

---

# CogDL Documentation

*Release 0.4.1*

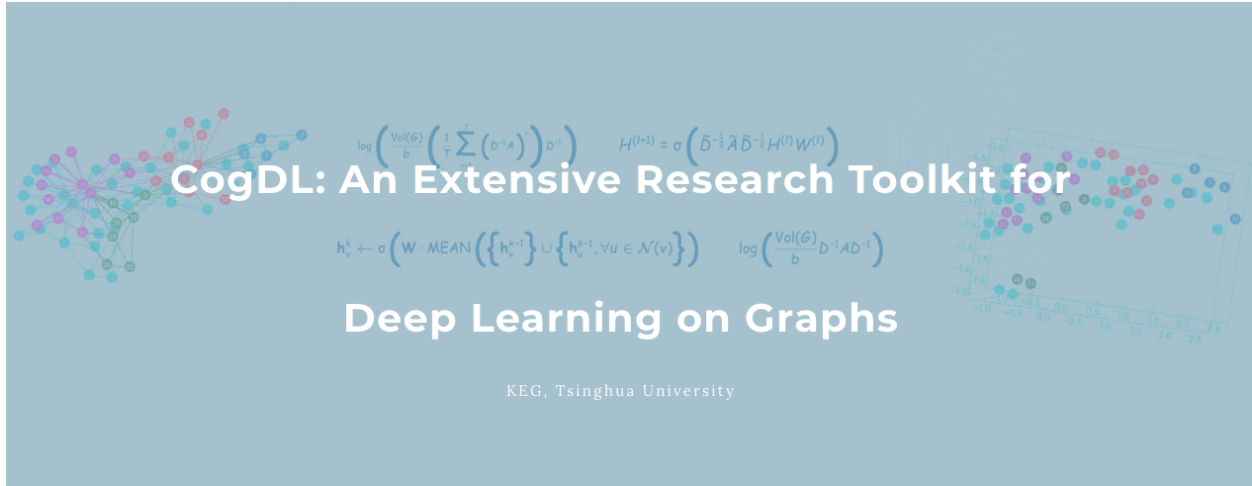
**KEG**

**Aug 13, 2021**



<b>1</b>	<b>News</b>	<b>3</b>
<b>2</b>	<b>Citing CogDL</b>	<b>5</b>
2.1	Install . . . . .	5
2.2	Quick Start . . . . .	6
2.3	Brief Tutorial . . . . .	7
2.4	Tasks . . . . .	16
2.5	Trainer . . . . .	29
2.6	Model . . . . .	30
2.7	Dataset . . . . .	31
2.8	data . . . . .	31
2.9	datasets . . . . .	35
2.10	tasks . . . . .	47
2.11	models . . . . .	52
2.12	layers . . . . .	85
2.13	options . . . . .	91
2.14	utils . . . . .	92
2.15	experiments . . . . .	93
2.16	pipelines . . . . .	93
<b>3</b>	<b>Indices and tables</b>	<b>95</b>
	<b>Python Module Index</b>	<b>97</b>
	<b>Index</b>	<b>99</b>





CogDL is a graph representation learning toolkit that allows researchers and developers to easily train and compare baseline or customized models for node classification, graph classification, and other important tasks in the graph domain.

We summarize the contributions of CogDL as follows:

- **High Efficiency:** CogDL utilizes well-optimized operators to speed up training and save GPU memory of GNN models.
- **Easy-to-Use:** CogDL provides easy-to-use APIs for running experiments with the given models and datasets using hyper-parameter search.
- **Extensibility:** The design of CogDL makes it easy to apply GNN models to new scenarios based on our framework.
- **Reproducibility:** CogDL provides reproducible leaderboards for state-of-the-art models on most of important tasks in the graph domain.



# CHAPTER 1

---

## News

---

- The new **v0.4.1 release** adds the implementation of Deep GNNs and the recommendation task. It also supports new pipelines for generating embeddings and recommendation. Welcome to join our tutorial on KDD 2021 at 10:30 am - 12:00 am, Aug. 14th (Singapore Time). More details can be found in <https://kdd2021graph.github.io/>.
- The new **v0.4.0 release** refactors the data storage (from `Data` to `Graph`) and provides more fast operators to speed up GNN training. It also includes many self-supervised learning methods on graphs. BTW, we are glad to announce that we will give a tutorial on KDD 2021 in August. Please see [this link](#) for more details.
- The new **v0.3.0 release** provides a fast spmm operator to speed up GNN training. We also release the first version of [CogDL paper](#) in arXiv. You can join [our slack](#) for discussion.
- The new **v0.2.0 release** includes easy-to-use `experiment` and `pipeline` APIs for all experiments and applications. The `experiment` API supports automl features of searching hyper-parameters. This release also provides `OAGBert` API for model inference (`OAGBert` is trained on large-scale academic corpus by our lab). Some features and models are added by the open source community (thanks to all the contributors).
- The new **v0.1.2 release** includes a pre-training task, many examples, OGB datasets, some knowledge graph embedding methods, and some graph neural network models. The coverage of CogDL is increased to 80%. Some new APIs, such as `Trainer` and `Sampler`, are developed and being tested.
- The new **v0.1.1 release** includes the knowledge link prediction task, many state-of-the-art models, and `optuna` support. We also have a [Chinese WeChat post](#) about the CogDL release.





Please cite our paper if you find our code or results useful for your research:

```
@article{cen2021cogdl,  
  title={CogDL: An Extensive Toolkit for Deep Learning on Graphs},  
  author={Yukuo Cen and Zhenyu Hou and Yan Wang and Qibin Chen and Yizhen Luo and  
↪Xingcheng Yao and Aohan Zeng and Shiguang Guo and Peng Zhang and Guohao Dai and Yu_  
↪Wang and Chang Zhou and Hongxia Yang and Jie Tang},  
  journal={arXiv preprint arXiv:2103.00959},  
  year={2021}  
}
```

## 2.1 Install

- Python version  $\geq 3.6$
- PyTorch version  $\geq 1.7.1$

Please follow the instructions here to install PyTorch (<https://github.com/pytorch/pytorch#installation>).

When PyTorch has been installed, cogdl can be installed using pip as follows:

```
pip install cogdl
```

Install from source via:

```
pip install git+https://github.com/thudm/cogdl.git
```

Or clone the repository and install with the following commands:

```
git clone git@github.com:THUDM/cogdl.git  
cd cogdl  
pip install -e .
```

If you want to use the modules from PyTorch Geometric (PyG), and Deep Graph Library (DGL), you can follow the instructions to install PyTorch Geometric ([https://github.com/rusty1s/pytorch\\_geometric/#installation](https://github.com/rusty1s/pytorch_geometric/#installation)) and Deep Graph Library (<https://docs.dgl.ai/install/index.html>).

## 2.2 Quick Start

### 2.2.1 API Usage

You can run all kinds of experiments through CogDL APIs, especially `experiment()`. You can also use your own datasets and models for experiments. A quickstart example can be found in the `quick_start.py`. More examples are provided in the `examples/`.

```
from cogdl import experiment

# basic usage
experiment(task="node_classification", dataset="cora", model="gcn")

# set other hyper-parameters
experiment(task="node_classification", dataset="cora", model="gcn", hidden_size=32,
↳max_epoch=200)

# run over multiple models on different seeds
experiment(task="node_classification", dataset="cora", model=["gcn", "gat"], seed=[1,
↳2])

# automl usage
def func_search(trial):
    return {
        "lr": trial.suggest_categorical("lr", [1e-3, 5e-3, 1e-2]),
        "hidden_size": trial.suggest_categorical("hidden_size", [32, 64, 128]),
        "dropout": trial.suggest_uniform("dropout", 0.5, 0.8),
    }

experiment(task="node_classification", dataset="cora", model="gcn", seed=[1, 2], func_
↳search=func_search)
```

### 2.2.2 Command-Line Usage

You can also use `python scripts/train.py --task example_task --dataset example_dataset --model example_model` to run `example_model` on `example_data` and evaluate it via `example_task`.

- `--task`, downstream tasks to evaluate representation like `node_classification`, `unsupervised_node_classification`, `graph_classification`. More tasks can be found in the `cogdl/tasks`.
- `--dataset`, dataset name to run, can be a list of datasets with space like `cora citeseer ppi`. Supported datasets include ‘cora’, ‘citeseer’, ‘pumbed’, ‘ppi’, ‘wikipedia’, ‘blogcatalog’, ‘flickr’. More datasets can be found in the `cogdl/datasets`.
- `--model`, model name to run, can be a list of models like `deepwalk line prone`. Supported models include ‘gcn’, ‘gat’, ‘graphsage’, ‘deepwalk’, ‘node2vec’, ‘hope’, ‘grarep’, ‘netmf’, ‘netsmf’, ‘prone’. More models can be found in the `cogdl/models`.

For example, if you want to run LINE, NetMF on Wikipedia with unsupervised node classification task, with 5 different seeds:

```
python scripts/train.py --task unsupervised_node_classification --dataset wikipedia --
↳model line netmf --seed 0 1 2 3 4
```

Expected output:

Variant	Micro-F1 0.1	Micro-F1 0.3	Micro-F1 0.5	Micro-F1 0.7	Micro-F1 0.9
('wikipedia', 'line')	0.4069±0.0011	0.4071±0.0010	0.4055±0.0013	0.4054±0.0020	0.4080±0.0042
('wikipedia', 'netmf')	0.4551±0.0024	0.4932±0.0022	0.5046±0.0017	0.5084±0.0057	0.5125±0.0035

If you want to run parallel experiments on your server with multiple GPUs on multiple models, GCN and GAT, on the Cora dataset with node classification task:

```
python scripts/parallel_train.py --task node_classification --dataset cora --model_
↳gcn gat --device-id 0 1 --seed 0 1 2 3 4
```

Expected output:

Variant	Acc
('cora', 'gcn')	0.8236±0.0033
('cora', 'gat')	0.8262±0.0032

## 2.2.3 Fast-Spmm Usage

CogDL provides a fast sparse matrix-matrix multiplication operator called **GE-SpMM** to speed up training of GNN models on the GPU. You can set `fast_spmm=True` in the API usage or `--fast-spmm` in the command-line usage to enable this feature. Note that this feature is still in testing and may not work under some versions of CUDA.

## 2.3 Brief Tutorial

### 2.3.1 Node Classification

Graph neural networks(GNN) have great power in tackling graph-related tasks. In this chapter, we take node classification as an example and show how to use CogDL to finish a workflow using GNN. In supervised setting, node classification aims to predict the ground truth label for each node.

#### Quick Start

CogDL provides abundant of common benchmark datasets and GNN models. On the one hand, you can simply start a running using models and datasets in CogDL. This is convenient when you want to test the reproducibility of proposed GNN or get baseline results in different datasets.

```
from cogdl import experiment
experiment(model="gcn", dataset="cora", task="node_classification")
```

Or you can create each component separately and manually run the process using `build_dataset`, `build_model`, `build_task` in CogDL.

```
from cogdl.datasets import build_dataset
from cogdl.models import build_model
from cogdl.tasks import build_task

args = build_args_from_dict(dict(task="node_classification", model="gcn", dataset=
↪"cora"))
dataset = build_dataset(args)
model = build_model(args)
task = build_task(args, dataset=dataset, model=model)
task.train()
```

As show above, model/dataset/task are 3 key components in establishing a training process. In fact, CogDL also supports customized model and datasets. This will be introduced in next chapter. In the following we will briefly show the details of each component.

### Save trained model

CogDL supports saving the trained model with `save_model` in command line or notebook. For example:

```
experiment(model="gcn", task="node_classification", dataset="cora", save_model="gcn_
↪cora.pt")
```

When the training stops, the model will be saved in `gcn_cora.py`. If you want to continue the training from previous checkpoint with different parameters(such as learning rate, weight decay and etc.), keep the same model parameters (such as hidden size, model layers) and do it as follows:

```
experiment(model="gcn", task="node_classification", dataset="cora", checkpoint="gcn_
↪cora.pt")
```

Or you may just want to do the inference to get prediction results without training. The prediction results will be automatically saved in `gcn_cora.pred`.

```
experiment(model="gcn", task="node_classification", dataset="cora", checkpoint="gcn_
↪cora.pt", inference=True)
```

In command line usage, the same results can be achieved with `--save-model {path}`, `--checkpoint {path}` and `--inference` set.

### Save embeddings

Graph representation learning (network embedding and unsupervised GNNs) aims to get node representation. The embeddings can be used in various downstream applications. CogDL will save node embeddings in directory `./embedding`. As shown below, the embeddings will be save in `./embedding/prone_blogcatalog.npy`.

```
experiment(model="prone", dataset="blogcatalog", task="unsupervised_node_
↪classification")
```

Evaluation on node classification will run as the end of training. We follow the same experimental settings used in DeepWalk, Node2Vec and ProNE. We randomly sample different percentages of labeled nodes for training a liblinear classifier and use the remaining for testing We repeat the training for several times and report the average Micro-F1. By default, CogDL samples 90% labeled nodes for training for one time. You are expected to change the setting with `--num-shuffle` and `--training-percents` to your needs.

In addition, CogDL supports evaluating node embeddings without training in different evaluation settings. The following code snippet evaluates the embedding we get above:

```

experiment (
    model="prone",
    dataset="blogcatalog",
    task="unsupervised_node_classification",
    load_emb_path="./embedding/prone_blogcatalog.npy",
    num_shuffle=5,
    training_percents=[0.1, 0.5, 0.9]
)

```

You can also use command line to achieve the same quickly

```

# Get embedding
python script/train.py --model prone --task unsupervised_node_classification --
↳dataset blogcatalog

# Evaluate only
python script/train.py --model prone --task unsupervised_node_classification --
↳dataset blogcatalog --load-emb-path ./embedding/prone_blogcatalog.npy --num-shuffle_
↳5 --training-percents 0.1 0.5 0.9

```

## 2.3.2 Graph Storage

A graph is used to store information of structured data. CogDL represents a graph with a `cogdl.data.Graph` object. Briefly, a `Graph` holds the following attributes:

- `x`: Node feature matrix with shape `[num_nodes, num_features]`, *torch.Tensor*
- `edge_index`: COO format sparse matrix, *Tuple*
- `edge_weight`: Edge weight with shape `[num_edges, ]`, *torch.Tensor*
- `edge_attr`: Edge attribute matrix with shape `[num_edges, num_attr]`
- `y`: Target labels of each node, with shape `[num_nodes, ]` for single label case and `[num_nodes, num_labels]` for multi-label case
- `row_indptr`: Row index pointer for CSR sparse matrix, *torch.Tensor*.
- `col_indices`: Column indices for CSR sparse matrix, *torch.Tensor*.
- `num_nodes`: The number of nodes in graph.
- `num_edges`: The number of edges in graph.

The above are the basic attributes but are not necessary. You may define a graph with `g = Graph(edge_index=edges)` and omit the others. Besides, `Graph` is not restricted to these attributes and other self-defined attributes, e.x. `graph.mask = mask`, are also supported.

`Graph` stores sparse matrix with COO or CSR format. COO format is easier to add or remove edges, e.x. `add_self_loops`, and CSR is stored for fast message-passing. `Graph` automatically convert between two formats and you can use both on demands without worrying. You can create a `Graph` with edges or assign edges to a created graph. `edge_weight` will be automatically initialized as all ones, and you can modify it to fit your need.

```

import torch
from cogdl.data import Graph
edges = torch.tensor([[0,1], [1,3], [2,1], [4,2], [0,3]]) .t()
g = Graph()
g.edge_index = edges
g = Graph(edge_index=edges) # equivalent to that above

```

(continues on next page)

(continued from previous page)

```
print(g.edge_weight)
>> tensor([1., 1., 1., 1., 1.])
g.num_nodes
>> 5
g.num_edges
>> 5
g.edge_weight = torch.rand(5)
print(g.edge_weight)
>> tensor([0.8399, 0.6341, 0.3028, 0.0602, 0.7190])
```

We also implement commonly used operations in Graph:

- `add_self_loops`: add self loops for nodes in graph,

$$\hat{A} = A + I$$

- `add_remaining_self_loops`: add self-loops for nodes without it.
- `sym_norm`: symmetric normalization of `edge_weight` used *GCN*:

$$\hat{A} = D^{-1/2}AD^{-1/2}$$

- `row_norm`: row-wise normalization of `edge_weight`:

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

- `degrees`: get degrees for each node. For directed graph, this function returns in-degrees of each node.

```
import torch
from cogdl.data import Graph
edge_index = torch.tensor([[0,1],[1,3],[2,1],[4,2],[0,3]]).t()
g = Graph(edge_index=edge_index)
>> Graph(edge_index=[2, 5])
g.add_remaining_self_loops()
>> Graph(edge_index=[2, 10], edge_weight=[10])
>> print(edge_weight) # tensor([1., 1., ..., 1.])
g.row_norm()
>> print(edge_weight) # tensor([0.3333, ..., 0.50])
```

- `subgraph`: get a subgraph containing given nodes and edges between them.
- `edge_subgraph`: get a subgraph containing given edges and corresponding nodes.
- `sample_adj`: sample a fixed number of neighbors for each given node.

```
from cogdl.datasets import build_dataset_from_name
g = build_dataset_from_name("cora")[0]
g.num_nodes
>> 2707
g.num_edges
>> 10184
# Get a subgraph containing nodes [0, .., 99]
sub_g = g.subgraph(torch.arange(100))
>> Graph(x=[100, 1433], edge_index=[2, 18], y=[100])
# Sample 3 neighbors for each nodes in [0, .., 99]
nodes, adj_g = g.sample_adj(torch.arange(100), size=3)
>> Graph(edge_index=[2, 300]) # adj_g
```

- `train/eval`: In inductive settings, some nodes and edges are unseen during training, `train/eval` provides access to switching backend graph for training/evaluation. In transductive setting, you may ignore this.

```
# train_step
model.train()
graph.train()

# inference_step
model.eval()
data.eval()
```

## Mini-batch Graphs

In node classification, all operations are in one single graph. But in tasks like graph classification, we need to deal with many graphs with mini-batch. Datasets for graph classification contains graphs which can be accessed with index, e.x. `data[2]`. To support mini-batch training/inference, CogDL combines graphs in a batch into one whole graph, where adjacency matrices form sparse block diagonal matrices and others (node features, labels) are concatenated in node dimension. `cogdl.data.DataLoader` handles the process.

```
from cogdl.data import DataLoader
from cogdl.datasets import build_dataset_from_name

dataset = build_dataset_from_name("mutag")
>> MUTAGDataset(188)
dataset[0]
>> Graph(x=[17, 7], y=[1], edge_index=[2, 38])
loader = DataLoader(dataset, batch_size=8)
for batch in loader:
    model(batch)
>> Batch(x=[154, 7], y=[8], batch=[154], edge_index=[2, 338])
```

`batch` is an additional attributes that indicate the respective graph the node belongs to. It is mainly used to do global pooling, or called *readout* to generate graph-level representation. Concretely, `batch` is a tensor like:

$$batch = [0, \dots, 0, 1, \dots, 1, N - 1, \dots, N - 1]$$

The following code snippet shows how to do global pooling to sum over features of nodes in each graph:

```
def batch_sum_pooling(x, batch):
    batch_size = int(torch.max(batch.cpu())) + 1
    res = torch.zeros(batch_size, x.size(1)).to(x.device)
    out = res.scatter_add_(
        dim=0,
        index=batch.unsqueeze(-1).expand_as(x),
        src=x
    )
    return out
```

## Editing Graphs

Mutation or changes can be applied to edges in some settings. In such cases, we need to *generate* a graph for calculation while keep the original graph. CogDL provides `graph.local_graph` to set up a local scape and any out-of-place operation will not reflect to the original graph. However, in-place operation will affect the original graph.

```
graph = build_dataset_from_name("cora")[0]
graph.num_edges
```

(continues on next page)

(continued from previous page)

```
>> 10184
with graph.local_graph():
    mask = torch.arange(100)
    row, col = graph.edge_index
    graph.edge_index = (row[mask], col[mask])
    graph.num_edges
>> 100
graph.num_edges
>> 10184

graph.edge_weight
>> tensor([1., ..., 1.])
with graph.local_graph():
    graph.edge_weight += 1
graph.edge_weight
>> tensor([2., ..., 2.]])
```

## Common benchmarks

CogDL provides a bunch of commonly used datasets for graph tasks like node classification, graph classification and many others. You can access them conveniently shown as follows. Statistics of datasets are on [this page](#).

```
from cogdl.datasets import build_dataset_from_name, build_dataset
dataset = build_dataset_from_name("cora")
dataset = build_dataset(args) # args.dataet = "cora"
```

For all datasets for node classification, we use *train\_mask*, *val\_mask*, *test\_mask* to denote train/validation/test split for nodes.

## 2.3.3 Using customized GNN

Sometimes you would like to design your own GNN module or use GNN for other purposes. In this chapter, we introduce how to use GNN layer in CogDL to write your own GNN model and how to write a GNN layer from scratch.

### GNN layers in CogDL to Define model

CogDL has implemented popular GNN layers in `cogdl.layers`, and they can serve as modules to help design new GNNs. Here is how we implement [Jumping Knowledge Network \(JKNet\)](#) with `GCNLayer` in CogDL.

JKNet collects the output of all layers and concatenate them together to get the result:

$$\begin{aligned}
 H^{(0)} &= X \\
 H^{(i+1)} &= \sigma(\hat{A}H^{(i)}W^{(i)}) \\
 OUT &= CONCAT([H^{(0)}, \dots, H^{(L)}])
 \end{aligned}$$

```
import torch
from cogdl.models import register_model

@register_model("jknet")
class JKNet(BaseModel):
```

(continues on next page)



(continued from previous page)

```

def __init__(self, in_feats, out_feats, hidden_size, num_layers):
    super(JKNet, self).__init__()
    shapes = [in_feats] + [hidden_size] * num_layers
    #
    self.layers = nn.ModuleList([
        GCNLayer(shape[i], shape[i+1])
        for i in range(num_layers)
    ])
    self.fc = nn.Linear(hidden_size * num_layers, out_feats)

def forward(self, graph):
    graph.add_remaining_self_loops()
    graph.sym_norm()
    h = graph.x
    out = []
    for layer in self.layers:
        h = layer(x)
        out.append(h)
    out = torch.cat(out, dim=1)
    return self.fc(out)

```

## Define your GNN Module

In most cases, you may build a layer module with new message propagation and aggregation scheme. Here the code snippet shows how to implement a GCNLayer using Graph and efficient sparse matrix operators in CogDL.

```

import torch
from cogdl.utils import spmm

class GCNLayer(torch.nn.Module):
    """
    Args:
        in_feats: int
            Input feature size
        out_feats: int
            Output feature size
    """
    def __init__(self, in_feats, out_feats):
        super(GCNLayer, self).__init__()
        self.fc = torch.nn.Linear(in_feats, out_feats)

    def forward(self, graph, x):
        # symmetric normalization of adjacency matrix
        graph.sym_norm()
        h = self.fc(x)
        h = spmm(graph, h)
        return h

```

spmm is sparse matrix multiplication operation frequently used in GNNs.

$$H = AH = SpMM(A, H)$$

Sparse matrix is stored in Graph and will be called automatically. Message-passing in spatial space is equivalent to matrix operations. CogDL also supports other efficient operators like `edge_softmax` and `multi_head_spmm`, you can refer to this [page](#) for usage.

## Use Custom models with CogDL

Now that you have defined your own GNN, you can use dataset/task in CogDL to immediately train and evaluate the performance of your model.

```
data = dataset.data
# Use the JKNet model as defined above
model = JKNet(data.num_features, data.num_classes, 32, 4)
task = build_task(args, dataset=dataset, model=model)
task.train()

# Or you may simple run the command after `register_model`
experiment(model="jknet", task="node_classification", dataset="cora")
```

### 2.3.4 Using customized Dataset

CogDL has provided lots of common datasets. But you may wish to apply GNN to new datasets for different applications. CogDL provides an interface for customized datasets. You take care of reading in the dataset and the rest is to CogDL

We provide `NodeDataset` and `GraphDataset` as abstract classes and implement necessary basic operations.

#### Dataset for node\_classification

To create a dataset for `node_classification`, you need to inherit `NodeDataset`. `NodeDataset` is for tasks like `node_classification` or `unsupervised_node_classification`, which focus on node-level prediction. Then you need to implement `process` method. In this method, you are expected to read in your data and preprocess raw data to the format available to CogDL with `Graph`. Afterwards, we suggest you to save the processed data (we will also help you do it as you return the data) to avoid doing the preprocessing again. Next time you run the code, CogDL will directly load it.

The running process of the module is as follows:

1. Specify the path to save processed data with `self.path`
2. Function `process` is called to load and preprocess data and your data is saved as `Graph` in `self.path`. This step will be implemented the first time you use your dataset. And then every time you use your dataset, the dataset will be loaded from `self.path` for convenience.
3. For dataset, for example, named `MyNodeDataset` in node-level tasks, You can access the data/Graph via `MyNodeDataset.data` or `MyDataset[0]`.

In addition, evaluation metric for your dataset should be specified. CogDL provides `accuracy` and `multiclass_f1` for multi-class classification, `multilabel_f1` for multi-label classification.

If `scale_feat` is set to be `True`, CogDL will normalize node features with mean  $u$  and variance  $s$ :

$$z = (x - u) / s$$

Here is an example:

```
from cogdl.data import Graph
from cogdl.datasets import NodeDataset, register_dataset

@register_dataset("node_dataset")
class MyNodeDataset(NodeDataset):
    def __init__(self, path="data.pt"):
        self.path = path
```

(continues on next page)

(continued from previous page)

```

    super(MyNodeDataset, self).__init__(path, scale_feat=False, metric="accuracy")

    def process(self):
        """You need to load your dataset and transform to `Graph`"""
        # Load and preprocess data
        edge_index = torch.tensor([[0, 1], [0, 2], [1, 2], [1, 3]].t())
        x = torch.randn(4, 10)
        mask = torch.bool(4)
        # Provide attributes as you need and save the data into `Graph`
        data = Graph(x=x, edge_index=edge_index)
        torch.save(data, self.path)
        return data

dataset = MyNodeDataset("data.pt")

```

### Dataset for graph\_classification

Similarly, you need to inherit `GraphDataset` when you want to build a dataset for graph-level tasks such as *graph\_classification*. The overall implementation is similar while the difference is in `process`. As `GraphDataset` contains a lot of graphs, you need to transform your data to `Graph` for each graph separately to form a list of `Graph`. An example is shown as follows:

```

from cogdl.datasets import GraphDataset

@register_dataset("graph_dataset")
class MyGraphDataset(GraphDataset):
    def __init__(self, path="data.pt"):
        self.path = path
        super(MyGraphDataset, self).__init__(path, metric="accuracy")

    def process(self):
        # Load and preprocess data
        # Here we randomly generate several graphs for simplicity as an example
        graphs = []
        for i in range(10):
            edges = torch.randint(0, 20, (2, 30))
            label = torch.randint(0, 7, (1,))
            graphs.append(Graph(edge_index=edges, y=label))
        torch.save(graphs, self.path)
        return graphs

```

### Use custom dataset with CogDL

Now that you have set up your dataset, you can use models/task in CogDL immediately to get results.

```

# Use the GCN model with the dataset we define above
dataset = MyNodeDataset("data.pt")
args.model = "gcn"
task = build_task(args, dataset=dataset)
task.train()

# Or you may simple run the command after `register_dataset`
experiment(model="gcn", task="node_classification", dataset="node_dataset")

```

(continues on next page)

(continued from previous page)

```
# That's the same for other tasks
experiment(model="gin", task="graph_classification", dataset="graph_dataset")
```

## 2.4 Tasks

### 2.4.1 Node Classification

In this tutorial, we will introduce a important task, node classification. In this task, we train a GNN model with partial node labels and use accuracy to measure the performance.

#### Semi-supervised Node Classification Methods

Method	Sampling	Inductive	Reproducibility
GCN			
GAT			
Chebyshev			
GraphSAGE			
GRAND			
GCNII			
DeeperGCN			
Dr-GAT			
U-net			
APPNP			
GraphMix			
DisenGCN			
SGC			
JKNet			
MixHop			
DropEdge			
SRGCN			

---

**Tip:** Reproducibility means whether the model is reproduced in our experimental setting currently.

---

First we define the *NodeClassification* class.

```
@register_task("node_classification")
class NodeClassification(BaseTask):
    """Node classification task."""

    @staticmethod
    def add_args(parser):
        """Add task-specific arguments to the parser."""

    def __init__(self, args):
        super(NodeClassification, self).__init__(args)
```

Then we can build dataset and model according to args. Generally the model and dataset should be placed in the same device using *.to(device)* instead of *.cuda()*. And then we set the optimizer.

```

self.device = torch.device('cpu' if args.cpu else 'cuda')
# build dataset with `build_dataset`
dataset = build_dataset(args)
self.data = dataset.data
self.data.apply(lambda x: x.to(self.device))
args.num_features = dataset.num_features
args.num_classes = dataset.num_classes

# build model with `build_model`
model = build_model(args)
self.model = model.to(self.device)
self.patience = args.patience
self.max_epoch = args.max_epoch

# set optimizer
self.optimizer = torch.optim.Adam(
    self.model.parameters(), lr=args.lr, weight_decay=args.weight_decay
)

```

For the training process, `train` must be implemented as it will be called as the entrance of training. We provide a training loop for node classification task. For each epoch, we first call `_train_step` to optimize our model and then call `_test_step` for validation and test to compute the accuracy and loss.

```

def train(self):
    epoch_iter = tqdm(range(self.max_epoch))
    for epoch in epoch_iter:
        self._train_step()
        train_acc, _ = self._test_step(split="train")
        val_acc, val_loss = self._test_step(split="val")
        epoch_iter.set_description(
            f"Epoch: {epoch:03d}, Train: {train_acc:.4f}, Val: {val_acc:.4f}"
        )

def _train_step(self):
    """train step per epoch"""
    self.model.train()
    self.optimizer.zero_grad()
    # In node classification task, `node_classification_loss` must be defined in_
    ↪model if you want to use this task directly.
    self.model.node_classification_loss(self.data).backward()
    self.optimizer.step()

def _test_step(self, split="val"):
    """test step"""
    self.model.eval()
    # `Predict` should be defined in model for inference.
    logits = self.model.predict(self.data)
    logits = F.log_softmax(logits, dim=-1)
    mask = self.data.test_mask
    loss = F.nll_loss(logits[mask], self.data.y[mask]).item()

    pred = logits[mask].max(1)[1]
    acc = pred.eq(self.data.y[mask]).sum().item() / mask.sum().item()
    return acc, loss

```

In supervised node classification tasks, we use early stopping to reduce over-fitting and save training time.

```

if val_loss <= min_loss or val_acc >= max_score:
    if val_loss <= best_loss: # and val_acc >= best_score:
        best_loss = val_loss
        best_score = val_acc
        best_model = copy.deepcopy(self.model)
    min_loss = np.min((min_loss, val_loss))
    max_score = np.max((max_score, val_acc))
    patience = 0
else:
    patience += 1
    if patience == self.patience:
        self.model = best_model
        epoch_iter.close()
        break

```

Finally, we compute the accuracy scores of test set for the trained model.

```

test_acc, _ = self._test_step(split="test")
print(f"Test accuracy = {test_acc}")
return dict(Acc=test_acc)

```

The overall implementation of *NodeClassification* is at ([https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/node\\_classification.py](https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/node_classification.py)).

To run *NodeClassification*, we can use the following command:

```

python scripts/train.py --task node_classification --dataset cora citeseer --model_
↳gcn gat --seed 0 1 --max-epoch 500

```

Then We get experimental results like this:

Variant	Acc
('cora', 'gcn')	0.8220±0.0010
('cora', 'gat')	0.8275±0.0015
('citeseer', 'gcn')	0.7060±0.0050
('citeseer', 'gat')	0.7060±0.0020

## 2.4.2 Unsupervised Node Classification

In this tutorial, we will introduce a important task, unsupervised node classification. In this task, we usually apply L2 normalized logistic regression to train a classifier and use *F1-score* or *Accuracy* to measure the performance.

Unsupervised node classification includes *network embedding* methods(DeepWalk, LINE, ProNE and etc.) and *GNN self-supervised* methods(DGI, GraphSAGE and etc.). In this section, we mainly introduce the part for *network embeddings* and the other will be presented in next section *trainer*.

### Unsupervised Graph Embedding Methods

Method	Weighted	shallow network	Matrix Factorization	Reproducibility	GPU support
DeepWalk					
LINE					
Node2Vec					
NetMF					
NetSMF					
HOPE					
GraRep					
SDNE					
DNGR					
ProNE					

### Unsupervised Graph Neural Network Representation Learning Methods

Method	Sampling	Inductive	Reproducibility
DGI			
MVGRL			
GRACE			
GraphSAGE			

First we define the *UnsupervisedNodeClassification* class, which has two parameters *hidden-size* and *num-shuffle*. *hidden-size* represents the dimension of node representation, while *num-shuffle* means the shuffle times in classifier.

```
@register_task("unsupervised_node_classification")
class UnsupervisedNodeClassification(BaseTask):
    """Node classification task."""

    @staticmethod
    def add_args(parser):
        """Add task-specific arguments to the parser."""
        # fmt: off
        parser.add_argument("--hidden-size", type=int, default=128)
        parser.add_argument("--num-shuffle", type=int, default=5)
        # fmt: on

    def __init__(self, args):
        super(UnsupervisedNodeClassification, self).__init__(args)
```

Then we can build dataset according to input graph's type, and get *self.label\_matrix*.

```
dataset = build_dataset(args)
self.data = dataset[0]
if isinstance(dataset.__class__.__bases__[0], InMemoryDataset):
    self.num_nodes = self.data.y.shape[0]
    self.num_classes = dataset.num_classes
    self.label_matrix = np.zeros((self.num_nodes, self.num_classes), dtype=int)
    self.label_matrix[range(self.num_nodes), self.data.y] = 1
    self.data.edge_attr = self.data.edge_attr.t()
else:
    self.label_matrix = self.data.y
    self.num_nodes, self.num_classes = self.data.y.shape
```

After that, we can build model and run *model.train(G)* to obtain node representation.

```

self.model = build_model(args)
self.is_weighted = self.data.edge_attr is not None

def train(self):
    G = nx.Graph()
    if self.is_weighted:
        edges, weight = (
            self.data.edge_index.t().tolist(),
            self.data.edge_attr.tolist(),
        )
        G.add_weighted_edges_from(
            [(edges[i][0], edges[i][1], weight[0][i]) for i in range(len(edges))]
        )
    else:
        G.add_edges_from(self.data.edge_index.t().tolist())
    embeddings = self.model.train(G)

```

The spectral propagation in ProNE/ProNE++ can improve the quality of representation learned from other methods, so we can use *enhance\_emb* to enhance performance. ProNE++ automatically searches for the best graph filter to help improve the embedding.

```

if self.enhance is True:
    embeddings = self.enhance_emb(G, embeddings)

```

When the embeddings are obtained, we can save them at *self.save\_dir*.

At last, we evaluate embedding via run *num\_shuffle* times classification under different training ratio with *features\_matrix* and *label\_matrix*.

```

def _evaluate(self, features_matrix, label_matrix, num_shuffle):
    # shuffle, to create train/test groups
    shuffles = []
    for _ in range(num_shuffle):
        shuffles.append(skshuffle(features_matrix, label_matrix))

    # score each train/test group
    all_results = defaultdict(list)
    training_percents = [0.1, 0.3, 0.5, 0.7, 0.9]
    for train_percent in training_percents:
        for shuf in shuffles:

```

In each shuffle, split data into two parts(training and testing) and use *LogisticRegression* to evaluate.

```

# ... shuffle to generate train/test set X_train/X_test, y_train/y_test

clf = TopKRanker(LogisticRegression())
clf.fit(X_train, y_train)

# find out how many labels should be predicted
top_k_list = list(map(int, y_test.sum(axis=1).T.tolist()[0]))
preds = clf.predict(X_test, top_k_list)
result = f1_score(y_test, preds, average="micro")
all_results[train_percent].append(result)

```

Node in graph may have multiple labels, so we conduct multilabel classification built from TopKRanker.



```

from sklearn.multiclass import OneVsRestClassifier

class TopKRanker(OneVsRestClassifier):
    def predict(self, X, top_k_list):
        assert X.shape[0] == len(top_k_list)
        probs = np.asarray(super(TopKRanker, self).predict_proba(X))
        all_labels = sp.lil_matrix(probs.shape)

        for i, k in enumerate(top_k_list):
            probs_ = probs[i, :]
            labels = self.classes_[probs_.argsort()[-k:]].tolist()
            for label in labels:
                all_labels[i, label] = 1
        return all_labels

```

Finally, we get the results of Micro-F1 score under different training ratio for different models on datasets.

Cogdl supports evaluating the trained embeddings ignoring the training process. With `-load-emb-path` set to the path of your result, Cogdl will skip the training and directly evaluate the embeddings.

The overall implementation of *UnsupervisedNodeClassification* is at ([https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/unsupervised\\_node\\_classification.py](https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/unsupervised_node_classification.py)).

To run *UnsupervisedNodeClassification*, we can use following instruction:

```

python scripts/train.py --task unsupervised_node_classification --dataset ppi_
↳wikipedia --model deepwalk prone -seed 0 1

```

Then We get experimental results like this:

Variant	Micro-F1 0.1	Micro-F1 0.3	Micro-F1 0.5	Micro-F1 0.7	Micro-F1 0.9
('ppi', 'deepwalk')	0.1547±0.0002	0.1846±0.0002	0.2033±0.0015	0.2161±0.0009	0.2243±0.0018
('ppi', 'prone')	0.1777±0.0016	0.2214±0.0020	0.2397±0.0015	0.2486±0.0022	0.2607±0.0096
('wikipedia', 'deepwalk')	0.4255±0.0027	0.4712±0.0005	0.4916±0.0011	0.5011±0.0017	0.5166±0.0043
('wikipedia', 'prone')	0.4834±0.0009	0.5320±0.0020	0.5504±0.0045	0.5586±0.0022	0.5686±0.0072

### 2.4.3 Supervised Graph Classification

In this section, we will introduce the implementation “Graph classification task”.

**\*\* Supervised Graph Classification Methods \*\***

Method	Node Feature	Kernel	Reproducibility
GIN			
DiffPool			
SortPool			
PATCH_SAN			
DGCNN			
SAGPool			

#### Task Design

1. Set up “SupervisedGraphClassification” class, which has two specific parameters.

- *degree-feature*: Use one-hot node degree as node feature, for datasets such as Imdb-binary and Imdb-multi, which don't have node features.
- *gamma*: Multiplicative factor of learning rate decay.
- *lr*: Learning rate.

2. Build dataset convert it to a list of *Data* defined in Cogdl. Specially, we reformat the data according to the input format of specific models. *generate\_data* is implemented to convert dataset.

```
dataset = build_dataset(args)
self.data = self.generate_data(dataset, args)

def generate_data(self, dataset, args):
    if "ModelNet" in str(type(dataset).__name__):
        train_set, test_set = dataset.get_all()
        args.num_features = 3
        return {"train": train_set, "test": test_set}
    else:
        datalist = []
        if isinstance(dataset[0], Data):
            return dataset
        for idata in dataset:
            data = Data()
            for key in idata.keys():
                data[key] = idata[key]
            datalist.append(data)

        if args.degree_feature:
            datalist = node_degree_as_feature(datalist)
            args.num_features = datalist[0].num_features
        return datalist
```

3. Then we build model and can run *train* to train the model.

```
def train(self):
    for epoch in epoch_iter:
        self._train_step()
        val_acc, val_loss = self._test_step(split="valid")
        # ...
    return dict(Acc=test_acc)

def _train_step(self):
    self.model.train()
    loss_n = 0
    for batch in self.train_loader:
        batch = batch.to(self.device)
        self.optimizer.zero_grad()
        output, loss = self.model(batch)
        loss_n += loss.item()
        loss.backward()
        self.optimizer.step()

def _test_step(self, split):
    """split in ['train', 'test', 'valid']"""
    # ...
    return acc, loss
```

The overall implementation of GraphClassification is at ([https://github.com/THUDDM/cogdl/blob/master/cogdl/tasks/graph\\_classification.py](https://github.com/THUDDM/cogdl/blob/master/cogdl/tasks/graph_classification.py)).

## Create a model

To create a model for task graph classification, the following functions have to be implemented.

1. *add\_args(parser)*: add necessary hyper-parameters used in model.

```
@staticmethod
def add_args(parser):
    parser.add_argument("--hidden-size", type=int, default=128)
    parser.add_argument("--num-layers", type=int, default=2)
    parser.add_argument("--lr", type=float, default=0.001)
    # ...
```

2. *build\_model\_from\_args(cls, args)*: this function is called in 'task' to build model.
3. *split\_dataset(cls, dataset, args)*: split train/validation/test data and return correspondent dataloader according to requirement of model.

```
def split_dataset(cls, dataset, args):
    random.shuffle(dataset)
    train_size = int(len(dataset) * args.train_ratio)
    test_size = int(len(dataset) * args.test_ratio)
    bs = args.batch_size
    train_loader = DataLoader(dataset[:train_size], batch_size=bs)
    test_loader = DataLoader(dataset[-test_size:], batch_size=bs)
    if args.train_ratio + args.test_ratio < 1:
        valid_loader = DataLoader(dataset[train_size:-test_size], batch_size=bs)
    else:
        valid_loader = test_loader
    return train_loader, valid_loader, test_loader
```

4. *forward*: forward propagation, and the return should be (predication, loss) or (prediction, None), respectively for training and test. Input parameters of *forward* is class *Batch*, which

```
def forward(self, batch):
    h = batch.x
    layer_rep = [h]
    for i in range(self.num_layers-1):
        h = self.gin_layers[i](h, batch.edge_index)
        h = self.batch_norm[i](h)
        h = F.relu(h)
        layer_rep.append(h)

    final_score = 0
    for i in range(self.num_layers):
        pooled = scatter_add(layer_rep[i], batch.batch, dim=0)
        final_score += self.dropout(self.linear_prediction[i](pooled))
    final_score = F.softmax(final_score, dim=-1)
    if batch.y is not None:
        loss = self.loss(final_score, batch.y)
        return final_score, loss
    return final_score, None
```

## Run

To run GraphClassification, we can use the following command:

```
python scripts/train.py --task graph_classification --dataset proteins --model gin_
↳diffpool sortpool dgcnn --seed 0 1
```

Then We get experimental results like this:

Variants	Acc
('proteins', 'gin')	0.7286±0.0598
('proteins', 'diffpool')	0.7530±0.0589
('proteins', 'sortpool')	0.7411±0.0269
('proteins', 'dgcnn')	0.6677±0.0355
('proteins', 'patchy_san')	0.7550±0.0812

## 2.4.4 Unsupervised Graph Classification

In this section, we will introduce the implementation “Unsupervised graph classification task”.

### Unsupervised Graph Classification Methods

Method	Node Feature	Kernel	Reproducibility
InfoGraph			
DGK			
Graph2Vec			
HGP_SL			

### Task Design

- Set up “UnsupervisedGraphClassification” class, which has two specific parameters.
  - num-shuffle* : Shuffle times in classifier
  - degree-feature*: Use one-hot node degree as node feature, for datasets such as Imdb-binary and Imdb-multi, which don't have node features.
  - lr*: learning

```
@register_task("unsupervised_graph_classification")
class UnsupervisedGraphClassification(BaseTask):
    r"""Unsupervised graph classification"""
    @staticmethod
    def add_args(parser):
        """Add task-specific arguments to the parser."""
        # fmt: off
        parser.add_argument("--num-shuffle", type=int, default=10)
        parser.add_argument("--degree-feature", dest="degree_feature", action="store_
→true")
        parser.add_argument("--lr", type=float, default=0.001)
        # fmt: on
    def __init__(self, args):
        # ...
```

- Build dataset and convert it to a list of *Data* defined in Cogdl.

```
dataset = build_dataset(args)
self.label = np.array([data.y for data in dataset])
self.data = [
    Data(x=data.x, y=data.y, edge_index=data.edge_index, edge_attr=data.edge_attr,
        pos=data.pos).apply(lambda x:x.to(self.device))
    for data in dataset
]
```

- Then we build model and can run *train* to train the model and obtain graph representation. In this part, the training process of shallow models and deep models are implemented separately.

```
self.model = build_model(args)
self.model = self.model.to(self.device)

def train(self):
    if self.use_nn:
        # deep neural network models
        epoch_iter = tqdm(range(self.epoch))
        for epoch in epoch_iter:
            loss_n = 0
            for batch in self.data_loader:
                batch = batch.to(self.device)
                predict, loss = self.model(batch.x, batch.edge_index, batch.batch)
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()
                loss_n += loss.item()
            # ...
    else:
        # shallow models
        prediction, loss = self.model(self.data)
        label = self.label
```

- When graph representation is obtained, we evaluate the embedding with *SVM* via running *num\_shuffle* times under different training ratio. You can also call *save\_emb* to save the embedding.

```
return self._evaluate(prediction, label)
def _evaluate(self, embedding, labels):
    # ...
    for training_percent in training_percents:
        for shuf in shuffles:
            # ...
            clf = SVC()
            clf.fit(X_train, y_train)
            preds = clf.predict(X_test)
            # ...
```

The overall implementation of `UnsupervisedGraphClassification` is at ([https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/unsupervised\\_graph\\_classification.py](https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/unsupervised_graph_classification.py)).

### Create a model

To create a model for task unsupervised graph classification, the following functions have to be implemented.

- add\_args(parser)*: add necessary hyper-parameters used in model.

```
@staticmethod
def add_args(parser):
    parser.add_argument("--hidden-size", type=int, default=128)
    parser.add_argument("--nn", type=bool, default=False)
    parser.add_argument("--lr", type=float, default=0.001)
    # ...
```

- build\_model\_from\_args(cls, args)*: this function is called in ‘task’ to build model.
- forward*: For shallow models, this function runs as training process of model and will be called only once; For deep neural network models, this function is actually the forward propagation process and will be called many times.

```
# shallow model
def forward(self, graphs):
    # ...
    self.model = Doc2Vec(
        self.doc_collections,
        ...
    )
    vectors = np.array([self.model["g_"+str(i)] for i in range(len(graphs))])
    return vectors, None
```

## Run

To run UnsupervisedGraphClassification, we can use the following command:

```
python scripts/train.py --task unsupervised_graph_classification --dataset proteins --
↪model dgk graph2vec
```

Then we get experimental results like this:

Variant	Acc
('proteins', 'dgk')	0.7259±0.0118
('proteins', 'graph2vec')	0.7330±0.0043
('proteins', 'infograph')	0.7393±0.0070

## 2.4.5 Link Prediction

In this tutorial, we will introduce a important link prediction. Overall speaking, the link prediction in CogDL can be divided into 3 types.

1. Network embeddings based link prediction(*HomoLinkPrediction*). All unsupervised network embedding methods supports this task for homogenous graphs without node features.
2. Knowledge graph completion(*KGLinkPrediction* and *TripleLinkPrediction*), including knowledge embedding methods(TransE, DistMult) and GNN base methods(RGCN and CompGCN).
3. GNN base homogenous graph link prediction(*GNNHomoLinkPrediction*). Theoretically, all GNN models works.

	Models
Network embeddings methods	DeepWalk, LINE, Node2Vec, ProNE NetMF, NetSMF, SDNE, Hope
Knowledge graph completion	TransE, DistMult, RotatE, RGCN, CompGCN
GNN methods	GCN and all the other GNN methods

To implement a new GNN model for link prediction, just implement *link\_prediction\_loss* in the model which accepting three parameters:

- Node features.
- Edge index.
- Labels. 0/1 for each item, indicating the edge exists in the graph or is a negative sample.

The overall implementation can be found at [https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/link\\_prediction.py](https://github.com/THUDM/cogdl/blob/master/cogdl/tasks/link_prediction.py)

## 2.4.6 Other Tasks

### Heterogeneous Graph Embedding Methods

Method	Multi-Node	Multi-Edge	Supervised	Attribute	MetaPath
GATNE					
Metapath2Vec					
PTE					
Hin2Vec					
GTN					
HAN					

### Attributed Graph Clustering

Method	Content	Spectral
kmeans		
spectral		
PRONE		
NetMF		
deepwalk		
line		
AGC		
DAEGC		

### Pretrained Graph Models

- STPGNN: Strategies for pretraining graph neural networks
- GCC: GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training

## 2.4.7 Create new tasks

You can build a new task in the CogDL. The BaseTask class are:

```
class BaseTask(object):
    @staticmethod
    def add_args(parser):
        """Add task-specific arguments to the parser."""
        pass

    def __init__(self, args):
        pass

    def train(self, num_epoch):
        raise NotImplementedError
```

You can create a subclass to implement 'train' method like CommunityDetection, which get representation of each node and apply clustering algorithm (K-means) to evaluate.

```
@register_task("community_detection")
class CommunityDetection(BaseTask):
    """Community Detection task."""
```

(continues on next page)

(continued from previous page)

```

@staticmethod
def add_args(parser):
    """Add task-specific arguments to the parser."""
    parser.add_argument("--hidden-size", type=int, default=128)
    parser.add_argument("--num-shuffle", type=int, default=5)

def __init__(self, args):
    super(CommunityDetection, self).__init__(args)
    dataset = build_dataset(args)
    self.data = dataset[0]

    self.num_nodes, self.num_classes = self.data.y.shape
    self.label = np.argmax(self.data.y, axis=1)
    self.model = build_model(args)
    self.hidden_size = args.hidden_size
    self.num_shuffle = args.num_shuffle

def train(self):
    G = nx.Graph()
    G.add_edges_from(self.data.edge_index.t().tolist())
    embeddings = self.model.train(G)

    clusters = [30, 50, 70]
    all_results = defaultdict(list)
    for num_cluster in clusters:
        for _ in range(self.num_shuffle):
            model = KMeans(n_clusters=num_cluster).fit(embeddings)
            nmi_score = normalized_mutual_info_score(self.label, model.labels_)
            all_results[num_cluster].append(nmi_score)

    return dict(
        (
            f"normalized_mutual_info_score {num_cluster}",
            sum(all_results[num_cluster]) / len(all_results[num_cluster]),
        )
        for num_cluster in sorted(all_results.keys())
    )

```

After creating your own task, you could run the task on different models and dataset. You can use ‘build\_model’, ‘build\_dataset’, ‘build\_task’ method to build them with corresponding hyper-parameters.

```

from cogdl.tasks import build_task
from cogdl.datasets import build_dataset
from cogdl.models import build_model
from cogdl.utils import build_args_from_dict

def run_deepwalk_ppi():
    default_dict = {'hidden_size': 64, 'num_shuffle': 1, 'cpu': True}
    args = build_args_from_dict(default_dict)

    # model, dataset and task parameters
    args.model = 'spectral'
    args.dataset = 'ppi'
    args.task = 'community_detection'

    # build model, dataset and task

```

(continues on next page)



(continued from previous page)

```

dataset = build_dataset(args)
model = build_model(args)
task = build_task(args)

# train model and get evaluate results
ret = task.train()
print(ret)

```

## 2.5 Trainer

In this section, we will introduce how to implement a specific *Trainer* for a model.

In previous section, we introduce the implementation of different *tasks*. But the training paradigm varies and is incompatible with the defined training process in some cases. Therefore, *CogDL* provides *Trainer* to customize the training and inference mode. Take *NeighborSamplingTrainer* as the example, this section will show how to define a trainer.

### Design

1. A self-defined trainer should inherits *BaseTrainer* and must implement function *fit* to define the training and evaluating process. Necessary parameters for training need to be added to the *add\_args* in models and can be obtained here in *\_\_init\_\_*.

```

class NeighborSamplingTrainer(BaseTrainer):
    def __init__(self, args):
        # ... get necessary parameters from args

    def fit(self, model, dataset):
        # ... implement the training and evaluation

    @classmethod
    def build_trainer_from_args(cls, args):
        return cls(args)

```

2. All training and evaluating process, including data preprocessing and defining optimizer, should be implemented in *fit*. In other words, given the model and dataset, the rest is up to you. *fit* accepts two parameters: model and dataset, which usually are in cpu. You need to move them to cuda if you want to train on GPU.

```

def fit(self, model, dataset):
    self.data = dataset[0]

    # preprocess data
    self.train_loader = NeighborSampler(
        data=self.data,
        mask=self.data.train_mask,
        sizes=self.sample_size,
        batch_size=self.batch_size,
        num_workers=self.num_workers,
        shuffle=True,
    )
    self.test_loader = NeighborSampler(
        data=self.data, mask=None, sizes=[-1], batch_size=self.batch_size,
        ↪shuffle=False
    )

```

(continues on next page)

(continued from previous page)

```

# move model to GPU
self.model = model.to(self.device)

# define optimizer
self.optimizer = torch.optim.Adam(self.model.parameters(), lr=self.lr, weight_
↪decay=self.weight_decay)
# training
best_model = self.train()
self.model = best_model
# evaluation
acc, loss = self._test_step()
return dict(Acc=acc["test"], ValAcc=acc["val"])

```

3. To make the training of a model use the trainer, we should assign the trainer to the model. In Cogdl, a model must implement `get_trainer` as static method if it has a customized training process. GraphSAGE depends on *Neighbor-SamplingTrainer*, so the following codes should exists in the implementation.

```

@staticmethod
def get_trainer(taskType, args):
    return NeighborSamplingTrainer

```

The details of training and evaluating are similar to the implementation in *Tasks*. The overall implementation of trainers is at <https://github.com/THUDM/cogdl/tree/master/cogdl/trainers>

## 2.6 Model

In this section, we will create a spectral clustering model, which is a very simple graph embedding algorithm. We name it `spectral.py` and put it in `cogdl/models/emb` directory.

First we import necessary library like `numpy`, `scipy`, `networkx`, `sklearn`, we also import API like ‘`BaseModel`’ and ‘`register_model`’ from `cogdl/models/` to build our new model:

```

import numpy as np
import networkx as nx
import scipy.sparse as sp
from sklearn import preprocessing
from .. import BaseModel, register_model

```

Then we use function decorator to declare new model for CogDL

```

@register_model('spectral')
class Spectral(BaseModel):
    (...)

```

We have to implement method ‘`build_model_from_args`’ in `spectral.py`. If it need more parameters to train, we can use ‘`add_args`’ to add model-specific arguments.

```

@staticmethod
def add_args(parser):
    """Add model-specific arguments to the parser."""
    pass

@classmethod
def build_model_from_args(cls, args):

```

(continues on next page)

(continued from previous page)

```

    return cls(args.hidden_size)

def __init__(self, dimension):
    super(Spectral, self).__init__()
    self.dimension = dimension

```

Each new model should provide a ‘train’ method to obtain representation.

```

def train(self, G):
    matrix = nx.normalized_laplacian_matrix(G).todense()
    matrix = np.eye(matrix.shape[0]) - np.asarray(matrix)
    ut, s, _ = sp.linalg.svds(matrix, self.dimension)
    emb_matrix = ut * np.sqrt(s)
    emb_matrix = preprocessing.normalize(emb_matrix, "l2")
    return emb_matrix

```

All implemented models are at <https://github.com/THUDM/cogdl/tree/master/cogdl/models>.

## 2.7 Dataset

In order to add a dataset into CogDL, you should know your dataset’s format. We have provided several graph format like edgelist, matlab\_matrix and pyg. If the format of your dataset is the same as the *ppi* dataset, which contains two matrices: *network* and *group*, you can register your dataset directly use the following code.

```

@register_dataset("ppi")
class PPIDataset(MatlabMatrix):
    def __init__(self):
        dataset, filename = "ppi", "Homo_sapiens"
        url = "http://snap.stanford.edu/node2vec/"
        path = osp.join("data", dataset)
        super(PPIDataset, self).__init__(path, filename, url)

```

You should declare the name of the dataset, the name of file and the url, where our script can download resource. More implemented datasets are at <https://github.com/THUDM/cogdl/tree/master/cogdl/datasets>.

## 2.8 data

```

class cogdl.data.Graph(x=None, y=None, **kwargs)
    Bases: cogdl.data.data.BaseGraph
    add_remaining_self_loops()
    clone()
    col_indices
    col_norm()
    csr_subgraph(node_idx)
    degrees()
    edge_attr
    edge_index

```

**edge\_subgraph** (*edge\_idx, require\_idx=True*)

**edge\_types**

**edge\_weight**  
Return actual edge\_weight

**eval** ()

**static from\_dict** (*dictionary*)  
Creates a data object from a python dictionary.

**static from\_pyg\_data** (*data*)

**in\_norm**

**is\_inductive** ()

**is\_symmetric** ()

**keys**  
Returns all names of graph attributes.

**local\_graph** ()

**mask2nid** (*split*)

**normalize** (*key='sym'*)

**num\_classes**

**num\_edges**  
Returns the number of edges in the graph.

**num\_features**  
Returns the number of features per node in the graph.

**num\_nodes**

**out\_norm**

**raw\_edge\_weight**  
Return edge\_weight without `__in_norm__` and `__out_norm__`, only used for SpMM

**remove\_self\_loops** ()

**row\_indptr**

**row\_norm** ()

**sample\_adj** (*batch, size=-1, replace=True*)

**set\_asymmetric** ()

**set\_symmetric** ()

**subgraph** (*node\_idx*)

**sym\_norm** ()

**test\_nid**

**to\_scipy\_csr** ()

**train** ()

**train\_nid**

**val\_nid**

```
class cogdl.data.Adjacency (row=None, col=None, row_ptr=None, weight=None, attr=None,  

                           num_nodes=None, types=None, **kwargs)
```

```
Bases: cogdl.data.data.BaseGraph
```

```
add_remaining_self_loops ()
```

```
clone ()
```

```
col_norm ()
```

```
convert_csr ()
```

```
degrees
```

```
device
```

```
edge_index
```

```
static from_dict (dictionary)
```

```
Creates a data object from a python dictionary.
```

```
generate_normalization (norm='sym')
```

```
get_weight (indicator=None)
```

```
If indicator is not None, the normalization will not be implemented
```

```
is_symmetric ()
```

```
keys
```

```
Returns all names of graph attributes.
```

```
normalize_adj (norm='sym')
```

```
num_edges
```

```
num_nodes
```

```
random_walk (start, length=1, restart_p=0.0)
```

```
random_walk_with_restart (start, length, restart_p)
```

```
remove_self_loops ()
```

```
row_indptr
```

```
row_norm ()
```

```
set_symmetric (val)
```

```
set_weight (weight)
```

```
sym_norm ()
```

```
to_scipy_csr ()
```

```
class cogdl.data.Batch (batch=None, **kwargs)
```

```
Bases: cogdl.data.data.Graph
```

A plain old python object modeling a batch of graphs as one big (dicconnected) graph. With `cogdl.data.Data` being the base class, all its methods can also be used here. In addition, single graphs can be reconstructed via the assignment vector `batch`, which maps each node to its respective graph identifier.

```
cumsum (key, item)
```

```
If True, the attribute key with content item should be added up cumulatively before concatenated together.
```

---

**Note:** This method is for internal use only, and should only be overridden if the batch concatenation process is corrupted for a specific data attribute.

---

**static from\_data\_list** (*data\_list*)

Constructs a batch object from a python list holding `cogdl.data.Data` objects. The assignment vector `batch` is created on the fly. Additionally, creates assignment batch vectors for each key in `follow_batch`.

**num\_graphs**

Returns the number of graphs in the batch.

**class** `cogdl.data.Dataset` (*root, transform=None, pre\_transform=None, pre\_filter=None*)

Bases: `torch.utils.data.dataset.Dataset`

Dataset base class for creating graph datasets. See [here](#) for the accompanying tutorial.

**Args:** `root` (string): Root directory where the dataset should be saved. `transform` (callable, optional): A function/transform that takes in an

`cogdl.data.Data` object and returns a transformed version. The data object will be transformed before every access. (default: `None`)

**pre\_transform** (callable, optional): A function/transform that takes in an `cogdl.data.Data` object and returns a transformed version. The data object will be transformed before being saved to disk. (default: `None`)

**pre\_filter** (callable, optional): A function that takes in an `cogdl.data.Data` object and returns a boolean value, indicating whether the data object should be included in the final dataset. (default: `None`)

**static add\_args** (*parser*)

Add dataset-specific arguments to the parser.

**download** ()

Downloads the dataset to the `self.raw_dir` folder.

**edge\_attr\_size**

**get** (*idx*)

Gets the data object at index `idx`.

**get\_evaluator** ()

**get\_loss\_fn** ()

**num\_classes**

The number of classes in the dataset.

**num\_features**

Returns the number of features per node in the graph.

**process** ()

Processes the dataset to the `self.processed_dir` folder.

**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**processed\_paths**

The filepaths to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

**raw\_paths**

The filepaths to find in order to skip the download.

**class** `cogdl.data.DataLoader` (*dataset, batch\_size=1, shuffle=True, \*\*kwargs*)

Bases: `torch.utils.data.dataloader.DataLoader`

Data loader which merges data objects from a `cogdl.data.dataset` to a mini-batch.

**Args:** `dataset` (Dataset): The dataset from which to load the data. `batch_size` (int, optional): How many samples per batch to load.

(default: 1)

**shuffle** (bool, optional): If set to **True**, the data will be reshuffled at every epoch (default: **True**)

**static collate\_fn** (*batch*)

**class** `cogdl.data.MultiGraphDataset` (*root=None, transform=None, pre\_transform=None, pre\_filter=None*)

Bases: `cogdl.data.dataset.Dataset`

**static from\_data\_list** (*data\_list*)

**get** (*idx*)

Gets the data object at index `idx`.

**len** ()

**num\_classes**

The number of classes in the dataset.

**num\_features**

Returns the number of features per node in the graph.

## 2.9 datasets

### 2.9.1 GATNE dataset

**class** `cogdl.datasets.gatne.AmazonDataset` (*data\_path='data'*)

Bases: `cogdl.datasets.gatne.GatneDataset`

**class** `cogdl.datasets.gatne.GatneDataset` (*root, name*)

Bases: `cogdl.data.dataset.Dataset`

The network datasets “Amazon”, “Twitter” and “YouTube” from the “Representation Learning for Attributed Multiplex Heterogeneous Network” paper.

**Args:** `root` (string): Root directory where the dataset should be saved. `name` (string): The name of the dataset (“Amazon”,

“Twitter”, “YouTube”).

**download** ()

Downloads the dataset to the `self.raw_dir` folder.

**get** (*idx*)

Gets the data object at index `idx`.

**process ()**

Processes the dataset to the `self.processed_dir` folder.

**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

**url = 'https://github.com/THUDM/GATNE/raw/master/data'**

**class cogdl.datasets.gatne.TwitterDataset (data\_path='data')**

Bases: `cogdl.datasets.gatne.GatneDataset`

**class cogdl.datasets.gatne.YouTubeDataset (data\_path='data')**

Bases: `cogdl.datasets.gatne.GatneDataset`

`cogdl.datasets.gatne.read_gatne_data (folder)`

## 2.9.2 GCC dataset

**class cogdl.datasets.gcc\_data.Edgelist (root, name)**

Bases: `cogdl.data.dataset.Dataset`

**download ()**

Downloads the dataset to the `self.raw_dir` folder.

**get (idx)**

Gets the data object at index `idx`.

**num\_classes**

The number of classes in the dataset.

**process ()**

Processes the dataset to the `self.processed_dir` folder.

**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

**url = 'https://github.com/cenyk1230/gcc-data/raw/master'**

**class cogdl.datasets.gcc\_data.GCCDataset (root, name)**

Bases: `cogdl.data.dataset.Dataset`

**download ()**

Downloads the dataset to the `self.raw_dir` folder.

**get (idx)**

Gets the data object at index `idx`.

**preprocess (root, name)**

**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

**url = 'https://github.com/cenyk1230/gcc-data/raw/master'**



```

class cogdl.datasets.gcc_data.KDD_ICDM_GCCDataset (data_path='data')
    Bases: cogdl.datasets.gcc_data.GCCDataset

class cogdl.datasets.gcc_data.SIGIR_CIKM_GCCDataset (data_path='data')
    Bases: cogdl.datasets.gcc_data.GCCDataset

class cogdl.datasets.gcc_data.SIGMOD_ICDE_GCCDataset (data_path='data')
    Bases: cogdl.datasets.gcc_data.GCCDataset

class cogdl.datasets.gcc_data.USAAirportDataset (data_path='data')
    Bases: cogdl.datasets.gcc_data.Edgelist

```

### 2.9.3 GTN dataset

```

class cogdl.datasets.gtn_data.ACM_GTNDataset (data_path='data')
    Bases: cogdl.datasets.gtn_data.GTNDataset

class cogdl.datasets.gtn_data.DBLP_GTNDataset (data_path='data')
    Bases: cogdl.datasets.gtn_data.GTNDataset

class cogdl.datasets.gtn_data.GTNDataset (root, name)
    Bases: cogdl.data.dataset.Dataset

```

The network datasets “ACM”, “DBLP” and “IMDB” from the “Graph Transformer Networks” paper.

**Args:** *root* (string): Root directory where the dataset should be saved. *name* (string): The name of the dataset ("gtn-acm", "gtn-dblp", "gtn-imdb").

**apply\_to\_device** (*device*)

**download** ()

Downloads the dataset to the `self.raw_dir` folder.

**get** (*idx*)

Gets the data object at index `idx`.

**num\_classes**

The number of classes in the dataset.

**process** ()

Processes the dataset to the `self.processed_dir` folder.

**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

**read\_gtn\_data** (*folder*)

```

class cogdl.datasets.gtn_data.IMDB_GTNDataset (data_path='data')
    Bases: cogdl.datasets.gtn_data.GTNDataset

```

### 2.9.4 HAN dataset

```

class cogdl.datasets.han_data.ACM_HANDataset (data_path='data')
    Bases: cogdl.datasets.han_data.HANDataset

```

**class** cogdl.datasets.han\_data.DBLP\_HANDataset (*data\_path='data'*)  
Bases: *cogdl.datasets.han\_data.HANDataset*

**class** cogdl.datasets.han\_data.HANDataset (*root, name*)  
Bases: *cogdl.data.dataset.Dataset*

The network datasets “ACM”, “DBLP” and “IMDB” from the “Heterogeneous Graph Attention Network” paper.

**Args:** *root* (string): Root directory where the dataset should be saved. *name* (string): The name of the dataset ("han-acm", "han-dblp", "han-imdb").

**apply\_to\_device** (*device*)

**download** ()  
Downloads the dataset to the *self.raw\_dir* folder.

**get** (*idx*)  
Gets the data object at index *idx*.

**num\_classes**  
The number of classes in the dataset.

**process** ()  
Processes the dataset to the *self.processed\_dir* folder.

**processed\_file\_names**  
The name of the files to find in the *self.processed\_dir* folder in order to skip the processing.

**raw\_file\_names**  
The name of the files to find in the *self.raw\_dir* folder in order to skip the download.

**read\_gtn\_data** (*folder*)

**class** cogdl.datasets.han\_data.IMDB\_HANDataset (*data\_path='data'*)  
Bases: *cogdl.datasets.han\_data.HANDataset*

*cogdl.datasets.han\_data.sample\_mask* (*idx, length*)  
Create mask.

## 2.9.5 KG dataset

**class** cogdl.datasets.kg\_data.BidirectionalOneShotIterator (*dataloader\_head, dataloader\_tail*)  
Bases: *object*

**static one\_shot\_iterator** (*dataloader*)  
Transform a PyTorch Dataloader into python iterator

**class** cogdl.datasets.kg\_data.FB13Datset (*data\_path='data'*)  
Bases: *cogdl.datasets.kg\_data.KnowledgeGraphDataset*

**class** cogdl.datasets.kg\_data.FB13SDatset (*data\_path='data'*)  
Bases: *cogdl.datasets.kg\_data.KnowledgeGraphDataset*

**class** cogdl.datasets.kg\_data.FB15k237Datset (*data\_path='data'*)  
Bases: *cogdl.datasets.kg\_data.KnowledgeGraphDataset*

**class** cogdl.datasets.kg\_data.FB15kDatset (*data\_path='data'*)  
Bases: *cogdl.datasets.kg\_data.KnowledgeGraphDataset*

```

class cogdl.datasets.kg_data.KnowledgeGraphDataset (root, name)
    Bases: cogdl.data.dataset.Dataset

    download()
        Downloads the dataset to the self.raw_dir folder.

    get (idx)
        Gets the data object at index idx.

    num_entities
    num_relations

    process()
        Processes the dataset to the self.processed_dir folder.

    processed_file_names
        The name of the files to find in the self.processed_dir folder in order to skip the processing.

    raw_file_names
        The name of the files to find in the self.raw_dir folder in order to skip the download.

    test_start_idx
    train_start_idx

    url = 'https://cloud.tsinghua.edu.cn/d/b567292338f2488699b7/files/?p=%2F{}%2F{}&dl=1'

    valid_start_idx

class cogdl.datasets.kg_data.TestDataset (triples, all_true_triples, nentity, nrelation, mode)
    Bases: torch.utils.data.dataset.Dataset

    static collate_fn (data)

class cogdl.datasets.kg_data.TrainDataset (triples, nentity, nrelation, negative_sample_size,
                                           mode)
    Bases: torch.utils.data.dataset.Dataset

    static collate_fn (data)

    static count_frequency (triples, start=4)
        Get frequency of a partial triple like (head, relation) or (relation, tail) The frequency will be used for
        subsampling like word2vec

    static get_true_head_and_tail (triples)
        Build a dictionary of true triples that will be used to filter these true triples for negative sampling

class cogdl.datasets.kg_data.WN18Datset (data_path='data')
    Bases: cogdl.datasets.kg_data.KnowledgeGraphDataset

class cogdl.datasets.kg_data.WN18RRDataset (data_path='data')
    Bases: cogdl.datasets.kg_data.KnowledgeGraphDataset

cogdl.datasets.kg_data.read_triplet_data (folder)

```

## 2.9.6 Matlab matrix dataset

```

class cogdl.datasets.matlab_matrix.BlogcatalogDataset (data_path='data')
    Bases: cogdl.datasets.matlab_matrix.MatlabMatrix

class cogdl.datasets.matlab_matrix.DblpNEDataset (data_path='data')
    Bases: cogdl.datasets.matlab_matrix.NetworkEmbeddingCMTYDataset

```

```
class cogdl.datasets.matlab_matrix.FlickrDataset (data_path='data')
    Bases: cogdl.datasets.matlab_matrix.MatlabMatrix

class cogdl.datasets.matlab_matrix.MatlabMatrix (root, name, url)
    Bases: cogdl.data.dataset.Dataset

    networks from the http://leitang.net/code/social-dimension/data/ or http://snap.stanford.edu/node2vec/

Args: root (string): Root directory where the dataset should be saved. name (string): The name of the dataset
    ("Blogcatalog").

download ()
    Downloads the dataset to the self.raw_dir folder.

get (idx)
    Gets the data object at index idx.

num_classes
    The number of classes in the dataset.

num_nodes

process ()
    Processes the dataset to the self.processed_dir folder.

processed_file_names
    The name of the files to find in the self.processed_dir folder in order to skip the processing.

raw_file_names
    The name of the files to find in the self.raw_dir folder in order to skip the download.

class cogdl.datasets.matlab_matrix.NetworkEmbeddingCMTYDataset (root, name, url)
    Bases: cogdl.data.dataset.Dataset

download ()
    Downloads the dataset to the self.raw_dir folder.

get (idx)
    Gets the data object at index idx.

num_classes
    The number of classes in the dataset.

num_nodes

process ()
    Processes the dataset to the self.processed_dir folder.

processed_file_names
    The name of the files to find in the self.processed_dir folder in order to skip the processing.

raw_file_names
    The name of the files to find in the self.raw_dir folder in order to skip the download.

class cogdl.datasets.matlab_matrix.PPIDataset (data_path='data')
    Bases: cogdl.datasets.matlab_matrix.MatlabMatrix

class cogdl.datasets.matlab_matrix.WikipediaDataset (data_path='data')
    Bases: cogdl.datasets.matlab_matrix.MatlabMatrix

class cogdl.datasets.matlab_matrix.YoutubeNEDataset (data_path='data')
    Bases: cogdl.datasets.matlab_matrix.NetworkEmbeddingCMTYDataset
```

## 2.9.7 PyG OGB dataset

```

class cogdl.datasets.ogb.MAGDataset (data_path='data')
    Bases: cogdl.data.dataset.Dataset

    get (idx)
        Gets the data object at index idx.

    get_evaluator ()

    num_authors

    num_edge_types

    num_field_of_study

    num_institutions

    num_node_types

    num_papers

    process ()
        Processes the dataset to the self.processed_dir folder.

    processed_file_names
        The name of the files to find in the self.processed_dir folder in order to skip the processing.

class cogdl.datasets.ogb.OGBArxivDataset (data_path='data')
    Bases: cogdl.datasets.ogb.OGBNDataset

    get_evaluator ()

class cogdl.datasets.ogb.OGBCodeDataset (data_path='data')
    Bases: cogdl.datasets.ogb.OGBGDataset

class cogdl.datasets.ogb.OGBGDataset (root, name)
    Bases: cogdl.data.dataset.Dataset

    get (idx)
        Gets the data object at index idx.

    get_loader (args)

    get_subset (subset)

    num_classes
        The number of classes in the dataset.

class cogdl.datasets.ogb.OGBMolbaceDataset (data_path='data')
    Bases: cogdl.datasets.ogb.OGBGDataset

class cogdl.datasets.ogb.OGBMolhivDataset (data_path='data')
    Bases: cogdl.datasets.ogb.OGBGDataset

class cogdl.datasets.ogb.OGBMolpcbaDataset (data_path='data')
    Bases: cogdl.datasets.ogb.OGBGDataset

class cogdl.datasets.ogb.OGBNDataset (root, name, transform=None)
    Bases: cogdl.data.dataset.Dataset

    get (idx)
        Gets the data object at index idx.

    get_evaluator ()

```

```
    get_loss_fn()

    process()
        Processes the dataset to the self.processed_dir folder.

    processed_file_names
        The name of the files to find in the self.processed_dir folder in order to skip the processing.

class cogdl.datasets.ogb.OGBPapers100MDataSet (data_path='data')
    Bases: cogdl.datasets.ogb.OGBNDataSet

class cogdl.datasets.ogb.OGBPpaDataSet
    Bases: cogdl.datasets.ogb.OGBGDataSet

class cogdl.datasets.ogb.OGBProductsDataSet (data_path='data')
    Bases: cogdl.datasets.ogb.OGBNDataSet

class cogdl.datasets.ogb.OGBProteinsDataSet (data_path='data')
    Bases: cogdl.datasets.ogb.OGBNDataSet

    edge_attr_size

    get_evaluator()

    get_loss_fn()

    process()
        Processes the dataset to the self.processed_dir folder.
```

## 2.9.8 PyG strategies dataset

This file is borrowed from <https://github.com/snap-stanford/pretrain-gnns/>

```
class cogdl.datasets.strategies_data.BACEDataset (transform=None,
                                                  pre_transform=None,
                                                  pre_filter=None, empty=False,
                                                  data_path='data')

    Bases: cogdl.data.dataset.MultiGraphDataset

    download()
        Downloads the dataset to the self.raw_dir folder.

    process()
        Processes the dataset to the self.processed_dir folder.

    processed_file_names
        The name of the files to find in the self.processed_dir folder in order to skip the processing.

    raw_file_names
        The name of the files to find in the self.raw_dir folder in order to skip the download.

class cogdl.datasets.strategies_data.BBBPDataSet (transform=None,
                                                  pre_transform=None,
                                                  pre_filter=None, empty=False,
                                                  data_path='data')

    Bases: cogdl.data.dataset.MultiGraphDataset

    download()
        Downloads the dataset to the self.raw_dir folder.

    process()
        Processes the dataset to the self.processed_dir folder.
```

**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

**class** `cogdl.datasets.strategies_data.BatchAE` (*batch=None, \*\*kwargs*)

Bases: `cogdl.data.data.Graph`

**cat\_dim** (*key*)

Returns the dimension in which the attribute `key` with content value gets concatenated when creating batches.

---

**Note:** This method is for internal use only, and should only be overridden if the batch concatenation process is corrupted for a specific data attribute.

---

**static from\_data\_list** (*data\_list*)

Constructs a batch object from a python list holding `torch_geometric.data.Data` objects. The assignment vector `batch` is created on the fly.

**num\_graphs**

Returns the number of graphs in the batch.

**class** `cogdl.datasets.strategies_data.BatchMasking` (*batch=None, \*\*kwargs*)

Bases: `cogdl.data.data.Graph`

**cumsum** (*key, item*)

If `True`, the attribute `key` with content `item` should be added up cumulatively before concatenated together. .. note:

This method **is for** internal use only, **and** should only be overridden **if** the batch concatenation process **is** corrupted **for** a specific data attribute.

**static from\_data\_list** (*data\_list*)

Constructs a batch object from a python list holding `torch_geometric.data.Data` objects. The assignment vector `batch` is created on the fly.

**num\_graphs**

Returns the number of graphs in the batch.

**class** `cogdl.datasets.strategies_data.BatchSubstructContext` (*batch=None, \*\*kwargs*)

Bases: `cogdl.data.data.Graph`

**cat\_dim** (*key*)

Returns the dimension in which the attribute `key` with content value gets concatenated when creating batches.

---

**Note:** This method is for internal use only, and should only be overridden if the batch concatenation process is corrupted for a specific data attribute.

---

**cumsum** (*key, item*)

If `True`, the attribute `key` with content `item` should be added up cumulatively before concatenated together. .. note:

This method **is for** internal use only, **and** should only be overridden **if** the batch concatenation process **is** corrupted **for** a specific data attribute.

**static from\_data\_list** (*data\_list*)

Constructs a batch object from a python list holding `torch_geometric.data.Data` objects. The assignment vector batch is created on the fly.

**num\_graphs**

Returns the number of graphs in the batch.

```
class cogdl.datasets.strategies_data.BioDataset (data_type='unsupervised',
                                             empty=False, transform=None,
                                             pre_transform=None,
                                             pre_filter=None, data_path='data')
```

Bases: `cogdl.data.dataset.MultiGraphDataset`

**download** ()

Downloads the dataset to the `self.raw_dir` folder.

**process** ()

Processes the dataset to the `self.processed_dir` folder.

**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

```
class cogdl.datasets.strategies_data.ChemExtractSubstructureContextPair (k,
                                                                           l1,
                                                                           l2)
```

Bases: `object`

```
class cogdl.datasets.strategies_data.DataLoaderAE (dataset, batch_size=1, shuffle=True, **kwargs)
```

Bases: `torch.utils.data.data_loader.DataLoader`

```
class cogdl.datasets.strategies_data.DataLoaderSubstructContext (dataset,
                                                                    batch_size=1,
                                                                    shuffle=True,
                                                                    **kwargs)
```

Bases: `torch.utils.data.data_loader.DataLoader`

```
class cogdl.datasets.strategies_data.ExtractSubstructureContextPair (l1, center=True)
```

Bases: `object`

```
class cogdl.datasets.strategies_data.MoleculeDataset (data_type='unsupervised',
                                                         transform=None,
                                                         pre_transform=None,
                                                         pre_filter=None, empty=False,
                                                         data_path='data')
```

Bases: `cogdl.data.dataset.MultiGraphDataset`

**download** ()

Downloads the dataset to the `self.raw_dir` folder.

**process** ()

Processes the dataset to the `self.processed_dir` folder.



**processed\_file\_names**

The name of the files to find in the `self.processed_dir` folder in order to skip the processing.

**raw\_file\_names**

The name of the files to find in the `self.raw_dir` folder in order to skip the download.

**class** `cogdl.datasets.strategies_data.NegativeEdge`

Bases: `object`

Borrowed from <https://github.com/snap-stanford/pretrain-gnns/>

**class** `cogdl.datasets.strategies_data.TestBioDataset` (*data\_type='unsupervised', root='testbio', transform=None, pre\_transform=None, pre\_filter=None*)

Bases: `cogdl.data.dataset.MultiGraphDataset`

**class** `cogdl.datasets.strategies_data.TestChemDataset` (*data\_type='unsupervised', root='testchem', transform=None, pre\_transform=None, pre\_filter=None*)

Bases: `cogdl.data.dataset.MultiGraphDataset`

`cogdl.datasets.strategies_data.build_batch` (*batch, data\_list, num\_nodes\_cum, num\_edges\_cum, keys*)

`cogdl.datasets.strategies_data.convert` (*data*)

`cogdl.datasets.strategies_data.graph_data_obj_to_nx` (*data*)

`cogdl.datasets.strategies_data.graph_data_obj_to_nx_simple` (*data*)

Converts graph Data object required by the pytorch geometric package to network x data object. NB: Uses simplified atom and bond features, and represent as indices. NB: possible issues with recapitulating relative stereochemistry since the edges in the nx object are unordered. :param data: pytorch geometric Data object :return: network x object

`cogdl.datasets.strategies_data.nx_to_graph_data_obj` (*g, center\_id, allowable\_features\_downstream=None, allowable\_features\_pretrain=None, node\_id\_to\_go\_labels=None*)

`cogdl.datasets.strategies_data.nx_to_graph_data_obj_simple` (*G*)

Converts nx graph to pytorch geometric Data object. Assume node indices are numbered from 0 to num\_nodes - 1. NB: Uses simplified atom and bond features, and represent as indices. NB: possible issues with recapitulating relative stereochemistry since the edges in the nx object are unordered. :param G: nx graph obj :return: pytorch geometric Data object

`cogdl.datasets.strategies_data.reset_idxes` (*G*)

Resets node indices such that they are numbered from 0 to num\_nodes - 1 :param G: :return: copy of G with relabelled node indices, mapping

## 2.9.9 TU dataset

**class** `cogdl.datasets.tu_data.CollabDataset` (*data\_path='data'*)

Bases: `cogdl.datasets.tu_data.TUDataset`

**class** `cogdl.datasets.tu_data.ENZYMES` (*data\_path='data'*)

Bases: `cogdl.datasets.tu_data.TUDataset`

```
class cogdl.datasets.tu_data.ImdbBinaryDataset (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.ImdbMultiDataset (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.MUTAGDataset (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.NCT109Dataset (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.NCT1Dataset (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.PTCMRDataset (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.ProteinsDataset (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.RedditBinary (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.RedditMulti12K (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.RedditMulti5K (data_path='data')
    Bases: cogdl.datasets.tu_data.TUDataSet

class cogdl.datasets.tu_data.TUDataSet (root, name)
    Bases: cogdl.data.dataset.MultiGraphDataset

    download ()
        Downloads the dataset to the self.raw_dir folder.

    num_classes
        The number of classes in the dataset.

    process ()
        Processes the dataset to the self.processed_dir folder.

    processed_file_names
        The name of the files to find in the self.processed_dir folder in order to skip the processing.

    raw_file_names
        The name of the files to find in the self.raw_dir folder in order to skip the download.

    url = 'https://www.chrsmrrs.com/graphkerneldatasets'

cogdl.datasets.tu_data.cat (seq)

cogdl.datasets.tu_data.coalesce (index, value, m, n)

cogdl.datasets.tu_data.normalize_feature (data)

cogdl.datasets.tu_data.num_edge_attributes (edge_attr=None)

cogdl.datasets.tu_data.num_edge_labels (edge_attr=None)

cogdl.datasets.tu_data.num_node_attributes (x=None)

cogdl.datasets.tu_data.num_node_labels (x=None)

cogdl.datasets.tu_data.parse_txt_array (src, sep=None, start=0, end=None, dtype=None, de-
vice=None)
```

```
cogdl.datasets.tu_data.read_file(folder, prefix, name, dtype=None)
cogdl.datasets.tu_data.read_tu_data(folder, prefix)
cogdl.datasets.tu_data.read_txt_array(path, sep=None, start=0, end=None, dtype=None, device=None)
cogdl.datasets.tu_data.segment(src, indptr)
cogdl.datasets.tu_data.split(data, batch)
```

## 2.9.10 Module contents

```
cogdl.datasets.build_dataset(args)
cogdl.datasets.build_dataset_from_name(dataset)
cogdl.datasets.build_dataset_from_path(data_path, task=None, dataset=None)
cogdl.datasets.register_dataset(name)
```

New dataset types can be added to cogdl with the `register_dataset()` function decorator.

For example:

```
@register_dataset('my_dataset')
class MyDataset():
    (...)
```

**Args:** name (str): the name of the dataset

```
cogdl.datasets.try_import_dataset(dataset)
```

## 2.10 tasks

### 2.10.1 Base Task

```
class cogdl.tasks.base_task.BaseTask(args)
    Bases: abc.ABC
    static add_args(parser: argparse.ArgumentParser)
        Add task-specific arguments to the parser.
    get_trainer(args)
    load_from_pretrained()
    save_checkpoint()
    set_evaluator(dataset)
    set_loss_fn(dataset)
    train()
class cogdl.tasks.base_task.LoadFrom
    Bases: abc.ABCMeta
```

## 2.10.2 Node Classification

```
class cogdl.tasks.node_classification.NodeClassification (args, dataset=None,  
                                                    model=None)
```

Bases: *cogdl.tasks.base\_task.BaseTask*

Node classification task.

```
static add_args (parser: argparse.ArgumentParser)  
    Add task-specific arguments to the parser.
```

```
inference ()
```

```
preprocess ()
```

```
train ()
```

## 2.10.3 Unsupervised Node Classification

```
class cogdl.tasks.unsupervised_node_classification.TopKRanker (estimator, *,  
                                                            n_jobs=None)
```

Bases: *sklearn.multiclass.OneVsRestClassifier*

```
predict (X, top_k_list)  
    Predict multi-class targets using underlying estimators.
```

**X** [(sparse) array-like of shape (n\_samples, n\_features)] Data.

**y** [(sparse) array-like of shape (n\_samples,) or (n\_samples, n\_classes)] Predicted multi-class targets.

```
class cogdl.tasks.unsupervised_node_classification.UnsupervisedNodeClassification (args,  
                                                                                dataset=None,  
                                                                                model=None)
```

Bases: *cogdl.tasks.base\_task.BaseTask*

Node classification task.

```
static add_args (parser: argparse.ArgumentParser)  
    Add task-specific arguments to the parser.
```

```
enhance_emb (G, embs)
```

```
save_emb (embs)
```

```
train ()
```

## 2.10.4 Heterogeneous Node Classification

```
class cogdl.tasks.heterogeneous_node_classification.HeterogeneousNodeClassification (args,  
                                                                                dataset=None,  
                                                                                model=None)
```

Bases: *cogdl.tasks.base\_task.BaseTask*

Heterogeneous Node classification task.

```
static add_args (_: argparse.ArgumentParser)  
    Add task-specific arguments to the parser.
```

```
train ()
```

## 2.10.5 Multiplex Node Classification

```
class cogdl.tasks.multiplex_node_classification.MultiplexNodeClassification (args,
                                                                    dataset=None,
                                                                    model=None)

    Bases: cogdl.tasks.base_task.BaseTask

    Node classification task.

    static add_args (parser: argparse.ArgumentParser)
        Add task-specific arguments to the parser.

    train ()
```

## 2.10.6 Link Prediction

```
class cogdl.tasks.link_prediction.GNNHomoLinkPrediction (args,          dataset=None,
                                                         model=None)

    Bases: torch.nn.modules.module.Module

    static get_link_labels (num_pos, num_neg, device=None)

    train ()
        Sets the module in training mode.

        This has any effect only on certain modules. See documentations of particular modules for details of their
        behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

    Args:

        mode (bool): whether to set training mode (True) or evaluation mode (False).      Default:
        True.

    Returns: Module: self

    static train_test_edge_split (edge_index, num_nodes, val_ratio=0.1, test_ratio=0.2)

class cogdl.tasks.link_prediction.HomoLinkPrediction (args,          dataset=None,
                                                         model=None)

    Bases: torch.nn.modules.module.Module

    train ()
        Sets the module in training mode.

        This has any effect only on certain modules. See documentations of particular modules for details of their
        behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

    Args:

        mode (bool): whether to set training mode (True) or evaluation mode (False).      Default:
        True.

    Returns: Module: self

class cogdl.tasks.link_prediction.KGLinkPrediction (args,          dataset=None,
                                                         model=None)

    Bases: torch.nn.modules.module.Module

    train ()
        Sets the module in training mode.

        This has any effect only on certain modules. See documentations of particular modules for details of their
        behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.
```

**Args:**

**mode (bool):** whether to set training mode (**True**) or evaluation mode (**False**). Default: **True**.

**Returns:** Module: self

**class** cogdl.tasks.link\_prediction.**LinkPrediction** (*args, dataset=None, model=None*)

Bases: *cogdl.tasks.base\_task.BaseTask*

**static add\_args** (*parser*)

Add task-specific arguments to the parser.

**load\_from\_pretrained** ()

**save\_checkpoint** ()

**train** ()

**class** cogdl.tasks.link\_prediction.**TripleLinkPrediction** (*args, dataset=None, model=None*)

Bases: *torch.nn.modules.module.Module*

Training process borrowed from *KnowledgeGraphEmbedding*<<https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding>>

**test\_step** (*model, test\_triples, all\_true\_triples, args*)

Evaluate the model on test or valid datasets

**train** ()

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool):** whether to set training mode (**True**) or evaluation mode (**False**). Default: **True**.

**Returns:** Module: self

**train\_step** (*model, optimizer, train\_iterator, args*)

A single train step. Apply back-propagation and return the loss

*cogdl.tasks.link\_prediction.divide\_data* (*input\_list, division\_rate*)

*cogdl.tasks.link\_prediction.evaluate* (*embs, true\_edges, false\_edges*)

*cogdl.tasks.link\_prediction.gen\_node\_pairs* (*train\_data, test\_data, negative\_ratio=5*)

*cogdl.tasks.link\_prediction.get\_score* (*embs, node1, node2*)

*cogdl.tasks.link\_prediction.log\_metrics* (*mode, step, metrics*)

Print the evaluation logs

*cogdl.tasks.link\_prediction.randomly\_choose\_false\_edges* (*nodes, true\_edges, num*)

*cogdl.tasks.link\_prediction.save\_model* (*model, optimizer, save\_variable\_list, args*)

Save the parameters of the model and the optimizer, as well as some other variables such as step and learning\_rate

*cogdl.tasks.link\_prediction.select\_task* (*model\_name=None, model=None*)

*cogdl.tasks.link\_prediction.set\_logger* (*args*)

Write logs to checkpoint and console

## 2.10.7 Multiplex Link Prediction

```
class cogdl.tasks.multiplex_link_prediction.MultiplexLinkPrediction (args,
                                                                    dataset=None,
                                                                    model=None)
```

Bases: *cogdl.tasks.base\_task.BaseTask*

**static add\_args** (*parser: argparse.ArgumentParser*)  
Add task-specific arguments to the parser.

**train** ()

`cogdl.tasks.multiplex_link_prediction.evaluate` (*embs, true\_edges, false\_edges*)

`cogdl.tasks.multiplex_link_prediction.get_score` (*embs, node1, node2*)

## 2.10.8 Graph Classification

```
class cogdl.tasks.graph_classification.GraphClassification (args, dataset=None,
                                                            model=None)
```

Bases: *cogdl.tasks.base\_task.BaseTask*

Supervised graph classification task.

**static add\_args** (*parser: argparse.ArgumentParser*)  
Add task-specific arguments to the parser.

**train** ()

`cogdl.tasks.graph_classification.node_degree_as_feature` (*data*)

Set each node feature as one-hot encoding of degree :param data: a list of class Data :return: a list of class Data

`cogdl.tasks.graph_classification.uniform_node_feature` (*data*)

Set each node feature to the same

## 2.10.9 Unsupervised Graph Classification

```
class cogdl.tasks.unsupervised_graph_classification.UnsupervisedGraphClassification (args,
                                                                                      dataset=None,
                                                                                      model=None)
```

Bases: *cogdl.tasks.base\_task.BaseTask*

Unsupervised graph classification

**static add\_args** (*parser: argparse.ArgumentParser*)  
Add task-specific arguments to the parser.

**save\_emb** (*embs*)

**train** ()

## 2.10.10 Attributed Graph Clustering

```
class cogdl.tasks.attributed_graph_clustering.AttributedGraphClustering (args,
                                                                              dataset=None,
                                                                              _=None)
```

Bases: *cogdl.tasks.base\_task.BaseTask*

Attributed graph clustering task.

```
static add_args (parser: argparse.ArgumentParser)  
    Add task-specific arguments to the parser.  
train () → Dict[str, float]
```

### 2.10.11 Similarity Search

```
class cogdl.tasks.similarity_search.SimilaritySearch (args, dataset=None,  
                                                    model=None)  
    Bases: cogdl.tasks.base_task.BaseTask  
    Similarity Search task.  
static add_args (_: argparse.ArgumentParser)  
    Add task-specific arguments to the parser.  
train ()
```

### 2.10.12 Pretrain

```
class cogdl.tasks.pretrain.PretrainTask (args)  
    Bases: cogdl.tasks.base_task.BaseTask  
static add_args (_: argparse.ArgumentParser)  
    Add task-specific arguments to the parser.  
train ()
```

### 2.10.13 Task Module

```
cogdl.tasks.build_task (args, dataset=None, model=None)
```

```
cogdl.tasks.register_task (name)
```

New task types can be added to cogdl with the `register_task()` function decorator.

For example:

```
@register_task('node_classification')  
class NodeClassification(BaseTask):  
    (...)
```

**Args:** name (str): the name of the task

```
cogdl.tasks.try_import_task (task)
```

## 2.11 models

### 2.11.1 BaseModel

```
class cogdl.models.base_model.BaseModel  
    Bases: torch.nn.modules.module.Module  
static add_args (parser)  
    Add model-specific arguments to the parser.
```



**classmethod** `build_model_from_args` (*args*)  
Build a new model instance.

**forward** (*\*args*)  
Defines the computation performed at every call.  
Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**static** `get_trainer` (*args=None*) → `Optional[Type[cogdl.trainers.base_trainer.BaseTrainer]]`

**graph\_classification\_loss** (*batch*)

**node\_classification\_loss** (*data, mask=None*)

**predict** (*data*)

**set\_device** (*device*)

**set\_loss\_fn** (*loss\_fn*)

## 2.11.2 Supervised Model

**class** `cogdl.models.supervised_model.SupervisedHeterogeneousNodeClassificationModel`  
Bases: `cogdl.models.base_model.BaseModel`, `abc.ABC`

**evaluate** (*data: Any, nodes: Any, targets: Any*) → `Any`

**static** `get_trainer` (*args: Any = None*) → `Optional[Type[SupervisedHeterogeneousNodeClassificationTrainer]]`

**loss** (*data: Any*) → `Any`

**class** `cogdl.models.supervised_model.SupervisedHomogeneousNodeClassificationModel`  
Bases: `cogdl.models.base_model.BaseModel`, `abc.ABC`

**static** `get_trainer` (*args: Any = None*) → `Optional[Type[SupervisedHomogeneousNodeClassificationTrainer]]`

**loss** (*data: Any*) → `Any`

**predict** (*data: Any*) → `Any`

**class** `cogdl.models.supervised_model.SupervisedModel`  
Bases: `cogdl.models.base_model.BaseModel`, `abc.ABC`

**loss** (*data: Any*) → `Any`

## 2.11.3 Embedding Model

**class** `cogdl.models.emb.hope.HOPE` (*dimension, beta*)  
Bases: `cogdl.models.base_model.BaseModel`

The HOPE model from the “Grarep: Asymmetric transitivity preserving graph embedding” paper.

**Args:** `hidden_size` (int) : The dimension of node representation. `beta` (float) : Parameter in katz decomposition.

**static** `add_args` (*parser*)  
Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

`model_name` = 'hope'

**train** (*G*)

The author claim that Katz has superior performance in related tasks  $S_{katz} = (M_g)^{-1} * M_l = (I - \beta * A)^{-1} * \beta * A = (I - \beta * A)^{-1} * (I - (I - \beta * A)) = (I - \beta * A)^{-1} - I$

**class** `cogdl.models.emb.spectral.Spectral` (*dimension*)

Bases: `cogdl.models.base_model.BaseModel`

The Spectral clustering model from the “Leveraging social media networks for classification” paper

**Args:** `hidden_size` (int) : The dimension of node representation.

**static** `add_args` (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

`model_name` = 'spectral'

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode** (bool): whether to set training mode (True) or evaluation mode (False). Default: True.

**Returns:** Module: self

**class** `cogdl.models.emb.hin2vec.Hin2vec` (*hidden\_dim, walk\_length, walk\_num, batch\_size, hop, negative, epochs, lr, cpu=True*)

Bases: `cogdl.models.base_model.BaseModel`

The Hin2vec model from the “HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning” paper.

**Args:** `hidden_size` (int) : The dimension of node representation. `walk_length` (int) : The walk length. `walk_num` (int) : The number of walks to sample for each node. `batch_size` (int) : The batch size of training in Hin2vec. `hop` (int) : The number of hop to construct training samples in Hin2vec. `negative` (int) : The number of nagative samples for each meta2path pair. `epochs` (int) : The number of training iteration. `lr` (float) : The initial learning rate of SGD. `cpu` (bool) : Use CPU or GPU to train hin2vec.

**static** `add_args` (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

`model_name` = 'hin2vec'

**train** (*G, node\_type*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool):** whether to set training mode (**True**) or evaluation mode (**False**). Default: **True**.

**Returns:** Module: self

**class** cogdl.models.emb.netmf.**NetMF** (*dimension, window\_size, rank, negative, is\_large=False*)  
Bases: *cogdl.models.base\_model.BaseModel*

The NetMF model from the “Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec” paper.

**Args:** *hidden\_size* (int) : The dimension of node representation. *window\_size* (int) : The actual context size which is considered in language model. *rank* (int) : The rank in approximate normalized laplacian. *negative* (int) : The number of nagative samples in negative sampling. *is-large* (bool) : When window size is large, use approximated deepwalk matrix to decompose.

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**model\_name** = 'netmf'

**train** (*G*)  
Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool):** whether to set training mode (**True**) or evaluation mode (**False**). Default: **True**.

**Returns:** Module: self

**class** cogdl.models.emb.distmult.**DistMult** (*nentity, nrelation, hidden\_dim, gamma, double\_entity\_embedding=False, double\_relation\_embedding=False*)  
Bases: *cogdl.models.emb.knowledge\_base.KGEModel*

The DistMult model from the ICLR 2015 paper “EMBEDDING ENTITIES AND RELATIONS FOR LEARNING AND INFERENCE IN KNOWLEDGE BASES” <[https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICLR2015\\_updated.pdf](https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICLR2015_updated.pdf)> borrowed from *KnowledgeGraphEmbedding*<<https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding>>

**model\_name** = 'distmult'

**score** (*head, relation, tail, mode*)

**class** cogdl.models.emb.transe.**TransE** (*nentity, nrelation, hidden\_dim, gamma, double\_entity\_embedding=False, double\_relation\_embedding=False*)  
Bases: *cogdl.models.emb.knowledge\_base.KGEModel*

The TransE model from paper “Translating Embeddings for Modeling Multi-relational Data” <<http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf>> borrowed from *KnowledgeGraphEmbedding*<<https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding>>

**model\_name** = 'transe'

**score** (*head, relation, tail, mode*)

**class** cogdl.models.emb.deepwalk.**DeepWalk** (*dimension, walk\_length, walk\_num, window\_size, worker, iteration*)

Bases: *cogdl.models.base\_model.BaseModel*

The DeepWalk model from the “DeepWalk: Online Learning of Social Representations” paper

**Args:** *hidden\_size* (int) : The dimension of node representation. *walk\_length* (int) : The walk length. *walk\_num* (int) : The number of walks to sample for each node. *window\_size* (int) : The actual context size which is considered in language model. *worker* (int) : The number of workers for word2vec. *iteration* (int) : The number of training iteration in word2vec.

**static add\_args** (*parser: argparse.ArgumentParser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*) → cogdl.models.emb.deepwalk.DeepWalk  
Build a new model instance.

**model\_name** = 'deepwalk'

**train** (*G: networkx.classes.graph.Graph, embedding\_model\_creator=<class 'gensim.models.word2vec.Word2Vec'>*)  
Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode** (bool): whether to set training mode (True) or evaluation mode (False). Default: True.

**Returns:** Module: self

**class** cogdl.models.emb.rotate.**RotateE** (*nentity, nrelation, hidden\_dim, gamma, double\_entity\_embedding=False, double\_relation\_embedding=False*)

Bases: cogdl.models.emb.knowledge\_base.KGEModel

Implementation of RotatE model from the paper “RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space” <<https://openreview.net/forum?id=HkgEQnRqYQ>>. borrowed from KnowledgeGraphEmbedding <<https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding>>

**model\_name** = 'rotate'

**score** (*head, relation, tail, mode*)

**class** cogdl.models.emb.gatne.**GATNE** (*dimension, walk\_length, walk\_num, window\_size, worker, epoch, batch\_size, edge\_dim, att\_dim, negative\_samples, neighbor\_samples, schema*)

Bases: *cogdl.models.base\_model.BaseModel*

The GATNE model from the “Representation Learning for Attributed Multiplex Heterogeneous Network” paper

**Args:** *walk\_length* (int) : The walk length. *walk\_num* (int) : The number of walks to sample for each node. *window\_size* (int) : The actual context size which is considered in language model. *worker* (int) : The number of workers for word2vec. *epoch* (int) : The number of training epochs. *batch\_size* (int) : The size of each training batch. *edge\_dim* (int) : Number of edge embedding dimensions. *att\_dim* (int) : Number of attention dimensions. *negative\_samples* (int) : Negative samples for optimization. *neighbor\_samples* (int) : Neighbor samples for aggregation schema (str) : The metapath schema used in model. Metapaths are splited with “;”, while each node type are connected with “-” in each metapath. For example:”0-1-0,0-1-2-1-0”

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'gatne'

**train** (*network\_data*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool):** whether to set training mode (**True**) or evaluation mode (**False**). Default: True.

**Returns:** Module: self

**class** cogdl.models.emb.dgk.**DeepGraphKernel** (*hidden\_dim, min\_count, window\_size, sampling\_rate, rounds, epoch, alpha, n\_workers=4*)

Bases: *cogdl.models.base\_model.BaseModel*

The Hin2vec model from the “Deep Graph Kernels” paper.

**Args:** hidden\_size (int) : The dimension of node representation. min\_count (int) : Parameter in word2vec. window (int) : The actual context size which is considered in language model. sampling\_rate (float) : Parameter in word2vec. iteration (int) : The number of iteration in WL method. epoch (int) : The number of training iteration. alpha (float) : The learning rate of word2vec.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**static feature\_extractor** (*data, rounds, name*)

**forward** (*graphs, \*\*kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'dgk'

**save\_embedding** (*output\_path*)

**static wl\_iterations** (*graph, features, rounds*)

**class** cogdl.models.emb.grarep.**GraRep** (*dimension, step*)

Bases: *cogdl.models.base\_model.BaseModel*

The GraRep model from the “Grarep: Learning graph representations with global structural information” paper.

**Args:** hidden\_size (int) : The dimension of node representation. step (int) : The maximum order of transition probability.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

`model_name = 'grarep'`

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool):** whether to set training mode (**True**) or evaluation mode (**False**). Default: True.

**Returns:** Module: self

**class** `cogdl.models.emb.dngr.DNGR` (*hidden\_size1, hidden\_size2, noise, alpha, step, max\_epoch, lr, cpu*)

Bases: `cogdl.models.base_model.BaseModel`

The DNGR model from the “Deep Neural Networks for Learning Graph Representations” paper

**Args:** `hidden_size1` (int) : The size of the first hidden layer. `hidden_size2` (int) : The size of the second hidden layer. `noise` (float) : Denoise rate of DAE. `alpha` (float) : Parameter in DNGR. `step` (int) : The max step in random surfing. `max_epoch` (int) : The max epoches in training step. `lr` (float) : Learning rate in DNGR.

**static** `add_args` (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

`get_denoised_matrix` (*mat*)

`get_emb` (*matrix*)

`get_ppmi_matrix` (*mat*)

`model_name = 'dngr'`

`random_surfing` (*adj\_matrix*)

`scale_matrix` (*mat*)

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool):** whether to set training mode (**True**) or evaluation mode (**False**). Default: True.

**Returns:** Module: self

**class** `cogdl.models.emb.pronepp.ProNEPP` (*filter\_types, svd, search, max\_evals=None, loss\_type=None, n\_workers=None*)

Bases: `cogdl.models.base_model.BaseModel`

**static** `add_args` (*parser*)

Add model-specific arguments to the parser.

```
classmethod build_model_from_args (args)
    Build a new model instance.
```

```
model_name = 'prone++'
```

```
class cogdl.models.emb.graph2vec.Graph2Vec (dimension, min_count, window_size, dm, sam-
                                           pling_rate, rounds, epoch, lr, worker=4)
```

```
Bases: cogdl.models.base_model.BaseModel
```

The Graph2Vec model from the “graph2vec: Learning Distributed Representations of Graphs” paper

**Args:** *hidden\_size* (int) : The dimension of node representation. *min\_count* (int) : Parameter in doc2vec. *window\_size* (int) : The actual context size which is considered in language model. *sampling\_rate* (float) : Parameter in doc2vec. *dm* (int) : Parameter in doc2vec. *iteration* (int) : The number of iteration in WL method. *epoch* (int) : The max epoches in training step. *lr* (float) : Learning rate in doc2vec.

```
static add_args (parser)
    Add model-specific arguments to the parser.
```

```
classmethod build_model_from_args (args)
    Build a new model instance.
```

```
static feature_extractor (data, rounds, name)
```

```
forward (graphs, **kwargs)
    Defines the computation performed at every call.
    Should be overridden by all subclasses.
```

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'graph2vec'
```

```
save_embedding (output_path)
```

```
static wl_iterations (graph, features, rounds)
```

```
class cogdl.models.emb.metapath2vec.Metapath2vec (dimension, walk_length, walk_num,
                                                  window_size, worker, iteration,
                                                  schema)
```

```
Bases: cogdl.models.base_model.BaseModel
```

The Metapath2vec model from the “metapath2vec: Scalable Representation Learning for Heterogeneous Networks” paper

**Args:** *hidden\_size* (int) : The dimension of node representation. *walk\_length* (int) : The walk length. *walk\_num* (int) : The number of walks to sample for each node. *window\_size* (int) : The actual context size which is considered in language model. *worker* (int) : The number of workers for word2vec. *iteration* (int) : The number of training iteration in word2vec. *schema* (str) : The metapath schema used in model. Metapaths are splited with “,”, while each node type are connected with “-” in each metapath. For example:”0-1-0,0-2-0,1-0-2-0-1”.

```
static add_args (parser)
    Add model-specific arguments to the parser.
```

```
classmethod build_model_from_args (args)
    Build a new model instance.
```

```
model_name = 'metapath2vec'
```

**train** (*G*, *node\_type*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool): whether to set training mode (True) or evaluation mode (False).** Default: True.

**Returns:** Module: self

**class** cogdl.models.emb.node2vec.**Node2vec** (*dimension*, *walk\_length*, *walk\_num*, *window\_size*, *worker*, *iteration*, *p*, *q*)

Bases: *cogdl.models.base\_model.BaseModel*

The node2vec model from the “node2vec: Scalable feature learning for networks” paper

**Args:** *hidden\_size* (int) : The dimension of node representation. *walk\_length* (int) : The walk length. *walk\_num* (int) : The number of walks to sample for each node. *window\_size* (int) : The actual context size which is considered in language model. *worker* (int) : The number of workers for word2vec. *iteration* (int) : The number of training iteration in word2vec. *p* (float) : Parameter in node2vec. *q* (float) : Parameter in node2vec.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'node2vec'

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool): whether to set training mode (True) or evaluation mode (False).** Default: True.

**Returns:** Module: self

**class** cogdl.models.emb.complex.**Complex** (*nentity*, *nrelation*, *hidden\_dim*, *gamma*, *double\_entity\_embedding=False*, *double\_relation\_embedding=False*)

Bases: *cogdl.models.emb.knowledge\_base.KGEModel*

the implementation of ComplEx model from the paper “Complex Embeddings for Simple Link Prediction” <<http://proceedings.mlr.press/v48/trouillon16.pdf>> borrowed from *KnowledgeGraphEmbedding* <<https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding>>

**model\_name** = 'complex'

**score** (*head*, *relation*, *tail*, *mode*)

**class** cogdl.models.emb.pte.**PTE** (*dimension*, *walk\_length*, *walk\_num*, *negative*, *batch\_size*, *alpha*)

Bases: *cogdl.models.base\_model.BaseModel*

The PTE model from the “PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks” paper.



**Args:** `hidden_size` (int) : The dimension of node representation. `walk_length` (int) : The walk length. `walk_num` (int) : The number of walks to sample for each node. `negative` (int) : The number of negative samples for each edge. `batch_size` (int) : The batch size of training in PTE. `alpha` (float) : The initial learning rate of SGD.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'pte'

**train** (*G*, *node\_type*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode** (bool): whether to set training mode (**True**) or evaluation mode (**False**). Default: True.

**Returns:** Module: self

**class** `cogdl.models.emb.net-smf.NetSMF` (*dimension*, *window\_size*, *negative*, *num\_round*, *worker*)

Bases: `cogdl.models.base_model.BaseModel`

The NetSMF model from the “NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization” paper.

**Args:** `hidden_size` (int) : The dimension of node representation. `window_size` (int) : The actual context size which is considered in language model. `negative` (int) : The number of negative samples in negative sampling. `num_round` (int) : The number of round in NetSMF. `worker` (int) : The number of workers for NetSMF.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'net-smf'

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode** (bool): whether to set training mode (**True**) or evaluation mode (**False**). Default: True.

**Returns:** Module: self

**class** `cogdl.models.emb.line.LINE` (*dimension*, *walk\_length*, *walk\_num*, *negative*, *batch\_size*, *alpha*, *order*)

Bases: `cogdl.models.base_model.BaseModel`

The LINE model from the “Line: Large-scale information network embedding” paper.

**Args:** `hidden_size` (int) : The dimension of node representation. `walk_length` (int) : The walk length. `walk_num` (int) : The number of walks to sample for each node. `negative` (int) : The number of negative samples for each edge. `batch_size` (int) : The batch size of training in LINE. `alpha` (float) : The initial learning rate of SGD. `order` (int) : 1 represents perserving 1-st order proximity, 2 represents 2-nd, while 3 means both of them (each of them having dimension/2 node representation).

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'line'

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool): whether to set training mode (True) or evaluation mode (False).** Default: True.

**Returns:** Module: self

**class** `cogdl.models.emb.sdne.SDNE` (*hidden\_size1, hidden\_size2, dropout, alpha, beta, nu1, nu2, max\_epoch, lr, cpu*)

Bases: `cogdl.models.base_model.BaseModel`

The SDNE model from the “Structural Deep Network Embedding” paper

**Args:** `hidden_size1` (int) : The size of the first hidden layer. `hidden_size2` (int) : The size of the second hidden layer. `dropout` (float) : Dropout rate. `alpha` (float) : Trade-off parameter between 1-st and 2-nd order objective function in SDNE. `beta` (float) : Parameter of 2-nd order objective function in SDNE. `nu1` (float) : Parameter of l1 normlization in SDNE. `nu2` (float) : Parameter of l2 normlization in SDNE. `max_epoch` (int) : The max epoches in training step. `lr` (float) : Learning rate in SDNE. `cpu` (bool) : Use CPU or GPU to train `hin2vec`.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'sdne'

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode (bool): whether to set training mode (True) or evaluation mode (False).** Default: True.

**Returns:** Module: self

**class** `cogdl.models.emb.prone.ProNE` (*dimension, step, mu, theta*)

Bases: `cogdl.models.base_model.BaseModel`

The ProNE model from the “ProNE: Fast and Scalable Network Representation Learning” paper.

**Args:** `hidden_size` (int) : The dimension of node representation. `step` (int) : The number of items in the chebyshev expansion. `mu` (float) : Parameter in ProNE. `theta` (float) : Parameter in ProNE.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'prone'

**train** (*G*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode** (bool): whether to set training mode (True) or evaluation mode (False). Default: True.

**Returns:** Module: self

## 2.11.4 GNN Model

**class** `cogdl.models.nn.dgi.DGIModel` (*in\_feats, hidden\_size, activation*)

Bases: `cogdl.models.self_supervised_model.SelfSupervisedContrastiveModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**augment** (*graph*)

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**embed** (*data, msk=None*)

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**static get\_trainer** (*args*)

**loss** (*data*)

**model\_name** = 'dgi'

**self\_supervised\_loss** (*data*)

**class** `cogdl.models.nn.mvgrl.MVGRL` (*in\_feats, hidden\_size, sample\_size=2000, batch\_size=4, alpha=0.2, dataset='cora'*)

Bases: `cogdl.models.self_supervised_model.SelfSupervisedContrastiveModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**augment** (*graph*)

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**embed** (*data, msk=None*)

**forward** (*graph*)  
Defines the computation performed at every call.  
Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**static get\_trainer** (*args*)

**loss** (*data*)

**model\_name** = 'mvgr1'

**preprocess** (*graph*)

**self\_supervised\_loss** (*data*)

**class** `cogdl.models.nn.patchy_san.PatchySAN` (*batch\_size, num\_features, num\_classes, num\_sample, stride, num\_neighbor, iteration*)

Bases: `cogdl.models.base_model.BaseModel`

The Patchy-SAN model from the “Learning Convolutional Neural Networks for Graphs” paper.

**Args:** `batch_size` (int) : The batch size of training. `sample` (int) : Number of chosen vertexes. `stride` (int) : Node selection stride. `neighbor` (int) : The number of neighbor for each node. `iteration` (int) : The number of training iteration.

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**build\_model** (*num\_channel, num\_sample, num\_neighbor, num\_class*)

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**forward** (*batch*)  
Defines the computation performed at every call.  
Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'patchy\_san'

**classmethod split\_dataset** (*dataset, args*)

**class** `cogdl.models.nn.pyg_cheb.Chebyshev` (*in\_feats, hidden\_size, out\_feats, num\_layers, dropout, filter\_size*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**forward** (*graph*)  
Defines the computation performed at every call.  
Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'chebyshev'

**predict** (*data*)

**class** `cogdl.models.nn.gcn.TKipfGCN` (*in\_feats, hidden\_size, out\_feats, num\_layers, dropout, activation='relu', residual=False, norm=None, actnn=False*)  
Bases: `cogdl.models.base_model.BaseModel`

The GCN model from the “Semi-Supervised Classification with Graph Convolutional Networks” paper

**Args:** *in\_features* (int) : Number of input features. *out\_features* (int) : Number of classes. *hidden\_size* (int) : The dimension of node representation. *dropout* (float) : Dropout rate for model training.

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**embed** (*graph*)

**forward** (*graph*)  
Defines the computation performed at every call.  
Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'gcn'

**predict** (*data*)

**class** `cogdl.models.nn.gdc_gcn.GDC_GCN` (*nfeat, nhid, nclass, dropout, alpha, t, k, eps, gdctype*)  
Bases: `cogdl.models.base_model.BaseModel`

The GDC model from the “Diffusion Improves Graph Learning” paper, with the PPR and heat matrix variants combined with GCN

**Args:** *num\_features* (int) : Number of input features in ppr-preprocessed dataset. *num\_classes* (int) : Number of classes. *hidden\_size* (int) : The dimension of node representation. *dropout* (float) : Dropout rate for model training. *alpha* (float) : PPR polynomial filter param, 0 to 1. *t* (float) : Heat polynomial filter param *k* (int) : Top k nodes retained during sparsification. *eps* (float) : Threshold for clipping. *gdc\_type* (str) : “none”, “ppr”, “heat”

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**forward** (*graph*)  
Defines the computation performed at every call.  
Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'gdc_gcn'
node_classification_loss (data)
predict (data=None)
preprocessing (data, gdc_type='ppr')
reset_data (data)
class cogdl.models.nn.pyg_hgpsl.HGPSL(num_features, num_classes, hidden_size, dropout,
                                     pooling, sample_neighbor, sparse_attention, struc-
                                     ture_learning, lamb)
Bases: cogdl.models.base_model.BaseModel
static add_args (parser)
    Add model-specific arguments to the parser.
classmethod build_model_from_args (args)
    Build a new model instance.
forward (data)
    Defines the computation performed at every call.
    Should be overridden by all subclasses.
```

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'hgpsl'
classmethod split_dataset (dataset, args)
class cogdl.models.nn.graphsage.Graphsage(num_features, num_classes, hidden_size,
                                          num_layers, sample_size, dropout, aggr)
Bases: cogdl.models.base_model.BaseModel
static add_args (parser)
    Add model-specific arguments to the parser.
classmethod build_model_from_args (args)
    Build a new model instance.
forward (*args)
    Defines the computation performed at every call.
```

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```

static get_trainer (args)
inference (x_all, data_loader)
mini_forward (graph)
mini_loss (data)
model_name = 'graphsage'
node_classification_loss (*args)
predict (data)
sampling (edge_index, num_sample)
set_data_device (device)

class cogdl.models.nn.compgcn.LinkPredictCompGCN (num_entities, num_rels, hid-
den_size, num_bases=0, lay-
ers=1, sampling_rate=0.01,
score_func='conve', penalty=0.001,
dropout=0.0, lbl_smooth=0.1,
opn='sub')
    Bases: cogdl.utils.link_prediction_utils.GNNLinkPredict, cogdl.models.
    base_model.BaseModel

    static add_args (parser)
        Add model-specific arguments to the parser.

    add_reverse_edges (edge_index, edge_types)

    classmethod build_model_from_args (args)
        Build a new model instance.

    forward (graph)
        Defines the computation performed at every call.

        Should be overridden by all subclasses.

```

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```

loss (data: cogdl.data.data.Graph, split='train')
model_name = 'compgcn'
predict (graph)

class cogdl.models.nn.drgcn.DrGCN (num_features, num_classes, hidden_size, num_layers,
dropout, norm=None, activation='relu')
    Bases: cogdl.models.base_model.BaseModel

```

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**forward** (*graph*)  
Defines the computation performed at every call.  
Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'drgcn'

**predict** (*graph*)

**class** `cogdl.models.nn.pyg_gpt_gnn.GPT_GNN`  
Bases: `cogdl.models.supervised_model.SupervisedHomogeneousNodeClassificationModel`,  
`cogdl.models.supervised_model.SupervisedHeterogeneousNodeClassificationModel`

**static add\_args** (*parser*)  
Add task-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**evaluate** (*data: Any, nodes: Any, targets: Any*) → *Any*

**static get\_trainer** (*args*) → *Optional[Type[Union[cogdl.trainers.gpt\_gnn\_trainer.GPT\_GNNHomogeneousTrainer, cogdl.trainers.gpt\_gnn\_trainer.GPT\_GNNHeterogeneousTrainer]]]*

**loss** (*data: Any*) → *Any*

**model\_name** = 'gpt\_gnn'

**predict** (*data: Any*) → *Any*

**class** `cogdl.models.nn.pyg_graph_unet.GraphUnet` (*in\_feats: int, hidden\_size: int, out\_feats: int, pooling\_layer: int, pooling\_rates: List[float], n\_dropout: float = 0.5, adj\_dropout: float = 0.3, activation: str = 'elu', improved: bool = False, aug\_adj: bool = False*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)  
Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
Build a new model instance.

**forward** (*graph: cogdl.data.data.Graph*) → `torch.Tensor`  
Defines the computation performed at every call.  
Should be overridden by all subclasses.



---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'unet'
```

```
class cogdl.models.nn.gcnmix.GCNMix(in_feat, hidden_size, num_classes, k, temperature, alpha,  
                                   rampup_starts, rampup_ends, final_consistency_weight,  
                                   ema_decay, dropout)
```

Bases: `cogdl.models.base_model.BaseModel`

```
static add_args(parser)
```

Add model-specific arguments to the parser.

```
classmethod build_model_from_args(args)
```

Build a new model instance.

```
forward(graph)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
forward_ema(graph)
```

```
model_name = 'gcnmix'
```

```
node_classification_loss(data)
```

```
predict(data)
```

```
class cogdl.models.nn.diffpool.DiffPool(in_feats, hidden_dim, embed_dim, num_classes,  
                                         num_layers, num_pool_layers, assign_dim,  
                                         pooling_ratio, batch_size, dropout=0.5,  
                                         no_link_pred=True, concat=False, use_bn=False)
```

Bases: `cogdl.models.base_model.BaseModel`

DIFFPOOL from paper [Hierarchical Graph Representation Learning with Differentiable Pooling](#).

**in\_feats** [int] Size of each input sample.

**hidden\_dim** [int] Size of hidden layer dimension of GNN.

**embed\_dim** [int] Size of embedded node feature, output size of GNN.

**num\_classes** [int] Number of target classes.

**num\_layers** [int] Number of GNN layers.

**num\_pool\_layers** [int] Number of pooling.

**assign\_dim** [int] Embedding size after the first pooling.

**pooling\_ratio** [float] Size of each pooling ratio.

**batch\_size** [int] Size of each mini-batch.

**dropout** [float, optional] Size of dropout, default: 0.5.

**no\_link\_pred** [bool, optional] If True, use link prediction loss, default: *True*.

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**after\_pooling\_forward** (*gnn\_layers, adj, x, concat=False*)

**classmethod build\_model\_from\_args** (*args*)  
 Build a new model instance.

**forward** (*batch*)  
 Defines the computation performed at every call.  
 Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**graph\_classificatoin\_loss** (*batch*)

**model\_name** = 'diffpool'

**reset\_parameters** ()

**classmethod split\_dataset** (*dataset, args*)

**class** `cogdl.models.nn.gcnii.GCNII` (*in\_feats, hidden\_size, out\_feats, num\_layers, dropout=0.5, alpha=0.1, lmbda=1, wd1=0.0, wd2=0.0, residual=False, actnn=False*)

Bases: `cogdl.models.base_model.BaseModel`

Implementation of GCNII in paper “Simple and Deep Graph Convolutional Networks” <<https://arxiv.org/abs/2007.02133>>.

**in\_feats** [int] Size of each input sample

**hidden\_size** [int] Size of each hidden unit

**out\_feats** [int] Size of each out sample

**num\_layers** : int **dropout** : float **alpha** : float

Parameter of initial residual connection

**lmbda** [float] Parameter of identity mapping

**wd1** [float] Weight-decay for Fully-connected layers

**wd2** [float] Weight-decay for convolutional layers

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
 Build a new model instance.

**forward** (*graph*)  
 Defines the computation performed at every call.  
 Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

`get_optimizer` (*args*)

`model_name` = 'gcnii'

`predict` (*graph*)

**class** `cogdl.models.nn.sign.MLP` (*in\_feats, out\_feats, hidden\_size, num\_layers, dropout=0.0, activation='relu', norm=None, act\_first=False, bias=True*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

`model_name` = 'mlp'

`predict` (*data*)

**class** `cogdl.models.nn.pyg_gcn.GCN` (*num\_features, num\_classes, hidden\_size, num\_layers, dropout*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

`get_embeddings` (*x, edge\_index, weight=None*)

`model_name` = 'pyg\_gcn'

```
class cogdl.models.nn.mixhop.MixHop(num_features, num_classes, dropout, layer1_pows,  
                                     layer2_pows)
```

Bases: *cogdl.models.base\_model.BaseModel*

```
static add_args(parser)
```

Add model-specific arguments to the parser.

```
classmethod build_model_from_args(args)
```

Build a new model instance.

```
forward(graph)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'mixhop'
```

```
predict(data)
```

```
class cogdl.models.nn.gat.GAT(in_feats, hidden_size, out_features, num_layers, dropout,  
                               attn_drop, alpha, nhead, residual, last_nhead, norm=None)
```

Bases: *cogdl.models.base\_model.BaseModel*

The GAT model from the “Graph Attention Networks” paper

**Args:** `num_features` (int) : Number of input features. `num_classes` (int) : Number of classes. `hidden_size` (int) : The dimension of node representation. `dropout` (float) : Dropout rate for model training. `alpha` (float) : Coefficient of leaky\_relu. `nheads` (int) : Number of attention heads.

```
static add_args(parser)
```

Add model-specific arguments to the parser.

```
classmethod build_model_from_args(args)
```

Build a new model instance.

```
forward(graph)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'gat'
```

```
predict(graph)
```

```
class cogdl.models.nn.han.HAN(num_edge, w_in, w_out, num_class, num_nodes, num_layers)
```

Bases: *cogdl.models.base\_model.BaseModel*

```
static add_args(parser)
```

Add model-specific arguments to the parser.

```
classmethod build_model_from_args(args)
```

Build a new model instance.

**evaluate** (*data, nodes, targets*)

**forward** (*graph, target\_x, target*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**loss** (*data*)

**model\_name** = 'han'

**class** `cogdl.models.nn.pnpn.PPNNP` (*nfeat, nhid, nclass, num\_layers, dropout, propagation, alpha, niter, cache=True*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'pnpn'

**predict** (*graph*)

**class** `cogdl.models.nn.grace.GRACE` (*in\_feats: int, hidden\_size: int, proj\_hidden\_size: int, num\_layers: int, drop\_feature\_rates: List[float], drop\_edge\_rates: List[float], tau: float = 0.5, activation: str = 'relu', batch\_size: int = -1*)

Bases: `cogdl.models.self_supervised_model.SelfSupervisedContrastiveModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**augment** (*graph*)

**batched\_loss** (*z1: torch.Tensor, z2: torch.Tensor, batch\_size: int*)

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**contrastive\_loss** (*z1: torch.Tensor, z2: torch.Tensor*)

**drop\_adj** (*graph: cogdl.data.data.Graph, drop\_rate: float = 0.5*)

**drop\_feature** (*x: torch.Tensor, droprate: float*)

**embed** (*data*)

**forward** (*graph*: *cogdl.data.data.Graph*, *x*: *torch.Tensor*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**static get\_trainer** (*args*)

**model\_name** = 'grace'

**prop** (*graph*: *cogdl.data.data.Graph*, *x*: *torch.Tensor*, *drop\_feature\_rate*: *float* = 0.0, *drop\_edge\_rate*: *float* = 0.0)

**self\_supervised\_loss** (*graph*)

**class** `cogdl.models.nn.dgl_jknet.JKNet` (*in\_features*, *out\_features*, *n\_layers*, *n\_units*, *node\_aggregation*, *layer\_aggregation*)

Bases: `cogdl.models.supervised_model.SupervisedHomogeneousNodeClassificationModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**static get\_trainer** (*args*)

**loss** (*data*)

**model\_name** = 'jknet'

**predict** (*data*)

**set\_graph** (*graph*)

**class** `cogdl.models.nn.pprgo.PPRGo` (*in\_feats*, *hidden\_size*, *out\_feats*, *num\_layers*, *alpha*, *dropout*, *activation='relu'*, *nprop=2*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*x*, *targets*, *ppr\_scores*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**static** `get_trainer` (*args: Any*)

**model\_name** = 'pprgo'

**node\_classification\_loss** (*x, targets, ppr\_scores, y*)

**predict** (*graph, batch\_size, norm*)

**class** `cogdl.models.nn.gin.GIN` (*num\_layers, in\_feats, out\_feats, hidden\_dim, num\_mlp\_layers, eps=0, pooling='sum', train\_eps=False, dropout=0.5*)

Bases: `cogdl.models.base_model.BaseModel`

Graph Isomorphism Network from paper “How Powerful are Graph Neural Networks?”.

**Args:**

**num\_layers** [int] Number of GIN layers

**in\_feats** [int] Size of each input sample

**out\_feats** [int] Size of each output sample

**hidden\_dim** [int] Size of each hidden layer dimension

**num\_mlp\_layers** [int] Number of MLP layers

**eps** [float32, optional] Initial *epsilon* value, default: 0

**pooling** [str, optional] Aggregator type to use, default:sum

**train\_eps** [bool, optional] If True, *epsilon* will be a learnable parameter, default: True

**static** `add_args` (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

**forward** (*batch*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'gin'

**classmethod** `split_dataset` (*dataset, args*)

**class** `cogdl.models.nn.pyg_dgcnn.DGCNN` (*in\_feats, hidden\_dim, out\_feats, k=20, dropout=0.5*)

Bases: `cogdl.models.base_model.BaseModel`

EdgeConv and DynamicGraph in paper “Dynamic Graph CNN for Learning on Point Clouds” <<https://arxiv.org/pdf/1801.07829.pdf>>\_\_.

**in\_feats** [int] Size of each input sample.

**out\_feats** [int] Size of each output sample.

**hidden\_dim** [int] Dimension of hidden layer embedding.

**k** [int] Number of nearest neighbors.

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
 Build a new model instance.

**forward** (*batch*)  
 Defines the computation performed at every call.  
 Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```

model_name = 'dgcnn'

classmethod split_dataset (dataset, args)

class cogdl.models.nn.grand.Grand (nfeat, nhid, nclass, input_dropout, hidden_dropout,
                                     use_bn, droptime_rate, tem, lam, order, sample, alpha)
    Bases: cogdl.models.base_model.BaseModel

    Implementation of GRAND in paper “Graph Random Neural Networks for Semi-Supervised Learning on
    Graphs” <https://arxiv.org/abs/2005.11079>

    nfeat [int] Size of each input features.
    nhid [int] Size of hidden features.
    nclass [int] Number of output classes.
    input_dropout [float] Dropout rate of input features.
    hidden_dropout [float] Dropout rate of hidden features.
    use_bn [bool] Using batch normalization.
    droptime_rate [float] Rate of dropping elements of input features
    tem [float] Temperature to sharpen predictions.
    lam [float] Proportion of consistency loss of unlabelled data
    order [int] Order of adjacency matrix
    sample [int] Number of augmentations for consistency loss
    alpha : float

    static add_args (parser)
        Add model-specific arguments to the parser.

    classmethod build_model_from_args (args)
        Build a new model instance.

    consis_loss (logps, train_mask)
    
```



**dropNode** (*x*)

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'grand'

**node\_classification\_loss** (*graph*)

**normalize\_x** (*x*)

**predict** (*data*)

**rand\_prop** (*graph, x*)

**class** cogdl.models.nn.pyg\_gtn.**GTN** (*num\_edge, num\_channels, w\_in, w\_out, num\_class, num\_nodes, num\_layers*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**evaluate** (*data, nodes, targets*)

**forward** (*graph, target\_x, target*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**loss** (*data*)

**model\_name** = 'gtn'

**norm** (*edge\_index, num\_nodes, edge\_weight, improved=False, dtype=None*)

**normalization** (*H*)

**class** cogdl.models.nn.rgcn.**LinkPredictRGCN** (*num\_entities, num\_rels, hidden\_size, num\_layers, regularizer='basis', num\_bases=None, self\_loop=True, sampling\_rate=0.01, penalty=0, dropout=0.0, self\_dropout=0.0*)

Bases: `cogdl.utils.link_prediction_utils.GNNLinkPredict`, `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)  
 Build a new model instance.

**forward** (*graph*)  
 Defines the computation performed at every call.  
 Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**loss** (*graph*, *split*='train')

**model\_name** = 'rgcn'

**predict** (*graph*)

**class** `cogdl.models.nn.deepergcn.DeeperGCN` (*in\_feat*, *hidden\_size*, *out\_feat*, *num\_layers*, *activation*='relu', *dropout*=0.0, *aggr*='max', *beta*=1.0, *p*=1.0, *learn\_beta*=False, *learn\_p*=False, *learn\_msg\_scale*=True, *use\_msg\_norm*=False, *edge\_attr\_size*=None)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)  
 Build a new model instance.

**forward** (*graph*)  
 Defines the computation performed at every call.  
 Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'deepergcn'

**predict** (*graph*)

**class** `cogdl.models.nn.drgat.DrGAT` (*num\_features*, *num\_classes*, *hidden\_size*, *num\_heads*, *dropout*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)  
 Build a new model instance.

**forward** (*graph*)  
 Defines the computation performed at every call.  
 Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'drgat'
```

```
class cogdl.models.nn.infograph.InfoGraph (in_feats, hidden_dim, out_feats, num_layers=3,
                                           sup=False)
```

```
Bases: cogdl.models.base_model.BaseModel
```

**Implimentation of Infograph in paper “InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization”** <<https://openreview.net/forum?id=r1lfF2NYvH>>.

**in\_feats** [int] Size of each input sample.

**out\_feats** [int] Size of each output sample.

**num\_layers** [int, optional] Number of MLP layers in encoder, default: 3.

**unsup** [bool, optional] Use unsupervised model if True, default: True.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*batch*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
graph_classification_loss (batch)
```

```
static mi_loss (pos_mask, neg_mask, mi, pos_div, neg_div)
```

```
model_name = 'infograph'
```

```
reset_parameters ()
```

```
classmethod split_dataset (dataset, args)
```

```
sup_forward (batch, x)
```

```
sup_loss (pred, batch)
```

```
unsup_forward (batch, x)
```

```
unsup_loss (graph_feat, node_feat, batch)
```

```
unsup_sup_loss (batch, x)
```

```
class cogdl.models.nn.dropegedge_gcn.DropEdge_GCN (nfeat, nhid, nclass, nhidlayer, dropout,
                                                    baseblock, inputlayer, outputlayer,
                                                    nbaselayer, activation, withbn, with-
                                                    loop, aggrmethod)
```

```
Bases: cogdl.models.base_model.BaseModel
```

DropEdge: Towards Deep Graph Convolutional Networks on Node Classification Applying DropEdge to GCN @ <https://arxiv.org/pdf/1907.10903.pdf>

The model for the single kind of deepgcn blocks. The model architecture likes: inputlayer(nfeat)–block(nbaselayer, nhid)–...–outputlayer(nclass)–softmax(nclass)

└─── **nhidlayer** ───┘

The total layer is nhidlayer\*nbaselayer + 2. All options are configurable.

**Args:** Initial function. :param nfeat: the input feature dimension. :param nhid: the hidden feature dimension. :param nclass: the output feature dimension. :param nhidlayer: the number of hidden blocks. :param dropout: the dropout ratio. :param baseblock: the baseblock type, can be “mutigcn”, “resgcn”, “densegcn” and “inceptiongcn”. :param inputlayer: the input layer type, can be “gcn”, “dense”, “none”. :param outputlayer: the input layer type, can be “gcn”, “dense”. :param nbaselayer: the number of layers in one hidden block. :param activation: the activation function, default is ReLu. :param withbn: using batch normalization in graph convolution. :param withloop: using self feature modeling in graph convolution. :param aggrmethod: the aggregation function for baseblock, can be “concat” and “add”. For “resgcn”, the default is “add”, for others the default is “concat”.

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'dropedge\_gcn'

**predict** (*data*)

**reset\_parameters** ()

**class** cogdl.models.nn.disengcn.**DisenGCN** (*in\_feats, hidden\_size, num\_classes, K, iterations, tau, dropout, activation*)

Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks

---

while the latter silently ignores them.

---

```
model_name = 'disengcn'
```

```
predict (data)
```

```
reset_parameters ()
```

```
class cogdl.models.nn.mlp.MLP (in_feats, out_feats, hidden_size, num_layers, dropout=0.0, activation='relu', norm=None, act_first=False, bias=True)
```

Bases: `cogdl.models.base_model.BaseModel`

```
static add_args (parser)
```

Add model-specific arguments to the parser.

```
classmethod build_model_from_args (args)
```

Build a new model instance.

```
forward (x)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'mlp'
```

```
predict (data)
```

```
class cogdl.models.nn.sgc.sgc (in_feats, out_feats)
```

Bases: `cogdl.models.base_model.BaseModel`

```
static add_args (parser)
```

Add model-specific arguments to the parser.

```
classmethod build_model_from_args (args)
```

Build a new model instance.

```
forward (graph)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
model_name = 'sgc'
```

```
predict (data)
```

```
class cogdl.models.nn.stpgnn.stpgnn (args)
```

Bases: `cogdl.models.base_model.BaseModel`

Implementation of models in paper “Strategies for Pre-training Graph Neural Networks”. <<https://arxiv.org/abs/1905.12265>>

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
 Build a new model instance.

**model\_name** = 'stpgnn'

**class** cogdl.models.nn.sortpool.**SortPool** (*in\_feats, hidden\_dim, num\_classes, num\_layers, out\_channel, kernel\_size, k=30, dropout=0.5*)  
 Bases: *cogdl.models.base\_model.BaseModel*

Implimentation of sortpooling in paper “An End-to-End Deep Learning Architecture for Graph Classification” <[https://www.cse.wustl.edu/~muhan/papers/AAAI\\_2018\\_DGCNN.pdf](https://www.cse.wustl.edu/~muhan/papers/AAAI_2018_DGCNN.pdf)>\_\_.

**in\_feats** [int] Size of each input sample.

**out\_feats** [int] Size of each output sample.

**hidden\_dim** [int] Dimension of hidden layer embedding.

**num\_classes** [int] Number of target classes.

**num\_layers** [int] Number of graph neural network layers before pooling.

**k** [int, optional] Number of selected features to sort, default: 30.

**out\_channel** [int] Number of the first convolution’s output channels.

**kernel\_size** [int] Size of the first convolution’s kernel.

**dropout** [float, optional] Size of dropout, default: 0.5.

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
 Build a new model instance.

**forward** (*batch*)  
 Defines the computation performed at every call.  
 Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'sortpool'

**classmethod split\_dataset** (*dataset, args*)

**class** cogdl.models.nn.pyg\_srgcn.**SRGCN** (*in\_feats, hidden\_size, out\_feats, attention, activation, nhop, normalization, dropout, node\_dropout, alpha, nhead, subheads*)  
 Bases: *cogdl.models.base\_model.BaseModel*

**static add\_args** (*parser*)  
 Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)  
 Build a new model instance.

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**model\_name** = 'srgcn'

**predict** (*data*)

**class** `cogdl.models.nn.dgl_gcc.GCC` (*load\_path*)  
 Bases: `cogdl.models.base_model.BaseModel`

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod build\_model\_from\_args** (*args*)

Build a new model instance.

**model\_name** = 'gcc'

**train** (*data*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

**Args:**

**mode** (bool): whether to set training mode (True) or evaluation mode (False). Default: True.

**Returns:** Module: self

**class** `cogdl.models.nn.unsup_graphsage.SAGE` (*num\_features*, *hidden\_size*, *num\_layers*, *sample\_size*, *dropout*, *walk\_length*, *negative\_samples*)

Bases: `cogdl.models.base_model.BaseModel`

Implementation of unsupervised GraphSAGE in paper “*Inductive Representation Learning on Large Graphs*” <<https://cs.stanford.edu/people/jure/pubs/graphsage-nips17.pdf>>

**num\_features** [int] Size of each input sample

**hidden\_size** : int **num\_layers** : int

The number of GNN layers.

**samples\_size** [list] The number sampled neighbors of different orders

**dropout** : float **walk\_length** : int

The length of random walk

**negative\_samples** : int

**static add\_args** (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

**embed** (*data*)

**forward** (*graph*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**static** `get_trainer` (*args*)

**loss** (*data*)

`model_name` = 'unsup\_graphsage'

**node\_classification\_loss** (*data*)

**sampling** (*edge\_index, num\_sample*)

**self\_supervised\_loss** (*data*)

**class** `cogdl.models.nn.pyg_sagpool.SAGPoolNetwork` (*nfeat, nhid, nclass, dropout, pooling\_ratio, pooling\_layer\_type*)

Bases: `cogdl.models.base_model.BaseModel`

**static** `add_args` (*parser*)

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*args*)

Build a new model instance.

**forward** (*batch*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

`model_name` = 'sagpool'

**classmethod** `split_dataset` (*dataset, args*)

## 2.11.5 AGC Model

## 2.11.6 Model Module

`cogdl.models.build_model` (*args*)

`cogdl.models.register_model` (*name*)

New model types can be added to cogdl with the `register_model()` function decorator.

For example:



```
@register_model('gat')
class GAT(BaseModel):
    (...)
```

**Args:** name (str): the name of the model

`cogdl.models.try_import_model(model)`

## 2.12 layers

### 2.12.1 Layers

**class** `cogdl.layers.gcn_layer.GCNLayer` (*in\_features*, *out\_features*, *dropout=0.0*, *activation=None*, *residual=False*, *norm=None*, *bias=True*)

Bases: `torch.nn.modules.module.Module`

Simple GCN layer, similar to <https://arxiv.org/abs/1609.02907>

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset\_parameters** ()

**class** `cogdl.layers.gat_layer.GATLayer` (*in\_features*, *out\_features*, *nhead=1*, *alpha=0.2*, *attn\_drop=0.5*, *activation=None*, *residual=False*, *norm=None*)

Bases: `torch.nn.modules.module.Module`

Sparse version GAT layer, similar to <https://arxiv.org/abs/1710.10903>

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset\_parameters** ()

**class** `cogdl.layers.sage_layer.MeanAggregator`

Bases: `object`

**class** `cogdl.layers.sage_layer.SAGELayer` (*in\_feats*, *out\_feats*, *normalize=False*, *aggr='mean'*, *dropout=0.0*, *norm=None*, *activation=None*)

Bases: `torch.nn.modules.module.Module`

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** `cogdl.layers.sage_layer.SumAggregator`

Bases: `object`

**class** `cogdl.layers.gin_layer.GINLayer` (*apply\_func=None*, *eps=0*, *train\_eps=True*)

Bases: `torch.nn.modules.module.Module`

Graph Isomorphism Network layer from paper “How Powerful are Graph Neural Networks?”.

$$h_i^{(l+1)} = f_{\Theta} \left( (1 + \epsilon) h_i^l + \text{sum} \left( \{ h_j^l, j \in \mathcal{N}(i) \} \right) \right)$$

**apply\_func** [callable layer function)] layer or function applied to update node feature

**eps** [float32, optional] Initial *epsilon* value.

**train\_eps** [bool, optional] If True, *epsilon* will be a learnable parameter.

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** `cogdl.layers.gcnii_layer.GCNIIayer` (*n\_channels*, *alpha=0.1*, *beta=1*, *residual=False*)

Bases: `torch.nn.modules.module.Module`

**forward** (*graph*, *x*, *init\_x*)

Symmetric normalization

**reset\_parameters** ()

**class** `cogdl.layers.deepergcn_layer.BondEncoder` (*bond\_dim\_list*, *emb\_size*)

Bases: `torch.nn.modules.module.Module`

**forward** (*edge\_attr*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** `cogdl.layers.deepergcn_layer.EdgeEncoder` (*in\_feats*, *out\_feats*, *bias=False*)

Bases: `torch.nn.modules.module.Module`

**forward** (*edge\_attr*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
class cogdl.layers.deepergcn_layer.GENConv (in_feats: int, out_feats: int, aggr: str
      = 'softmax_sg', beta: float = 1.0, p:
      float = 1.0, learn_beta: bool = False,
      learn_p: bool = False, use_msg_norm: bool
      = False, learn_msg_scale: bool = True,
      norm: Optional[str] = None, residual: bool
      = False, activation: Optional[str] = None,
      num_mlp_layers: int = 2, edge_attr_size: Op-
      tional[list] = None)
```

Bases: `torch.nn.modules.module.Module`

**forward** (*graph, x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**message\_norm** (*x, msg*)

```
class cogdl.layers.deepergcn_layer.ResGNNLayer (conv,          in_channels,    activa-
      tion='relu',          norm='batchnorm',
      dropout=0.0,          out_norm=None,
      out_channels=-1,      residual=True,
      checkpoint_grad=False)
```

Bases: `torch.nn.modules.module.Module`

Implementation of DeeperGCN in paper “DeeperGCN: All You Need to Train Deeper GCNs” <<https://arxiv.org/abs/2006.07739>>

**conv** [`nn.Module`] An instance of GNN Layer, receiving (`graph, x`) as inputs

**n\_channels** [`int`] size of input features

**activation** : `str` **norm**: `str`

type of normalization, `batchnorm` as default

**dropout** : `float` **checkpoint\_grad** : `bool`

**forward** (*graph, x, dropout=None, \*args, \*\*kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

---

**class** `cogdl.layers.disengcn_layer.DisenGCNLayer` (*in\_feats*, *out\_feats*, *K*, *iterations*,  
*tau=1.0*, *activation='leaky\_relu'*)

Bases: `torch.nn.modules.module.Module`

Implementation of “Disentangled Graph Convolutional Networks” <<http://proceedings.mlr.press/v97/ma19a.html>>.

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset\_parameters** ()

**class** `cogdl.layers.han_layer.AttentionLayer` (*num\_features*)

Bases: `torch.nn.modules.module.Module`

**forward** (*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** `cogdl.layers.han_layer.HANLayer` (*num\_edge*, *w\_in*, *w\_out*)

Bases: `torch.nn.modules.module.Module`

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** `cogdl.layers.mlp_layer.MLP` (*in\_feats*, *out\_feats*, *hidden\_size*, *num\_layers*, *dropout=0.0*,  
*activation='relu'*, *norm=None*, *act\_first=False*, *bias=True*)

Bases: `torch.nn.modules.module.Module`

Multilayer perception with normalization

$$x^{(i+1)} = \sigma(W^i x^{(i)})$$

**in\_feats** [int] Size of each input sample.

**out\_feats** [int] Size of each output sample.

**hidden\_dim** [int] Size of hidden layer dimension.

---

**use\_bn** [bool, optional] Apply batch normalization if True, default: `True`).

**forward** (*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset\_parameters** ()

**class** `cogdl.layers.pprgo_layer.LinearLayer` (*in\_features*, *out\_features*, *bias=True*)

Bases: `torch.nn.modules.module.Module`

**forward** (*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset\_parameters** ()

**class** `cogdl.layers.pprgo_layer.PPRGoLayer` (*in\_feats*, *hidden\_size*, *out\_feats*, *num\_layers*, *dropout*, *activation='relu'*)

Bases: `torch.nn.modules.module.Module`

**forward** (*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**class** `cogdl.layers.rgcn_layer.RGCNLayer` (*in\_feats*, *out\_feats*, *num\_edge\_types*, *regularizer='basis'*, *num\_bases=None*, *self\_loop=True*, *dropout=0.0*, *self\_dropout=0.0*, *layer\_norm=True*, *bias=True*)

Bases: `torch.nn.modules.module.Module`

**Implementation of Relational-GCN in paper “Modeling Relational Data with Graph Convolutional Networks”**  
<https://arxiv.org/abs/1703.06103>

**in\_feats** [int] Size of each input embedding.

**out\_feats** [int] Size of each output embedding.

**num\_edge\_type** [int] The number of edge type in knowledge graph.

**regularizer** [str, optional] Regularizer used to avoid overfitting, `basis` or `bdd`, default: `basis`.

**num\_bases** [int, optional] The number of basis, only used when *regularizer* is *basis*, default: `None`.

**self\_loop** [bool, optional] Add self loop embedding if True, default : True.

dropout : float self\_dropout : float, optional

Dropout rate of self loop embedding, default : 0.0

**layer\_norm** [bool, optional] Use layer normalization if True, default : True

bias : bool

**basis\_forward** (*graph*, *x*)

**bdd\_forward** (*graph*, *x*)

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset\_parameters** ()

Modified from <https://github.com/GraphSAINT/GraphSAINT>

```
class cogdl.layers.saint_layer.SAINTLayer (dim_in, dim_out, dropout=0.0, act='relu',
                                          order=1, aggr='mean', bias='norm-nn',
                                          **kwargs)
```

Bases: `torch.nn.modules.module.Module`

**forward** (*graph*, *x*)

**Inputs:** graph normalized adj matrix of the subgraph x 2D matrix of input node features

**Outputs:** feat\_out 2D matrix of output node features

```
class cogdl.layers.sgc_layer.SGCLayer (in_features, out_features, order=3)
```

Bases: `torch.nn.modules.module.Module`

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

```
class cogdl.layers.mixhop_layer.MixHopLayer (num_features, adj_pows, dim_per_pow)
```

Bases: `torch.nn.modules.module.Module`

**adj\_pow\_x** (*graph*, *x*, *p*)

**forward** (*graph*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset\_parameters** ()

**class** `cogdl.layers.se_layer.SELayer` (*in\_channels, se\_channels*)

Bases: `torch.nn.modules.module.Module`

Squeeze-and-excitation networks

**forward** (*x*)

## 2.12.2 GCC module

## 2.12.3 GPT-GNN module

## 2.12.4 Link Prediction module

## 2.12.5 PPRGo module

## 2.12.6 ProNE module

## 2.12.7 SRGCN module

## 2.12.8 Strategies module

## 2.13 options

`cogdl.options.add_dataset_args` (*parser*)

`cogdl.options.add_model_args` (*parser*)

`cogdl.options.add_task_args` (*parser*)

`cogdl.options.add_trainer_args` (*parser*)

`cogdl.options.get_default_args` (*task: str, dataset, model, \*\*kwargs*)

`cogdl.options.get_display_data_parser` ()

`cogdl.options.get_download_data_parser` ()

`cogdl.options.get_parser` ()

`cogdl.options.get_task_model_args` (*task, model=None*)

`cogdl.options.get_training_parser` ()

`cogdl.options.parse_args_and_arch` (*parser, args*)

## 2.14 utils

**class** cogdl.utils.utils.**ArgClass**

Bases: `object`

cogdl.utils.utils.**alias\_draw** (*J, q*)

Draw sample from a non-uniform discrete distribution using alias sampling.

cogdl.utils.utils.**alias\_setup** (*probs*)

Compute utility lists for non-uniform sampling from discrete distributions. Refer to <https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-sampling-with-many-discrete-outcomes/> for details

cogdl.utils.utils.**batch\_max\_pooling** (*x, batch*)

cogdl.utils.utils.**batch\_mean\_pooling** (*x, batch*)

cogdl.utils.utils.**batch\_sum\_pooling** (*x, batch*)

cogdl.utils.utils.**build\_args\_from\_dict** (*dic*)

cogdl.utils.utils.**cycle\_index** (*num, shift*)

cogdl.utils.utils.**download\_url** (*url, folder, name=None, log=True*)

Downloads the content of an URL to a specific folder.

**Args:** *url* (string): The url. *folder* (string): The folder. *name* (string): saved filename. *log* (bool, optional): If `False`, will not print anything to the console. (default: `True`)

cogdl.utils.utils.**get\_activation** (*act: str*)

cogdl.utils.utils.**get\_memory\_usage** (*print\_info=False*)

Get accurate gpu memory usage by querying torch runtime

cogdl.utils.utils.**get\_norm\_layer** (*norm: str, channels: int*)

**Args:**

**norm: str** type of normalization: *layernorm, batchnorm, instancenorm*

**channels: int** size of features for normalization

cogdl.utils.utils.**identity\_act** (*input, inplace=True*)

cogdl.utils.utils.**makedirs** (*path*)

cogdl.utils.utils.**print\_result** (*results, datasets, model\_name*)

cogdl.utils.utils.**set\_random\_seed** (*seed*)

cogdl.utils.utils.**split\_dataset\_general** (*dataset, args*)

cogdl.utils.utils.**tabulate\_results** (*results\_dict*)

cogdl.utils.utils.**untar** (*path, fname, deleteTar=True*)

Unpacks the given archive file to the same directory, then (by default) deletes the archive file.

cogdl.utils.utils.**update\_args\_from\_dict** (*args, dic*)

cogdl.utils.evaluator.**accuracy** (*y\_pred, y\_true*)

cogdl.utils.evaluator.**bce\_with\_logits\_loss** (*y\_pred, y\_true, reduction='mean'*)

cogdl.utils.evaluator.**cross\_entropy\_loss** (*y\_pred, y\_true*)

cogdl.utils.evaluator.**multiclass\_f1** (*y\_pred, y\_true*)



`cogdl.utils.evaluator.multilabel_f1` (*y\_pred*, *y\_true*, *sigmoid=False*)

**class** `cogdl.utils.sampling.RandomWalker` (*adj=None*, *num\_nodes=None*)

Bases: `object`

**build\_up** (*adj*, *num\_nodes*)

**walk** (*start*, *walk\_length*, *restart\_p=0.0*)

`cogdl.utils.sampling.random_walk`

**Parameters:** *start* : `np.array(dtype=np.int32)` *length* : `int` *indptr* : `np.array(dtype=np.int32)` *indices* : `np.array(dtype=np.int32)` *p* : `float`

**Return:** `list(np.array(dtype=np.int32))`

## 2.15 experiments

**class** `cogdl.experiments.AutoML` (*task*, *dataset*, *model*, *n\_trials=3*, *\*\*kwargs*)

Bases: `object`

**Args:** *func\_search*: function to obtain hyper-parameters to search

**run** ()

`cogdl.experiments.auto_experiment` (*task: str*, *dataset*, *model*, *\*\*kwargs*)

`cogdl.experiments.check_task_dataset_model_match` (*task*, *variants*)

`cogdl.experiments.experiment` (*task: str*, *dataset*, *model*, *\*\*kwargs*)

`cogdl.experiments.gen_variants` (*\*\*items*)

`cogdl.experiments.output_results` (*results\_dict*, *tablefmt='github'*)

`cogdl.experiments.raw_experiment` (*task: str*, *dataset*, *model*, *\*\*kwargs*)

`cogdl.experiments.set_best_config` (*args*)

`cogdl.experiments.train` (*args*)

`cogdl.experiments.variant_args_generator` (*args*, *variants*)

Form variants as group with size of *num\_workers*

## 2.16 pipelines

**class** `cogdl.pipelines.DatasetPipeline` (*app: str*, *\*\*kwargs*)

Bases: `cogdl.pipelines.Pipeline`

**class** `cogdl.pipelines.DatasetStatsPipeline` (*app: str*, *\*\*kwargs*)

Bases: `cogdl.pipelines.DatasetPipeline`

**class** `cogdl.pipelines.DatasetVisualPipeline` (*app: str*, *\*\*kwargs*)

Bases: `cogdl.pipelines.DatasetPipeline`

**class** `cogdl.pipelines.GenerateEmbeddingPipeline` (*app: str*, *model: str*, *\*\*kwargs*)

Bases: `cogdl.pipelines.Pipeline`

**class** `cogdl.pipelines.OAGBertInferencePipepline` (*app: str*, *model: str*, *\*\*kwargs*)

Bases: `cogdl.pipelines.Pipeline`

```
class cogdl.pipelines.Pipeline (app: str, **kwargs)  
    Bases: object
```

```
class cogdl.pipelines.RecommendationPipepline (app: str, model: str, **kwargs)  
    Bases: cogdl.pipelines.Pipeline
```

```
cogdl.pipelines.check_app (app: str)
```

```
cogdl.pipelines.pipeline (app: str, **kwargs) → cogdl.pipelines.Pipeline
```

## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### C

`cogdl.data`, 31  
`cogdl.datasets`, 47  
`cogdl.datasets.gatne`, 35  
`cogdl.datasets.gcc_data`, 36  
`cogdl.datasets.gtn_data`, 37  
`cogdl.datasets.han_data`, 37  
`cogdl.datasets.kg_data`, 38  
`cogdl.datasets.matlab_matrix`, 39  
`cogdl.datasets.ogb`, 41  
`cogdl.datasets.strategies_data`, 42  
`cogdl.datasets.tu_data`, 45  
`cogdl.experiments`, 93  
`cogdl.layers.deepergcn_layer`, 86  
`cogdl.layers.disengcn_layer`, 88  
`cogdl.layers.gat_layer`, 85  
`cogdl.layers.gcn_layer`, 85  
`cogdl.layers.gcnii_layer`, 86  
`cogdl.layers.gin_layer`, 86  
`cogdl.layers.han_layer`, 88  
`cogdl.layers.mixhop_layer`, 90  
`cogdl.layers.mlp_layer`, 88  
`cogdl.layers.pprgo_layer`, 89  
`cogdl.layers.rgcn_layer`, 89  
`cogdl.layers.sage_layer`, 85  
`cogdl.layers.saint_layer`, 90  
`cogdl.layers.se_layer`, 91  
`cogdl.layers.sgc_layer`, 90  
`cogdl.models`, 84  
`cogdl.models.base_model`, 52  
`cogdl.models.supervised_model`, 53  
`cogdl.options`, 91  
`cogdl.pipelines`, 93  
`cogdl.tasks`, 52  
`cogdl.tasks.attributed_graph_clustering`,  
51  
`cogdl.tasks.base_task`, 47  
`cogdl.tasks.graph_classification`, 51  
`cogdl.tasks.heterogeneous_node_classification`,  
48  
`cogdl.tasks.link_prediction`, 49  
`cogdl.tasks.multiplex_link_prediction`,  
51  
`cogdl.tasks.multiplex_node_classification`,  
49  
`cogdl.tasks.node_classification`, 48  
`cogdl.tasks.pretrain`, 52  
`cogdl.tasks.similarity_search`, 52  
`cogdl.tasks.unsupervised_graph_classification`,  
51  
`cogdl.tasks.unsupervised_node_classification`,  
48  
`cogdl.utils.evaluator`, 92  
`cogdl.utils.sampling`, 93  
`cogdl.utils.utils`, 92



## A

- accuracy() (in module *cogdl.utils.evaluator*), 92
- ACM\_GTNDataset (class in *cogdl.datasets.gtn\_data*), 37
- ACM\_HANDataset (class in *cogdl.datasets.han\_data*), 37
- add\_args() (*cogdl.data.Dataset* static method), 34
- add\_args() (*cogdl.models.base\_model.BaseModel* static method), 52
- add\_args() (*cogdl.models.emb.deepwalk.DeepWalk* static method), 56
- add\_args() (*cogdl.models.emb.dgk.DeepGraphKernel* static method), 57
- add\_args() (*cogdl.models.emb.dngr.DNGR* static method), 58
- add\_args() (*cogdl.models.emb.gatne.GATNE* static method), 56
- add\_args() (*cogdl.models.emb.graph2vec.Graph2Vec* static method), 59
- add\_args() (*cogdl.models.emb.grarep.GraRep* static method), 57
- add\_args() (*cogdl.models.emb.hin2vec.Hin2vec* static method), 54
- add\_args() (*cogdl.models.emb.hope.HOPE* static method), 53
- add\_args() (*cogdl.models.emb.line.LINE* static method), 62
- add\_args() (*cogdl.models.emb.metapath2vec.Metapath2vec* static method), 59
- add\_args() (*cogdl.models.emb.netmf.NetMF* static method), 55
- add\_args() (*cogdl.models.emb.netsmf.NetSMF* static method), 61
- add\_args() (*cogdl.models.emb.node2vec.Node2vec* static method), 60
- add\_args() (*cogdl.models.emb.prone.ProNE* static method), 63
- add\_args() (*cogdl.models.emb.pronepp.ProNEPP* static method), 58
- add\_args() (*cogdl.models.emb.pte.PTE* static method), 61
- add\_args() (*cogdl.models.emb.sdne.SDNE* static method), 62
- add\_args() (*cogdl.models.emb.spectral.Spectral* static method), 54
- add\_args() (*cogdl.models.nn.compgcn.LinkPredictCompGCN* static method), 67
- add\_args() (*cogdl.models.nn.deepergcn.DeeperGCN* static method), 78
- add\_args() (*cogdl.models.nn.dgi.DGIModel* static method), 63
- add\_args() (*cogdl.models.nn.dgl\_gcc.GCC* static method), 83
- add\_args() (*cogdl.models.nn.dgl\_jknet.JKNet* static method), 74
- add\_args() (*cogdl.models.nn.diffpool.DiffPool* static method), 70
- add\_args() (*cogdl.models.nn.disengcn.DisenGCN* static method), 80
- add\_args() (*cogdl.models.nn.drgat.DrGAT* static method), 78
- add\_args() (*cogdl.models.nn.drgcn.DrGCN* static method), 67
- add\_args() (*cogdl.models.nn.dropedge\_gcn.DropEdge\_GCN* static method), 80
- add\_args() (*cogdl.models.nn.gat.GAT* static method), 72
- add\_args() (*cogdl.models.nn.gcn.TKipfGCN* static method), 65
- add\_args() (*cogdl.models.nn.gcnii.GCNII* static method), 70
- add\_args() (*cogdl.models.nn.gcnmix.GCNMix* static method), 69
- add\_args() (*cogdl.models.nn.gdc\_gcn.GDC\_GCN* static method), 65
- add\_args() (*cogdl.models.nn.gin.GIN* static method), 75
- add\_args() (*cogdl.models.nn.grace.GRACE* static method), 73

`add_args()` (*cogdl.models.nn.grand.Grand* static method), 76  
`add_args()` (*cogdl.models.nn.graphsage.Graphsage* static method), 66  
`add_args()` (*cogdl.models.nn.han.HAN* static method), 72  
`add_args()` (*cogdl.models.nn.infograph.InfoGraph* static method), 79  
`add_args()` (*cogdl.models.nn.mixhop.MixHop* static method), 72  
`add_args()` (*cogdl.models.nn.mlp.MLP* static method), 81  
`add_args()` (*cogdl.models.nn.mvgrl.MVGRL* static method), 63  
`add_args()` (*cogdl.models.nn.patchy\_san.PatchySAN* static method), 64  
`add_args()` (*cogdl.models.nn.pppp.PPP* static method), 73  
`add_args()` (*cogdl.models.nn.pprgo.PPRGo* static method), 74  
`add_args()` (*cogdl.models.nn.pyg\_cheb.Chebyshev* static method), 64  
`add_args()` (*cogdl.models.nn.pyg\_dgcnn.DGCNN* static method), 76  
`add_args()` (*cogdl.models.nn.pyg\_gcn.GCN* static method), 71  
`add_args()` (*cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN* static method), 68  
`add_args()` (*cogdl.models.nn.pyg\_graph\_unet.GraphUnet* static method), 68  
`add_args()` (*cogdl.models.nn.pyg\_gtn.GTN* static method), 77  
`add_args()` (*cogdl.models.nn.pyg\_hgpsl.HGPSL* static method), 66  
`add_args()` (*cogdl.models.nn.pyg\_sagpool.SAGPoolNetwork* static method), 84  
`add_args()` (*cogdl.models.nn.pyg\_srgcn.SRGCN* static method), 82  
`add_args()` (*cogdl.models.nn.rgcn.LinkPredictRGCN* static method), 77  
`add_args()` (*cogdl.models.nn.sgc.sgc* static method), 81  
`add_args()` (*cogdl.models.nn.sign.MLP* static method), 71  
`add_args()` (*cogdl.models.nn.sortpool.SortPool* static method), 82  
`add_args()` (*cogdl.models.nn.stpgnn.stpgnn* static method), 81  
`add_args()` (*cogdl.models.nn.unsup\_graphsage.SAGE* static method), 83  
`add_args()` (*cogdl.tasks.attributed\_graph\_clustering.AttributedGraphClustering* static method), 51  
`add_args()` (*cogdl.tasks.base\_task.BaseTask* static method), 47  
`add_args()` (*cogdl.tasks.graph\_classification.GraphClassification* static method), 51  
`add_args()` (*cogdl.tasks.heterogeneous\_node\_classification.HeterogeneousNodeClassification* static method), 48  
`add_args()` (*cogdl.tasks.link\_prediction.LinkPrediction* static method), 50  
`add_args()` (*cogdl.tasks.multiplex\_link\_prediction.MultiplexLinkPrediction* static method), 51  
`add_args()` (*cogdl.tasks.multiplex\_node\_classification.MultiplexNodeClassification* static method), 49  
`add_args()` (*cogdl.tasks.node\_classification.NodeClassification* static method), 48  
`add_args()` (*cogdl.tasks.pretrain.PretrainTask* static method), 52  
`add_args()` (*cogdl.tasks.similarity\_search.SimilaritySearch* static method), 52  
`add_args()` (*cogdl.tasks.unsupervised\_graph\_classification.UnsupervisedGraphClassification* static method), 51  
`add_args()` (*cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification* static method), 48  
`add_dataset_args()` (in module *cogdl.options*), 91  
`add_model_args()` (in module *cogdl.options*), 91  
`add_remaining_self_loops()` (*cogdl.data.Adjacency* method), 33  
`add_remaining_self_loops()` (*cogdl.data.Graph* method), 31  
`add_reverse_edges()` (*cogdl.models.nn.compvcn.LinkPredictCompVCN* method), 67  
`add_task_args()` (in module *cogdl.options*), 91  
`add_trainer_args()` (in module *cogdl.options*), 91  
`adj_pow_x()` (*cogdl.layers.mixhop\_layer.MixHopLayer* method), 90  
*Adjacency* (class in *cogdl.data*), 32  
`adjacency_pooling_forward()` (*cogdl.models.nn.diffpool.DiffPool* method), 70  
`alias_draw()` (in module *cogdl.utils.utils*), 92  
`alias_setup()` (in module *cogdl.utils.utils*), 92  
*AmazonDataset* (class in *cogdl.datasets.gatne*), 35  
`apply_to_device()` (*cogdl.datasets.gtn\_data.GTNDataset* method), 37  
`apply_to_device()` (*cogdl.datasets.han\_data.HANDataset* method), 38  
*ArgClass* (class in *cogdl.utils.utils*), 92  
*AttentionLayer* (class in *cogdl.layers.han\_layer*), 88  
*AttributedGraphClustering* (class in *cogdl.tasks.attributed\_graph\_clustering*), 51  
`augment()` (*cogdl.models.nn.dgi.DGIModel* method), 63  
`augment()` (*cogdl.models.nn.grace.GRACE* method), 63



- 73  
 augment() (*cogdl.models.nn.mvgrl.MVGRL method*), 63  
 auto\_experiment() (*in module cogdl.experiments*), 93  
 AutoML (*class in cogdl.experiments*), 93
- ## B
- BACEDataset (*class in cogdl.datasets.strategies\_data*), 42  
 BaseModel (*class in cogdl.models.base\_model*), 52  
 BaseTask (*class in cogdl.tasks.base\_task*), 47  
 basis\_forward() (*cogdl.layers.rgcn\_layer.RGCNLayer method*), 90  
 Batch (*class in cogdl.data*), 33  
 batch\_max\_pooling() (*in module cogdl.utils.utils*), 92  
 batch\_mean\_pooling() (*in module cogdl.utils.utils*), 92  
 batch\_sum\_pooling() (*in module cogdl.utils.utils*), 92  
 BatchAE (*class in cogdl.datasets.strategies\_data*), 43  
 batched\_loss() (*cogdl.models.nn.grace.GRACE method*), 73  
 BatchMasking (*class in cogdl.datasets.strategies\_data*), 43  
 BatchSubstructContext (*class in cogdl.datasets.strategies\_data*), 43  
 BBBPDataset (*class in cogdl.datasets.strategies\_data*), 42  
 bce\_with\_logits\_loss() (*in module cogdl.utils.evaluator*), 92  
 bdd\_forward() (*cogdl.layers.rgcn\_layer.RGCNLayer method*), 90  
 BidirectionalOneShotIterator (*class in cogdl.datasets.kg\_data*), 38  
 BioDataset (*class in cogdl.datasets.strategies\_data*), 44  
 BlogcatalogDataset (*class in cogdl.datasets.matlab\_matrix*), 39  
 BondEncoder (*class in cogdl.layers.deepergcn\_layer*), 86  
 build\_args\_from\_dict() (*in module cogdl.utils.utils*), 92  
 build\_batch() (*in module cogdl.datasets.strategies\_data*), 45  
 build\_dataset() (*in module cogdl.datasets*), 47  
 build\_dataset\_from\_name() (*in module cogdl.datasets*), 47  
 build\_dataset\_from\_path() (*in module cogdl.datasets*), 47  
 build\_model() (*cogdl.models.nn.patchy\_san.PatchySAN method*), 64  
 build\_model() (*in module cogdl.models*), 84  
 build\_model\_from\_args() (*cogdl.models.base\_model.BaseModel class method*), 52  
 build\_model\_from\_args() (*cogdl.models.emb.deepwalk.DeepWalk class method*), 56  
 build\_model\_from\_args() (*cogdl.models.emb.dgk.DeepGraphKernel class method*), 57  
 build\_model\_from\_args() (*cogdl.models.emb.dngr.DNGR class method*), 58  
 build\_model\_from\_args() (*cogdl.models.emb.gatne.GATNE class method*), 56  
 build\_model\_from\_args() (*cogdl.models.emb.graph2vec.Graph2Vec class method*), 59  
 build\_model\_from\_args() (*cogdl.models.emb.grarep.GraRep class method*), 57  
 build\_model\_from\_args() (*cogdl.models.emb.hin2vec.Hin2vec class method*), 54  
 build\_model\_from\_args() (*cogdl.models.emb.hope.HOPE class method*), 53  
 build\_model\_from\_args() (*cogdl.models.emb.line.LINE class method*), 62  
 build\_model\_from\_args() (*cogdl.models.emb.metapath2vec.Metapath2vec class method*), 59  
 build\_model\_from\_args() (*cogdl.models.emb.netmf.NetMF class method*), 55  
 build\_model\_from\_args() (*cogdl.models.emb.netsmf.NetSMF class method*), 61  
 build\_model\_from\_args() (*cogdl.models.emb.node2vec.Node2vec class method*), 60  
 build\_model\_from\_args() (*cogdl.models.emb.prone.ProNE class method*), 63  
 build\_model\_from\_args() (*cogdl.models.emb.pronepp.ProNEPP class method*), 58  
 build\_model\_from\_args() (*cogdl.models.emb.ptc.PTE class method*), 61  
 build\_model\_from\_args() (*cogdl.models.emb.sdne.SDNE class method*), 62  
 build\_model\_from\_args() (*cogdl.models.emb.sdne.SDNE class method*), 62  
 build\_model\_from\_args() (*cogdl.models.emb.sdne.SDNE class method*), 62

<i>(cogdl.models.emb.spectral.Spectral</i> class method), 54	<i>(cogdl.models.nn.grand.Grand</i> class method), 76
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.compgcn.LinkPredictCompGCN</i> class method), 67	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.graphsage.Graphsage</i> class method), 66
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.deepergcgn.DeeperGCN</i> class method), 78	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.han.HAN</i> class method), 72
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.dgi.DGIModel</i> class method), 63	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.infograph.InfoGraph</i> class method), 79
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.dgl_gcc.GCC</i> class method), 83	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.mixhop.MixHop</i> class method), 72
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.dgl_jknet.JKNet</i> class method), 74	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.mlp.MLP</i> class method), 81
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.diffpool.DiffPool</i> class method), 70	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.mvgrl.MVGRL</i> class method), 64
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.disengcn.DisenGCN</i> class method), 80	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.patchy_san.PatchySAN</i> class method), 64
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.drgat.DrGAT</i> class method), 78	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pppnp.PPPNP</i> class method), 73
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.drgcn.DrGCN</i> class method), 68	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pprgo.PPRGo</i> class method), 74
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.droppedge_gcgn.DropEdge_GCgn</i> class method), 80	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_cheb.Chebyshev</i> class method), 65
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.gat.GAT</i> class method), 72	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_dgcnn.DGCNN</i> class method), 76
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.gcn.TKipfGCN</i> class method), 65	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_gcgn.GCgn</i> class method), 71
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.gcnii.GCNII</i> class method), 70	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_gpt_gnn.GPT_GNN</i> class method), 68
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.gcnmix.GCNMix</i> class method), 69	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_graph_unet.GraphUnet</i> class method), 68
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.gdc_gcgn.GDC_GCgn</i> class method), 66	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_gtm.GTM</i> class method), 77
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.gin.GIN</i> class method), 75	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_hgpsl.HGPSL</i> class method), 66
<i>build_model_from_args()</i> ( <i>cogdl.models.nn.grace.GRACE</i> class method), 73	<i>build_model_from_args()</i> ( <i>cogdl.models.nn.pyg_sagpool.SAGPoolNetwork</i> class method), 84
<i>build_model_from_args()</i>	<i>build_model_from_args()</i>

- (*cogdl.models.nn.pyg\_srgcn.SRGCN class method*), 82
- build\_model\_from\_args()* (*cogdl.models.nn.rgcn.LinkPredictRGCN class method*), 77
- build\_model\_from\_args()* (*cogdl.models.nn.sgc.sgc class method*), 81
- build\_model\_from\_args()* (*cogdl.models.nn.sign.MLP class method*), 71
- build\_model\_from\_args()* (*cogdl.models.nn.sortpool.SortPool class method*), 82
- build\_model\_from\_args()* (*cogdl.models.nn.stpgnn.stpgnn class method*), 82
- build\_model\_from\_args()* (*cogdl.models.nn.unsup\_graphsage.SAGE class method*), 83
- build\_task()* (*in module cogdl.tasks*), 52
- build\_up()* (*cogdl.utils.sampling.RandomWalker method*), 93
- ## C
- cat()* (*in module cogdl.datasets.tu\_data*), 46
- cat\_dim()* (*cogdl.datasets.strategies\_data.BatchAE method*), 43
- cat\_dim()* (*cogdl.datasets.strategies\_data.BatchSubstructureContext method*), 43
- Chebyshev* (*class in cogdl.models.nn.pyg\_cheb*), 64
- check\_app()* (*in module cogdl.pipelines*), 94
- check\_task\_dataset\_model\_match()* (*in module cogdl.experiments*), 93
- ChemExtractSubstructureContextPair* (*class in cogdl.datasets.strategies\_data*), 44
- clone()* (*cogdl.data.Adjacency method*), 33
- clone()* (*cogdl.data.Graph method*), 31
- coalesce()* (*in module cogdl.datasets.tu\_data*), 46
- cogdl.data* (*module*), 31
- cogdl.datasets* (*module*), 47
- cogdl.datasets.gatne* (*module*), 35
- cogdl.datasets.gcc\_data* (*module*), 36
- cogdl.datasets.gtn\_data* (*module*), 37
- cogdl.datasets.han\_data* (*module*), 37
- cogdl.datasets.kg\_data* (*module*), 38
- cogdl.datasets.matlab\_matrix* (*module*), 39
- cogdl.datasets.ogb* (*module*), 41
- cogdl.datasets.strategies\_data* (*module*), 42
- cogdl.datasets.tu\_data* (*module*), 45
- cogdl.experiments* (*module*), 93
- cogdl.layers.deepergcn\_layer* (*module*), 86
- cogdl.layers.disengcn\_layer* (*module*), 88
- cogdl.layers.gat\_layer* (*module*), 85
- cogdl.layers.gcn\_layer* (*module*), 85
- cogdl.layers.gcnii\_layer* (*module*), 86
- cogdl.layers.gin\_layer* (*module*), 86
- cogdl.layers.han\_layer* (*module*), 88
- cogdl.layers.mixhop\_layer* (*module*), 90
- cogdl.layers.mlp\_layer* (*module*), 88
- cogdl.layers.pprgo\_layer* (*module*), 89
- cogdl.layers.rgcn\_layer* (*module*), 89
- cogdl.layers.sage\_layer* (*module*), 85
- cogdl.layers.saint\_layer* (*module*), 90
- cogdl.layers.se\_layer* (*module*), 91
- cogdl.layers.sgc\_layer* (*module*), 90
- cogdl.models* (*module*), 84
- cogdl.models.base\_model* (*module*), 52
- cogdl.models.supervised\_model* (*module*), 53
- cogdl.options* (*module*), 91
- cogdl.pipelines* (*module*), 93
- cogdl.tasks* (*module*), 52
- cogdl.tasks.attributed\_graph\_clustering* (*module*), 51
- cogdl.tasks.base\_task* (*module*), 47
- cogdl.tasks.graph\_classification* (*module*), 51
- cogdl.tasks.heterogeneous\_node\_classification* (*module*), 48
- cogdl.tasks.link\_prediction* (*module*), 49
- cogdl.tasks.multiplex\_link\_prediction* (*module*), 51
- cogdl.tasks.multiplex\_node\_classification* (*module*), 49
- cogdl.tasks.node\_classification* (*module*), 48
- cogdl.tasks.pretrain* (*module*), 52
- cogdl.tasks.similarity\_search* (*module*), 52
- cogdl.tasks.unsupervised\_graph\_classification* (*module*), 51
- cogdl.tasks.unsupervised\_node\_classification* (*module*), 48
- cogdl.utils.evaluator* (*module*), 92
- cogdl.utils.sampling* (*module*), 93
- cogdl.utils.utils* (*module*), 92
- col\_indices* (*cogdl.data.Graph attribute*), 31
- col\_norm()* (*cogdl.data.Adjacency method*), 33
- col\_norm()* (*cogdl.data.Graph method*), 31
- CollabDataset* (*class in cogdl.datasets.tu\_data*), 45
- collate\_fn()* (*cogdl.data.DataLoader static method*), 35
- collate\_fn()* (*cogdl.datasets.kg\_data.TestDataset static method*), 39
- collate\_fn()* (*cogdl.datasets.kg\_data.TrainDataset static method*), 39
- Complex* (*class in cogdl.models.emb.complex*), 60

- `consis_loss()` (*cogdl.models.nn.grand.Grand method*), 76  
`contrastive_loss()` (*cogdl.models.nn.grace.GRACE method*), 73  
`convert()` (*in module cogdl.datasets.strategies\_data*), 45  
`convert_csr()` (*cogdl.data.Adjacency method*), 33  
`count_frequency()` (*cogdl.datasets.kg\_data.TrainDataset static method*), 39  
`cross_entropy_loss()` (*in module cogdl.utils.evaluator*), 92  
`csr_subgraph()` (*cogdl.data.Graph method*), 31  
`cumsum()` (*cogdl.data.Batch method*), 33  
`cumsum()` (*cogdl.datasets.strategies\_data.BatchMasking method*), 43  
`cumsum()` (*cogdl.datasets.strategies\_data.BatchSubstructContext method*), 43  
`cycle_index()` (*in module cogdl.utils.utils*), 92
- ## D
- `DataLoader` (*class in cogdl.data*), 35  
`DataLoaderAE` (*class in cogdl.datasets.strategies\_data*), 44  
`DataLoaderSubstructContext` (*class in cogdl.datasets.strategies\_data*), 44  
`Dataset` (*class in cogdl.data*), 34  
`DatasetPipeline` (*class in cogdl.pipelines*), 93  
`DatasetStatsPipeline` (*class in cogdl.pipelines*), 93  
`DatasetVisualPipeline` (*class in cogdl.pipelines*), 93  
`DBLP_GTNDataset` (*class in cogdl.datasets.gtn\_data*), 37  
`DBLP_HANDataset` (*class in cogdl.datasets.han\_data*), 37  
`DblpNEDataset` (*class in cogdl.datasets.matlab\_matrix*), 39  
`DeeperGCN` (*class in cogdl.models.nn.deepergcn*), 78  
`DeepGraphKernel` (*class in cogdl.models.emb.dgk*), 57  
`DeepWalk` (*class in cogdl.models.emb.deepwalk*), 55  
`degrees` (*cogdl.data.Adjacency attribute*), 33  
`degrees()` (*cogdl.data.Graph method*), 31  
`device` (*cogdl.data.Adjacency attribute*), 33  
`DGCNN` (*class in cogdl.models.nn.pyg\_dgcnn*), 75  
`DGIModel` (*class in cogdl.models.nn.dgi*), 63  
`DiffPool` (*class in cogdl.models.nn.diffpool*), 69  
`DisenGCN` (*class in cogdl.models.nn.disengcn*), 80  
`DisenGCNLayer` (*class in cogdl.layers.disengcn\_layer*), 88  
`DistMult` (*class in cogdl.models.emb.distmult*), 55  
`divide_data()` (*in cogdl.tasks.link\_prediction module*), 50  
`DNGR` (*class in cogdl.models.emb.dngr*), 58  
`download()` (*cogdl.data.Dataset method*), 34  
`download()` (*cogdl.datasets.gatne.GatneDataset method*), 35  
`download()` (*cogdl.datasets.gcc\_data.Edgelist method*), 36  
`download()` (*cogdl.datasets.gcc\_data.GCCDataset method*), 36  
`download()` (*cogdl.datasets.gtn\_data.GTNDataset method*), 37  
`download()` (*cogdl.datasets.han\_data.HANDataset method*), 38  
`download()` (*cogdl.datasets.kg\_data.KnowledgeGraphDataset method*), 39  
`download()` (*cogdl.datasets.matlab\_matrix.MatlabMatrix method*), 40  
`download()` (*cogdl.datasets.matlab\_matrix.NetworkEmbeddingCMTYD method*), 40  
`download()` (*cogdl.datasets.strategies\_data.BACEDataset method*), 42  
`download()` (*cogdl.datasets.strategies\_data.BBBPDataset method*), 42  
`download()` (*cogdl.datasets.strategies\_data.BioDataset method*), 44  
`download()` (*cogdl.datasets.strategies\_data.MoleculeDataset method*), 44  
`download()` (*cogdl.datasets.tu\_data.TUDataset method*), 46  
`download_url()` (*in module cogdl.utils.utils*), 92  
`DrGAT` (*class in cogdl.models.nn.drgat*), 78  
`DrGCN` (*class in cogdl.models.nn.drgcn*), 67  
`drop_adj()` (*cogdl.models.nn.grace.GRACE method*), 73  
`drop_feature()` (*cogdl.models.nn.grace.GRACE method*), 73  
`DropEdge_GCN` (*class in cogdl.models.nn.dropedge\_gcn*), 79  
`dropNode()` (*cogdl.models.nn.grand.Grand method*), 76
- ## E
- `edge_attr` (*cogdl.data.Graph attribute*), 31  
`edge_attr_size` (*cogdl.data.Dataset attribute*), 34  
`edge_attr_size` (*cogdl.datasets.ogb.OGBProteinsDataset attribute*), 42  
`edge_index` (*cogdl.data.Adjacency attribute*), 33  
`edge_index` (*cogdl.data.Graph attribute*), 31  
`edge_subgraph()` (*cogdl.data.Graph method*), 31  
`edge_types` (*cogdl.data.Graph attribute*), 32  
`edge_weight` (*cogdl.data.Graph attribute*), 32  
`EdgeEncoder` (*class in cogdl.layers.deepergcn\_layer*), 86

- Edgelist (class in cogdl.datasets.gcc\_data), 36
- embed () (cogdl.models.nn.dgi.DGIModel method), 63
- embed () (cogdl.models.nn.gcn.TKipfGCN method), 65
- embed () (cogdl.models.nn.grace.GRACE method), 73
- embed () (cogdl.models.nn.mvgrl.MVGRL method), 64
- embed () (cogdl.models.nn.unsup\_graphsage.SAGE method), 84
- enhance\_emb () (cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification method), 48
- ENZYMES (class in cogdl.datasets.tu\_data), 45
- eval () (cogdl.data.Graph method), 32
- evaluate () (cogdl.models.nn.han.HAN method), 72
- evaluate () (cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN method), 68
- evaluate () (cogdl.models.nn.pyg\_gtn.GTN method), 77
- evaluate () (cogdl.models.supervised\_model.SupervisedHeterogeneousNodeClassificationModel method), 53
- evaluate () (in module cogdl.tasks.link\_prediction), 50
- evaluate () (in module cogdl.tasks.multiplex\_link\_prediction), 51
- experiment () (in module cogdl.experiments), 93
- ExtractSubstructureContextPair (class in cogdl.datasets.strategies\_data), 44
- ## F
- FB13Datset (class in cogdl.datasets.kg\_data), 38
- FB13SDatset (class in cogdl.datasets.kg\_data), 38
- FB15k237Datset (class in cogdl.datasets.kg\_data), 38
- FB15kDatset (class in cogdl.datasets.kg\_data), 38
- feature\_extractor () (cogdl.models.emb.dgk.DeepGraphKernel static method), 57
- feature\_extractor () (cogdl.models.emb.graph2vec.Graph2Vec static method), 59
- FlickrDataset (class in cogdl.datasets.matlab\_matrix), 39
- forward () (cogdl.layers.deepergcn\_layer.BondEncoder method), 86
- forward () (cogdl.layers.deepergcn\_layer.EdgeEncoder method), 86
- forward () (cogdl.layers.deepergcn\_layer.GENConv method), 87
- forward () (cogdl.layers.deepergcn\_layer.ResGNNLayer method), 87
- forward () (cogdl.layers.disengcn\_layer.DisenGCNLayer method), 88
- forward () (cogdl.layers.gat\_layer.GATLayer method), 85
- forward () (cogdl.layers.gcn\_layer.GCNLayer method), 85
- forward () (cogdl.layers.gcnii\_layer.GCNIILayer method), 86
- forward () (cogdl.layers.gin\_layer.GINLayer method), 86
- forward () (cogdl.layers.han\_layer.AttentionLayer method), 88
- forward () (cogdl.layers.han\_layer.HANLayer method), 88
- forward () (cogdl.layers.mixhop\_layer.MixHopLayer method), 90
- forward () (cogdl.layers.mlp\_layer.MLP method), 89
- forward () (cogdl.layers.pprgo\_layer.LinearLayer method), 89
- forward () (cogdl.layers.pprgo\_layer.PPRGoLayer method), 89
- forward () (cogdl.layers.rgcn\_layer.RGCNLayer method), 85
- forward () (cogdl.layers.sage\_layer.SAGELayer method), 85
- forward () (cogdl.layers.saint\_layer.SAINTLayer method), 90
- forward () (cogdl.layers.se\_layer.SELayer method), 91
- forward () (cogdl.layers.sgc\_layer.SGCLayer method), 90
- forward () (cogdl.models.base\_model.BaseModel method), 53
- forward () (cogdl.models.emb.dgk.DeepGraphKernel method), 57
- forward () (cogdl.models.emb.graph2vec.Graph2Vec method), 59
- forward () (cogdl.models.nn.compvcn.LinkPredictCompVCN method), 67
- forward () (cogdl.models.nn.deepergcn.DeeperGCN method), 78
- forward () (cogdl.models.nn.dgi.DGIModel method), 63
- forward () (cogdl.models.nn.dgl\_jknet.JKNet method), 74
- forward () (cogdl.models.nn.diffpool.DiffPool method), 70
- forward () (cogdl.models.nn.disengcn.DisenGCN method), 80
- forward () (cogdl.models.nn.drgat.DrGAT method), 78
- forward () (cogdl.models.nn.drgcn.DrGCN method), 68
- forward () (cogdl.models.nn.dropedge\_gcn.DropEdge\_GCIN method), 80
- forward () (cogdl.models.nn.gat.GAT method), 72
- forward () (cogdl.models.nn.gcn.TKipfGCN method), 65
- forward () (cogdl.models.nn.gcnii.GCNII method), 70
- forward () (cogdl.models.nn.gcnmix.GCNMix method), 69

- forward() (*cogdl.models.nn.gdc\_gcn.GDC\_GCN method*), 66
- forward() (*cogdl.models.nn.gin.GIN method*), 75
- forward() (*cogdl.models.nn.grace.GRACE method*), 73
- forward() (*cogdl.models.nn.grand.Grand method*), 77
- forward() (*cogdl.models.nn.graphsage.Graphsage method*), 66
- forward() (*cogdl.models.nn.han.HAN method*), 73
- forward() (*cogdl.models.nn.infograph.InfoGraph method*), 79
- forward() (*cogdl.models.nn.mixhop.MixHop method*), 72
- forward() (*cogdl.models.nn.mlp.MLP method*), 81
- forward() (*cogdl.models.nn.mvgrl.MVGRL method*), 64
- forward() (*cogdl.models.nn.patchy\_san.PatchySAN method*), 64
- forward() (*cogdl.models.nn.pnp.PPNP method*), 73
- forward() (*cogdl.models.nn.pprgo.PPRGo method*), 74
- forward() (*cogdl.models.nn.pyg\_cheb.Chebyshev method*), 65
- forward() (*cogdl.models.nn.pyg\_dgcnn.DGCNN method*), 76
- forward() (*cogdl.models.nn.pyg\_gcn.GCN method*), 71
- forward() (*cogdl.models.nn.pyg\_graph\_unet.GraphUnet method*), 68
- forward() (*cogdl.models.nn.pyg\_gtn.GTN method*), 77
- forward() (*cogdl.models.nn.pyg\_hgpsl.HGPSL method*), 66
- forward() (*cogdl.models.nn.pyg\_sagpool.SAGPoolNetwork method*), 84
- forward() (*cogdl.models.nn.pyg\_srgcn.SRGCN method*), 82
- forward() (*cogdl.models.nn.rgcn.LinkPredictRGCN method*), 78
- forward() (*cogdl.models.nn.sgc.sgc method*), 81
- forward() (*cogdl.models.nn.sign.MLP method*), 71
- forward() (*cogdl.models.nn.sortpool.SortPool method*), 82
- forward() (*cogdl.models.nn.unsup\_graphsage.SAGE method*), 84
- forward\_ema() (*cogdl.models.nn.gcnmix.GCNMix method*), 69
- from\_data\_list() (*cogdl.data.Batch static method*), 34
- from\_data\_list() (*cogdl.data.MultiGraphDataset static method*), 35
- from\_data\_list() (*cogdl.datasets.strategies\_data.BatchSubstructCon static method*), 43
- from\_data\_list() (*cogdl.datasets.strategies\_data.BatchMasking static method*), 43
- from\_data\_list() (*cogdl.datasets.strategies\_data.BatchSubstructCon static method*), 44
- from\_dict() (*cogdl.data.Adjacency static method*), 33
- from\_dict() (*cogdl.data.Graph static method*), 32
- from\_pyg\_data() (*cogdl.data.Graph static method*), 32
- ## G
- GAT (*class in cogdl.models.nn.gat*), 72
- GATLayer (*class in cogdl.layers.gat\_layer*), 85
- GATNE (*class in cogdl.models.emb.gatne*), 56
- GatneDataset (*class in cogdl.datasets.gatne*), 35
- GCC (*class in cogdl.models.nn.dgl\_gcc*), 83
- GCCDataset (*class in cogdl.datasets.gcc\_data*), 36
- GCN (*class in cogdl.models.nn.pyg\_gcn*), 71
- GCNII (*class in cogdl.models.nn.gcnii*), 70
- GCNIILayer (*class in cogdl.layers.gcnii\_layer*), 86
- GCNLayer (*class in cogdl.layers.gcn\_layer*), 85
- GCNMix (*class in cogdl.models.nn.gcnmix*), 69
- GDC\_GCN (*class in cogdl.models.nn.gdc\_gcn*), 65
- gen\_node\_pairs() (*in cogdl.tasks.link\_prediction module*), 50
- gen\_variants() (*in module cogdl.experiments*), 93
- GENConv (*class in cogdl.layers.deepergcn\_layer*), 87
- generate\_normalization() (*cogdl.data.Adjacency method*), 33
- GenerateEmbeddingPipeline (*class in cogdl.pipelines*), 93
- get() (*cogdl.data.Dataset method*), 34
- get() (*cogdl.data.MultiGraphDataset method*), 35
- get() (*cogdl.datasets.gatne.GatneDataset method*), 35
- get() (*cogdl.datasets.gcc\_data.Edgelist method*), 36
- get() (*cogdl.datasets.gcc\_data.GCCDataset method*), 36
- get() (*cogdl.datasets.gtn\_data.GTNDataset method*), 37
- get() (*cogdl.datasets.han\_data.HANDataset method*), 38
- get() (*cogdl.datasets.kg\_data.KnowledgeGraphDataset method*), 39
- get() (*cogdl.datasets.matlab\_matrix.MatlabMatrix method*), 40
- get() (*cogdl.datasets.matlab\_matrix.NetworkEmbeddingCMTYDataset method*), 40
- get() (*cogdl.datasets.ogb.MAGDataset method*), 41
- get() (*cogdl.datasets.ogb.OGBGDataset method*), 41
- get() (*cogdl.datasets.ogb.OGBNDataset method*), 41
- get\_activation() (*in module cogdl.utils.utils*), 92
- get\_default\_args() (*in module cogdl.options*), 91
- get\_denoised\_matrix() (*cogdl.models.emb.dngr.DNGR method*), 58

`get_display_data_parser()` (in module `cogdl.options`), 91  
`get_download_data_parser()` (in module `cogdl.options`), 91  
`get_emb()` (`cogdl.models.emb.dngr.DNGR` method), 58  
`get_embeddings()` (`cogdl.models.nn.pyg_gcn.GCN` method), 71  
`get_evaluator()` (`cogdl.data.Dataset` method), 34  
`get_evaluator()` (`cogdl.datasets.ogb.MAGDataset` method), 41  
`get_evaluator()` (`cogdl.datasets.ogb.OGBArxivDataset` method), 41  
`get_evaluator()` (`cogdl.datasets.ogb.OGBNDataset` method), 41  
`get_evaluator()` (`cogdl.datasets.ogb.OGBProteinsDataset` method), 42  
`get_link_labels()` (`cogdl.tasks.link_prediction.GNNHomoLinkPrediction` static method), 49  
`get_loader()` (`cogdl.datasets.ogb.OGBGDataset` method), 41  
`get_loss_fn()` (`cogdl.data.Dataset` method), 34  
`get_loss_fn()` (`cogdl.datasets.ogb.OGBNDataset` method), 41  
`get_loss_fn()` (`cogdl.datasets.ogb.OGBProteinsDataset` method), 42  
`get_memory_usage()` (in module `cogdl.utils.utils`), 92  
`get_norm_layer()` (in module `cogdl.utils.utils`), 92  
`get_optimizer()` (`cogdl.models.nn.gcnii.GCNII` method), 71  
`get_parser()` (in module `cogdl.options`), 91  
`get_ppmi_matrix()` (`cogdl.models.emb.dngr.DNGR` method), 58  
`get_score()` (in module `cogdl.tasks.link_prediction`), 50  
`get_score()` (in module `cogdl.tasks.multiplex_link_prediction`), 51  
`get_subset()` (`cogdl.datasets.ogb.OGBGDataset` method), 41  
`get_task_model_args()` (in module `cogdl.options`), 91  
`get_trainer()` (`cogdl.models.base_model.BaseModel` static method), 53  
`get_trainer()` (`cogdl.models.nn.dgi.DGIModel` static method), 63  
`get_trainer()` (`cogdl.models.nn.dgl_jknet.JKNet` static method), 74  
`get_trainer()` (`cogdl.models.nn.grace.GRACE` static method), 74  
`get_trainer()` (`cogdl.models.nn.graphsage.Graphsage` static method), 67  
`get_trainer()` (`cogdl.models.nn.mvgrl.MVGRL` static method), 64  
`get_trainer()` (`cogdl.models.nn.pprgo.PPRGo` static method), 75  
`get_trainer()` (`cogdl.models.nn.pyg_gpt_gnn.GPT_GNN` static method), 68  
`get_trainer()` (`cogdl.models.nn.unsup_graphsage.SAGE` static method), 84  
`get_trainer()` (`cogdl.models.supervised_model.SupervisedHeterogeneous` static method), 53  
`get_trainer()` (`cogdl.models.supervised_model.SupervisedHomogeneous` static method), 53  
`get_trainer()` (`cogdl.tasks.base_task.BaseTask` method), 47  
`get_training_parser()` (in module `cogdl.options`), 91  
`get_true_head_and_tail()` (`cogdl.datasets.kg_data.TrainDataset` static method), 39  
`get_weight()` (`cogdl.data.AdjacencyList` method), 33  
GIN (class in `cogdl.models.nn.gin`), 75  
GINLayer (class in `cogdl.layers.gin_layer`), 86  
GNNHomoLinkPrediction (class in `cogdl.tasks.link_prediction`), 49  
GPT\_GNN (class in `cogdl.models.nn.pyg_gpt_gnn`), 68  
GRACE (class in `cogdl.models.nn.grace`), 73  
Grand (class in `cogdl.models.nn.grand`), 76  
Graph (class in `cogdl.data`), 31  
Graph2Vec (class in `cogdl.models.emb.graph2vec`), 59  
`graph_classification_loss()` (`cogdl.models.base_model.BaseModel` method), 53  
`graph_classification_loss()` (`cogdl.models.nn.infograph.InfoGraph` method), 79  
`graph_classification_loss()` (`cogdl.models.nn.diffpool.DiffPool` method), 70  
`graph_data_obj_to_nx()` (in module `cogdl.datasets.strategies_data`), 45  
`graph_data_obj_to_nx_simple()` (in module `cogdl.datasets.strategies_data`), 45  
GraphClassification (class in `cogdl.tasks.graph_classification`), 51  
Graphsage (class in `cogdl.models.nn.graphsage`), 66  
GraphUnet (class in `cogdl.models.nn.pyg_graph_unet`), 68  
GraRep (class in `cogdl.models.emb.grarep`), 57  
GTN (class in `cogdl.models.nn.pyg_gtn`), 77  
GTNDataset (class in `cogdl.datasets.gtn_data`), 37

## H

HAN (class in `cogdl.models.nn.han`), 72  
HANDataset (class in `cogdl.datasets.han_data`), 38  
HANLayer (class in `cogdl.layers.han_layer`), 88

- HeterogeneousNodeClassification (class in *cogdl.tasks.heterogeneous\_node\_classification*), 48
- HGPSL (class in *cogdl.models.nn.pyg\_hgpsl*), 66
- Hin2vec (class in *cogdl.models.emb.hin2vec*), 54
- HomoLinkPrediction (class in *cogdl.tasks.link\_prediction*), 49
- HOPE (class in *cogdl.models.emb.hope*), 53
- ## I
- identity\_act() (in module *cogdl.utils.utils*), 92
- IMDB\_GTNDataset (class in *cogdl.datasets.gtn\_data*), 37
- IMDB\_HANDataset (class in *cogdl.datasets.han\_data*), 38
- ImdbBinaryDataset (class in *cogdl.datasets.tu\_data*), 45
- ImdbMultiDataset (class in *cogdl.datasets.tu\_data*), 46
- in\_norm(*cogdl.data.Graph* attribute), 32
- inference() (*cogdl.models.nn.graphsage.Graphsage* method), 67
- inference() (*cogdl.tasks.node\_classification.NodeClassification* method), 48
- InfoGraph (class in *cogdl.models.nn.infograph*), 79
- is\_inductive() (*cogdl.data.Graph* method), 32
- is\_symmetric() (*cogdl.data.Adjacency* method), 33
- is\_symmetric() (*cogdl.data.Graph* method), 32
- ## J
- JKNet (class in *cogdl.models.nn.dgl\_jknet*), 74
- ## K
- KDD\_ICDM\_GCCDataset (class in *cogdl.datasets.gcc\_data*), 36
- keys(*cogdl.data.Adjacency* attribute), 33
- keys(*cogdl.data.Graph* attribute), 32
- KGLinkPrediction (class in *cogdl.tasks.link\_prediction*), 49
- KnowledgeGraphDataset (class in *cogdl.datasets.kg\_data*), 38
- ## L
- len() (*cogdl.data.MultiGraphDataset* method), 35
- LINE (class in *cogdl.models.emb.line*), 61
- LinearLayer (class in *cogdl.layers.pprgo\_layer*), 89
- LinkPredictCompGCN (class in *cogdl.models.nn.compGCN*), 67
- LinkPrediction (class in *cogdl.tasks.link\_prediction*), 50
- LinkPredictRGCN (class in *cogdl.models.nn.rgcN*), 77
- load\_from\_pretrained() (*cogdl.tasks.base\_task.BaseTask* method), 47
- load\_from\_pretrained() (*cogdl.tasks.link\_prediction.LinkPrediction* method), 50
- LoadFrom (class in *cogdl.tasks.base\_task*), 47
- local\_graph() (*cogdl.data.Graph* method), 32
- log\_metrics() (in *cogdl.tasks.link\_prediction* module), 50
- loss() (*cogdl.models.nn.compGCN.LinkPredictCompGCN* method), 67
- loss() (*cogdl.models.nn.dgi.DGIModel* method), 63
- loss() (*cogdl.models.nn.dgl\_jknet.JKNet* method), 74
- loss() (*cogdl.models.nn.han.HAN* method), 73
- loss() (*cogdl.models.nn.mvgrl.MVGRL* method), 64
- loss() (*cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN* method), 68
- loss() (*cogdl.models.nn.pyg\_gtn.GTN* method), 77
- loss() (*cogdl.models.nn.rgcN.LinkPredictRGCN* method), 78
- loss() (*cogdl.models.nn.unsup\_graphsage.SAGE* method), 84
- loss() (*cogdl.models.supervised\_model.SupervisedHeterogeneousNodeClassification* method), 53
- loss() (*cogdl.models.supervised\_model.SupervisedHomogeneousNodeClassification* method), 53
- loss() (*cogdl.models.supervised\_model.SupervisedModel* method), 53
- ## M
- MAGDataset (class in *cogdl.datasets.ogb*), 41
- makedirs() (in module *cogdl.utils.utils*), 92
- mask2nid() (*cogdl.data.Graph* method), 32
- MatlabMatrix (class in *cogdl.datasets.matlab\_matrix*), 40
- MeanAggregator (class in *cogdl.layers.sage\_layer*), 85
- message\_norm() (*cogdl.layers.deeperGCN\_layer.GENConv* method), 87
- Metapath2vec (class in *cogdl.models.emb.metapath2vec*), 59
- mi\_loss() (*cogdl.models.nn.infograph.InfoGraph* static method), 79
- mini\_forward() (*cogdl.models.nn.graphsage.Graphsage* method), 67
- mini\_loss() (*cogdl.models.nn.graphsage.Graphsage* method), 67
- MixHop (class in *cogdl.models.nn.mixhop*), 71
- MixHopLayer (class in *cogdl.layers.mixhop\_layer*), 90
- MLP (class in *cogdl.layers.mlp\_layer*), 88
- MLP (class in *cogdl.models.nn.mlp*), 81
- MLP (class in *cogdl.models.nn.sign*), 71



- `model_name (cogdl.models.emb.complex.ComplEx attribute)`, 60
- `model_name (cogdl.models.emb.deepwalk.DeepWalk attribute)`, 56
- `model_name (cogdl.models.emb.dgk.DeepGraphKernel attribute)`, 57
- `model_name (cogdl.models.emb.distmult.DistMult attribute)`, 55
- `model_name (cogdl.models.emb.dngr.DNGR attribute)`, 58
- `model_name (cogdl.models.emb.gatne.GATNE attribute)`, 57
- `model_name (cogdl.models.emb.graph2vec.Graph2Vec attribute)`, 59
- `model_name (cogdl.models.emb.grarep.GraRep attribute)`, 58
- `model_name (cogdl.models.emb.hin2vec.Hin2vec attribute)`, 54
- `model_name (cogdl.models.emb.hope.HOPE attribute)`, 54
- `model_name (cogdl.models.emb.line.LINE attribute)`, 62
- `model_name (cogdl.models.emb.metapath2vec.Metapath2vec attribute)`, 59
- `model_name (cogdl.models.emb.netmf.NetMF attribute)`, 55
- `model_name (cogdl.models.emb.netsmf.NetSMF attribute)`, 61
- `model_name (cogdl.models.emb.node2vec.Node2vec attribute)`, 60
- `model_name (cogdl.models.emb.prone.ProNE attribute)`, 63
- `model_name (cogdl.models.emb.pronepp.ProNEPP attribute)`, 59
- `model_name (cogdl.models.emb.pte.PTE attribute)`, 61
- `model_name (cogdl.models.emb.rotate.Rotate attribute)`, 56
- `model_name (cogdl.models.emb.sdne.SDNE attribute)`, 62
- `model_name (cogdl.models.emb.spectral.Spectral attribute)`, 54
- `model_name (cogdl.models.emb.transe.TransE attribute)`, 55
- `model_name (cogdl.models.nn.compvcn.LinkPredictCompVCN attribute)`, 67
- `model_name (cogdl.models.nn.deepergcn.DeeperGCN attribute)`, 78
- `model_name (cogdl.models.nn.dgi.DGIModel attribute)`, 63
- `model_name (cogdl.models.nn.dgl_gcc.GCC attribute)`, 83
- `model_name (cogdl.models.nn.dgl_jknet.JKNet attribute)`, 74
- `model_name (cogdl.models.nn.diffpool.DiffPool attribute)`, 70
- `model_name (cogdl.models.nn.disengcn.DisenGCN attribute)`, 81
- `model_name (cogdl.models.nn.drgat.DrGAT attribute)`, 79
- `model_name (cogdl.models.nn.drgcn.DrGCN attribute)`, 68
- `model_name (cogdl.models.nn.dropedge_gcn.DropEdge_GCN attribute)`, 80
- `model_name (cogdl.models.nn.gat.GAT attribute)`, 72
- `model_name (cogdl.models.nn.gcn.TKipfGCN attribute)`, 65
- `model_name (cogdl.models.nn.gcnii.GCNII attribute)`, 71
- `model_name (cogdl.models.nn.gcnmix.GCNMix attribute)`, 69
- `model_name (cogdl.models.nn.gdc_gcn.GDC_GCN attribute)`, 66
- `model_name (cogdl.models.nn.gin.GIN attribute)`, 75
- `model_name (cogdl.models.nn.grace.GRACE attribute)`, 74
- `model_name (cogdl.models.nn.grand.Grand attribute)`, 77
- `model_name (cogdl.models.nn.graphsage.Graphsage attribute)`, 67
- `model_name (cogdl.models.nn.han.HAN attribute)`, 73
- `model_name (cogdl.models.nn.infograph.InfoGraph attribute)`, 79
- `model_name (cogdl.models.nn.mixhop.MixHop attribute)`, 72
- `model_name (cogdl.models.nn.mlp.MLP attribute)`, 81
- `model_name (cogdl.models.nn.mvgrl.MVGRL attribute)`, 64
- `model_name (cogdl.models.nn.patchy_san.PatchySAN attribute)`, 64
- `model_name (cogdl.models.nn.ppnpp.PPNP attribute)`, 73
- `model_name (cogdl.models.nn.pprgo.PPRGo attribute)`, 75
- `model_name (cogdl.models.nn.pyg_cheb.Chebyshev attribute)`, 65
- `model_name (cogdl.models.nn.pyg_dgcnn.DGCNN attribute)`, 76
- `model_name (cogdl.models.nn.pyg_gcn.GCN attribute)`, 71
- `model_name (cogdl.models.nn.pyg_gpt_gnn.GPT_GNN attribute)`, 68
- `model_name (cogdl.models.nn.pyg_graph_unet.GraphUnet attribute)`, 69
- `model_name (cogdl.models.nn.pyg_gtn.GTN attribute)`, 77
- `model_name (cogdl.models.nn.pyg_hgpsl.HGPSL attribute)`, 66
- `model_name (cogdl.models.nn.pyg_sagpool.SAGPoolNetwork attribute)`, 66

- attribute*), 84
- `model_name` (*cogdl.models.nn.pyg\_srgcn.SRGCN attribute*), 83
- `model_name` (*cogdl.models.nn.rgcn.LinkPredictRGCN attribute*), 78
- `model_name` (*cogdl.models.nn.sgc.sgc attribute*), 81
- `model_name` (*cogdl.models.nn.sign.MLP attribute*), 71
- `model_name` (*cogdl.models.nn.sortpool.SortPool attribute*), 82
- `model_name` (*cogdl.models.nn.stpgnn.stpgnn attribute*), 82
- `model_name` (*cogdl.models.nn.unsup\_graphsage.SAGE attribute*), 84
- `MoleculeDataset` (class in *cogdl.datasets.strategies\_data*), 44
- `multiclass_f1()` (in module *cogdl.utils.evaluator*), 92
- `MultiGraphDataset` (class in *cogdl.data*), 35
- `multilabel_f1()` (in module *cogdl.utils.evaluator*), 92
- `MultiplexLinkPrediction` (class in *cogdl.tasks.multiplex\_link\_prediction*), 51
- `MultiplexNodeClassification` (class in *cogdl.tasks.multiplex\_node\_classification*), 49
- `MUTAGDataset` (class in *cogdl.datasets.tu\_data*), 46
- `MVGRL` (class in *cogdl.models.nn.mvgrl*), 63
- ## N
- `NCT109Dataset` (class in *cogdl.datasets.tu\_data*), 46
- `NCT1Dataset` (class in *cogdl.datasets.tu\_data*), 46
- `NegativeEdge` (class in *cogdl.datasets.strategies\_data*), 45
- `NetMF` (class in *cogdl.models.emb.netmf*), 55
- `NetSMF` (class in *cogdl.models.emb.netsmf*), 61
- `NetworkEmbeddingCMTYDataset` (class in *cogdl.datasets.matlab\_matrix*), 40
- `Node2vec` (class in *cogdl.models.emb.node2vec*), 60
- `node_classification_loss()` (*cogdl.models.base\_model.BaseModel method*), 53
- `node_classification_loss()` (*cogdl.models.nn.gcnmix.GCNMix method*), 69
- `node_classification_loss()` (*cogdl.models.nn.gdc\_gcn.GDC\_GCN method*), 66
- `node_classification_loss()` (*cogdl.models.nn.grand.Grand method*), 77
- `node_classification_loss()` (*cogdl.models.nn.graphsage.Graphsage method*), 67
- `node_classification_loss()` (*cogdl.models.nn.pprgo.PPRGo method*), 75
- `node_classification_loss()` (*cogdl.models.nn.unsup\_graphsage.SAGE method*), 84
- `node_degree_as_feature()` (in module *cogdl.tasks.graph\_classification*), 51
- `NodeClassification` (class in *cogdl.tasks.node\_classification*), 48
- `norm()` (*cogdl.models.nn.pyg\_gtn.GTN method*), 77
- `normalization()` (*cogdl.models.nn.pyg\_gtn.GTN method*), 77
- `normalize()` (*cogdl.data.Graph method*), 32
- `normalize_adj()` (*cogdl.data.Adjacency method*), 33
- `normalize_feature()` (in module *cogdl.datasets.tu\_data*), 46
- `normalize_x()` (*cogdl.models.nn.grand.Grand method*), 77
- `num_authors` (*cogdl.datasets.ogb.MAGDataset attribute*), 41
- `num_classes` (*cogdl.data.Dataset attribute*), 34
- `num_classes` (*cogdl.data.Graph attribute*), 32
- `num_classes` (*cogdl.data.MultiGraphDataset attribute*), 35
- `num_classes` (*cogdl.datasets.gcc\_data.Edgelist attribute*), 36
- `num_classes` (*cogdl.datasets.gtn\_data.GTNDataset attribute*), 37
- `num_classes` (*cogdl.datasets.han\_data.HANDataset attribute*), 38
- `num_classes` (*cogdl.datasets.matlab\_matrix.MatlabMatrix attribute*), 40
- `num_classes` (*cogdl.datasets.matlab\_matrix.NetworkEmbeddingCMTY attribute*), 40
- `num_classes` (*cogdl.datasets.ogb.OGBGDataset attribute*), 41
- `num_classes` (*cogdl.datasets.tu\_data.TUDataset attribute*), 46
- `num_edge_attributes()` (in module *cogdl.datasets.tu\_data*), 46
- `num_edge_labels()` (in module *cogdl.datasets.tu\_data*), 46
- `num_edge_types` (*cogdl.datasets.ogb.MAGDataset attribute*), 41
- `num_edges` (*cogdl.data.Adjacency attribute*), 33
- `num_edges` (*cogdl.data.Graph attribute*), 32
- `num_entities` (*cogdl.datasets.kg\_data.KnowledgeGraphDataset attribute*), 39
- `num_features` (*cogdl.data.Dataset attribute*), 34
- `num_features` (*cogdl.data.Graph attribute*), 32
- `num_features` (*cogdl.data.MultiGraphDataset attribute*), 35
- `num_field_of_study` (*cogdl.datasets.ogb.MAGDataset attribute*), 41
- `num_graphs` (*cogdl.data.Batch attribute*), 34

- num\_graphs (*cogdl.datasets.strategies\_data.BatchAE attribute*), 43
- num\_graphs (*cogdl.datasets.strategies\_data.BatchMasking attribute*), 43
- num\_graphs (*cogdl.datasets.strategies\_data.BatchSubstructContext attribute*), 44
- num\_institutions (*cogdl.datasets.ogb.MAGDataset attribute*), 41
- num\_node\_attributes() (in module *cogdl.datasets.tu\_data*), 46
- num\_node\_labels() (in module *cogdl.datasets.tu\_data*), 46
- num\_node\_types (*cogdl.datasets.ogb.MAGDataset attribute*), 41
- num\_nodes (*cogdl.data.Adjacency attribute*), 33
- num\_nodes (*cogdl.data.Graph attribute*), 32
- num\_nodes (*cogdl.datasets.matlab\_matrix.MatlabMatrix attribute*), 40
- num\_nodes (*cogdl.datasets.matlab\_matrix.NetworkEmbeddingCMTYDataset attribute*), 40
- num\_papers (*cogdl.datasets.ogb.MAGDataset attribute*), 41
- num\_relations (*cogdl.datasets.kg\_data.KnowledgeGraphDataset attribute*), 39
- nx\_to\_graph\_data\_obj() (in module *cogdl.datasets.strategies\_data*), 45
- nx\_to\_graph\_data\_obj\_simple() (in module *cogdl.datasets.strategies\_data*), 45
- ## O
- OAGBertInferencePipepline (class in *cogdl.pipelines*), 93
- OGBArxivDataset (class in *cogdl.datasets.ogb*), 41
- OGBCodeDataset (class in *cogdl.datasets.ogb*), 41
- OGBGDataset (class in *cogdl.datasets.ogb*), 41
- OGBMolbaseDataset (class in *cogdl.datasets.ogb*), 41
- OGBMolhivDataset (class in *cogdl.datasets.ogb*), 41
- OGBMolpcbaDataset (class in *cogdl.datasets.ogb*), 41
- OGBNDataset (class in *cogdl.datasets.ogb*), 41
- OGBPapers100MDataset (class in *cogdl.datasets.ogb*), 42
- OGBPPaDataset (class in *cogdl.datasets.ogb*), 42
- OGBProductsDataset (class in *cogdl.datasets.ogb*), 42
- OGBProteinsDataset (class in *cogdl.datasets.ogb*), 42
- one\_shot\_iterator() (*cogdl.datasets.kg\_data.BidirectionalOneShotIterator static method*), 38
- out\_norm (*cogdl.data.Graph attribute*), 32
- output\_results() (in module *cogdl.experiments*), 93
- ## P
- parse\_args\_and\_arch() (in module *cogdl.options*), 91
- parse\_txt\_array() (in module *cogdl.datasets.tu\_data*), 46
- PatchySAN (class in *cogdl.models.nn.patchy\_san*), 64
- Pipeline (class in *cogdl.pipelines*), 93
- pipeline() (in module *cogdl.pipelines*), 94
- PPIDataset (class in *cogdl.datasets.matlab\_matrix*), 40
- PPNP (class in *cogdl.models.nn.pppnp*), 73
- PPRGo (class in *cogdl.models.nn.pprgo*), 74
- PPRGoLayer (class in *cogdl.layers.pprgo\_layer*), 89
- predict() (*cogdl.models.base\_model.BaseModel method*), 53
- predict() (*cogdl.models.nn.compvcn.LinkPredictCompGCN method*), 67
- predict() (*cogdl.models.nn.deepergcn.DeeperGCN method*), 78
- predict() (*cogdl.models.nn.dgl\_jknet.JKNet method*), 74
- predict() (*cogdl.models.nn.disengcn.DisenGCN method*), 81
- predict() (*cogdl.models.nn.drgcn.DrGCN method*), 68
- predict() (*cogdl.models.nn.dropege\_gcnn.DropEdge\_GCNN method*), 80
- predict() (*cogdl.models.nn.gat.GAT method*), 72
- predict() (*cogdl.models.nn.gcn.TKipfGCN method*), 65
- predict() (*cogdl.models.nn.gcnii.GCNII method*), 71
- predict() (*cogdl.models.nn.gcnmix.GCNMix method*), 69
- predict() (*cogdl.models.nn.gdc\_gcnn.GDC\_GCNN method*), 66
- predict() (*cogdl.models.nn.grand.Grand method*), 77
- predict() (*cogdl.models.nn.graphsage.Graphsage method*), 67
- predict() (*cogdl.models.nn.mixhop.MixHop method*), 72
- predict() (*cogdl.models.nn.mlp.MLP method*), 81
- predict() (*cogdl.models.nn.pppnp.PPPNP method*), 73
- predict() (*cogdl.models.nn.pprgo.PPRGo method*), 75
- predict() (*cogdl.models.nn.pyg\_cheb.Chebyshev method*), 65
- predict() (*cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN method*), 68
- predict() (*cogdl.models.nn.pyg\_srgcn.SRGCN method*), 83
- predict() (*cogdl.models.nn.rgcn.LinkPredictRGCN method*), 78
- predict() (*cogdl.models.nn.sgc.sgc method*), 81
- predict() (*cogdl.models.nn.sign.MLP method*), 71

predict () (*cogdl.models.supervised\_model.SupervisedHomogeneousNodeClassificationModel*  
     *method*), 53  
 predict () (*cogdl.tasks.unsupervised\_node\_classification.TopKRanking* (*cogdl.datasets.gtn\_data.GTNDataset*  
     *method*), 48  
 preprocess () (*cogdl.datasets.gcc\_data.GCCDataset* *processed\_file\_names*  
     *method*), 36  
 preprocess () (*cogdl.models.nn.mvgrl.MVGRL* *processed\_file\_names*  
     *method*), 64  
 preprocess () (*cogdl.tasks.node\_classification.NodeClassification* (*cogdl.datasets.kg\_data.KnowledgeGraphDataset*  
     *method*), 48  
 preprocessing () (*cogdl.models.nn.gdc\_gcn.GDC\_GCN* *processed\_file\_names*  
     *method*), 66  
 PretrainTask (*class in cogdl.tasks.pretrain*), 52  
 print\_result () (*in module cogdl.utils.utils*), 92  
 process () (*cogdl.data.Dataset method*), 34  
 process () (*cogdl.datasets.gatne.GatneDataset*  
     *method*), 35  
 process () (*cogdl.datasets.gcc\_data.Edgelist method*),  
     36  
 process () (*cogdl.datasets.gtn\_data.GTNDataset*  
     *method*), 37  
 process () (*cogdl.datasets.han\_data.HANDataset* *processed\_file\_names*  
     *method*), 38  
 process () (*cogdl.datasets.kg\_data.KnowledgeGraphDataset*  
     *method*), 39  
 process () (*cogdl.datasets.matlab\_matrix.MatlabMatrix*  
     *method*), 40  
 process () (*cogdl.datasets.matlab\_matrix.NetworkEmbeddingCMTYDataset*  
     *method*), 40  
 process () (*cogdl.datasets.ogb.MAGDataset method*),  
     41  
 process () (*cogdl.datasets.ogb.OGBNDataset*  
     *method*), 42  
 process () (*cogdl.datasets.ogb.OGBProteinsDataset* *processed\_file\_names*  
     *method*), 42  
 process () (*cogdl.datasets.strategies\_data.BACEDataset*  
     *method*), 42  
 process () (*cogdl.datasets.strategies\_data.BBBPDataset* *processed\_file\_names*  
     *method*), 42  
 process () (*cogdl.datasets.strategies\_data.BioDataset*  
     *method*), 44  
 process () (*cogdl.datasets.strategies\_data.MoleculeDataset*  
     *method*), 44  
 process () (*cogdl.datasets.tu\_data.TUDataset*  
     *method*), 46  
 processed\_file\_names (*cogdl.data.Dataset*  
     *attribute*), 34  
 processed\_file\_names  
     (*cogdl.datasets.gatne.GatneDataset attribute*),  
     36  
 processed\_file\_names  
     (*cogdl.datasets.gcc\_data.Edgelist attribute*), 36  
 processed\_file\_names  
     (*cogdl.datasets.gcc\_data.GCCDataset at-*  
     *tribute*), 37  
 processed\_file\_names  
     (*cogdl.datasets.han\_data.HANDataset at-*  
     *tribute*), 38  
 processed\_file\_names  
     (*cogdl.datasets.kg\_data.KnowledgeGraphDataset*  
     *attribute*), 39  
 processed\_file\_names  
     (*cogdl.datasets.matlab\_matrix.MatlabMatrix*  
     *attribute*), 40  
 processed\_file\_names  
     (*cogdl.datasets.matlab\_matrix.NetworkEmbeddingCMTYDataset*  
     *attribute*), 40  
 processed\_file\_names  
     (*cogdl.datasets.ogb.MAGDataset attribute*), 41  
 processed\_file\_names  
     (*cogdl.datasets.ogb.OGBNDataset attribute*),  
     42  
 processed\_file\_names  
     (*cogdl.datasets.strategies\_data.BACEDataset*  
     *attribute*), 42  
 processed\_file\_names  
     (*cogdl.datasets.strategies\_data.BBBPDataset*  
     *attribute*), 42  
 processed\_file\_names  
     (*cogdl.datasets.strategies\_data.BioDataset*  
     *attribute*), 44  
 processed\_file\_names  
     (*cogdl.datasets.strategies\_data.MoleculeDataset*  
     *attribute*), 44  
 processed\_file\_names  
     (*cogdl.datasets.tu\_data.TUDataset attribute*),  
     46  
 processed\_paths (*cogdl.data.Dataset attribute*), 34  
 ProNE (*class in cogdl.models.emb.prone*), 62  
 ProNEPP (*class in cogdl.models.emb.pronepp*), 58  
 prop () (*cogdl.models.nn.grace.GRACE method*), 74  
 ProtainsDataset (*class in cogdl.datasets.tu\_data*),  
     46  
 PTCMRDataset (*class in cogdl.datasets.tu\_data*), 46  
 PTE (*class in cogdl.models.emb.pte*), 60

## R

rand\_prop () (*cogdl.models.nn.grand.Grand method*),  
     77  
 random\_surfing () (*cogdl.models.emb.dngr.DNGR*  
     *method*), 58  
 random\_walk (*in module cogdl.utils.sampling*), 93  
 random\_walk () (*cogdl.data.Adjacency method*), 33  
 random\_walk\_with\_restart ()  
     (*cogdl.data.Adjacency method*), 33

*randomly\_choose\_false\_edges()* (in module *cogdl.tasks.link\_prediction*), 50  
*RandomWalker* (class in *cogdl.utils.sampling*), 93  
*raw\_edge\_weight* (*cogdl.data.Graph* attribute), 32  
*raw\_experiment()* (in module *cogdl.experiments*), 93  
*raw\_file\_names* (*cogdl.data.Dataset* attribute), 34  
*raw\_file\_names* (*cogdl.datasets.gatne.GatneDataset* attribute), 36  
*raw\_file\_names* (*cogdl.datasets.gcc\_data.Edgelist* attribute), 36  
*raw\_file\_names* (*cogdl.datasets.gcc\_data.GCCDataset* attribute), 36  
*raw\_file\_names* (*cogdl.datasets.gtn\_data.GTNDataset* attribute), 37  
*raw\_file\_names* (*cogdl.datasets.han\_data.HANDataset* attribute), 38  
*raw\_file\_names* (*cogdl.datasets.kg\_data.KnowledgeGraphDataset* attribute), 39  
*raw\_file\_names* (*cogdl.datasets.matlab\_matrix.MatlabMatrix* attribute), 40  
*raw\_file\_names* (*cogdl.datasets.matlab\_matrix.NetworkEmbeddingGMLMDataMatrix* attribute), 40  
*raw\_file\_names* (*cogdl.datasets.strategies\_data.BACEDataset* attribute), 42  
*raw\_file\_names* (*cogdl.datasets.strategies\_data.BBBPDataset* attribute), 43  
*raw\_file\_names* (*cogdl.datasets.strategies\_data.BioDataset* attribute), 44  
*raw\_file\_names* (*cogdl.datasets.strategies\_data.MoleculeDataset* attribute), 45  
*raw\_file\_names* (*cogdl.datasets.tu\_data.TUDataset* attribute), 46  
*raw\_paths* (*cogdl.data.Dataset* attribute), 35  
*read\_file()* (in module *cogdl.datasets.tu\_data*), 46  
*read\_gatne\_data()* (in module *cogdl.datasets.gatne*), 36  
*read\_gtn\_data()* (*cogdl.datasets.gtn\_data.GTNDataset* method), 37  
*read\_gtn\_data()* (*cogdl.datasets.han\_data.HANDataset* method), 38  
*read\_triplet\_data()* (in module *cogdl.datasets.kg\_data*), 39  
*read\_tu\_data()* (in module *cogdl.datasets.tu\_data*), 47  
*read\_txt\_array()* (in module *cogdl.datasets.tu\_data*), 47  
*RecommendationPipepline* (class in *cogdl.pipelines*), 94  
*RedditBinary* (class in *cogdl.datasets.tu\_data*), 46  
*RedditMulti12K* (class in *cogdl.datasets.tu\_data*), 46  
*RedditMulti5K* (class in *cogdl.datasets.tu\_data*), 46  
*register\_dataset()* (in module *cogdl.datasets*), 47  
*register\_model()* (in module *cogdl.models*), 84  
*register\_task()* (in module *cogdl.tasks*), 52  
*remove\_self\_loops()* (*cogdl.data.Adjacency* method), 33  
*remove\_self\_loops()* (*cogdl.data.Graph* method), 32  
*reset\_data()* (*cogdl.models.nn.gdc\_gcn.GDC\_GCN* method), 66  
*reset\_idxes()* (in module *cogdl.datasets.strategies\_data*), 45  
*reset\_parameters()* (*cogdl.layers.disengcn\_layer.DisenGCNLayer* method), 88  
*reset\_parameters()* (*cogdl.layers.gat\_layer.GATLayer* method), 85  
*reset\_parameters()* (*cogdl.layers.gcn\_layer.GCNLayer* method), 85  
*reset\_parameters()* (*cogdl.layers.gcnii\_layer.GCNIILayer* method), 86  
*reset\_parameters()* (*cogdl.layers.gmlm\_data\_matrixhop\_layer.MixHopLayer* method), 91  
*reset\_parameters()* (*cogdl.layers.mlp\_layer.MLP* method), 89  
*reset\_parameters()* (*cogdl.layers.pprgo\_layer.LinearLayer* method), 89  
*reset\_parameters()* (*cogdl.layers.rgc\_n\_layer.RGCNLayer* method), 90  
*reset\_parameters()* (*cogdl.models.nn.diffpool.DiffPool* method), 70  
*reset\_parameters()* (*cogdl.models.nn.disengcn.DisenGCN* method), 81  
*reset\_parameters()* (*cogdl.models.nn.dropedge\_gcn.DropEdge\_GCN* method), 80  
*reset\_parameters()* (*cogdl.models.nn.infograph.InfoGraph* method), 79  
*ResGNNLayer* (class in *cogdl.layers.deepergcn\_layer*), 87  
*RGCNLayer* (class in *cogdl.layers.rgc\_n\_layer*), 89  
*RotatE* (class in *cogdl.models.emb.rotate*), 56  
*row\_indptr* (*cogdl.data.Adjacency* attribute), 33  
*row\_indptr* (*cogdl.data.Graph* attribute), 32  
*row\_norm()* (*cogdl.data.Adjacency* method), 33  
*row\_norm()* (*cogdl.data.Graph* method), 32  
*run()* (*cogdl.experiments.AutoML* method), 93

## S

*SAGE* (class in *cogdl.models.nn.unsup\_graphsage*), 83

- SAGELayer (class in cogdl.layers.sage\_layer), 85
- SAGPoolNetwork (class in cogdl.models.nn.pyg\_sagpool), 84
- SAINTLayer (class in cogdl.layers.saint\_layer), 90
- sample\_adj() (cogdl.data.Graph method), 32
- sample\_mask() (in module cogdl.datasets.han\_data), 38
- sampling() (cogdl.models.nn.graphsage.Graphsage method), 67
- sampling() (cogdl.models.nn.unsup\_graphsage.SAGE method), 84
- save\_checkpoint() (cogdl.tasks.base\_task.BaseTask method), 47
- save\_checkpoint() (cogdl.tasks.link\_prediction.LinkPrediction method), 50
- save\_emb() (cogdl.tasks.unsupervised\_graph\_classification.UnsupervisedGraphClassification method), 51
- save\_emb() (cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification method), 48
- save\_embedding() (cogdl.models.emb.dgk.DeepGraphKernel class in cogdl.models.nn.sgc), 81
- save\_embedding() (cogdl.models.emb.graph2vec.Graph2Vec method), 59
- save\_model() (in module cogdl.tasks.link\_prediction), 50
- scale\_matrix() (cogdl.models.emb.dngr.DNGR method), 58
- score() (cogdl.models.emb.complex.Complex method), 60
- score() (cogdl.models.emb.distmult.DistMult method), 55
- score() (cogdl.models.emb.rotate.RotatE method), 56
- score() (cogdl.models.emb.transe.TransE method), 55
- SDNE (class in cogdl.models.emb.sdne), 62
- segment() (in module cogdl.datasets.tu\_data), 47
- SELayer (class in cogdl.layers.se\_layer), 91
- select\_task() (in module cogdl.tasks.link\_prediction), 50
- self\_supervised\_loss() (cogdl.models.nn.dgi.DGIModel method), 63
- self\_supervised\_loss() (cogdl.models.nn.grace.GRACE method), 74
- self\_supervised\_loss() (cogdl.models.nn.mvgrl.MVGRL method), 64
- self\_supervised\_loss() (cogdl.models.nn.unsup\_graphsage.SAGE method), 84
- set\_asymmetric() (cogdl.data.Graph method), 32
- set\_best\_config() (in module cogdl.experiments), 93
- set\_data\_device() (cogdl.models.nn.graphsage.Graphsage method), 67
- set\_device() (cogdl.models.base\_model.BaseModel method), 53
- set\_evaluator() (cogdl.tasks.base\_task.BaseTask method), 47
- set\_graph() (cogdl.models.nn.dgl\_jknet.JKNet method), 74
- set\_logger() (in module cogdl.tasks.link\_prediction), 50
- set\_loss\_fn() (cogdl.models.base\_model.BaseModel method), 53
- set\_loss\_fn() (cogdl.tasks.base\_task.BaseTask method), 47
- set\_random\_seed() (in module cogdl.utils.utils), 92
- set\_weight() (cogdl.data.Adjacency method), 33
- SGCLayer (class in cogdl.layers.sgc\_layer), 90
- SGCCDataset (class in cogdl.datasets.gcc\_data), 37
- SIGMOD\_ICDE\_GCCDataset (class in cogdl.datasets.gcc\_data), 37
- SimilaritySearch (class in cogdl.tasks.similarity\_search), 52
- SortPool (class in cogdl.models.nn.sortpool), 82
- Spectral (class in cogdl.models.emb.spectral), 54
- split() (in module cogdl.datasets.tu\_data), 47
- split\_dataset() (cogdl.models.nn.diffpool.DiffPool class method), 70
- split\_dataset() (cogdl.models.nn.gin.GIN class method), 75
- split\_dataset() (cogdl.models.nn.infograph.InfoGraph class method), 79
- split\_dataset() (cogdl.models.nn.patchy\_san.PatchySAN class method), 64
- split\_dataset() (cogdl.models.nn.pyg\_dgcnn.DGCNN class method), 76
- split\_dataset() (cogdl.models.nn.pyg\_hgpsl.HGPSL class method), 66
- split\_dataset() (cogdl.models.nn.pyg\_sagpool.SAGPoolNetwork class method), 84
- split\_dataset() (cogdl.models.nn.sortpool.SortPool class method), 82
- split\_dataset\_general() (in module cogdl.utils.utils), 92
- SRGCN (class in cogdl.models.nn.pyg\_srgcn), 82
- stpgnn (class in cogdl.models.nn.stpgnn), 81
- subgraph() (cogdl.data.Graph method), 32
- SumAggregator (class in cogdl.layers.sage\_layer), 86

`sup_forward()` (*cogdl.models.nn.infograph.InfoGraph method*), 79  
`sup_loss()` (*cogdl.models.nn.infograph.InfoGraph method*), 79  
`SupervisedHeterogeneousNodeClassificationModel` (class in *cogdl.models.supervised\_model*), 53  
`SupervisedHomogeneousNodeClassificationModel` (class in *cogdl.models.supervised\_model*), 53  
`SupervisedModel` (class in *cogdl.models.supervised\_model*), 53  
`sym_norm()` (*cogdl.data.Adjacency method*), 33  
`sym_norm()` (*cogdl.data.Graph method*), 32

## T

`tabulate_results()` (in module *cogdl.utils.utils*), 92  
`test_nid` (*cogdl.data.Graph attribute*), 32  
`test_start_idx` (*cogdl.datasets.kg\_data.KnowledgeGraphDataset attribute*), 39  
`test_step()` (*cogdl.tasks.link\_prediction.TripleLinkPrediction method*), 50  
`TestBioDataset` (class in *cogdl.datasets.strategies\_data*), 45  
`TestChemDataset` (class in *cogdl.datasets.strategies\_data*), 45  
`TestDataset` (class in *cogdl.datasets.kg\_data*), 39  
`TKipfGCN` (class in *cogdl.models.nn.gcn*), 65  
`to_scipy_csr()` (*cogdl.data.Adjacency method*), 33  
`to_scipy_csr()` (*cogdl.data.Graph method*), 32  
`TopKRanker` (class in *cogdl.tasks.unsupervised\_node\_classification*), 48  
`train()` (*cogdl.data.Graph method*), 32  
`train()` (*cogdl.models.emb.deepwalk.DeepWalk method*), 56  
`train()` (*cogdl.models.emb.dngr.DNGR method*), 58  
`train()` (*cogdl.models.emb.gatne.GATNE method*), 57  
`train()` (*cogdl.models.emb.grarep.GraRep method*), 58  
`train()` (*cogdl.models.emb.hin2vec.Hin2vec method*), 54  
`train()` (*cogdl.models.emb.hope.HOPE method*), 54  
`train()` (*cogdl.models.emb.line.LINE method*), 62  
`train()` (*cogdl.models.emb.metapath2vec.Metapath2vec method*), 59  
`train()` (*cogdl.models.emb.netmf.NetMF method*), 55  
`train()` (*cogdl.models.emb.netsmf.NetSMF method*), 61  
`train()` (*cogdl.models.emb.node2vec.Node2vec method*), 60  
`train()` (*cogdl.models.emb.prono.ProNE method*), 63  
`train()` (*cogdl.models.emb.pte.PTE method*), 61  
`train()` (*cogdl.models.emb.sdne.SDNE method*), 62  
`train()` (*cogdl.models.emb.spectral.Spectral method*), 54  
`train()` (*cogdl.models.nn.dgl\_gcc.GCC method*), 83  
`train()` (*cogdl.tasks.attributed\_graph\_clustering.AttributedGraphClustering method*), 52  
`train()` (*cogdl.tasks.base\_task.BaseTask method*), 47  
`train()` (*cogdl.tasks.graph\_classification.GraphClassification method*), 51  
`train()` (*cogdl.tasks.heterogeneous\_node\_classification.HeterogeneousNodeClassification method*), 48  
`train()` (*cogdl.tasks.link\_prediction.GNNHomoLinkPrediction method*), 49  
`train()` (*cogdl.tasks.link\_prediction.HomoLinkPrediction method*), 49  
`train()` (*cogdl.tasks.link\_prediction.KGLinkPrediction method*), 49  
`train()` (*cogdl.tasks.link\_prediction.LinkPrediction method*), 50  
`train()` (*cogdl.tasks.link\_prediction.TripleLinkPrediction method*), 50  
`train()` (*cogdl.tasks.multiplex\_link\_prediction.MultiplexLinkPrediction method*), 51  
`train()` (*cogdl.tasks.multiplex\_node\_classification.MultiplexNodeClassification method*), 49  
`train()` (*cogdl.tasks.node\_classification.NodeClassification method*), 48  
`train()` (*cogdl.tasks.pretrain.PretrainTask method*), 52  
`train()` (*cogdl.tasks.similarity\_search.SimilaritySearch method*), 52  
`train()` (*cogdl.tasks.unsupervised\_graph\_classification.UnsupervisedGraphClassification method*), 51  
`train()` (*cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification method*), 48  
`train()` (in module *cogdl.experiments*), 93  
`train_nid` (*cogdl.data.Graph attribute*), 32  
`train_start_idx` (*cogdl.datasets.kg\_data.KnowledgeGraphDataset attribute*), 39  
`train_step()` (*cogdl.tasks.link\_prediction.TripleLinkPrediction method*), 50  
`train_test_edge_split()` (*cogdl.tasks.link\_prediction.GNNHomoLinkPrediction static method*), 49  
`TrainDataset` (class in *cogdl.datasets.kg\_data*), 39  
`TransE` (class in *cogdl.models.emb.transe*), 55  
`TripleLinkPrediction` (class in *cogdl.tasks.link\_prediction*), 50  
`try_import_dataset()` (in module *cogdl.datasets*), 47  
`try_import_model()` (in module *cogdl.models*), 85  
`try_import_task()` (in module *cogdl.tasks*), 52  
`TUDataset` (class in *cogdl.datasets.tu\_data*), 46  
`TwitterDataset` (class in *cogdl.datasets.gatne*), 36

## U

`uniform_node_feature()` (in module `cogdl.tasks.graph_classification`), 51

`unsup_forward()` (`cogdl.models.nn.infograph.InfoGraph` method), 79

`unsup_loss()` (`cogdl.models.nn.infograph.InfoGraph` method), 79

`unsup_sup_loss()` (`cogdl.models.nn.infograph.InfoGraph` method), 79

`UnsupervisedGraphClassification` (class in `cogdl.tasks.unsupervised_graph_classification`), 51

`UnsupervisedNodeClassification` (class in `cogdl.tasks.unsupervised_node_classification`), 48

`untar()` (in module `cogdl.utils.utils`), 92

`update_args_from_dict()` (in module `cogdl.utils.utils`), 92

`url` (`cogdl.datasets.gatne.GatneDataset` attribute), 36

`url` (`cogdl.datasets.gcc_data.Edgelist` attribute), 36

`url` (`cogdl.datasets.gcc_data.GCCDataset` attribute), 36

`url` (`cogdl.datasets.kg_data.KnowledgeGraphDataset` attribute), 39

`url` (`cogdl.datasets.tu_data.TUDataset` attribute), 46

`USAAirportDataset` (class in `cogdl.datasets.gcc_data`), 37

## V

`val_nid` (`cogdl.data.Graph` attribute), 32

`valid_start_idx` (`cogdl.datasets.kg_data.KnowledgeGraphDataset` attribute), 39

`variant_args_generator()` (in module `cogdl.experiments`), 93

## W

`walk()` (`cogdl.utils.sampling.RandomWalker` method), 93

`WikipediaDataset` (class in `cogdl.datasets.matlab_matrix`), 40

`wl_iterations()` (`cogdl.models.emb.dgk.DeepGraphKernel` static method), 57

`wl_iterations()` (`cogdl.models.emb.graph2vec.Graph2Vec` static method), 59

`WN18Dataset` (class in `cogdl.datasets.kg_data`), 39

`WN18RRDataset` (class in `cogdl.datasets.kg_data`), 39

## Y

`YouTubeDataset` (class in `cogdl.datasets.gatne`), 36

`YoutubeNEDataset` (class in `cogdl.datasets.matlab_matrix`), 40