Universität Konstanz

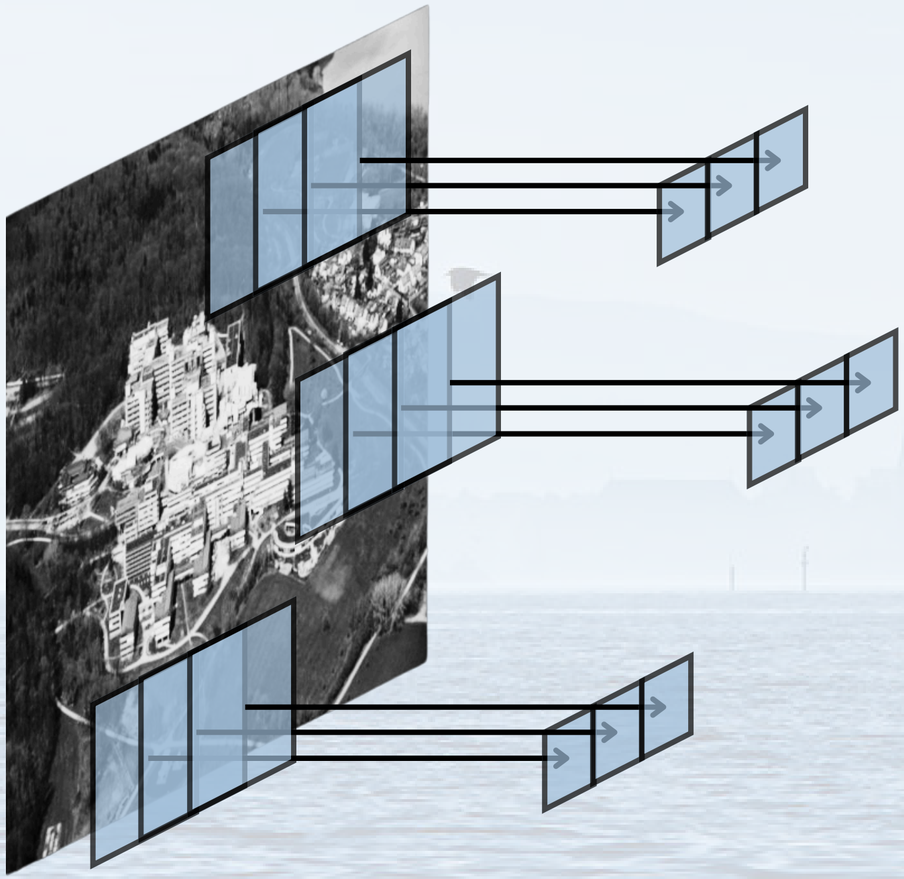
# 06 | Graph Neural Networks

Giordano De Marzo

https://giordano-demarzo.github.io/

## Deep Learning for the Social Sciences

# Learning Filters



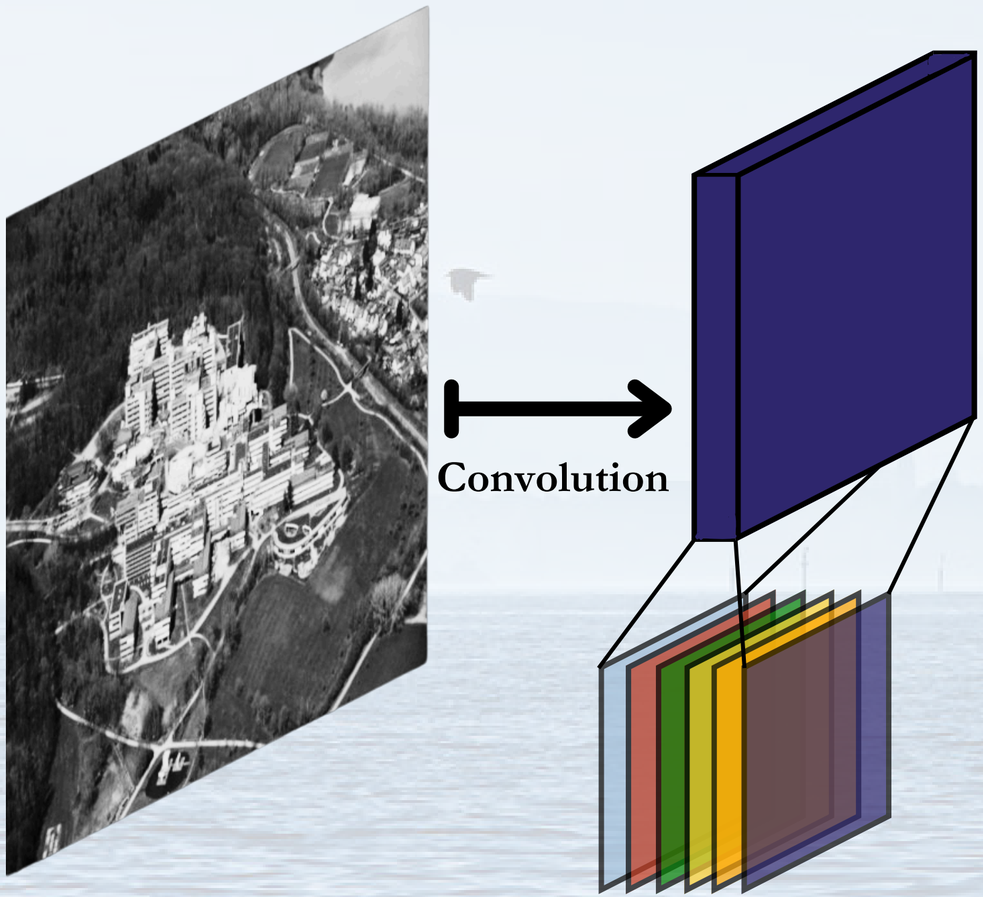
Convolutional Neural Networks are based on a very simple idea:

**The parameters we want to learn are the numbers inside the filter**

This is a very powerful approach:

- we reuse parameters because the same filter is applied to all the locations of the images
- the filter captures local geometrical features, such as edges, independently of where they are placed in the image
- the output is another (smaller) matrix that can be further processed to extract more complex feattures

# Convolutional Layer




A convolutional layer consists of F different filters:

- it takes in input an image of dimension WxH
- it applies each filter to the input image
- the result of each filter is passed through an activation function (ReLU)
- the output consists of F stacked images , one for each filter
- each output image captures different features of the original image
- the hyperparameters are: number of filters, filters size, stride and padding

Universität Konstanz

# Outline

1. Graph Theory
2. Machine Learning on Graphs
3. Convolution on Graphs
4. Graph Convolutional Neural Networks
5. Applications


Universität
Konstanz

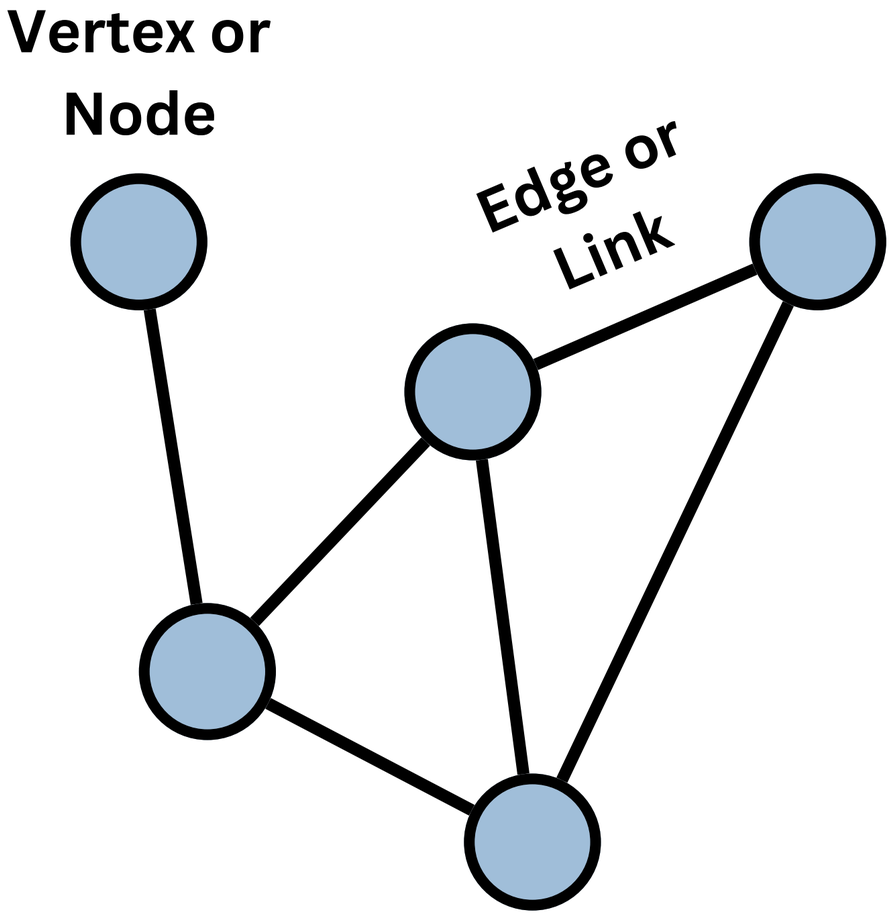
# What is a Graph

A Graph G(V, E) is a set of vertices or nodes V and and edges or links E

- nodes represent entities in the system (eg. people on a social network)
- edges represent connections among the nodes (eg. friendship in a social network)

We denote by

- N the number of nodes
- E the number of links



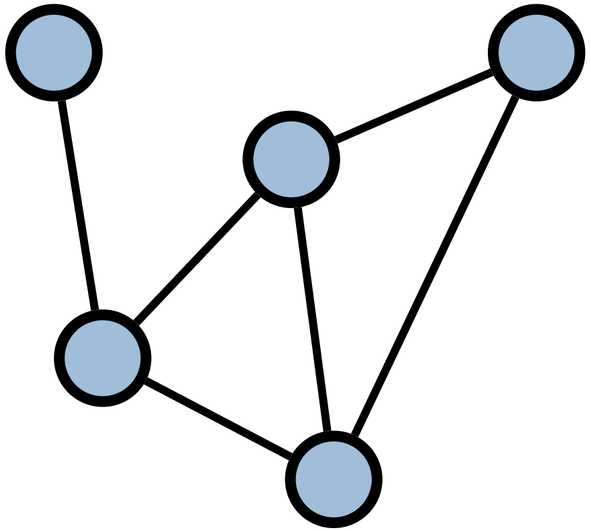
# Graph Types

There are three main types of graphs

**Undirected**

Links are bidirectional

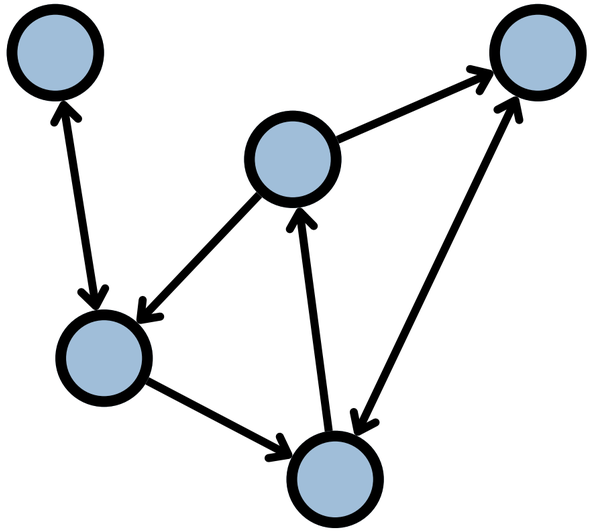
E.g. Facebook



**Directed**

Links are directional

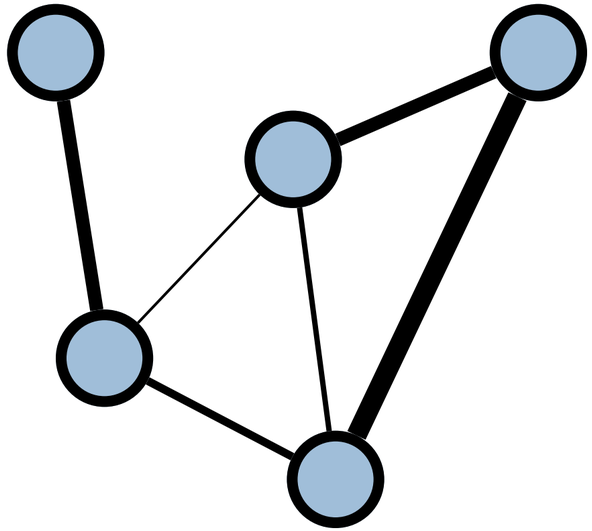
E.g. Twitter



**Weighted**

Links are weighted

E.g. Road Network

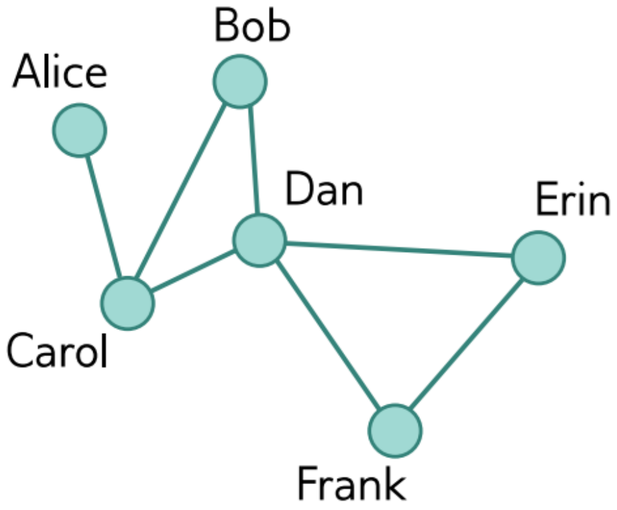


# Examples of Graphs

Graphs are everywhere, many systems can be described using this formalism

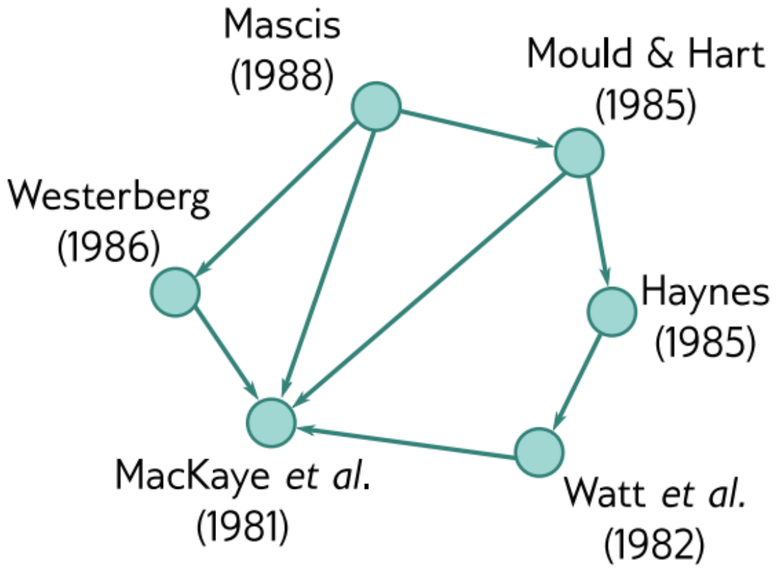
**Undirected**

Friendship Networks



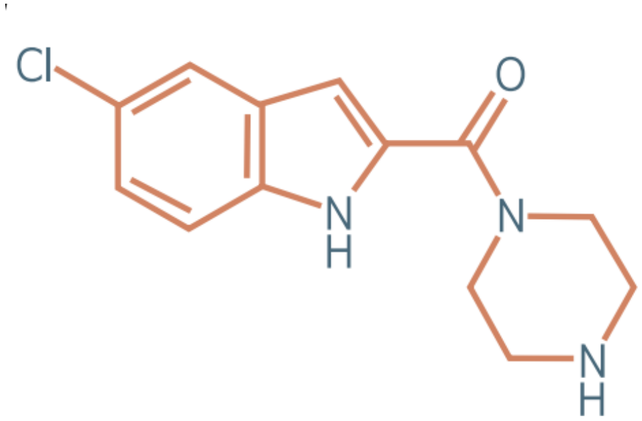

**Directed**

Citation Networks




**Weighted**

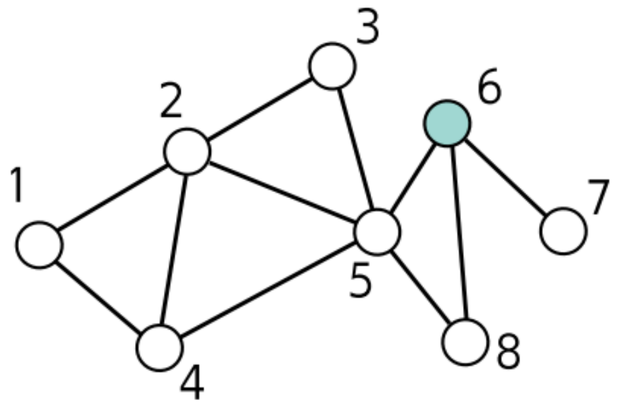
Molecules



# The Adjacency Matrix

A graph is mathematically represented by its adjacency matrix A

- it's an NxN matrix
- the element $A_{ij}$ of the matrix is different from zero if there is a link going from i to j
  - $A_{ij}=1$ in unweighted graphs
  - $A_{ij}=w_{ij}$ in weighted graphs
- if the graphs is undirected the adjacency matrix is simmetric
  - $A_{ij}=A_{ji}$
- elements on the diagonal (self-loops) are null

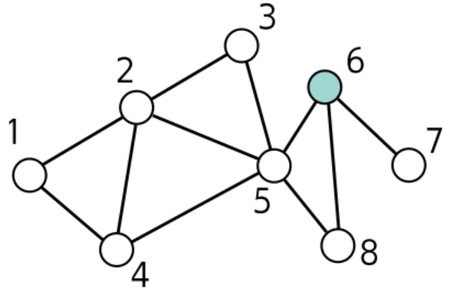



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

# Properties of the Adjacency Matrix

The adjacency matrix tell us which are the neighbors of each node in the graphs. We can use it to propagate information in the graph

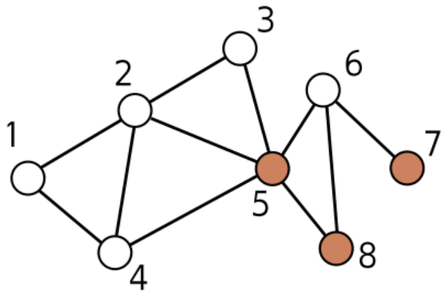
- if we multiply **A** by the one hot representation of a node **x** we get the number of path from that node to all other nodes
- if we multiply again we get the number of paths of length two and so on




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

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{Ax} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



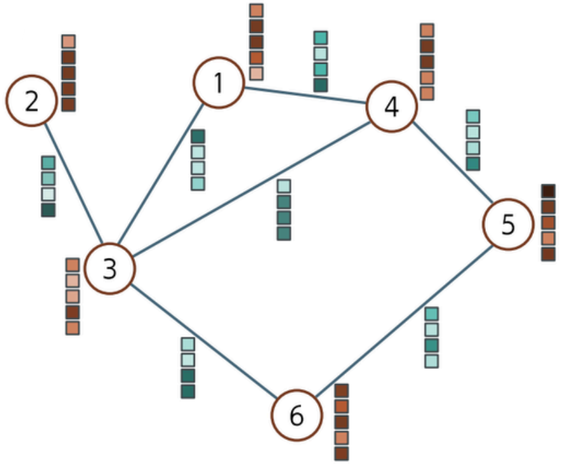

We can also use the adjacency matrix to compute the degree $d_i$ (number of links) of each node

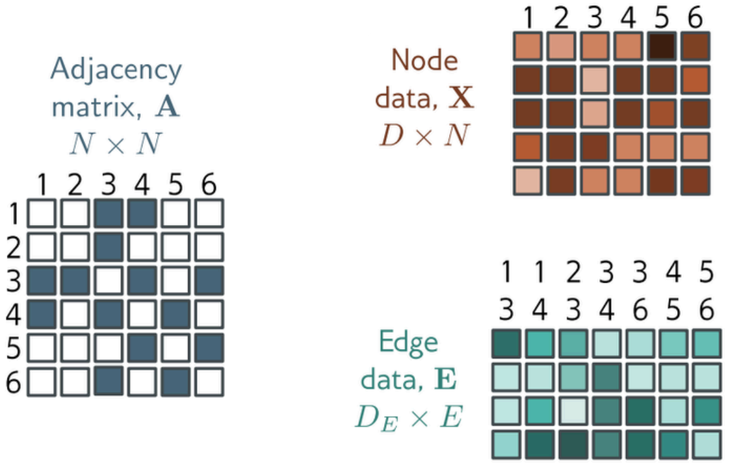
$d_i = \Sigma_j A_{ij}$

# Graph Structured Data

Graph structured data are generally described by three distinct matrices:

- **Adjacency Matrix** Contains the network structure (following links on Instagram)
  - Size is NxN
- **Node Data Matrix** Contains the D nodes features (gender, age, number of followers)
  - Size in DxN
- **Edge Data Matrix** Contains the $D_e$ edges features (when the following was made)
  - Size is $D_e$xN



# Challenges of Graph Data

Graphs present features that make it very hard to apply machine learning techniques to them

**Heterogeneous Structure**

While all images in a dataset all have the same number of pixels and the same structure, each graph in a dataset may have different number of nodes, edges and structure.

**Huge Sizes**

Many graphs have enormous sizes. For instance online social networks have billions of users (nodes) and hundred of billions of connections (links).

**Monolithic Graphs**

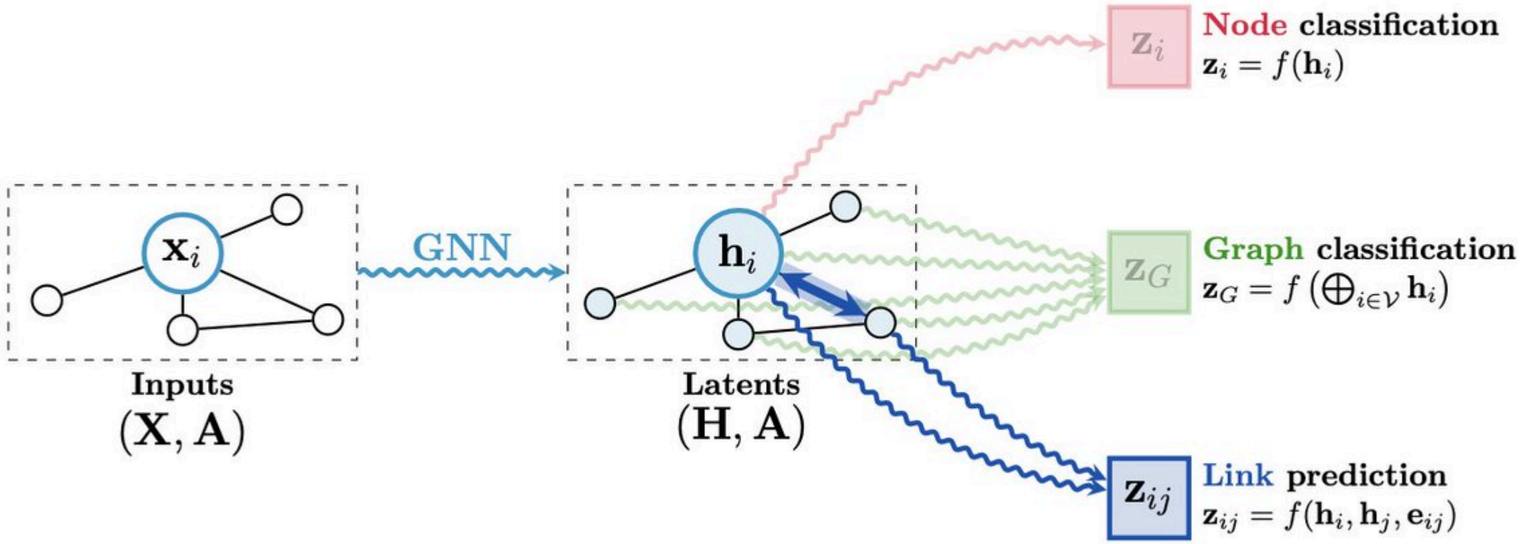
Often instead of having many graphs, only a single very large graph is available. Training and testing must be performed using just this large graph, requiring a different approach.


Universität
Konstanz
Machine Learning on Graphs

# The General Setup

The general approach to graph structured data consists of two conceptual steps

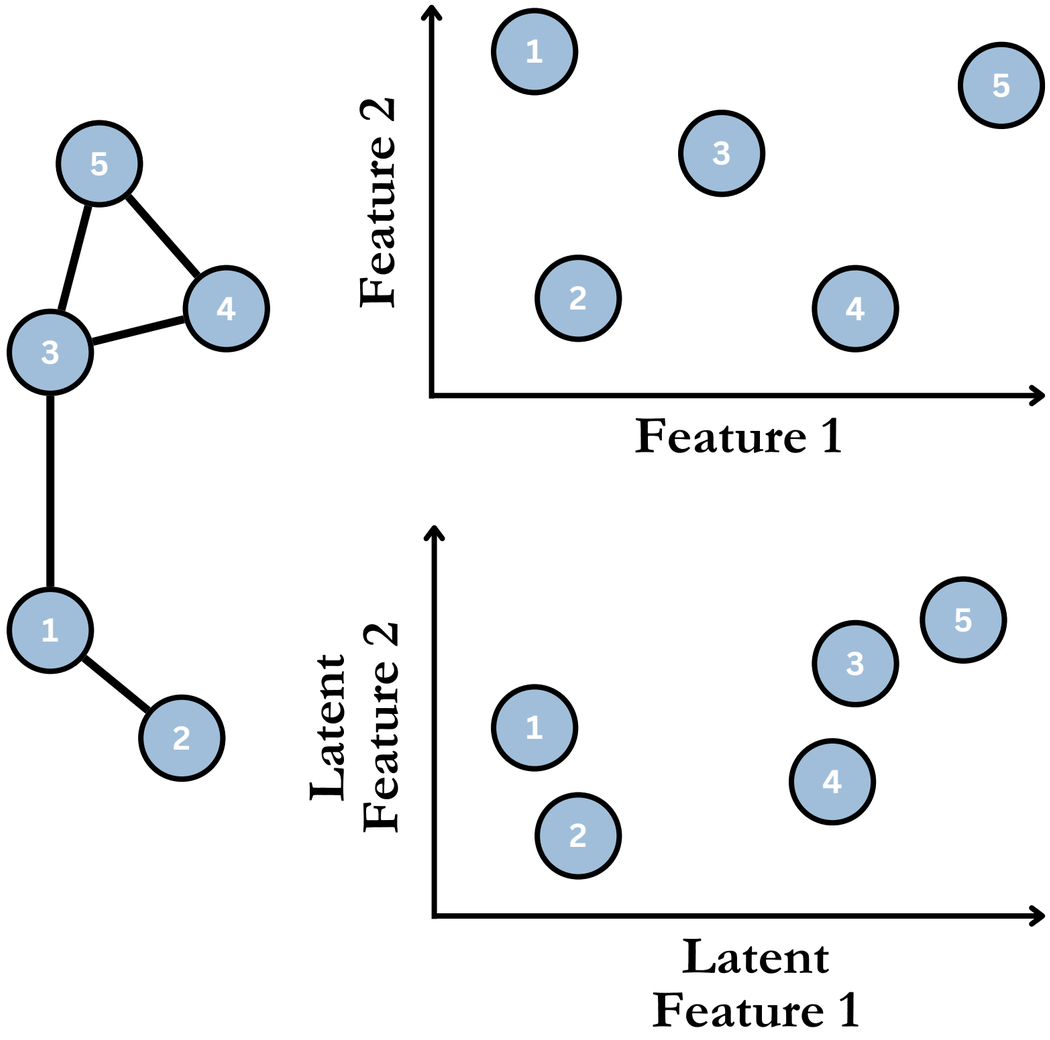
1. Use the nodes (and edge) features and the graph structure to build an hidden (or latent) representation of each node
2. Use these nodes representations as input for performing other machine learning tasks



# Nodes Embedding

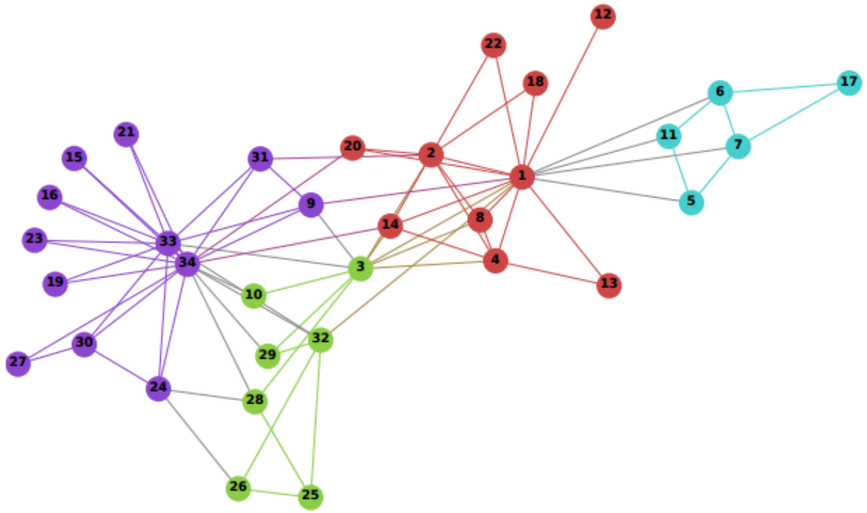
The latent representation of nodes is often called embedding

- initially each node is characterized by a vector of features
- the GNN combines these features together and with those of other nodes in the graph
- the result are new vector (latent features) that describer the nodes better, having also information about how nodes are positioned within the graph
- in generale the latent vector can have any dimension

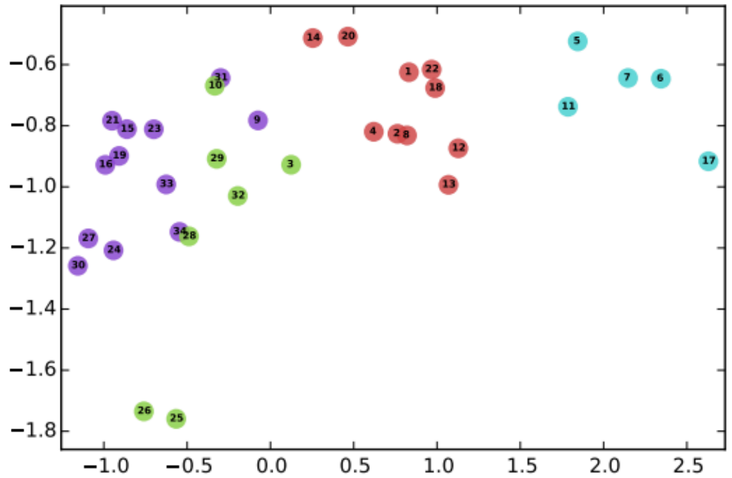


# Example: Karate Club

For instance we could get node embedding just looking at the structural aspect. Here the idea is that two nodes will have similar embedding if they are linked in the network or, more generally, if they have high similarity in the network.



(a) Input: Karate Graph

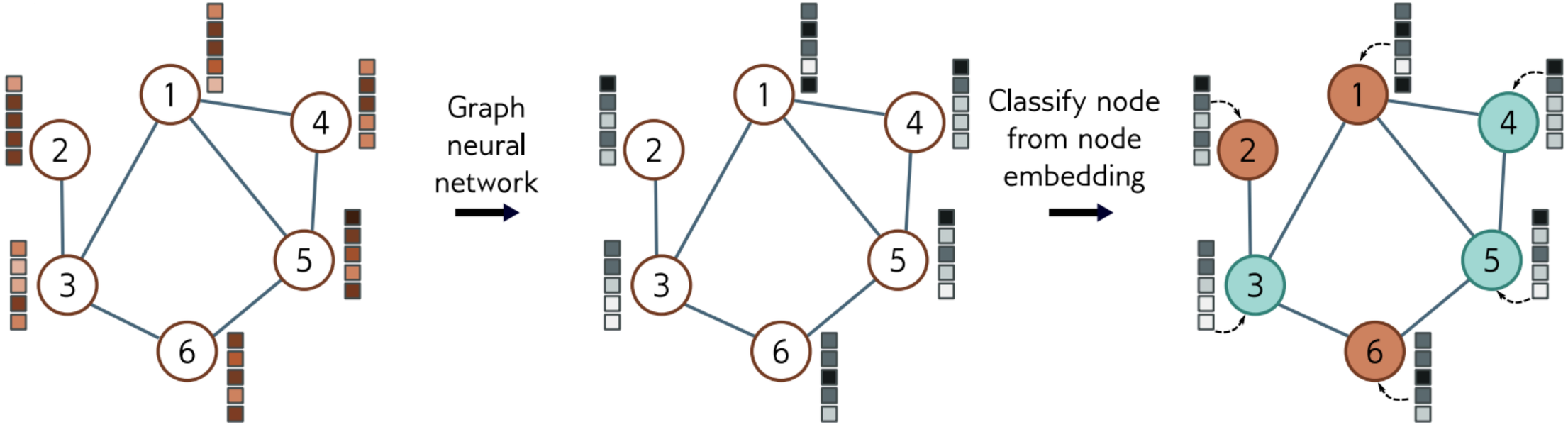



(b) Output: Representation

# Nodes Classification

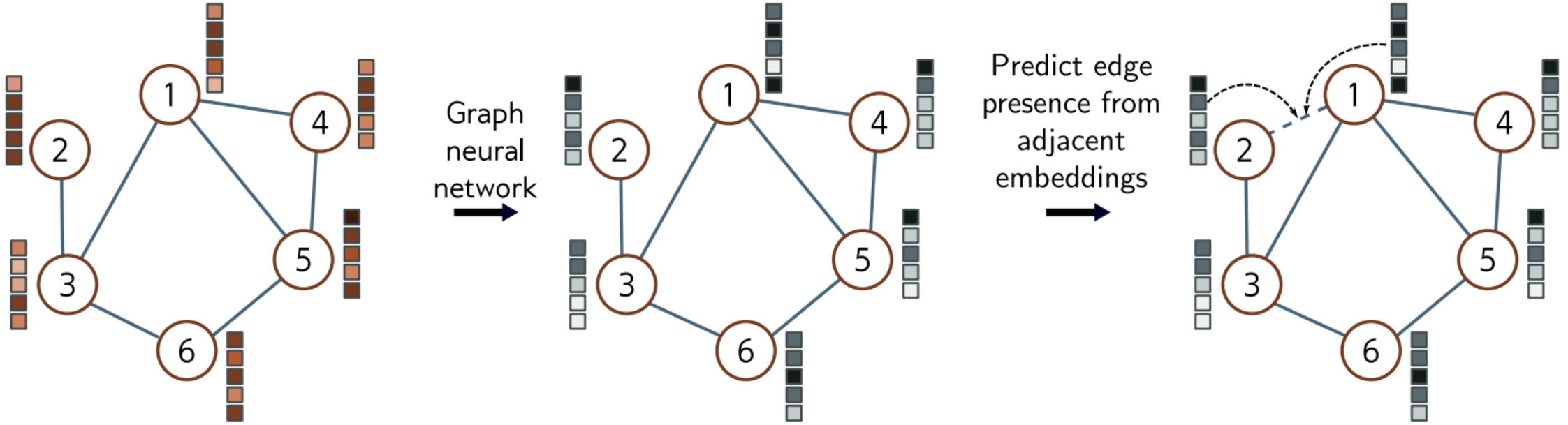
In a node classification task the goal is to assign each node in one (or more) graphs to two (or more) classes. In this case the embedding vectors are feed into a ML algorithm (can also be a neural network) and the problem is treated as a standard classification.

Example: Determine targets for ads campaign on social network



# Link Prediction

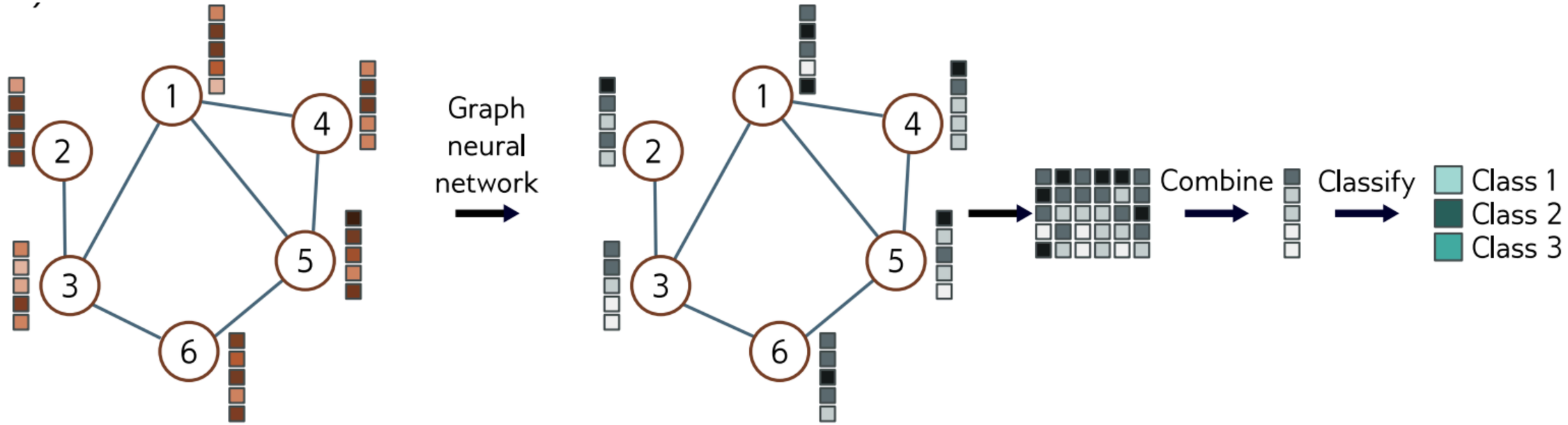
In a prediction task the goal is to predict whether or not a link between two nodes exists or may exist in the future. In this case the embedding from both involved nodes (and also the edge feature if present) are feed into a ML algorithm and again the problem is treated as a standard classification.
Example: recommend people to follow on social network



# Graph Classification

In a graph classification task the goal is to predict features of the graph as a whole. In this case the embedding from all nodes are combined and then feed into a ML algorithm that performs regression or classification on the graph.

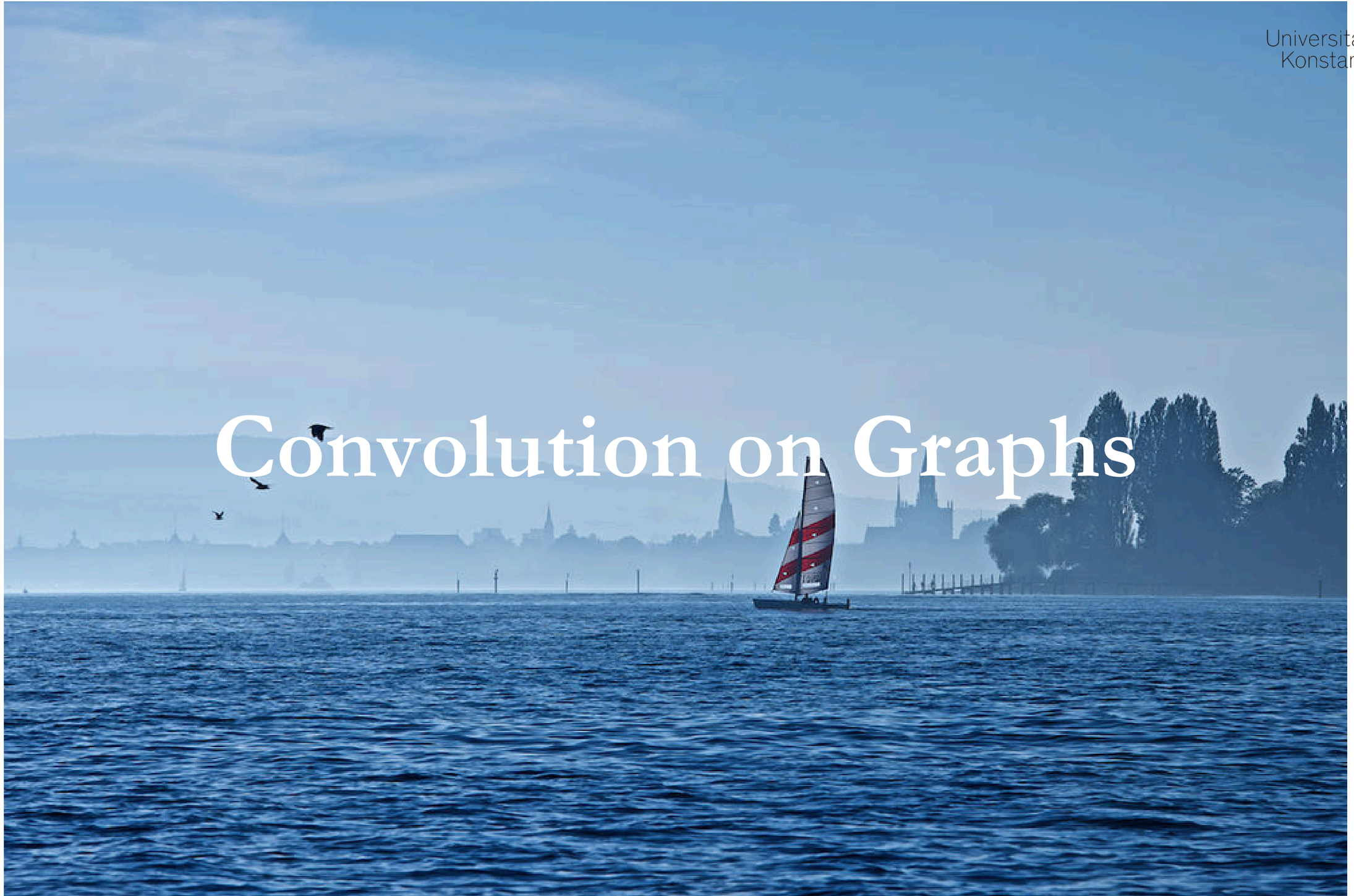
Example: predict whether or not a given molecule is poisonous



# Is the Graph Relevant?

All this is nice, but is the graph structure really relevant? Couldn't we achieve the same results only using the nodes (and edges) features as input for a MLP or a Random Forest?

- the chemical properties of a molecule strongly depend on their structure. Knowing the chemical formula is generally completely unusefull
- on a social network, the behavior of a person can be strongly influenced by their friends. Often the features of the majority of people in a social group are more relevant than the individual features
- in order to perform recommendation of followers or friends, it is useful to take into account the social circle. The friend of my friend is more likely to be, in its turn, a friend of mine I haven't connected with yet.



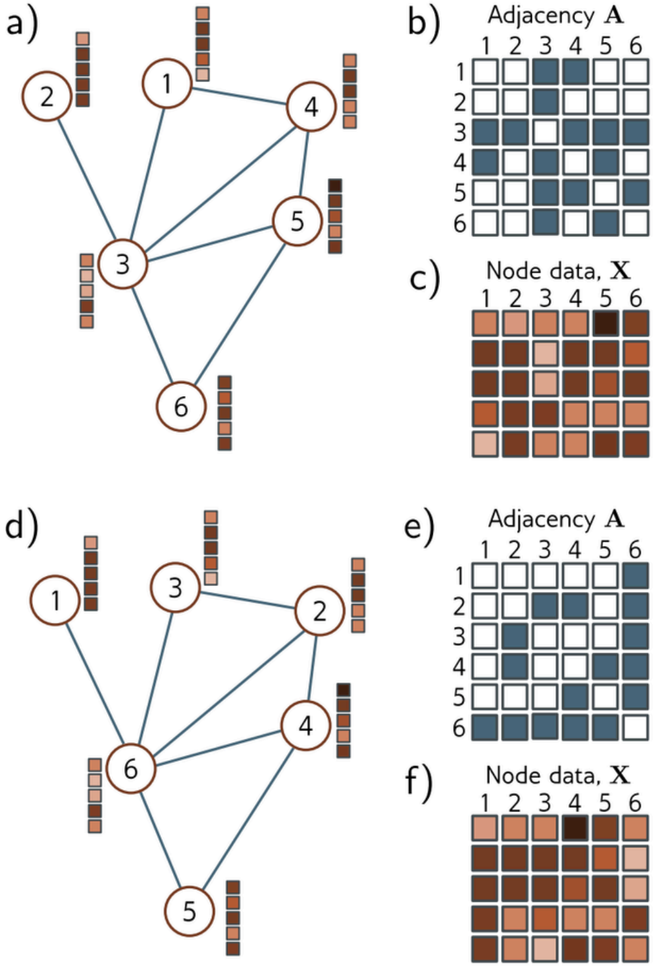
Universität
Konstanz

# Convolution on Graphs

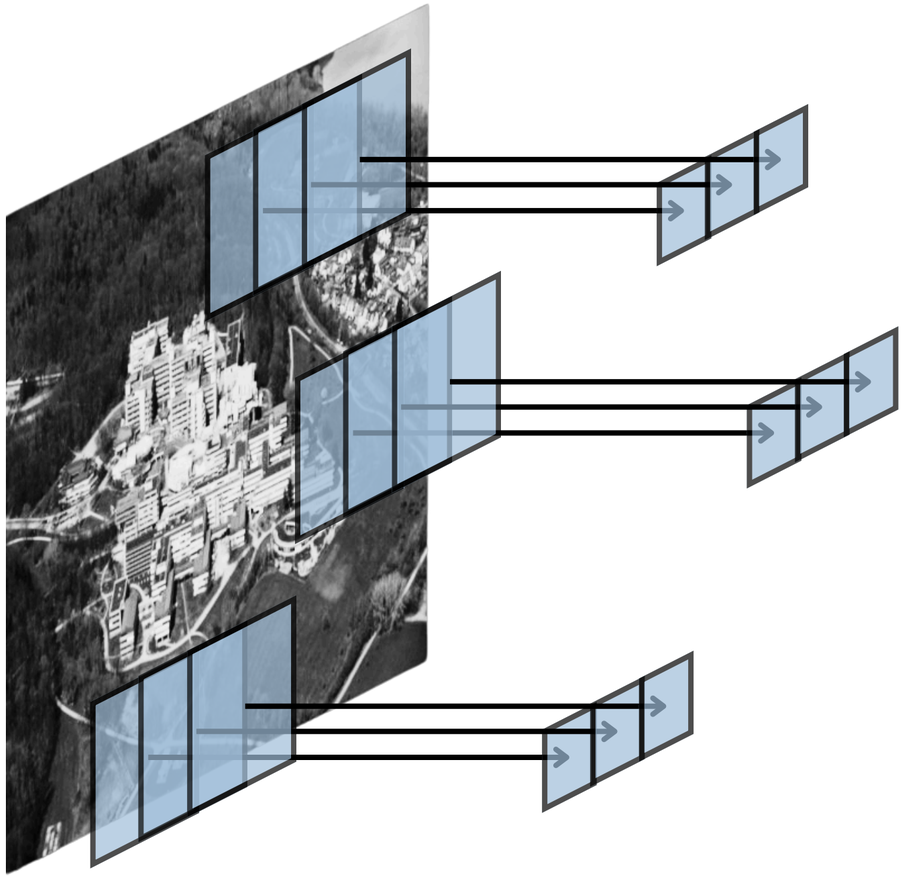
# Properties of Graphs

Similarly to images, also graphs have properties that we need to consider in order to build a neural network capable of analyzing them

- **Structural Locality.** Like images, graphs have a locality prior, meaning that nodes are more likely influenced by their neighbors than by nodes 4 or 5 links away from them.
- **Permutation Invariance.** If we change the order by which we list the nodes, the adjacency matrix will change, but the graph will remain the same. All operation must preserve this property.



# Recap: Image Convolution



Before considering graphs we recap how we approached the problem for images

- instead of using a MLP, we apply convolutional layers
- these layers are based on the idea of parameter sharing
- each layer is composed of many filters and each filter learns a different feature
- filters operate by applying linear operations to a pixel and its neighborhood

# Images are Special Graphs!

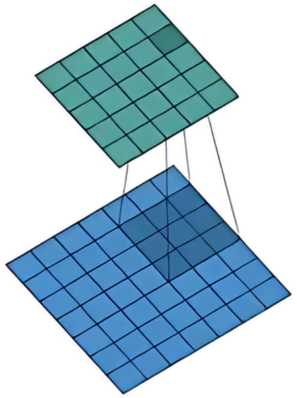
Images can be seen as special graphs:

- each pixel is a node
- each node is connected to the 8 closest pixels
- each node has a feature vector (B/W 1 number, color 3 numbers)

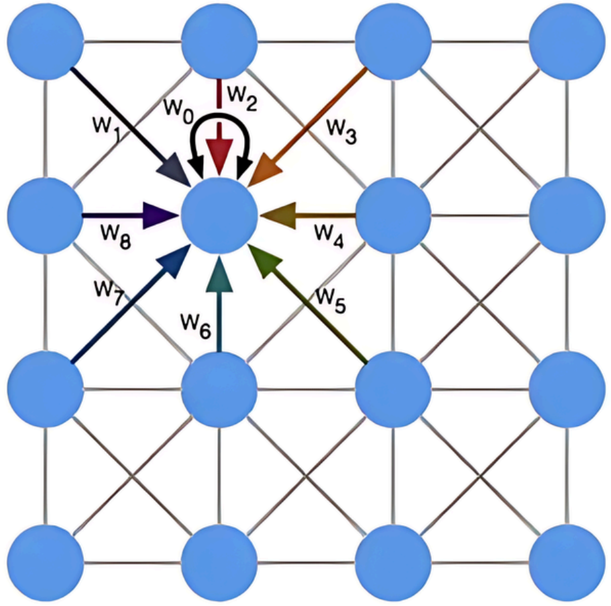
The convolution is combining all the information coming from a pixel and its neighbors **x** into a single value $h_i$

$$h_i = a\left(W_0 x_i + \sum_{n=1}^{8} W_n x_n\right)$$

Latent Representation $\boldsymbol{h}$



Input Image $\boldsymbol{x}$

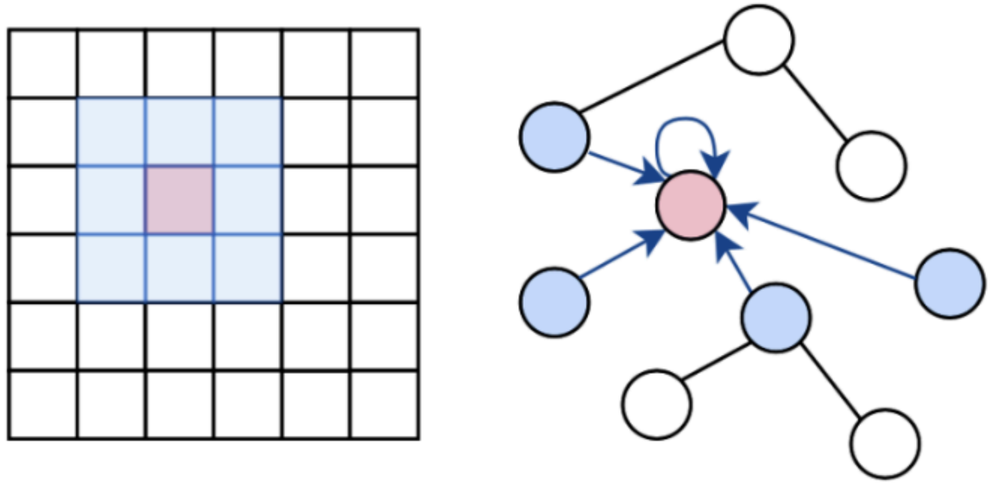


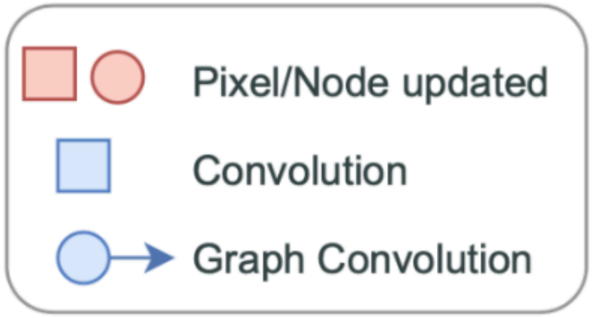
# From Images to Arbitrary Graphs

We want to generalize the procedure to arbitrary graphs:

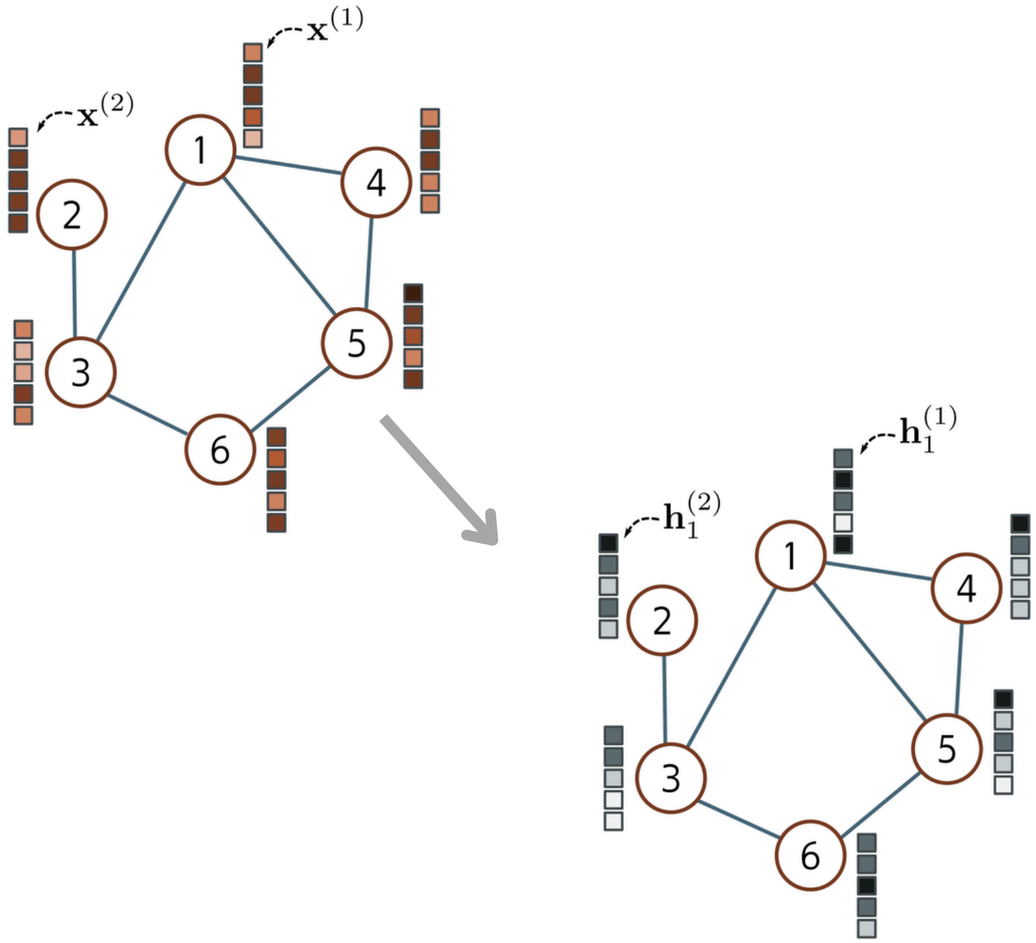
- now the number of neighbors is no longer fixed
- we can not use kernels of fixed size to learn parameters

Despite these limits the idea is the same, we want to use the graph structure to combine features coming from neighbors. Each node will have different number of incoming contributions.





# Translating this into Math




Let us for one moment forget about the trainable weights. If we want to combine information from close neighbors we can use the adjacency matrix

$$\mathbf{h_i} = a\left(\mathbf{x}_i + \sum_j^N A_{ij}\mathbf{x_j} + \beta_i\right)$$

Note that we add to explicitly include the self-loop, otherwise we lose information from the node itself. We set $\bar{A}=A+I$

$$\mathbf{h_i} = a\left(\sum_j^N \tilde{A}_{ij}\mathbf{x_j} + \beta_i\right)$$

# Adding Learnable Shared Weights

What type of parameters do we want to learn?

- in CNN the weights are inside the filters. These weights are weighting in a different way all the contributions coming from the 8 different neighbors
- this approach is not feasible in our case since there is no fixed number of neighbors
- instead we want the weights to learn relations among features
- these relations are independent on where the node is placed on the graph, so we can reuse parameters

This is achieved by multiplying the input features by a matrix of learnable parameters $\mathbf{W}$

$$\mathbf{h_i} = a\left(\sum_j^N \tilde{A}_{ij}\mathbf{W} \cdot \mathbf{x_j} + \beta_i\right)$$

The size of $\mathbf{W}$ is HxD, where D is the input dimension and H the latent features dimension

# What are the Weights Doing?

Let's try to better understand what the weights are doing

- we assume each node to have two features
  - $\boldsymbol{x_j}$ is a vector with two components
- we set the latent dimension H to 3
  - $\boldsymbol{W}$ is a 3x2 matrix

When we multiply the matrix with the feature vector, we obtain a new feature vectors with 3 components that contains linear combinations of the original vectors. This is analogous to what we would do in a perceptron

$$\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \cdot \begin{pmatrix} x_{j1} \\ x_{j2} \end{pmatrix} = \begin{pmatrix} w_{11}x_{j1} + w_{12}x_{j2} \\ w_{21}x_{j1} + w_{22}x_{j2} \\ w_{31}x_{j1} + w_{32}x_{j2} \end{pmatrix}$$

# The Graph Convolutional Layer

We are almost there

- the layer we defined in the previous slides has a small issue
- every time we apply the operation we are summing many vectors, one for each neighbor
- if we built a Deep Neural Network, layer after layer these values will grow causing problems

In order to solve this issue we need to add a normalization. The standard choice is

$$\mathbf{h_i} = a\left(\sum_j^N \frac{\tilde{A}_{ij}}{\sqrt{d_i d_j}}\mathbf{W}\cdot\mathbf{x_j} + \beta_i\right)$$

Here $d_i$ denotes the degree of node i (number of connections). This expression defines the so called Graph Convolutional Layer. Similarly to a CNN we are aggregating spatial information and we are reusing parameters in different locations of the graph.

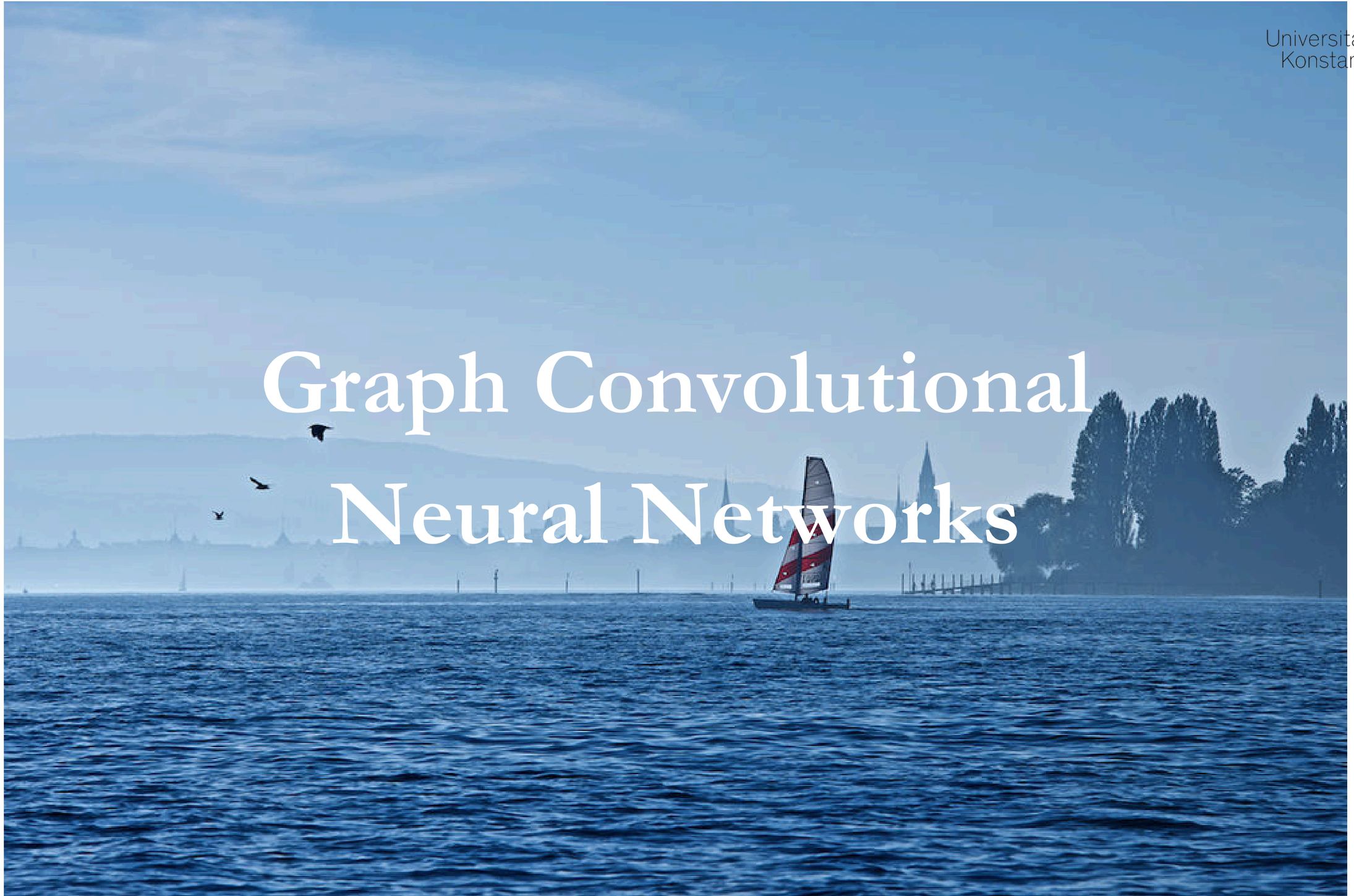
# Other Possible Approaches

The normalization we presented in the previous slide is the standard approach, but there are also other viable options

- when the structural properties are very relevant it is better not to normalize. Normalization destroys some information, for instance about the degree. It should only be applied when we are mostly interested in the features
- another possibility is to only use node i degree to perform the normalization. This gives the following Graph Neural Network Layer

$$\mathbf{h_i} = a\left(\sum_j^N \frac{\tilde{A}_{ij}}{d_i}\mathbf{W} \cdot \mathbf{x_j} + \beta_i\right)$$

There are also more powerful approaches where we automatically learn the normalization instead of deciding it a priori. We will shortly introduce these techniques later.



Universität
Konstanz

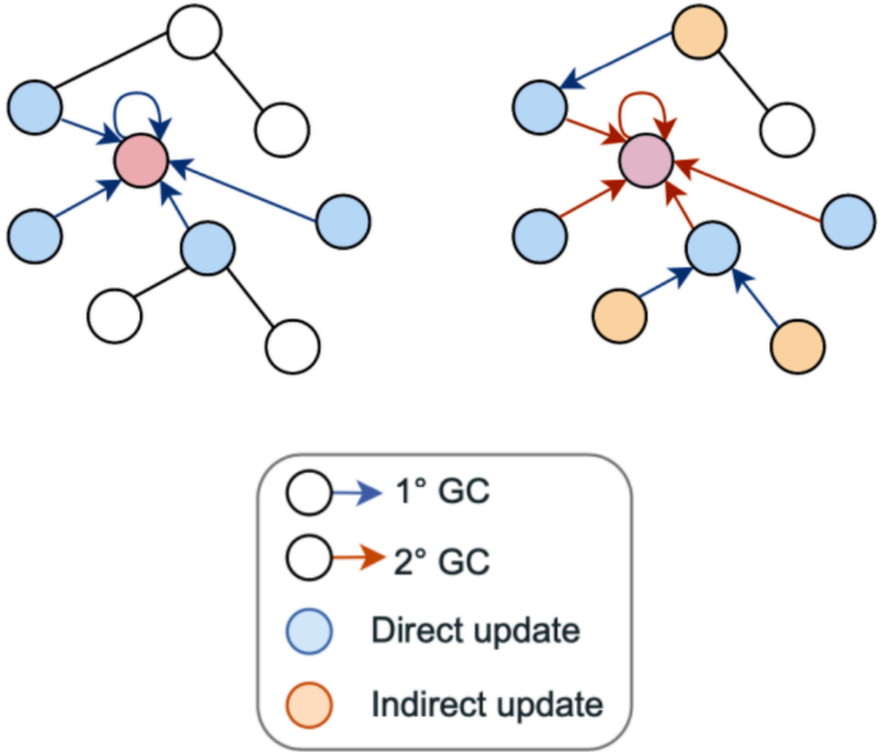
# Graph Convolutional Neural Networks

# Stacking Layers

A Graph Convolutional Neural Network consists of several graph convolutional layers stacked one after the other

- the first layer aggregate information (messages) coming from the first neighbors
- the second layer aggregate information coming from the second neighbors
- deeper layer allow to better capture how the node is placed within the graph

We see that the approach is similar to standard CNN.

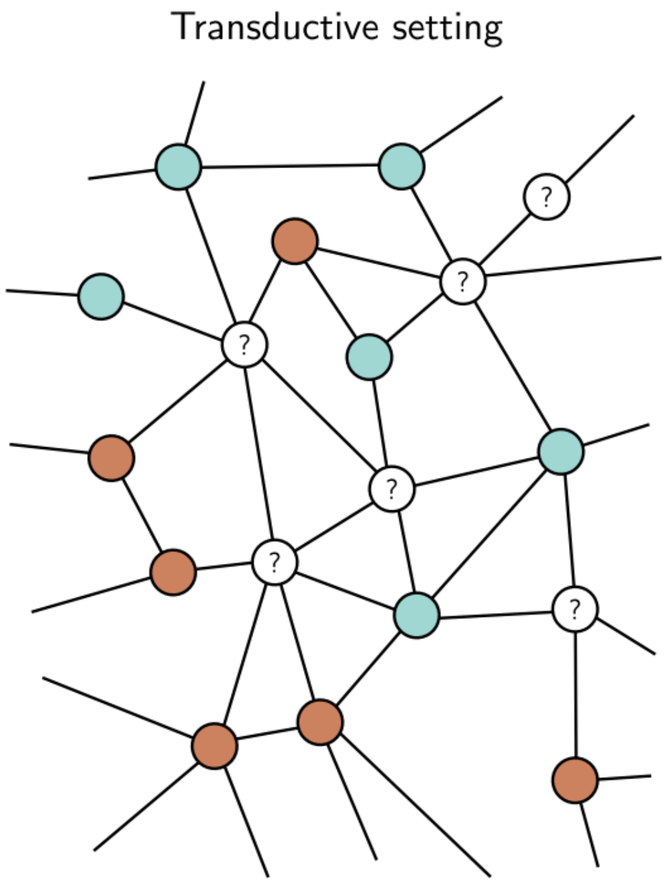


# Node Classification Task

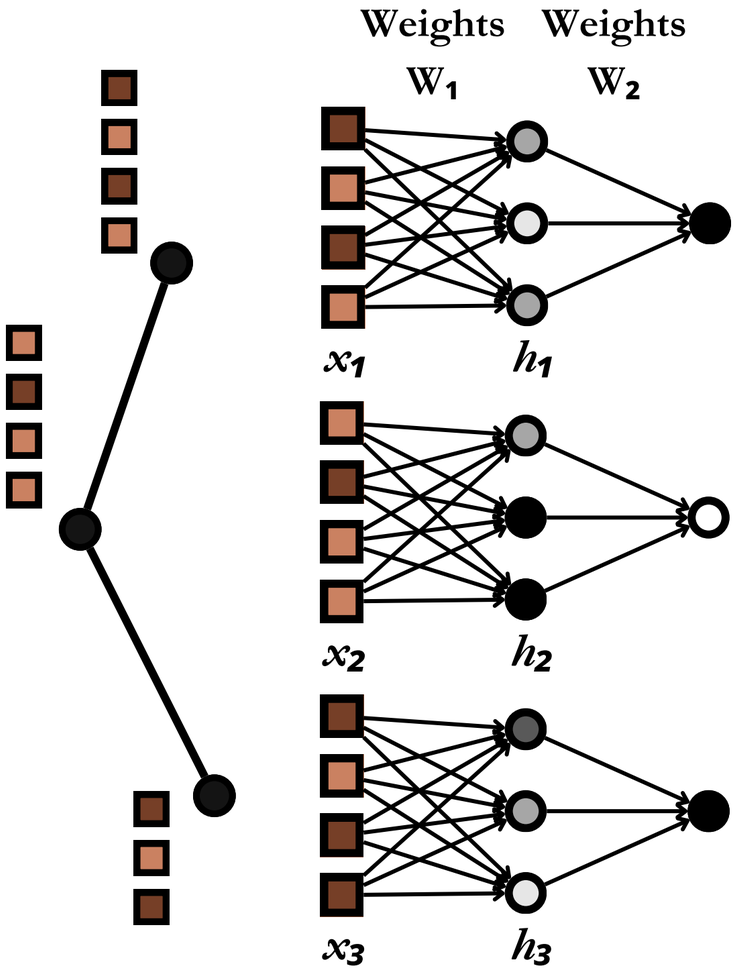
As an example we consider a node classification task in a transductive setting:

- we only have one large graph
- some of its nodes are labelled (train set)
- the rest of the nodes have no label (test set)
- the goal is to classify the test nodes

In most applications the number of labelled nodes is quite small compared with the size of the graph. Moreover the training is performed just on a single instance of the graph.



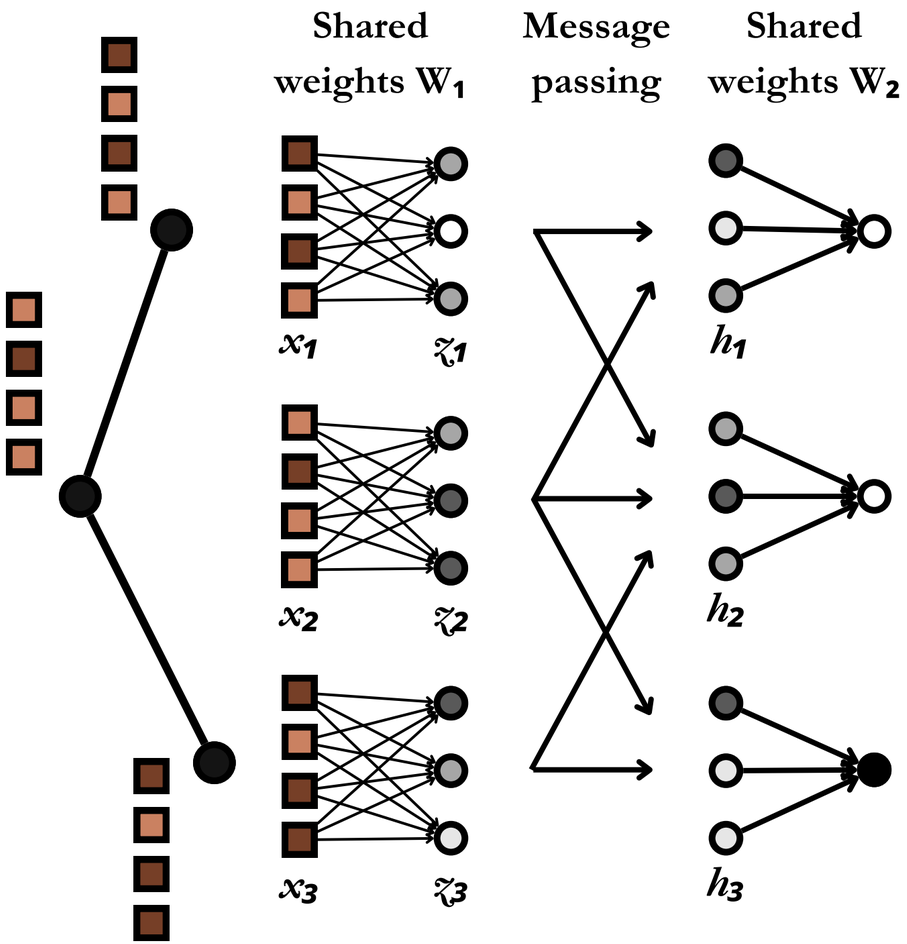
# MLP Approach




First we consider the problem from a MLP perspective

- each node i is characterized by a vector of features $x_i$
- we focus on these vectors and we completely discard the graph
- we train a MLP using all the vectors in the training set and then we use it in the test set
- the hidden layers of the MLP produce latenr representation of the data $h_i$
- this is the standard procedure for MLP classification

# GCNN Approach



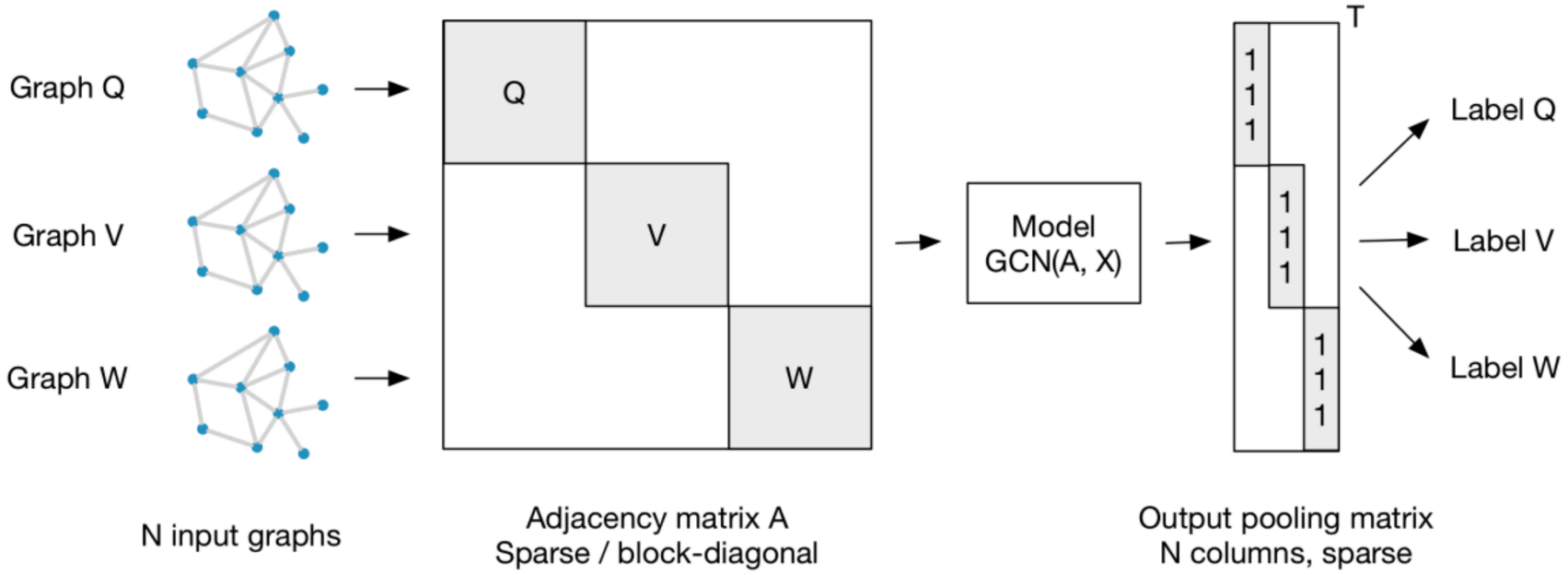

Using a GCNN is not that different

- each node i is characterized by a vector of features $x_i$
- we focus on these vectors and also on the graph structure
- now a dense layer is used to create an intermediate representation of features $z_j$
- these representations are combined using the graph structure to get the latent representation $h_i$
- these latent representations are then used for the classification

The key difference is that we are making the various latent representations talk using the graph.

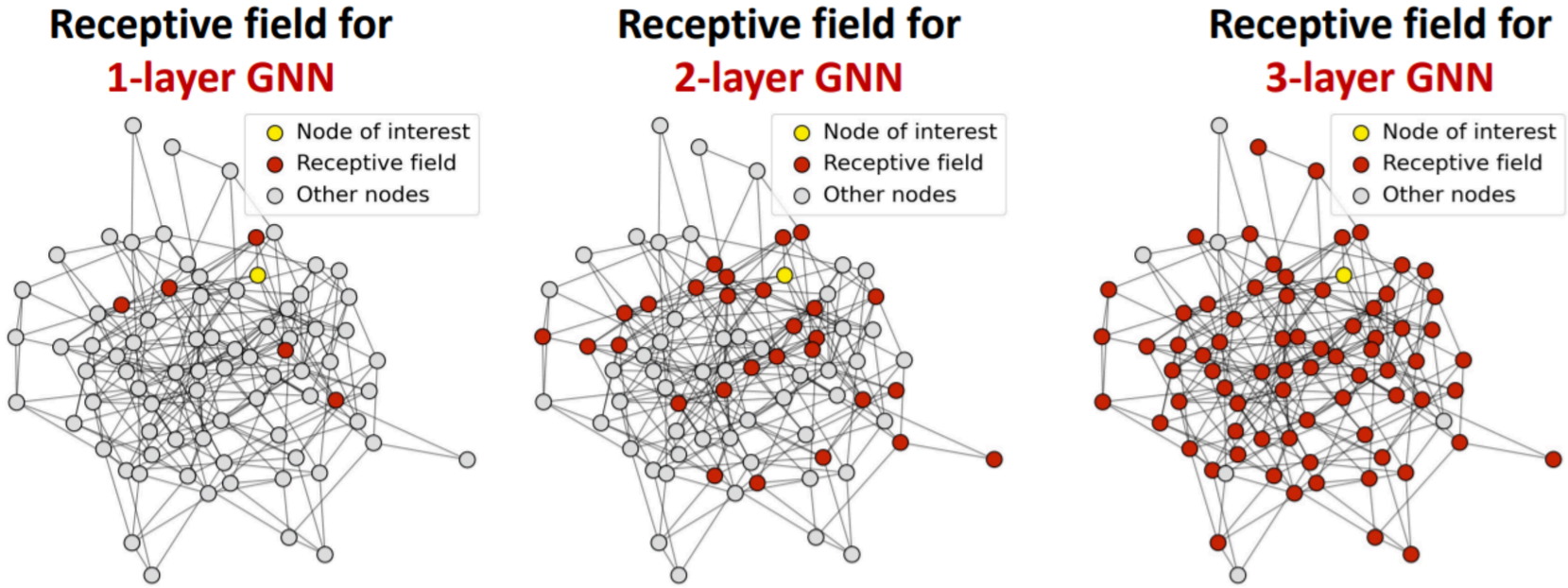
# Batching Graphs

When we apply GNN to a set of many graphs we need a way to batch them for doing parallel computing. Since graphs have all different sizes, we can not just stack them like with images. The idea is to merge many graphs in a giant block graph to process all of them at the same time.



# Over-Smoothing Problem

GCNN suffer from the over-smoothing problem. The point is that when we combine many graph convolutional layers, we are making each node talk with larger and larger portions of the graph. Since many graphs have the "small world" property, just a few layers are often enough for making all nodes embedding very similar.



# Limits of GCNN

Another relevant limit of GCNN is their tendency to treat all links in the same way:

- in a CNN the weights in the filters give different importance to the various neighbors
- in a GCNN the weights given to the neighbors are not learned and instead are fixed a priori
  - 1 in absence of normalization
  - $1/\sqrt{(d_i d_j)}$ for the standard convolutional layer
  - $1/d_i$ in the other variation

However we may expect not all links to be equally relevant and we want to make the network learn which are the most important links by its own

- on a social network I can be linked to many people, but the most relevant are those more similar to me, so with similar feature vectors

# Graph Attention Networks

The limit of GCNN is effectively overcome by the so called Graph Attention Networks (GAT)

- the idea is to learn how important is each node for each of its neighbors
- this is similar to the attention mechanism in transformers
  - transformers can be seen as special GNN
  - nodes are the words and they live on a fully connected layer

We can then modify the graph convolutional layer by adding explicit weights to links

$$\mathbf{h_i} = a\left(\sum_j^N a_{ij}\mathbf{W}\cdot\mathbf{x_j} + \beta_i\right)$$

Learning the $a_{ij}$ is computationally expensive. For these reason in GAT the $a_{ij}$ are implicitly defined using the feature vectors and a vector of learnable weights.

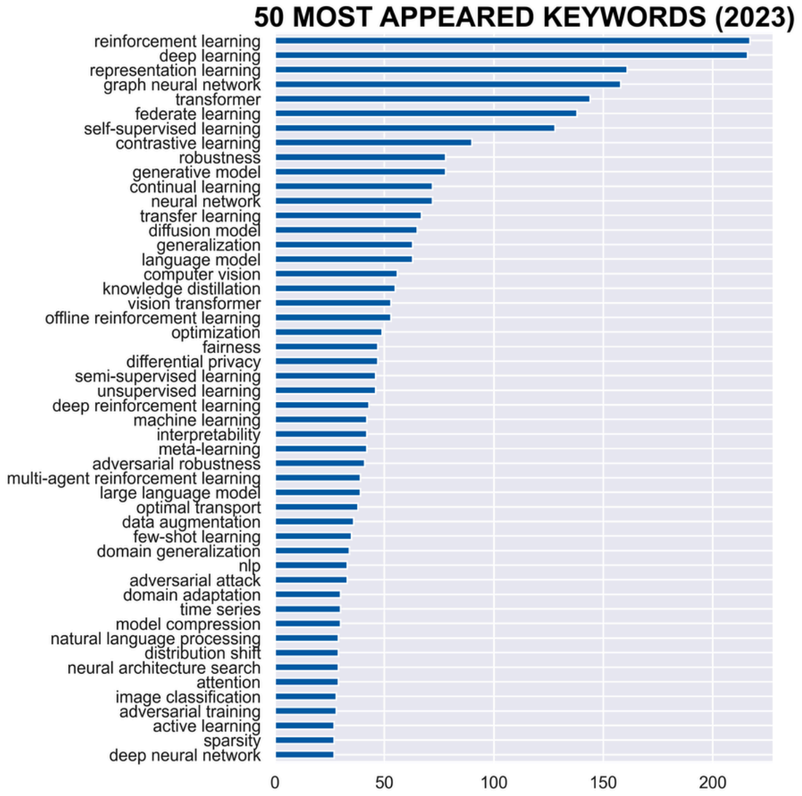
Universität
Konstanz

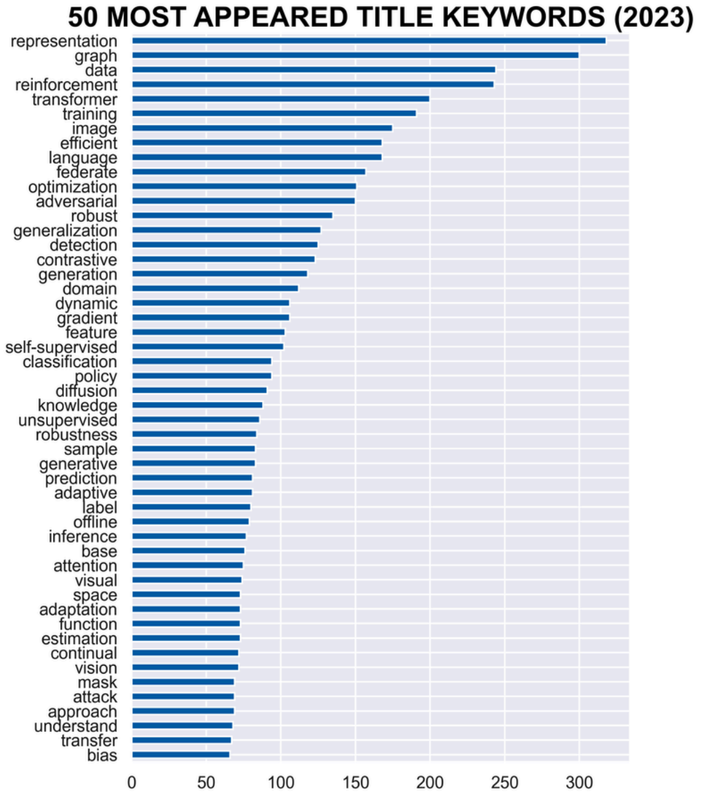
# GNN Applications

Universität Konstanz

# Graphs are Everywhere!

Graph structured data are ubiquitous and for this reason GNN are a very hot topic. They find application in very diverse fields, from social networks to chemistry.



# Drug Discovery

Molecules can be described by graphs with both node features (chemical properties of the atoms) and edge features (bond type). Drugs are just special molecules, so we can use GNN to understand if a molecule can be a drug without having to test it

- we use all known drugs to build a train set
- we train a GNN over the known drugs
- we use the GNN  to get a drugs embedding
- we use the embeddings to predict whether or not a given molecule can be used a drug

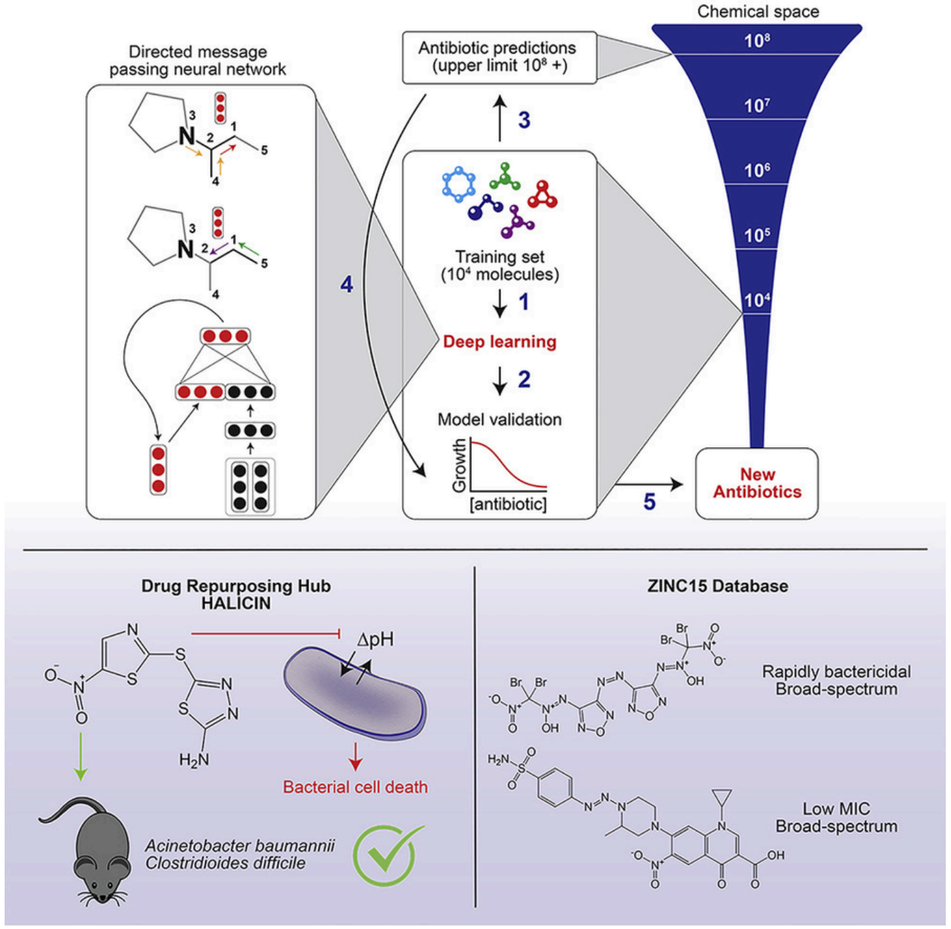
In this case the setting is similar to the standard Machine Learning approach, since we have many graphs to classify rather than a single monolithic structure. Once we have a large sample of potential drugs we can ask chemists to test them.

# A Novel Antibiotic

Researchers applied this techniques to antibiotics

- they took large databases of known drugs never
- they predicted which of them could be repurposed to work as antibiotics
- chemists studied the ~ top 100 predictions from the model

This lead to the “discovery” of a very powerful antibiotic, the Halicin. It was a known molecule, but it was previously used for other purposes.



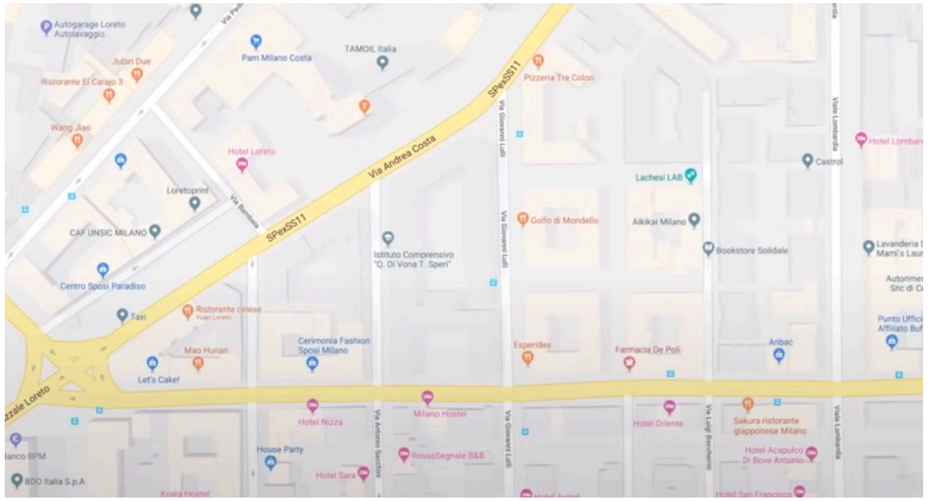
# Road Networks

A system of road can be described by a network

- nodes are the intersections
- links are the streets connecting these intersections

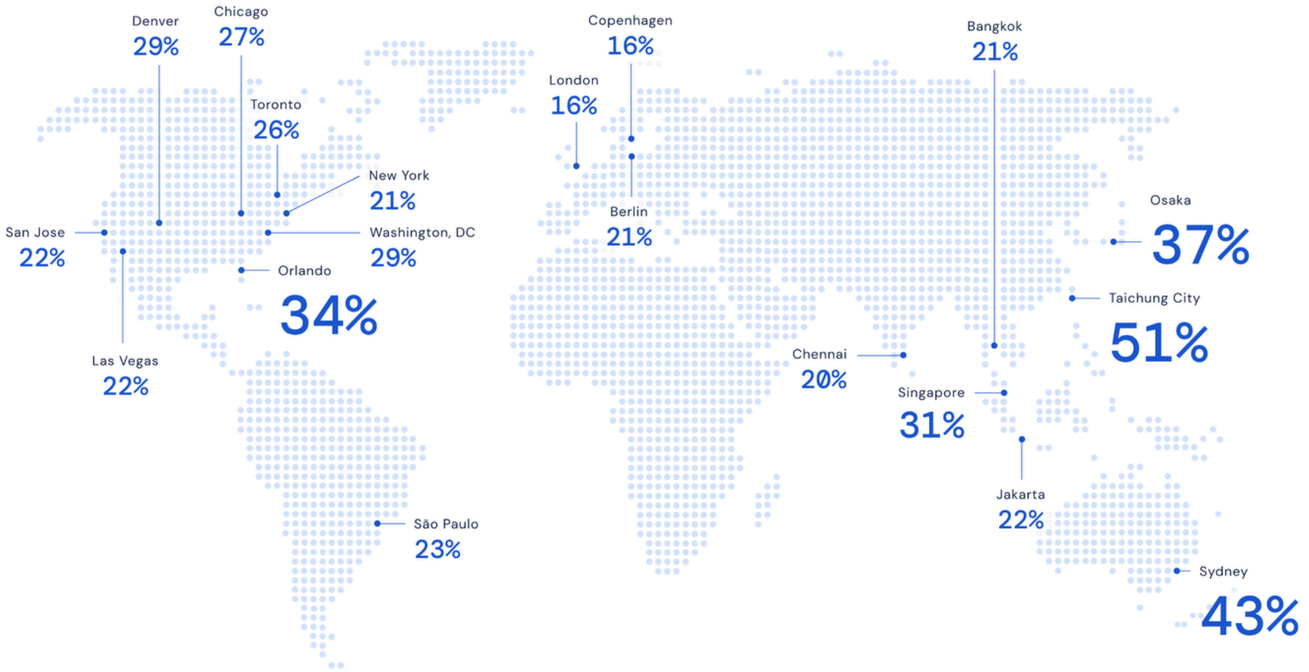
Clearly in such a setting we have both node and edge features

- **Node features.** Presence of traffic light, presence of roundabout etc
- **Link features.** Road length and width, average speed, current speed




# Getting better ETAs

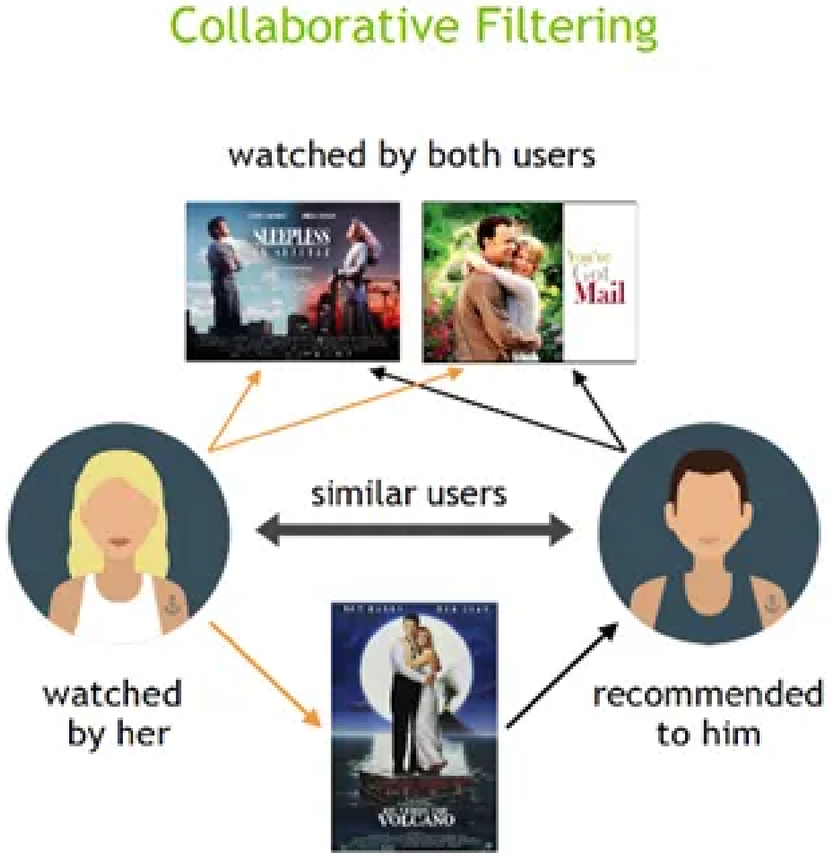
GNN can be applied on road networks to get more a precise Estimated Time of Arrival. This is a crucial point for services such as Google Maps and is already being implemented in major cities, leading to a major improvement in ETAs.



# Recommendation Algorithms

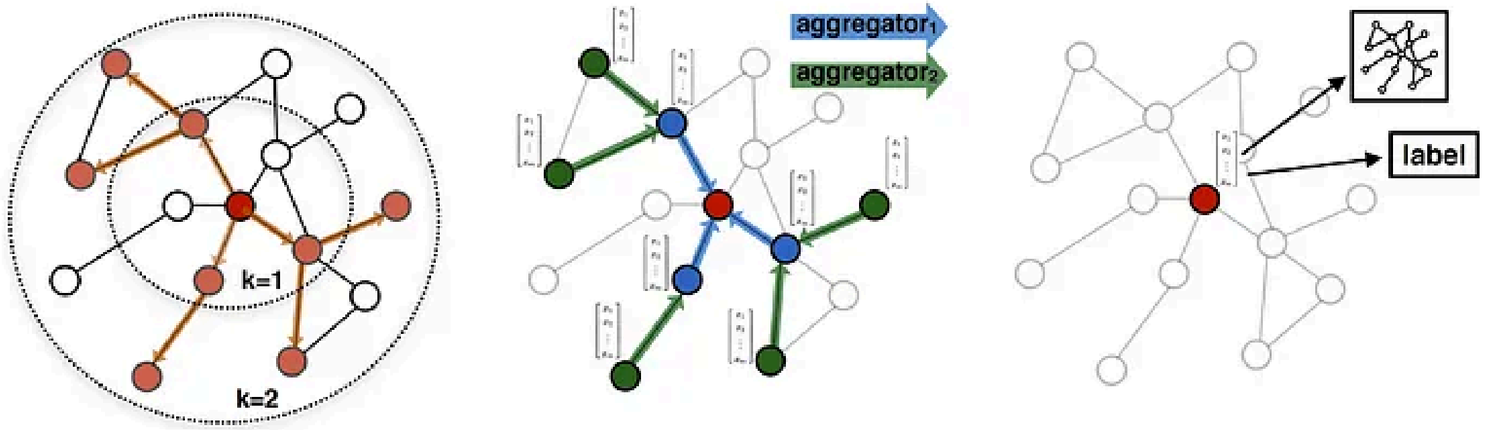
All online platforms use recommendation algorithms to suggest us content to consume or people to link to. Users and contents can be arranged in a (bipartite) network, with arrows connecting users to the content they already consumed

- the idea is to use previous users' tastes and choices to recommend new content
- the problem can be treated as a link-prediction task and can thus be approached with a GNN



# GraphSAGE

Applying GNNs to social networks presents a big challenge: networks are huge! In order to solve this problem researchers proposed GraphSAGE. In this model, instead of considering the full graph to update each node, only a neighborhood is considered. This technique is currently being applied, among others, by Pinterest and Facebook.




1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

# Summary

**Graphs**

Many systems can be represented as graphs, mathematical objects composed of nodes and links

**Convolution on Graphs**

We generalized the concept of convolution to graphs. The idea is to aggregate features coming from the neighbors to get nodes embeddings.

**Graph Convolutional Neural Networks**

By stacking many Convolutional Layers we can build Graph Convolutional Neural Network. We saw how to implement them using dense layers.

**Applications**

Since graphs are ubiquitous GNN have countless applications ranging from drug discovery to recommendation algorithms