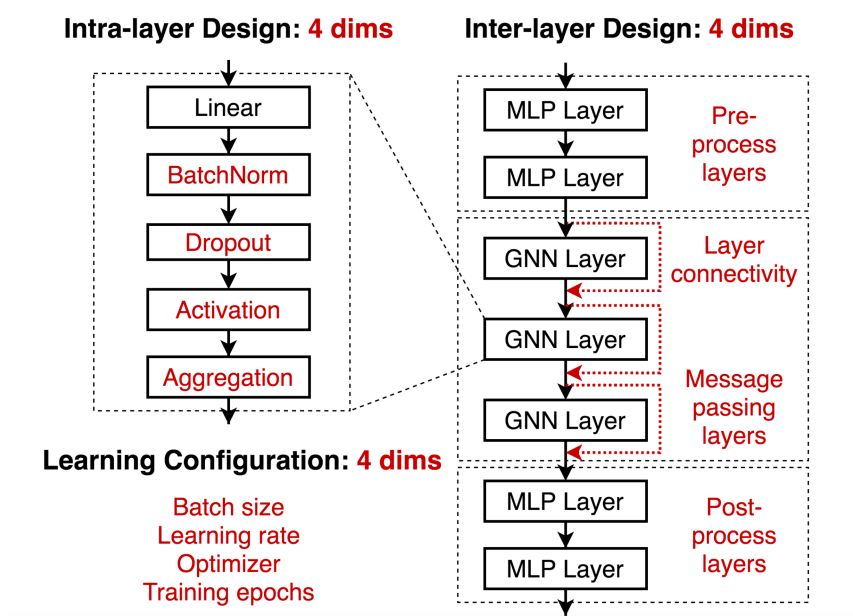


Design space for GNNs

The paper ([You et al., 2020](#)) introduced a taxonomy of various design dimensions for graph neural networks. These can be divided into intra- and inter-layer design dimensions, and different choices of learning configurations. (Note that some new design choices that have emerged over the past five years are not captured by this taxonomy.)



Intra-layer design

Linear transformation

Each graph neural network layer is built upon a *linear transformation*, such as a single-layer perceptron. Note that well-performing graph neural networks are typically not "deep" but shallow, consisting of only a few layers. However, each individual layer can itself be a deep neural network, such as a multi-layer perceptron.

Batch normalization

Batch normalization can be used to stabilize the training of graph neural networks ([Lofte & Szegedy, 2015](#)). Given a minibatch of node embeddings $\{h_i\}_{i=1}^n \in \mathbb{R}^{n \times d}$, the goal is to recenter each embedding dimension to have zero mean and rescale it to unit variance.

Specifically, for each embedding dimension $j \in \{1, \dots, d\}$, one first computes its mean μ_j and variance σ_j^2 .

$$\mu_j = \frac{1}{n} \sum_{i=1}^n h_{ij}$$

$$\sigma_j^2 = \frac{1}{n} \sum_{i=1}^n (h_{ij} - \mu_j)^2$$

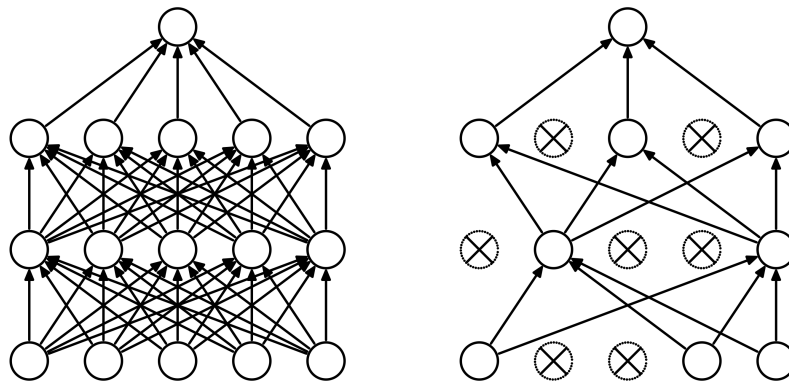
Then, the embeddings are normalized as

$$\hat{h}_{ij} = \alpha_j \frac{h_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} + \beta_j,$$

where $\alpha_j, \beta_j \in \mathbb{R}$ are learnable parameters and $\epsilon > 0$ is a small number preventing division by zero.

Regularization by dropout

Dropout is an effective regularization technique that helps prevent neural networks from overfitting ([Srivastava et al., 2014](#)), especially when the available data is limited or noisy. The idea is to randomly drop or set neurons in the neural network to zero (i.e. turn them off). This encourages the neural network to rely on all neurons rather than depending too heavily on any single one. Alternatively, instead of dropping neurons, one can randomly drop individual edges in the neural network. Note that dropout is applied only during training, while the complete neural network is employed for testing.



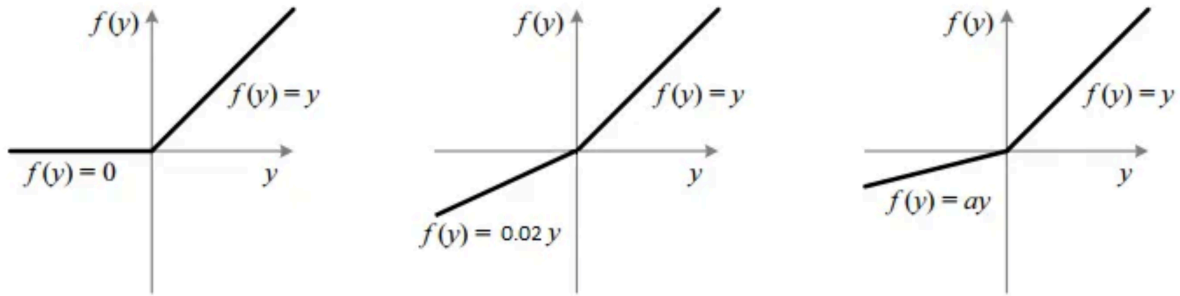
In graph neural networks, *node dropout* is applied after the linear transformation by randomly setting each embedding dimension to zero with some predefined probability. Alternatively, *edge dropout* can be employed, where elements of the learnable weight matrices are randomly masked out with a specified probability.

Non-linear activation

Non-linear activation functions σ increase the expressivity of neural networks, enabling them to learn or approximate non-linear mappings. In graph neural networks, a popular choice is to use parametric ReLU defined as

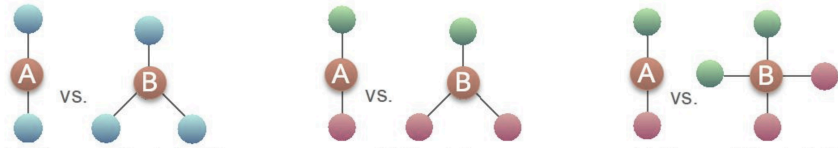
$$\sigma(y) = \max(ay, y),$$

where $a \in [0, 1)$ is a learnable parameter.



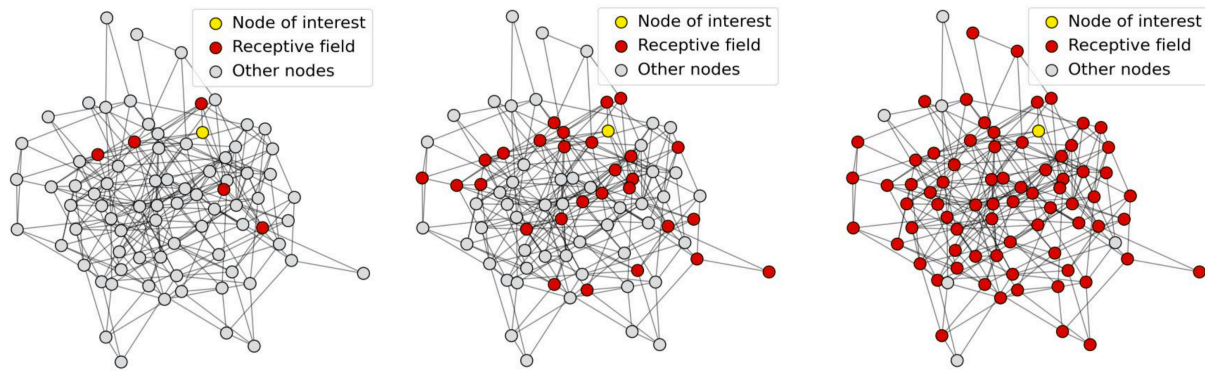
Aggregation function

At each layer of a graph neural network, every node i aggregates the embeddings of its neighbors Γ_i using a permutation-invariant function. In general, summation provides the most expressive aggregation, followed by mean, with maximum being the least expressive ([Xu et al., 2019](#)).



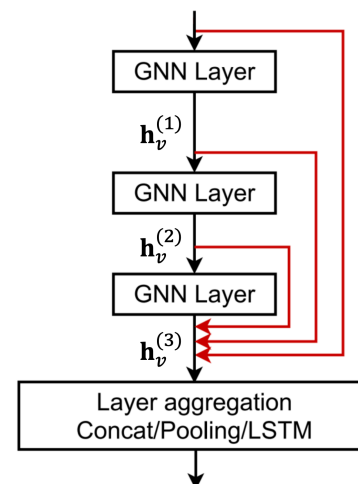
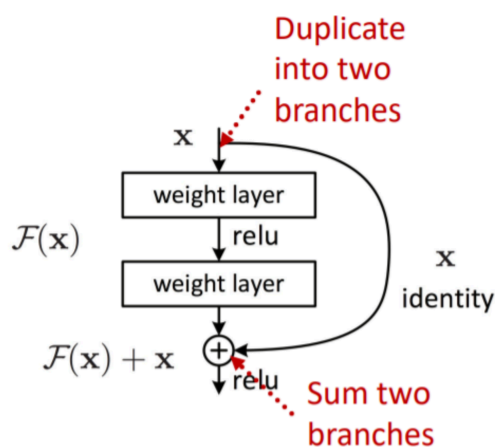
Inter-layer design

A graph neural network consists of L message passing layers, where L is typically small, $L \in \{2, 3, 4, \dots\}$. This is because, in small-world networks, the number of nodes in a node's L -hop neighborhood grows exponentially with L (also called its *receptive field*). As a result, for large L , all node embeddings tend to converge to the same value, a phenomenon known as the *over-smoothing problem*. In practice, one should be cautious when adding additional message passing layers. Note, however, that in spatial graphs such as molecular networks with very large diameters, the number of layers L often needs to be much higher, e.g., $L \gg 10$.



Several approaches exist to increase the expressivity of otherwise shallow graph neural networks.

- Design each individual message passing layer as a "deep" neural network.
- Add pre- and post-processing neural network layers that do not perform message passing. Pre-processing layers are especially important when encoding node features is required (e.g., text or images). Post-processing layers are useful when additional transformations over node embeddings are needed (e.g., in knowledge graphs).
- Incorporate *skip connections* into the graph neural network design. Node embeddings from earlier layers can sometimes better differentiate nodes, so adding shortcuts allows these earlier embeddings to have a greater influence on the final node representations.



Learning configurations

More training epochs generally lead to better performance, provided that an appropriate early-stopping strategy is employed based on validation set performance. On the other hand, the learning rate $\eta \in \{0.1, 0.01, 0.001, \dots\}$ is notoriously difficult to set in practice, so it should be treated as a hyperparameter and tuned using an appropriate validation technique.

Study of design space

([You et al., 2020](#)) conducted an extensive empirical study comparing 315,000 graph neural

network designs across 32 different prediction tasks. The main findings are summarized below, where a lower rank indicates better overall performance. Furthermore, the authors released the [GraphGym](#) platform, which enables systematic exploration of different graph neural network designs and tasks, as well as the transfer of optimal designs across tasks.

