GRAPH NEURAL NETWORKS (GNN)

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

Zhang, Z., Cui, P., & Zhu, W. (2018). Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907.*

GRAPH TASKS

- Machine Learning tasks on Networks
  - Link prediction (follow recommendation, drug/illness relationship…)
  - Node classification (bot detection in social media)
  - Attribute regression (age of individuals in a social media…)
  - Link classification/regression (relationship is: friend/colleague/lover ?)
  - Graph classification (Molecule classification)
  - Community detection
  - …

WHY NN

- Neural networks are especially useful with structured data
  - Images (each pixel has left/right/top/bottom pixels)
  - Text (each word has a specific position in a sentence)

- Graphs are pure structure!

INTRO TO DEEP NEURAL NETWORKS

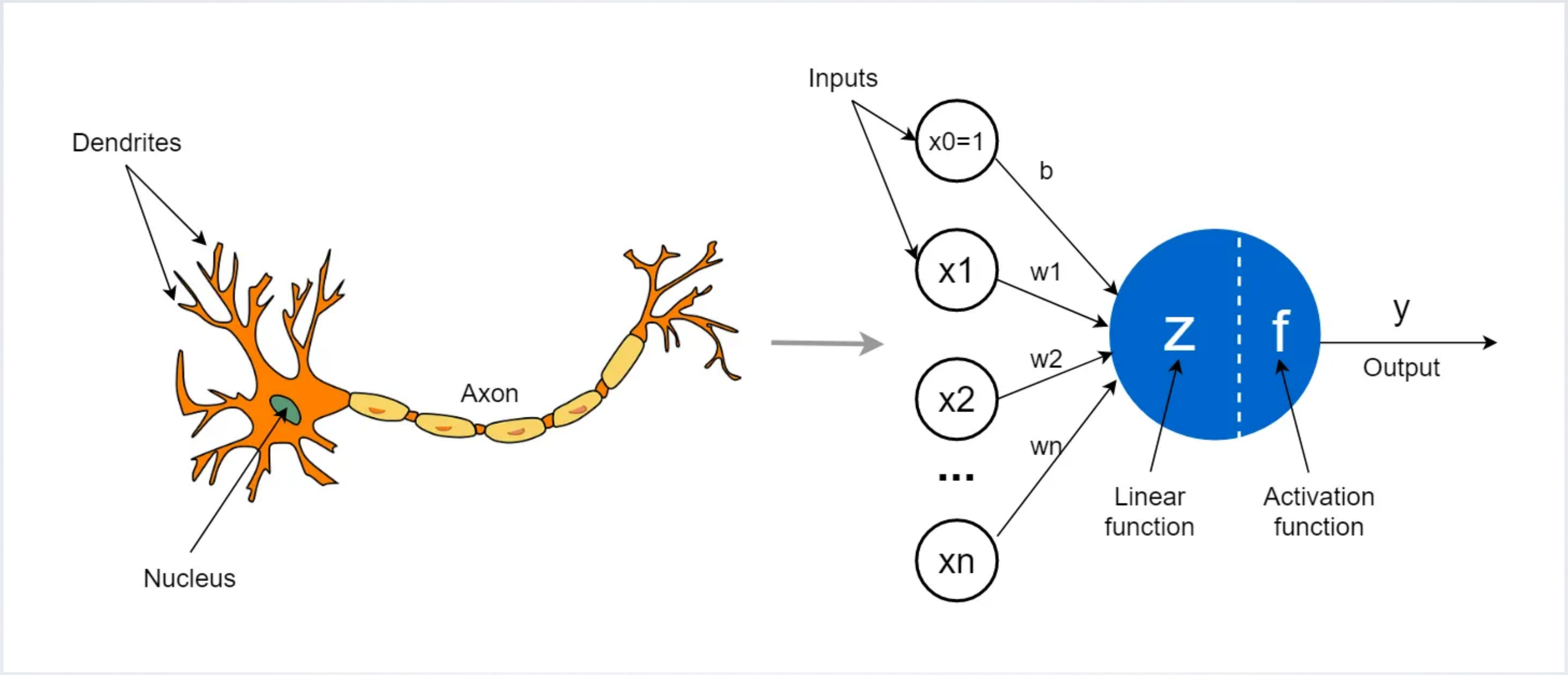
DNN

LINEAR REGRESSION

- We have variables describing an object $(x_1, x_2 \dots)$
  - i.e., apartments:
    - Surface
    - Number of rooms
    - Which flat
    - …
- A target to "learn to guess"
  - e.g., The price of the apartment in euros
- We assume that the target can be expressed as a linear combination of the inputs
  - A weighted sum…
  - We need to learn the weights $(b_1, b_2 \dots)$ + intercept (b_0)
- $y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$

PERCEPTRON

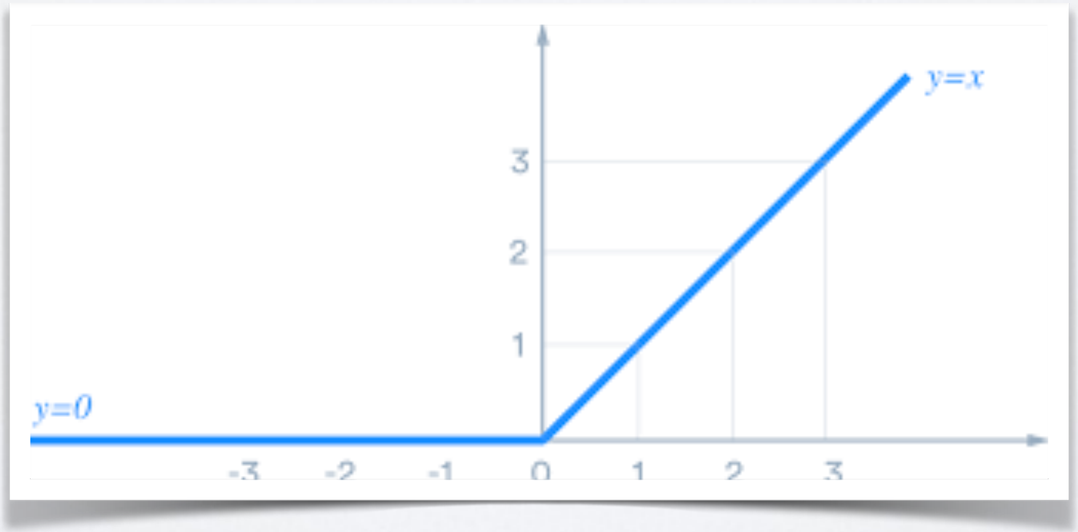
Artificial neuron



https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc

PERCEPTRON

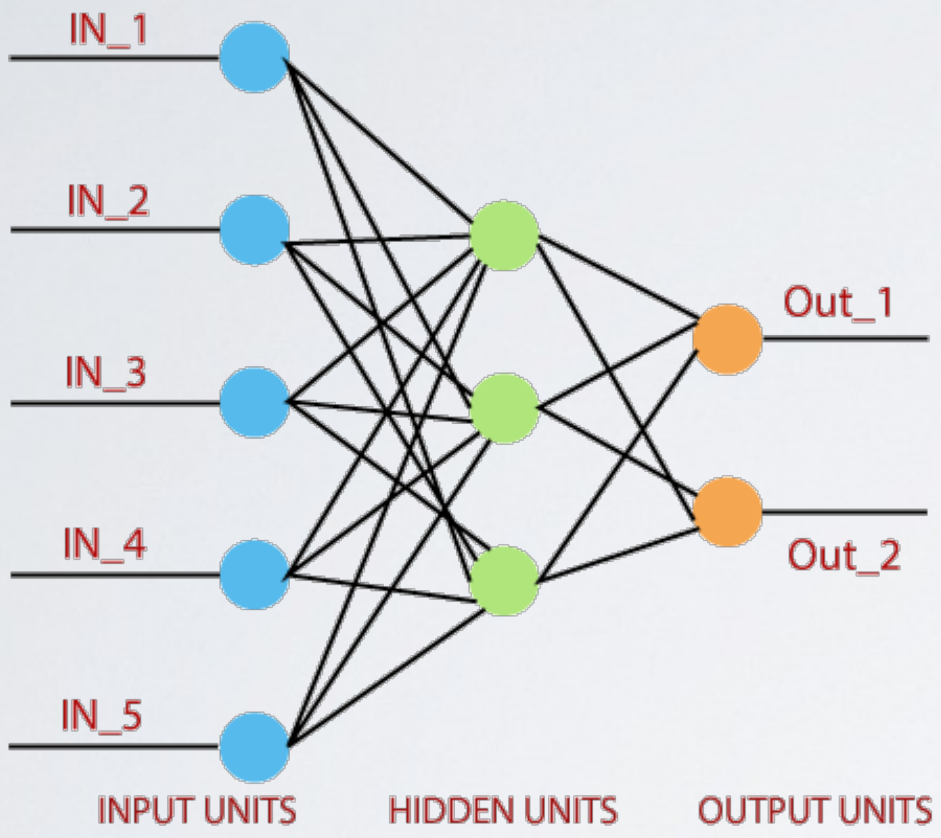
- Perceptron = linear regression + activation function
- Common activation function used here:
  - ReLu: Rectified Linear Unit: $f(x) = \max(0, x)$
  - Necessary to introduce non-linearity



PERCEPTRON

- Perceptron = linear regression + activation function
- Common activation function used here:
  - ReLu: Rectified Linear Unit: $f(x) = \max(0,x)$

NEURAL NET



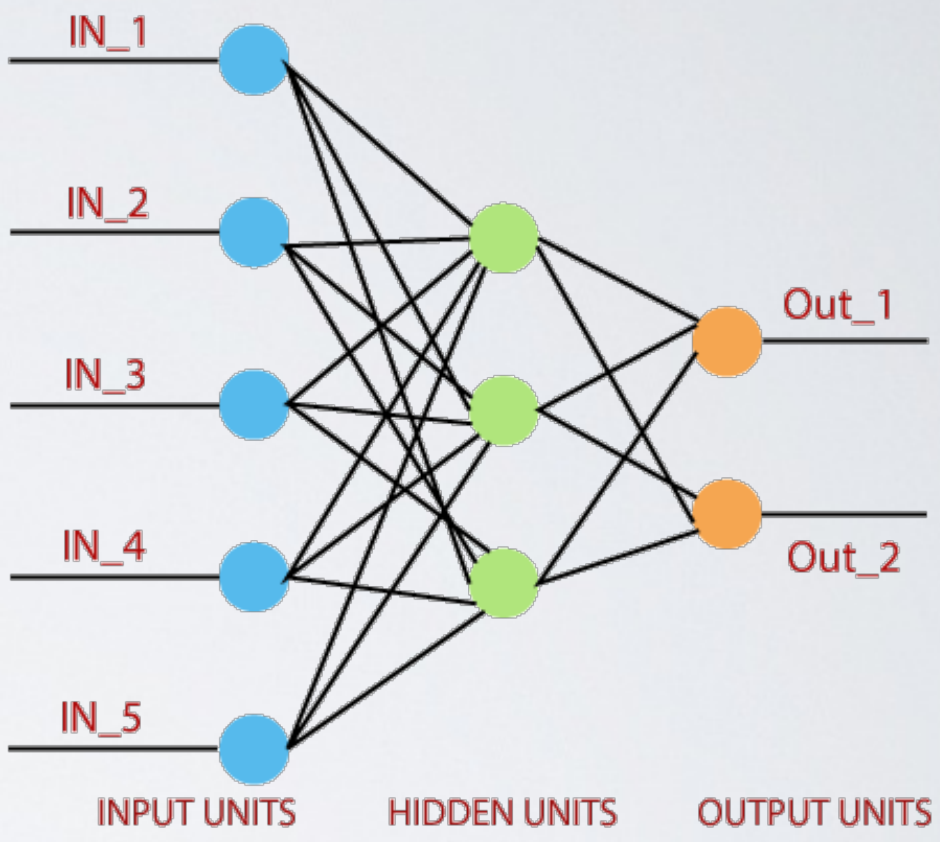
3 perceptrons

Chained with 2 perceptrons

NEURAL NET

- A key element to understand:
  - <u>All weights are trained simultaneously</u>

- Gradient descent
  - Beyond the scope of this class: consider a recipe that allows you to find a good solution in a reasonable time.



BACKWARD STEP

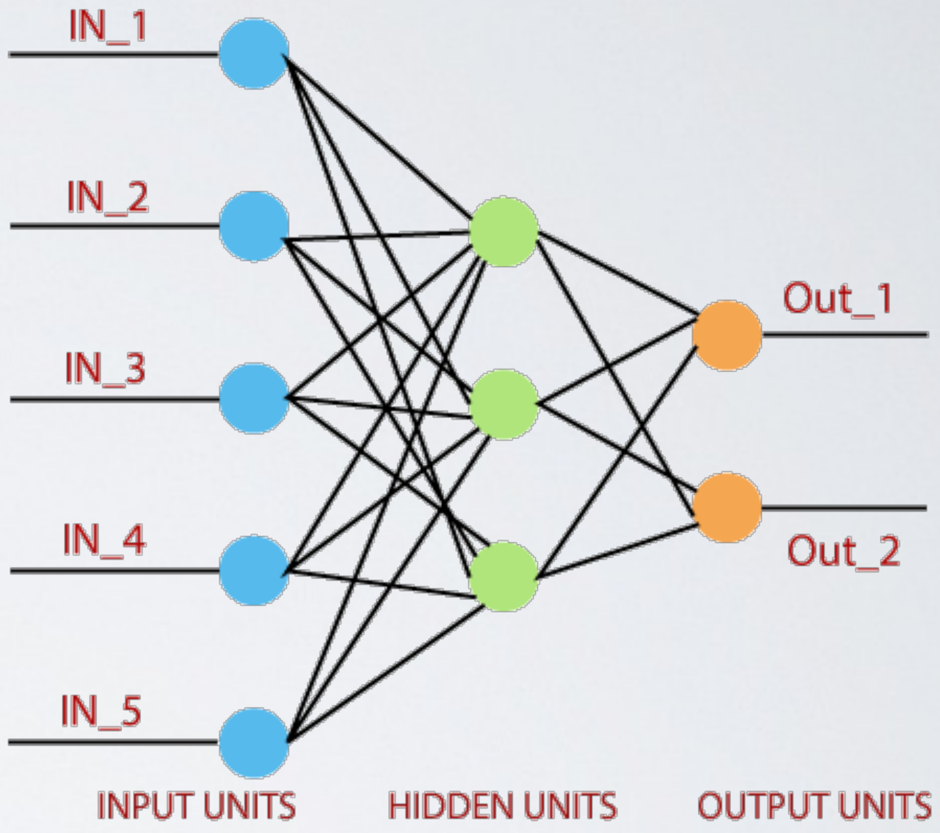
- To learn the weights, we use **back-propagation**
- Short summary
  - A **loss** function is defined to compare the "predicted values" with ground truth labels (at this point, we need some labels…)
  - The **derivative** of the cost function relative to weights is computed
  - Weights are updated using **grading descent** (i.e., weights are modified in the direction that will minimize the loss)

https://en.wikipedia.org/wiki/Backpropagation

EXAMPLE, FLATS IN PARIS

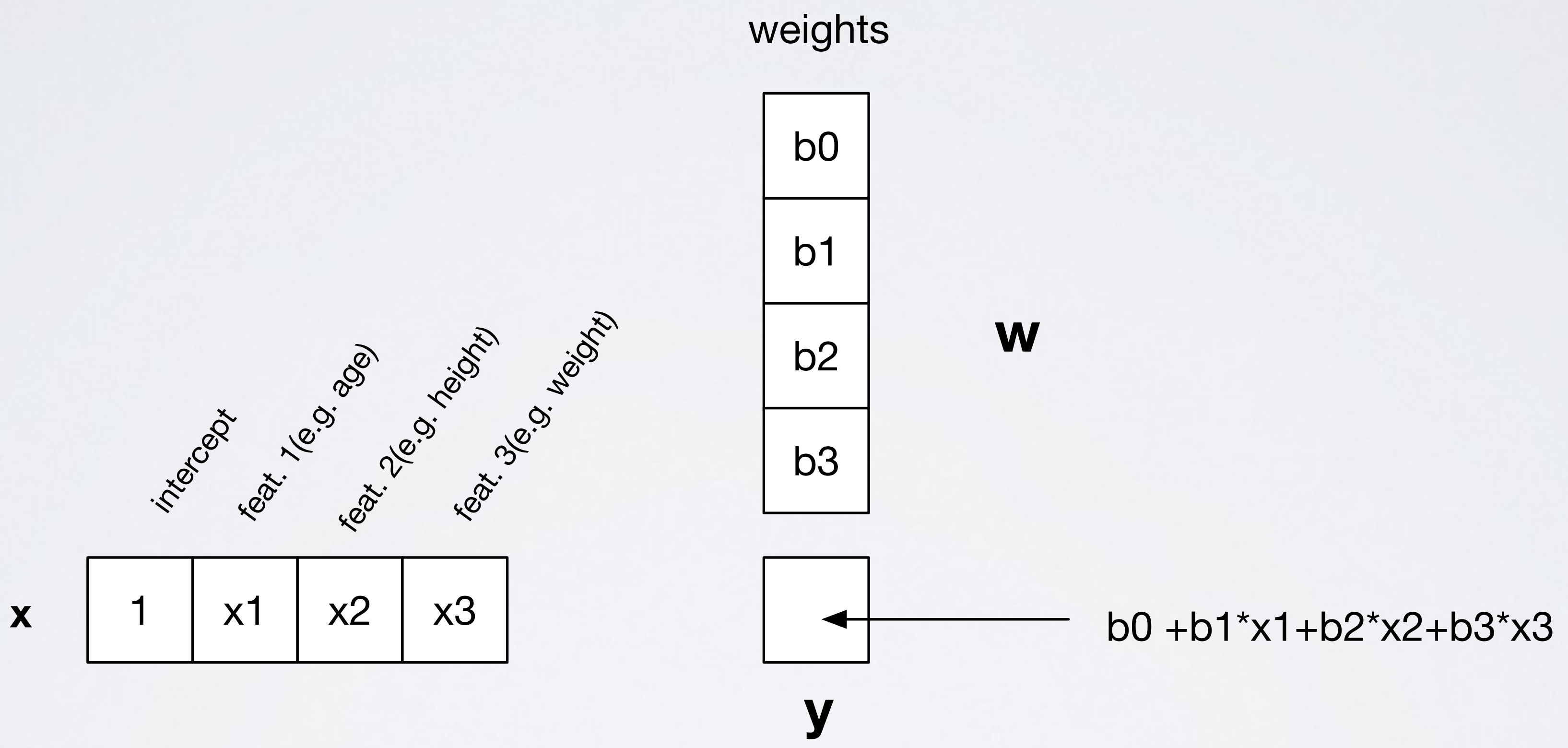
- Input features:
  - Floor
  - Surface
  - Age
- Target: Price
- Simple linear regression:
  - Higher floor increase price (view…)
  - Higher surface increase price
  - Age lower price

- Hidden units capture intermediate notions
  - Small surface, old age, high floor =>"chambre de bonne"
  - Age>100, large surface => Prestigious building
- Final layout combines those factors
  - "Chambre de bonne" negative effect on price computed based on surface…



MATRIX EXPRESSION

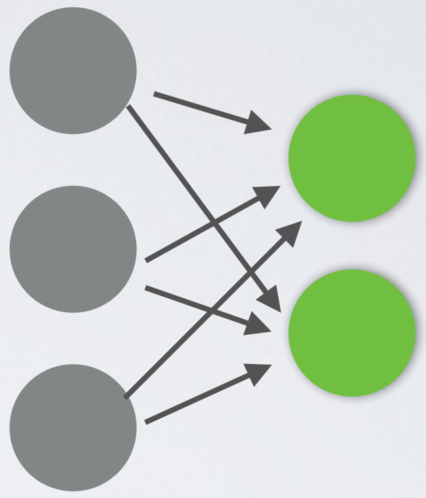
$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$



MATRIX EXPRESSION

$$\mathbf{W} = \begin{array}{c|cc} & \text{neuron1} & \text{neuron2} \\ \hline \text{w1} & x & \\ \text{w2} & y & \\ \text{w3} & z & \end{array}$$

$$\mathbf{X} = \begin{array}{c|ccc} & \text{feat. 1(e.g. age)} & \text{feat. 2(e.g. height)} & \text{feat. 3(e.g. weight)} \\ \hline \text{n1} & a & b & c \\ \text{n2} & & & \\ \text{n3} & & & \\ \text{n4} & & & \\ \text{...} & & & \end{array}
\qquad
\begin{array}{|c|c|} \hline & \\ \hline & \\ \hline & \\ \hline & \\ \hline \end{array}$$

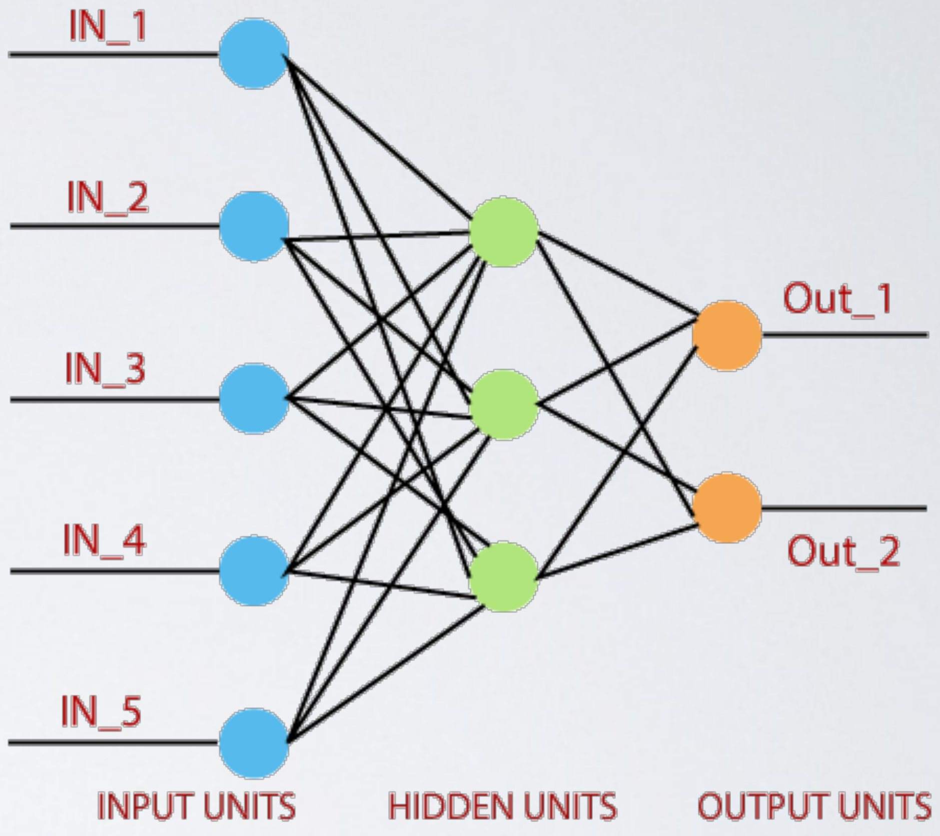


-3 input units
-2 hidden units

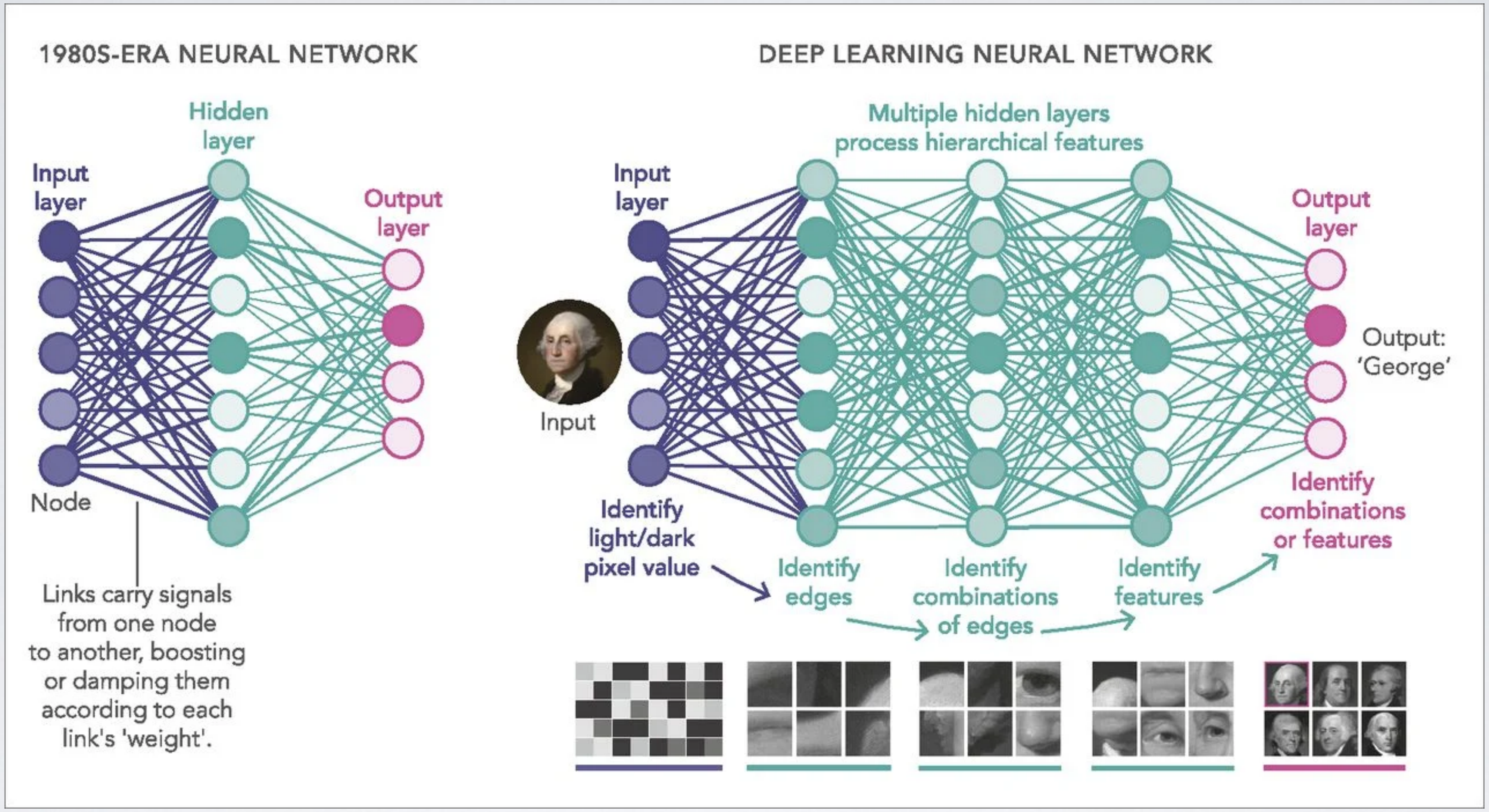
$$H = XW$$

MATRIX EXPRESSION

- With activation function
  - $H = \mathrm{ReLU}(XW^T + 1b^T)$
    - 1 is unit vector of size n
- With hidden units
  - $H = \sigma(XW^T)W_2^T + 1b_2^T)$
    - Here, no intercept in hidden



DEEP NEURAL NETWORKS


1980S-ERA NEURAL NETWORK
Input layer
Hidden layer
Output layer
Node
Links carry signals from one node to another, boosting or damping them according to each link's 'weight'.
DEEP LEARNING NEURAL NETWORK
Multiple hidden layers process hierarchical features
Input layer
Input
Output layer
Output: 'George'
Identify light/dark pixel value
Identify edges
Identify combinations of edges
Identify features
Identify combinations or features

https://www.pnas.org/doi/10.1073/pnas.1821594116

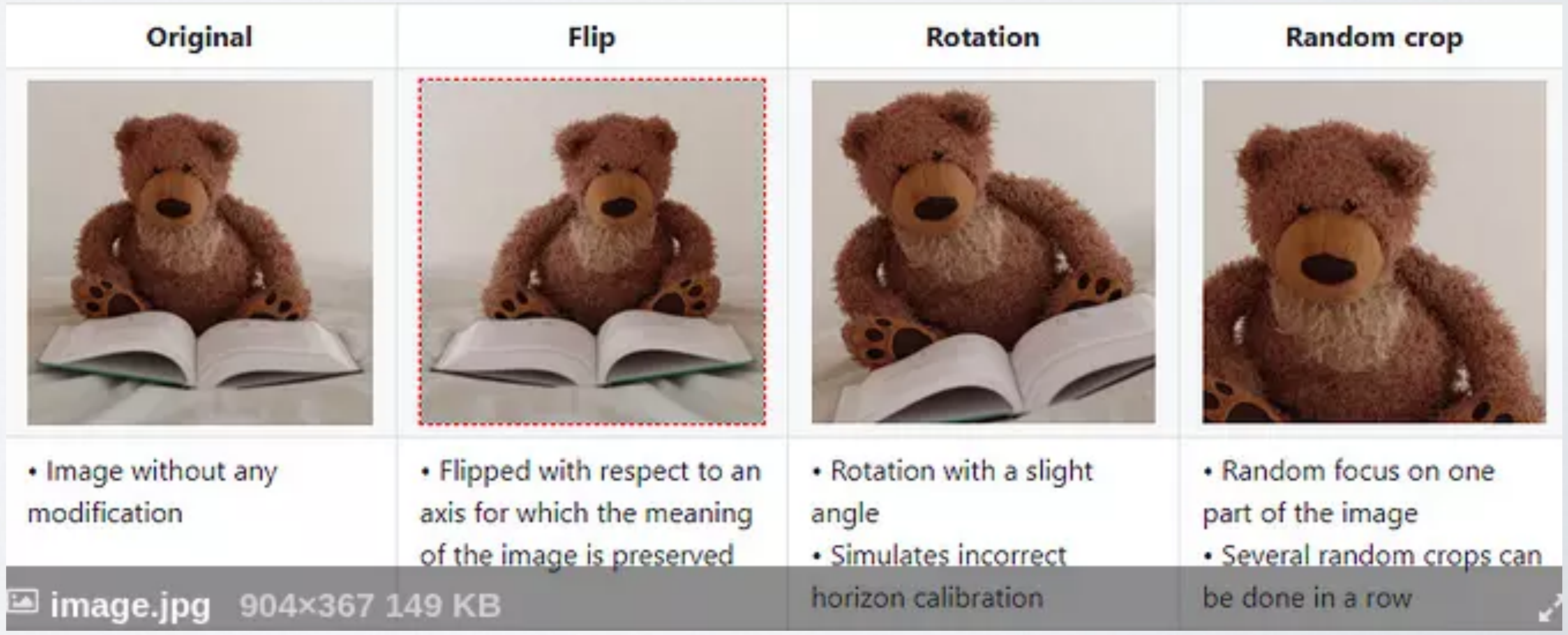
LIMITS OF FULLY CONNECTED DNN

- Problem 1: number of parameters
  - Imagine working with image with 1000x1000 pixels
    - First layer = 1 million parameters * nb layers
    - Hard to compute, hard to converge
- Problem 2: structure

STRUCTURED OBJECT

- An image is a structured object:
  - "Features" have preferential relations with other features
  - i.e., a pixel grid, but
    - Relative pixel positions are important (top, right, left, group of pixels forming a circle, an eye, a face…)
    - Global position is not important (shifted image)

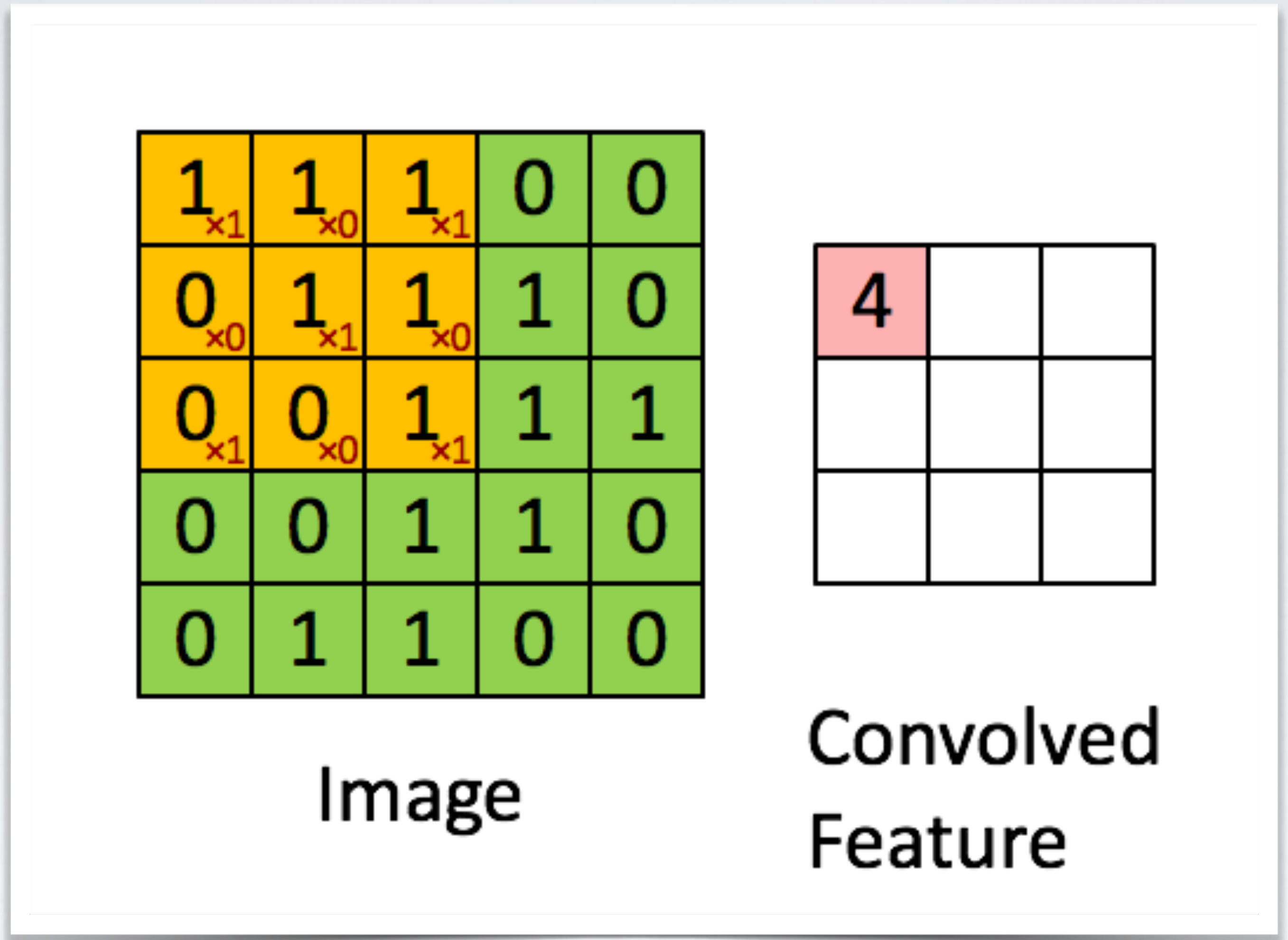
- Using fully connected does not capture efficiently those properties
  - =>Convolution



Original	Flip	Rotation	Random crop
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row

image.jpg 904×367 149 KB

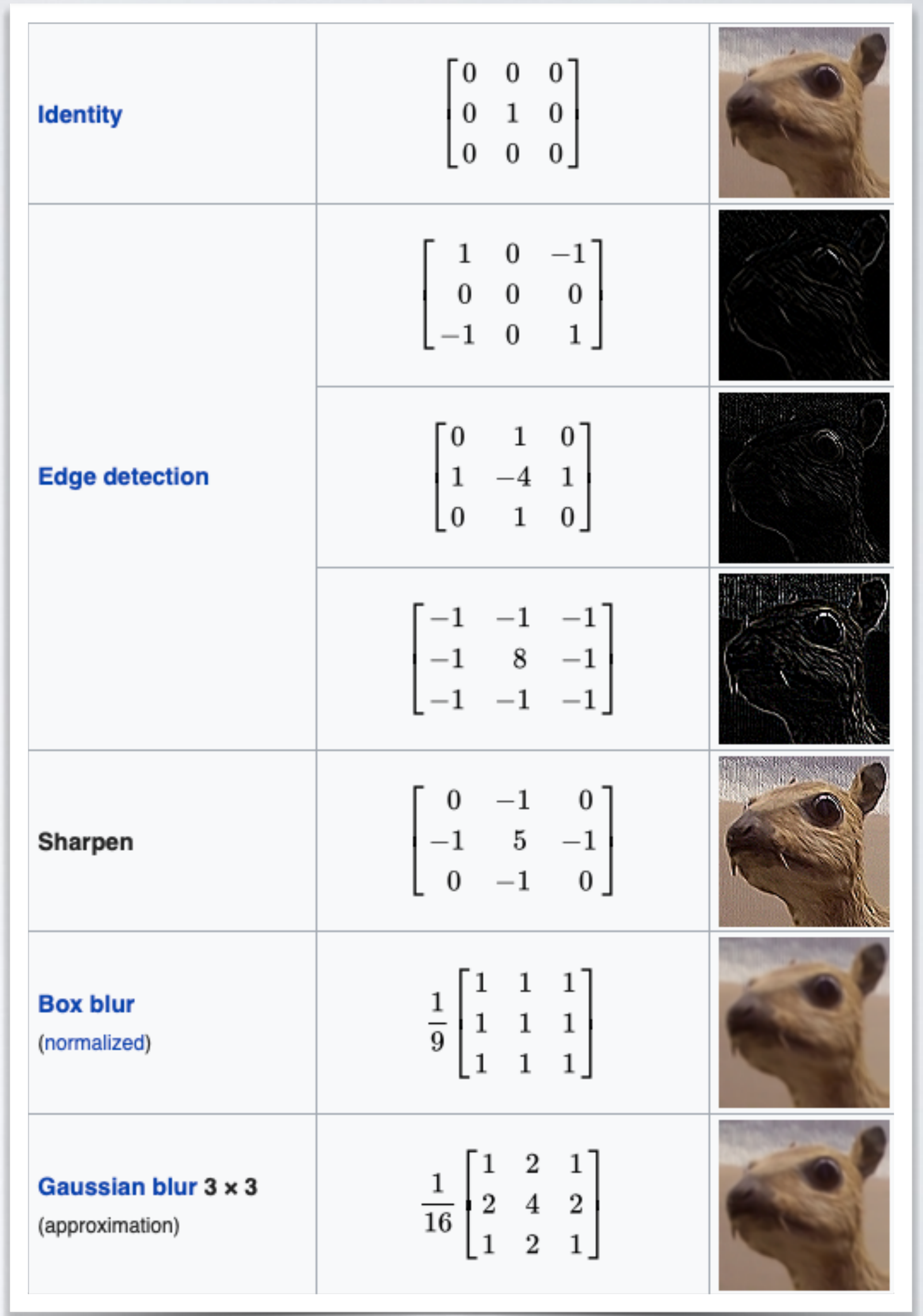
CONVOLUTION



- Extract “features” of “higher level”
  - Pixels => lines, curves, dots => circles, long lines, curvy shapes => eye, hand, leaves => Animal, Car, sky …

CONVOLUTION

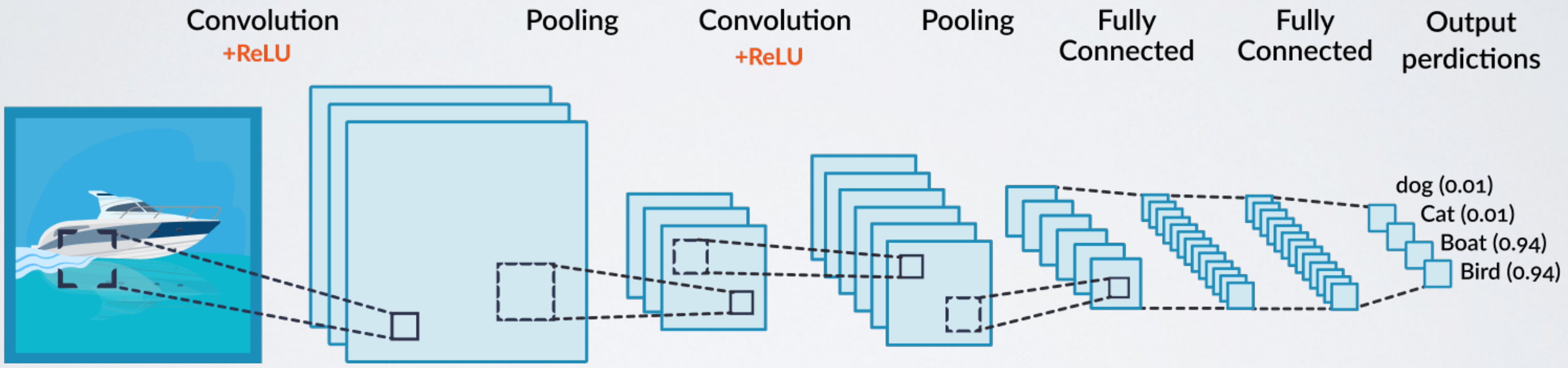
- A convolution is defined by the weights of its kernel
- Which kernel(s) should we use?
- Weights of the kernel can be learnt, too



Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

CONVOLUTIONAL NEURAL NETWORK



Only 9 parameters per layer

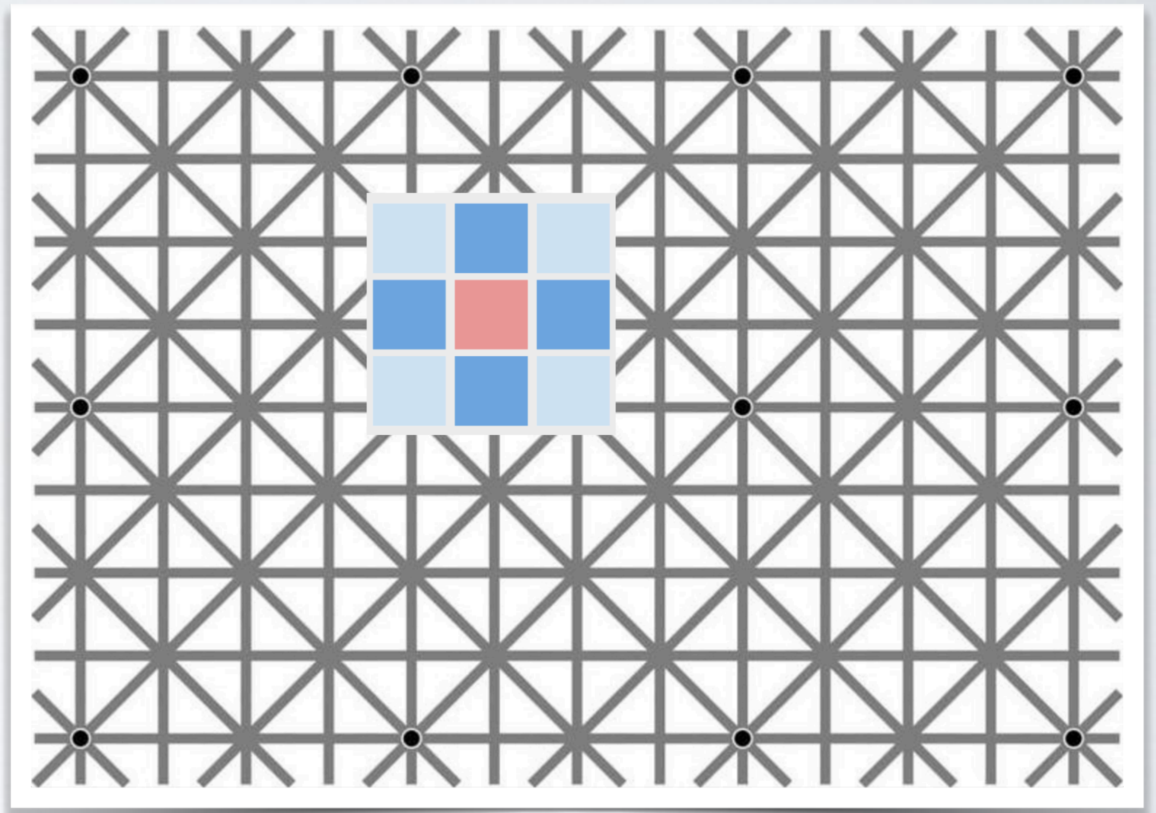
Pooling: e.g., max of the square, reduce size

GRAPH CONVOLUTION

- GCN : Graph Convolutional Network
  - An adaptation of the Convolution used on images to graphs
  - Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

CONVOLUTIONAL NEURAL NETWORK

- Convolution on a picture can be seen as a special case of a graph operation:
  - Combine weights of neighbors
  - With an image represented as a regular grid

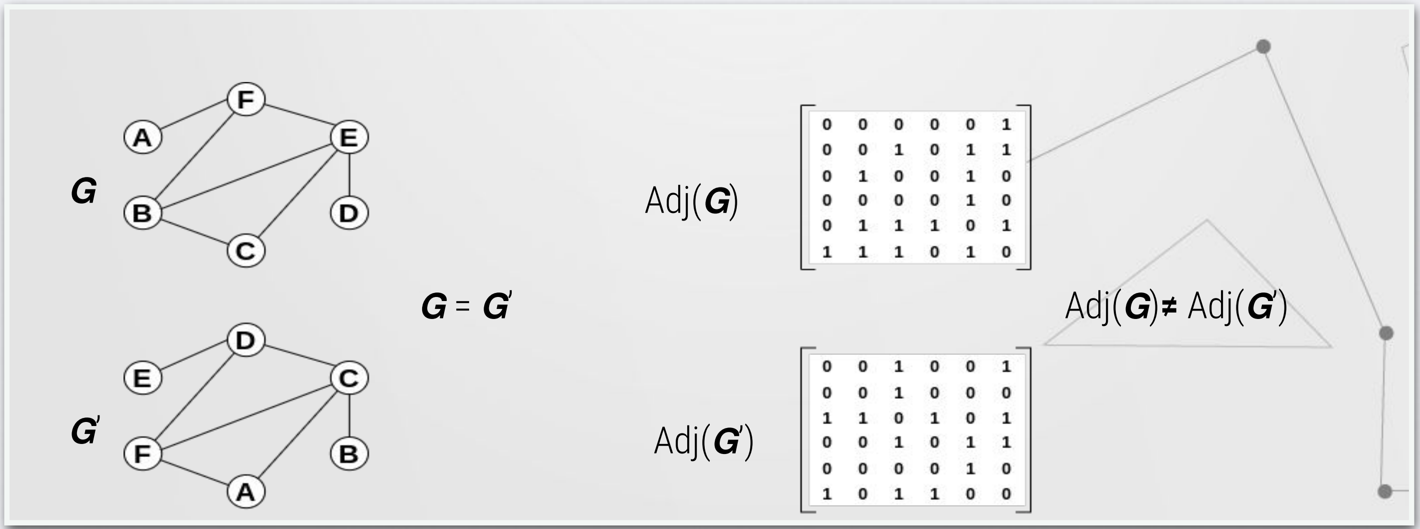


https://www.inference.vc/how-powerful-are-graph-convolutions-review-of-kipf-welling-2016-2/

DIFFERENCES

- In networks, number of neighbors different for each node
  - Impossible to have a “fix” convolution kernel

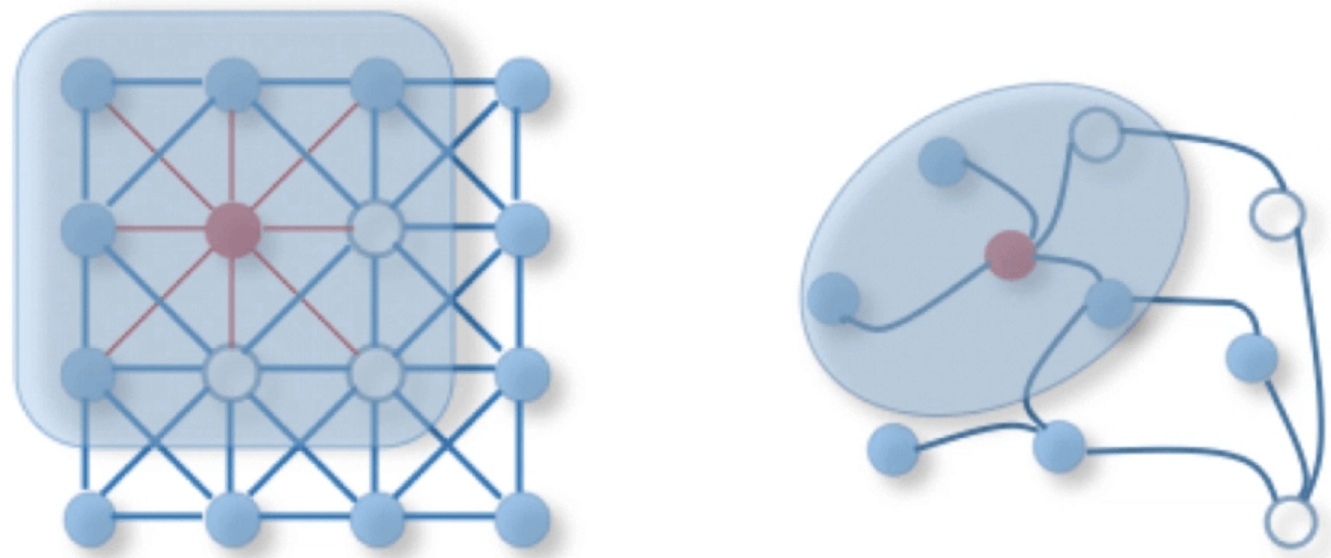
- Matrix representations of images vs graphs
  - Same object, completely different interpreation
  - Graphs: position in the matrix (row, column) has no meaning
    - **Invariance to node ordering**



$$\mathrm{Adj}(G) = \begin{bmatrix} 0&0&0&0&0&1\\ 0&0&1&0&1&1\\ 0&1&0&0&1&0\\ 0&0&0&0&1&0\\ 0&1&1&1&0&1\\ 1&1&1&0&1&0 \end{bmatrix}$$

$$\mathrm{Adj}(G') = \begin{bmatrix} 0&0&1&0&0&1\\ 0&0&1&0&0&0\\ 1&1&0&1&0&1\\ 0&0&1&0&1&1\\ 0&0&0&0&1&0\\ 1&0&1&1&0&0 \end{bmatrix}$$

GRAPH CONVOLUTION



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of graph convolution operation takes the average value of node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Fig. 1: 2D Convolution vs. Graph Convolution.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

GRAPH CONVOLUTION

- **Message passing** interpretation
  - Each node sends its information to its neighbors
  - Nodes "combine" (convolution) their neighbors' information (+ their own) to construct new features

- Tell me who your friends are, I'll tell you who you are

- Can be related to:
  - Information Diffusion on Networks
  - PageRank
  - Label propagation algorithms
  - …

GCN LAYER INTUITION

- Convolution in images:
  - 1)Computes directly a **weighted sum** of neighbors' values
    - Learn the proper weights
  - 2)Often followed by pooling

- Convolution in graphs:
  - Weights cannot be learned directly
  - 1)Average the neighbors' features (pooling-like)
    - Using fix, predefined weights
  - 2)Computes the weighted sum of neighbors' values
    - Learn the proper weights

GCN LAYER INTUITION

- A graph convolution can be understood as a linear (fully connected) layer, with:
  - As input the average features of the neighbors
  - As output a node embedding in the desired number of dimensions
    - Equivalent to the number of neurons in a linear layer
    - But also interpretable as the number of *channels* in Conv layer

GRAPHS ≠ INDEPENDENT ITEMS DATASET

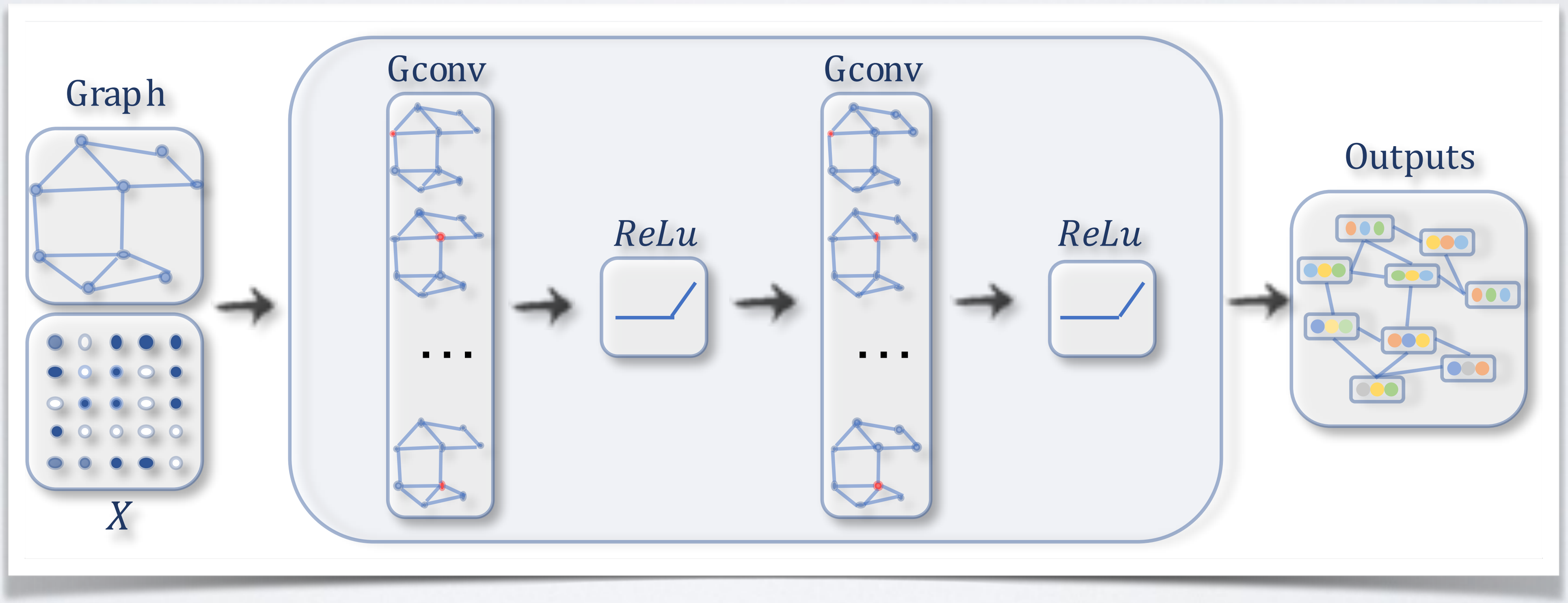
- Graphs are inherently different from image/tabular datasets
  - Images/tabular
    - Each item is independent of the others
    - => We train for each item independently
    - => The test set is composed of new, never-seen items
  - Graphs (general case)
    - A single graph, composed of (connected) nodes
    - =>Each node is treated as an independent item
    - =>But all nodes features are used in training
    - =>Only target can be split in training/test
    - =>"Semi-supervised learning"

GRAPHS ≠ INDEPENDENT ITEMS DATASET

- Example: Network of Twitter users
  - Nodes: users
  - Edges: followers
- Attributes: date joined, likes, geographical position, keywords,…
- Target: Male/Female, Left/Right, etc.
  - We know it for some users, but not all
- Using all users' properties to guess the target for some users, training on the known one

GRAPH CONVOLUTION

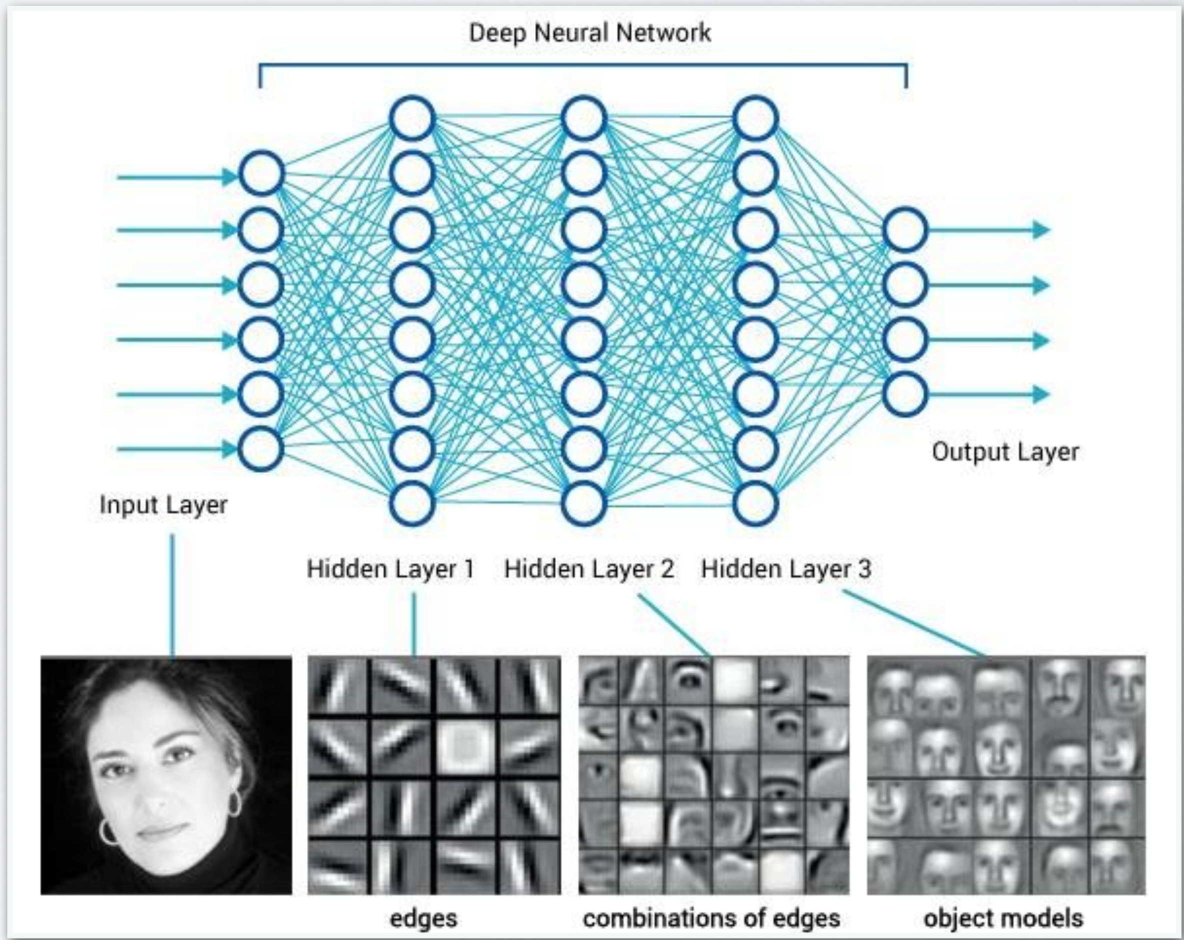
Stacking convolution layers



GRAPH CONVOLUTION

- Each convolution layer allows to depend on nodes farther in the network
    - Layer 1: results depend only on direct neighbors
    - Layer 2:
        - direct neighbors' features are result of Layer 1
        - =>results depends on nodes at distance 1 and 2
    - Etc.

- Similar as convolutions in images



GRAPH CONVOLUTION

- Good news: average distance in real graphs is short
  - 6 degrees of separation
- Even on a large graph, a moderate number of convolutional layers should allow to have impact from most of the graph

GCN EQUATION

GRAPH CONVOLUTION

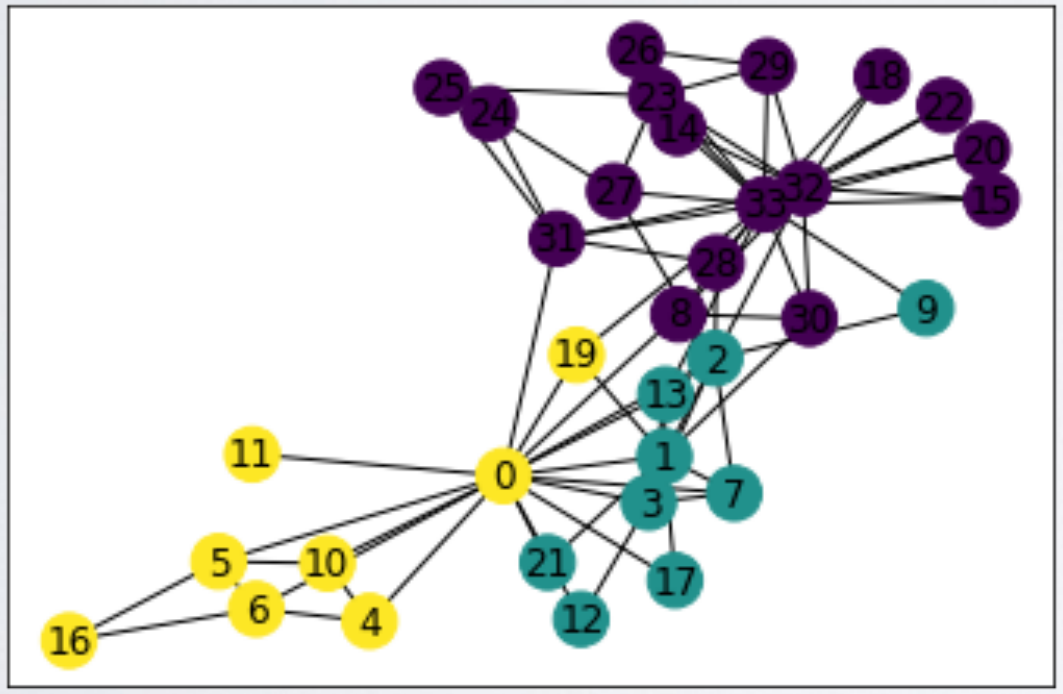
$$H^{(l+1)} = f(H^{(l)}, A)$$

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

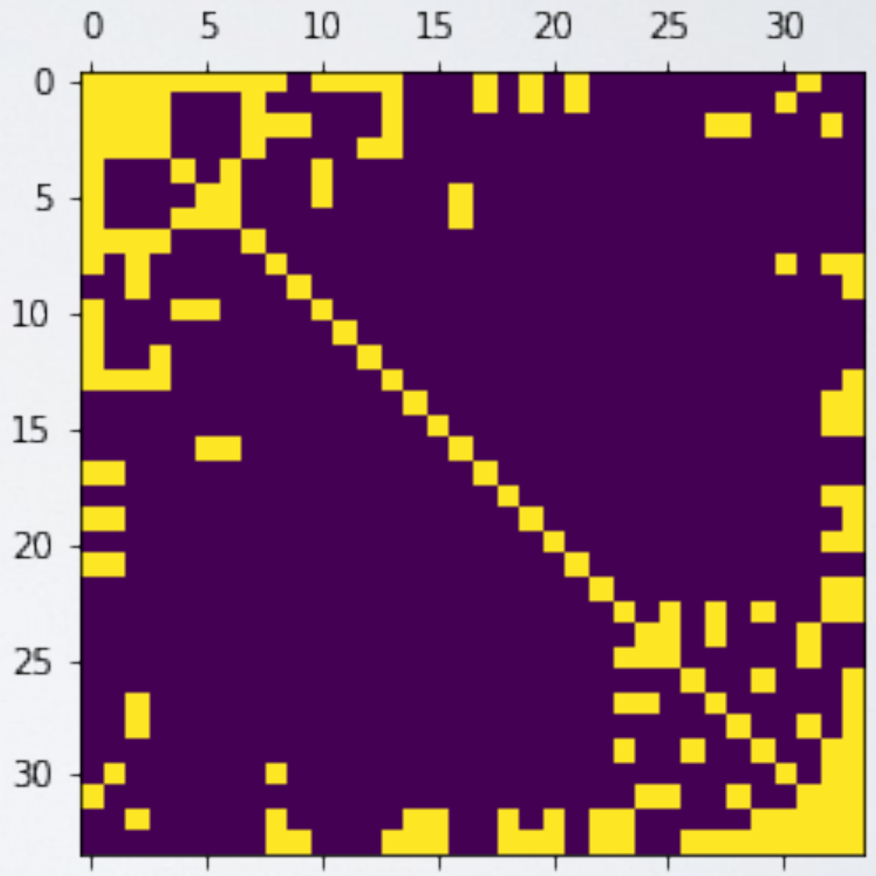
H: node features
A: adjacency matrix ($\hat{A} = A + I$)
l: layer index
D: Degree matrix (degrees on the diagonal)
W: learnable weights
σ: activation fonction (often ReLU)

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

ADJACENCY MATRIX A

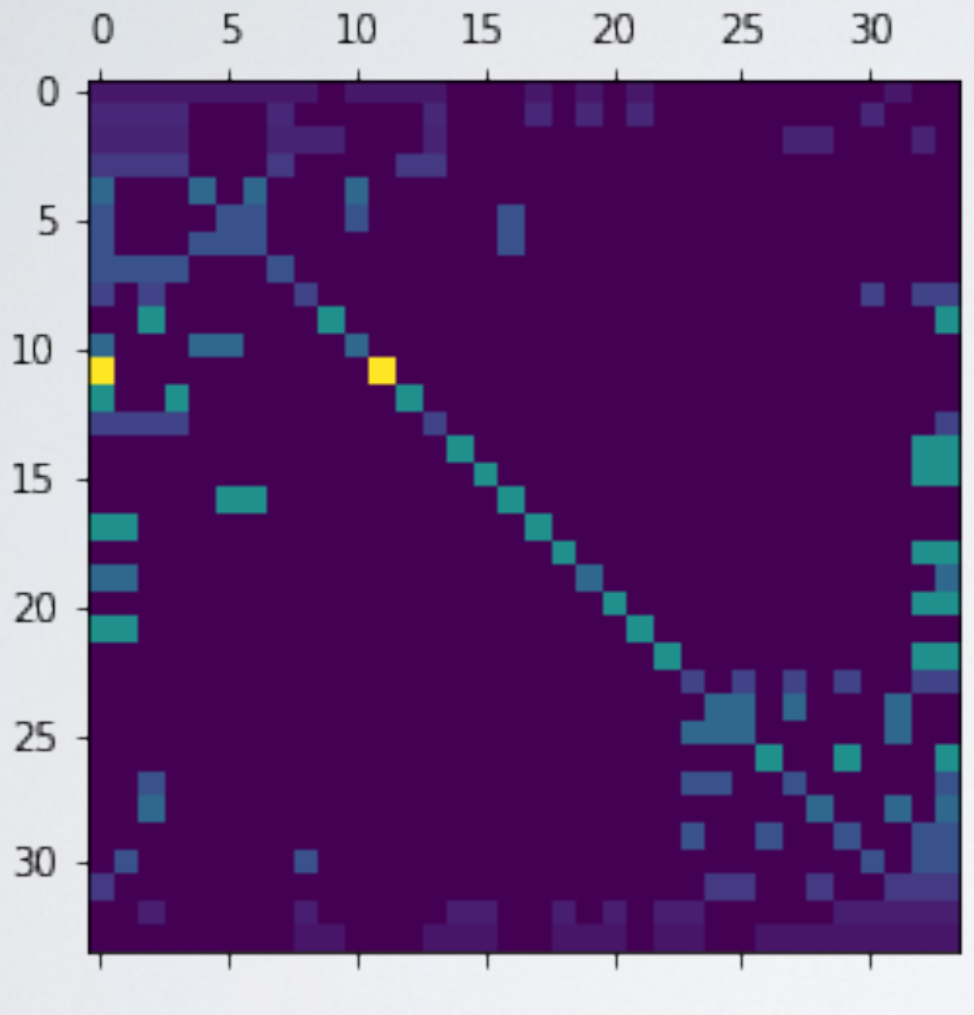


Zackary Karate club
(with communities for reference)



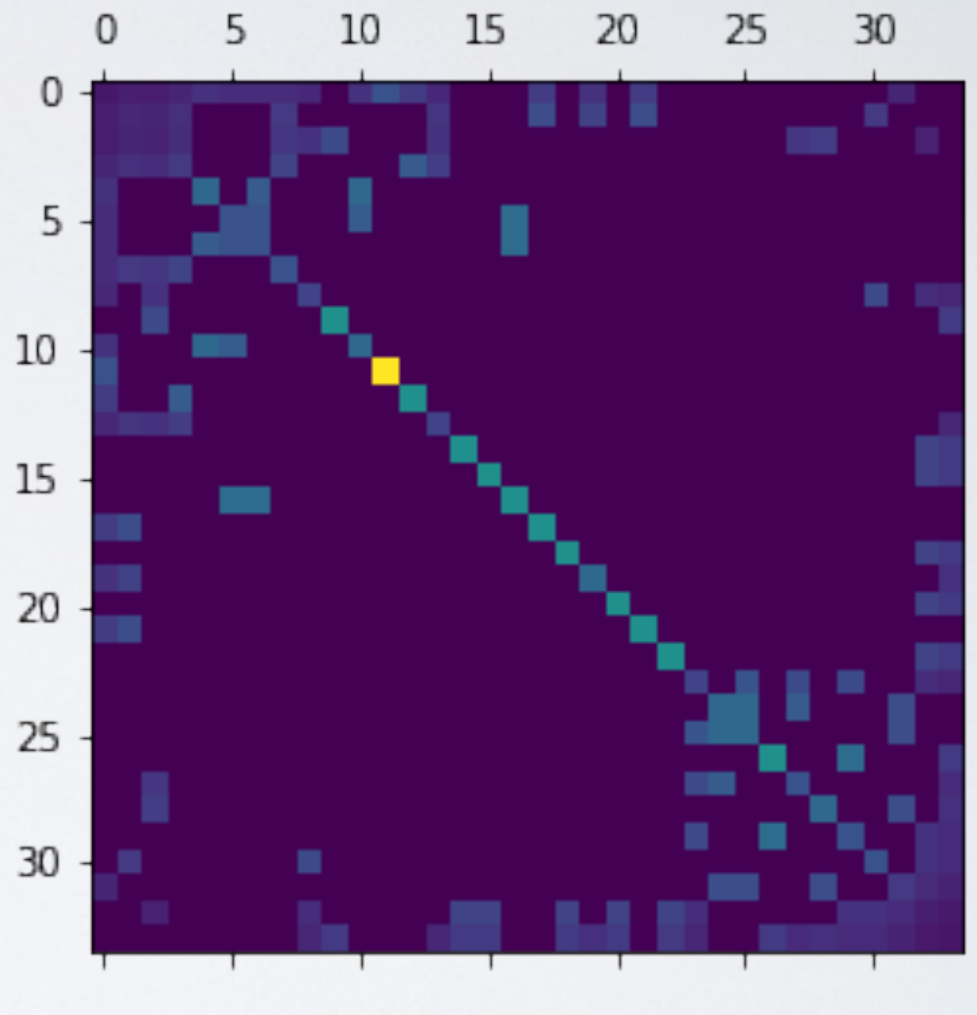
$\hat{A}$

NORMALIZED A



$D^{-1}\hat{A}$

Performs an average



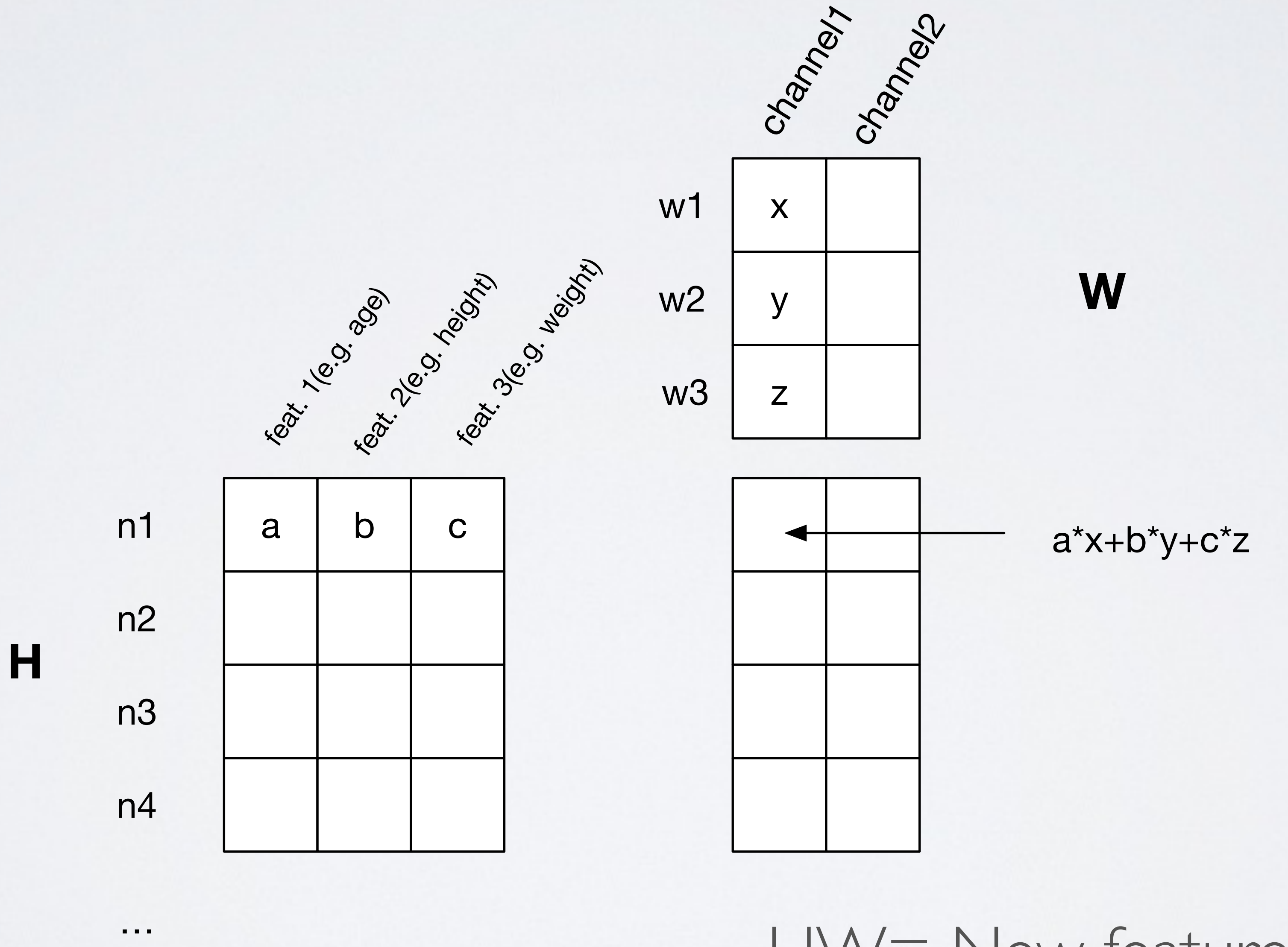
$D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}$

Average weighted by degree

Normalisation of the adjacency matrix

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

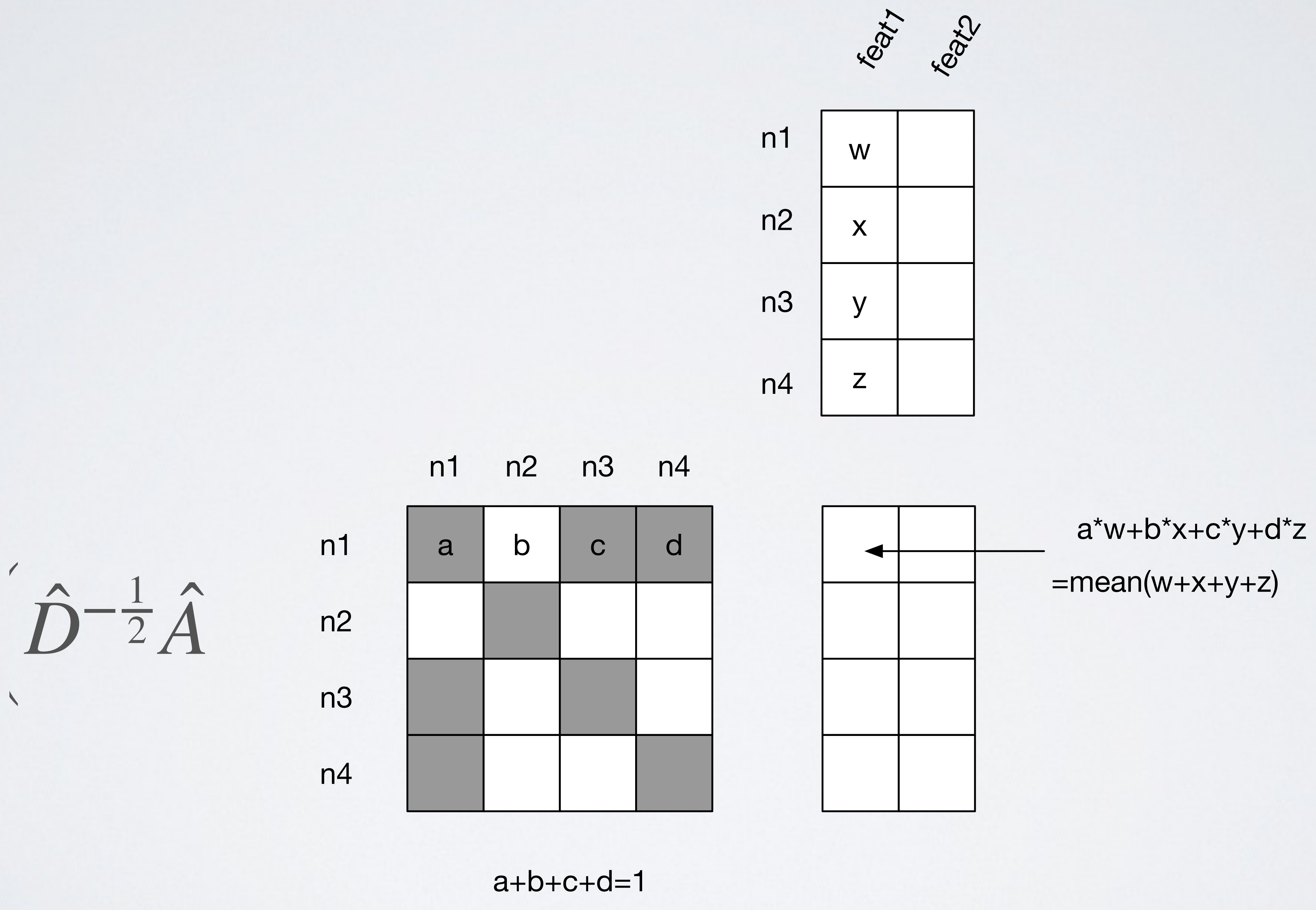
HW=COMBINE FEATURES



HW= New features

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

A(HW)= AVERAGE OVER NEIGHBORS (OF TRANSFORMED FEATURES)



$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

A(HW)= AVERAGE NEIGHBORS EMBEDDING

(AH)W= EMBED AVERAGE OF NEIGHBORS FEATURES

MATRIX MULTIPLICATION IS ASSOCIATIVE

GRAPH CONVOLUTION

- Individual embeddings computed as

- $$h_i^{l+1} = \sum_{j \in N_i} \frac{1}{\sqrt{deg(i)}\sqrt{deg(j)}} h_j^l W^T$$
    - h_j^l embedding of node j in the previous layer
    - Embedding of node i is a weighted sum of its neighbors' attributes multiplied by weights

GCN: STEP-BY-STEP

Without features: Structure only

LAYERS SIZE

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Size of the weight matrices by layer

$$W_0 : d_0 \times d_1$$
$$W_1 : d_1 \times d_2$$
$$\ldots$$
$$W_n : d_n \times d_{n+1}$$

d_0 is the number of features per node in the original network data, d_{n+1} is the number of desired features (usually followed by a normal classifier, e.g., logistic)

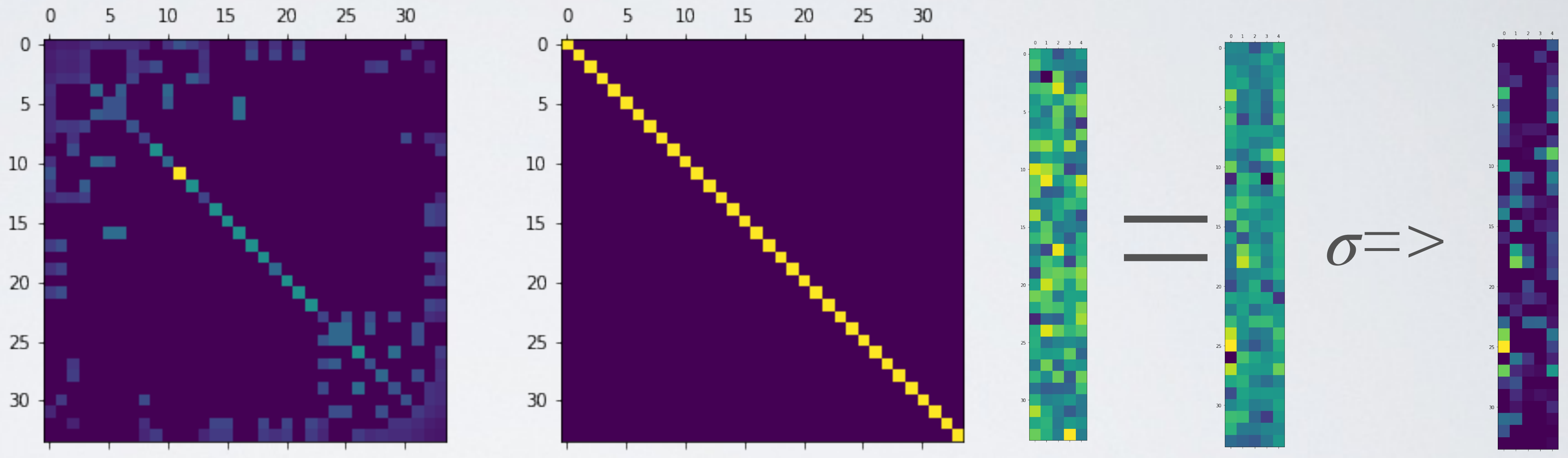
FORWARD STEP

- We can first look at what happens **without weight learning**, i.e., doing only the forward step.
- We set the original features to the identity matrix, $H_0 = I$. Each node's features is a *one hot vector* of itself (1 at its position, 0 otherwise)
- Weights are random (normal distribution centered on 0)
- Two layers, with W sizes $n \times 5, 5 \times 2$

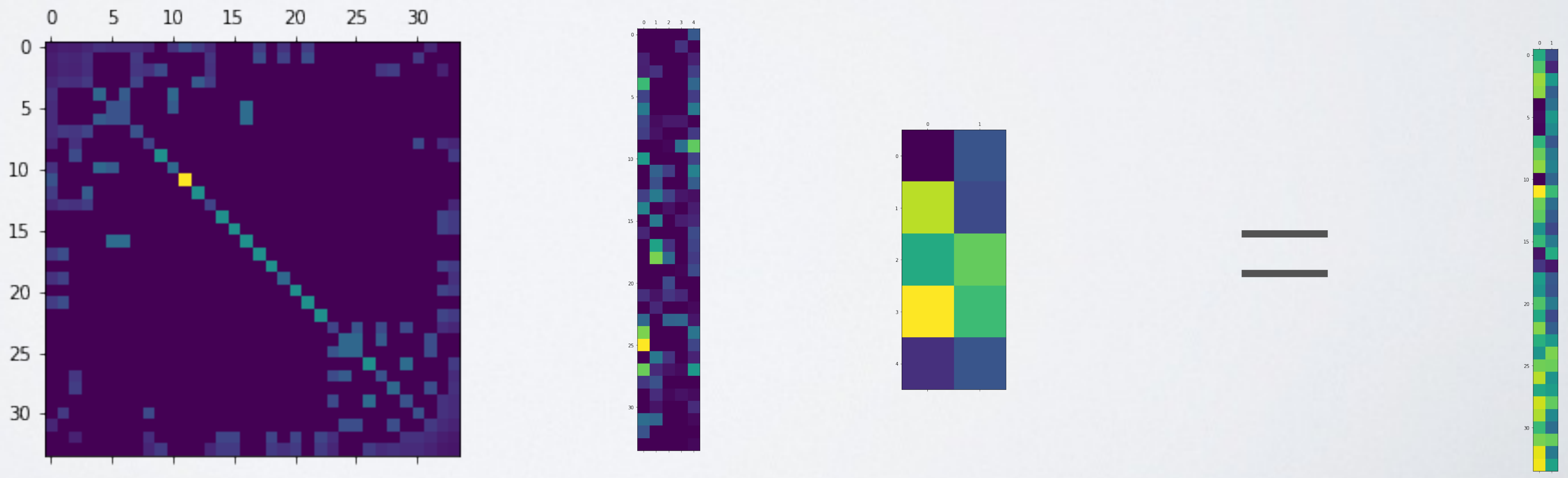
FORWARD STEP

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

L1 = n to 5 features

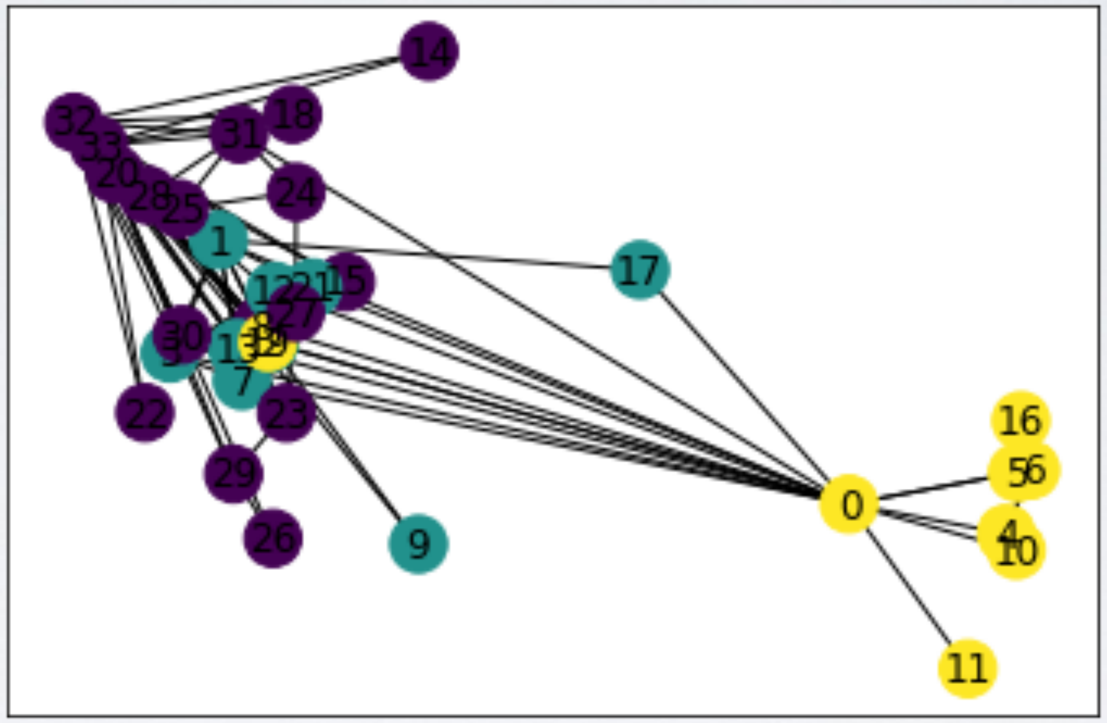


L1 = 5 to 2 features



FORWARD STEP

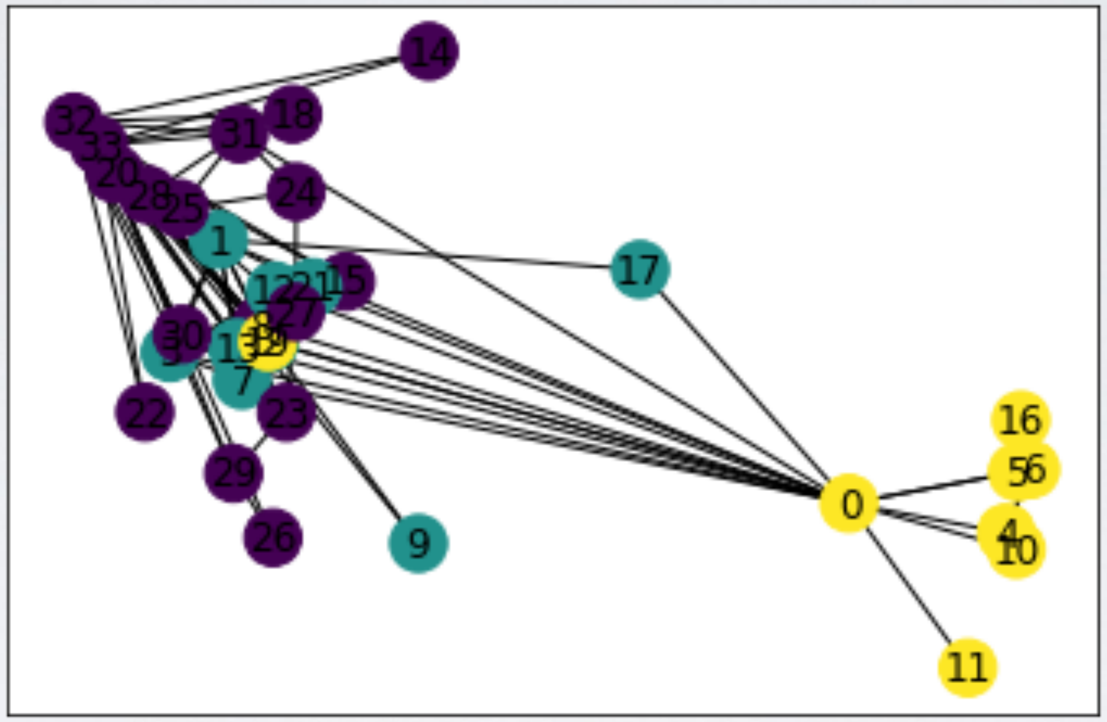
Dimension 2



Dimension 1

Even with random weights, some structure is preserved in the "embedding" (colors=communities)

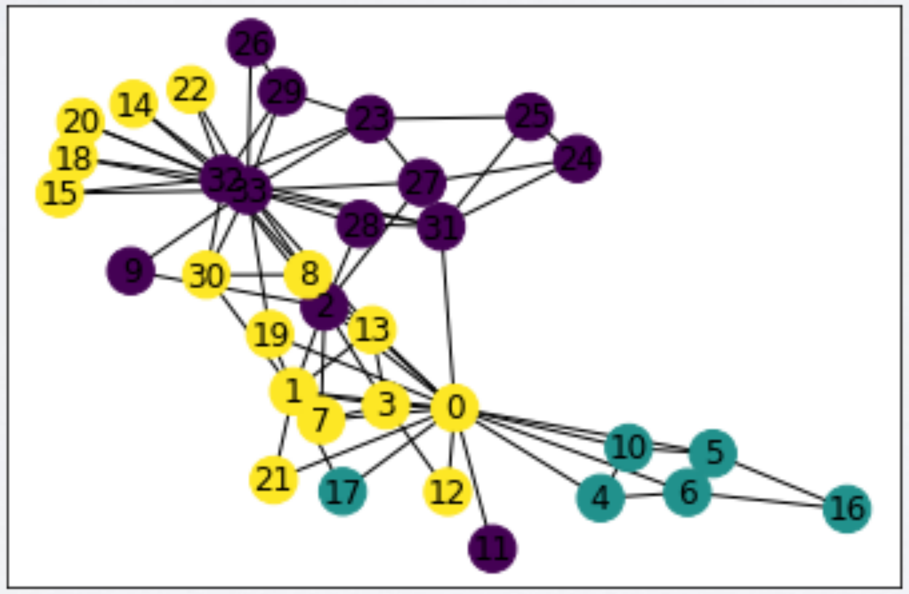
FORWARD STEP



Why is some information preserved?
=>Label propagation mechanism, due to local structure (communities, transitivity…), close nodes receive similar values, convergence to a particular value…

FORWARD STEP

K-means on the 2D "embedding"
(paramater k=3 clusters)



Node positions based on spring layout, colors=clusters

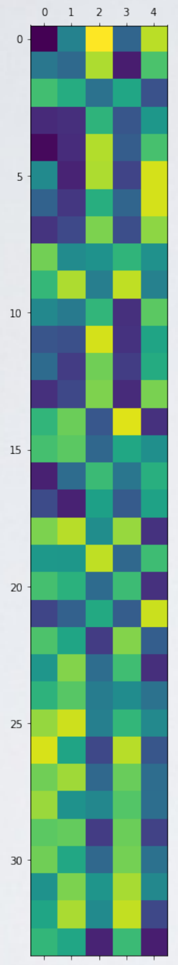
LEARNING STEP

FITTING THE GCN

- We define a "semi-supervised" objective:
  - Labels are known only for a few nodes (the 2 instructors)
  - Choose a loss function for binary classification (logistic…)
  - The loss is computed only for the two instructors
- We run e steps ("epoch") of back-propagation, until convergence

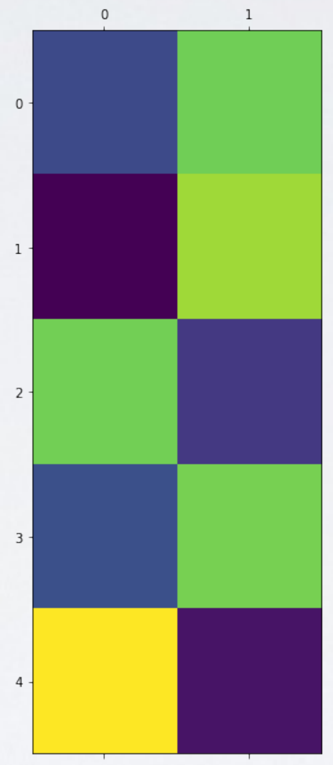
FITTING THE GCN

W_1



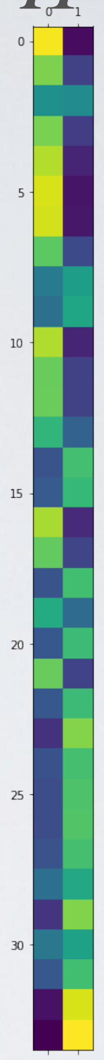
Step1:
Combine one-hot to 5D

W_2



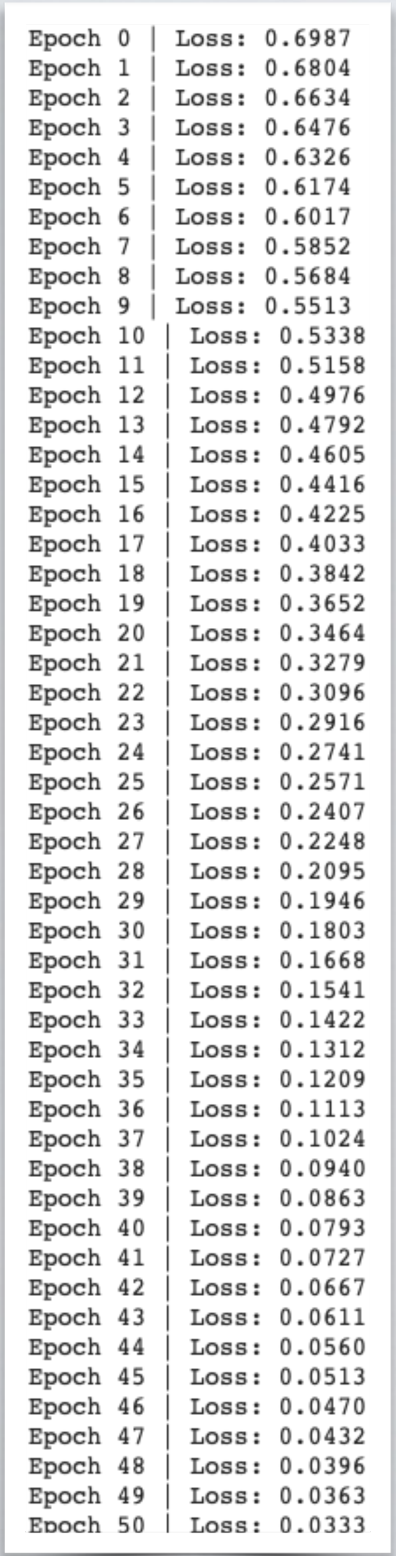
Step2:
Combine 5D to 2D

H

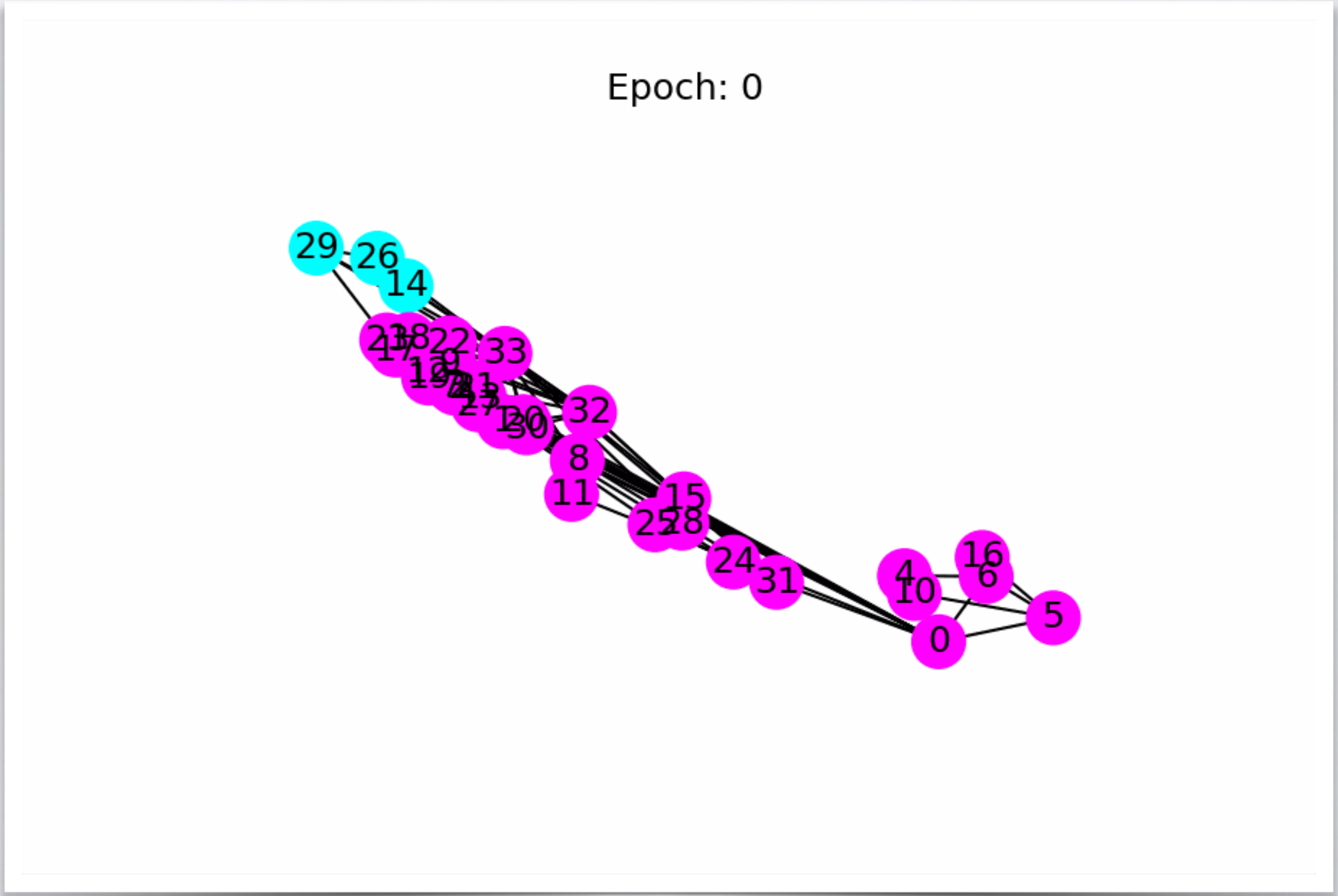


Result:
Computed feature vector
As expected, values for nodes
0 and 33 are opposed

FITTING THE GCN

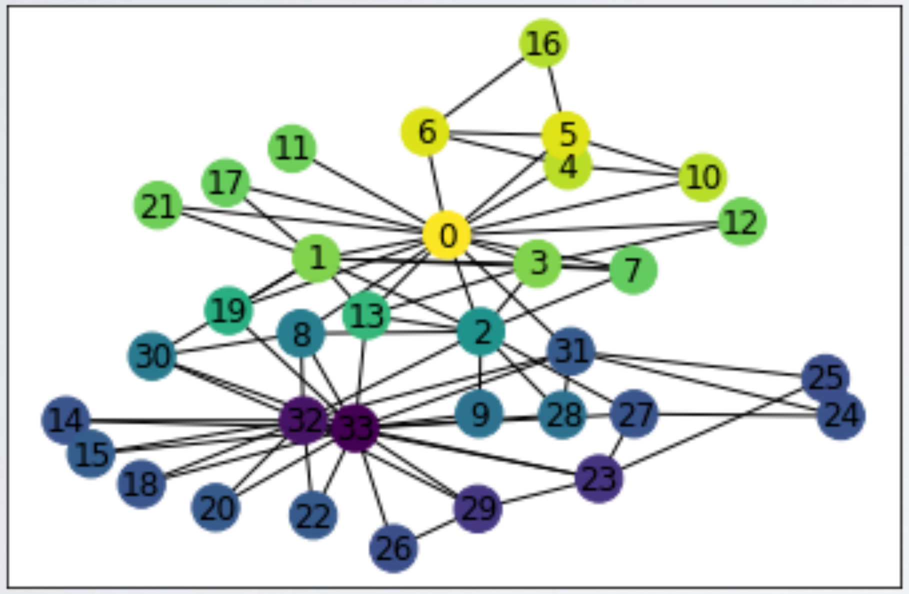


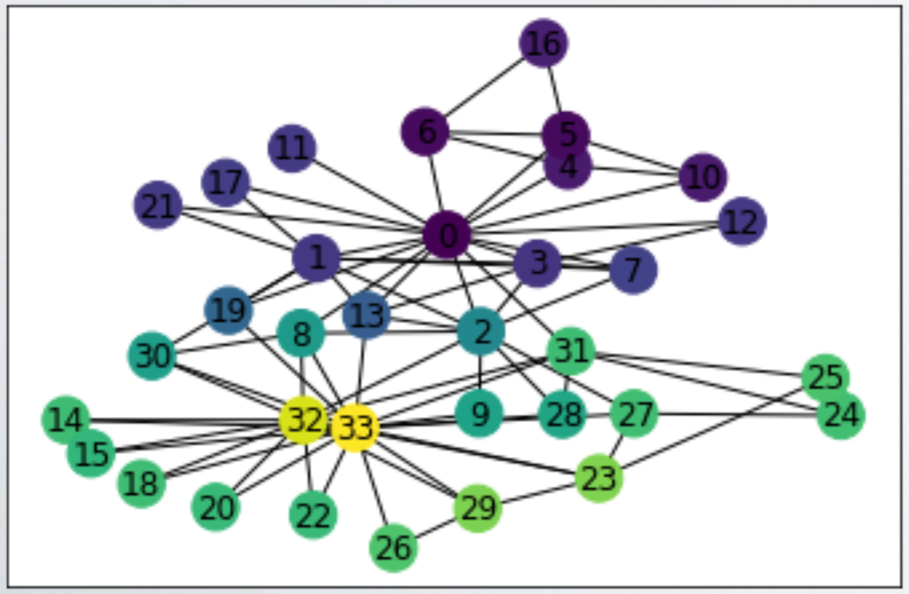
```
Epoch 0 | Loss: 0.6987
Epoch 1 | Loss: 0.6804
Epoch 2 | Loss: 0.6634
Epoch 3 | Loss: 0.6476
Epoch 4 | Loss: 0.6326
Epoch 5 | Loss: 0.6174
Epoch 6 | Loss: 0.6017
Epoch 7 | Loss: 0.5852
Epoch 8 | Loss: 0.5684
Epoch 9 | Loss: 0.5513
Epoch 10 | Loss: 0.5338
Epoch 11 | Loss: 0.5158
Epoch 12 | Loss: 0.4976
Epoch 13 | Loss: 0.4792
Epoch 14 | Loss: 0.4605
Epoch 15 | Loss: 0.4416
Epoch 16 | Loss: 0.4225
Epoch 17 | Loss: 0.4033
Epoch 18 | Loss: 0.3842
Epoch 19 | Loss: 0.3652
Epoch 20 | Loss: 0.3464
Epoch 21 | Loss: 0.3279
Epoch 22 | Loss: 0.3096
Epoch 23 | Loss: 0.2916
Epoch 24 | Loss: 0.2741
Epoch 25 | Loss: 0.2571
Epoch 26 | Loss: 0.2407
Epoch 27 | Loss: 0.2248
Epoch 28 | Loss: 0.2095
Epoch 29 | Loss: 0.1946
Epoch 30 | Loss: 0.1803
Epoch 31 | Loss: 0.1668
Epoch 32 | Loss: 0.1541
Epoch 33 | Loss: 0.1422
Epoch 34 | Loss: 0.1312
Epoch 35 | Loss: 0.1209
Epoch 36 | Loss: 0.1113
Epoch 37 | Loss: 0.1024
Epoch 38 | Loss: 0.0940
Epoch 39 | Loss: 0.0863
Epoch 40 | Loss: 0.0793
Epoch 41 | Loss: 0.0727
Epoch 42 | Loss: 0.0667
Epoch 43 | Loss: 0.0611
Epoch 44 | Loss: 0.0560
Epoch 45 | Loss: 0.0513
Epoch 46 | Loss: 0.0470
Epoch 47 | Loss: 0.0432
Epoch 48 | Loss: 0.0396
Epoch 49 | Loss: 0.0363
Epoch 50 | Loss: 0.0333
```



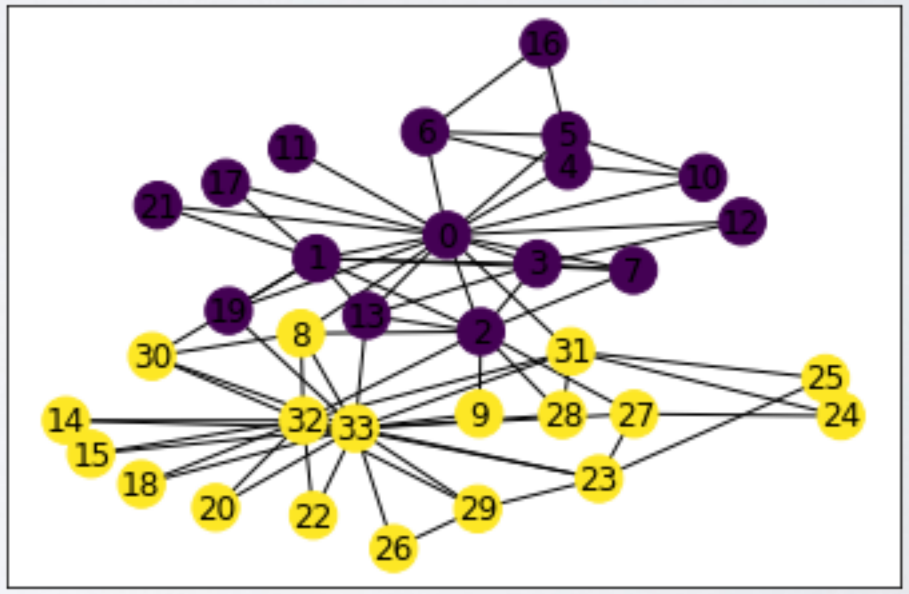
RESULTS

Features values





Highest feature as label



We retrieve the expected "communities"

APPLICATIONS

- Most GNN works consider that we have attributes on nodes, and use supervised tasks
  - Classify bots in social media based on activity
  - Predict congestion in road network based on past traffic
  - Classify protein role in protein/protein interaction network
  - Classify users based on item/user interactions (recommender systems)
  - …

GRAPH AUTOENCODERS

AUTOENCODERS

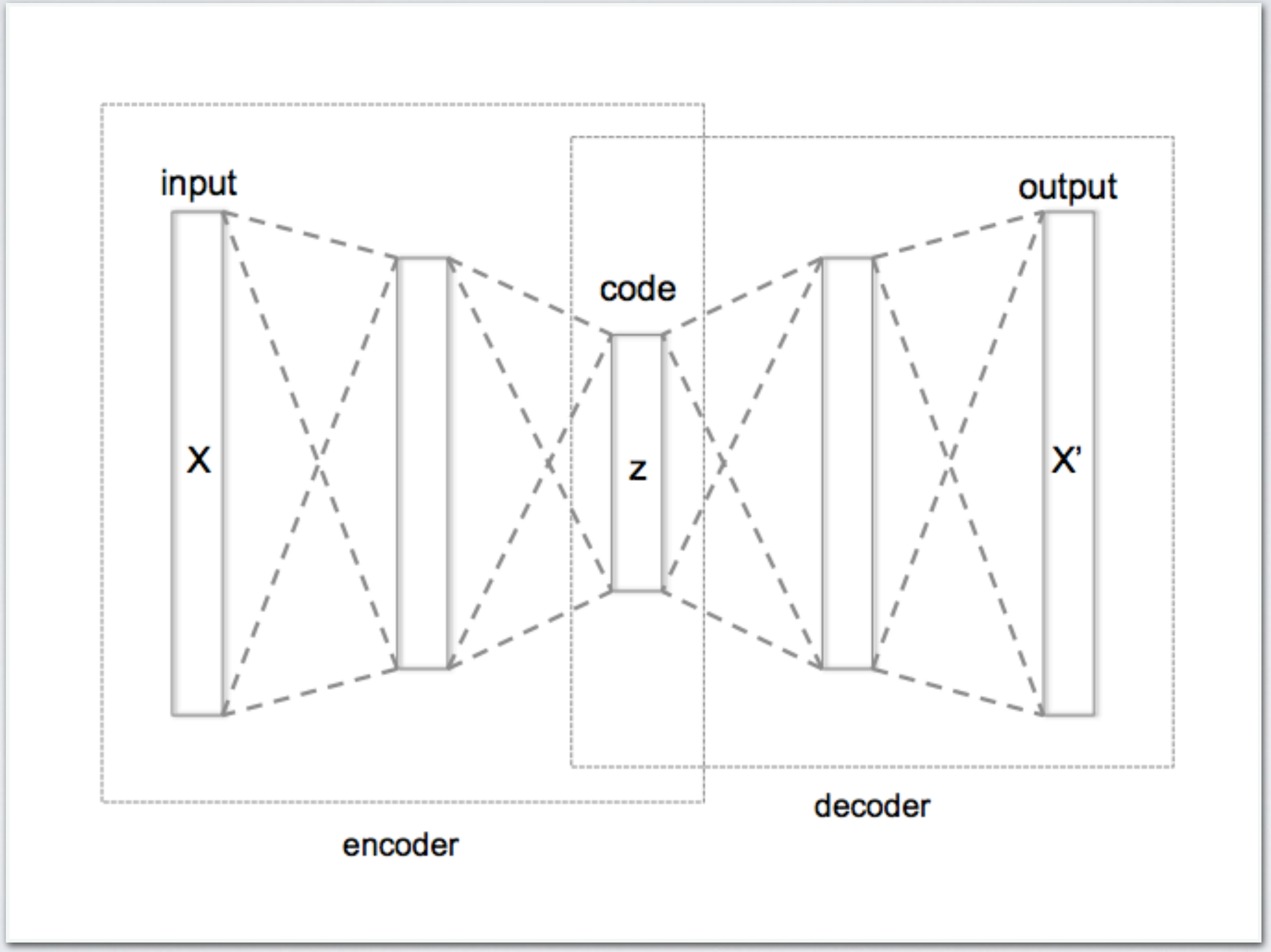
- Autoencoders are mostly used for unsupervised learning using deep neural networks

- Typically, for images.

- Composed of two parts
  - An encoder
    - e.g., a classic sequence of convolutional layers
  - A decoder
    - e.g., an inverse architecture (e.g., the same layers in inverse order)

- In the middle is the "embedding", what we are interested in
  - Constrained to be small

AUTOENCODERS



AUTOENCODERS

- The objective is to
  - Encode a complex object
    - e.g., a 3 color layers, 256 x 256 image
  - Into a small-dimensional vector
    - e.g., vector of size128

- Such that these vectors allow to reproduce the output with minimal loss of information

- Many applications:
  - Visualization (like PCA/tSNE)
  - Downstream task (these vectors can be used for classification, etc.)
  - Generate variations (Generative image models…)

GRAPH AUTOENCODERS

- Same principle, but with graphs :)
- Classic architecture[1]:
  - Encoder: GCN layers (e.g., 2 layers)
  - Decoder: Dot product between embeddings (+activation)
  - Minimize the binary cross entropy between input and output adjacency matrices
- =>Compute vectors for each node
  - such that their dot product is
    - Close to 1 if they are connected (parallel => similar vectors)
    - Close to 0 if they are not (orthogonal => different vectors)

[1]Kipf, T. N., & Welling, M. (2016). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

GAE INTEREST

- Node embedding capturing the structure of the graph
  - Community detection?
- Obtain a parsimonious model of the graph
  - Relation with SBM, or even more with RDPG (Random Dot Product Graph), a related inferential framework to discover embedding such as edge probability depends on dot products between node vectors.
- Link prediction (next)

LINK PREDICTION

LINK PREDICTION

- Observed network: current state
- Link prediction: What edge
  - Might appear in the future (*future link* prediction)
  - Might have been missed (*missing link* prediction)
- Many applications
  - Recommender systems
  - Drug/healness prediction, …

LINK PREDICTION

- Classification objective
  - Binary classes: edge/No edge
  - Usually, evaluation based on class probability
    - AUC(ROC), AP (Average Precision)…
- Evaluation process
  - Hide some of the edges in the graph
  - Check that
    - Training on the remaining edges
    - We predict well the removed ones

LINK PREDICTION

- Classic methods
  - Common Neighbors
  - Adamic Adar
  - …
    - =>Work only on nodes at distance two

- Advanced methods
  - Graph embedding (DeepWalk, Node2Vec, etc.)
    - Use dot product of embedding as score, or other variants, e.g., training a classification on vectors
  - Community structure, random walks

  - =>Do not take node features into account

LINK PREDICTION

- Using GAE
  - The objective of GAE is to reconstruct the graph, i.e., to predict which edge is present or not =>Directly a link prediction objective
  - GAE final step: dot product of embeddings
- Edge prediction score: result of the dot product of node vectors

LINK PREDICTION

- Using directly a GNN
  - GNNs produce node embeddings in the output
  - We need to combine node embeddings
- Two (main) solutions
  - Create a combined vector from two independent vectors, and add a linear layer for classification
  - Use directly a vector-to-scalar operation

LINK PREDICTION

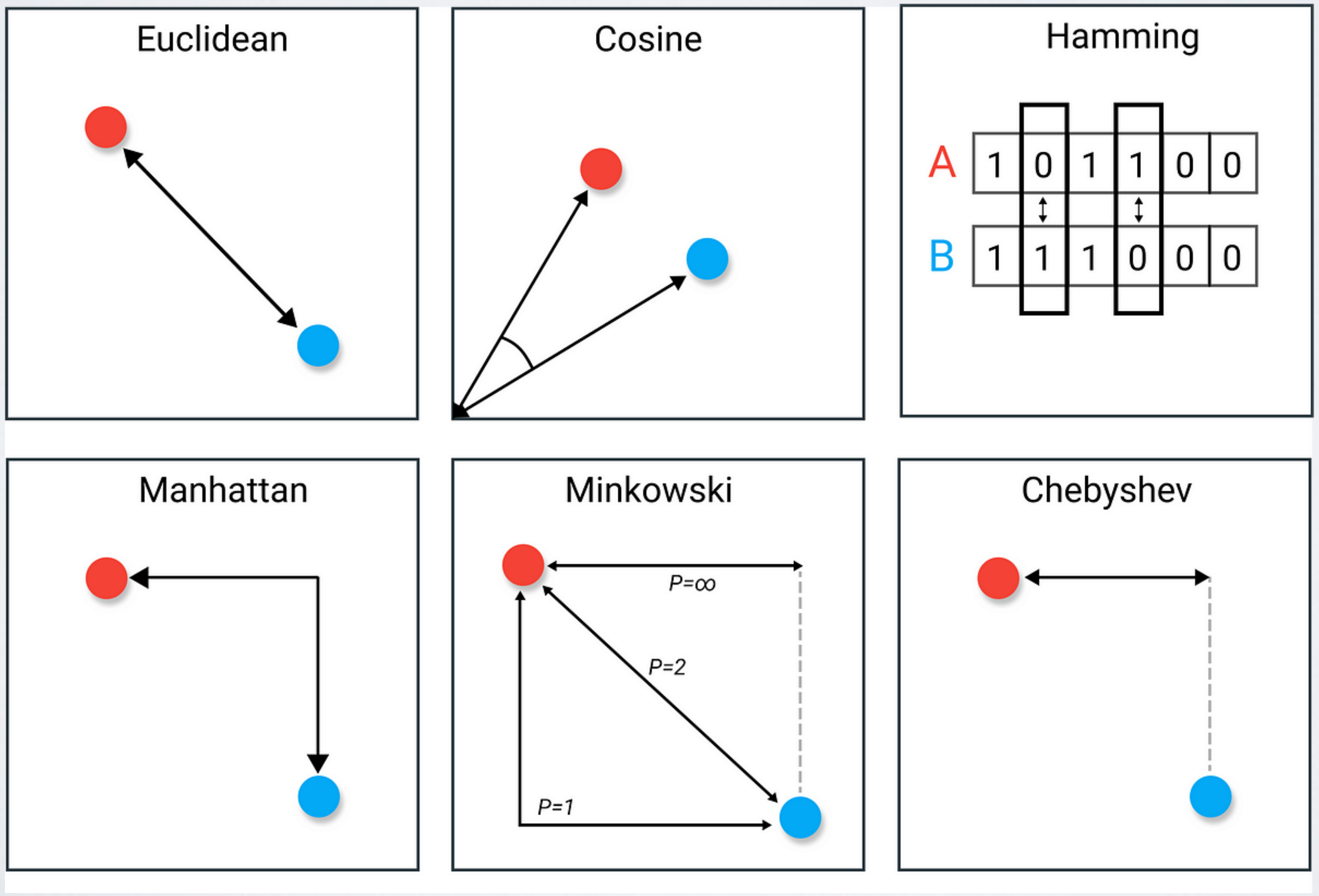
- Combining two node vectors into a node-pair vector
  - Vector concatenation [x1,x2] [x3,x4]=>[x1,x2,x3,x4]
  - L1 difference [x1,x2][x3,x4]=>[x1-x3 , x2-x4]
  - Hadamard Product [x1,x2][x3,x4]=>[x1*x3 , x2*x4]
  - …

- Followed by a classification task on this new vector

LINK PREDICTION

- Combining two node vectors into a scalar

dot product≈unnormalized cosine similarity



VGAE

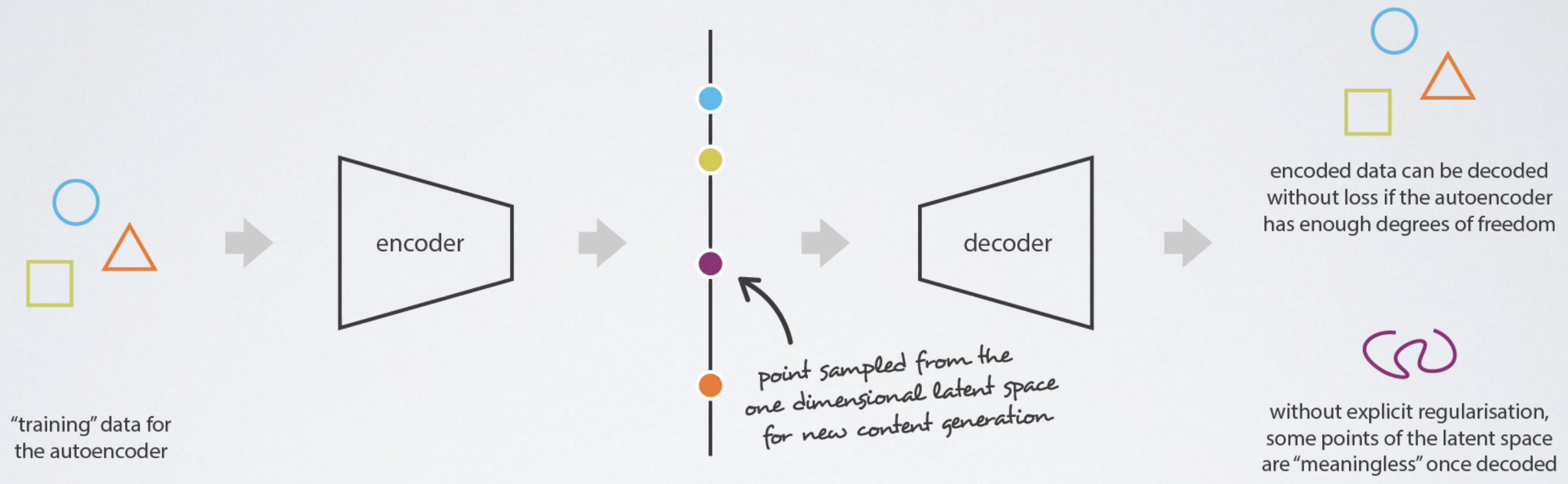
VARIANT: VGAE

- VAE : Variational AutoEncoder
  - Popular improvement over classic AutoEncoder

- Limits of Autoencoders:
  - Embedding space is often poorly structured
    - Poor **continuity**: The "middle" vector between two vectors (v1,v2) do not correspond to a middle image between the two corresponding to v1/v2
    - Poor **completeness**: Space seems "sparse": many vectors correspond to nothing meaningful

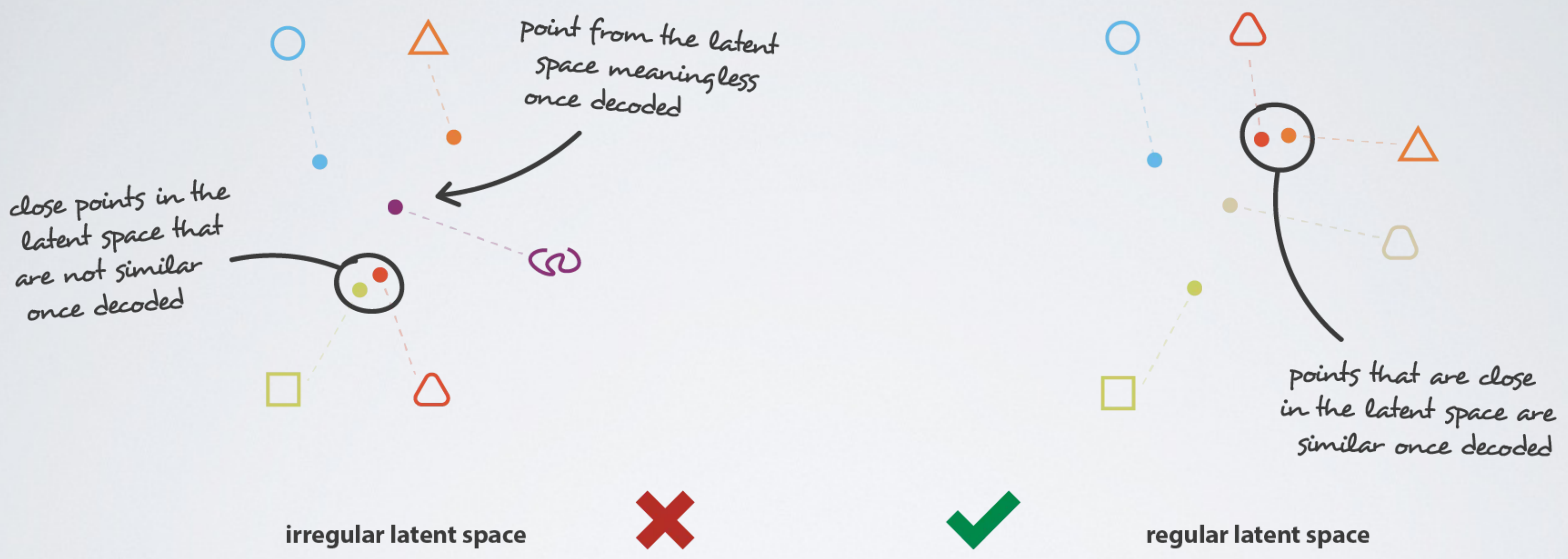
- VAE solution:
  - Instead of encoding an input as a single point, we encode it as a distribution over the latent space

VAE


"training" data for
the autoencoder
encoder
decoder
point sampled from the
one dimensional latent space
for new content generation
encoded data can be decoded
without loss if the autoencoder
has enough degrees of freedom
without explicit regularisation,
some points of the latent space
are "meaningless" once decoded

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VAE


point from the latent space meaningless once decoded
close points in the latent space that are not similar once decoded
points that are close in the latent space are similar once decoded
irregular latent space
regular latent space

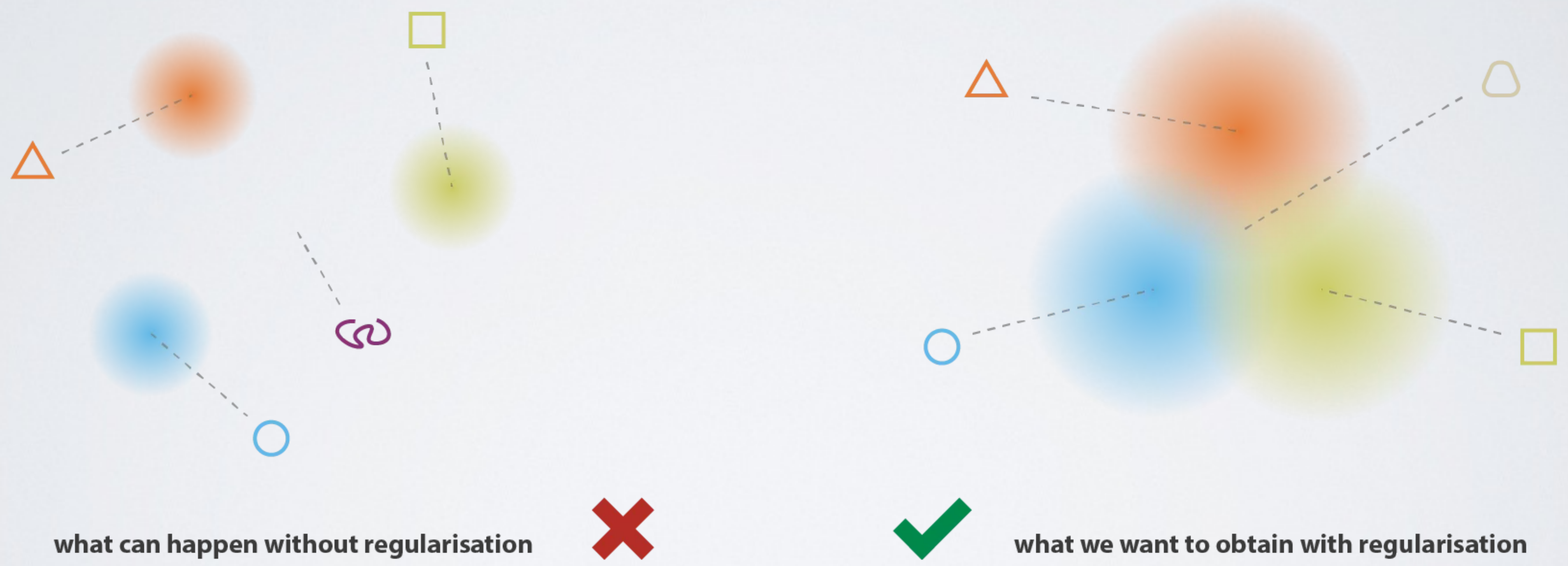
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VAE

- The model is trained as follows:
  - 1)the input is encoded as gaussian **distribution** over the latent space
  - 2) a point from the latent space is sampled from that distribution
  - 3) the sampled point is decoded and the reconstruction error can be computed
  - 4) finally, the reconstruction error is backpropagated through the network

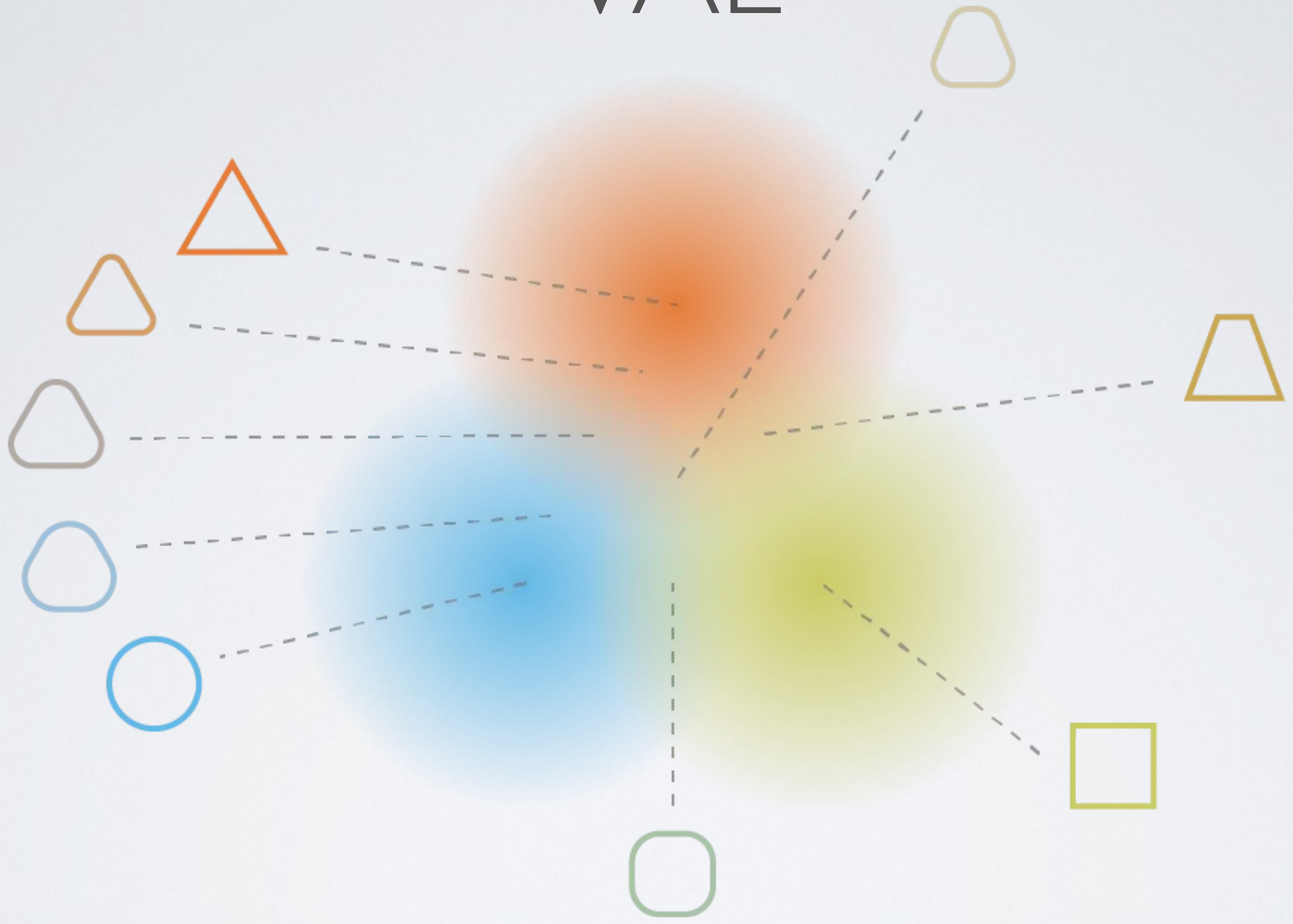
VAE

Regularization: trade-off between best fit to data and distance between each gaussian and a standard gaussian(centered, unit variance)


what can happen without regularisation
what we want to obtain with regularisation

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VAE



https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VGAE

- Simple adaptation to graphs, i.e., a classic graph autoencoder in which the encoding part is replaced by Variational mechanism.

- In practice:
  - Layer 1: normal GCN
  - Layer 2: two parallel GCN layers
    - One to learn the centroid
    - One to learn the variance (diagonal of the covariance matrix)
    - =>For each node, instead of having 1 vector of size d, we have two vectors of size d
  - To decode, we take a random point from the multivariate gaussian

GOING FURTHER

TRANSDUCTIVE / INDUCTIVE

- Transductive
  - What we discussed until now:
    - We have access to the whole graph at training time
    - We just don't see all the labels (test, prediction)
- Inductive
  - Train on a set of nodes/graphs
  - Results can be applied to unseen nodes/graphs
    - A GCN layer can be trained on multiple (sub)networks, and learned weights used on a new scenario (but not very efficient)
    - GraphSAGE=>Works for each node on a local graph centered on the node, by sampling a fixed number of neighbors. Transform the graph problem in a more classic problem.

MULTI-PARTITE GRAPHS

- Nodes of multiple types:
  - Items/Users
  - Drug/illness
  - …
- Each type of node has their own attributes
  - Cannot learn a single GCN layer
- =>Learn 2 independent layers
  - User attributes to Item attributes
  - Item attributes to User attributes

GAT

Graph ATtention networks

SELF-ATTENTION MECHANISM

- Mechanisms coming mostly from Language models
  - Transformers (as in GP**T**) are a particular type of self-attention

GRAPH ATTENTION

- In the normal GCN, a limit is the fix rule used to combine the neighbors attributes (weighted average)
  - $$h_i^{l+1} = \sum_{j \in N_i} \frac{1}{\sqrt{deg(i)}\sqrt{deg(j)}} h_j^l W^T$$
- *Graph attention* principle is to allow each node to "choose" what "attention" to give to each neighbor
  - $$h_i^{l+1} = \sum_{j \in N_i} \alpha_{ij} h_j^l W^T$$
    - α_{ij} attention from i to j