Machine Learning for Complex Networks

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

Lecture 11
Graph Neural Networks

July 10, 2024

Notes:

- **Lecture L11:** Graph Neural Networks 10.07.2024
- **Educational objective:** We motivate the application of neural networks to structured data. We introduce convolutional neural networks for images, which can be viewed as regular lattice graphs of pixels. Introducing Graph Neural Networks, we then show how we can use neural message passing to directly apply deep neural networks to graph-structured data.
  - Image Data and Convolutional Neural Networks
  - Graph Neural Networks and Message Passing
  - Graph Convolutional Networks (GCNs)

Motivation

- we have applied neural networks to **Euclidean representations of graphs**

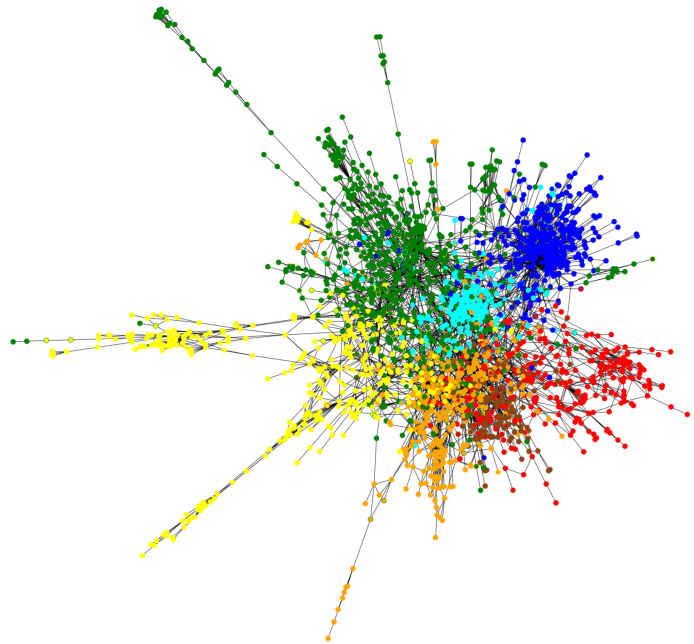
  → L08/L09

- DeepWalk and node2vec use neural networks to encode **node neighborhoods** in complex networks → L10

- graph topology implicitly contained in **node-context pairs** used to train SkipGram model

- can we directly **apply neural networks to data with graph structure**?

- how can we leverage both **graph topology and node features**?



CORA citation network

image credit:
`https://stellargraph.readthedocs.io`

Notes:

- In the previous lectures we considered different approaches to apply (deep) neural networks to graph data. Let us briefly recap this. In lecture 08, we considered graph representation learning techniques that enable us to embed nodes in a Euclidean feature space. In lecture 09, we then demonstrated that the application of neural networks to those embeddings enables us to address a number of graph learning tasks. Adopting the SkipGram model, in lecture 10 we then used neural networks to learn graph representations, which could again be used in downstream learning tasks.
- However, none of those approaches constitute a direct application of neural networks to graph data. In DeepWalk and node2vec, we rather used different random walk models to encode node neighborhoods in node-context pairs, which were then used to train the SkipGram model.
- This raises the question how we can apply neural networks to graph data in a more direct way, i.e. where we directly feed the graph to a (deep) neural network. Moreover, this would also allow us to incorporate additional node features, which are often available in real data on complex networks. Consider, e.g., node classification in the CORA citation graph, where we have access to the network of citations between articles and node feature vectors that capture term frequencies in scientific articles.
- In today's lecture, we will introduce recent advances in the application of deep neural networks to graph-structured data, which have largely been inspired by spectral graph theory, convolutional neural networks, inference in Bayesian networks, and the study of dynamical systems on graphs.

Neural Networks for Structured Data?

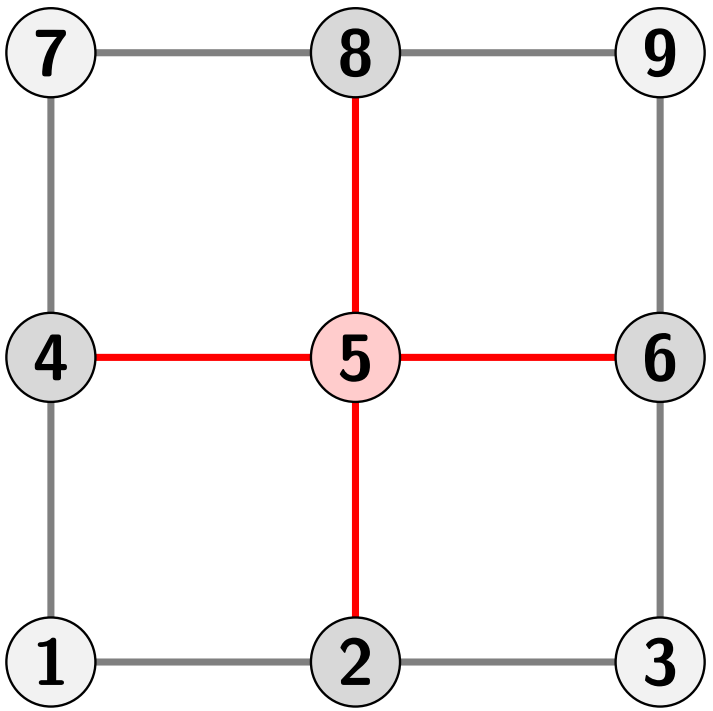
- in standard feed-forward network each neuron in hidden layer receives all inputs, i.e. **fully connected topology**

example: image data

- consider grayscale image with $1000 \cdot 1000$ pixels
- 1 million input nodes
- 1 million learnable weights per hidden neuron

- images contain **strong spatial correlations**, i.e. from a neuron's perspective not all pixels are equally important

- idea: use structure of data to **constrain topology of neural network**



lattice network of pixels in 3×3 image

Notes:

- The starting point for our discussion is the general problem of applying neural networks to data that has an internal structure. As a simple example, consider the application of a feed-forward neural network to a $W \times H$ image. If we consider a black/white image, each pixel is associated with one (e.g. 8 bit) value, which results in one million inputs for a 1000×1000 pixel image. For such high-dimensional data, the naive approach of connecting each input to each neuron in the hidden layer is not practical, because it results in one million learnable weights per hidden neuron. Such an architecture also does not consider that image data are likely to exhibit strong spatial correlations between pixels that are close to each other.
- We can actually view images as a special type of graph, where pixels are nodes in a two-dimensional lattice graph that connects neighboring pixels.

Neural Networks for Image Data

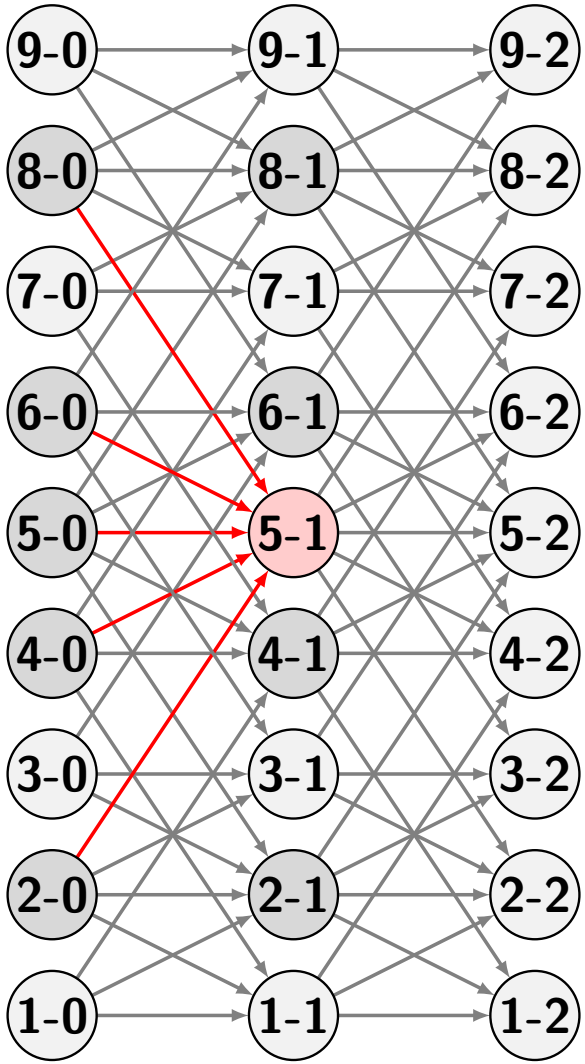
- we can **use lattice structure of pixels to connect neurons** in subsequent layers

  → K Fukushima, 1979

- neurons corresponding to each pixel $\mathbf{I}_{xy}$ receives input from a **receptive field** defined by neighboring pixels

- facilitates exploitation of **spatial correlations in images**

- to model spatial correlations in image data we can compute **cross-correlation** or **convolution**



sparse topology of neural network for 3×3 pixel image

Notes:

- The topology of our neural network should account for this structure of image data. To do so, we can create a neural network where each neuron only receives input from one pixel and its neighboring pixels, rather than receiving inputs from all pixels in the image. In the example above, the first layer consists of nine input nodes that correspond to the nine pixels of the 3×3 pixel image shown in the previous slide. The hidden neuron $5 - 1$ corresponding to pixel 5 in the first hidden layer only receives inputs from the four neighbor pixels $2, 4, 6, 8$ (and itself).
- This approach has several advantages (and interpretations): First, by limiting the number of connections in the neural network we effectively "regularize" the model, i.e. we use the structure of data to reduce the number of learnable parameters and thus simplify the model. This counters the curse of dimensionality that we would face in a fully connected topology. Second, we assign a localized "perceptive field" to each neuron, which has a (superficial) biological interpretation based on the structure of the visual cortex in animals. And third, we facilitate the detection of local patterns in image data that are likely to exhibit spatial correlations.
- Each (hidden) neuron in this architecture receives inputs from multiple neighboring pixels (and itself) and produces a single aggregate "pixel" as output. This approach to compute a function based on the aggregate function value of neighboring locations has a natural interpretation in terms of a *cross-correlation* or *convolution*. For discrete image data some of those convolutions correspond to specific filters that are applied to the image, and which we will discuss in the following.

Image Convolutions

- **discrete convolution** $f * g$ of functions f, g given as

$$(f*g)(x) := \sum_{i=-\infty}^{\infty} f(i)\cdot g(x-i) = \sum_{i=-\infty}^{\infty} f(x-i)\cdot g(i)$$

- for image $\mathbf{I} \in \mathbb{R}^{w\times h}$ we can use **convolution kernel** $\omega \in \mathbb{R}^{n\times n}$ to compute pixels based on neighboring pixels

$$\mathbf{I}'(x,y) := \sum_{i=1}^{n}\sum_{j=1}^{n} \omega_{ij}\cdot \mathbf{I}(x-i+c, y-j+c)$$

with $c \in \{1,\ldots,n\}$ index of center element in ω (e.g. $c=2$)

- depending on kernel, we can apply **blur, sharpening, edge detection, etc.**

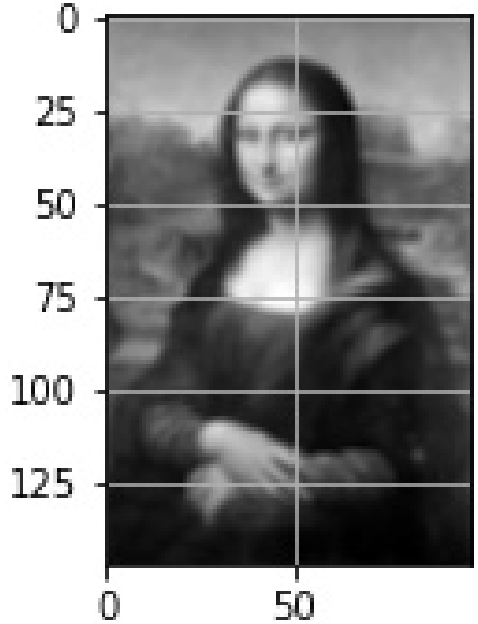


image after applying **box blur filter**

example: blur filter

$$\omega = \frac{1}{9}\cdot \begin{matrix}1\\2\\3\end{matrix}\begin{matrix}\begin{matrix}1 & 2 & 3\end{matrix}\\ \begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}\end{matrix}$$

Notes:

- We can consider an image as a function f that returns pixel values for different coordinates (x, y) that are given as discrete arguments. For two functions f and g we can generally define a discrete convolution between f and g, a mathematical operation that yields a new function that depends on the function values of f and g in neighboring locations. For real-valued functions, a convolution is actually identical to the negative cross-correlation between two functions. For the specific case of images with (x, y) coordinates, a discrete convolution can be defined based on a convolution kernel $\omega \in \mathbb{R}^{n \times n}$ of size n as given above. This convolution kernel is a matrix that tells us how to compute new pixel values based on the values of neighboring pixels.
- Certain convolution kernels translate to filters that have a straight-forward interpretation in image data that are also the basis for image processing applications. As an example, the kernel given on the slide above corresponds to a box blur operation. Other kernels corresponds to a sharpen operation that you may know from photo processing software.

Laplacian kernel

- consider **Laplacian operator** describing heat diffusion in **continuous Euclidean space** → Statistical Network Analysis, L12

$$\nabla f := \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- $\nabla f(x, y)$ captures how $f(x, y)$ deviates from average of f in neighborhood of (x, y)
- for discrete lattice, Laplacian operator corresponds to **Laplacian matrix** → L02
- we can use **Laplacian kernel** to **detect edges** in images

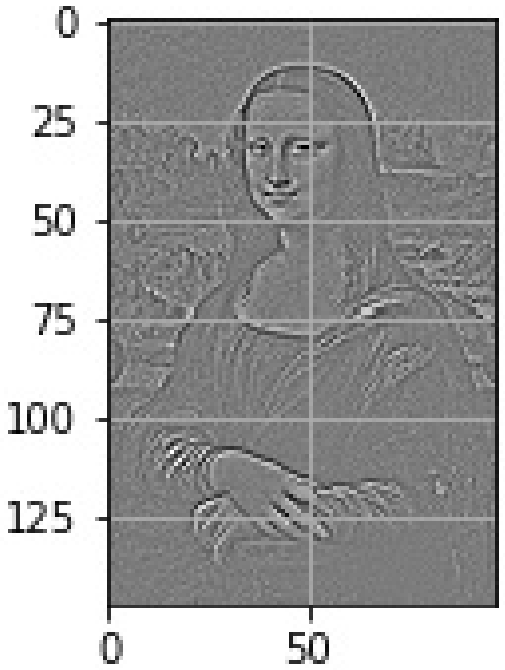


image after applying **Laplace filter**

Laplace filter

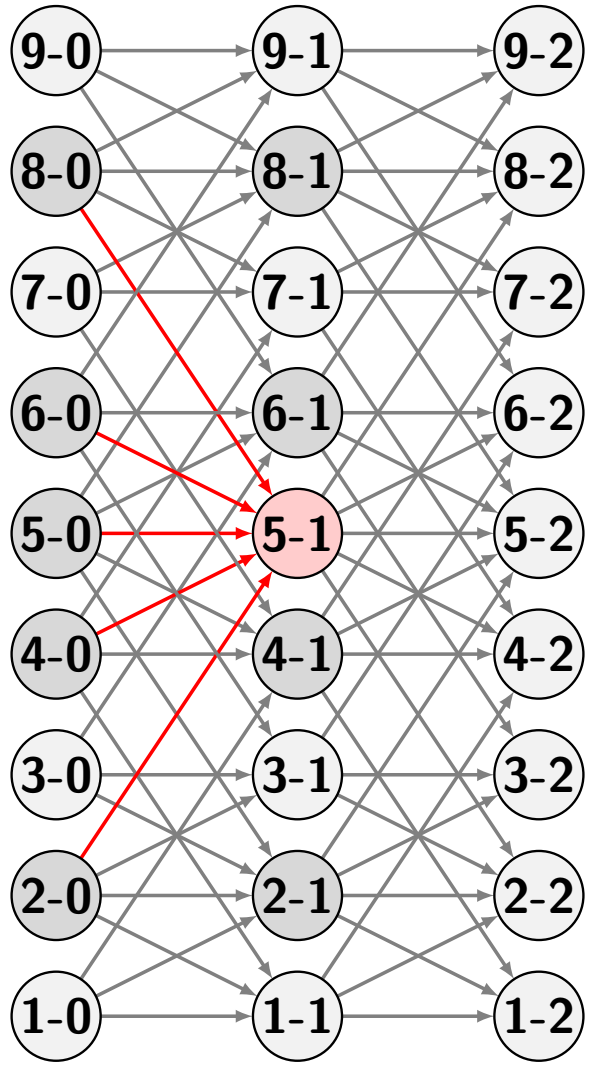
$$\omega = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \end{array}$$

Notes:

- Let us consider a special convolution kernel (or filter), the Laplacian kernel. One the one hand it gives an idea how convolutional neural networks are able to extract features from images. On the other hand it has an interesting relationship with concepts from spectral graph theory that we have discussed in Lecture 02.
- We start with an excursion to physics, where we use the Laplacian operator to describe heat diffusion in a continuous Euclidean space. In a continuous time and continuous space setting, the Laplacian operator is a differential operator that consists of the sum of the second-order partial derivatives in the different dimensions of the space. Above, we give the Laplacian operator for a two-dimensional, continuous Euclidean space. In a nutshell, if we evaluate the Laplacian operator of a function f at point (x, y) it tells us how much the function value $f(x, y)$ at this point deviates from the average value of f in the neighborhood of (x, y).
- In our course Statistical Network Analysis, we have shown that a reformulation of the continuous-time dynamical system in Euclidean space to a graph topology naturally gives rise to the Laplacian matrix, which is a discrete operator on a graph topology with discrete nodes and links (see SNA script in WueCampus, Lecture 12).
- Applying this idea to a lattice graph of pixel values, we can see that the Laplacian kernel is an discrete operator that captures how much the value of a pixel at position (x, y) deviates from the average pixel values in its neighborhood. This interpretation of the Laplacian kernel explains why we can use it for edge detection. Hence, we can use convolution kernel to extract (hierarchies of) “shapes” in images that can be used, e.g., for object or face recognition tasks.

Convolutional Neural Networks

- we can incorporate one or more **convolutional layers** into architecture of **deep neural network**
- each hidden neuron h_i in convolutional layer l applies **image convolution with learnable kernel** $\omega^{(l)}$
- perceptrons within one layer **share parameters**, i.e. we restrict model to learn **one convolution kernel per layer**
- we add additional (hidden) layers to **learn latent representation** of images and **generate output**

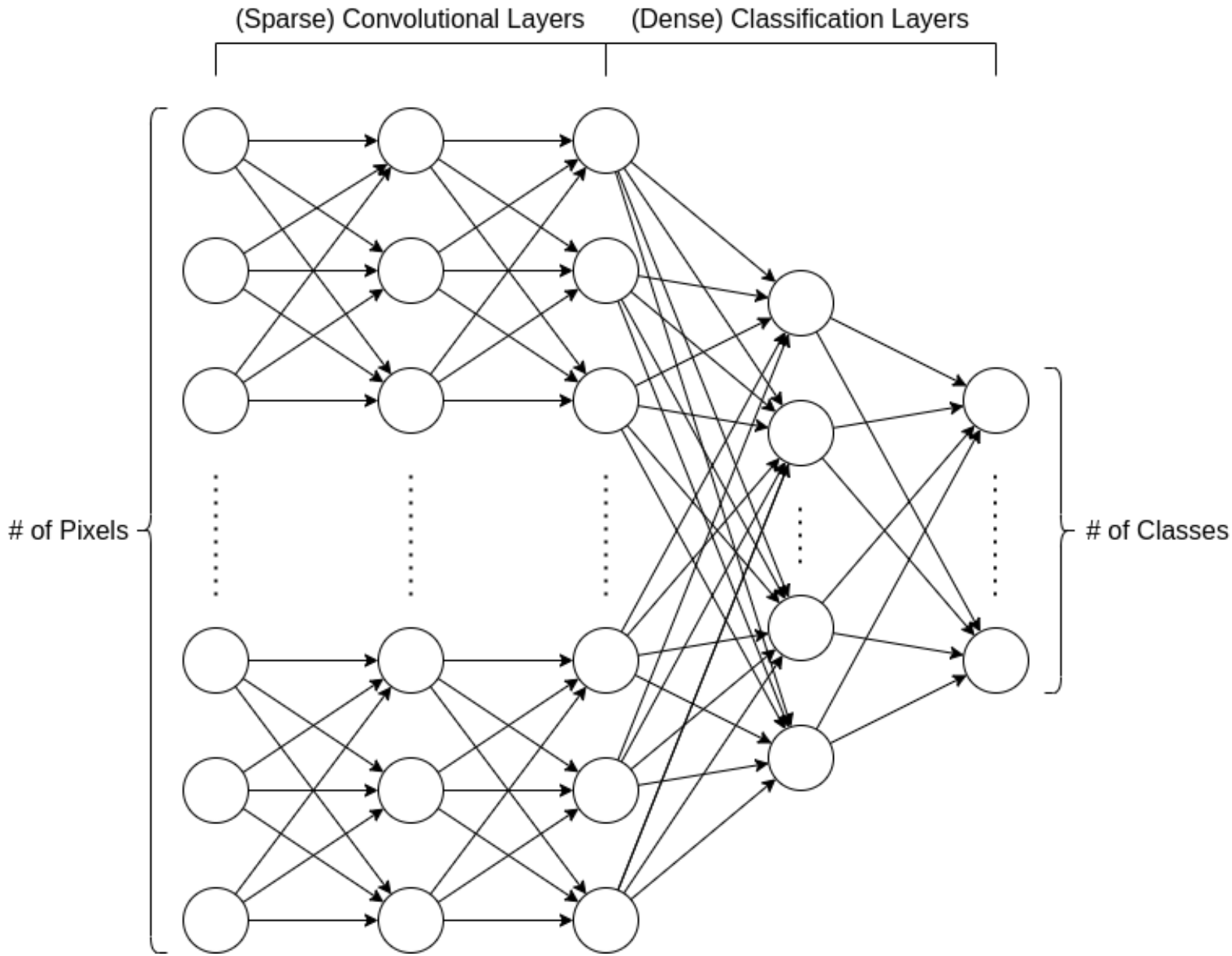


architecture of **convolutional neural network** for 3×3 pixel image

Notes:

- Building on the concept of image convolutions, which can be described as a matrix that is used to compute new pixel values, we can now address the question how a neuron in the hidden layer of a neural network can calculate a new pixel value based on the neighboring pixels. Each neuron just applies an image convolution with a learnable kernel ω, i.e. the entries of the kernel matrix are the learnable parameters of our model. To further limit the number of parameters, we further assume that each neuron in one (hidden) layer applies **the same** convolution, i.e. all neurons share the same learnable kernel parameters.
- We can add multiple subsequent layers that apply the convolution, which - by means of the topology of those layers- allows the information contained in one pixel of the image to propagate to neurons that correspond to pixels that are further away. In other words, the depth of the convolutional layers influences the type of patterns or spatial correlations that our neural network can learn.
- We further add one or more fully connected hidden perceptron layers with non-linear activation function, where we typically use a number of neurons that is much smaller than the number of pixels. This enables the neural network to capture arbitrary non-linear patterns and the activations of those neurons in the hidden layers can be viewed as representation of an image in a latent space.
- We call the first layers that apply the image convolution based on a learnable kernel **convolutional layer** and the overall architecture (including the additional hidden layers) is called a **convolutional neural network**.

CNN Architecture

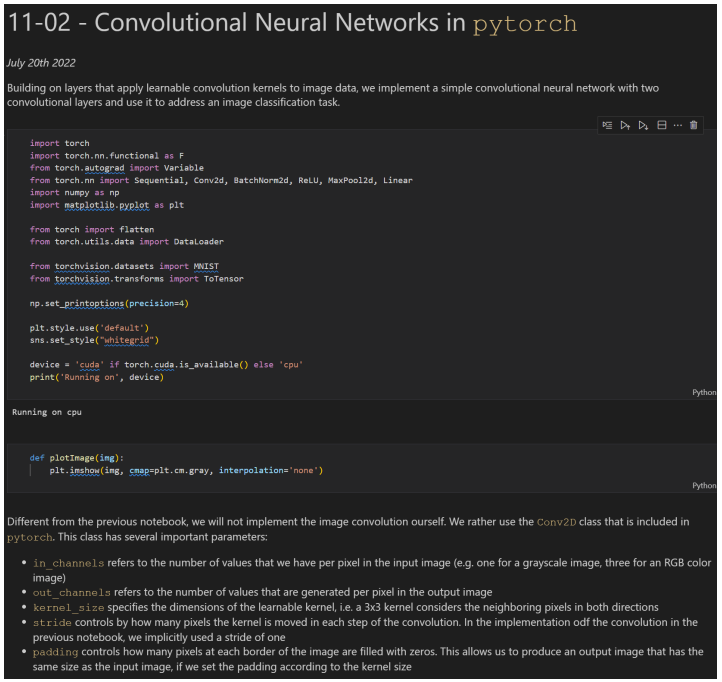


Notes:

- The figure above (thanks to Chester Tan) gives an overview of the architecture of a Convolutional Neural Network with two convolutional layers, one fully connected hidden layer and an output layer for an image classification task.

Practice session

- we define kernels and compute **convolutions in image data**
- we implement a **convolutional neural network** for an object recognition task in `pytorch`



11-02 - Convolutional Neural Networks in `pytorch`

July 20th 2022

Building on layers that apply learnable convolution kernels to image data, we implement a simple convolutional neural network with two convolutional layers and use it to address an image classification task.

```
import torch
import torch.nn.functional as F
from torch.autograd import Variable
from torch.nn import Sequential, Conv2d, BatchNorm2d, ReLU, MaxPool2d, Linear
import numpy as np
import matplotlib.pyplot as plt

from torch import flatten
from torch.utils.data import DataLoader

from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor

np.set_printoptions(precision=4)

plt.style.use('default')
sns.set_style("whitegrid")

device = 'cuda' if torch.cuda.is_available() else 'cpu'
print('Running on', device)
```

Python

```
Running on cpu
```

```
def plotImage(img):
    plt.imshow(img, cmap=plt.cm.gray, interpolation='none')
```

Python

Different from the previous notebook, we will not implement the image convolution ourself. We rather use the `Conv2D` class that is included in `pytorch`. This class has several important parameters:

- `in_channels` refers to the number of values that we have per pixel in the input image (e.g. one for a grayscale image, three for an RGB color image)
- `out_channels` refers to the number of values that are generated per pixel in the output image
- `kernel_size` specifies the dimensions of the learnable kernel, i.e. a 3x3 kernel considers the neighboring pixels in both directions
- `stride` controls by how many pixels the kernel is moved in each step of the convolution. In the implementation odf the convolution in the previous notebook, we implicitly used a stride of one
- `padding` controls how many pixels at each border of the image are filled with zeros. This allows us to produce an output image that has the same size as the input image, if we set the padding according to the kernel size

practice session

see notebooks 11-01 – 11-02 in `gitlab` repository at

→ `https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_sose_ml4nets_notebooks`

Notes:

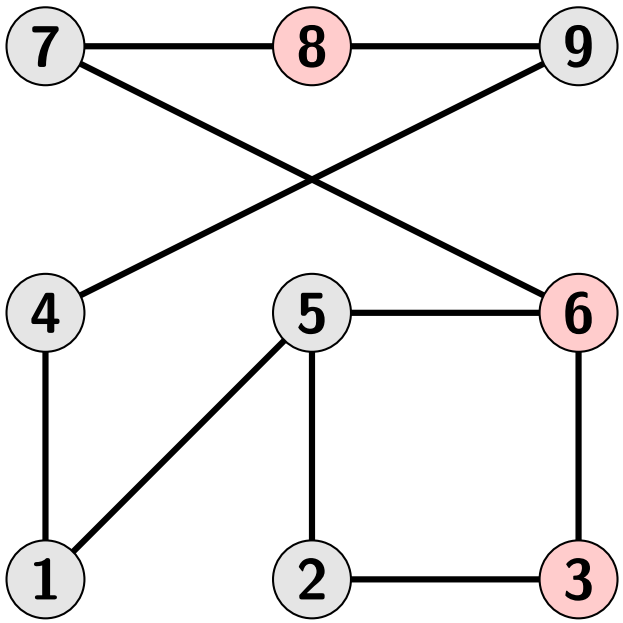
- In the first practice session we first implement a function that allows us to apply convolution kernels to image data. We then use `pytorch` to implement a convolutional neural network and show how we can apply it in a simple image recognition task.

From Image Data to Graph Neural Networks

- how can we apply neural networks to **graph-structured data**?

similarities between image and graph data?

- fully connected layers for large graph $\Rightarrow$ **curse of dimensionality**
- we want to incorporate information both from **(graph) topology and (node) features**
- node features are likely to exhibit **topological correlations**



- we can use graph topology to define **graph neural network (GNN)** $\rightarrow$ F Scarselli et al. 2009
- **message passing layer** updates node features based on features of neighboring nodes
- first k layers aggregate information along **paths up to length** k

Notes:

- Since our course is on machine learning for complex networks, you may ask yourself why we consider convolutional neural networks for image data. If we want to directly apply neural networks to graph-structured data we face similar challenges as in the application of neural networks to image data. To better understand this let us consider a supervised node classification setting, where nodes i have features $x_i \in \mathbb{R}^d$ and belong to discrete classes C. Taking a naive approach, we could connect the input from each node to all neurons in our hidden layers, which means that each of the neurons has $|V| \cdot d$ learnable weights. This again introduces the curse of dimensionality. Moreover, we need to answer the question how we can input both information on the graph topology and the node features and we want to be able to utilize potential topological correlations between features of different nodes, e.g. nodes in the same cluster exhibiting similar features.
- To define the topology of a convolutional neural network, we considered images as lattice network of pixel values, where pixels with adjacent pixel coordinates are connected by a link. Here we can use the same approach: We encode the graph topology in the connections between neurons in the first layer(s) of a neural network, i.e. we use the topology of a graph to define the neural network.
- Convolutional neural networks address the curse of dimensionality by applying a convolution filter, which aggregate information of a pixel with the neighboring pixels of its neighbors and feed the result into additional hidden layers that learn a latent representation of the image. We can use the same idea in graphs, i.e. we can apply "convolution filters" to graphs. We can implement them based on **message passing**, where nodes repeatedly exchange features with neighbors.

Neural Message Passing

- GNNs build on **message passing algorithm** that uses graph to **update node states**

  → J Gilmer et al. 2017

- consider discrete-time dynamics where $h_i^{(t)} \in \mathbb{R}^d$ denotes **state of node i at time t**

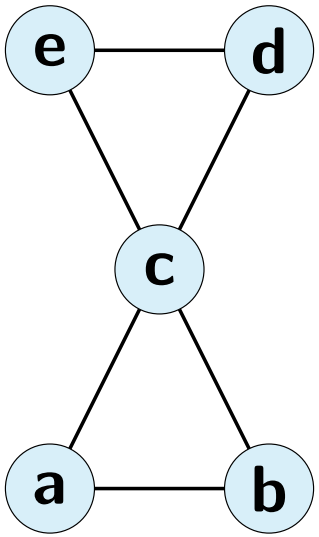
- nodes update their state $h_i^{(t)}$ based on **states of their neighbors**, i.e.

$$h_i^{(t)} = F_{j \in N(i)} h_j^{(t-1)}$$

  where F is **aggregation function** and $N(i)$ is set of neighbors of i

- for **add aggregation** we get update rule

$$h_i^{(t)} = \sum_{j \in N(i)} h_j^{(t-1)}$$



add aggregation rule

node	$t=0$	$t=1$	$t=2$
a	1	5	16
b	2	4	17
c	3	12	24
d	4	8	19
e	5	7	20

Notes:

- Let us have a closer look at **neural message passing**, which is the foundation of **graph neural networks**. The basic idea is to define message passing layers that use the topology of a graph to update node states based on the states of their neighbors in the graph. Each message passing layer of the graph neural network performs one round of message passing. More formally, this defines a discrete-time dynamical system where we use $h_i^{(t)}$ to denote the state of a node i after t rounds of message passing. The initial state $h_i^{(t=0)}$ is given by the input features of our graph neural network, and for $t > 0$ $h_i^{(t)}$ represents the state of a hidden neuron associated with node i.
- In each step t of the message passing, a node i calculates a new state $h_i^{(t)}$ based on the previous state $h_i^{(t-1)}$ of its neighbors $j \in N(i)$, which requires us to apply some aggregation function.
- Considering a simple add aggregation, we obtain the simple update rule above, which we apply two times in the toy example network. The initial states of the nodes for $t = 0$ are given as inputs of the neural message passing algorithm and the outputs after two rounds of message passing are given for $t = 2$.

Neural Message Passing

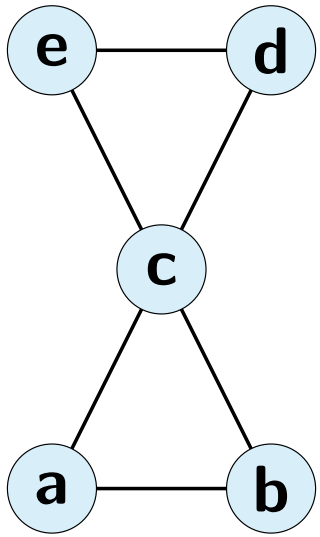
- for networks without self-loops, nodes do not consider their **own prior state**
- to avoid this, we explicitly **add self-loops**

$$h_i^{(t)} = \sum_{j \in N(i) \cup \{i\}} h_j^{(t-1)}$$

- we additionally transform updated node state with **differentiable function** g, i.e.

$$h_i^{(t)} = g\left(\sum_{j \in N(i) \cup \{i\}} h_j^{(t-1)} \right)$$

- message passing is **permutation equivariant**, i.e. node permutation $\rightarrow$ consistent permutation of outputs $h_i^{(t)}$



add aggregation with self-loops and $g(x) = 0.5 + 2 \cdot x$

node	$t = 0$	$t = 1$	$t = 2$
a	1	12.5	111.5
b	2	12.5	111.5
c	3	30.5	209.5
d	4	24.5	159.5
e	5	24.5	159.5

Notes:

- If we carefully inspect the output of the message massing, we notice that for networks without self-loops, nodes do not remember their **own** prior state when computing the next state based on their neighbors. This is unfortunate, as it means that at least after one round of message passing (and possible even after multiple rounds if the network does not contain short loops), the information originating in node i is not available to node i itself. This may remind you of some issues with the periodicity of Markov chains that we discussed in our coverage of random walks.
- The solution to this issue is similar than the solution that we took to ensure the aperiodicity of a Markov chain: we explicitly add self-loops, which means that nodes consider the prior state of their neighbors and their own prior state.
- A major difference of this aggregation scheme is that - so far - we only add up the states of our neighbors, which means that different from the convolutional neural network there are no "learnable" parameters for our model. To address this, we can include an additional transformation of each node state by means of an arbitrary differentiable function g. We will eventually use a perceptron (with learnable weights and bias parameter) and a non-linear activation function to compute this transformation. Above, we apply a simple linear transformation by a function $g(x) = 0.5 \cdot 2x$.
- Message passing is a **permutation equivariant** operation, i.e. if we change the ordering of rows/columns in the adjacency matrix (i.e. we change the ordering of nodes) we get a consistent reordering of the resulting values after the message passing, i.e. the resutling node states are the same, they are just differently ordered.

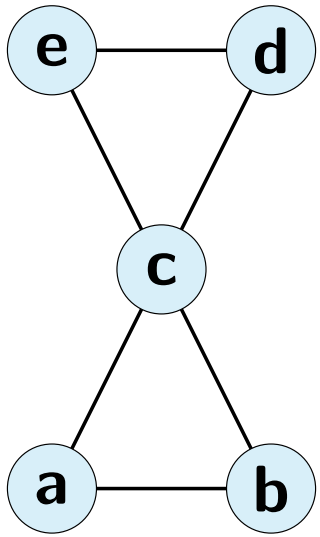
Degree-based Normalization

- heterogeneity of networks requires application of **degree-based normalization**
- we can use **mean rather than add aggregation**, i.e.

$$h_i^{(t)} = g\left(\sum_{j \in N(i)} \frac{h_j^{(t-1)}}{d_i}\right)$$

- we can apply **symmetric degree-based normalization**, i.e.

$$h_i^{(t)} = g\left(\sum_{j \in N(i)} \frac{h_j^{(t-1)}}{\sqrt{d_i d_j}}\right)$$



symmetric normalization (and self-loops)

node	$t=0$	$t=1$	$t=2$
a	1	1.8	2.1
b	2	1.8	2.1
c	3	2.7	3.6
d	4	3.8	3.5
e	5	3.8	3.5

Notes:

- In the example on the previous slide, we observe another important difference between (many) graphs and image data. While the number of neighboring pixels for each image position is the same (with the exception of border pixels that we can exclude from the convolution), nodes in graphs can have highly heterogeneous degrees. If we simply add up the states of neighbors we will get largely heterogeneous states that are also strongly correlation with the node degrees. This hinders learning in networks with heterogeneous degrees and requires degree-based normalization techniques. A simple approach would be to use the mean rather than the sum as an aggregation function, i.e. each node normalizes the values based on its degree, i.e. the number of neighbors from which it receives a state or message. This simple normalization scheme has been applied in some early works on graph neural networks.
- We can alternatively use a **symmetric degree-based normalization** where we scale the messages received from each neighbor individually. As normalization factor, we use the geometric mean of the degrees of the node i and the respective neighbor node j.
- In the example above, we used the symmetric degree-based normalization in two rounds of message passing. For the sake of simplicity, we did not apply an additional transformation, i.e. we simply used $g(x) = x$.

Message Passing and Graph Laplacians

- for graph with adjacency matrix $\mathbf{A}$ consider **Laplacian matrix** → L02

$$\mathcal{L} := \mathbf{D} - \mathbf{A}$$

where $\mathbf{D}$ is diagonal degree matrix

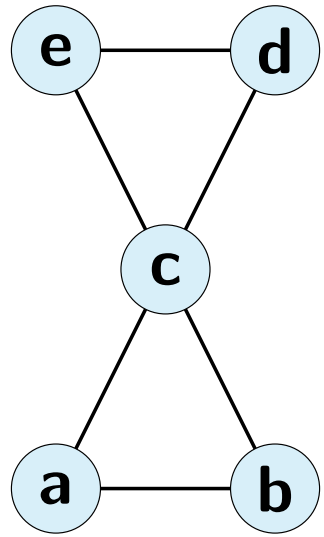
- symmetric degree-based normalization yields **symmetric normalized Laplacian**

→ F Chung, 1997

$$\mathcal{L}^* = \mathbf{D}^{\frac{1}{2}} \mathcal{L} \mathbf{D}^{\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$$

with entries

$$\mathcal{L}^*_{ij} = \begin{cases} -\frac{1}{\sqrt{d_i \cdot d_j}} \text{ if } i \neq j \text{ and } A_{ij} = 1 \\ 1 \text{ if } i = j \\ 0 \text{ else} \end{cases}$$



Symmetric Normalized Laplacian

$$\mathcal{L}^* = \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2\sqrt{2}} & 0 & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2\sqrt{2}} & 0 & 0 \\ -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 1 & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \\ 0 & 0 & -\frac{1}{2\sqrt{2}} & 1 & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2\sqrt{2}} & -\frac{1}{2} & 1 \end{pmatrix}$$

Notes:

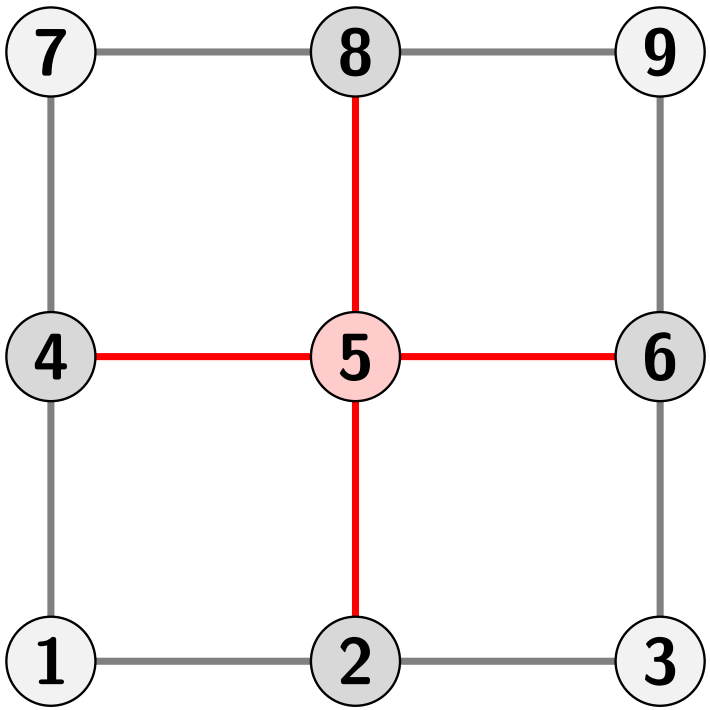
- Let us comment on some interesting links between neural message passing with symmetric degree-based normalization and spectral graph theory. In lecture 02, we have introduced the Laplacian matrix, which can be seen as a discrete generalization of the continuous Laplacian operator in Euclidean space to arbitrary topologies. We can also use the Laplacian operator to model continuous time diffusion dynamics (for more details see Statistical Network Analysis, L12).
- In matrix form, the Laplacian matrix is defined as difference between the degree-diagonal and the adjacency matrix. The normalization term in the symmetric degree-based normalization naturally corresponds to the so-called symmetric normalized Laplacian. In matrix form, it can be given as the difference between an identity matrix and the product $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. Note that the degree diagonal matrix is invertible if we do not have nodes with zero degrees. We find that, except for the sign, the entries in the symmetric normalized Laplacian correspond to the factors that are used in the neural message passing with symmetric degree normalization.

Spectral Graph Convolution

- we used **eigenvectors corresponding to smallest Laplacian eigenvalues** to embed graph in Euclidean space → L08
- we can use eigenvectors of graph Laplacian to **generalize convolutional neural networks to graph data**

  → J Bruna et al., 2013
- neural message passing with self-loops and symmetric degree-based normalization can be viewed as **efficient localized version of spectral graph convolution**

  → T Kipf and M Welling, 2017
- inclusion of **non-local topological patterns** requires **multiple message passing layers**



Notes:

- We can thus view multiple rounds of neural message passing with symmetric degree-based normalization as a model for the diffusion of node features in a graph. In Lecture 12 of our course Statistical Network Analysis, we have seen that we can use the eigenvalues and eigenvectors of the Laplacian matrix to characterize such a diffusion process, where the eigenvalues determine the speed of the diffusion and the eigenvectors give the different “modes” of the diffusion dynamics. Moreover, in Lecture 08 of our course we used the Laplacian eigenvectors corresponding to the smallest eigenvalues to represent graphs in a Euclidean space. Referring to our discussion of PCA, this can be viewed as a representation of nodes and edges in a high-dimensional Euclidean space with suitably rotated and ordered dimensions. As shown in → J Bruna et al. 2013 , we can actually use such a vector space representation to generalize convolutional neural networks to spectral graph convolutional networks.
- While this is an interesting approach to generalize neural networks to graph data, it comes at the prize that we need to calculate the eigenvalues and eigenvectors of a potentially very large matrix. The neural message passing algorithm with self-loops and symmetric degree-based normalization (which links it to a Laplacian matrix) can actually ve viewed as a localized version of such a spectral graph convolution, which can be efficiently calculated in linear time.
- Naturally, a single round of message passing only allows information to propagate along a distance of one in the network. If we want to utilize non-local patterns in the graph, we thus need multiple subsequent message passing layers. Unfortunately, if we use a number of layers that is too large, we suffer the problem of over-smoothing

Graph Convolutional Networks (GCN)

- message passing with self-loops and symmetric degree-normalization defines **Graph Convolutional Networks (GCN)** → T Kipf, M Welling, 2016

- update rule in **message passing layer of GCN** given as

$$h_i^{(k)} := \sigma\left(\mathbf{W}^{(k)} \sum_{j \in N(i) \cup \{i\}} \frac{h_j^{(k-1)}}{\sqrt{d_i d_j}}\right)$$

  where $\mathbf{W}^{(k)} \in \mathbb{R}^{d^{(k)}} \times d^{(k-1)}$ are **learnable weights** and σ is **non-linear activation function**

- message passing layer k maps node representations $h_i^{(k-1)} \in \mathbb{R}^{d^{(k-1)}}$ to $h_i^{(k)} \in \mathbb{R}^{d^{(k)}}$

- similar to CNN, we can add **hidden layer(s) and output layer**

Thomas Kipf

Published as a conference paper at ICLR 2017

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

ABSTRACT

We present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. We motivate the choice of our convolutional architecture via a localized first-order approximation of spectral graph convolutions. Our model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on citation networks and on a knowledge graph dataset we demonstrate that our approach outperforms related methods by a significant margin.

1 INTRODUCTION

We consider the problem of classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes. This problem can be framed as graph-based semi-supervised learning, where label information is smoothed over the graph via some form of explicit graph-based regularization (Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2006; Weston et al., 2012), e.g. by using a graph Laplacian regularization term in the loss function:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \|f(X_i) - f(X_j)\|^2 = f(X)^\top \Delta f(X), \qquad (1)$$

Here, $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph, $f(\cdot)$ can be a neural network-like differentiable function, λ is a weighing factor and X is a matrix of node feature vectors X_i. $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $\mathcal{G} = (V, \mathcal{E})$ with N nodes $v_i \in V$, edges $(v_i, v_j) \in \mathcal{E}$, an adjacency matrix $A \in \mathbb{R}^{N \times N}$ (binary or weighted) and a degree matrix $D_{ii} = \sum_j A_{ij}$. The formulation of Eq. 1 relies on the assumption that connected nodes in the graph are likely to share the same label. This assumption, however, might restrict modeling capacity, as graph edges need not necessarily encode node similarity, but could contain additional information.

In this work, we encode the graph structure directly using a neural network model $f(X, A)$ and train on a supervised target $\mathcal{L}_0$ for all nodes with labels, thereby avoiding explicit graph-based regularization in the loss function. Conditioning $f(\cdot)$ on the adjacency matrix of the graph will allow the model to distribute gradient information from the supervised loss $\mathcal{L}_0$ and will enable it to learn representations of nodes both with and without labels.

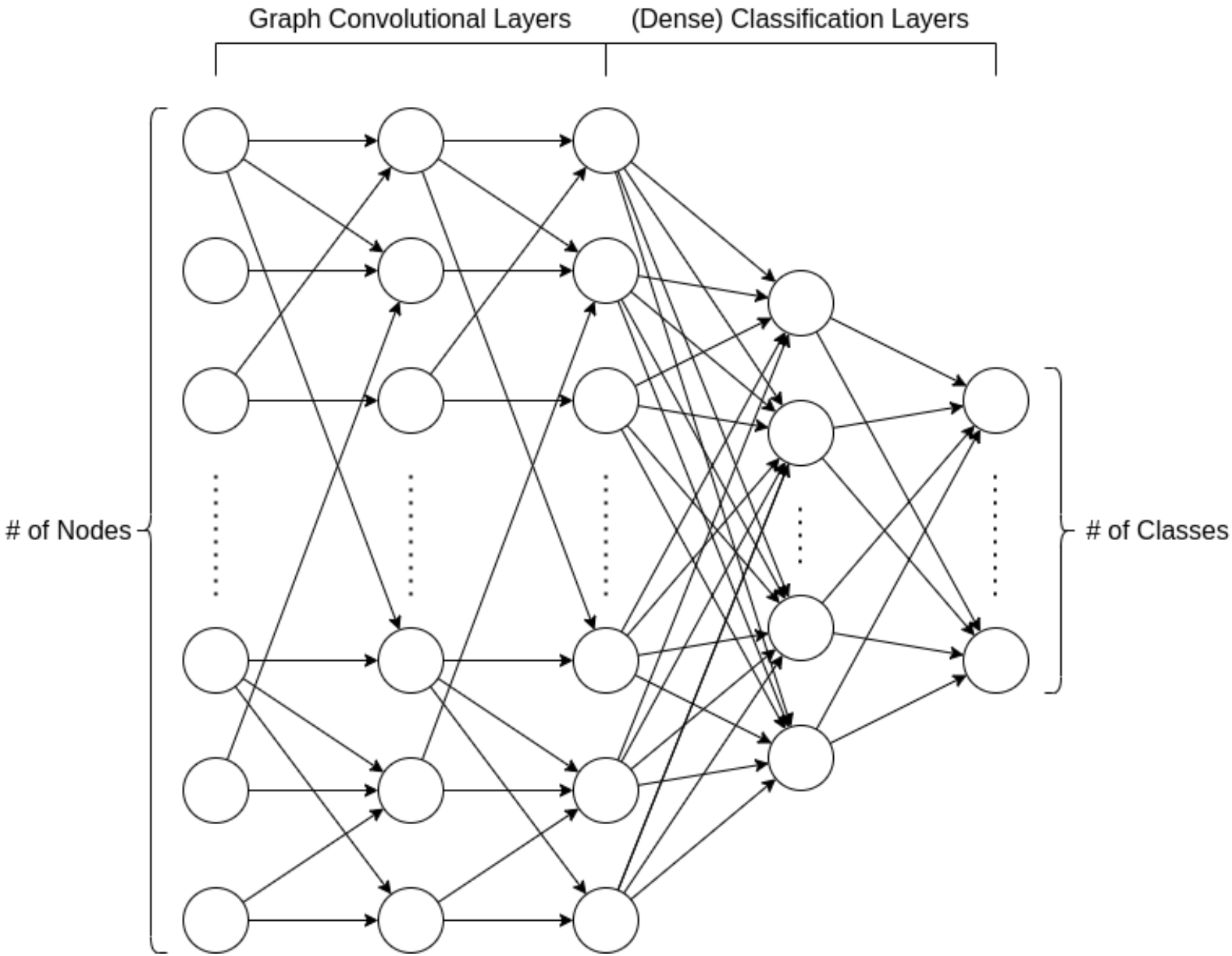
Our contributions are two-fold. Firstly, we introduce a simple and well-behaved layer-wise propagation rule for neural network models which operate directly on graphs and show how it can be motivated from a first-order approximation of spectral graph convolutions (Hammond et al., 2011). Secondly, we demonstrate how this form of a graph-based neural network model can be used for fast and scalable semi-supervised classification of nodes in a graph. Experiments on a number of datasets demonstrate that our model compares favorably both in classification accuracy and efficiency (measured in wall-clock time) against state-of-the-art methods for semi-supervised learning.

image credit:
https://tkipf.github.io/

Notes:

- The neural message passing algorithm with self-loops and symmetric degree-based normalization is the basis for Graph Convolutional Networks, an architecture first proposed in 2016 by Thomas Kipf and Max Welling. We can write the update rule for each message passing layer of a GCN as given above, where we have included a perceptron-based transformation with learnable weights and a subsequent application of a non-linear activation function. Like for CNNs, all neurons in a single message passing layer share the same parameters. Depending on the weight parameters, we can further change the dimensionality of the node states/features as we pass the information through the message passing layers.
- The number of message passing layers k determines the depth of the GCN, where deeper GCNs aggregate information from a longer distance in the graph. We further add one or more layers of fully connected feed-forward networks, where the hidden neuron activations can be interpreted as latent representations of the nodes in the network. We finally add an output layer.
- We can train the network as before, i.e. we pass the inputs to the network, compute the loss function based on the output and the ground truth, and use backpropagation and stochastic gradient descent to optimize the parameters.

GCN Architecture

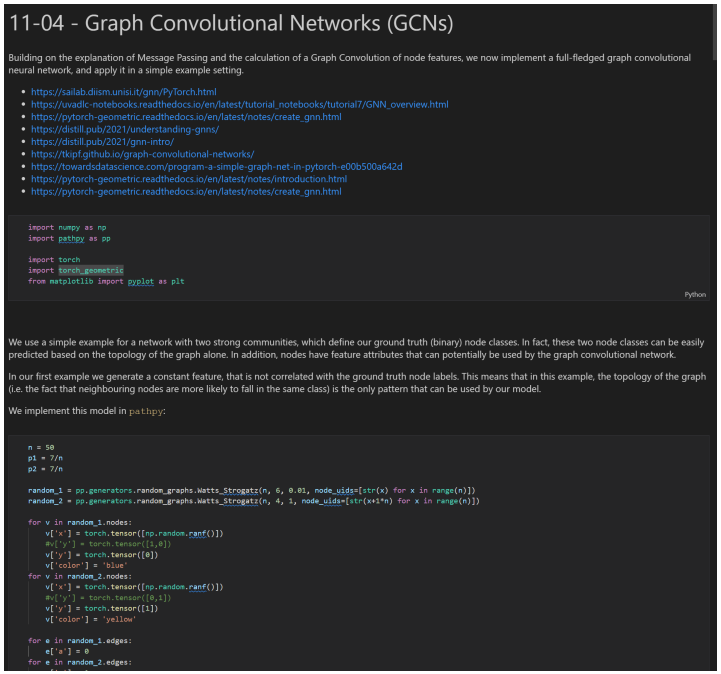


Notes:

- The figure above (thanks to Chester Tan) gives an overview of the architecture of a Graph Convolutional Neural Network with two graph convolutional (i.e. message passing) layers, one fully connected hidden layer and an output layer for a node classification problem.

Practice session

- we introduce the geometric deep learning package **torch-geometric** and convert `pathpy` networks to `torch-geometric` data structures
- we explore **neural message passing** with `torch-geometric`
- we implement **graph convolutional networks** and use them for node classification
- we show how we can use hidden layer activations to visualize **node feature maps**



practice session

see notebooks 11-03 and 11-04 in `gitlab` repository at

→ `https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_sose_ml4nets_notebooks`

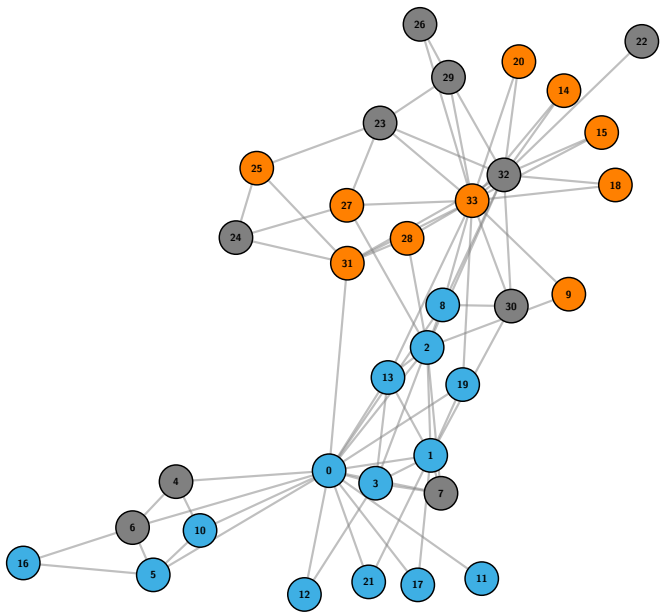
Notes:

- In the second practice session, we introduce the deep learning package `torch-geometric` (pyG) and show how we can convert `pathpy` networks to `torch` data structures.
- We implement and test neural message passing with `torch-geometric` and develop a simple graph convolutional network (GCN). We then apply it to a test data set and show that we can use hidden layer activations to visualize node feature maps.

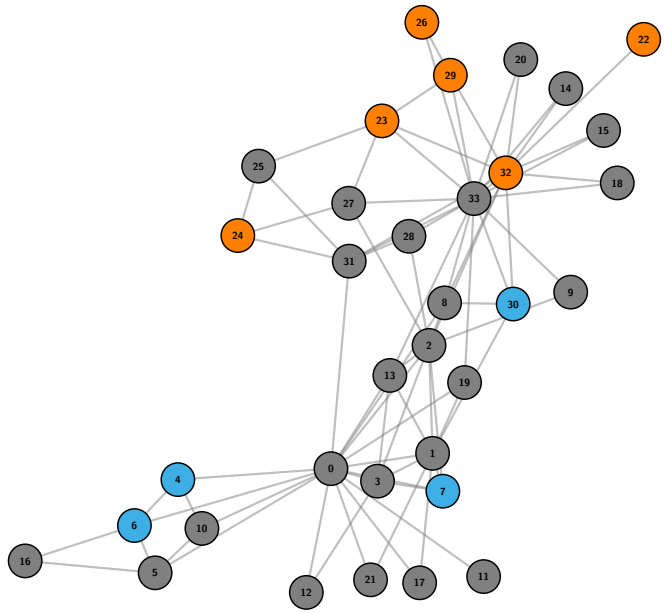
Example: GCN-based Node Classification

example

Karate club network with $n = 34$ nodes and $m = 77$ links, where ground truth node classes $\hat{y}$ are given by groups



training network with 70 % labeled nodes



predicted node classes in test set (accuracy 90%)

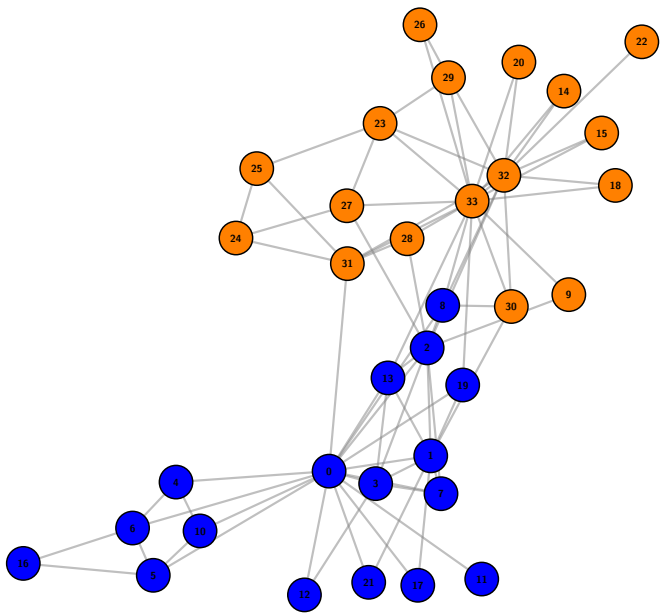
Notes:

- The example above demonstrates the application of a GCN in a supervised node classification task. We use the Karate club network, which provides information on ground truth classes based on two factions in the Karate club. We use the full graph to define the topology of the GCN and use 70 % of the nodes to train the GCN. We then use the trained model to predict node classes for the remaining nodes. In this example, with the exception of the single node 30, the predicted classes match the ground truth labels.

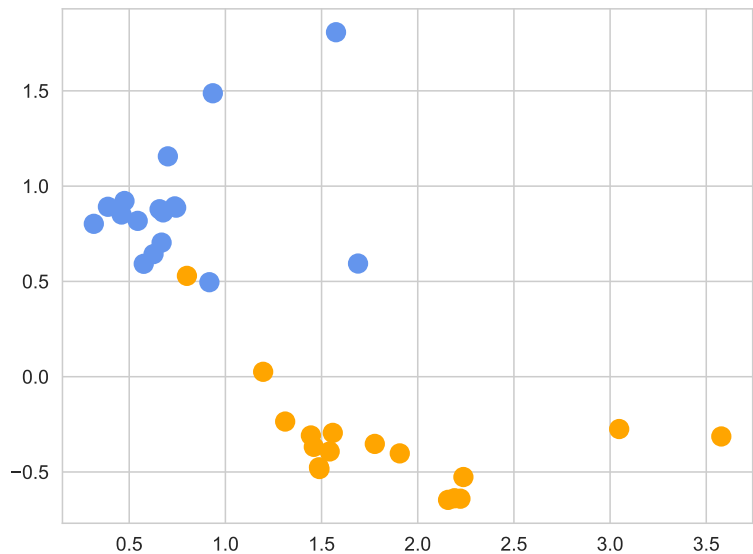
Example: Latent Node Representations

example

Karate club network with $n = 34$ nodes and $m = 77$ links, where ground truth node classes $\hat{y}$ are given by groups



Karate club network with ground truth node labels



latent representation of nodes extracted from activations in first hidden layer ($d = 16$) of GCN (representation in $\mathbb{R}^2$ via Truncated SVD)

Notes:

- We can also use GCNs to extract latent representations of nodes. For this, instead of the output of the model, we simply use the hidden layer activations that we get if we feed in the one-hot encoding of a node. Above, we have done this for all one-hot encodings of nodes for a two-layer GCN where the first layer has 16 hidden dimensions. We applied a subsequent truncated SVD to map those 16-dimensional vectors to $\mathbb{R}^2$.

Graph-Structured Data with Node Features

- consider training of GCN on graph $G = (V, E)$ with n nodes and target node labels y_i
- inputs $x_i := h_i^{(0)}$ of neurons in first message passing layer can be given as

$$x_i := (\underbrace{0, \ldots, 0}_{\text{i times}}, 1, 0, \ldots 0) \in \mathbb{R}^n$$

  i.e. we use **one-hot encoding of nodes** to initialize node states $h_i^{(k)}$ for first graph convolution layer

- for graphs with **additional node features** $f : V \to \mathbb{R}^d$ we can concatenate features with one-hot encoding, i.e.

$$x_i := (\underbrace{0, \ldots, 0}_{\text{i times}}, 1, 0, \ldots 0, f(i)) \in \mathbb{R}^{n+d}$$

- allows us to train GCN based on **graph topology** and **node features**

Notes:

- In our applications of the GCN model so far, we used one-hot-encodings $x_i \in \mathbb{R}^n$ of nodes as input to the first graph convolutional layer. Hence, the state of each node i in the first message passing layer is initialized with an n-dimensional vector that has a one at position i and zero elsewhere.
- For networks where we have additional node features $f(i) \in \mathbb{R}^d$, we can concatenate the feature of node i to this one-hot encoding, i.e. we get an input with $n + d$ dimensions, where d is the dimensionality of the node features. We now initialize the state of each node i in the first message passing layer with an $n + d$ dimensional vector, where the first n entries are the one-hot-encoding of node i and the last d entries contain the associated feature of node i.
- We can train the GCN in the same way as before.

Semi-supervised Learning in Graphs

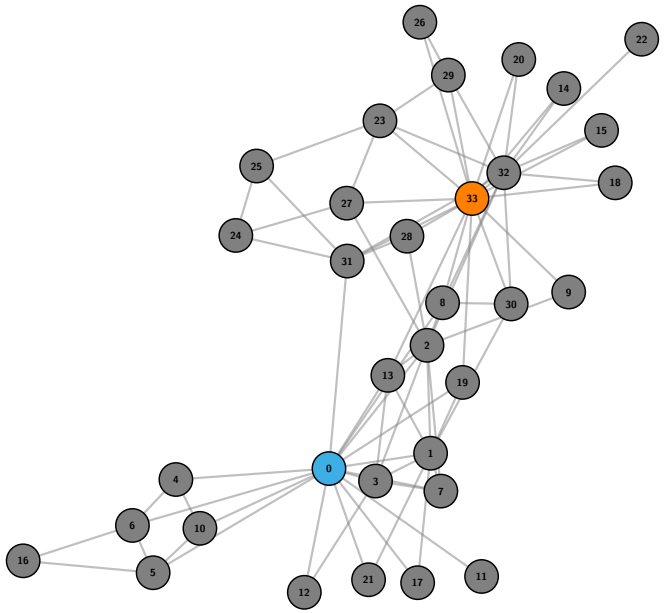
- use of topological features enables application of GCN to **semi-supervised learning** in graphs

semi-supervised learning

machine learning techniques that can simultaneously use **large amounts of unlabeled data** as well as **small amounts of labeled data**

example

semi-supervised node classification in network with a single labeled node per class



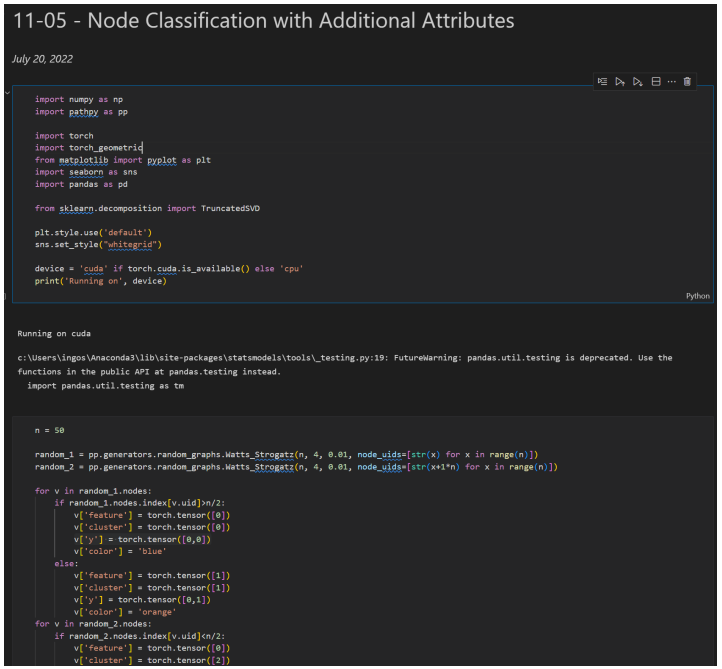
- message passing layers **smoothen** existing labels across unlabeled nodes close to labeled ones

Notes:

- We finally highlight that, thanks to their ability to utilize information from the topology of the graph, GCNs naturally support **semi-supervised learning**, i.e. settings where we have a large amount of unlabeled data as well as a very small number of labeled data.
- as an example, consider semi-supervised node classification in the Karate club network, where we only have a single labeled node for each community. A GCN can use the information provided by those labeled nodes as well as the community structures in the graph to predict the labels of the remaining nodes with high accuracy. This is due to the fact that the message passing layers effectively “smoothen” existing labels across the unlabeled nodes in the graph, where nodes with many paths to a node with a given label will be assigned that label with higher probability.
- Technically, we can handle such data sets by applying the message passing and calculating the loss function only for those nodes that are labeled.

Practice session

- we use a GCN for supervised learning in **graph with additional node features**
- we demonstrate that GCNs learn **patterns both in the graph topology and in node features**
- we use a **graph convolutional network** to address semi-supervised node classification



practice session

see notebooks 11-05 and 11-06 in `gitlab` repository at

→ `https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_sose_ml4nets_notebooks`

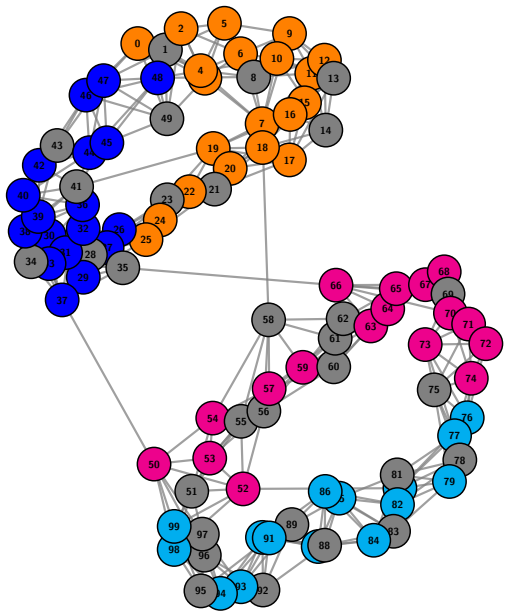
Notes:

- In the last practice session of this week, we use GCNs to address node classification in a network with additional node features.
- We further use a GCN to address semi-supervised node classification.

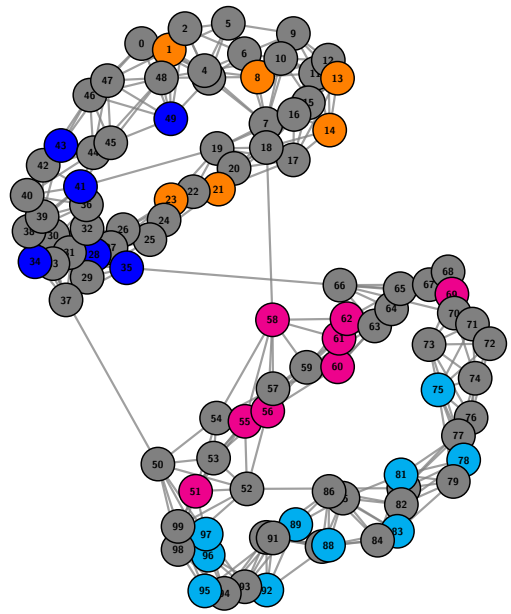
Example: Learning with node features

example

- synthetic network with **four ground-truth clusters** C_1, C_2, C_3, C_4 and **two topological communities** $M_1 = C_1 \cup C_2$ and $M_2 = C_3 \cup C_4$
- nodes i in C_1 and C_3 have feature $f(i) = 0$
- nodes j in C_2 and C_4 have feature $f(j) = 1$



training network with 70 % labeled nodes (test nodes in gray)



predicted node classes in test set (ttraining nodes in gray)

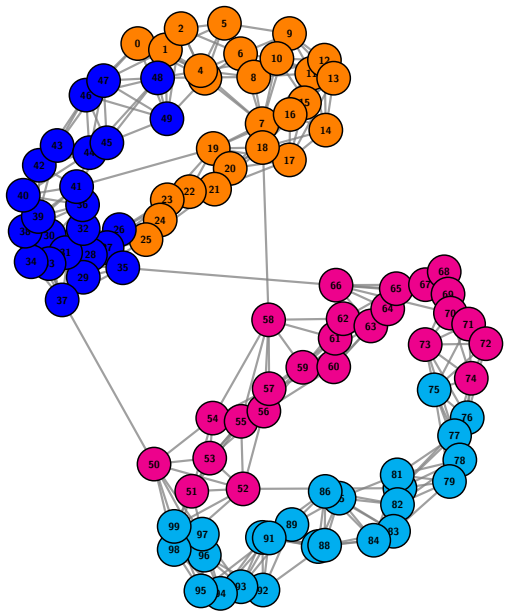
Notes:

- For the example above, we have generated a synthetic network with two strong communities (based on two interconnected Watts-Strogatz networks). We further assign two different features (0 or 1) to nodes, half of the nodes in each community are assigned label 0, while the other half is assigned label 1. This defines four ground truth clusters depending on (i) the cluster membership, and (ii) the node feature. Neither the community structure nor the node features are enough to correctly predict all four cluster labels.
- We find that the GCN is able to handle this example, where we have a mix of information that is due to the topology of the graph as well as to the node features.

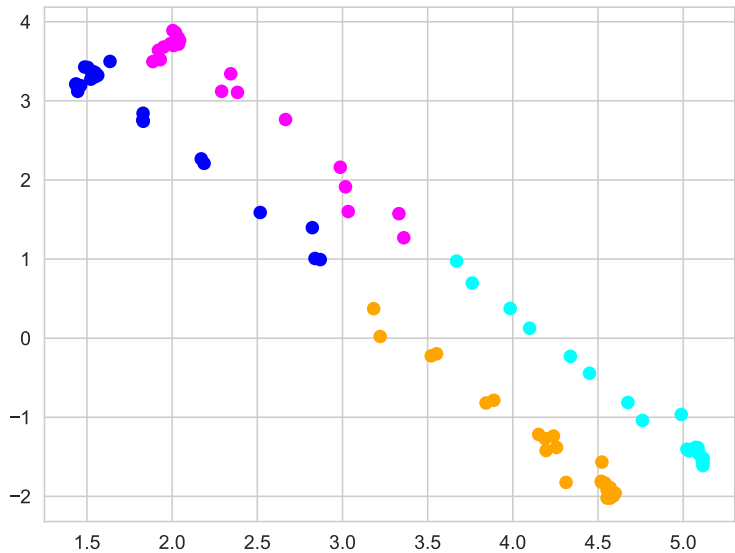
Example: Learning with node features

example

- synthetic network with **four ground-truth clusters** C_1, C_2, C_3, C_4 and **two topological communities** $M_1 = C_1 \cup C_2$ and $M_2 = C_3 \cup C_4$
- nodes i in C_1 and C_3 have feature $f(i) = 0$
- nodes j in C_2 and C_4 have feature $f(j) = 1$



synthetic network with ground truth clusters



latent node representation extracted from activations in hidden layer of GCN (representation in $\mathbb{R}^2$ via Truncated SVD)

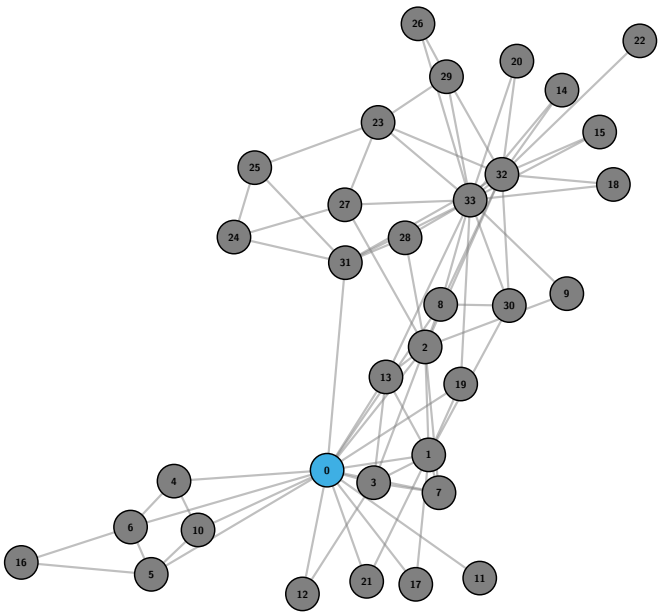
Notes:

- What is even more interesting, we can use the hidden layers of a GCN to recover these two independent dimensions of information. Above, we show a latent space embedding of nodes generated based on the hidden neuron activations in the first convolutional layer of a two-layer GCN.
- We further apply a truncated SVD to obtain a two-dimensional Euclidean representation. We find that the two different dimensions in the resulting representation capture the topological dimension (i.e. communities, from bottom left to top right) as well as the feature dimension (i.e. feature 0 or 1, top left to bottom right) in the graph.
- The GCN is able to generate a latent representation that incorporates both dimensions and uses them for the classification.

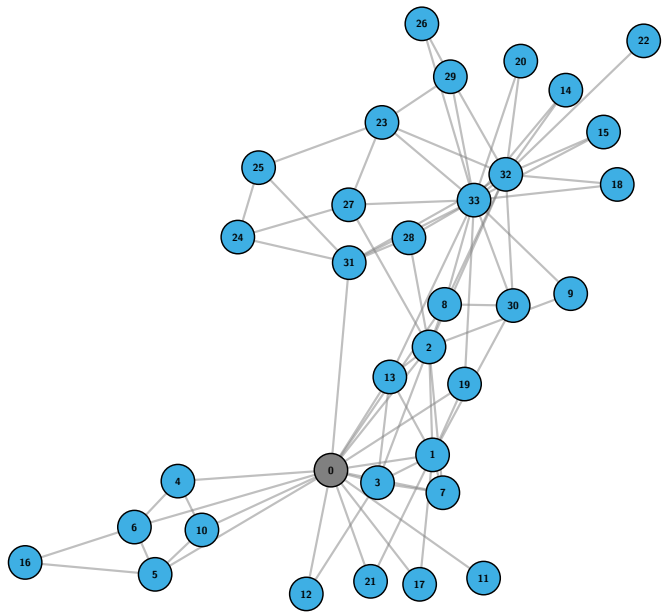
Example: Semi-Supervised Graph Learning 1/2

example

- **semi-supervised node classification** in Karate club network with $n = 34$ nodes and $m = 77$ links
- ground truth node class given for **one node in one community**



training network with **single labeled node**



predicted node classes in test set using GCN with single message passing layer

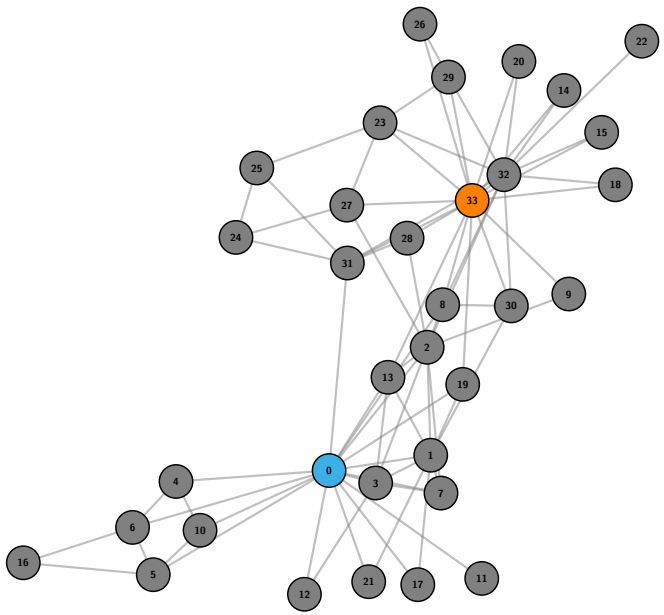
Notes:

- We finally demonstrate the performance of a GCN in a semi-supervised node classification scenario. We start with a setting where all except one nodes are unlabeled. We train the GCN based on this single labeled node. Not surprisingly, the trained GCN predicts the class of this labeled node for all other nodes in the test set.

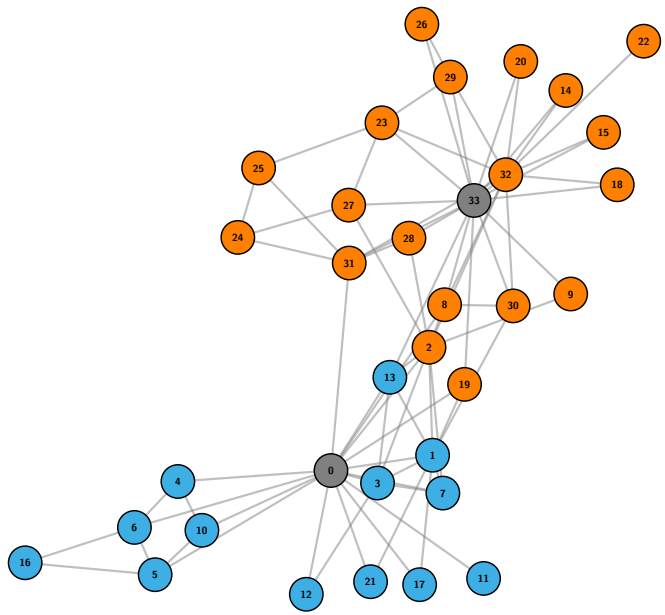
Example: Semi-Supervised Graph Learning 2/2

example

- **semi-supervised node classification** in Karate club network with $n = 34$ nodes and $m = 77$ links
- ground truth node class given for **two nodes in two communities**



training network with **two labeled nodes**



predicted node classes in test set using GCN with single message passing layer (accuracy 87.8%)

Notes:

- Adding a single labeled node in the other community of the graph is sufficient to train a model that correct classifies the majority of remaining nodes in the test. We obtain a model that achieves an accuracy of close to 88%.

Conclusion

- we incorporate information on **structure of data** in **neural network topology**
- sparse computation graph utilizes **spatial and/or topological correlations** in data
- relationship between **image convolutions** and **spectral graph theory**
- spectral graph convolution can be efficiently approximated with **neural message passing**
- basis for highly efficient **supervised, unsupervised and semi-supervised graph learning techniques**
- deep learning in graphs = **important machine learning innovation** of past decade

Notes:

- In summary, we have seen how we can apply deep neural networks to structured data. A key idea behind both convolutional neural networks for image data and graph neural networks for graph-structured data is to utilize the spatial/topological correlation between features to generate a sparse neural network topology. For CNNs, we can use multiple layers of learnable image convolutions that are able to extract hierarchies of shapes in images. The concept of image convolutions can be generalized to graphs, which is closely related to the modelling of dynamical processes in networks, and thus, spectral graph theory.
- Spectral graph convolutions can be efficiently approximated via a simple neural message passing algorithm that is a defining feature of graph neural networks. For the special case of neural message passing with self-loops, symmetric degree-normalization, and perceptron-based feature transformation, we obtain the popular Graph Convolutional Neural Network architecture.
- GNNs and GCNs are the basis for highly efficient supervised, unsupervised, and semi-supervised graph learning techniques. Their development can be considered one of the major innovations in machine learning in the past decade.

Questions

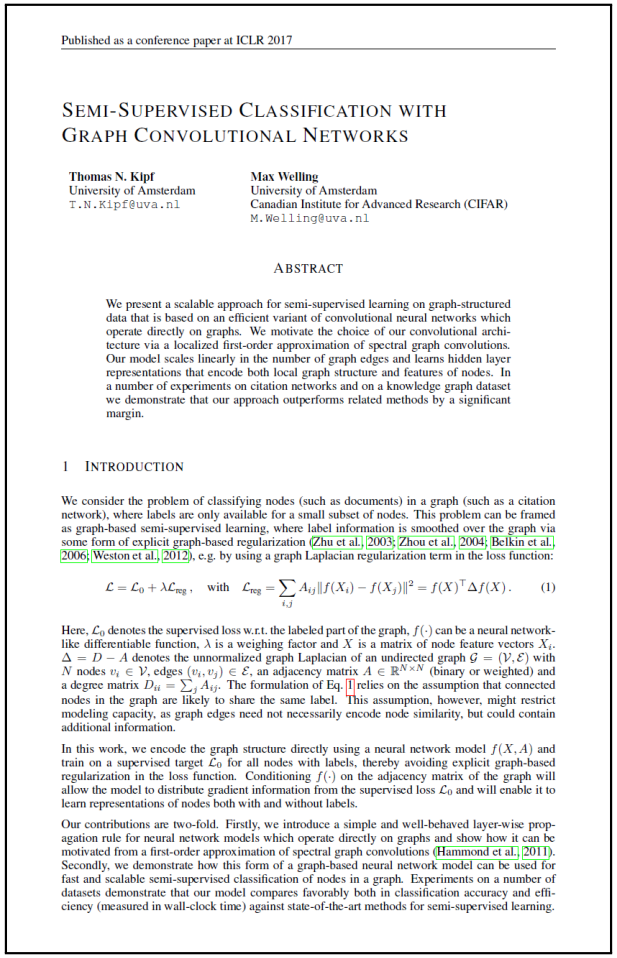
1. Consider a CNN with two convolutional layers with 3×3 kernel. Calculate the number of parameters with/without parameter sharing for a one megapixel image.
2. What is the difference between a *discrete convolution* $f * g$ and a *discrete cross-correlation* between two functions f and g?
3. What is a Graph Neural Network (GNN) and how is it different from a Graph Convolutional Network (GCN)?
4. How can we apply a GCN to a graph where nodes have no features? How do we define the initial node states $h_i^{(t)}$ for $t = 0$?
5. Investigate the so-called *loopy belief propagation* algorithm for general graphs and discuss its relationship with neural message passing.
6. Why and how can we use GCNs to address semi-supervised node classification.
7. For Laplacian matrix $\mathcal{L}$ show that $\mathbf{D}^{\frac{1}{2}}\mathcal{L}\mathbf{D}^{\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$.
8. Discuss similarities and differences between convolutional neural networks (CNNs) and graph convolutional networks (GCNs).
9. Consider a graph with 1000 nodes and 2000 edges. Calculate the number of parameters for a GCN with two message passing layers, one hidden layer with $d = 16$ dimensions, and a single binary output.

Notes:

References

reading list

- K Fukushima: **Neural network model for a mechanism of pattern recognition unaffected by shift in position — Neocognitron**, Trans. IECE, 1979
- FRK Chung: **Spectral Graph Theory**, American Mathematical Society, 1997
- H Lee et al.: **Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations**, ICML, 2009
- F Scarselli, M Gori, A Chung Tsoi, M Hagenbuchner: **The Graph Neural Network Model**, IEEE Trans. on Neural Networks, 2009
- J Bruna, W Zaremba, A Szlam, Y LeCun: **Spectral Networks and Deep Locally Connected Networks on Graphs**, arXiv 1312.6203, 2013
- J Gilmer et al.: **Neural message passing for quantum chemistry**, ICML, 2017
- T Kipf, M Welling: **Semi-Supervised Classification with Graph Convolutional Networks**, ICLR 2017
- MM Bronstein, J Bruna, Y LeCun, A Szlam, P Vandergheynst: **Geometric Deep Learning**, IEEE Signal Processing, 2017
- M Fey, JE Lenssen: **Fast Graph Representation Learning with PyTorch Geometric**, arXiv:1903.02428v3, 2019
- WL Hamilton: **Graph Representation Learning**, Morgan & Claypool Publishers, 2020 → Chapter 5
- MM Bronstein, J Bruna, T Cohen, P Velickovic: **Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges**, arXiv 2104.13478v@, 2021



Published as a conference paper at ICLR 2017

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

ABSTRACT

We present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. We motivate the choice of our convolutional architecture via a localized first-order approximation of spectral graph convolutions. Our model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on citation networks and on a knowledge graph dataset we demonstrate that our approach outperforms related methods by a significant margin.

1 INTRODUCTION

We consider the problem of classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes. This problem can be framed as graph-based semi-supervised learning, where label information is smoothed over the graph via some form of explicit graph-based regularization (Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2006; Weston et al., 2012), e.g. by using a graph Laplacian regularization term in the loss function:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \|f(X_i) - f(X_j)\|^2 = f(X)^\top \Delta f(X). \tag{1}$$

Here, $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph, $f(\cdot)$ can be a neural network-like differentiable function, λ is a weighing factor and X is a matrix of node feature vectors X_i. $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$, an adjacency matrix $A \in \mathbb{R}^{N \times N}$ (binary or weighted) and a degree matrix $D_{ii} = \sum_j A_{ij}$. The formulation of Eq. 1 relies on the assumption that connected nodes in the graph are likely to share the same label. This assumption, however, might restrict modeling capacity, as graph edges need not necessarily encode node similarity, but could contain additional information.

In this work, we encode the graph structure directly using a neural network model $f(X, A)$ and train on a supervised target $\mathcal{L}_0$ for all nodes with labels, thereby avoiding explicit graph-based regularization in the loss function. Conditioning $f(\cdot)$ on the adjacency matrix of the graph will allow the model to distribute gradient information from the supervised loss $\mathcal{L}_0$ and will enable it to learn representations of nodes both with and without labels.

Our contributions are two-fold. Firstly, we introduce a simple and well-behaved layer-wise propagation rule for neural network models which operate directly on graphs and show how it can be motivated from a first-order approximation of spectral graph convolutions (Hammond et al., 2011). Secondly, we demonstrate how this form of a graph-based neural network model can be used for fast and scalable semi-supervised classification of nodes in a graph. Experiments on a number of datasets demonstrate that our model compares favorably both in classification accuracy and efficiency (measured in wall-clock time) against state-of-the-art methods for semi-supervised learning.

Notes: