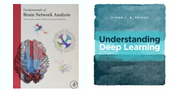
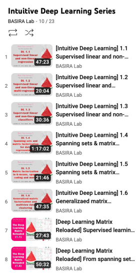
Special thanks to Simon Prince, Alex Fornito, Andrew Zalesky, Edward Bullmore, Jure Leskovec and all those who shared their passion about graphs and deep learning.



https://github.com/udlbook/

Preliminaries: machine & deep learning, feature extraction and graphs

Preliminaries

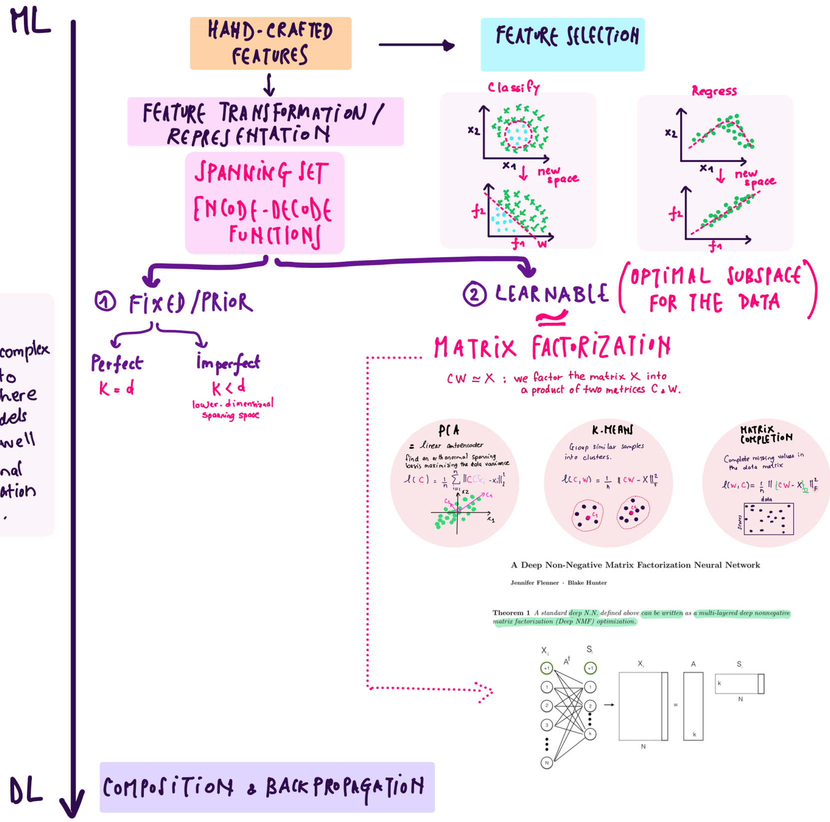


DL 1∞

CLASSIC FEATURE REPRESENTATION METHODS

AIM

Transform complex features into a space where linear models can work well in the final representation space.

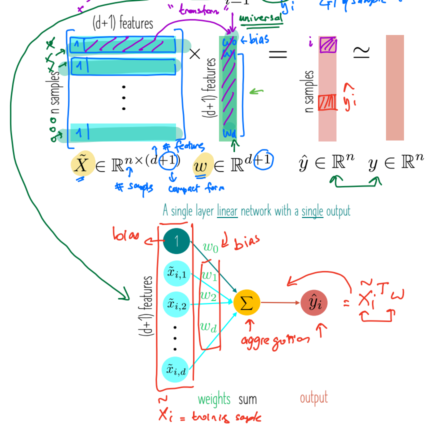


Reminder: Supervised regression (linear)

Single regression model

$$l(w) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i^T w - y_i)^2$$

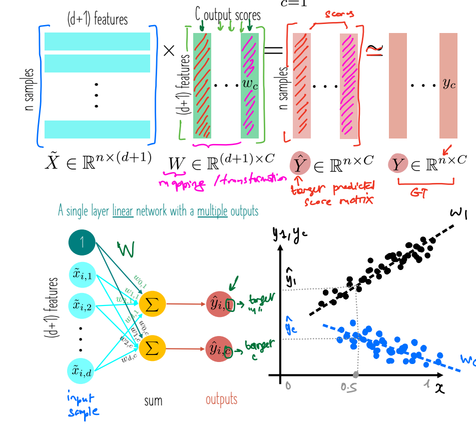
$\tilde{X} \in \mathbb{R}^{n\times(d+1)}$, $w \in \mathbb{R}^{d+1}$, $\hat{y} \in \mathbb{R}^n$, $y \in \mathbb{R}^n$

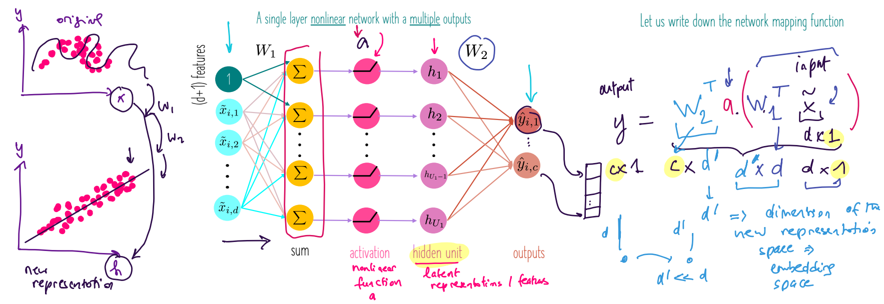


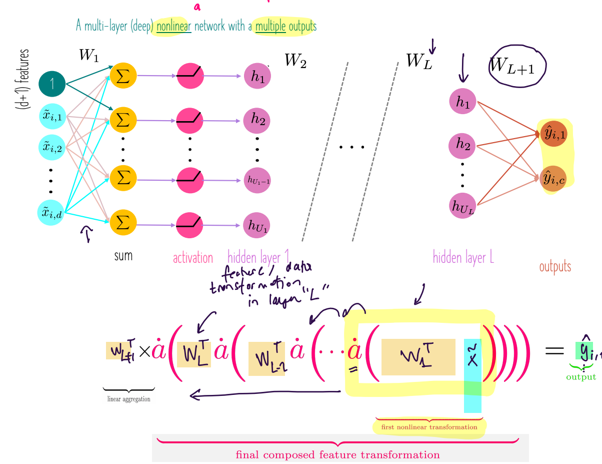
Multi-regression models

$$l(W) = \|\tilde{X}W - Y\|_2^2 = \frac{1}{c}\sum_{c=1}^{C}(\tilde{X}w_c - y_c)^2$$

$\tilde{X} \in \mathbb{R}^{n\times(d+1)}$, $W \in \mathbb{R}^{(d+1)\times C}$, $\hat{Y} \in \mathbb{R}^{n\times C}$, $Y \in \mathbb{R}^{n\times C}$





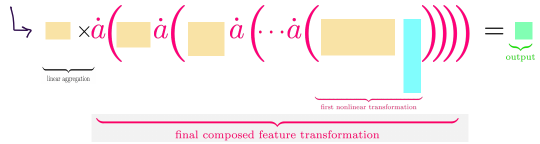


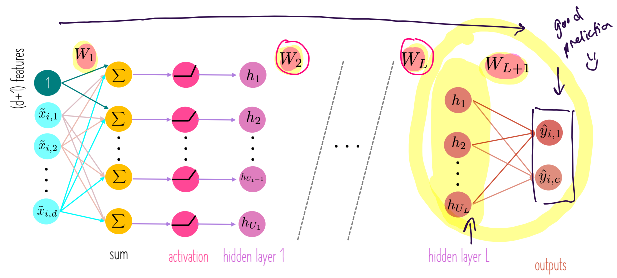
Special thanks to Simon Prince, Alex Fornito, Andrew Zalesky, Edward Bullmore, Jure Leskovec and all those who shared their passion about graphs and deep learning.

https://github.com/udlbook/

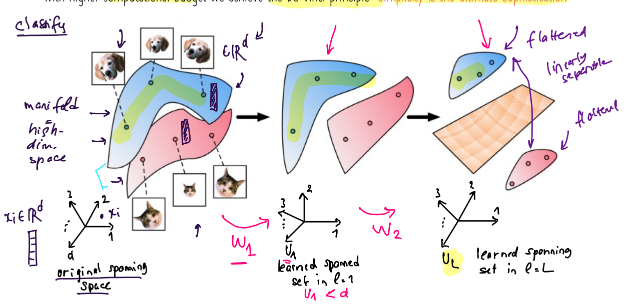
Feature embedding

Feature embedding in deep learning and manifold learning



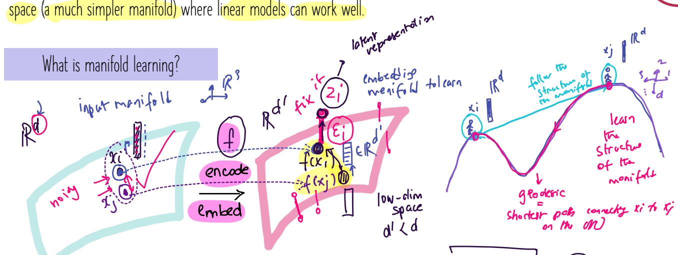


With higher computational budget we achieve the Da Vinci principle "Simplicity is the Ultimate Sophistication"



☺ The power of DL models lies in learning how to embed complex features sequentially into manifolds (subspaces to learn) that decrease in complexity until 'finding' an ultimate low-dimensional representation space (a much simpler manifold) where linear models can work well.

What is manifold learning?



$f: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ is an embedding function , $f(x_i) + \varepsilon_i$

d : intrinsic dimension , d' : extrinsic dimension

f : unknown , distributions of ε_i and $f(x_i)$ are unknown.

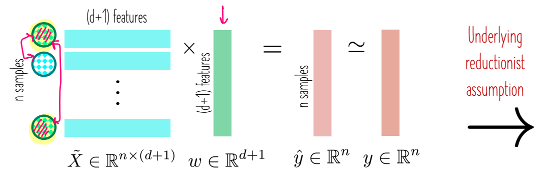
☹ Underlying reductionist hypothesis: neighborhood preservation across the original and embedding spaces. ??

☹ The neighborhood in the original domain may be highly noisy (outliers in the local neighborhood of a sample).

The evolving landscape of feature embedding

ML problem formalization

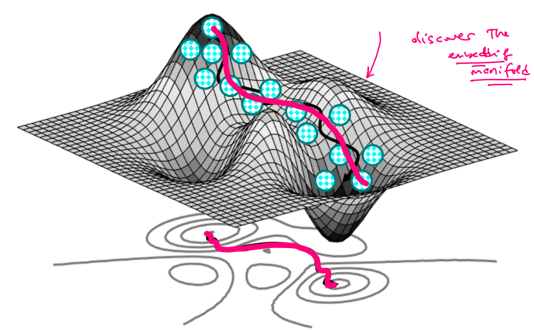
$$l(w) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i^T w - y_i)^2$$



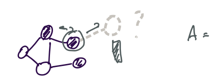
Underlying reductionist assumption

We learn a shared mapping w across all training samples.
However, our training samples are not allowed to directly interact.

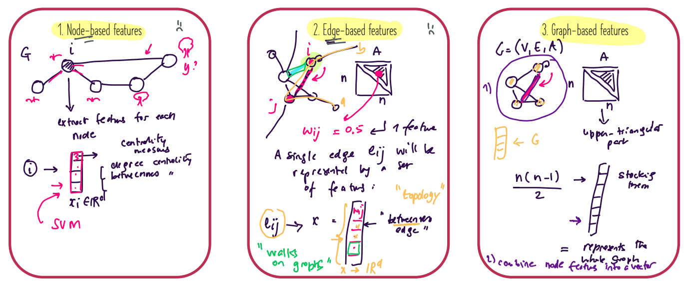
Individualistic points -- data samples live "independently" on a high-dimensional manifold.



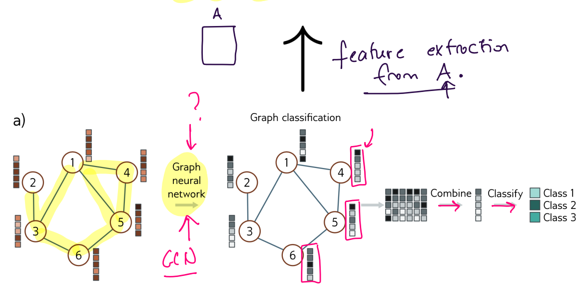
Even in manifold learning, we learn how to calculate the similarity between points but not how to connect them. We are one step away from that!



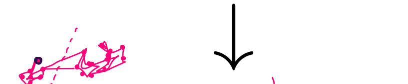
☹ Cannot utilize node, edge and graph features.
☹ Such approaches cannot obtain embeddings for nodes not in the training set. If we add a node, we need to ~~recomplete~~ recalculate all node feature representations (embeddings).

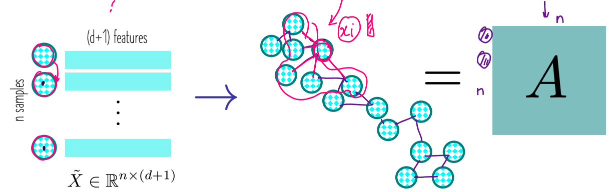


Classical node, edge and graph-based feature extraction methods



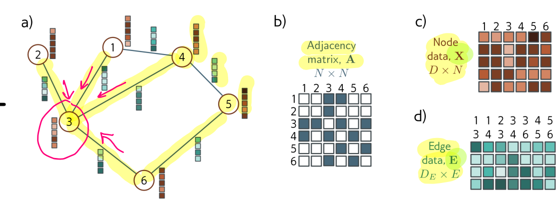
How to get the node features?
How to get the edge or graph features?





Use a graph to model and capture the relationships between pairs of data points. Each sample interacts with all other samples.

Collectivist points -- data samples co-interact and co-learn together via a graph structure (A).



We can also add edge features. Remember you've got a graph to learn on!

Special thanks to Simon Prince, Alex Fornito, Andrew Zalesky, Edward Bullmore, Jure Leskovec and all those who shared their passion about graphs and deep learning.
https://github.com/udlbook/

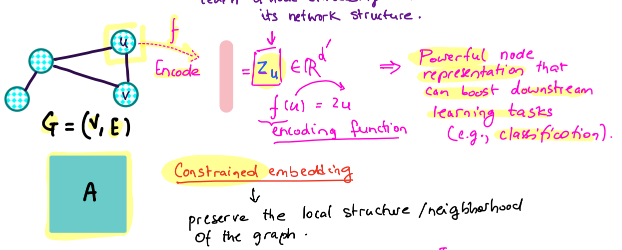
Graph Convolutional Networks (UDL-13)

Classical node encoding/embedding (with no features)

$x_i \in \mathbb{R}^d$

Sample embedding = latent/hidden representation = encoding = transformation = mapping

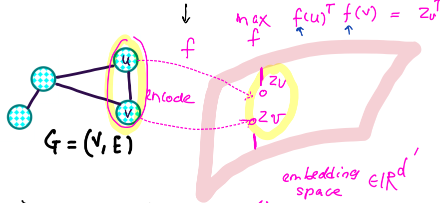
learn a node embedding that encodes its network structure.



⇒ Powerful node representation that can boost downstream learning tasks (e.g., classification).

Constrained embedding
↓
preserve the local structure / neighborhood of the graph.

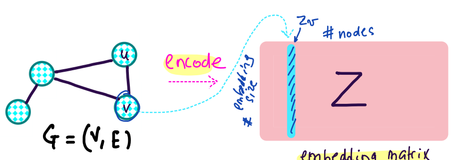
$\max_f \; f(u)^T f(v) = z_u^T z_v$

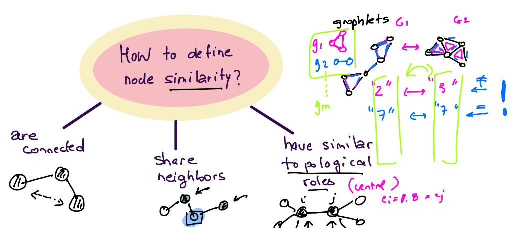


ML principle
"Tell me who your neighbors are
I'll tell who you are"

→ similarity$(u, v) \approx z_u^T z_v \approx f(u)^T f(v)$

objective : $\{$ maximize $f(u)^T f(v) = z_u^T z_v$ for pairs (u, v) that are close / similar.





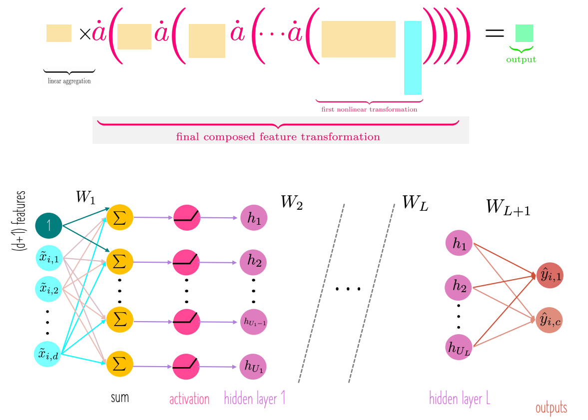
★ TRUE OR FALSE :

F 1) Shallow node embedding is a task-dependent learning problem → unsupervised. "SL - regression" "targets"

T 2) Node embedding is unsupervised or self-supervised.

F 3) We utilize node labels and features for embedding.

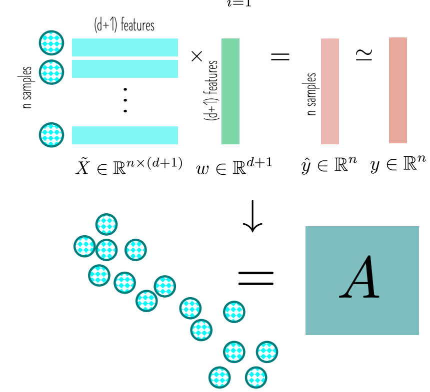
A single GCN layer (shallow)



With higher computational budget we achieve the Da Vinci principle "Simplicity is the Ultimate Sophistication"

Our goal: Shifting from an "individualistic" to a "collectivist" learning paradigm through the graph matrix $\boldsymbol{A}$

$$l(w) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i^T w - y_i)^2$$



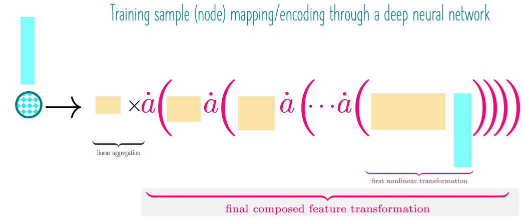
$$l(w) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i^T w - y_i)^2$$

Deep (sequential) node representation learning

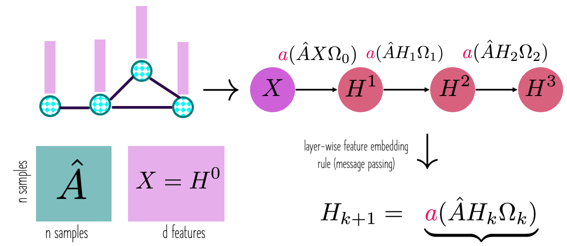
$$\begin{aligned}
\mathbf{H}_1 &= \mathbf{F}[\mathbf{X}, \mathbf{A}, \phi_0] \\
\mathbf{H}_2 &= \mathbf{F}[\mathbf{H}_1, \mathbf{A}, \phi_1] \\
\mathbf{H}_3 &= \mathbf{F}[\mathbf{H}_2, \mathbf{A}, \phi_2] \\
\vdots &= \vdots \\
\mathbf{H}_K &= \mathbf{F}[\mathbf{H}_{K-1}, \mathbf{A}, \phi_{K-1}],
\end{aligned}$$

A vanilla GCN layer

Training sample (node) mapping/encoding through a deep neural network



Graph mapping/embedding through a graph neural network



$\hat{A} = A + I$

The output of a node in a hidden layer depends on itself and its neighbors.

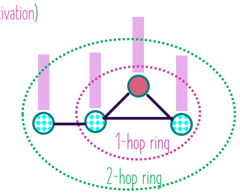
layer-wise feature embedding rule (message passing)

$$H_{k+1} = a(\hat{A}H_k\Omega_k)$$

propagation rule: feature aggregation, feature transformation, activation (nonlinearity)

Let us linearize the network by removing the nonlinearity (activation)

$$H_1 = \hat{A}H_0\Omega_0$$

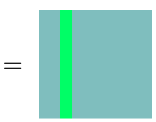


$H_{k+1} =$

IMPORTANT PROPERTY

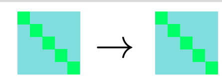
Can we map any node with any neighborhood size?

$\underbrace{\hat{A}^{k+1}}_{\text{receptive field}} =$



Reminder: for each node i, this matrix stores the total number of walks of length (k+1) from node i to a node j in the graph. If you get "0", it means you can't get to j via (k+1) hops or walk.

Layer-wise normalization and propagation rule



$H_{k+1} = a(\hat{D}^{-1}\hat{A}H_k\Omega_k)$

normalized propagation rule

$H_{k+1} = f(H_k, A)$

generalized propagation rule

→ controls the latent embedding variance between layers, which may suppress the gradient explosion or vanishing phenomena.

→ such embedding normalization maintains the scale of the output feature vectors.

Special thanks to Simon Prince, Alex Fornito, Andrew Zalesky, Edward Bullmore, Jure Leskovec and all those who shared their passion about graphs and deep learning.

https://github.com/udlbook/

Graph Convolutional Networks (UDL-13)

Introducing layer-wise non-linearity and biases into GCN

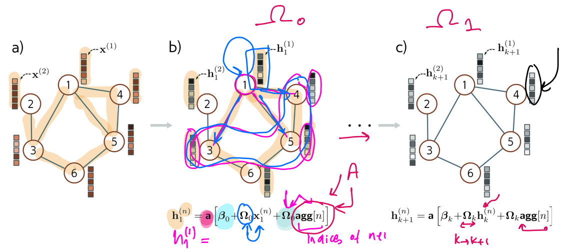


Figure Simple Graph CNN layer. a) Input graph consists of structure (embodied in graph adjacency matrix $\mathbf{A}$, not shown) and node embeddings (stored in columns of $\mathbf{X}$). b) Each node in the first hidden layer is updated by (i) aggregating the neighboring nodes to form a single vector, (ii) applying a linear transformation Ω_0 to the aggregated nodes, (iii) applying the same linear transformation Ω_0 to the original node, (iv) adding these together with a bias β_0, and finally (v) applying a nonlinear activation function $\mathbf{a}[\bullet]$ like a ReLU. This process is repeated at subsequent layers (but with different parameters for each layer) until we produce the final embeddings at the end of the network.

$f(x) = \max(0, x)$

Node embedding update rule

$A = [\ \ ]$, j, $ne(i) = \{j\}$

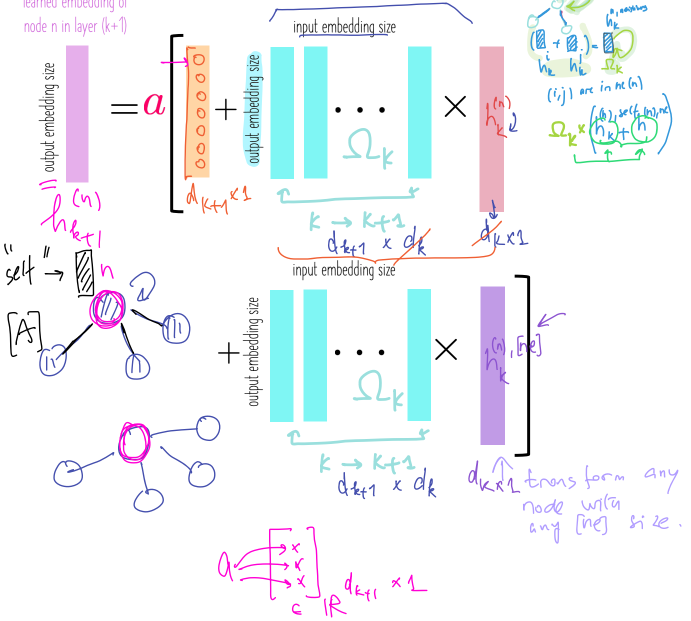
message (embedding) aggregation across neighbors

$$\text{agg}[n,k] = \sum_{m \in ne[n]} \mathbf{h}_k^{(m)},$$

$$\mathbf{h}_{k+1}^{(n)} = \mathbf{a}\left[\beta_k + \Omega_k \cdot \mathbf{h}_k^{(n)} + \Omega_k \cdot \text{agg}[n,k]\right]$$

self-update, self-aggr., neighborhood aggregation

learned embedding of node n in layer (k+1)



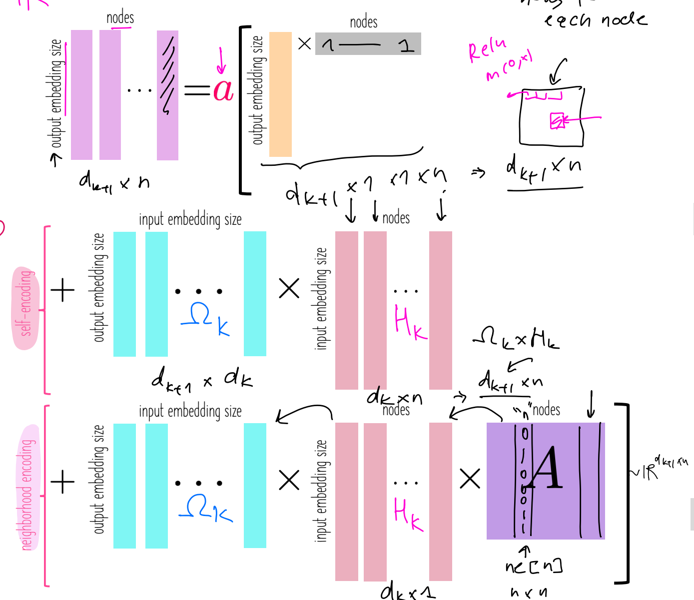
Update rule for all nodes

$$\mathbf{H}_{k+1} = \mathbf{a}\left[\beta_k \mathbf{1}^T + \Omega_k \mathbf{H}_k + \Omega_k \mathbf{H}_k \mathbf{A}\right]$$

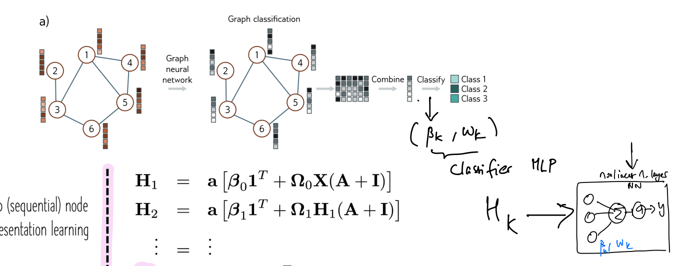
$$= \mathbf{a}\left[\beta_k \mathbf{1}^T + \Omega_k \mathbf{H}_k(\mathbf{A}+\mathbf{I})\right],$$

$\mathbb{R}^{d_{k+1} \times n}$

agg neighbors nods for each node



Example: graph classification



Deep (sequential) node representation learning

$$\begin{aligned}
\mathbf{H}_1 &= \mathbf{a}[\beta_0 \mathbf{1}^T + \Omega_0 \mathbf{X}(\mathbf{A}+\mathbf{I})] \\
\mathbf{H}_2 &= \mathbf{a}[\beta_1 \mathbf{1}^T + \Omega_1 \mathbf{H}_1(\mathbf{A}+\mathbf{I})] \\
\vdots &= \vdots \\
\mathbf{H}_K &= \mathbf{a}[\beta_{K-1} \mathbf{1}^T + \Omega_{K-1} \mathbf{X}_{k-1}(\mathbf{A}+\mathbf{I})]
\end{aligned}$$

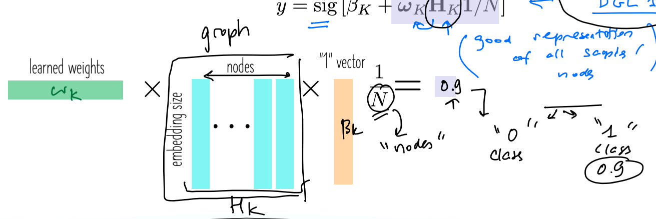
Network output function (output prediction)

$$\mathbf{f}[\mathbf{X}, \mathbf{A}, \Phi] = \text{sig}\left[\beta_K + \omega_K \mathbf{H}_K \mathbf{1}/N\right]$$

$$y = \text{sig}\left[\beta_K + \omega_K \mathbf{H}_K \mathbf{1}/N\right]$$

Section 3 of DGL 1.3

(good representation of all sample / node)



Going deeper by adding more layers

How to combine the current node and aggregated neighbors?

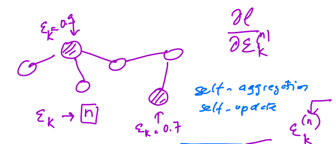
Simply sum them up

neighborhood aggregation; self-embedding update = self-aggregation

$$\mathbf{H}_{k+1} = \mathbf{a}\left[\beta_k \mathbf{1}^T + \Omega_k \mathbf{H}_k(\mathbf{A}+\mathbf{I})\right]$$

$\hat{A}$ → vanilla GCN

Add a diagonal enhancement



$$\mathbf{H}_{k+1} = \mathbf{a}\left[\beta_k \mathbf{1}^T + \Omega_k \mathbf{H}_k(\mathbf{A} + (1+\epsilon_k)\mathbf{I})\right]$$

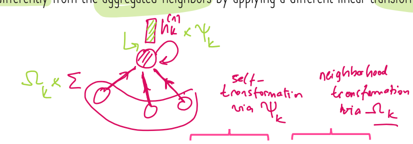
$\epsilon \in \mathbb{R}$ to learn for each layer k.

control this influence

give more weight & control to the node itself!

Going deeper with more layers

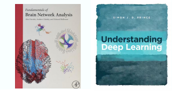
Treat the current node differently from the aggregated neighbors by applying a different linear transform Ψ_k to the current node:



$$\begin{aligned}
\mathbf{H}_{k+1} &= \mathbf{a}\left[\beta_k \mathbf{1}^T + \Psi_k \mathbf{H}_k + \Omega_k \mathbf{H}_k \mathbf{A}\right] \\
&= \mathbf{a}\left[\beta_k \mathbf{1}^T + \begin{bmatrix}\Omega_k & \Psi_k\end{bmatrix} \begin{bmatrix}\mathbf{H}_k \mathbf{A} \\ \mathbf{H}_k\end{bmatrix}\right] \\
&= \mathbf{a}\left[\beta_k \mathbf{1}^T + \Omega_k' \begin{bmatrix}\mathbf{H}_k \mathbf{A} \\ \mathbf{H}_k\end{bmatrix}\right],
\end{aligned}$$

DGL 2∞ FEATURE EMBEDDING FROM ML TO GCNs

Special thanks to Simon Prince, Alex Fornito, Andrew Zalesky, Edward Bullmore, Jure Leskovec and all those who shared their passion about graphs and deep learning.



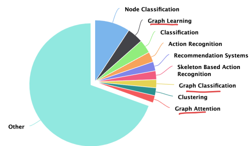
https://github.com/udlbook/

Graph Convolutional Networks (UDL-13)

GCNs are hot topics!

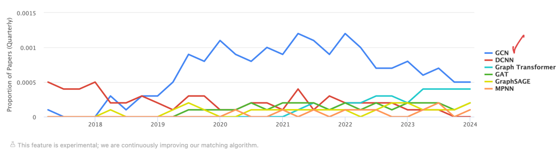
Papers with code

Tasks



Task	Papers	Share
Node Classification	73	9.29%
Graph Learning	33	4.20%
Classification	28	3.56%
Action Recognition	21	2.67%
Recommendation Systems	19	2.42%
Skeleton Based Action Recognition	18	2.29%
Graph Classification	17	2.16%
Clustering	16	2.04%
Graph Attention	15	1.91%

Usage Over Time



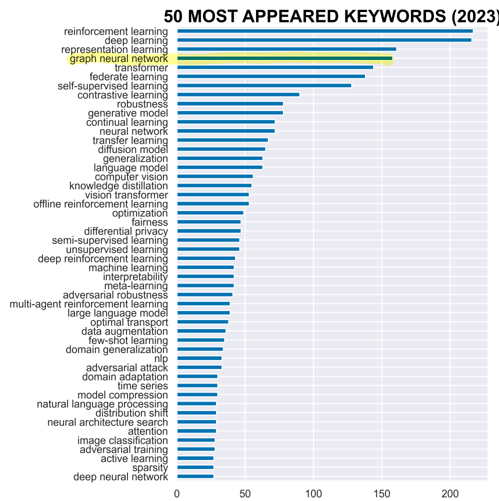
⚠ This feature is experimental; we are continuously improving our matching algorithm.

https://paperswithcode.com/method/gcn

ICLR 2023 open review data

- **Top-10 Ranking between 2022 and 2023** (full list please refer to keywords.md)

Keyword	2022	2023
reinforcement learning	1	1
deep learning	2	2
representation learning	4	3
graph neural network	3	4
transformer	5	5
federate learning	7	6
self-supervised learning	6	7
contrastive learning	10	8
robustness	9	9
generative model	8	10



https://github.com/EdisonLeeeee/ICLR2023-OpenReviewData?tab=readme-ov-file

Recall box: summarize what you remember from this lecture below