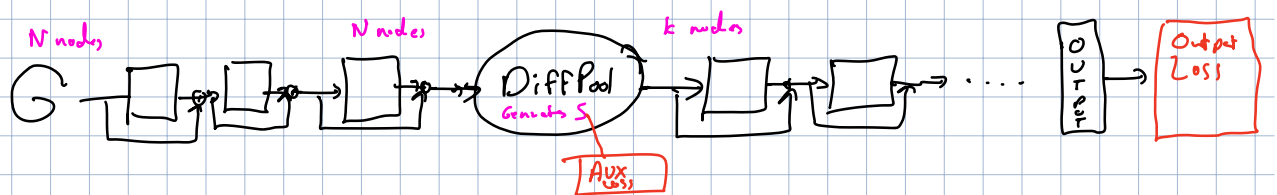
$$\underbrace{A^{(l+1)}}_{k} = \underbrace{S^T}_{N}\, A^{(l)}\, \underbrace{S}_{k}$$

where $S = \begin{bmatrix} s_1^T \\ s_2^T \\ \vdots \\ s_N^T \end{bmatrix}\} N$ (k columns)

So $A_{i,j}$ represents how strong the edge is between cluster i and cluster j

Now we have a new graph! With k nodes and a new Adjacency Matrix

Put things together



Can Train end-to-end. But we can do better.

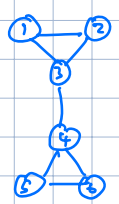
Auxillary Losses can help get desirable properties.

Note: When passing activations across one of these edges, can multiply by A_{ij} to account for the strength of the edge. [Important therefore to not immediately normalize these messages]

Property 1: Have clustering respect graph topology

$$\| A - {}^{N}S^{k}\,{}^{k}S^{N\,T} \|_F$$

What kind of information does this encourage in the embeddings?



$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

${}^{N}S^{k}\,{}^{k}S^{N\,T}$ has low-rank structure

That's what we want from the example here

Note: The entries in SS^T are pairwise inner products of the $\vec{S}_i^T$ with $\vec{S}_j^T$.

Property 2: Want to be close to hard cluster assignments

$$\frac{1}{N}\sum_i H(\vec{S}_i^T)$$

Probabilities

Entropy minimized by $[0, 0, \ldots, 1, 0, \ldots 0]$

Note: Hyperparameters: weights for Aux losses. Follow what others do...

	Edge Labels (finite)	Edge Labels (Infinite)	None
Directed	✓ Like CNN		
Undirected		DiffPool results in weights on edges (in particular scalars)	✓ without Pooling