Universität Bonn

# MA-INF 4316: Graph Representation Learning

Examiner: Dr. Pascal Welke

Online Exam

The **4 problems** in this exam are graded out of **72 points**. Passing the exam requires to achieve at least **36 points**.

**Note:** You must hand in your solution within **12 hours**. Solutions received later than **22:00h CET, 2023-02-07** cannot be accepted. In other words: **Handing in solutions later than 22:00h CET, 2023-02-07 implies failing the exam**.

**Note:** When handing in your solutions, carefully follow the instructions on the next page.

**Note:** There are no restrictions on the use of *static* auxiliary tools or resources during the examination. For example, you are allowed to use all materials from the lecture and the exercises, and you may access the internet. However, **during the examination period, you must not communicate with anybody or anything except the examiner about the problems in this exam or their solutions**.
Communication includes, but is not limited to, asking your fellow students, asking questions on e.g. `StackOverflow`, `Quora`, `discord`, `Signal`, `WhatsApp`, `Telegram` groups, etc. It also includes chat bots or large language models that generate personalized replies based on your input.

**Note:** At the begin of the exam during **10.00h–12.00h CET, 2023-02-07**, you may ask questions, regarding technical details of the exam or exercises in a zoom meeting. The meeting ID is 966 5236 0278 and the passcode is 218838.

## HowTo: Submit Your Solutions

1. Use the Jupyter notebook `exam.ipynb` to hand in your solution. You have received it with this task description. You are expected to know how to work with Jupyter notebooks.

2. Fill the first cell of the notebook with your personal information.

3. Enter your solutions to the individual tasks separately in one or more cells below the respective headings. Please ensure that code, as well as results (plots, printed results, etc.) are shown in the notebook. Please also make sure that the notebook can be rerun to compute the same results (possibly up to randomization). We may rerun your code to check your solutions.

4. Once you are done preparing the notebook, **export** it to `html` format. (Typically, this option is provided via File→Export Notebook as or File→Download as)

5. Check if the export worked by inspecting the resulting `html` file.

6. Create a file `affirmation.txt` containing the following text and fill it with your personal information.

   > Last name:
   > First name:
   > Student ID number:
   > Degree program:
   > I hereby swear that I completed the examination MA-INF 4316: Graph Representation Learning completely on my own and without any impermissible external assistance or through the use of non-permitted aids. I am aware that cheating during the execution of an examination (as detailed in § 63 Para 5 of the Higher Education Act NRW) is a violation of the legal regulations for examinations and an administrative offense. The submission of false affirmation in lieu of an oath is a criminal offense.

7. Upload the `html` file, the `ipynb` notebook, and the `txt` affirmation file within the ecampus module, as seen in the exercises.

8. **Only in case uploading with ecampus does not work,** you may attach a zip file with your solution to an email. In this case, please rename it to `YOUR_MATRICULATION_NO.[zip|tar|tar.gz]`. **Using your University email account** send this email with **subject** "Exam MA-INF 4316: Graph Representation Learning" to welke@cs.uni-bonn.de Your email must be received **no later than 22:00h CET, 2023-02-07**.

# Task 1 Unsupervised Node Representations [21 Points]

## Task 1.1 Load the Data [2 Points]

Let $G$ be the `arxiv_exam.pickle` network graph provided with this exam. Load the graph as in the first exercise sheet and remove the set of test edges listed in the file `arxiv_testedges.txt` to obtain a training graph $G_{\text{train}}$.
Report the number of vertices and edges of $G_{train}$.

## Task 1.2 Path Enumeration [4 Points]

Implement a method that, given vertex $v \in V(G_{\text{train}})$ and length $l \in \mathbb{N}$, returns a single path from $v$ to $w$ for each $w$ that has (shortest-path) distance $l$ from $v$ in $G_{\text{train}}$.
Starting at the vertex with id $1337$, output the set of such paths for $l = 2$. Furthermore, compute and print the number of paths returned by the method using $l = 3$ and starting from the same vertex.

## Task 1.3 Implement "DeepPath" [2 Points]

Implement a DeepWalk variant that, instead of random walks, uses the set of paths of a certain length, as outputted by the above method.

## Task 1.4 Choose an appropriate $l$ [3 Points]

Select 20 random vertices from $G_{\text{train}}$. For each of these vertices, print its id and the number of paths of length $l \in \{1, 2, 3, 4, 5\}$ your method from Task 1.2 finds.

Which $l$ would you choose for your DeepPath variant, and why? Give a short answer. What benefits do random walks of length $l$ have compared to the set of your paths of length $l$?

## Task 1.5 Apply DeepPath [2 Points]

Run DeepPath on $G_{\text{train}}$ for your choice of $l$, an appropriate window size $w \leq l$ and embedding dimension. Output the DeepPath embedding of the vertex with id $1337$.

## Task 1.6 Apply DeepWalk [2 Points]

Implement and run *DeepWalk* with identical parameters, but choose $30$ sampled walks per vertex.

## Task 1.7 Compare DeepPath and DeepWalk [6 Points]

Train a logistic regression classifier using the cosine similarity between the endpoints of edges in $G_{\text{train}}$ for both your DeepPath and DeepWalk embeddings. Remember to extend the train and test sets by equally many negative instances.
DeepPath fails to compute representations for some vertices. Argue why this is the case and decide on a way to deal with it when training and evaluating your classifier.
Report the ROC-AUC score on the test edges. Which one works better, DeepPath or DeepWalk?

# Task 2 Weisfeiler Lehman Set Kernels [10 Points]

**Note:** This is a theory task. You do not need to implement anything.
**Note:** Jupyter notebooks support latex pidgin math mode. You can write formulas in markdown cells by enclosing latex formulas in dollars or double dollars (for display math) like this: `$k \in \{1,2\}$`
**Note:** You may also write solutions by hand and include a scan in the notebook or the upload to ecampus.

## Task 2.1 Set Kernel [5 Points]

Consider the following variant of the Weisfeiler Lehman Subtree kernel. **Show that** the similarity function

$$\overline{\kappa_k^{WL}}(G_1, G_2) = \sum_{c \in \mathcal{X}_k} \delta_c(G_1, G_2)$$

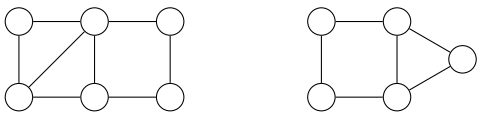
**is a kernel** on any graph class $\mathcal{G}$, where $\mathcal{X}_k = \left\{ r_k^{WL}(v) \mid v \in V(G), G \in \mathcal{G} \right\}$ is the set of all Weisfeiler Lehman labels in $\mathcal{G}$ in iteration $k$ (as in the lecture) and

$$\delta_c(G_1, G_2) := \begin{cases} 1 & \text{if } c \in R_k^{WL}(G_1) \wedge c \in R_k^{WL}(G_2) \\ 0 & \text{otherwise} \end{cases}$$

## Task 2.2 Corresponding Feature Embedding [2 Points]

Can you give the feature embedding function $\Phi$ for $\overline{\kappa_k^{WL}}$?

## Task 2.3 Compute the Kernel [3 Points]



$G_1$ $G_2$

Draw the graphs $G_1$ and $G_2$ after one and two Weisfeiler-Lehman relabeling iterations. Explicitly indicate the vertex labels in the graphs by using, e.g., colors or integers. Subsequently, provide the kernel values $\overline{\kappa_1^{WL}}(G_1, G_2)$ and $\overline{\kappa_2^{WL}}(G_1, G_2)$.

# Task 3 Weisfeiler Lehman Kernel Variants [21 Points]

**Note:** Recall that you can load a (standard) dataset e.g. using

```
from grakel.datasets import fetch_dataset

DATASET = fetch_dataset("DATASET", verbose=False, as_graphs=True)
graphs = DATASET.data
labels = DATASET.target
```

or a similar function using pytorch_geometric.

## Task 3.1 Using the Weisfeiler Lehman Kernel [4 Points]

Load the `AIDS` dataset and compute the Gram matrix of the (cumulative) Weisfeiler Lehman Subtree Kernel $\kappa_{\leq 4}^{WL}$.
`AIDS` contains 2000 graphs. Let $G_i$ for $i \in \{0, \ldots, 1999\}$ be the graph with index $i$ in the `AIDS` dataset. Print the kernel values for the following pairs of graphs:

1. $G_{45}$ and $G_{84}$
2. $G_{23}$ and $G_{42}$
3. $G_{187}$ and $G_{187}$

## Task 3.2 Crossvalidate an SVM Classifier [3 Points]

Evaluate the Weisfeiler Lehman subtree kernel on `AIDS` by running a 10-fold crossvalidation for an SVM classifier that uses $\kappa_{\leq 4}^{WL}$. Fix the SVMs $C$ parameter to $0.1$.
Print the mean test accuracy and its standard deviation over the 10 folds.

## Task 3.3 An Altered Version of the Weisfeiler Lehman Subtree Kernel [6 Points]

Recall the Weisfeiler Lehman Subtree kernel $\kappa_k^{WL} := \langle \Phi_k^{WL}(G_1), \Phi_k^{WL}(G_2) \rangle$.
Now consider the following weighted cumulative kernel variant:

$$\tilde{\kappa}_{\leq k}^{WL}(G_1, G_2) = \sum_{i=0}^{k} \langle \tilde{\Phi}_i^{WL}(G_1), \tilde{\Phi}_i^{WL}(G_2) \rangle$$

with

$$\tilde{\Phi}_k^{WL}(G) := \frac{1}{k+1} \Phi_k^{WL}(G) \, .$$

Implement this kernel, compute the corresponding gram matrix and print the kernel values for the same pairs of graphs as in task Task 3.1.

### Task 3.4 Crossvalidate an SVM Classifier [1 Point]

Evaluate your newly implemented kernel on `AIDS` by running a 10-fold crossvalidation for an SVM classifier that uses $\tilde{\kappa}^{WL}_{\leq k}$. Fix the SVMs $C$ parameter to $0.1$.
Print the mean test accuracy and its standard deviation over the 10 folds.

### Task 3.5 Weisfeiler Lehman Embeddings and Isomorphism [5 Points]

We now consider two graphs $G_1, G_2$ equivalent iff their embeddings $\Phi^{WL}_4(G_1)$ and $\Phi^{WL}_4(G_2)$ are equal.

1. How many different equivalence classes do we have on `AIDS`?
2. How many graphs does the largest equivalence class on `AIDS` contain?
3. Will these numbers be different if we consider $\tilde{\Phi}^{WL}_4$ instead of $\Phi^{WL}_4$ in the definition of the equivalence relation? Argue why or why not.

### Task 3.6 Plotting the AIDS Dataset [2 Points]

Given the gram matrices for $\kappa^{WL}_{\leq 4}$ and $\tilde{\kappa}^{WL}_{\leq 4}$, we can plot the `AIDS` dataset (twice) using a kernelized low-dimensional embedding technique, such as kernel PCA. Use the following snippet to plot the dataset: `AIDS` dataset:

```
from sklearn.decomposition import KernelPCA
import matplotlib.pyplot as plt

kpca = KernelPCA(n_components=2, kernel='precomputed')
emb = kpca.fit_transform(K)
plt.scatter(emb[:, 0], emb[:, 1], c=DATASET.target)
plt.show()
```

What do you observe? Give a short answer to the question.

# Task 4 (Message Passing) Graph Neural Networks [20 Points]

For this task, you will be working with the DHFR dataset from the TU-Dortmund collection. You can find some possibly useful code snippets in your jupyter notebook. However, you are not required to use them if you find another setup more to your liking. It is, however, beneficial to use the `pytorch_geometric` package.

## Task 4.1 Layers [4 Points]

Create four neural networks with different values for the node representation dimension $d$ according to the following specifications:

- Fix the number of GCN (Graph Convolution Network) layers to $3$.
- Vary the dimension $d$ of the vertex representations for all GCN layers for $d \in \{2, 8, 32, 128\}$.
- Add a ReLU activation function after each layer.
- For each graph $G$, aggregate the vertex representations $r_3(v)$ for all $v \in V(G)$ using a mean aggregator (`global_mean_pool`).
- Add two fully connected layers, with the first one followed by a ReLU activation function. The first fully connected layer should halve the dimension of your graph representations; the second fully connected layer should reduce this dimension further to the number of classes in DHFR.
- Finally, apply `log_softmax` to obtain predictions for classes.

## Task 4.2 Evaluation [4 Points]

Randomly, split the set of graphs into $80\%/20\%$ train/test sets. Evaluate your four neural network architectures after 100 epochs with an Adam optimizer with learning rate of $0.005$ which you halve after 50 and 75 epochs. You may use the code provided in the Jupyter notebook.
What do you observe? Briefly describe the results. Which vertex representation dimension would you choose? Give a short argument why.

## Task 4.3 Re-Evaluation [3 Points]

Repeat the training from the previous task six times for six freshly initialized network architectures with the representation dimension $d$ that you have chosen in Task 4.2. Print the mean and standard deviation of the train and test errors. What do you observe?

### Task 4.4 GNNs without GNN layers [4 Points]

Now, create networks *without* any GNN layers. For each $d \in \{2, 8, 32, 128\}$, create a network according to the following specifications:

- Apply a `global_add_pool` aggregation on the initial vertex representations.
- Add two fully connected layers, with the first one followed by ReLU activation function. The first fully connected layer should output a graph representation of dimension $d$; the second fully connected layer should reduce this dimension further to the number of classes in DHFR.
- Finally, apply `log_softmax` to obtain predictions for classes.

### Task 4.5 Interpretation [2 Points]

Recall (or note) that the initial vertex representations of DHFR nodes in pytorch are one-hot encoded discrete labels. Argue what the "GNN" in Task 4.4 does. In particular, explain the role of the `global_add_pool` function.

### Task 4.6 Evaluation [3 Points]

Evaluate your neural network architectures from Task 4.4 in the same way as in Task 4.2.
What do you observe? Briefly describe the results. How does this compare to the results of your best GCN architecture? What does this imply?