GRB

Release 0.0.2

Qinkai Zheng

Oct 01, 2022

GET STARTED

CHAPTER

ONE

INSTALL AND SETUP

1.1 System requirements

GRB works with Ubuntu 16.04 or higher and requires Python version 3.6 or higher.

1.2 Install via Pip

We recommend installing GRB using `pip`.

```
pip install grb
```

1.3 Install from source

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

CHAPTER

TWO

GRB

2.1 grb.attack

2.1.1 grb.attack.base

class grb.attack.base.**Attack**

Bases: object

Abstract class for graph adversarial attack.

abstract attack(*model*, *adj*, *features*, ***kwargs*)

Parameters

- **model** (*torch.nn.module*) – Model implemented based on torch.nn.module.
- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of N * N sparse matrix.
- **features** (*torch.FloatTensor*) – Features in form of N * D torch float tensor.
- **kwargs** – Keyword-only arguments.

class grb.attack.base.**EarlyStop**(*patience=1000*, *epsilon=0.0001*)

Bases: object

Strategy to early stop attack process.

class grb.attack.base.**InjectionAttack**

Bases: *Attack*

Abstract class for graph injection attack.

abstract attack(***kwargs*)

Parameters
kwargs – Keyword-only arguments.

abstract injection(***kwargs*)

Parameters
kwargs – Keyword-only arguments.

abstract update_features(***kwargs*)

Parameters
kwargs – Keyword-only arguments.

class grb.attack.base.**ModificationAttack**

Bases: *Attack*

Abstract class for graph modification attack.

abstract attack(***kwargs*)

Parameters
kwargs – Keyword-only arguments.

abstract modification(***kwargs*)

Parameters
kwargs – Keyword-only arguments.

2.1.2 grb.attack.fgsm

2.1.3 grb.attack.pgd

2.1.4 grb.attack.rnd

2.1.5 grb.attack.speit

2.1.6 grb.attack.tdgia

2.2 grb.dataset

2.2.1 grb.dataset.dataset

class grb.dataset.dataset.**CogDLDataset**(*name*, *data_dir=None*, *mode='origin'*, *verbose=True*)

Bases: object

property COGDL_GRAPH_CLASSIFICATION_DATASETS

static build_adj(*attr*, *edge_index*, *adj_type='csr'*)

static graph_splitting(*num_graphs*, *train_ratio=0.8*, *val_ratio=0.1*)

class grb.dataset.dataset.**CustomDataset**(*adj*, *features*, *labels*, *train_mask=None*, *val_mask=None*, *test_mask=None*, *name=None*, *data_dir=None*, *mode='full'*, *feat_norm=None*, *save=False*, *verbose=True*, *seed=42*)

Bases: object

Class that helps to build customized dataset for GRB evaluation.

Parameters

- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.
- **features** (*torch.FloatTensor*) – Features in form of `N * D` torch float tensor.
- **labels** (*torch.LongTensor*) – Labels in form of `N * L`. L=1 for multi-class classification, otherwise for multi-label classification.
- **train_mask** (*torch.Tensor, optional*) – Mask of train nodes in form of `N * 1` torch bool tensor. Default: `None`. If is `None`, generated by default splitting scheme.

- **val_mask** (*torch.Tensor, optional*) – Mask of validation nodes in form of `N * 1` torch bool tensor. Default: `None`. If is `None`, generated by default splitting scheme.
- **test_mask** (*torch.Tensor, optional*) – Mask of test nodes in form of `N * 1` torch bool tensor. Default: `None`. If is `None`, generated by default splitting scheme.
- **name** (*str*) – Name of dataset.
- **data_dir** (*str, optional*) – Directory of dataset.
- **mode** (*str, optional*) – Mode of dataset. One of `["easy", "medium", "hard", "full"]`. Default: `full`.
- **feat_norm** (*str, optional*) – Feature normalization that transform all features to range [-1, 1]. Choose from `["arctan", "sigmoid", "tanh"]`. Default: `None`.
- **save** (*bool, optional*) – Whether to save data as files.
- **verbose** (*bool, optional*) – Whether to display logs. Default: `True`.
- **name** – Name of dataset, supported datasets: `["grb-cora", "grb-citeseer", "grb-aminer", "grb-reddit", "grb-flickr"]`.
- **data_dir** – Directory for dataset. If not provided, default is `"./data/"`.
- **mode** – Difficulty determined according to the average degree of test nodes. Choose from `["easy", "medium", "hard", "full"]`. Default: `"full"` is to use the entire test set.
- **feat_norm** – Feature normalization that transform all features to range [-1, 1]. Choose from `["arctan", "sigmoid", "tanh"]`. Default: `None`.
- **verbose** – Whether to display logs. Default: `True`.

adj

Adjacency matrix in form of `N * N` sparse matrix.

Type
scipy.sparse.csr.csr_matrix

features

Features in form of `N * D` torch float tensor.

Type
torch.FloatTensor

labels

Labels in form of `N * L`. L=1 for multi-class classification, otherwise for multi-label classification.

Type
torch.LongTensor

num_nodes

Number of nodes `N`.

Type
int

num_edges

Number of edges.

Type
int

num_features

Dimension of features D.

Type
int

num_classes

Number of classes L.

Type
int

num_train

Number of train nodes.

Type
int

num_val

Number of validation nodes.

Type
int

num_test

Number of test nodes.

Type
int

mode

Mode of dataset. One of `["easy", "medium", "hard", "full"]`.

Type
str

index_train

Index of train nodes.

Type
np.ndarray

index_val

Index of validation nodes.

Type
np.ndarray

index_test

Index of test nodes.

Type
np.ndarray

train_mask

Mask of train nodes in form of `N * 1` torch bool tensor.

Type
torch.Tensor

val_mask

Mask of validation nodes in form of `N * 1` torch bool tensor.

Type
torch.Tensor

test_mask

Mask of test nodes in form of `N * 1` torch bool tensor.

Type
torch.Tensor

class grb.dataset.dataset.**Dataset**(*name*, *data_dir=None*, *mode='easy'*, *feat_norm='arctan'*, *verbose=True*, *custom=False*)

Bases: `object`

Class that loads GRB datasets for evaluating adversarial robustness.

Parameters

- **name** (*str*) – Name of dataset, supported datasets: `["grb-cora", "grb-citeseer", "grb-aminer", "grb-reddit", "grb-flickr"]`.
- **data_dir** (*str, optional*) – Directory for dataset. If not provided, default is `"./data/"`.
- **mode** (*str, optional*) – Difficulty determined according to the average degree of test nodes. Choose from `["easy", "medium", "hard", "full"]`. Default: `"full"` is to use the entire test set.
- **feat_norm** (*str, optional*) – Feature normalization that transform all features to range [-1, 1]. Choose from `["arctan", "sigmoid", "tanh"]`. Default: `None`.
- **verbose** (*bool, optional*) – Whether to display logs. Default: `True`.

adj

Adjacency matrix in form of `N * N` sparse matrix.

Type
scipy.sparse.csr.csr_matrix

features

Features in form of `N * D` torch float tensor.

Type
torch.FloatTensor

labels

Labels in form of `N * L`. L=1 for multi-class classification, otherwise for multi-label classification.

Type
torch.LongTensor

num_nodes

Number of nodes `N`.

Type
int

num_edges

Number of edges.

Type
int

num_features

Dimension of features D.

Type
int

num_classes

Number of classes L.

Type
int

num_train

Number of train nodes.

Type
int

num_val

Number of validation nodes.

Type
int

num_test

Number of test nodes.

Type
int

mode

Mode of dataset. One of `["easy", "medium", "hard", "full"]`.

Type
str

index_train

Index of train nodes.

Type
np.ndarray

index_val

Index of validation nodes.

Type
np.ndarray

index_test

Index of test nodes.

Type
np.ndarray

train_mask

Mask of train nodes in form of `N * 1` torch bool tensor.

Type
torch.Tensor

val_mask

Mask of validation nodes in form of `N * 1` torch bool tensor.

Type
torch.Tensor

test_mask

Mask of test nodes in form of `N * 1` torch bool tensor.

Type
torch.Tensor

Example

```
>>> import grb
>>> from grb.dataset import Dataset
>>> dataset = Dataset(name='grb-cora', mode='easy', feat_norm="arctan")
```

class grb.dataset.dataset.**OGBDataset**(*name*, *data_dir=None*, *verbose=True*)

Bases: `object`

property OGB_GRAPH_CLASSIFICATION_DATASETS

property OGB_NODE_CLASSIFICATION_DATASETS

grb.dataset.dataset.**feat_normalize**(*features*, *norm=None*, *lim_min=-1.0*, *lim_max=1.0*)

Feature normalization function.

Parameters

- **features** (*torch.FloatTensor*) – Features in form of `N * D` torch float tensor.
- **norm** (*str, optional*) – Type of normalization. Choose from `["linearize", "arctan", "tanh", "standarize"]`. Default: `None`.
- **lim_min** (*float*) – Minimum limit of feature value. Default: `-1.0`.
- **lim_max** (*float*) – Maximum limit of feature value. Default: `1.0`.

Returns
features – Normalized features in form of `N * D` torch float tensor.

Return type
torch.FloatTensor

grb.dataset.dataset.**splitting**(*adj*, *range_min=(0.0, 0.05)*, *range_max=(0.95, 1.0)*, *range_easy=(0.05, 0.35)*, *range_medium=(0.35, 0.65)*, *range_hard=(0.65, 0.95)*, *ratio_train=0.6*, *ratio_val=0.1*, *ratio_test=0.1*, *seed=42*)

GRB splitting scheme designed for adversarial robustness evaluation.

Parameters

- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.
- **range_min** (*tuple of float, optional*) – Range of nodes with minimum degrees to be ignored. Value in percentage. Default: `(0.0, 0.05)`.
- **range_max** (*tuple of float, optional*) – Range of nodes with maximum degrees to be ignored. Value in percentage. Default: `(0.95, 1.0)`.

- **range_easy** (*tuple of float, optional*) – Range of nodes for `easy` difficulty. Value in percentage. Default: `(0.05, 0.35)`.
- **range_medium** (*tuple of float, optional*) – Range of nodes for `medium` difficulty. Value in percentage. Default: `(0.35, 0.65)`.
- **range_hard** (*tuple of float, optional*) – Range of nodes for `hard` difficulty. Value in percentage. Default: `(0.65, 0.95)`.
- **ratio_train** (*float, optional*) – Ratio of train nodes. Default: `0.6`.
- **ratio_val** (*float, optional*) – Ratio of validation nodes. Default: `0.1`.
- **ratio_test** (*float, optional*) – Ratio of test nodes. Default: `0.1`.
- **seed** (*int, optional*) – Random seed. Default: `42`.

Returns
index – Dictionary containing `{"index_train", "index_val", "index_test", "index_test_easy", "index_test_medium", "index_test_hard"}`.

Return type
dict

2.3 grb.defense

2.3.1 grb.defense.adv_trainer

class grb.defense.adv_trainer.**AdvTrainer**(*dataset*, *optimizer*, *loss*, *feat_norm=None*, *attack=None*, *attack_mode='injection'*, *lr_scheduler=None*, *lr_patience=100*, *lr_factor=0.75*, *lr_min=1e-05*, *early_stop=None*, *early_stop_patience=100*, *early_stop_epsilon=1e-05*, *eval_metric=<function eval_acc>*, *device='cpu'*)

Bases: `object`

evaluate(*model*, *mask=None*)

Evaluation of a GNN model.

Parameters

- **model** (*torch.nn.module*) – Model implemented based on `torch.nn.module`.
- **mask** (*torch.tensor, optional*) – Mask of target nodes. Default: `None`.

Returns
score – Score on masked nodes.

Return type
float

inference(*model*)

Inference of a GNN model.

Parameters
model (*torch.nn.module*) – Model implemented based on `torch.nn.module`.

Returns
logits – Output logits of model.

Return type
torch.Tensor

train(*model*, *n_epoch*, *save_dir=None*, *save_name=None*, *eval_every=10*, *save_after=0*, *train_mode='trasductive'*, *verbose=True*)

class grb.defense.adv_trainer.**EarlyStop**(*patience=1000*, *epsilon=1e-05*)

Bases: object

2.3.2 grb.defense.base

class grb.defense.base.**Defense**

Bases: object

Abstract class for defense.

abstract defense(*model*, *adj*, *features*, ***kwargs*)

Parameters

- **model** (*torch.nn.module*) – Model implemented based on torch.nn.module.
- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of N * N sparse matrix.
- **features** (*torch.FloatTensor*) – Features in form of N * D torch float tensor.
- **kwargs** – Keyword-only arguments.

2.3.3 grb.defense.gcnsvd

class grb.defense.gcnsvd.**GCNSVD**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *activation=<function relu>*, *layer_norm=False*, *feat_norm=None*, *adj_norm_func=None*, *residual=False*, *dropout=0.0*, *k=50*)

Bases: Module

forward(*x*, *adj*)

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.

---

property model_type

training: bool

truncatedSVD(*adj*, *k=50*)

2.3.4 grb.defense.gnnguard

class grb.defense.gnnguard.**GATGuard**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *n_heads*, *activation=<function leaky_relu>*, *layer_norm=False*, *feat_norm=None*, *adj_norm_func=<function GCNAdjNorm>*, *drop=False*, *attention=True*, *dropout=0.0*)

Bases: Module

att_coef(*features*, *adj*)

forward(*x*, *adj*)

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.

---

property model_type

training: bool

class grb.defense.gnnguard.**GCNGuard**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *activation=<function relu>*, *layer_norm=False*, *dropout=True*, *feat_norm=None*, *adj_norm_func=<function GCNAdjNorm>*, *drop=0.0*, *attention=True*)

Bases: Module

att_coef(*features*, *adj*)

forward(*x*, *adj*)

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.

---

property model_type

reset_parameters()

training: bool

2.4 grb.evaluator

2.4.1 grb.evaluator.evaluator

Evaluator Module for Unified Evaluation of Attacks vs. Defenses.

class grb.evaluator.evaluator.**AttackEvaluator**(*dataset*, *build_model*, *device='cpu'*)

Bases: object

Evaluator used to evaluate the attack performance on a dataset across different models.

Parameters

- **dataset** (*grb.dataset.Dataset or grb.dataset.CustomDataset*) – GRB supported dataset.
- **build_model** (*func*) – Function that builds a model with specific configuration.
- **device** (*str, optional*) – Device used to host data. Default: cpu.

eval(*model*, *adj*, *features*, *adj_norm_func=None*)

Evaluate attack results on a single model.

Parameters

- **model** (*torch.nn.module*) – Model implemented based on torch.nn.module.
- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of N * N sparse matrix.
- **features** (*torch.FloatTensor*) – Features in form of N * D torch float tensor.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: None.

Returns
test_score – The test score of the model on input adjacency matrix and features.

Return type
float

eval_attack(*model_dict*, *adj_attack*, *features_attack*, *verbose=False*)

Evaluate attack results on single/multiple model(s).

Parameters

- **model_dict** (*dict*) – Dictionary in form of {'model_name', 'model_path'}. model_name should be compatible with build_model func.
- **adj_attack** (*scipy.sparse.csr.csr_matrix*) – Adversarial adjacency matrix in form of N * N sparse matrix.
- **features_attack** (*torch.FloatTensor*) – Features of nodes after attacks in form of N * D torch float tensor.
- **verbose** (*bool, optional*) – Whether to display logs. Default: False.

Returns
test_score_dict – Dictionary in form of {'model_name', 'evaluation score'}.

Return type
dict

static eval_metric(*test_score_sorted*, *metric_type='polynomial'*, *order='a'*)

Parameters

- **test_score_sorted** – Array of sorted test scores.
- **metric_type** (*str, optional*) – Type of metric. Default: `polynomial`.
- **order** (*str, optional*) – Ascending order `a` or descending order `d`. Default: `a`.

Returns
final_score – Final general score across methods.

Return type
float

class grb.evaluator.evaluator.**DefenseEvaluator**(*dataset*, *build_model*, *device='cpu'*)

Bases: `object`

2.4.2 grb.evaluator.metric

Evaluation metrics

grb.evaluator.metric.**eval_acc**(*pred*, *labels*, *mask=None*)

Accuracy metric for node classification.

Parameters

- **pred** (*torch.Tensor*) – Output logits of model in form of `N * 1`.
- **labels** (*torch.LongTensor*) – Labels in form of `N * 1`.
- **mask** (*torch.Tensor, optional*) – Mask of nodes to evaluate in form of `N * 1` torch bool tensor. Default: `None`.

Returns
acc – Node classification accuracy.

Return type
float

grb.evaluator.metric.**eval_f1multilabel**(*pred*, *labels*, *mask=None*)

F1 score for multi-label node classification.

Parameters

- **pred** (*torch.Tensor*) – Output logits of model in form of `N * 1`.
- **labels** (*torch.LongTensor*) – Labels in form of `N * 1`.
- **mask** (*torch.Tensor, optional*) – Mask of nodes to evaluate in form of `N * 1` torch bool tensor. Default: `None`.

Returns
f1 – Average F1 score across different labels.

Return type
float

grb.evaluator.metric.**eval_rocauc**(*pred*, *labels*, *mask=None*)

ROC-AUC score for multi-label node classification.

Parameters

- **pred** (*torch.Tensor*) – Output logits of model in form of `N * 1`.
- **labels** (*torch.LongTensor*) – Labels in form of `N * 1`.
- **mask** (*torch.Tensor, optional*) – Mask of nodes to evaluate in form of `N * 1` torch bool tensor. Default: `None`.

Returns
rocauc – Average ROC-AUC score across different labels.

Return type
float

grb.evaluator.metric.**get_weights_arithmetic**(*n*, *w_1*, *order='a'*)

Arithmetic weights for calculating weighted robust score.

Parameters

- **n** (*int*) – Number of scores.
- **w_1** (*float*) – Initial weight of the first term.
- **order** (*str, optional*) – `a` for ascending order, `d` for descending order. Default: `a`.

Returns
weights – List of weights.

Return type
list

grb.evaluator.metric.**get_weights_polynomial**(*n*, *p=2*, *order='a'*)

Arithmetic weights for calculating weighted robust score.

Parameters

- **n** (*int*) – Number of scores.
- **p** (*float*) – Power of denominator.
- **order** (*str, optional*) – `a` for ascending order, `d` for descending order. Default: `a`.

Returns
weights_norms – List of normalized polynomial weights.

Return type
list

2.5 grb.model

2.5.1 grb.model.cogdl

2.5.2 grb.model.dgl

grb.model.dgl.gat

class grb.model.dgl.gat.**GAT**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *n_heads*, *activation=<function leaky_relu>*, *layer_norm=False*, *feat_norm=None*, *adj_norm_func=None*, *feat_dropout=0.0*, *attn_dropout=0.0*, *residual=False*, *dropout=0.0*)

Bases: Module

Graph Attention Networks (GAT)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: False.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: torch.nn.functional.leaky_relu.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: None.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: None.
- **feat_dropout** (*float, optional*) – Dropout rate for input features. Default: 0.0.
- **attn_dropout** (*float, optional*) – Dropout rate for attention. Default: 0.0.
- **residual** (*bool, optional*) – Whether to use residual connection. Default: False.
- **dropout** (*float, optional*) – Dropout rate during training. Default: 0.0.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of layer.

Return type
torch.Tensor

property model_name

property model_type

training: bool

grb.model.dgl.gcn

class grb.model.dgl.gcn.**GCN**(*in_features*, *out_features*, *hidden_features*, *activation=<function relu>*, *layer_norm=False*)

Bases: Module

forward(*x*, *adj*, *dropout=0*)

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.

---

property model_type

training: bool

grb.model.dgl.gin

class grb.model.dgl.gin.**ApplyNodeFunc**(*mlp*)

Bases: Module

Update the node feature hv with MLP, BN and ReLU.

forward(*h*)

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.

---

training: bool

class grb.model.dgl.gin.**GIN**(*in_features*, *hidden_features*, *out_features*, *learn_eps=True*, *neighbor_pooling_type='sum'*, *num_mlp_layers=1*)

Bases: Module

GIN model

forward(*x*, *adj*, *dropout=0*)

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.

---

property model_type

training: bool

class grb.model.dgl.gin.MLP(*num_layers*, *input_dim*, *hidden_dim*, *output_dim*)

Bases: Module

MLP with linear output

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.

training: bool

grb.model.dgl.grand

class grb.model.dgl.grand.GRAND(*in_features*, *out_features*, *hidden_features*, *n_layers=2*, *s=1*, *k=3*, *temp=1.0*, *lam=1.0*, *feat_norm=None*, *adj_norm_func=None*, *node_dropout=0.0*, *input_dropout=0.0*, *hidden_dropout=0.0*)

Bases: Module

Graph Random Neural Networks (GRAND)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **s** (*int*) – Number of Augmentation samples
- **k** (*int*) – Number of Propagation Steps
- **node_dropout** (*float*) – Dropout rate on node features.
- **input_dropout** (*float*) – Dropout rate of the input layer of a MLP
- **hidden_dropout** (*float*) – Dropout rate of the hidden layer of a MLP

forward(*x*, *adj*)

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.

property model_name

property model_type

training: bool

grb.model.dgl.grand.**GRANDConv**(*graph*, *features*, *hop*)

Parameters

- **graph** (*dgl.Graph*) – The input graph
- **features** (*torch.Tensor*) – Tensor of node features
- **hop** (*int*) – Propagation Steps

grb.model.dgl.grand.**drop_node**(*features*, *drop_rate*, *training=True*)

2.5.3 grb.model.torch

grb.model.torch.appnp

Torch module for APPNP.

class grb.model.torch.appnp.**APPNP**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *layer_norm=False*, *activation=<function relu>*, *edge_drop=0.0*, *alpha=0.01*, *k=10*, *feat_norm=None*, *adj_norm_func=<function GCNAdjNorm>*, *dropout=0.0*)

Bases: Module

Approximated Personalized Propagation of Neural Predictions (APPNP)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: False.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: torch.nn.functional.relu.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: None.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: GCNAdjNorm.
- **edge_drop** (*float, optional*) – Rate of edge drop.
- **alpha** (*float, optional*) – Hyper-parameter, refer to original paper. Default: 0.01.
- **k** (*int, optional*) – Hyper-parameter, refer to original paper. Default: 10.
- **dropout** (*float, optional*) – Dropout rate during training. Default: 0.0.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of model (logits without activation).

Return type
torch.Tensor

property model_name

property model_type
Indicate type of implementation.

reset_parameters()
Reset parameters.

training: bool

class grb.model.torch.appnp.**SparseEdgeDrop**(*edge_drop*)
Bases: Module

Sparse implementation of edge drop.

Parameters
edge_drop (*float*) – Rate of edge drop.

forward(*adj*)
Sparse edge drop

training: bool

grb.model.torch.gcn

Torch module for GCN.

class grb.model.torch.gcn.**GCN**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *activation=<function relu>*, *layer_norm=False*, *residual=False*, *feat_norm=None*, *adj_norm_func=<function GCNAdjNorm>*, *dropout=0.0*)
Bases: Module

Graph Convolutional Networks (GCN)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: False.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: torch.nn.functional.relu.

- **residual** (*bool, optional*) – Whether to use residual connection. Default: False.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: None.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: GCNAdjNorm.
- **dropout** (*float, optional*) – Dropout rate during training. Default: 0.0.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of model (logits without activation).

Return type
torch.Tensor

property model_name

property model_type
Indicate type of implementation.

reset_parameters()
Reset parameters.

training: bool

class grb.model.torch.gcn.**GCNConv**(*in_features*, *out_features*, *activation=None*, *residual=False*, *dropout=0.0*)

Bases: Module

GCN convolutional layer.

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: None.
- **residual** (*bool, optional*) – Whether to use residual connection. Default: False.
- **dropout** (*float, optional*) – Dropout rate during training. Default: 0.0.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of layer.

Return type
torch.Tensor

reset_parameters()

Reset parameters.

training: bool

class grb.model.torch.gcn.GCNGC(*in_features*, *out_features*, *hidden_features*, *n_layers*, *activation=<function relu>*, *layer_norm=False*, *residual=False*, *feat_norm=None*, *adj_norm_func=<function GCNAdjNorm>*, *dropout=0.0*)

Bases: Module

Graph Convolutional Networks (GCN)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: False.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: torch.nn.functional.relu.
- **residual** (*bool, optional*) – Whether to use residual connection. Default: False.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: None.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: GCNAdjNorm.
- **dropout** (*float, optional*) – Dropout rate during training. Default: 0.0.

forward(*x*, *adj*, *batch_index=None*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns

x – Output of model (logits without activation).

Return type

torch.Tensor

property model_name

property model_type

Indicate type of implementation.

reset_parameters()

Reset parameters.

training: bool

grb.model.torch.gin

Torch module for GIN.

class grb.model.torch.gin.**GIN**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *n_mlp_layers=2*, *activation=<function relu>*, *layer_norm=False*, *batch_norm=True*, *eps=0.0*, *feat_norm=None*, *adj_norm_func=None*, *dropout=0.0*)

Bases: `Module`

Graph Isomorphism Network (GIN)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **n_mlp_layers** (*int*) – Number of layers.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: `False`.
- **batch_norm** (*bool, optional*) – Whether to apply batch normalization. Default: `True`.
- **eps** (*float, optional*) – Hyper-parameter, refer to original paper. Default: `0.0`.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: `torch.nn.functional.relu`.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: `None`.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: `None`.
- **dropout** (*float, optional*) – Rate of dropout. Default: `0.0`.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of model (logits without activation).

Return type
torch.Tensor

property model_name

property model_type

Indicate type of implementation.

reset_parameters()

Reset parameters.

training: **bool**

class grb.model.torch.gin.**GINConv**(*in_features*, *out_features*, *activation=<function relu>*, *eps=0.0*, *batch_norm=True*, *dropout=0.0*)

Bases: Module

GIN convolutional layer.

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: None.
- **eps** (*float, optional*) – Hyper-parameter, refer to original paper. Default: 0.0.
- **batch_norm** (*bool, optional*) – Whether to apply batch normalization. Default: True.
- **dropout** (*float, optional*) – Rate of dropout. Default: 0.0.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of layer.

Return type
torch.Tensor

reset_parameters()

Reset parameters.

training: bool

grb.model.torch.graphsage

Torch module for GraphSAGE.

class grb.model.torch.graphsage.**GraphSAGE**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *activation=<function relu>*, *layer_norm=False*, *feat_norm=None*, *adj_norm_func=<function SAGEAdjNorm>*, *mu=2.0*, *dropout=0.0*)

Bases: Module

Inductive Representation Learning on Large Graphs (GraphSAGE)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **n_layers** (*int*) – Number of layers.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: False.

- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: `torch.nn.functional.relu`.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: `None`.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: `SAGEAdjNorm`.
- **mu** (*float, optional*) – Hyper-parameter, refer to original paper. Default: `2.0`.
- **dropout** (*float, optional*) – Rate of dropout. Default: `0.0`.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of model (logits without activation).

Return type
torch.Tensor

property model_name

property model_type

Indicate type of implementation.

reset_parameters()

Reset parameters.

training: bool

class grb.model.torch.graphsage.**SAGEConv**(*in_features*, *pool_features*, *out_features*, *activation=None*, *mu=2.0*, *dropout=0.0*)

Bases: `Module`

SAGE convolutional layer.

Parameters

- **in_features** (*int*) – Dimension of input features.
- **pool_features** (*int*) – Dimension of pooling features.
- **out_features** (*int*) – Dimension of output features.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: `None`.
- **dropout** (*float, optional*) – Rate of dropout. Default: `0.0`.
- **mu** (*float, optional*) – Hyper-parameter, refer to original paper. Default: `2.0`.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of layer.

Return type
torch.Tensor

`reset_parameters()`

Reset parameters.

`training: bool`

grb.model.torch.robustgcn

grb.model.torch.sgcn

Torch module for SGCN.

`class grb.model.torch.sgcn.SGCN(`*in_features*, *out_features*, *hidden_features*, *n_layers*, *activation=<built-in method tanh of type object>*, *feat_norm=None*, *adj_norm_func=<function GCNAdjNorm>*, *layer_norm=False*, *batch_norm=False*, *k=4*, *dropout=0.0*`)`

Bases: `Module`

Simplifying Graph Convolutional Networks (SGCN)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: `False`.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: `torch.tanh`.
- **k** (*int, optional*) – Hyper-parameter, refer to original paper. Default: `4`.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: `None`.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: `GCNAdjNorm`.
- **dropout** (*float, optional*) – Rate of dropout. Default: `0.0`.

`forward(`*x*, *adj*`)`

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of model (logits without activation).

Return type
torch.Tensor

property model_name

property model_type

Indicate type of implementation.

training: bool

class grb.model.torch.sgcn.**SGConv**(*in_features*, *out_features*, *k*)

Bases: Module

SGCN convolutional layer.

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **k** (*int, optional*) – Hyper-parameter, refer to original paper. Default: 4.

Returns
x – Output of layer.

Return type
torch.Tensor

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of layer.

Return type
torch.Tensor

training: bool

grb.model.torch.tagcn

Torch module for TAGCN.

class grb.model.torch.tagcn.**TAGCN**(*in_features*, *out_features*, *hidden_features*, *n_layers*, *k*, *activation=<function leaky_relu>*, *feat_norm=None*, *adj_norm_func=<function GCNAdjNorm>*, *layer_norm=False*, *batch_norm=False*, *dropout=0.0*)

Bases: Module

Topological Adaptive Graph Convolutional Networks (TAGCN)

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.

- **hidden_features** (*int or list of int*) – Dimension of hidden features. List if multi-layer.
- **n_layers** (*int*) – Number of layers.
- **k** (*int*) – Hyper-parameter, k-hop adjacency matrix, refer to original paper.
- **layer_norm** (*bool, optional*) – Whether to use layer normalization. Default: `False`.
- **batch_norm** (*bool, optional*) – Whether to apply batch normalization. Default: `False`.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: `torch.nn.functional.leaky_relu`.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: `None`.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: `GCNAdjNorm`.
- **dropout** (*float, optional*) – Rate of dropout. Default: `0.0`.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of model (logits without activation).

Return type
torch.Tensor

property model_name

property model_type
Indicate type of implementation.

reset_parameters()
Reset paramters.

training: bool

class grb.model.torch.tagcn.**TAGConv**(*in_features*, *out_features*, *k=2*, *activation=None*, *batch_norm=False*, *dropout=0.0*)

Bases: `Module`

TAGCN convolutional layer.

Parameters

- **in_features** (*int*) – Dimension of input features.
- **out_features** (*int*) – Dimension of output features.
- **k** (*int, optional*) – Hyper-parameter, refer to original paper. Default: `2`.
- **activation** (*func of torch.nn.functional, optional*) – Activation function. Default: `None`.
- **batch_norm** (*bool, optional*) – Whether to apply batch normalization. Default: `False`.
- **dropout** (*float, optional*) – Rate of dropout. Default: `0.0`.

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – Tensor of input features.
- **adj** (*torch.SparseTensor*) – Sparse tensor of adjacency matrix.

Returns
x – Output of layer.

Return type
torch.Tensor

reset_parameters()
Reset parameters.

training: bool

2.6 grb.utils

2.6.1 grb.utils.normalize

grb.utils.normalize.**GCNAdjNorm**(*adj*, *order=-0.5*)
Normalization of adjacency matrix proposed in GCN.

Parameters

- **adj** (*scipy.sparse.csr.csr_matrix or torch.FloatTensor*) – Adjacency matrix in form of `N * N` sparse matrix (or in form of `N * N` dense tensor).
- **order** (*float, optional*) – Order of degree matrix. Default: `-0.5`.

Returns
adj – Normalized adjacency matrix in form of `N * N` sparse matrix.

Return type
scipy.sparse.csr.csr_matrix

grb.utils.normalize.**RobustGCNAdjNorm**(*adj*)
Normalization of adjacency matrix proposed in RobustGCN.

Parameters
adj (*tuple of scipy.sparse.csr.csr_matrix*) – Tuple of adjacency matrix in form of `N * N` sparse matrix.

Returns

- **adj0** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.
- **adj1** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.

grb.utils.normalize.**SAGEAdjNorm**(*adj*, *order=-1*)
Normalization of adjacency matrix proposed in GraphSAGE.

Parameters

- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.
- **order** (*float, optional*) – Order of degree matrix. Default: `-0.5`.

Returns
adj – Normalized adjacency matrix in form of `N * N` sparse matrix.

Return type
scipy.sparse.csr.csr_matrix

grb.utils.normalize.**SPARSEAdjNorm**(*adj*, *order=-0.5*)

Normalization of adjacency matrix proposed in GCN.

Parameters

- **adj** (*scipy.sparse.csr.csr_matrix or torch.FloatTensor*) – Adjacency matrix in form of `N * N` sparse matrix (or in form of `N * N` dense tensor).
- **order** (*float, optional*) – Order of degree matrix. Default: `-0.5`.

Returns
adj – Normalized adjacency matrix in form of `N * N` sparse matrix.

Return type
scipy.sparse.csr.csr_matrix

grb.utils.normalize.**feature_normalize**(*features*)

2.6.2 grb.utils.trainer

2.6.3 grb.utils.utils

grb.utils.utils.**adj_preprocess**(*adj*, *adj_norm_func=None*, *mask=None*, *model_type='torch'*, *device='cpu'*)

Preprocess the adjacency matrix.

Parameters

- **adj** (*scipy.sparse.csr.csr_matrix or a tuple*) – Adjacency matrix in form of `N * N` sparse matrix.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: `None`.
- **mask** (*torch.Tensor, optional*) – Mask of nodes in form of `N * 1` torch bool tensor. Default: `None`.
- **model_type** (*str, optional*) – Type of model's backend, choose from ["torch", "cogdl", "dgl"]. Default: `"torch"`.
- **device** (*str, optional*) – Device used to host data. Default: `cpu`.

Returns
adj – Adjacency matrix in form of `N * N` sparse tensor or a tuple.

Return type
torch.Tensor or a tuple

grb.utils.utils.**adj_to_tensor**(*adj*)

Convert adjacency matrix in scipy sparse format to torch sparse tensor.

Parameters
adj (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.

Returns
adj_tensor – Adjacency matrix in form of `N * N` sparse tensor.

Return type
torch.Tensor

grb.utils.utils.**build_adj**(*attr*, *edge_index*, *adj_type='csr'*)

grb.utils.utils.**check_feat_range**(*features*, *feat_lim_min*, *feat_lim_max*)

Check if the generated features are within the limited range.

Parameters

- **features** (*torch.Tensor*) – Features in form of torch tensor.
- **feat_lim_min** (*float*) – Minimum limit of feature range.
- **feat_lim_max** (*float*) – Maximum limit of feature range.

Return type
bool

grb.utils.utils.**check_symmetry**(*adj*)

Check if the adjacency matrix is symmetric.

Parameters
adj (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.

Return type
bool

grb.utils.utils.**download**(*url*, *save_path*)

Download dataset from URL.

Parameters

- **url** (*str*) – URL to the dataset.
- **save_path** (*str*) – Path to save the downloaded dataset.

grb.utils.utils.**evaluate**(*model*, *features*, *adj*, *labels*, *feat_norm=None*, *adj_norm_func=None*, *eval_metric=<function eval_acc>*, *mask=None*, *device='cpu'*)

Parameters

- **model** (*torch.nn.module*) – Model implemented based on `torch.nn.module`.
- **features** (*torch.Tensor or numpy.array*) – Features in form of torch tensor or numpy array.
- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of `N * N` sparse matrix.
- **labels** (*torch.Tensor or numpy.array*) – Labels in form of torch tensor or numpy array.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: `None`.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: `None`.
- **eval_metric** (*func of grb.metric, optional*) – Evaluation metric, like accuracy or F1 score. Default: `grb.metric.eval_acc`.
- **mask** (*torch.tensor, optional*) – Mask of target nodes. Default: `None`.
- **device** (*str, optional*) – Device used to host data. Default: `cpu`.

Returns
score – Score on masked nodes.

Return type
float

grb.utils.utils.**feat_preprocess**(*features*, *feat_norm=None*, *device='cpu'*)

Preprocess the features.

Parameters

- **features** (*torch.Tensor or numpy.array*) – Features in form of torch tensor or numpy array.
- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: None.
- **device** (*str, optional*) – Device used to host data. Default: cpu.

Returns
features – Features in form of torch tensor on chosen device.

Return type
torch.Tensor

grb.utils.utils.**fix_seed**(*seed=0*)

Fix random process by a seed.

Parameters
seed (*int, optional*) – Random seed. Default: 0.

grb.utils.utils.**get_index_induc**(*index_a*, *index_b*)

Get index under the inductive training setting.

Parameters

- **index_a** (*tuple*) – Tuple of index.
- **index_b** (*tuple*) – Tuple of index.

Returns

- **index_a_new** (*tuple*) – Tuple of mapped index.
- **index_b_new** (*tuple*) – Tuple of mapped index.

grb.utils.utils.**get_num_params**(*model*)

Convert scipy sparse matrix to torch sparse tensor.

Parameters
model (*torch.nn.module*) – Model implemented based on torch.nn.module.

grb.utils.utils.**inference**(*model*, *features*, *adj*, *feat_norm=None*, *adj_norm_func=None*, *device='cpu'*)

Inference of model.

Parameters

- **model** (*torch.nn.module*) – Model implemented based on torch.nn.module.
- **features** (*torch.Tensor or numpy.array*) – Features in form of torch tensor or numpy array.
- **adj** (*scipy.sparse.csr.csr_matrix*) – Adjacency matrix in form of N * N sparse matrix.

- **feat_norm** (*str, optional*) – Type of features normalization, choose from ["arctan", "tanh", None]. Default: None.
- **adj_norm_func** (*func of utils.normalize, optional*) – Function that normalizes adjacency matrix. Default: None.
- **device** (*str, optional*) – Device used to host data. Default: cpu.

Returns
logits – Output logits of model.

Return type
torch.Tensor

grb.utils.utils.**label_preprocess**(*labels*, *device='cpu'*)

Convert labels to torch tensor.

Parameters

- **labels** (*torch.Tensor*) – Labels in form of torch tensor.
- **device** (*str, optional*) – Device used to host data. Default: cpu.

Returns
labels – Features in form of torch tensor on chosen device.

Return type
torch.Tensor

grb.utils.utils.**save_adj**(*adj*, *file_dir*, *file_name='adj.pkl'*)

Save generated adversarial adjacency matrix.

Parameters

- **adj** (*scipy.sparse.csr.csr_matrix or a tuple*) – Adjacency matrix in form of N * N sparse matrix.
- **file_dir** (*str*) – Directory to save the file.
- **file_name** (*str, optional*) – Name of file to save. Default: adj.pkl.

grb.utils.utils.**save_df_to_csv**(*df*, *file_dir*, *file_name='result.csv'*, *verbose=False*)

Save dataframe to .csv file.

Parameters

- **df** (*pandas.DataFrame*) – Dataframe containing evaluation results.
- **file_dir** (*str*) – Directory to save the file.
- **file_name** (*str, optional*) – Name of saved file. Default: result.csv.
- **verbose** (*bool, optional*) – Whether to display logs. Default: False.

grb.utils.utils.**save_df_to_xlsx**(*df*, *file_dir*, *file_name='result.xlsx'*, *verbose=False*)

Save dataframe to .xlsx file.

Parameters

- **df** (*pandas.DataFrame*) – Dataframe containing evaluation results.
- **file_dir** (*str*) – Directory to save the file.
- **file_name** (*str, optional*) – Name of saved file. Default: result.xlsx.
- **verbose** (*bool, optional*) – Whether to display logs. Default: False.

grb.utils.utils.**save_dict_to_json**(*result_dict*, *file_dir*, *file_name*, *verbose=False*)

Save dictinary to .json file.

Parameters

- **result_dict** (*dict*) – Dictionary containing evaluation results.
- **file_dir** (*str*) – Directory to save the file.
- **file_name** (*str*) – Name of saved file.
- **verbose** (*bool, optional*) – Whether to display logs. Default: `False`.

grb.utils.utils.**save_dict_to_xlsx**(*result_dict*, *file_dir*, *file_name='result.xlsx'*, *index=0*, *verbose=False*)

Save result dictionary to .xlsx file.

Parameters

- **result_dict** (*dict*) – Dictionary containing evaluation results.
- **file_dir** (*str*) – Directory to save the file.
- **file_name** (*str, optional*) – Name of saved file. Default: `result.xlsx`.
- **index** (*int, optional*) – Index of dataframe. Default: `0`.
- **verbose** (*bool, optional*) – Whether to display logs. Default: `False`.

grb.utils.utils.**save_features**(*features*, *file_dir*, *file_name='features.npy'*)

Save generated adversarial features.

Parameters

- **features** (*torch.Tensor or numpy.array*) – Features in form of torch tensor or numpy array.
- **file_dir** (*str*) – Directory to save the file.
- **file_name** (*str, optional*) – Name of file to save. Default: `features.npy`.

grb.utils.utils.**save_model**(*model*, *save_dir*, *name*, *verbose=True*)

Save trained model.

Parameters

- **model** (*torch.nn.module*) – Model implemented based on `torch.nn.module`.
- **save_dir** (*str*) – Directory to save the model.
- **name** (*str*) – Name of saved model.
- **verbose** (*bool, optional*) – Whether to display logs. Default: `False`.

2.6.4 grb.utils.visualize

grb.utils.visualize.**plot_graph**(*adj*, *pos*, *labels*, *nodelist=None*, *figsize=(12, 12)*, *title=None*)

Graph Robustness Benchmark (GRB) provides scalable, general, unified, and reproducible evaluation on the adversarial robustness of graph machine learning, especially Graph Neural Networks (GNNs). GRB has elaborated datasets, unified evaluation pipeline, reproducible leaderboards, and modular coding framework, which facilitates a fair comparison among various attacks & defenses on GNNs and promotes future research in this field.

CHAPTER

THREE

INDEX

- genindex

PYTHON MODULE INDEX

g

INDEX

G

I

L

M

N

O

P

R

S

T

U

V