EECS 182/282A

Today: Graph Neural Nets

Reading: Prince through Ch 11 + Ch 13

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

Packages: dgl.ai, PyG, Spektral

More info: STANFORD CS 224W

Fri TA OH 531 Cory 2-3PM

Architecture Order In Class:

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

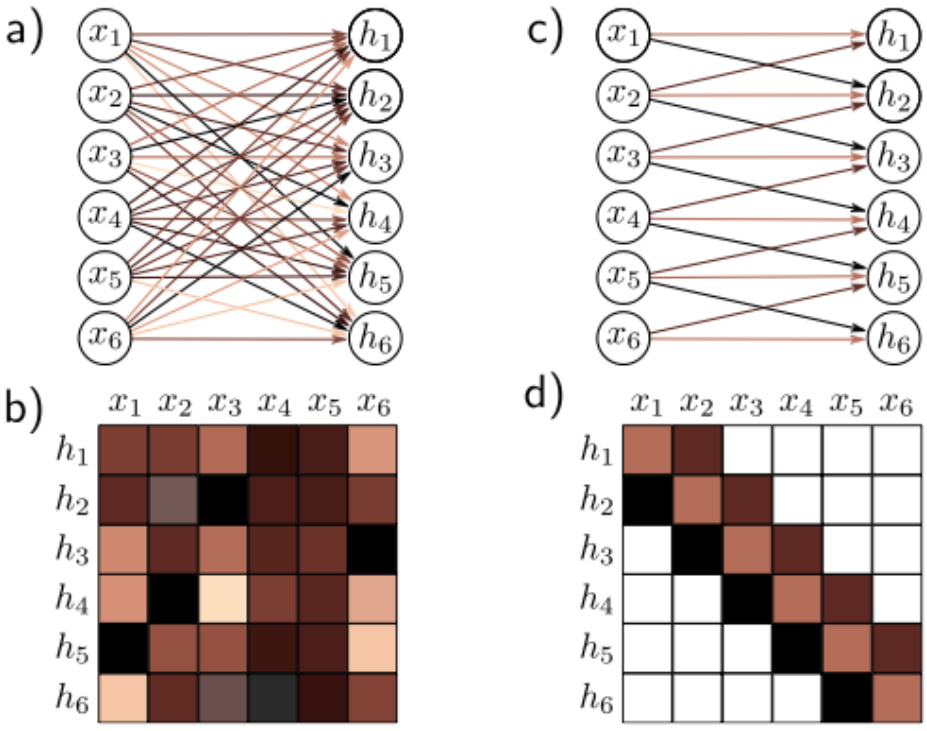
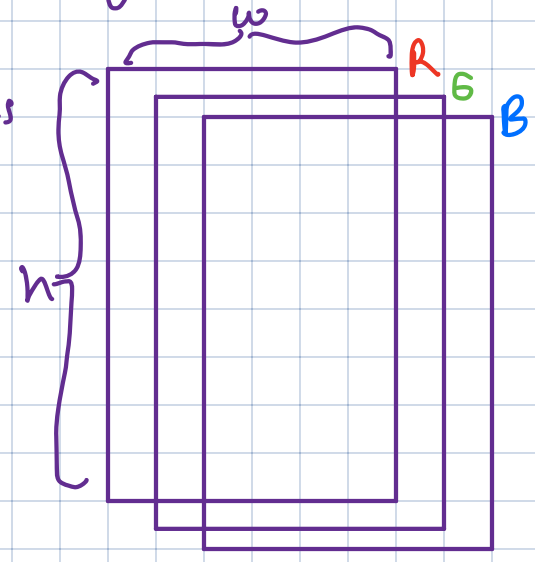


Fig 10.4 in Prince

MLP CNN

Inspired by computer vision problems: classification,



Images

$h \times w \times 3$ 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

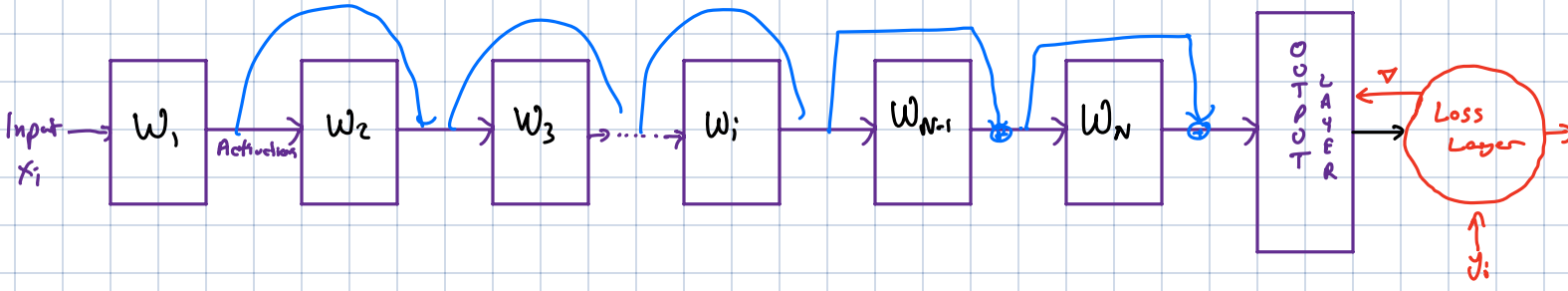
Residual Connections for successful gradient flow to all layers

Downsampling/Pooling to get the big picture faster

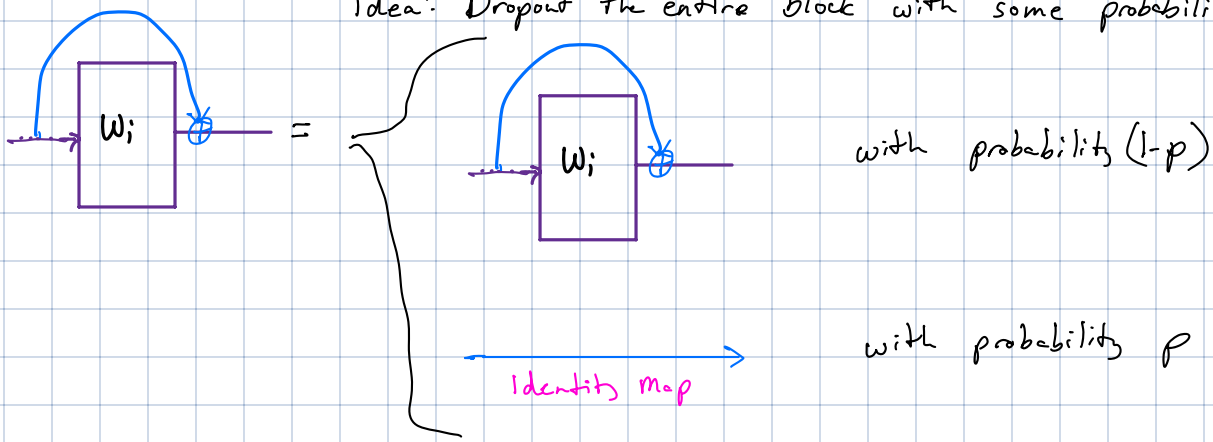
Upsampling and U-Nets to restore appropriate detail at original scale

---

One loose end: stochastic depth regularization



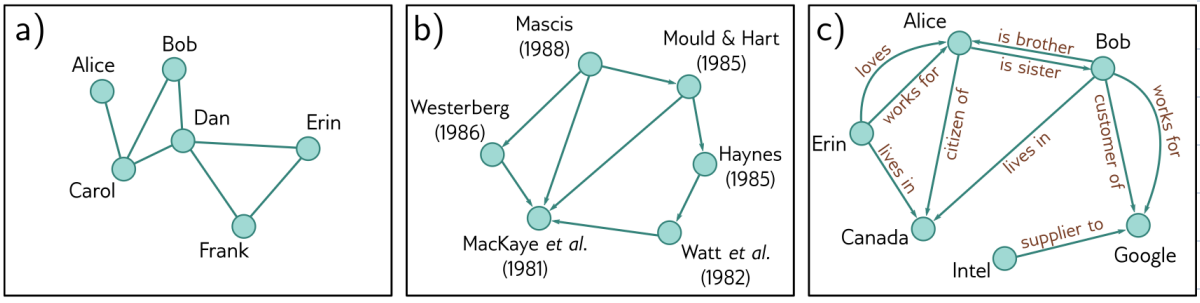
Idea: Dropout the entire block with some probability

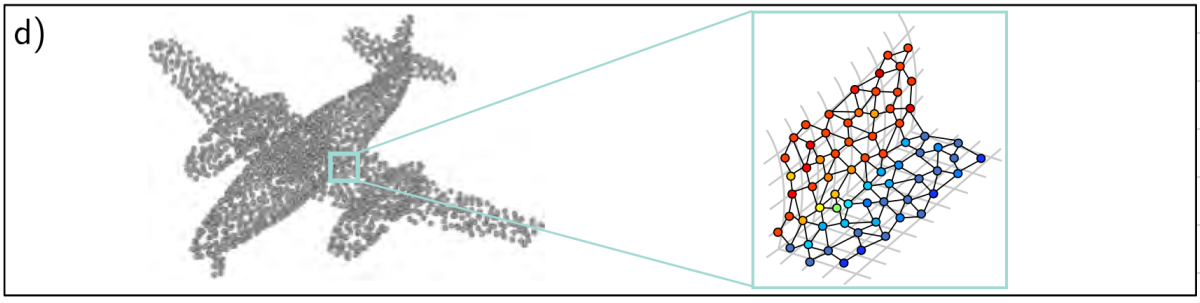


Like Dropout, helps with overfitting and creating functional redundancy (across blocks)

Note: Unlike Dropout, don't need to adjust anything at inference/test time. ← Added in O.H.
(Because of Identity Map & Presence of Normalizations inside blocks)

Generalizing CNNs: Images → General Graphs





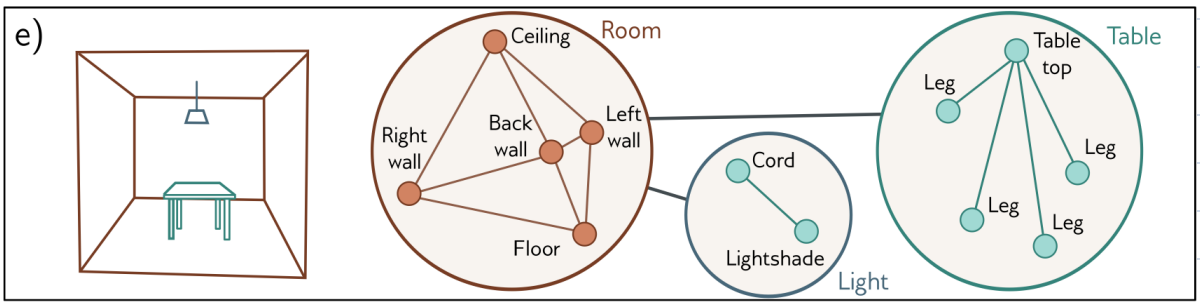


Fig 13.2 in Prince

Image Classification

Semantic Segmentation

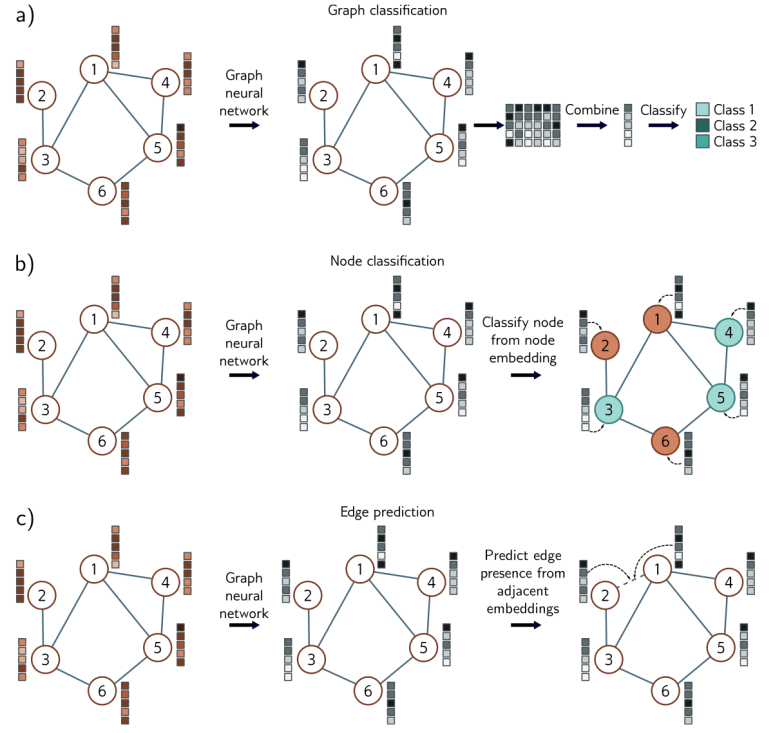
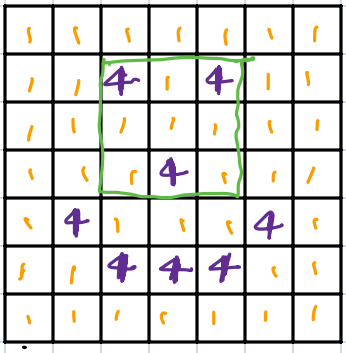
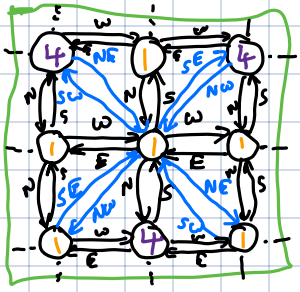


Fig 13.6 in Prince

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 3x3 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

1x1 Convs are local to a node

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

Recall Picture:

3x3 example:

W_{NW}	W_N	W_{NE}
W_W	W_{me}	W_E
W_{SW}	W_S	W_{SE}

C_{out} $\{ \overbrace{W_i}^{C_{in}}, \vec{b} \}$ C_{out}

$$\vec{h}_{out} = \vec{b} + \sum_i W_i \vec{h}_{in,i}$$

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)

3x3 convs (As above, Matrices for edge-labels)

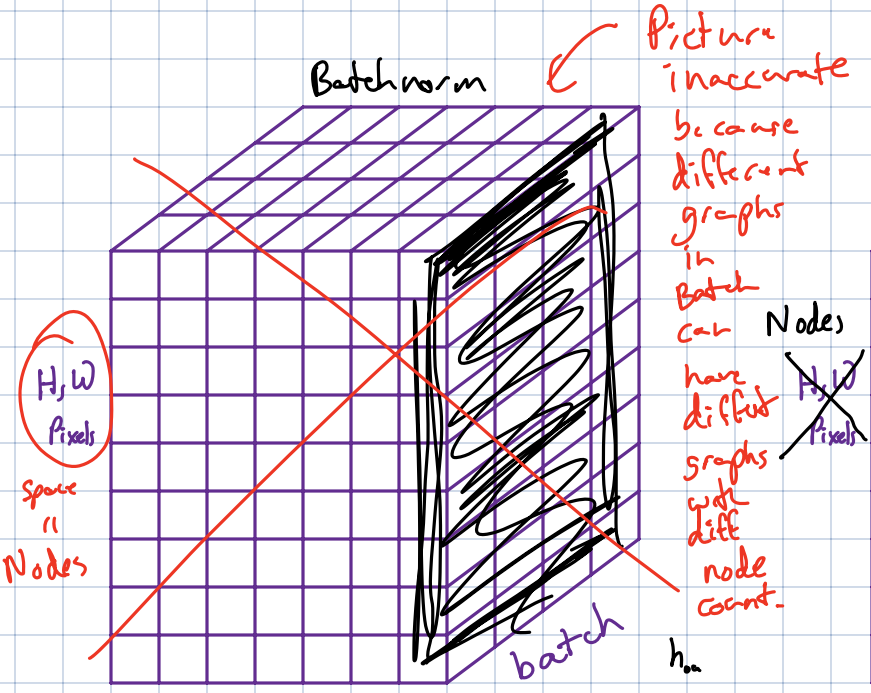
Additional Channels, Residual Connections, Dropout for MLPs, Stochastic Depth,

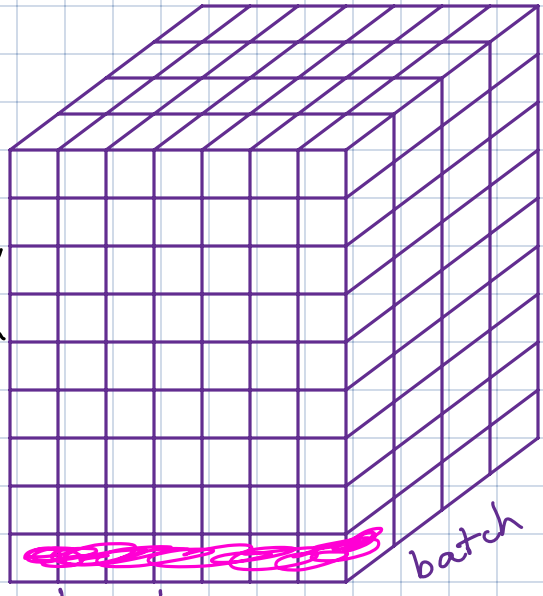
What does not generalize: Pooling / Downsampling and Upsampling

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





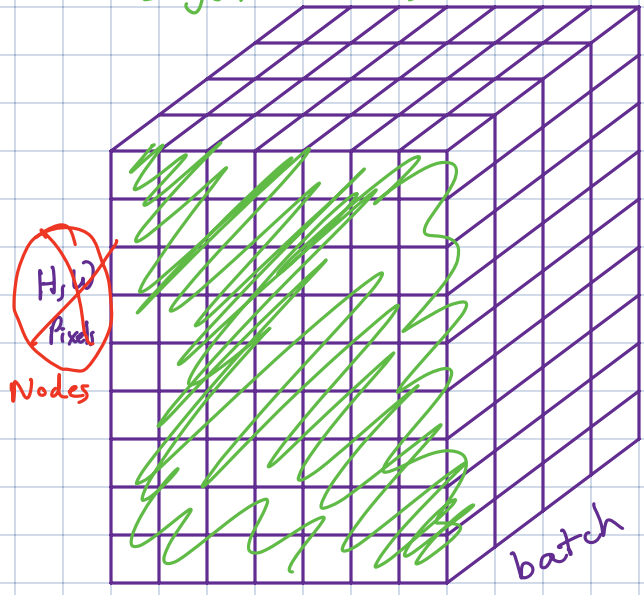
Layernorm: Avg over space & chan

Easy Choice: Just per-embedding normalization

Same applied at each node

↳ Weight-share any learned γ, δ

γ: scale, δ: shift



Can do this

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

Other choices for Σ: $\prod$ multiplication

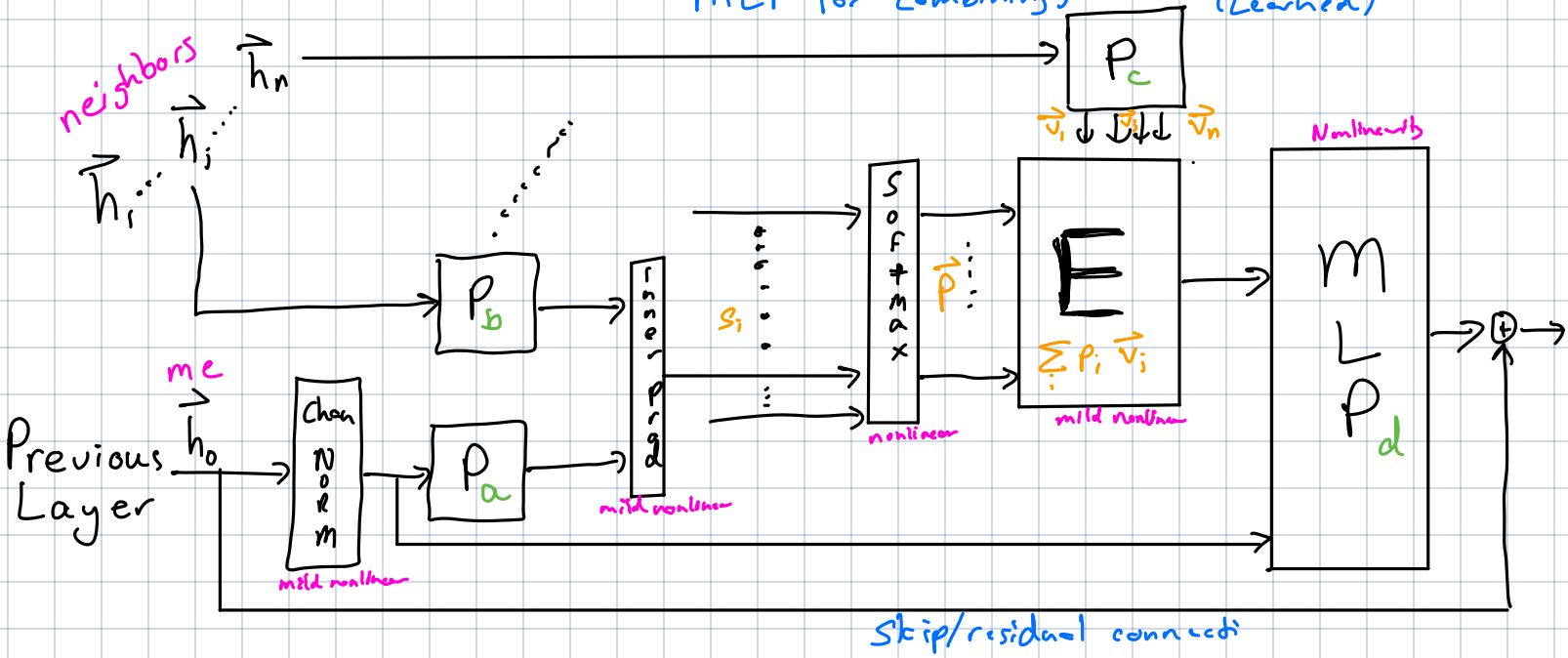
max, min, median (General order-statistics) + mode
Norm (right kind)
Average, Variance
Count Unique
Correlations with self
Least Squares ← Huge Family, Think this down.
Softmax

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

or with more structure $f_{w_1}(\vec{h}_0, \underbrace{\sum s_{w_2}(\vec{h}_c, \vec{h}_i)\, \vec{g}_{w_3}(\vec{h}_i)}_{\text{Should this be an expectation?}})$

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 (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.