ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет информатики, математики и компьютерных наук

**Программа подготовки магистров по направлению 01.04.02
Прикладная математика и информатика**

Магистерская программа «Интеллектуальный анализ данных»

Артамонов Денис Сергеевич

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Neural Networks for Graph Analysis

Рецензент:
Д.ф.-м.н., Профессор, кафедра прикладной математики и информатики НИУ ВШЭ — Нижний Новгород

Калягин Валерий Александрович

Научный руководитель:
Старший преподаватель, кафедра прикладной математики и информатики НИУ ВШЭ — Нижний Новгород

Слащинин Сергей Владимирович

Нижний Новгород, 2022

Contents

1 Introduction

Neural networks algorigthms have already become a state-of-the-art solution for a wide range of tasks in many different fileds, such as computer vision, natural language processings, speech recognition and many others. Most of these tasks have in common the possibility of representing processed data in Euclidean domain. However, a large number of learning tasks have to deal with data expressed with graph. Graphs allow to represent information about relationship between different objects, but requires another approach to apply neural networks algorithms, as grah data is non-Euclidean.

To solve this issue researches developed a sepecial kind of neural networks, which can operate graph data, called Graph Neural Network, or GNN [15]. Since the first introduction of GNN, a lot of new upgraded, capable and more powerful models have been developed and used for many practical applications in such areas as physics simulations [1], traffic prediction [18], biology[13] [36], fake news detection [5], social analysis [31], computer vision [42] [27] recommendation systems [41] and many others.

GNN models can be used to solve three general types of prediction tasks on graphs: node level, edge level and graph level tasks. Node level tasks aims to predict some propeprty of node, for example to predict node label. Edge level tasks include tasks related to edge properties prediction, such as labels of edges. Edge level tasks allow to discover

connection types between the entites in graph. The remaining group of tasks is graph level tasks. Such tasks are related to predicting of graph level properties, e.g. class of the given graph, which is known as graph classification problem. This is analogous to image classification task in computer vision field or sentiment analysis tasks in natural language processing.

The graph theory-based approaches are commonly used for extracting knowledge for biomedical research. Graphs represent structural networks, which can be generated, from anatomical regions connections (as in the case of brain networks [29]), molecular structures data (e.g., chemical compounds [26], proteins structure [24] networks), or physical interactions between molecules (e.g., protein-protein [32], lncRNA protein interaction networks [22]).

In the era of big data, the size and complexity of the biomedical networks are continuously increasing. So, new chellenges in biomedical networks analysis requires new approaches. And one of them is using neural networks for solving prediction tasks on graphs, such as graph classification problem, which arises in many biomedical applications.

The main goal of the current work is to compare several different approaches to solve graph classification problem on biomedical data. This approaches will include the usage of several graph neural networks architectures : GCN [23], GAT[39] and GCNII [7]. Besides, several approaches for generation of node feature matrix will be applied.

For node feature matrix generation will be used encoded information about labels and degrees of node, as well as node distance distribution matrix(NDD), proposed in [28] and used to obtain graph embeddings.

The paper is organized as follows. Section2 contains detailed description of used biomedical dataset. Short review of GNN algorigthms and description of used algorigthms are presented in Section 3. Section 4 contains implementation details and experiments results. Finally, Section 6 reports conclusion.

2 Datasets

This sections describes datasets used in this paper. Datasets can be devided into two groups. The first one includes datasets from TUDatasets [30], which is a large collection of graph datasets, introduced in [30] and very common and widely used for benchmarking of GNN models. In current paper there are used bioinformatics datsets containing networks representin micro- and macromolecules: `MUTAG`, `PROTEINS`, `ENZYMES`, `DD`. Properties of datasets are shown in table 1

`MUTAG` dataset [9] contains 188 graphs representing chemical nitroaromatic compounds. Vertices af graphs respresent atoms, labeled by one-hot encoded atom type. Edges represent bonds between the cor-

	MUTAG	PROTEINS	ENZYMES	DD	Brain fMRI	Kidney RNA Seq
# Graphs	188	1113	600	1778	124	299
# Classes	2	2	6	2	2	3
# Samples per class	63/125	663/450	100/100/100/100/100	691/487	70/54	159/90/50
# Vertices	≈ 18	≈ 39	≈ 33	≈ 284	263	1034
Node features	7	3	3	89	0	0
Max node degree	4	25	9	19	238	105
Max diameter	15	64	37	83	2	7
Has edge weights	no	no	no	no	yes	yes
Has edge attrubutes	yes	no	no	no	no	no

Table 1: Datasets properties

responding atoms correponds to their mutagenicity on Salmonella typhimurium.

Grpoup of `PROTEINS`, `ENZYMES`, `DD` datasets contains networks representng macromolecules of proteins. Graphs from `PROTEINS` and `ENZYMES` datasets are based on graph model for proteins introduced in [4]. In that graph model nodes represent secondary structure elements and are annotated by their types. Nodes are connected by the edge if they are neighbors along the amino acid sequence or one of three nearest neighbors in space. Using this approach `ENZYMES` was derived from BRENDA (BRaunschweig ENzyme DAtabase)[35] dataset, which contains a large collection of enzyme and metabolic information, based on primary literature. Each graph in `ENZYMES` has one of six classes, which reflects the catalyzed chemical reaction (6 classes). It contains 600 graphs. Dataset `PROTEINS` containing 1113 graphs, was derived from abother dataset, presented in [11]. The task for `PROTEINS` dataset is binary classification: whether a protein is an enzyme. Dataset [10]

contains 1178 protein structures having nodes representing individual amino acids and edges their spatial proximity.

Another group consists of two real medical datasets used for benchmarking `Netpro2vec` model in [28]: `Brain fMRI` and `Kidney RNASeq` datasets.

`Brain fMRI` dataset includes 124 real graphs derived in [33] from the Center for Biomedical Research Excellence (COBRE) dataset, which contains functional magnetic resonance imaging (fMRI) time-series data. Brain fMRI dataset includes 124 graphs: 54 graphs from Schizophrenia subjects (Sch) and 70 graphs from healthy controls (Ctrl). Each graph contains 263 nodes corresponding to different brain regions. The edges have weights representing Fisher-transformed correlation between the fMRI time-series of the nodes after ranking.

`Kidney RNASeq` dataset consists of metabolic networks constructed by mapping gene expression data on the biochemical reactions extracted from the kidney tissue metabolic model. Gene expression data is obtained from TCGA-KIRC and TCGA-KIRP projects of the Genomic Data Commons [1] portal. Datasets was prepared by authors of [28] Kidney RNASeq dataset contains 299 networks devided into three classes. Two of them corresponds to the two most common types(according to National Cancer Institute [2]) of kidney cancer: Clear

[1] `https://portal.gdc.cancer.gov`

[2] `https://www.cancer.gov/pediatric-adult-rare-tumor/rare-tumors/rare-kidney-tumors/clear-cell-renal-cell-carcinoma`

Cell Renal Cell Carcinoma (159 samples) and Papillary Renal Cell Carcinoma (90 samples). The third classs Solid Tissue Normal represents the absence of pathology and contains 50 samples.

Both datasets were downloaded from cbs-group repository[3] [4], authors of Netpro2vec model [28].

3 GNN models overview

In this section, there will be provided a short literature review from the first attempts to apply neural nets to graph data to modern approaches using using convolutions and attention mechanism.

Early attempts to adapt neural network architectures to operate with graph data were proposed in works [14] and [37]. These models used recursive neural networks (RNN) architecture. However these methods could be applied only to directed acyclic graphs, which led to a serious restriction in their usage. Models known as Graph Neural Networks (GNNs) were propesed in [17], [34], [15]. These models generalized recursive neural networks to deal with more general classes of graphs. Presented in [15] GNN architecture is commonly refered as vanilla GNN.

Over the last decade, there have been proposed a huge number of new architectures and frameworks, which outperform vanilla GNNs.

[3] https://github.com/cds-group/GraphDatasets

[4] https://github.com/cds-group/Netpro2vec

Below there will be described several of them.

3.1 GCN

One of the most common modern approaches to construct of graph neural network is neural networks based on convolutions.

Convolution based neural networks (CNNs) have been applied to a wide range of computer vision tasks and have proved their effectiveness to operate with grid-like structure data, such as images. This approach has been proposed in [3].

Any grid can be seen as a graph of grid like structure in the Euclidean space. So the idea to extend and generalize convolutions to graph data seemed to be very natural and aroused great interest among researchers.

Convolution used in CNNs has predefined reception field. Which means, that convolution kernel is applied to the neighbourhood of predefined size. In case of grid-like structure data, such as images, the structure of the image doesn't change: each pixel has the same number of neighbors. However, such property is not guaranteed for graph data. Each node of the graph may have different number of neighbors, so neighbourhood structure differs from node to node. So, to apply convolutions to graph data, it should be prepared to operate with neighborhood with different structure.

The first solution of this problem was proposed in Spectral Network

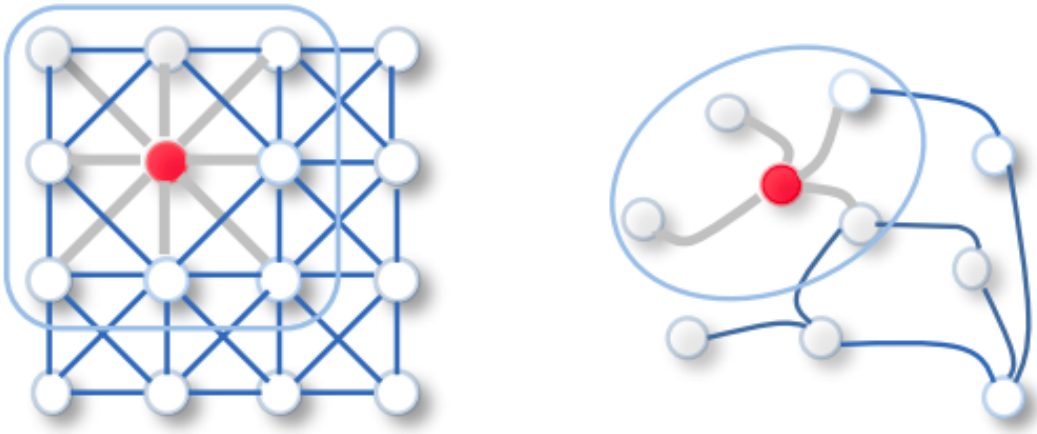

Figure 1: 2D Convolution vs. Graph Convolution [40]

model [6].This work proposes to define convolution operation on graphs using eigendecomposition of the graph Laplacian.

For a undirected graph $G = (V, E)$ with $|V| = n$, graph Laplacian matrix $L \in \mathbb{R}^{n \times n}$ is defined as

$$L = D - A \tag{1}$$

where A is adjacency matrix of the graph, and D is degree matrix.

Adjacency matrix $A \in \mathbb{R}^{n \times n}$ contains information about graph structure and is defined as:

$$A_{ij} = \begin{cases} 1 \text{ if } \{v_i, v_j\} \in E, i \neq j, \\ 0 \text{ otherwise,} \end{cases} \tag{2}$$

Degree matrix $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing degrees

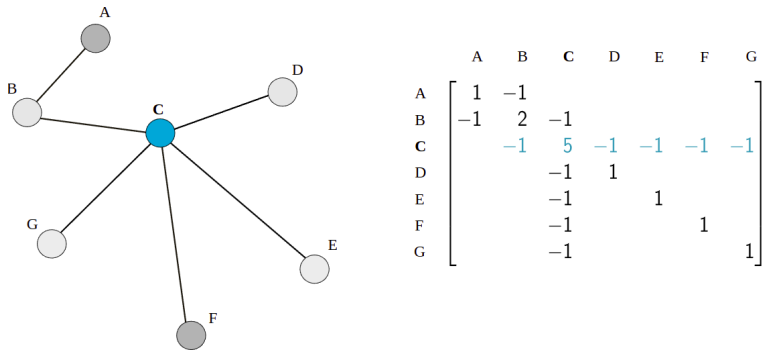

Figure 2: The example of Laplacian for a graph from [8]

of vertices and defined as:

$$D_{ii} = deg(v_i) \tag{3}$$

Symmetic normalized Laplacian is defined as

$$L = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{4}$$

The convolution operation as for a feature node vector $x \in \mathbb{R}^N$ with a filter $g_\theta = diag(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ is defined as:

$$Conv(g_\theta, x) = U g_\theta(\Lambda) U^T x \tag{5}$$

where U is the matrix of eigenvectors of the normalized Laplacian, Λ is a diagonal matrix of eigenvalues from the spectral decomposition of normalized Laplacian: $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$.

Such approach to the definition of convolution on graph data is known as localized spectral filter. And used convolutions is known

as spectral convolutions. However, this convolution operation is very computational expensive as multiplication with matrix U has complexity $O(n^2)$.

Authors of [19] susuggested to use approximation to calculate $Conv(g_\theta, x)$ using Chebyshev polinomials $T_k(x)$ up to K^{th}, which is defined as

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \tag{6}$$

where $T_0(x) = 1$ an $T_1(x) = x$

So convolution of a signal x with filter g_θ is suggested to be:

$$Conv(g_\theta, x) \approx \sum_{k=0}^{K} \theta_k T_k(\hat{L})x \tag{7}$$

where $\hat{L} = \frac{2}{\lambda_{max}}L - I_N$ and λ_max stands for the largest eigenvalue of L. The complexity of evaluating convolution in proposed way is $O(|E|)$, i.e. is linear in the number of edges.

In [21] there was proposed ChebNet - a convolutional neural network on grpahs based on K-localized convolution.

Authors [23] developed the idea of usage spectral convolution and presented GCN model. The key idea is to build convolutional neural network by stacking multiple simplified convolution layers with limited parameter $K = 1$. Single layer of such network can be expressed as:

$$Conv(g_\theta, x) \approx \theta_0 T_0(\hat{L})x + \theta_1 T_1(\hat{L})x = \theta_0 x + \theta_1(\frac{2}{\lambda_{max}}L - I_N)x \tag{8}$$

Then authors of [23] suggested to approximate $\lambda_{max} \approx 2$, so the expresion turns into the following one:

$$Conv(g_\theta, x) \approx \theta_0 x + \theta_1 (L - I_N)x = \theta_0 x - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \qquad (9)$$

However, instead of two parameters θ_0 and θ_1 authors suggested to use the single one - $\theta = \theta_0 = -\theta_1$. Thus it allows to address overfitting problem and to simplify computations per layer. So, the follwoing expression is

$$Conv(g_\theta, x) \approx \theta (I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x \qquad (10)$$

Such expression of the convolution operation has a flow. The multiplier $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ has eigenvalues in range $[0, 2]$, so that it leads to numerical instabilities when its repeated applying as layers in neural network. To alleviate this problem, authors propose to replace numerical unstable multiplier $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ with $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, where $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

The final rule for linear convolutional layer for GCN can be generalized as following. Let signal $X \in \mathbb{R}^{N \times C}$ is a matrix of C-dimensional feature vectors for each of N nodes, and $\Theta \in \mathbb{R}^{C \times F}$ is a matrix of filter parameters. So convoled signal $X' \in \mathbb{R}^{N \times F}$ is expessed as

$$X' = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \qquad (11)$$

So, GCN model allows to calculate node representations as vectors (embeddings) using layer-wise convolutional neural network.

We can rewrite the equation 11 in term of neural network layer. So, obtained the graph convolutional layer is the following:

$$H^{(l+1)} = ReLU(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{12}$$

where $H^{(l)} \in \mathbb{R}^{n \times d}$ is a hidden state matrix on layer l, d is a dimension of the convolution and $W^{(l)} \in \mathbb{R}^{d\ timed}$ is a matrix of trainable parmeters of the layer l.

ReLU function is an activation function, used to add non-linearity to neural network layer. It is defined as

$$ReLU(x) = max(0, x) \tag{13}$$

3.2 GAT

Along with spectral convolution approach, there are other ways to prepare node embeddings. One of such approaches, is attention-based architecture model called Graph Attention Networks (GAT) proposed in [39]. The main idea of this model is to apply attention mechanism to graph data. Based on attention models have become the most common solution for many sequence-based tasks [2], [16]. The main feature of attention mechanism is that it allows to focus on the most relevant parts of the input to make a desicion.

Authors of [39] proposes a graph attentional layer, which is a building block for the arbitrary graph atteniton network. The input of the layer is a set of node feature $h = \{h_1, \ldots, h_N\}$, where $h_i \in \mathbb{R}^F$, N - number of nodes, F - the number of node features. As result of layer processing is a new set of node features with other dimension: $h' = \{h'_1 \ldots h'_N\}$. Each layer computes attentions coefficients α_{ij} by applying attention mechanism function $\alpha : R^{F'} \times R^{F'} \to R$. to input node features multiplied to the shared weight matrix $W \in \mathbb{R}^{F'} \times R^F$:

$$e_{ij} = \alpha(Wh_i, Wh_j) \tag{14}$$

This coefficient indicates the importance of of node j's features to node i. The next step is performing masked attention to be able to compute only coefficients for a node is some defined node's neighborhood N and normalizing coefficients by neighborhood:

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in N_i} exp(e_{ik})} \tag{15}$$

The last element is an attention function α itself. Authors of [39] proposed to use single-layer feedforward neural network parameterized by $a \in \mathbb{R}^{2F'}$ and apply LeakyReLU nonlinearity.

As a result, the coefficients computed by the attention mechanism can be expressed as

$$\alpha_{ij} = \frac{exp(LeackyReLU(a^T[Wh_i||Wh_j]))}{\sum_{k \in N_i} LeackyReLU(a^T[Wh_i||Wh_k])} \tag{16}$$

where || denotes concatenation operation. LeackyReLU - a activation function similar to ReLU defined in 13, but apable of taking negative values:

$$LeackyReLU(x) = \begin{cases} x, \text{ if } \geq 0, \\ ax, \text{ otherwise} \end{cases} \tag{17}$$

When coefficients are obtained - we can apply them to corresponding nodes:

$$h'_i = \sigma \left(\sum_{j \in N_i} \alpha_{ij} W h_j \right) \tag{18}$$

To stabilize training procedure, authors extends self-attention mechanism to multi-head attention, proposed in [38]. The idea of multi-head attention is to train several representations for each nodes at the same layer. And the final embedding is just a concatenation of the all produced ones. For K independant heads there is a following final representation of the node:

$$h'_i = ||_{k=1}^{K} \sigma \left(\sum_{j \in N_i} \alpha_{ij}^k W^k h_j \right) \tag{19}$$

On the prediction (final) layer, instead of concatenation, authors suggest to employ averaging over all attention heads:

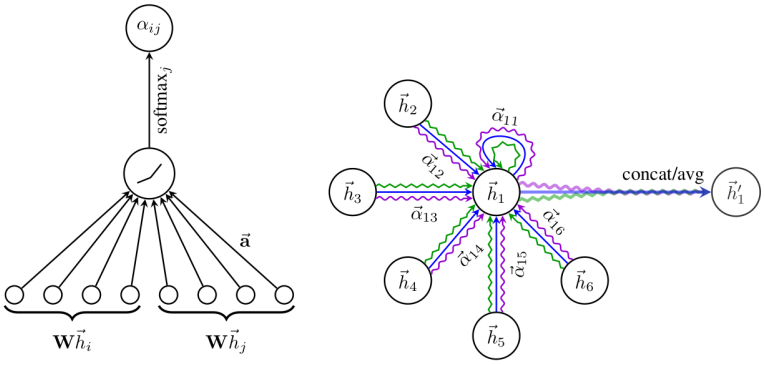

Figure 3: GAT network illustration from [39]. Left: calculating of attention for each node. Right: feature aggregation from 3 head attention.

$$h'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N_i}\alpha_{ij}^k W^k h_j\right) \tag{20}$$

This last step is called aggregation and show on image 3.

So, GAT model allows to assign different importances to nodes of a same neighborhood which distinguishes it favorably from GCNs model. Besides GAT model has another feature: the computation of the node-neighbor pairs is parallelizable thus the operation of attention calculating is very efficient.

3.3 GCNII

Both GCN[23] and GAT[39] models are shallow. Attempts to stacking more layers and to add non-linearity tends to degrade the per-

formance of these models [7] because of over-smoothing phenomena, described in [25]. Over-smoothing suggests that node representations becomes indistinguishable as thaey are inclined to converge to a certain value. This fact assigns restrictions on the number of layers in the network and as a result limitations network's ability to extract information from high-order neighbors.

Authors of [7] extend GCN model with two modifications to prevent over-smoothing issue. Proposed architecture called Graph Convolutional Network via Initial residual and Identity mapping (GCNII).

The l-th layer of GCNII is defined as the following:

$$H^{l+1} = ReLU\left(((1-\alpha_l)\tilde{P}H^l + \alpha_l H^0)((1-\beta_l)I_n + \beta_l W^l)\right) \qquad (21)$$

where α_l and β_l are hyperparameters. and $\tilde{P} = \tilde{D}^{-\frac{1}{2}}A\tilde{D}^{-\frac{1}{2}}$ is the graph convolution matrix with the renormalization trick used in equation 11 inside the GCN's layers.

Comparing with GCN layer defined in 12 GCNII has two modifications. The first one is to add initial residual connection to the first layer H^0. It simulates residual connections between different layers of the network originally proposed in ResNet model [20] in the filed of computer vision. The initial residual connection in GCNII ensures that that the final representation of each node retains at least a fraction of α from the input layer even if the network contains many layers stacked. Authors of [7] suggest to set hyperparameter $\alpha_l = 0.1$ or 0.2.

The second modification is to add identity mapping I_n to each layer's matrix of weights. Adding of identity mapping provides regularization on weight matrix $W^{(l)}$ to avoid over-fitting. The value of β depends of the number of the layer. Authors suggest to set $\beta_l = \log(\frac{\lambda}{l} + 1) \approx \frac{\lambda}{l}$, where λ is hyperparameter.

3.4 Pooling

Originally, GCN, GAT and GCNII models were developed for solving node classification problem. These methods allow to get node embedding and learn neural network to match it to the specified label.

However, all these methods can be easily applied for solving graph classification problem using pooling operation [8]. Pooling operation allows to aggregate information from final node embeddings and the apply predictor for final graph classification. In this work as aggregation function simple mean pooling function will be used:

$$r_i = \frac{1}{N_i} \sum_{n=1}^{N_i} x_n \tag{22}$$

where embedding vector r_i of graph i is calculated as a maen of node embeddings x_n.

As a final classifier, simple 2-layer perceptron will be used:

$$c_i = W_2(Relu(W_1 r_i)) \tag{23}$$

In the current paper the folowing models will be trained and evaluated: GCN[23], GAT[39] and GCNII[7].

4 Experiments

This section provides information about computational experiments, their configuration, results and details of implementation.

Code for experiments has been developed on `Python 3.8` using `Pytroch Geometric` framework [5] [12]. This frameworks contains a lot of implemented graph neural network models, provides a convenient tools to operate data and suggests an algorithm to train and validate models.

Below there are description of the data preparation process, followed by model implementation details and description of the organization of the training process.

4.1 Data preparing process

Framework `Pytorch` proposes a unified way [6] to work with data. It consists of two stages: preapare dataset object with defined methods to access data and configure DataLoader to form batch of predefined size. Batch consists of vectors of features from datasets and labels

[5] `https://pytorch-geometric.readthedocs.io/en/latest/index.html`

[6] `https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#datasets-dataloaders`

corresponding them in the case of solving the classification problem. DataLoader allows to itearete data within one train epoch.

The same approach is used in `Pytroch Geometric` geometric framework.

The difference is in the way of forming batches. Implementation of `DataLoader` class in `torch_geometric.data` allows to make batch from several graph data by stacking adjacency matrices in a diagonal fashion. So that batch is just a large graph with miltiple isolated subgraphs.Example is shown on the image 4 from Pytroch Geometric tutorial [7].

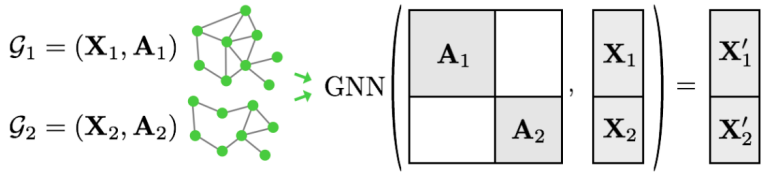

Figure 4: Pytorch Geometric approach to batches

In the current two groups of datasets are used. The first one contains datasets from TUdataset[30] collection: `MUTAG`, `PROTEINS`, `DD` and `ENZYMES`. `Pytroch Geometric` provides convenient access to all of them with `torch_geometric.datasets.TUDataset` class. [8]

The second group contains `Brain fMRI` dataset and `Kidney RNASeq` dataset used in [28]. Graphs from these datasets are available in

[7] `https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html`

[8] `https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html#torch_geometric.datasets.TUDataset`

`graphml` format. So to operate them in the way defined in framework, there has been developed a custom `Dataset` object, which extends `InMemoryDataset` [9] class and allows to process them the same way as datasets from the first group.

4.2 Models configuration

Three graph neural networks are trained and inferenced in the current paper. Below there are information about these models' configuration used in exeriments.

GCN model contains 3 convolution layers with hidden layer dimension equal to 128, one hidden layer with dimension 64 and the final linear layer to calculate logits. The values of hyperparameters are the following: $\alpha = 0.5$ and $\lambda = 0.1$.

GAT model contains two GAT convolution with 8 and 1 heads respectively and hidden demension equal to 64 and linear layer with dimension 32, followed by final linear layer to calculate logits.

GCNII model contains of 16 convolutions with hidden size 64, linear layer with dimension 32, followed by final linear layer to calculate logits.

Hyperparameters for models where searched using the following strategy. Dataset is split into validation(20%) and training set(80%)

[9] `https://pytorch-geometric.readthedocs.io/en/latest/modules/data.html#torch_geometric.data.InMemoryDataset`

Conv dimension	1 Conv	3 Conv
128	0.9286	0.962
64	0.9103	0.9107
32	0.8913	**0.963**
16	0.8454	0.8454

Table 2: Results of hyperparameters tuning for GCN network on MUTAG datset

and train model with defined set of hyperparameters. The example of hyperparameters tuning for GCN model on `MUTAG` dataset can be found in table 2.

When the best set is found, the process of testing starts: dataset is split into train set(80%) and test set(20%). Then train set is split into validation set(30%) and train set (70%). After each epoch, model is tested on validation set. When training process is finished, the best model is selected with respect to the metrics measured on validation set. Then best model is evaluated on test set.

4.3 Models training pipeline

To train and evaluate model there has been prepared the following pipeline.

Firstly, dataset prepared and uploaded to `Google Drive`. Then code for training is wrapped into command line util and pushed to

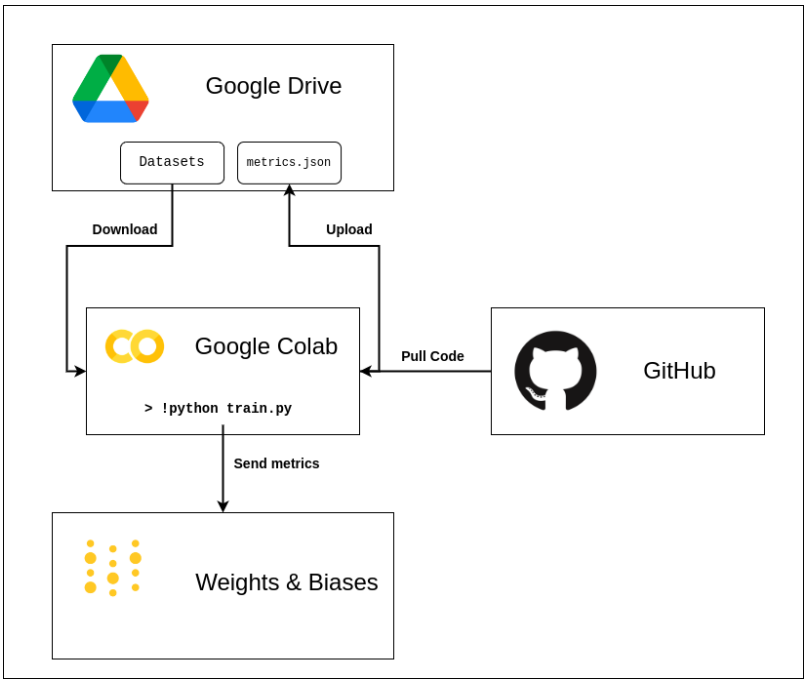

Figure 5: Training pipeline schema

`GitHub` repository. Training and evaluation process run in `Google Colab`.Training metrics are uploaded to `Weight And Biasis` - application for displaying and storing graphs in runtime. Testing metrics are uploaded to `Google Drive`. The whole pipeline is shown on figure 5. Example of graph on `Weight And Biasis` are shown on figure 6

4.4 Node features transformation

All neural networks models, which will be used in experiments requires node feature matrix. This matrix acts as a signal in term of signal processing with convolutions and the base embeddings of nodes, which transformation should be applied to.

Figure 6: Graphs example from Weight And Biasis. Graps are drawing in training runtime

In some cases, encoded node labels may be used as node features. However, some networks do not have neither node labels, nor features. In that case, node features can be constructed additionaly as a preprocessing step.

In the current paper two approches will be used to construct node features.

The first one is to use one-hot reprosentation of node degrees. So the node features matrix is matrix $X \in \{0,1\}^{N\times d}$ having property $\sum_{j=1}^{d} X_{ij} = 1$ for any $i \in 1 \dots N$, where N is the number of nodes in graph G and d is max degree over all nodes in graph.

The second approach is to use Node Distance Distribution matrix(NDD) presented in [28]. For a graph $G = (V, E), |V| = N$ matrix $NDD \in \mathbb{R}^{N\times D}$ where D is diameter of the graph is calculated as follows:

$$N_{ih} = \frac{|v_j \in V : dist(v_i, v_j) \in [h, h + \delta_h]|}{N - 1} \tag{24}$$

where $dist(v_i, v_j)$ is a distance between nodes v_i and v_j, $h \in \mathbb{N}, h \in [1, D]$, δ_h used to quantizes interval $[0, D]$. So NDD matrix provides information about the global topology of the graph.

Node features matrix constuction is implemented using

`torch_geometric.transforms` [10] package. It allows to apply transformation to feature matrix by concatenating or replacing current matrix with a new one.

One-hot encoded degrees features are added using built-in transformation `OneHotDegree` [11]

Encoded degrees were concatenated to the current node labels features. To use node distance matrix as a node feature matrix, new custom transformation was developed.

4.5 Results

Computational experiments includes training chosen graph neural network models on different datasets with two different Node feature matrices: one hot encoded degree matrix and node distance distribution matrix.

[10] `https://pytorch-geometric.readthedocs.io/en/latest/modules/transforms.html`

[11] `https://pytorch-geometric.readthedocs.io/en/latest/modules/transforms.html#torch_geometric.transforms.OneHotDegree`

		MUTAG	PROTEINS	ENZYMES	DD	Brain fMRI	Kidney RNA Seq
GCN	deg	**0.884±0.043**	**0.7468±0.011**	**0.3458±0.038**	0.756±0.01	0.56±0.0442	0.5583±0.032
	ndd	0.844 ± 0.037	0.7371±0.016	0.3±0.019	**0.795±0.009**	0.4667±0.084	0.5611±0.066
GAT	deg	0.687±0.045	0.6524±0.04	0.2667±0.044	0.7128±0.018	**0.6067±0.044**	0.5306±0.034
	ndd	0.853±0.033	0.7206±0.019	0.2639±0.038	0.7936±0.005	0.5767±0.05	**0.5639±0.04**
GCNII	deg	0.733±0.034	0.6996±0.03	0.2889±0.03	0.6929±0.023	0.58±0.093	0.475±0.087
	ndd	0.7156±0.029	0.7393±0.023	0.2333±0.038	0.7766±0.013	0.52±0.134	0.5556±0.049

Table 3: Experiments results

Accuracy is measured as mean of 5 separate runs, which include training dataset on train set, choosing best model using validation set and evaluate accuracy on test set, which is not used during train step. Based on these test accuracy metric, mean and standard deviation is measured.

Table 3 shows the result of experiments. For each model there are two types of result: "deg" and "ndd". Model marked with "deg" uses one hot encoded node degrees as a feature matrix, while "ndd" mark stands for NDD matrix as a feature matrix.

As result shows, that for all chosen TU Datasets, GCN model shows the best result. Accuracy of models trained and tested on one hot encoded degree feature matrices are is higher then one on NDD matrix, except accuracy of tests on DD dataset.

Tests on datasets "Brain fMRI" and "Kidney RNA Seq" show rather low accuracy. The best accuracy for "Brain fMRI" shows GAT model with encoded degrees matrix. The best accuracy for "Kidney RNA Seq" shows GAT model with NDD matrix.

5 Conclusion

Within current papper several GNN architectures have been applied to the task of graph classification in biomedical field and compared results. The following models have been compared: GCN[23], GAT[39] and GCNII[7]. There have been a process of hyperparamets tuning, training and evaluation of models. Mentioned models were applied to the following biomedical datasets: : `MUTAG`, `PROTEINS`, `ENZYMES`, `DD`, `Brain fMRI` and `Kidney RNASeq`. The result of experiments were provided in the corresponding section. All code for preparing dataset, model architectures and scripts for training and evaluation of neural networks are available in GitHub repository: `https://github.com/ArtamonovDen/gnn-benchmark`.

References

[1] T. Pfaff R. Ying J. Leskovec P.W. Battaglia A. Sanchez-Gonzalez J. Godwin. "Learning to simulate complex physics with graph networks". In: (2020).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473.

[3] Y. Bengio and Yann Lecun. "Convolutional Networks for Images, Speech, and Time-Series". In: (Nov. 1997).

[4] KM Borgwardt et al. "Protein function prediction via graph kernels". In: *Bioinformatics* 21 Suppl 1 (Jan. 2005), pp. i47–56.

[5] F. Monti F. Frasca D Eynard D. Mannion M.M. Bronstein. "Fake News Detection on Social Media using Geometric Deep Learning". In: (2019).

[6] Joan Bruna et al. *Spectral Networks and Locally Connected Networks on Graphs*. 2013. DOI: 10.48550/ARXIV.1312.6203. URL: https://arxiv.org/abs/1312.6203.

[7] Ming Chen et al. *Simple and Deep Graph Convolutional Networks*. 2020. DOI: 10.48550/ARXIV.2007.02133. URL: https://arxiv.org/abs/2007.02133.

[8] Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. "Understanding Convolutions on Graphs". In: *Distill* (2021). https://distill.pub/2021/understanding-gnns. DOI: 10.23915/distill.00032.

[9] Shusterman A. J. Hansch C. Debnath A. K. Lopez de Compadre R. L. Debnath G. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity". In: *Journal of medicinal chemistry* 34.2 (1991). URL: https://doi.org/10.1021/jm00106a046.

[10] Paul Dobson and Andrew Doig. "Distinguishing Enzyme Structures from Non-enzymes Without Alignments". In: *Journal of molecular biology* 330 (Aug. 2003), pp. 771–83. DOI: 10.1016/S0022-2836(03)00628-4.

[11] Paul D. Dobson and Andrew J. Doig. "Distinguishing Enzyme Structures from Non-enzymes Without Alignments". In: *Journal of Molecular Biology* 330.4 (2003), pp. 771–783. ISSN: 0022-2836. DOI: https://doi.org/10.1016/S0022-2836(03)00628-4. URL: https://www.sciencedirect.com/science/article/pii/S0022283603006284.

[12] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. DOI: 10.48550/ARXIV.1903.02428. URL: https://arxiv.org/abs/1903.02428.

[13] Alex Fout et al. "Protein Interface Prediction using Graph Convolutional Networks". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf.

[14] P. Frasconi, M. Gori, and A. Sperduti. "A general framework for adaptive processing of data structures". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 768–786. DOI: 10.1109/72.712151.

[15] Scarselli F. Gori M. Tsoi A. Hagenbuchner M. Monfardini G. "The graph neural network model". In: *IEEE Transactions on Neural Networks,* 20 (2009).

[16] Jonas Gehring et al. *A Convolutional Encoder Model for Neural Machine Translation*. 2016. DOI: 10.48550/ARXIV.1611.02344. URL: https://arxiv.org/abs/1611.02344.

[17] M. Gori, G. Monfardini, and F. Scarselli. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International*

Joint Conference on Neural Networks, 2005. Vol. 2. 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942.

[18] Guo S. Lin Y. Feng N. Song C. Wan H. "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting". In: *AAAI* (2019).

[19] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. *Wavelets on Graphs via Spectral Graph Theory.* 2009. DOI: 10.48550/ARXIV.0912.3848. URL: https://arxiv.org/abs/0912.3848.

[20] Kaiming He et al. *Deep Residual Learning for Image Recognition.* 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[21] Mingguo He, Zhewei Wei, and Ji-Rong Wen. *Convolutional Neural Networks on Graphs with Chebyshev Approximation, Revisited.* 2022. DOI: 10.48550/ARXIV.2202.03580. URL: https://arxiv.org/abs/2202.03580.

[22] W. Zhang X. Yue G. Tang W. Wu F. Huang and X. Zhang. "Sfpel-lpi: Sequence-based feature projection ensemble learning for predicting lncrna-protein interactions". In: *PLoS computational biology* 14 (2018).

[23] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks.* 2016. DOI: 10.48550/

ARXIV.1609.02907. URL: https://arxiv.org/abs/1609.02907.

[24] J. S. Viloria M. F. Allega M. Lambrughi and E. Papaleo. "An optimal distance cutoff for contact-based protein structure networks using side-chain centers of mass". In: *Scientific reports* 7 (2017).

[25] Qimai Li, Zhichao Han, and Xiao-Ming Wu. *Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning*. 2018. DOI: 10.48550/ARXIV.1801.07606. URL: https://arxiv.org/abs/1801.07606.

[26] K. Sato M. Hirohara Y. Saito Y. Koda and Y. Sakakibara. "Convolutional neural network based on smiles representation of compounds for detecting chemical motif". In: *BMC bioinformatics* 19 (2018).

[27] Jingwei Ma et al. "MMM: Multi-source Multi-net Micro-video Recommendation with Clustered Hidden Item Representation Learning". In: *Data Science and Engineering* 4 (Sept. 2019). DOI: 10.1007/s41019-019-00101-4.

[28] Ichcha Manipur et al. "Netpro2vec: a Graph Embedding Framework for Biomedical Applications". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2021).

[29] C. Stam E. Van Straaten E. Van Dellen P. Tewarie G. Gong A. Hillebrand J. Meier and P. Van Mieghem. "The relation between structural and functional connectivity patterns in complex brain networks". In: *Psychophysiol* 103 (2016).

[30] Christopher Morris et al. "TUDataset: A collection of benchmark datasets for learning with graphs". In: *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. 2020. arXiv: 2007.08663. URL: www.graphlearning.io.

[31] Jiezhong Qiu et al. "DeepInf". In: ACM, July 2018. DOI: 10.1145/3219819.3220077. URL: https://doi.org/10.1145%2F3219819.3220077.

[32] S. Rasti and C. Vogiatzis. "A survey of computational methods in protein–protein interaction networks". In: *Ann. Oper. Res.,* 276 (2019).

[33] Jesús D. Arroyo Relión et al. "Network classification with applications to brain connectomics". In: *The Annals of Applied Statistics* 13.3 (Sept. 2019). DOI: 10.1214/19-aoas1252. URL: https://doi.org/10.1214%2F19-aoas1252.

[34] Franco Scarselli et al. "Graphical-Based Learning Environments for Pattern Recognition". In: *Structural, Syntactic, and Statistical Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 42–56. ISBN: 978-3-540-27868-9.

[35] Ida Schomburg et al. "BRENDA, the enzyme database: Updates and major new developments". In: *Nucleic acids research* 32 (Jan. 2004), pp. D431–3. DOI: 10.1093/nar/gkh081.

[36] Junyuan Shang et al. *GAMENet: Graph Augmented MEmory Networks for Recommending Medication Combination*. 2018. DOI: 10.48550/ARXIV.1809.01852. URL: https://arxiv.org/abs/1809.01852.

[37] A. Sperduti and A. Starita. "Supervised neural networks for the classification of structures". In: *IEEE Transactions on Neural Networks* 8.3 (1997), pp. 714–735. DOI: 10.1109/72.572108.

[38] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[39] Petar Veličković et al. *Graph Attention Networks*. 2017. DOI: 10.48550/ARXIV.1710.10903. URL: https://arxiv.org/abs/1710.10903.

[40] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. DOI: 10.1109/tnnls.2020.2978386. URL: https://doi.org/10.1109%2Ftnnls.2020.2978386.

[41] Rex Ying et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: ACM, July 2018. DOI: 10.1145/3219819.3219890. URL: https://doi.org/10.1145%2F3219819.3219890.

[42] Long Zhao et al. “Semantic Graph Convolutional Networks for 3D Human Pose Regression”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019. DOI: 10.1109/cvpr.2019.00354. URL: https://doi.org/10.1109%2Fcvpr.2019.00354.