

---

# CogDL Documentation

**KEG**

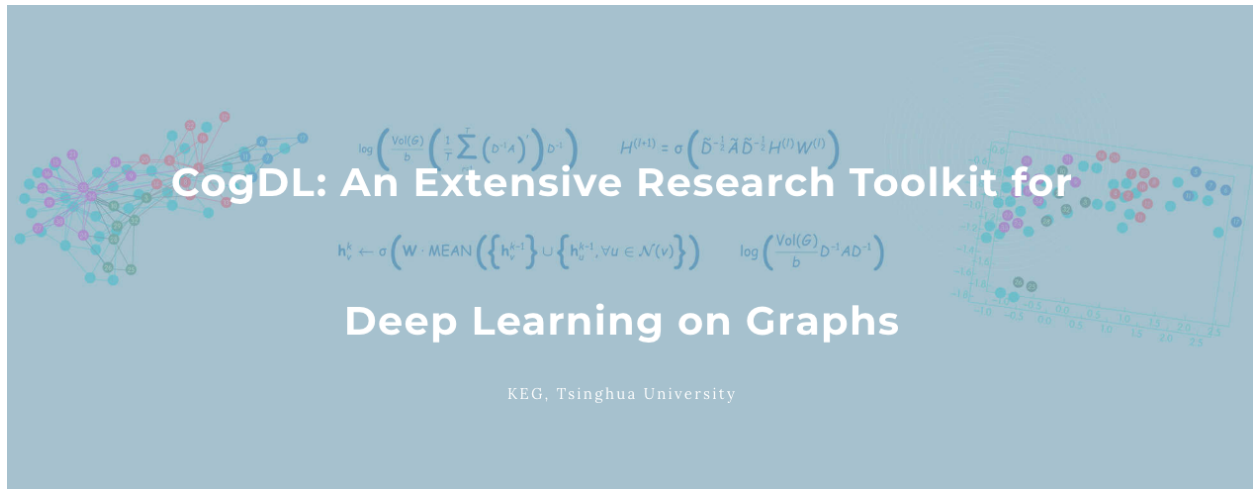
**Mar 03, 2021**



## CONTENTS:

|          |   |            |
|----------|---|------------|
| <b>1</b> | <b>Install</b>                              | <b>3</b>   |
| <b>2</b> | <b>Tutorial</b>                             | <b>5</b>   |
| 2.1      | Create a model . . . . .                    | 5          |
| 2.2      | Create a dataset . . . . .                  | 6          |
| 2.3      | Create a task . . . . .                     | 6          |
| 2.4      | Combine model, dataset and task . . . . .   | 7          |
| <b>3</b> | <b>Tasks</b>                                | <b>9</b>   |
| 3.1      | Node Classification . . . . .               | 9          |
| 3.2      | Unsupervised Node Classification . . . . .  | 11         |
| 3.3      | Supervised Graph Classification . . . . .   | 14         |
| 3.4      | Unsupervised Graph Classification . . . . . | 16         |
| <b>4</b> | <b>License</b>                              | <b>19</b>  |
| <b>5</b> | <b>Citing</b>                               | <b>21</b>  |
| <b>6</b> | <b>API Reference</b>                        | <b>23</b>  |
| 6.1      | cogdl . . . . .                             | 23         |
| <b>7</b> | <b>Indices and tables</b>                   | <b>135</b> |
|          | <b>Python Module Index</b>                  | <b>137</b> |
|          | <b>Index</b>                                | <b>139</b> |





CogDL is a graph representation learning toolkit that allows researchers and developers to easily train and compare baseline or custom models for node classification, link prediction and other tasks on graphs. It provides implementations of many popular models, including: non-GNN Baselines like Deepwalk, LINE, NetMF, GNN Baselines like GCN, GAT, GraphSAGE.

CogDL provides these features:

- Task-Oriented: CogDL focuses on tasks on graphs and provides corresponding models, datasets, and leaderboards.
- Easy-Running: CogDL supports running multiple experiments simultaneously on multiple models and datasets under a specific task using multiple GPUs.
- Multiple Tasks: CogDL supports node classification and link prediction tasks on homogeneous/heterogeneous networks, as well as graph classification.
- Extensibility: You can easily add new datasets, models and tasks and conduct experiments for them!
- Supported tasks:
  - Node classification
  - Link prediction
  - Graph classification
  - Community detection (testing)
  - Social influence prediction (testing)
  - Graph reasoning (todo)
  - Graph pre-training (todo)
  - Combinatorial optimization on graphs (todo)



## INSTALL

- PyTorch version  $\geq 1.0.0$
- Python version  $\geq 3.6$
- PyTorch Geometric (optional)

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

Please follow the instructions here to install PyTorch Geometric: [https://github.com/rusty1s/pytorch\\_geometric/#installation](https://github.com/rusty1s/pytorch_geometric/#installation).

Install other dependencies:

```
>>> pip install -e .
```





## TUTORIAL

This guide can help you start working with CogDL.

### 2.1 Create a model

Here, 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):
    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
```

## 2.2 Create a 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 your dataset is same as the 'ppi' dataset, which contains two matrices: 'network' and 'group', you can register your dataset directly use above code.

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

## 2.3 Create a task

In order to evaluate some methods on several datasets, we can build a task and evaluate learned representation. 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
```

we 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."""

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

(continues on next page)

(continued from previous page)

```

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())
    )

```

## 2.4 Combine model, dataset and task

After create your model, dataset and task, we could combine them together to learn representation from a model on a dataset and evaluate its performance according to a task. We use 'build\_model', 'build\_dataset', 'build\_task' method to build them with coresponding 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 test_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'

```

(continues on next page)

(continued from previous page)

```
# build model, dataset and task
dataset = build_dataset(args)
model = build_model(args)
task = build_task(args)

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

## 3.1 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.

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 according to args.

```
self.device = torch.device('cpu' if args.cpu else 'cuda')
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
```

After that, we can build model and use *Adam* to optimize the model.

```
model = build_model(args)
self.model = model.to(self.device)
self.patience = args.patience
self.max_epoch = args.max_epoch
self.optimizer = torch.optim.Adam(
    self.model.parameters(), lr=args.lr, weight_decay=args.weight_decay
)
```

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* to compute the accuracy and loss.

```
def train(self):
    epoch_iter = tqdm(range(self.max_epoch))
    patience = 0
```

(continues on next page)

```

best_score = 0
best_loss = np.inf
max_score = 0
min_loss = np.inf
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}"
    )
    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

def _train_step(self):
    self.model.train()
    self.optimizer.zero_grad()
    self.model.loss(self.data).backward()
    self.optimizer.step()

def _test_step(self, split="val"):
    self.model.eval()
    logits = self.model.predict(self.data)
    _, mask = list(self.data(f"{split}_mask"))[0]
    loss = F.nll_loss(logits[mask], self.data.y[mask])

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

```

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_
↳ pyg_gcn pyg_gat --seed 0 1 --max-epoch 500

```

Then We get experimental results like this:

| Variant                 | Acc           |
|-------------------------|---------------|
| ('cora', 'pyg_gcn')     | 0.7785±0.0165 |
| ('cora', 'pyg_gat')     | 0.7925±0.0045 |
| ('citeseer', 'pyg_gcn') | 0.6535±0.0195 |
| ('citeseer', 'pyg_gat') | 0.6675±0.0025 |

## 3.2 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 to measure the performance.

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 issubclass(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.model_name = args.model
self.hidden_size = args.hidden_size
self.num_shuffle = args.num_shuffle
self.save_dir = args.save_dir
self.enhance = args.enhance
self.args = args
self.is_weighted = self.data.edge_attr is not None
```

(continues on next page)

(continued from previous page)

```

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 can improve the quality of representation learned from other methods, so we can use `enhance_emb` to enhance performance.

```

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

def enhance_emb(self, G, embs):
    A = sp.csr_matrix(nx.adjacency_matrix(G))
    self.args.model = 'prone'
    self.args.step, self.args.theta, self.args.mu = 5, 0.5, 0.2
    model = build_model(self.args)
    embs = model._chebyshev_gaussian(A, embs)
    return embs

```

When the embeddings are obtained, we can save them at `self.save_dir`.

```

# Map node2id
features_matrix = np.zeros((self.num_nodes, self.hidden_size))
for vid, node in enumerate(G.nodes()):
    features_matrix[node] = embeddings[vid]

self.save_emb(features_matrix)

def save_emb(self, embs):
    name = os.path.join(self.save_dir, self.model_name + '_emb.npy')
    np.save(name, embs)

```

At last, we evaluate embedding via run `num_shuffle` times classification under different training ratio with `features_matrix` and `label_matrix`.

```

return self._evaluate(features_matrix, label_matrix, self.num_shuffle)

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]

```

(continues on next page)



(continued from previous page)

```

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.

```

X, y = shuf

training_size = int(train_percent * self.num_nodes)

X_train = X[:training_size, :]
y_train = y[:training_size, :]

X_test = X[training_size:, :]
y_test = y[training_size:, :]

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.

```

return dict(
    (
        f"Micro-F1 {train_percent}",
        sum(all_results[train_percent]) / len(all_results[train_percent]),
    )
    for train_percent in sorted(all_results.keys())
)

```

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 |

### 3.3 Supervised Graph Classification

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

#### Task Design

- Set up “SupervisedGraphClassification” class, which has two specific parameters.
  - degree-feature*: Use one-hot node degree as node feature, for datasets such as lmbd-binary and lmbd-multi, which don't have node features.
  - gamma*: Multiplicative factor of learning rate decay.
  - lr*: Learning rate.
- 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
...
```

- 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/THUDM/cogdl/blob/master/cogdl/tasks/graph\\_classification.py](https://github.com/THUDM/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 |

## 3.4 Unsupervised Graph Classification

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

### Task Design

1. 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."""

```

(continues on next page)

(continued from previous page)

```

    # 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):
        # ...

```

## 2. 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
]

```

## 3. 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

```

## 4. 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_percent:
        for shuf in shuffles:
            # ...
            clf = SVC()
            clf.fit(X_train, y_train)
            preds = clf.predict(X_test)

```

(continues on next page)

```
... # ...
```

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.

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("--nn", type=bool, default=False)
    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. `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 |

**LICENSE**

MIT License

Copyright (c) 2020

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.





## CITING

- [Perozzi et al. (2014): Deepwalk: Online learning of social representations](<http://arxiv.org/abs/1403.6652>)
- [Tang et al. (2015): Line: Large-scale information network embedding](<http://arxiv.org/abs/1503.03578>)
- [Grover and Leskovec. (2016): node2vec: Scalable feature learning for networks](<http://dl.acm.org/citation.cfm?doid=2939672.2939754>)- [Cao et al. (2015):Grarep: Learning graph representations with global structural information ](<http://dl.acm.org/citation.cfm?doid=2806416.2806512>)
- [Ou et al. (2016): Asymmetric transitivity preserving graph embedding](<http://dl.acm.org/citation.cfm?doid=2939672.2939751>)
- [Qiu et al. (2017): Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec](<http://arxiv.org/abs/1710.02971>)
- [Qiu et al. (2019): NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization](<http://arxiv.org/abs/1710.02971>)
- [Zhang et al. (2019): Spectral Network Embedding: A Fast and Scalable Method via Sparsity](<http://arxiv.org/abs/1806.02623>)
- [Kipf and Welling (2016): Semi-Supervised Classification with Graph Convolutional Networks](<https://arxiv.org/abs/1609.02907>)
- [Hamilton et al. (2017): Inductive Representation Learning on Large Graphs](<https://arxiv.org/abs/1706.02216>)
- [Veličković et al. (2017): Graph Attention Networks](<https://arxiv.org/abs/1710.10903>)
- [Ding et al. (2018): Semi-supervised Learning on Graphs with Generative Adversarial Nets](<https://arxiv.org/abs/1809.00130>)
- [Han et al. (2019): GroupRep: Unsupervised Structural Representation Learning for Groups in Networks](<https://www.overleaf.com/read/nqxjtkmmgmff>)
- [Zhang et al. (2019): Revisiting Graph Convolutional Networks: Neighborhood Aggregation and Network Sampling](<https://www.overleaf.com/read/xzykmvvhxjmxxy>)
- [Zhang et al. (2019): Co-training Graph Convolutional Networks with Network Redundancy](<https://www.overleaf.com/read/fbhqqgzqgmyn>)
- [Qiu et al. (2019): NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization](<http://keg.cs.tsinghua.edu.cn/jietang/publications/www19-Qiu-et-al-NetSMF-Large-Scale-Network-Embedding.pdf>)
- [Zhang et al. (2019): ProNE: Fast and Scalable Network Representation Learning](<https://www.overleaf.com/read/dhgpkyfhdhj>)
- [Cen et al. (2019): Representation Learning for Attributed Multiplex Heterogeneous Network](<https://arxiv.org/abs/1905.01669>)



## API REFERENCE

This page contains auto-generated API reference documentation<sup>1</sup>.

### 6.1 cogdl

#### 6.1.1 Subpackages

`cogdl.data`

##### Submodules

`cogdl.data.batch`

##### Module Contents

##### Classes

---

|              |   |
|--------------|---|
| <i>Batch</i> | A plain old python object modeling a batch of graphs as one big |
|--------------|---|

---

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

Bases: `cogdl.data.Data`

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.

**static from\_data\_list** (*data\_list, follow\_batch=[]*)

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

**cumsum** (*self, 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

---

<sup>1</sup> Created with sphinx-autoapi

process is corrupted for a specific data attribute.

---

**to\_data\_list** (*self*)

Reconstructs the list of `torch_geometric.data.Data` objects from the batch object. The batch object must have been created via `from_data_list()` in order to be able reconstruct the initial objects.

**property num\_graphs** (*self*)

Returns the number of graphs in the batch.

`cogdl.data.data`

## Module Contents

### Classes

---

|             |  |
|-------------|--|
| <i>Data</i> | A plain old python object modeling a single graph with various |
|-------------|--|

---

**class** `cogdl.data.data.Data` (*x=None, edge\_index=None, edge\_attr=None, y=None, pos=None*)

Bases: `object`

A plain old python object modeling a single graph with various (optional) attributes:

**Args:**

**x (Tensor, optional): Node feature matrix with shape :obj:`[num\_nodes, num\_node\_features]`.** (default: `None`)

**edge\_index (LongTensor, optional): Graph connectivity in COO format with shape `[2, num_edges]`.** (default: `None`)

**edge\_attr (Tensor, optional): Edge feature matrix with shape `[num_edges, num_edge_features]`.** (default: `None`)

**y (Tensor, optional): Graph or node targets with arbitrary shape.** (default: `None`)

**pos (Tensor, optional): Node position matrix with shape `[num_nodes, num_dimensions]`.** (default: `None`)

The data object is not restricted to these attributes and can be extended by any other additional data.

**static from\_dict** (*dictionary*)

Creates a data object from a python dictionary.

**\_\_getitem\_\_** (*self, key*)

Gets the data of the attribute `key`.

**\_\_setitem\_\_** (*self, key, value*)

Sets the attribute `key` to `value`.

**property keys** (*self*)

Returns all names of graph attributes.

**\_\_len\_\_** (*self*)

Returns the number of all present attributes.

**\_\_contains\_\_** (*self, key*)

Returns `True`, if the attribute `key` is present in the data.

---

`__iter__` (*self*)

Iterates over all present attributes in the data, yielding their attribute names and content.

`__call__` (*self*, \**keys*)

Iterates over all attributes \**keys* in the data, yielding their attribute names and content. If \**keys* is not given this method will iterative over all present attributes.

`cat_dim` (*self*, *key*, *value*)

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.

---

`__inc__` (*self*, *key*, *value*)

“Returns the incremental count to cumulatively increase the value of the next attribute of *key* 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.

---

`property num_edges` (*self*)

Returns the number of edges in the graph.

`property num_features` (*self*)

Returns the number of features per node in the graph.

`property num_nodes` (*self*)

`is_coalesced` (*self*)

Returns `True`, if edge indices are ordered and do not contain duplicate entries.

`apply` (*self*, *func*, \**keys*)

Applies the function *func* to all attributes \**keys*. If \**keys* is not given, *func* is applied to all present attributes.

`contiguous` (*self*, \**keys*)

Ensures a contiguous memory layout for all attributes \**keys*. If \**keys* is not given, all present attributes are ensured to have a contiguous memory layout.

`to` (*self*, *device*, \**keys*)

Performs tensor dtype and/or device conversion to all attributes \**keys*. If \**keys* is not given, the conversion is applied to all present attributes.

`cuda` (*self*, \**keys*)

`clone` (*self*)

`__repr__` (*self*)

Return `repr(self)`.

`cogdl.data.data_loader`

### Module Contents

#### Classes

---

|                        |  |
|------------------------|--|
| <i>DataLoader</i>      | Data loader which merges data objects from a |
| <i>DataListLoader</i>  | Data loader which merges data objects from a |
| <i>DenseDataLoader</i> | Data loader which merges data objects from a |

---

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

Bases: `torch.utils.data.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**)

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

Bases: `torch.utils.data.DataLoader`

Data loader which merges data objects from a `cogdl.data.dataset` to a python list.

---

**Note:** This data loader should be used for multi-gpu support via `cogdl.nn.DataParallel`.

---

**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**)

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

Bases: `torch.utils.data.DataLoader`

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

---

**Note:** To make use of this data loader, all graphs in the dataset needs to have the same shape for each its attributes. Therefore, this data loader should only be used when working with *dense* adjacency matrices.

---

**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**)

`cogdl.data.dataset`

## Module Contents

### Classes

---

|                      |   |
|----------------------|---|
| <code>Dataset</code> | Dataset base class for creating graph datasets. |
|----------------------|---|

---

### Functions

---

`to_list(x)`


---

`files_exist(files)`


---

`cogdl.data.dataset.to_list(x)``cogdl.data.dataset.files_exist(files)`

```
class cogdl.data.dataset.Dataset (root,          transform=None,          pre_transform=None,
                                pre_filter=None)
```

Bases: `torch.utils.data.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`)**property** `raw_file_names` (*self*)The name of the files to find in the `self.raw_dir` folder in order to skip the download.**property** `processed_file_names` (*self*)The name of the files to find in the `self.processed_dir` folder in order to skip the processing.**abstract** `download` (*self*)Downloads the dataset to the `self.raw_dir` folder.**abstract** `process` (*self*)Processes the dataset to the `self.processed_dir` folder.**abstract** `__len__` (*self*)

The number of examples in the dataset.

**abstract** `get (self, idx)`

Gets the data object at index `idx`.

**property** `num_features (self)`

Returns the number of features per node in the graph.

**property** `raw_paths (self)`

The filepaths to find in order to skip the download.

**property** `processed_paths (self)`

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

`_download (self)`

`_process (self)`

`__getitem__ (self, idx)`

Gets the data object at index `idx` and transforms it (in case a `self.transform` is given).

`__repr__ (self)`

**cogdl.data.download**

### Module Contents

#### Functions

---

`download_url(url, folder, name=None, log=True)` Downloads the content of an URL to a specific folder.

---

`cogdl.data.download.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. `log` (bool, optional): If `False`, will not print anything to the

console. (default: `True`)

**cogdl.data.extract**

### Module Contents

#### Functions

---

`maybe_log(path, log=True)`

---

`extract_tar(path, folder, mode='r:gz', log=True)` Extracts a tar archive to a specific folder.

---

`extract_zip(path, folder, log=True)` Extracts a zip archive to a specific folder.

---

`extract_bz2(path, folder, log=True)`

---

`extract_gz(path, folder, log=True)`

---

`cogdl.data.extract.maybe_log (path, log=True)`



`cogdl.data.extract.extract_tar` (*path, folder, mode='r:gz', log=True*)

Extracts a tar archive to a specific folder.

**Args:** `path` (string): The path to the tar archive. `folder` (string): The folder. `mode` (string, optional): The compression mode. (default: "r:gz") `log` (bool, optional): If `False`, will not print anything to the console. (default: `True`)

`cogdl.data.extract.extract_zip` (*path, folder, log=True*)

Extracts a zip archive to a specific folder.

**Args:** `path` (string): The path to the tar archive. `folder` (string): The folder. `log` (bool, optional): If `False`, will not print anything to the

console. (default: `True`)

`cogdl.data.extract.extract_bz2` (*path, folder, log=True*)

`cogdl.data.extract.extract_gz` (*path, folder, log=True*)

## `cogdl.data.makedirs`

### Module Contents

#### Functions

---

`makedirs`(*path*)

---

`cogdl.data.makedirs.makedirs` (*path*)

## `cogdl.data.sampler`

### Module Contents

#### Classes

---

`Sampler`

---

`SAINTSampler`

---

`NodeSampler`

---

`EdgeSampler`

---

`RWSampler`

---

`MRWSampler`

---

`LayerSampler`

---

**class** cogdl.data.sampler.**Sampler** (*data, args\_params*)

**sample** (*self*)

**class** cogdl.data.sampler.**SAINTSampler** (*data, args\_params*)

Bases: *cogdl.data.sampler.Sampler*

**estimate** (*self*)

**gen\_subgraph** (*self*)

**sample** (*self*)

**extract\_subgraph** (*self, edge\_idx, directed=True*)

**get\_subgraph** (*self, phase, require\_norm=True*)

Generate one minibatch for model. In the ‘train’ mode, one minibatch corresponds to one subgraph of the training graph. In the ‘valid’ or ‘test’ mode, one batch corresponds to the full graph (i.e., full-batch rather than minibatch evaluation for validation / test sets).

**Inputs:** mode str, can be ‘train’, ‘valid’, ‘test’ require\_norm boolean

**Outputs:** data Data object, modeling the sampled subgraph data.norm\_aggr aggregation normalization  
data.norm\_loss normalization normalization

**class** cogdl.data.sampler.**NodeSampler** (*data, args\_params*)

Bases: *cogdl.data.sampler.SAINTSampler*

**sample** (*self*)

**class** cogdl.data.sampler.**EdgeSampler** (*data, args\_params*)

Bases: *cogdl.data.sampler.SAINTSampler*

**sample** (*self*)

**class** cogdl.data.sampler.**RWSampler** (*data, args\_params*)

Bases: *cogdl.data.sampler.SAINTSampler*

**sample** (*self*)

**class** cogdl.data.sampler.**MRWSampler** (*data, args\_params*)

Bases: *cogdl.data.sampler.SAINTSampler*

**sample** (*self*)

**class** cogdl.data.sampler.**LayerSampler** (*data, model, params\_args*)

Bases: *cogdl.data.sampler.Sampler*

**get\_batches** (*self, train\_nodes, train\_labels, batch\_size=64, shuffle=True*)

## Package Contents

### Classes

|                |   |
|----------------|---|
| <i>Data</i>    | A plain old python object modeling a single graph with various  |
| <i>Batch</i>   | A plain old python object modeling a batch of graphs as one big |
| <i>Dataset</i> | Dataset base class for creating graph datasets.                 |

continues on next page

Table 10 – continued from previous page

|                              |  |
|------------------------------|--|
| <code>DataLoader</code>      | Data loader which merges data objects from a |
| <code>DataListLoader</code>  | Data loader which merges data objects from a |
| <code>DenseDataLoader</code> | Data loader which merges data objects from a |

## Functions

|   |   |
|---|---|
| <code>download_url(url, folder, name=None, log=True)</code>   | Downloads the content of an URL to a specific folder. |
| <code>extract_tar(path, folder, mode='r:gz', log=True)</code> | Extracts a tar archive to a specific folder.          |
| <code>extract_zip(path, folder, log=True)</code>              | Extracts a zip archive to a specific folder.          |
| <code>extract_bz2(path, folder, log=True)</code>              |   |
| <code>extract_gz(path, folder, log=True)</code>               |   |

**class** `cogdl.data.Data` (*x=None, edge\_index=None, edge\_attr=None, y=None, pos=None*)

Bases: `object`

A plain old python object modeling a single graph with various (optional) attributes:

### Args:

**x (Tensor, optional): Node feature matrix with shape `:obj: [num_nodes, num_node_features]`.** (default: `None`)

**edge\_index (LongTensor, optional): Graph connectivity in COO format with shape `[2, num_edges]`.** (default: `None`)

**edge\_attr (Tensor, optional): Edge feature matrix with shape `[num_edges, num_edge_features]`.** (default: `None`)

**y (Tensor, optional): Graph or node targets with arbitrary shape.** (default: `None`)

**pos (Tensor, optional): Node position matrix with shape `[num_nodes, num_dimensions]`.** (default: `None`)

The data object is not restricted to these attributes and can be extended by any other additional data.

**static from\_dict** (*dictionary*)

Creates a data object from a python dictionary.

**\_\_getitem\_\_** (*self, key*)

Gets the data of the attribute `key`.

**\_\_setitem\_\_** (*self, key, value*)

Sets the attribute `key` to `value`.

**property keys** (*self*)

Returns all names of graph attributes.

**\_\_len\_\_** (*self*)

Returns the number of all present attributes.

**\_\_contains\_\_** (*self, key*)

Returns `True`, if the attribute `key` is present in the data.

**\_\_iter\_\_** (*self*)

Iterates over all present attributes in the data, yielding their attribute names and content.

`__call__` (*self*, \**keys*)

Iterates over all attributes \**keys* in the data, yielding their attribute names and content. If \**keys* is not given this method will iterative over all present attributes.

`cat_dim` (*self*, *key*, *value*)

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.

---

`__inc__` (*self*, *key*, *value*)

“Returns the incremental count to cumulatively increase the value of the next attribute of *key* 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.

---

**property** `num_edges` (*self*)

Returns the number of edges in the graph.

**property** `num_features` (*self*)

Returns the number of features per node in the graph.

**property** `num_nodes` (*self*)

**is\_coalesced** (*self*)

Returns `True`, if edge indices are ordered and do not contain duplicate entries.

**apply** (*self*, *func*, \**keys*)

Applies the function *func* to all attributes \**keys*. If \**keys* is not given, *func* is applied to all present attributes.

**contiguous** (*self*, \**keys*)

Ensures a contiguous memory layout for all attributes \**keys*. If \**keys* is not given, all present attributes are ensured to have a contiguous memory layout.

**to** (*self*, *device*, \**keys*)

Performs tensor dtype and/or device conversion to all attributes \**keys*. If \**keys* is not given, the conversion is applied to all present attributes.

**cuda** (*self*, \**keys*)

**clone** (*self*)

`__repr__` (*self*)

Return `repr(self)`.

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

Bases: `cogdl.data.Data`

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.

**static** `from_data_list` (*data\_list*, *follow\_batch=[]*)

Constructs a batch object from a python list holding `torch_geometric.data.Data` objects. The

assignment vector `batch` is created on the fly. Additionally, creates assignment batch vectors for each key in `follow_batch`.

**cumsum** (*self*, *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.

---

**to\_data\_list** (*self*)

Reconstructs the list of `torch_geometric.data.Data` objects from the batch object. The batch object must have been created via `from_data_list()` in order to be able reconstruct the initial objects.

**property num\_graphs** (*self*)

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 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`)

**property raw\_file\_names** (*self*)

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

**property processed\_file\_names** (*self*)

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

**abstract download** (*self*)

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

**abstract process** (*self*)

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

**abstract \_\_len\_\_** (*self*)

The number of examples in the dataset.

**abstract get** (*self*, *idx*)

Gets the data object at index `idx`.

**property num\_features** (*self*)

Returns the number of features per node in the graph.

**property raw\_paths** (*self*)

The filepaths to find in order to skip the download.

**property processed\_paths** (*self*)

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

**\_download** (*self*)

**\_process** (*self*)

**\_\_getitem\_\_** (*self*, *idx*)

Gets the data object at index `idx` and transforms it (in case a `self.transform` is given).

**\_\_repr\_\_** (*self*)

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

Bases: `torch.utils.data.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**)

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

Bases: `torch.utils.data.DataLoader`

Data loader which merges data objects from a `cogdl.data.dataset` to a python list.

---

**Note:** This data loader should be used for multi-gpu support via `cogdl.nn.DataParallel`.

---

**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**)

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

Bases: `torch.utils.data.DataLoader`

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

---

**Note:** To make use of this data loader, all graphs in the dataset needs to have the same shape for each its attributes. Therefore, this data loader should only be used when working with *dense* adjacency matrices.

---

**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**)

`cogdl.data.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. log (bool, optional): If `False`, will not print anything to the

console. (default: `True`)

`cogdl.data.extract_tar` (*path, folder, mode='r:gz', log=True*)

Extracts a tar archive to a specific folder.

**Args:** path (string): The path to the tar archive. folder (string): The folder. mode (string, optional): The compression mode. (default: `"r:gz"`) log (bool, optional): If `False`, will not print anything to the

console. (default: `True`)

`cogdl.data.extract_zip` (*path, folder, log=True*)

Extracts a zip archive to a specific folder.

**Args:** path (string): The path to the tar archive. folder (string): The folder. log (bool, optional): If `False`, will not print anything to the

console. (default: `True`)

`cogdl.data.extract_bz2` (*path, folder, log=True*)

`cogdl.data.extract_gz` (*path, folder, log=True*)

## `cogdl.datasets`

### Submodules

`cogdl.datasets.dgl_data`

### Module Contents

#### Classes

---

*MUTAGDataset*

---

*CollabDataset*

---

*ImdbBinaryDataset*

---

*ImdbMultiDataset*

---

*ProtainsDataset*

---

**class** `cogdl.datasets.dgl_data.MUTAGDataset`

Bases: `dgl.data.tu.TUDataset`

**class** `cogdl.datasets.dgl_data.CollabDataset`

Bases: `dgl.data.tu.TUDataset`

**class** `cogdl.datasets.dgl_data.ImdbBinaryDataset`

Bases: `dgl.data.tu.TUDataset`

**class** `cogdl.datasets.dgl_data.ImdbMultiDataset`

Bases: `dgl.data.tu.TUDataset`

**class** `cogdl.datasets.dgl_data.ProtainsDataset`  
Bases: `dgl.data.tu.TUDataset`

`cogdl.datasets.gatne`

### Module Contents

#### Classes

---

|                             |   |
|-----------------------------|---|
| <code>GatneDataset</code>   | The network datasets “Amazon”, “Twitter” and “YouTube” from the |
| <code>AmazonDataset</code>  | The network datasets “Amazon”, “Twitter” and “YouTube” from the |
| <code>TwitterDataset</code> | The network datasets “Amazon”, “Twitter” and “YouTube” from the |
| <code>YouTubeDataset</code> | The network datasets “Amazon”, “Twitter” and “YouTube” from the |

---

#### Functions

---

|                                      |
|--------------------------------------|
| <code>read_gatne_data(folder)</code> |
|--------------------------------------|

---

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

**class** `cogdl.datasets.gatne.GatneDataset` (*root, name*)  
Bases: `cogdl.data.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”).

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

**property** `raw_file_names` (*self*)

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

**property** `processed_file_names` (*self*)

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

**get** (*self, idx*)

Gets the data object at index *idx*.

**download** (*self*)

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

**process** (*self*)

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

**\_\_repr\_\_** (*self*)



**class** `cogdl.datasets.gatne.AmazonDataset`

Bases: `cogdl.datasets.gatne.GatneDataset`

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”).

**class** `cogdl.datasets.gatne.TwitterDataset`

Bases: `cogdl.datasets.gatne.GatneDataset`

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”).

**class** `cogdl.datasets.gatne.YouTubeDataset`

Bases: `cogdl.datasets.gatne.GatneDataset`

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”).

`cogdl.datasets.gcc_data`

## Module Contents

### Classes

|                                |   |
|--------------------------------|---|
| <code>Edgelist</code>          | Dataset base class for creating graph datasets. |
| <code>USAAirportDataset</code> | Dataset base class for creating graph datasets. |

**class** `cogdl.datasets.gcc_data.Edgelist` (`root`, `name`)

Bases: `cogdl.data.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)

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

**property raw\_file\_names** (*self*)

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

**property processed\_file\_names** (*self*)

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

**download** (*self*)

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

**get** (*self*, *idx*)

Gets the data object at index *idx*.

**process** (*self*)

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

**class** `cogdl.datasets.gcc_data.USAAirportDataset`

Bases: `cogdl.datasets.gcc_data.Edgelist`

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`)

`cogdl.datasets.gtn_data`

## Module Contents

### Classes

|                              |  |
|------------------------------|--|
| <code>GTNDataset</code>      | The network datasets “ACM”, “DBLP” and “IMDB” from the |
| <code>ACM_GTNDataset</code>  | The network datasets “ACM”, “DBLP” and “IMDB” from the |
| <code>DBLP_GTNDataset</code> | The network datasets “ACM”, “DBLP” and “IMDB” from the |
| <code>IMDB_GTNDataset</code> | The network datasets “ACM”, “DBLP” and “IMDB” from the |

## Functions

---

|   |   |
|---|---|
| <code>untar(path, fname, deleteTar=True)</code> | Unpacks the given archive file to the same directory, then (by default) |
|---|---|

---

`cogdl.datasets.gtn_data.untar` (*path, fname, deleteTar=True*)

Unpacks the given archive file to the same directory, then (by default) deletes the archive file.

**class** `cogdl.datasets.gtn_data.GTNDataset` (*root, name*)

Bases: `cogdl.data.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").

**property** `raw_file_names` (*self*)

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

**property** `processed_file_names` (*self*)

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

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

**get** (*self, idx*)

Gets the data object at index *idx*.

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

**download** (*self*)

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

**process** (*self*)

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

**\_\_repr\_\_** (*self*)

**class** `cogdl.datasets.gtn_data.ACM_GTNDataset`

Bases: `cogdl.datasets.gtn_data.GTNDataset`

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").

**class** `cogdl.datasets.gtn_data.DBLP_GTNDataset`

Bases: `cogdl.datasets.gtn_data.GTNDataset`

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").

**class** `cogdl.datasets.gtn_data.IMDB_GTNDataset`

Bases: `cogdl.datasets.gtn_data.GTNDataset`

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").

`cogdl.datasets.han_data`

### Module Contents

#### Classes

|                        |  |
|------------------------|--|
| <i>HANDataset</i>      | The network datasets “ACM”, “DBLP” and “IMDB” from the |
| <i>ACM_HANDataset</i>  | The network datasets “ACM”, “DBLP” and “IMDB” from the |
| <i>DBLP_HANDataset</i> | The network datasets “ACM”, “DBLP” and “IMDB” from the |
| <i>IMDB_HANDataset</i> | The network datasets “ACM”, “DBLP” and “IMDB” from the |

#### Functions

|  |   |
|--|---|
| <i>untar</i> (path, fname, deleteTar=True) | Unpacks the given archive file to the same directory, then (by default) |
| <i>sample_mask</i> (idx, l)                | Create mask.  |

`cogdl.datasets.han_data.untar` (path, fname, deleteTar=True)

Unpacks the given archive file to the same directory, then (by default) deletes the archive file.

`cogdl.datasets.han_data.sample_mask` (idx, l)

Create mask.

**class** `cogdl.datasets.han_data.HANDataset` (root, name)

Bases: `cogdl.data.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").

**property** `raw_file_names` (self)

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

**property** `processed_file_names` (self)

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

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

**get** (self, idx)

Gets the data object at index `idx`.

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

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

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

**\_\_repr\_\_** (*self*)

**class** `cogdl.datasets.han_data.ACM_HANDataset`

Bases: `cogdl.datasets.han_data.HANDataset`

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”).

**class** `cogdl.datasets.han_data.DBLP_HANDataset`

Bases: `cogdl.datasets.han_data.HANDataset`

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”).

**class** `cogdl.datasets.han_data.IMDB_HANDataset`

Bases: `cogdl.datasets.han_data.HANDataset`

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”).

`cogdl.datasets.kg_data`

## Module Contents

### Classes

---

*BidirectionalOneShotIterator*

---

*TestDataset*

---

*TrainDataset*

---

*KnowledgeGraphDataset*

Dataset base class for creating graph datasets.

---

*FB13Dataset*

Dataset base class for creating graph datasets.

---

*FB15kDataset*

Dataset base class for creating graph datasets.

---

*FB15k237Dataset*

Dataset base class for creating graph datasets.

---

*WN18Dataset*

Dataset base class for creating graph datasets.

---

*WN18RRDataset*

Dataset base class for creating graph datasets.

---

*FB13SDataset*

Dataset base class for creating graph datasets.

---

## Functions

---

```
read_triplet_data(folder)
```

---

```
class cogdl.datasets.kg_data.BidirectionalOneShotIterator (dataloader_head, dataloader_tail)
```

```
Bases: object
```

```
__next__ (self)
```

```
static one_shot_iterator (dataloader)
```

```
Transform a PyTorch Dataloader into python iterator
```

```
class cogdl.datasets.kg_data.TestDataset (triples, all_true_triples, nentity, nrelation, mode)
```

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

```
__len__ (self)
```

```
__getitem__ (self, idx)
```

```
static collate_fn (data)
```

```
class cogdl.datasets.kg_data.TrainDataset (triples, nentity, nrelation, negative_sample_size, mode)
```

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

```
__len__ (self)
```

```
__getitem__ (self, idx)
```

```
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
```

```
cogdl.datasets.kg_data.read_triplet_data (folder)
```

```
class cogdl.datasets.kg_data.KnowledgeGraphDataset (root, name)
```

```
Bases: cogdl.data.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)
```

```
url = https://raw.githubusercontent.com/thunlp/OpenKE/OpenKE-PyTorch/benchmarks
```

**property raw\_file\_names** (*self*)

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

**property processed\_file\_names** (*self*)

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

**property train\_start\_idx** (*self*)

**property valid\_start\_idx** (*self*)

**property test\_start\_idx** (*self*)

**property num\_entities** (*self*)

**property num\_relations** (*self*)

**get** (*self*, *idx*)

Gets the data object at index `idx`.

**download** (*self*)

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

**process** (*self*)

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

**class** `cogdl.datasets.kg_data.FB13Dataset`

Bases: `cogdl.datasets.kg_data.KnowledgeGraphDataset`

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`)

**class** `cogdl.datasets.kg_data.FB15kDataset`

Bases: `cogdl.datasets.kg_data.KnowledgeGraphDataset`

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`)

**class** `cogdl.datasets.kg_data.FB15k237Dataset`

Bases: `cogdl.datasets.kg_data.KnowledgeGraphDataset`

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`)

**class** `cogdl.datasets.kg_data.WN18Dataset`

Bases: `cogdl.datasets.kg_data.KnowledgeGraphDataset`

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`)

**class** `cogdl.datasets.kg_data.WN18RRDataset`

Bases: `cogdl.datasets.kg_data.KnowledgeGraphDataset`

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`)

**class** `cogdl.datasets.kg_data.FB13SDataset`

Bases: `cogdl.datasets.kg_data.KnowledgeGraphDataset`



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`)

```
url = https://raw.githubusercontent.com/cenyk1230/test-data/main
```

`cogdl.datasets.matlab_matrix`

## Module Contents

### Classes

|                           |  |
|---------------------------|--|
| <i>MatlabMatrix</i>       | networks from the <a href="http://leitang.net/code/social-dimension/data/">http://leitang.net/code/social-dimension/data/</a> or <a href="http://snap.stanford.edu/node2vec/">http://snap.stanford.edu/node2vec/</a> |
| <i>BlogcatalogDataset</i> | networks from the <a href="http://leitang.net/code/social-dimension/data/">http://leitang.net/code/social-dimension/data/</a> or <a href="http://snap.stanford.edu/node2vec/">http://snap.stanford.edu/node2vec/</a> |
| <i>FlickrDataset</i>      | networks from the <a href="http://leitang.net/code/social-dimension/data/">http://leitang.net/code/social-dimension/data/</a> or <a href="http://snap.stanford.edu/node2vec/">http://snap.stanford.edu/node2vec/</a> |
| <i>WikipediaDataset</i>   | networks from the <a href="http://leitang.net/code/social-dimension/data/">http://leitang.net/code/social-dimension/data/</a> or <a href="http://snap.stanford.edu/node2vec/">http://snap.stanford.edu/node2vec/</a> |
| <i>PPIDataset</i>         | networks from the <a href="http://leitang.net/code/social-dimension/data/">http://leitang.net/code/social-dimension/data/</a> or <a href="http://snap.stanford.edu/node2vec/">http://snap.stanford.edu/node2vec/</a> |

**class** `cogdl.datasets.matlab_matrix.MatlabMatrix` (`root`, `name`, `url`)

Bases: `cogdl.data.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").

**property** `raw_file_names` (`self`)

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

**property** `processed_file_names` (`self`)

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

**download** (`self`)

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

**get** (*self*, *idx*)

Gets the data object at index *idx*.

**process** (*self*)

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

**class** `cogdl.datasets.matlab_matrix.BlogcatalogDataset`

Bases: `cogdl.datasets.matlab_matrix.MatlabMatrix`

networks from the <http://leidang.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").

**class** `cogdl.datasets.matlab_matrix.FlickrDataset`

Bases: `cogdl.datasets.matlab_matrix.MatlabMatrix`

networks from the <http://leidang.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").

**class** `cogdl.datasets.matlab_matrix.WikipediaDataset`

Bases: `cogdl.datasets.matlab_matrix.MatlabMatrix`

networks from the <http://leidang.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").

**class** `cogdl.datasets.matlab_matrix.PPIDataset`

Bases: `cogdl.datasets.matlab_matrix.MatlabMatrix`

networks from the <http://leidang.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").

`cogdl.datasets.pyg`

## Module Contents

### Classes

---

*CoraDataset*

---

*CiteSeerDataset*

---

*PubMedDataset*

---

*RedditDataset*

---

*MUTAGDataset*

---

continues on next page

Table 23 – continued from previous page

---

*ImdbBinaryDataset*

---

*ImdbMultiDataset*

---

*CollabDataset*

---

*ProtainsDataset*

---

*RedditBinary*

---

*RedditMulti5K*

---

*RedditMulti12K*

---

*PTCMRDataset*

---

*NCT1Dataset*

---

*NCT109Dataset*

---

*ENZYMES*

---

*QM9Dataset*

---

## Functions

---

*normalize\_feature(data)*

---

`cogdl.datasets.pyg.normalize_feature(data)`

**class** `cogdl.datasets.pyg.CoraDataset`  
Bases: `torch_geometric.datasets.Planetoid`

**class** `cogdl.datasets.pyg.CiteSeerDataset`  
Bases: `torch_geometric.datasets.Planetoid`

**class** `cogdl.datasets.pyg.PubMedDataset`  
Bases: `torch_geometric.datasets.Planetoid`

**class** `cogdl.datasets.pyg.RedditDataset`  
Bases: `torch_geometric.datasets.Reddit`

**class** `cogdl.datasets.pyg.MUTAGDataset`  
Bases: `torch_geometric.datasets.TUDataset`

**class** `cogdl.datasets.pyg.ImdbBinaryDataset`  
Bases: `torch_geometric.datasets.TUDataset`

**class** `cogdl.datasets.pyg.ImdbMultiDataset`  
Bases: `torch_geometric.datasets.TUDataset`

```
class cogdl.datasets.pyg.CollabDataset
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.ProteinsDataset
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.RedditBinary
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.RedditMulti5K
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.RedditMulti12K
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.PTCMRDataset
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.NCT1Dataset
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.NCT109Dataset
    Bases: torch_geometric.datasets.TUDataset

class cogdl.datasets.pyg.ENZYMES
    Bases: torch_geometric.datasets.TUDataset
    __getitem__ (self, idx)

class cogdl.datasets.pyg.QM9Dataset
    Bases: torch_geometric.datasets.QM9
```

`cogdl.datasets.pyg_ogb`

## Module Contents

### Classes

---

*OGBNDataset*

---

*OGBArxivDataset*

---

*OGBProductsDataset*

---

*OGBProductsDataset*

---

*OGBProductsDataset*

---

*OGBPapers100MDataset*

---

*OGBGDataset*

---

*OGBMolbaceDataset*

---

continues on next page

Table 25 – continued from previous page

---

*OGBMolhivDataset*

---

*OGBMolpcbaDataset*

---

*OGBPpaDataset*

---

*OGBCodeDataset*

---

```
class cogdl.datasets.pyg_ogb.OGBNDataset (root, name)
    Bases: ogb.nodeproppred.PygNodePropPredDataset
    get (self, idx)

class cogdl.datasets.pyg_ogb.OGBArxivDataset
    Bases: cogdl.datasets.pyg_ogb.OGBNDataset

class cogdl.datasets.pyg_ogb.OGBProductsDataset
    Bases: cogdl.datasets.pyg_ogb.OGBNDataset

class cogdl.datasets.pyg_ogb.OGBProductsDataset
    Bases: cogdl.datasets.pyg_ogb.OGBNDataset

class cogdl.datasets.pyg_ogb.OGBProductsDataset
    Bases: cogdl.datasets.pyg_ogb.OGBNDataset

class cogdl.datasets.pyg_ogb.OGBPapers100MDataset
    Bases: cogdl.datasets.pyg_ogb.OGBNDataset

class cogdl.datasets.pyg_ogb.OGBGDataset (root, name)
    Bases: ogb.graphproppred.PygGraphPropPredDataset
    get_loader (self, args)
    get (self, idx)

class cogdl.datasets.pyg_ogb.OGBMolbaceDataset
    Bases: cogdl.datasets.pyg_ogb.OGBGDataset

class cogdl.datasets.pyg_ogb.OGBMolhivDataset
    Bases: cogdl.datasets.pyg_ogb.OGBGDataset

class cogdl.datasets.pyg_ogb.OGBMolpcbaDataset
    Bases: cogdl.datasets.pyg_ogb.OGBGDataset

class cogdl.datasets.pyg_ogb.OGBPpaDataset
    Bases: cogdl.datasets.pyg_ogb.OGBGDataset

class cogdl.datasets.pyg_ogb.OGBCodeDataset
    Bases: cogdl.datasets.pyg_ogb.OGBGDataset
```

`cogdl.datasets.pyg_strategies_data`

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

### Module Contents

#### Classes

|   |               |   |
|---|---------------|---|
| <i>NegativeEdge</i>                       | Borrowed from | <a href="https://github.com/snap-stanford/pretrain-gnns/">https://github.com/snap-stanford/pretrain-gnns/</a> |
| <i>MaskEdge</i>                           | Borrowed from | <a href="https://github.com/snap-stanford/pretrain-gnns/">https://github.com/snap-stanford/pretrain-gnns/</a> |
| <i>MaskAtom</i>                           | Borrowed from | <a href="https://github.com/snap-stanford/pretrain-gnns/">https://github.com/snap-stanford/pretrain-gnns/</a> |
| <i>ExtractSubstructureContextPair</i>     |               |   |
| <i>ChemExtractSubstructureContextPair</i> |               |   |
| <i>BatchFinetune</i>                      |               |   |
| <i>BatchMasking</i>                       |               |   |
| <i>BatchAE</i>                            |               |   |
| <i>BatchSubstructContext</i>              |               |   |
| <i>DataLoaderFinetune</i>                 |               |   |
| <i>DataLoaderMasking</i>                  |               |   |
| <i>DataLoaderAE</i>                       |               |   |
| <i>DataLoaderSubstructContext</i>         |               |   |
| <i>TestBioDataset</i>                     |               |   |
| <i>TestChemDataset</i>                    |               |   |
| <i>BioDataset</i>                         |               |   |
| <i>MoleculeDataset</i>                    |               |   |
| <i>BACEDataset</i>                        |               |   |
| <i>BBBPDataset</i>                        |               |   |

## Functions

|  |   |
|--|---|
| <code>nx_to_graph_data_obj(g, center_id, allowable_features_downstream=None, allowable_features_pretrain=None, node_id_to_go_labels=None)</code> |   |
| <code>graph_data_obj_to_nx(data)</code>  |   |
| <code>graph_data_obj_to_nx_simple(data)</code>   | Converts graph Data object required by the pytorch geometric package to |
| <code>nx_to_graph_data_obj_simple(G)</code>  | Converts nx graph to pytorch geometric Data object. Assume node indices |
| <code>reset_idxes(G)</code>  | Resets node indices such that they are numbered from 0 to num_nodes - 1 |

```
cogdl.datasets.pyg_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.pyg_strategies_data.graph_data_obj_to_nx(data)
```

```
cogdl.datasets.pyg_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.pyg_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

```
class cogdl.datasets.pyg_strategies_data.NegativeEdge
```

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

```
__call__(self, data)
```

```
class cogdl.datasets.pyg_strategies_data.MaskEdge(mask_rate)
```

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

```
__call__(self, data, masked_edge_indices=None)
```

```
class cogdl.datasets.pyg_strategies_data.MaskAtom(num_atom_type, num_edge_type, mask_rate, mask_edge=True)
```

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

```
__call__(self, data, masked_atom_indices=None)
```

**Parameters data** – pytorch geometric data object. Assume that the edge

ordering is the default pytorch geometric ordering, where the two directions of a single edge occur in pairs. Eg. data.edge\_index = tensor([[0, 1, 1, 2, 2, 3],

[1, 0, 2, 1, 3, 2]])

**Parameters masked\_atom\_indices** – If None, then randomly samples num\_atoms

- mask rate number of atom indices

Otherwise a list of atom idx that sets the atoms to be masked (for debugging only) :return: None, Creates new attributes in original data object: data.mask\_node\_idx data.mask\_node\_label data.mask\_edge\_idx data.mask\_edge\_label

`__repr__ (self)`  
Return repr(self).

`cogdl.datasets.pyg_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

**class** `cogdl.datasets.pyg_strategies_data.ExtractSubstructureContextPair (ll, center=True)`

`__call__ (self, data, root_idx=None)`

`__repr__ (self)`  
Return repr(self).

**class** `cogdl.datasets.pyg_strategies_data.ChemExtractSubstructureContextPair (k, ll, l2)`

`__call__ (self, data, root_idx=None)`

#### Parameters

- **data** – pytorch geometric data object
- **root\_idx** – If None, then randomly samples an atom idx.

Otherwise sets atom idx of root (for debugging only) :return: None. Creates new attributes in original data object: data.center\_substruct\_idx data.x\_substruct data.edge\_attr\_substruct data.edge\_index\_substruct data.x\_context data.edge\_attr\_context data.edge\_index\_context data.overlap\_context\_substruct\_idx

`__repr__ (self)`  
Return repr(self).

**class** `cogdl.datasets.pyg_strategies_data.BatchFinetune (batch=None, **kwargs)`  
Bases: `torch_geometric.data.Data`

**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.

**property num\_graphs (self)**  
Returns the number of graphs in the batch.

**class** `cogdl.datasets.pyg_strategies_data.BatchMasking (batch=None, **kwargs)`  
Bases: `torch_geometric.data.Data`

**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.

**cumsum (self, 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.

**property num\_graphs** (*self*)

Returns the number of graphs in the batch.

**class** cogdl.datasets.pyg\_strategies\_data.**BatchAE** (*batch=None, \*\*kwargs*)

Bases: torch\_geometric.data.Data

**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.

**property num\_graphs** (*self*)

Returns the number of graphs in the batch.

**cat\_dim** (*self, key*)

**class** cogdl.datasets.pyg\_strategies\_data.**BatchSubstructContext** (*batch=None, \*\*kwargs*)

Bases: torch\_geometric.data.Data

**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.

**cat\_dim** (*self, key*)

**cumsum** (*self, 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.

**property num\_graphs** (*self*)

Returns the number of graphs in the batch.

**class** cogdl.datasets.pyg\_strategies\_data.**DataLoaderFinetune** (*dataset, batch\_size=1, shuffle=True, \*\*kwargs*)

Bases: torch.utils.data.DataLoader

**class** cogdl.datasets.pyg\_strategies\_data.**DataLoaderMasking** (*dataset, batch\_size=1, shuffle=True, \*\*kwargs*)

Bases: torch.utils.data.DataLoader

**class** cogdl.datasets.pyg\_strategies\_data.**DataLoaderAE** (*dataset, batch\_size=1, shuffle=True, \*\*kwargs*)

Bases: torch.utils.data.DataLoader

**class** cogdl.datasets.pyg\_strategies\_data.**DataLoaderSubstructContext** (*dataset, batch\_size=1, shuffle=True, \*\*kwargs*)

Bases: torch.utils.data.DataLoader

```
class cogdl.datasets.pyg_strategies_data.TestBioDataset (data_type='unsupervised',  
                                                    root=None,      trans-  
                                                    form=None,  
                                                    pre_transform=None,  
                                                    pre_filter=None)
```

Bases: torch\_geometric.data.InMemoryDataset

```
class cogdl.datasets.pyg_strategies_data.TestChemDataset (data_type='unsupervised',  
                                                    root=None,      trans-  
                                                    form=None,  
                                                    pre_transform=None,  
                                                    pre_filter=None)
```

Bases: torch\_geometric.data.InMemoryDataset

```
get (self, idx)
```

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

Bases: torch\_geometric.data.InMemoryDataset

```
property raw_file_names (self)
```

```
property processed_file_names (self)
```

```
download (self)
```

```
process (self)
```

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

Bases: torch\_geometric.data.InMemoryDataset

```
get (self, idx)
```

```
property raw_file_names (self)
```

```
property processed_file_names (self)
```

```
download (self)
```

```
process (self)
```

```
class cogdl.datasets.pyg_strategies_data.BACEDataset (transform=None,  
                                                    pre_transform=None,  
                                                    pre_filter=None,  
                                                    empty=False)
```

Bases: torch\_geometric.data.InMemoryDataset

```
get (self, idx)
```

```
property raw_file_names (self)
```

```
property processed_file_names (self)
```

```
download (self)
```

```
process (self)
```

```

class cogdl.datasets.pyg_strategies_data.BBBPDataset (transform=None,
                                                    pre_transform=None,
                                                    pre_filter=None,
                                                    empty=False)

Bases: torch_geometric.data.InMemoryDataset

get (self, idx)

property raw_file_names (self)

property processed_file_names (self)

download (self)

process (self)

```

## Package Contents

### Classes

---

|                |   |
|----------------|---|
| <i>Dataset</i> | Dataset base class for creating graph datasets. |
|----------------|---|

---

### Functions

---

|  |  |
|--|--|
| <i>register_dataset</i> (name)           | New dataset types can be added to cogdl with the <i>register_dataset()</i> |
| <i>build_dataset</i> (args)              |  |
| <i>build_dataset_from_name</i> (dataset) |  |

---

```

class cogdl.datasets.Dataset (root, transform=None, pre_transform=None, pre_filter=None)
Bases: torch.utils.data.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*)

```
property raw_file_names (self)
```

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

```
property processed_file_names (self)
```

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

**abstract download** (*self*)

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

**abstract process** (*self*)

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

**abstract \_\_len\_\_** (*self*)

The number of examples in the dataset.

**abstract get** (*self, idx*)

Gets the data object at index `idx`.

**property num\_features** (*self*)

Returns the number of features per node in the graph.

**property raw\_paths** (*self*)

The filepaths to find in order to skip the download.

**property processed\_paths** (*self*)

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

**\_download** (*self*)

**\_process** (*self*)

**\_\_getitem\_\_** (*self, idx*)

Gets the data object at index `idx` and transforms it (in case a `self.transform` is given).

**\_\_repr\_\_** (*self*)

`cogdl.datasets.pyg = False`

`cogdl.datasets.dgl_import = False`

`cogdl.datasets.DATASET_REGISTRY`

`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.dataset_name`

`cogdl.datasets.build_dataset` (*args*)

`cogdl.datasets.build_dataset_from_name` (*dataset*)

`cogdl.layers`**Submodules**`cogdl.layers.gcc_module`**Module Contents****Classes**


---

|                         |   |
|-------------------------|---|
| <i>GATLayer</i>         |   |
| <i>SELayer</i>          | Squeeze-and-excitation networks                   |
| <i>ApplyNodeFunc</i>    | Update the node feature hv with MLP, BN and ReLU. |
| <i>MLP</i>              | MLP with linear output                            |
| <i>UnsupervisedGAT</i>  |   |
| <i>UnsupervisedMPNN</i> | MPNN from   |
| <i>UnsupervisedGIN</i>  | GIN model   |
| <i>GraphEncoder</i>     | MPNN from   |

---

**class** `cogdl.layers.gcc_module.GATLayer` (*g*, *in\_dim*, *out\_dim*)

Bases: `torch.nn.Module`

**edge\_attention** (*self*, *edges*)

**message\_func** (*self*, *edges*)

**reduce\_func** (*self*, *nodes*)

**forward** (*self*, *h*)

**class** `cogdl.layers.gcc_module.SELayer` (*in\_channels*, *se\_channels*)

Bases: `torch.nn.Module`

Squeeze-and-excitation networks

**forward** (*self*, *x*)

**class** `cogdl.layers.gcc_module.ApplyNodeFunc` (*mlp*, *use\_selayer*)

Bases: `torch.nn.Module`

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

**forward** (*self*, *h*)

**class** `cogdl.layers.gcc_module.MLP` (*num\_layers*, *input\_dim*, *hidden\_dim*, *output\_dim*,  
*use\_selayer*)

Bases: `torch.nn.Module`

MLP with linear output

**forward** (*self*, *x*)

**class** `cogdl.layers.gcc_module.UnsupervisedGAT` (*node\_input\_dim*, *node\_hidden\_dim*,  
*edge\_input\_dim*, *num\_layers*,  
*num\_heads*)

Bases: `torch.nn.Module`

**forward** (*self*, *g*, *n\_feat*, *e\_feat*)

```
class cogdl.layers.gcc_module.UnsupervisedMPNN (output_dim=32, node_input_dim=32,
node_hidden_dim=32,
edge_input_dim=32,
edge_hidden_dim=32,
num_step_message_passing=6,
lstm_as_gate=False)
```

Bases: torch.nn.Module

MPNN from [Neural Message Passing for Quantum Chemistry](#)

**node\_input\_dim** [int] Dimension of input node feature, default to be 15.

**edge\_input\_dim** [int] Dimension of input edge feature, default to be 15.

**output\_dim** [int] Dimension of prediction, default to be 12.

**node\_hidden\_dim** [int] Dimension of node feature in hidden layers, default to be 64.

**edge\_hidden\_dim** [int] Dimension of edge feature in hidden layers, default to be 128.

**num\_step\_message\_passing** [int] Number of message passing steps, default to be 6.

**num\_step\_set2set** [int] Number of set2set steps

**num\_layer\_set2set** [int] Number of set2set layers

**forward** (*self*, *g*, *n\_feat*, *e\_feat*)

Predict molecule labels

**g** [DGLGraph] Input DGLGraph for molecule(s)

**n\_feat** [tensor of dtype float32 and shape (B1, D1)] Node features. B1 for number of nodes and D1 for the node feature size.

**e\_feat** [tensor of dtype float32 and shape (B2, D2)] Edge features. B2 for number of edges and D2 for the edge feature size.

res : Predicted labels

```
class cogdl.layers.gcc_module.UnsupervisedGIN (num_layers, num_mlp_layers, input_dim,
hidden_dim, output_dim, final_dropout,
learn_eps, graph_pooling_type, neighbor_pooling_type, use_selayer)
```

Bases: torch.nn.Module

GIN model

**forward** (*self*, *g*, *h*, *e\_feat*)

```
class cogdl.layers.gcc_module.GraphEncoder (positional_embedding_size=32,
max_node_freq=8, max_edge_freq=8,
max_degree=128, freq_embedding_size=32,
degree_embedding_size=32, output_dim=32,
node_hidden_dim=32,
edge_hidden_dim=32, num_layers=6,
num_heads=4, num_step_set2set=6,
num_layer_set2set=3, norm=False,
gnn_model='mpnn', degree_input=False,
lstm_as_gate=False)
```

Bases: torch.nn.Module

MPNN from [Neural Message Passing for Quantum Chemistry](#)

**node\_input\_dim** [int] Dimension of input node feature, default to be 15.

**edge\_input\_dim** [int] Dimension of input edge feature, default to be 15.

**output\_dim** [int] Dimension of prediction, default to be 12.

**node\_hidden\_dim** [int] Dimension of node feature in hidden layers, default to be 64.

**edge\_hidden\_dim** [int] Dimension of edge feature in hidden layers, default to be 128.

**num\_step\_message\_passing** [int] Number of message passing steps, default to be 6.

**num\_step\_set2set** [int] Number of set2set steps

**num\_layer\_set2set** [int] Number of set2set layers

**forward** (*self*, *g*, *return\_all\_outputs=False*)

Predict molecule labels

**g** [DGLGraph] Input DGLGraph for molecule(s)

**n\_feat** [tensor of dtype float32 and shape (B1, D1)] Node features. B1 for number of nodes and D1 for the node feature size.

**e\_feat** [tensor of dtype float32 and shape (B2, D2)] Edge features. B2 for number of edges and D2 for the edge feature size.

res : Predicted labels

`cogdl.layers.gpt_gnn_module`

## Module Contents

### Classes

|                            |  |
|----------------------------|--|
| <i>Graph</i>               |  |
| <i>HGTConv</i>             |  |
| <i>RelTemporalEncoding</i> | Implement the Temporal Encoding (Sinusoid) function.                 |
| <i>GeneralConv</i>         |  |
| <i>GNN</i>                 |  |
| <i>GPT_GNN</i>             |  |
| <i>Classifier</i>          |  |
| <i>Matcher</i>             | Matching between a pair of nodes to conduct link prediction.         |
| <i>RNNModel</i>            | Container module with an encoder, a recurrent module, and a decoder. |

## Functions

|   |  |
|---|--|
| <code>args_print(args)</code>   |  |
| <code>dcg_at_k(r, k)</code>   |  |
| <code>ndcg_at_k(r, k)</code>  |  |
| <code>mean_reciprocal_rank(rs)</code>   |  |
| <code>normalize(mx)</code>  | Row-normalize sparse matrix  |
| <code>sparse_mx_to_torch_sparse_tensor(sparse_mx)</code>  | Convert a scipy sparse matrix to a torch sparse tensor.                              |
| <code>randint()</code>  |  |
| <code>feature_OAG(layer_data, graph)</code>   |  |
| <code>feature_reddit(layer_data, graph)</code>  |  |
| <code>load_gnn(_dict)</code>  |  |
| <code>defaultDictDict()</code>  |  |
| <code>defaultDictList()</code>  |  |
| <code>defaultDictInt()</code>   |  |
| <code>defaultDictDictInt()</code>   |  |
| <code>defaultDictDictDictInt()</code>   |  |
| <code>defaultDictDictDictDictInt()</code>   |  |
| <code>defaultDictDictDictDictDictInt()</code>   |  |
| <code>sample_subgraph(graph, time_range, sampled_depth=2, sampled_number=8, inp=None, feature_extractor=feature_OAG)</code> | Sample Sub-Graph based on the connection of other nodes with currently sampled nodes |
| <code>to_torch(feature, time, edge_list, graph)</code>  | Transform a sampled sub-graph into pytorch Tensor                                    |
| <code>preprocess_dataset(dataset)</code>  | →  |
| <code>cogdl.layers.gpt_gnn_module.Graph</code>  |  |

`cogdl.layers.gpt_gnn_module.args_print(args)`

`cogdl.layers.gpt_gnn_module.dcg_at_k(r, k)`

`cogdl.layers.gpt_gnn_module.ndcg_at_k(r, k)`

`cogdl.layers.gpt_gnn_module.mean_reciprocal_rank(rs)`

`cogdl.layers.gpt_gnn_module.normalize(mx)`

Row-normalize sparse matrix

`cogdl.layers.gpt_gnn_module.sparse_mx_to_torch_sparse_tensor(sparse_mx)`

Convert a scipy sparse matrix to a torch sparse tensor.



```

cogdl.layers.gpt_gnn_module.randint ()
cogdl.layers.gpt_gnn_module.feature_OAG (layer_data, graph)
cogdl.layers.gpt_gnn_module.feature_reddit (layer_data, graph)
cogdl.layers.gpt_gnn_module.load_gnn (_dict)
cogdl.layers.gpt_gnn_module.defaultDictDict ()
cogdl.layers.gpt_gnn_module.defaultDictList ()
cogdl.layers.gpt_gnn_module.defaultDictInt ()
cogdl.layers.gpt_gnn_module.defaultDictDictInt ()
cogdl.layers.gpt_gnn_module.defaultDictDictDictInt ()
cogdl.layers.gpt_gnn_module.defaultDictDictDictDictInt ()
cogdl.layers.gpt_gnn_module.defaultDictDictDictDictDictInt ()
class cogdl.layers.gpt_gnn_module.Graph

    node_feature
        edge_list: index the adjacency matrix (time) by <target_type, source_type, relation_type, target_id,
        source_id>

    add_node (self, node)

    add_edge (self, source_node, target_node, time=None, relation_type=None, directed=True)

    update_node (self, node)

    get_meta_graph (self)

    get_types (self)

cogdl.layers.gpt_gnn_module.sample_subgraph (graph, time_range, sampled_depth=2,
                                             sampled_number=8, inp=None, feature_extractor=feature_OAG)

    Sample Sub-Graph based on the connection of other nodes with currently sampled nodes We maintain budgets
    for each node type, indexed by <node_id, time>. Currently sampled nodes are stored in layer_data. After nodes
    are sampled, we construct the sampled adjacency matrix.

cogdl.layers.gpt_gnn_module.to_torch (feature, time, edge_list, graph)
    Transform a sampled sub-graph into pytorch Tensor node_dict: {node_type: <node_number, node_type_ID>}
    node_number is used to trace back the nodes in original graph. edge_dict: {edge_type: edge_type_ID}

class cogdl.layers.gpt_gnn_module.HGTConv (in_dim, out_dim, num_types, num_relations,
                                             n_heads, dropout=0.2, use_norm=True,
                                             use_RTE=True, **kwargs)
    Bases: torch_geometric.nn.conv.MessagePassing

    forward (self, node_inp, node_type, edge_index, edge_type, edge_time)

    message (self, edge_index_i, node_inp_i, node_inp_j, node_type_i, node_type_j, edge_type, edge_time)
        j: source, i: target; <j, i>

    update (self, aggr_out, node_inp, node_type)
        Step 3: Target-specific Aggregation  $x = W[\text{node\_type}] * \text{gelu}(\text{Agg}(x)) + x$ 

    __repr__ (self)

```

---

```

class cogdl.layers.gpt_gnn_module.RelTemporalEncoding (n_hid, max_len=240,
                                                    dropout=0.2)
    Bases: torch.nn.Module
    Implement the Temporal Encoding (Sinusoid) function.
    forward (self, x, t)

class cogdl.layers.gpt_gnn_module.GeneralConv (conv_name, in_hid, out_hid, num_types,
                                                num_relations, n_heads, dropout,
                                                use_norm=True, use_RTE=True)
    Bases: torch.nn.Module
    forward (self, meta_xs, node_type, edge_index, edge_type, edge_time)

class cogdl.layers.gpt_gnn_module.GNN (in_dim, n_hid, num_types, num_relations,
                                        n_heads, n_layers, dropout=0.2, conv_name='hgt',
                                        prev_norm=False, last_norm=False, use_RTE=True)
    Bases: torch.nn.Module
    forward (self, node_feature, node_type, edge_time, edge_index, edge_type)

class cogdl.layers.gpt_gnn_module.GPT_GNN (gnn, rem_edge_list, attr_decoder, types,
                                            neg_samp_num, device, neg_queue_size=0)
    Bases: torch.nn.Module
    neg_sample (self, source_node_list, pos_node_list)
    forward (self, node_feature, node_type, edge_time, edge_index, edge_type)
    link_loss (self, node_emb, rem_edge_list, ori_edge_list, node_dict, target_type, use_queue=True, update_queue=False)
    text_loss (self, reps, texts, w2v_model, device)
    feat_loss (self, reps, out)

class cogdl.layers.gpt_gnn_module.Classifier (n_hid, n_out)
    Bases: torch.nn.Module
    forward (self, x)
    __repr__ (self)

class cogdl.layers.gpt_gnn_module.Matcher (n_hid, n_out, temperature=0.1)
    Bases: torch.nn.Module
    Matching between a pair of nodes to conduct link prediction. Use multi-head attention as matching model.
    forward (self, x, ty, use_norm=True)
    __repr__ (self)

class cogdl.layers.gpt_gnn_module.RNNModel (n_word, ninp, nhid, nlayers, dropout=0.2)
    Bases: torch.nn.Module
    Container module with an encoder, a recurrent module, and a decoder.
    forward (self, inp, hidden=None)
    from_w2v (self, w2v)

cogdl.layers.gpt_gnn_module.preprocess_dataset (dataset) →
                                                    cogdl.layers.gpt_gnn_module.Graph

```

`cogdl.layers.link_prediction_module`

## Module Contents

## Classes

---

`DistMultLayer`

---

---

`ConvELayer`

---

---

`GNNLinkPredict`

---

## Functions

---

`cal_mrr(embedding, rel_embedding, edge_index, edge_type, scoring, protocol='raw', batch_size=1000, hits=[])`

---

---

`sampling_edge_uniform(edge_index, edge_types, edge_set, sampling_rate, num_rels, label_smoothing=0.0, num_entities=1)` Args:

---

---

`get_rank(scores, target)`

---

---

`get_raw_rank(heads, tails, rels, embedding, rel_embedding, batch_size, scoring)`

---

---

`get_filtered_rank(heads, tails, rels, embedding, rel_embedding, batch_size, seen_data)`

---

---

`cogdl.layers.link_prediction_module.cal_mrr(embedding, rel_embedding, edge_index, edge_type, scoring, protocol='raw', batch_size=1000, hits=[])`

---

**class** `cogdl.layers.link_prediction_module.DistMultLayer`Bases: `torch.nn.Module`**forward** (`self, sub_emb, obj_emb, rel_emb`)**predict** (`self, sub_emb, obj_emb, rel_emb`)**class** `cogdl.layers.link_prediction_module.ConvELayer` (`dim, num_filter=20, kernel_size=7, k_w=10, dropout=0.3`)Bases: `torch.nn.Module`**concat** (`self, ent, rel`)**forward** (`self, sub_emb, obj_emb, rel_emb`)**predict** (`self, sub_emb, obj_emb, rel_emb`)**class** `cogdl.layers.link_prediction_module.GNNLinkPredict` (`score_func, dim`)Bases: `torch.nn.Module`**forward** (`self, edge_index, edge_type`)

`get_score` (*self*, *heads*, *tails*, *rels*)

`get_edge_set` (*self*, *edge\_index*, *edge\_types*)

`_loss` (*self*, *head\_embed*, *tail\_embed*, *rel\_embed*, *labels*)

`_regularization` (*self*, *embs*)

`cogdl.layers.link_prediction_module.sampling_edge_uniform` (*edge\_index*, *edge\_types*,  
*edge\_set*, *sampling\_rate*, *num\_rels*,  
*label\_smoothing=0.0*,  
*num\_entities=1*)

**Args:** *edge\_index*: edge index of graph *edge\_types*: *edge\_set*: set of all edges of the graph, (h, t, r) *sampling\_rate*: *num\_rels*: *label\_smoothing*(Optional): *num\_entities* (Optional):

**Returns:** *sampled\_edges*: sampled existing edges *rels*: types of sampled existing edges *sampled\_edges\_all*: existing edges with corrupted edges *sampled\_types\_all*: types of existing and corrupted edges *labels*: 0/1

`cogdl.layers.link_prediction_module.get_rank` (*scores*, *target*)

`cogdl.layers.link_prediction_module.get_raw_rank` (*heads*, *tails*, *rels*, *embedding*,  
*rel\_embedding*, *batch\_size*, *scoring*)

`cogdl.layers.link_prediction_module.get_filtered_rank` (*heads*, *tails*, *rels*, *embedding*,  
*rel\_embedding*, *batch\_size*,  
*seen\_data*)

`cogdl.layers.maggregator`

## Module Contents

### Classes

---

*MeanAggregator*

---

**class** `cogdl.layers.maggregator.MeanAggregator` (*in\_channels*, *out\_channels*, *improved=False*, *cached=False*, *bias=True*)

Bases: `torch.nn.Module`

**static norm** (*x*, *edge\_index*)

**forward** (*self*, *x*, *edge\_index*, *edge\_weight=None*, *bias=True*)

**update** (*self*, *aggr\_out*)

**\_\_repr\_\_** (*self*)

---

`cogdl.layers.mixhop_layer`

## Module Contents

### Classes

---

*MixHopLayer*

---

**class** `cogdl.layers.mixhop_layer.MixHopLayer` (*num\_features, adj\_pows, dim\_per\_pow*)

Bases: `torch.nn.Module`

**reset\_parameters** (*self*)

**adj\_pow\_x** (*self, x, adj, p*)

**forward** (*self, x, edge\_index*)

`cogdl.layers.mixhop_layer.layer`

`cogdl.layers.prone_module`

## Module Contents

### Classes

---

*HeatKernel*

---



---

*HeatKernelApproximation*

---



---

*Gaussian*

---



---

*PPR*

applying sparsification to accelerate computation

---

*SignalRescaling*

- rescale signal of each node according to the degree of the node:
- 

---

*ProNE*

---



---

*NodeAdaptiveEncoder*

- shrink negative values in signal/feature matrix
-

### Functions

---

`propagate(mx, emb, stype, space=None)`

---

`get_embedding_dense(matrix, dimension)`

---

**class** `cogdl.layers.prone_module.HeatKernel` (*t=0.5, theta0=0.6, theta1=0.4*)

Bases: `object`

**prop\_adjacency** (*self, mx*)

**prop** (*self, mx, emb*)

**class** `cogdl.layers.prone_module.HeatKernelApproximation` (*t=0.2, k=5*)

Bases: `object`

**taylor** (*self, mx, emb*)

**chebyshev** (*self, mx, emb*)

**prop** (*self, mx, emb*)

**class** `cogdl.layers.prone_module.Gaussian` (*mu=0.5, theta=1, rescale=False, k=3*)

Bases: `object`

**prop** (*self, mx, emb*)

**class** `cogdl.layers.prone_module.PPR` (*alpha=0.5, k=10*)

Bases: `object`

applying sparsification to accelerate computation

**prop** (*self, mx, emb*)

**class** `cogdl.layers.prone_module.SignalRescaling`

Bases: `object`

- rescale signal of each node according to the degree of the node:
- `sigmoid(degree)`
- `sigmoid(1/degree)`

**prop** (*self, mx, emb*)

**class** `cogdl.layers.prone_module.ProNE`

Bases: `object`

**\_\_call\_\_** (*self, A, a, order=10, mu=0.1, s=0.5*)

**class** `cogdl.layers.prone_module.NodeAdaptiveEncoder`

Bases: `object`

- shrink negative values in signal/feature matrix
- no learning

**static prop** (*signal*)

`cogdl.layers.prone_module.propagate` (*mx, emb, stype, space=None*)

`cogdl.layers.prone_module.get_embedding_dense` (*matrix, dimension*)

---

`cogdl.layers.se_layer`

## Module Contents

### Classes

---

|                |                                 |
|----------------|---------------------------------|
| <i>SELayer</i> | Squeeze-and-excitation networks |
|----------------|---------------------------------|

---

**class** `cogdl.layers.se_layer.SELayer` (*in\_channels*, *se\_channels*)

Bases: `torch.nn.Module`

Squeeze-and-excitation networks

**forward** (*self*, *x*)

`cogdl.layers.srgcn_module`

## Module Contents

### Classes

---

*NodeAttention*

---

*EdgeAttention*

---

*Identity*

---

*PPR*

---

*HeatKernel*

---

*NormIdentity*

---

*RowUniform*

---

*RowSoftmax*

---

*ColumnUniform*

---

*SymmetryNorm*

---

### Functions

---

`act_attention(attn_type)`

---

`act_normalization(norm_type)`

---

`act_map(act)`

---

**class** cogdl.layers.srgcn\_module.**NodeAttention** (*in\_feat*)

Bases: torch.nn.Module

**forward** (*self*, *x*, *edge\_index*, *edge\_attr*)

**class** cogdl.layers.srgcn\_module.**EdgeAttention** (*in\_feat*)

Bases: torch.nn.Module

**forward** (*self*, *x*, *edge\_index*, *edge\_attr*)

**class** cogdl.layers.srgcn\_module.**Identity** (*in\_feat*)

Bases: torch.nn.Module

**forward** (*self*, *x*, *edge\_index*, *edge\_attr*)

**class** cogdl.layers.srgcn\_module.**PPR** (*in\_feat*)

Bases: torch.nn.Module

**forward** (*self*, *x*, *edge\_index*, *edge\_attr*)

**class** cogdl.layers.srgcn\_module.**HeatKernel** (*in\_feat*)

Bases: torch.nn.Module

**forward** (*self*, *x*, *edge\_index*, *edge\_attr*)

cogdl.layers.srgcn\_module.**act\_attention** (*attn\_type*)

**class** cogdl.layers.srgcn\_module.**NormIdentity**

Bases: torch.nn.Module

**forward** (*self*, *edge\_index*, *edge\_attr*, *N*)

**class** cogdl.layers.srgcn\_module.**RowUniform**

Bases: torch.nn.Module

**forward** (*self*, *edge\_index*, *edge\_attr*, *N*)

**class** cogdl.layers.srgcn\_module.**RowSoftmax**

Bases: torch.nn.Module

**forward** (*self*, *edge\_index*, *edge\_attr*, *N*)

**class** cogdl.layers.srgcn\_module.**ColumnUniform**

Bases: torch.nn.Module

**forward** (*self*, *edge\_index*, *edge\_attr*, *N*)

**class** cogdl.layers.srgcn\_module.**SymmetryNorm**

Bases: torch.nn.Module

**forward** (*self*, *edge\_index*, *edge\_attr*, *N*)

cogdl.layers.srgcn\_module.**act\_normalization** (*norm\_type*)



`cogdl.layers.srgcn_module.act_map` (*act*)

**`cogdl.layers.strategies_layers`**

## Module Contents

### Classes

---

*GINConv*

---

*GNN*

---

*GNNPred*

---

*Pretrainer*

---

*Discriminator*

---

*InfoMaxTrainer*

---

*ContextPredictTrainer*

---

*MaskTrainer*

---

*SupervisedTrainer*

---

*Finetuner*

---

**class** `cogdl.layers.strategies_layers.GINConv` (*hidden\_size*, *input\_layer=None*,  
*edge\_emb=None*, *edge\_encode=None*,  
*pooling='sum'*, *feature\_concat=False*)

Bases: `torch.nn.Module`

**forward** (*self*, *x*, *edge\_index*, *edge\_attr*, *self\_loop\_index=None*, *self\_loop\_type=None*)

**aggr** (*self*, *x*, *edge\_index*, *num\_nodes*)

**class** `cogdl.layers.strategies_layers.GNN` (*num\_layers*, *hidden\_size*, *JK='last'*, *dropout=0.5*,  
*input\_layer=None*, *edge\_encode=None*,  
*edge\_emb=None*, *num\_atom\_type=None*,  
*num\_chirality\_tag=None*, *concat=False*)

Bases: `torch.nn.Module`

**forward** (*self*, *x*, *edge\_index*, *edge\_attr*, *self\_loop\_index=None*, *self\_loop\_type=None*)

**class** `cogdl.layers.strategies_layers.GNNPred` (*num\_layers*, *hidden\_size*,  
*num\_tasks*, *JK='last'*, *dropout=0*,  
*graph\_pooling='mean'*, *in-*  
*put\_layer=None*, *edge\_encode=None*,  
*edge\_emb=None*, *num\_atom\_type=None*,  
*num\_chirality\_tag=None*, *concat=True*)

Bases: `torch.nn.Module`

```
    load_from_pretrained (self, path)
    forward (self, data, self_loop_index, self_loop_type)
    pool (self, x, batch)
class cogdl.layers.strategies_layers.Pretrainer (args, transform=None)
    Bases: torch.nn.Module
    get_dataset (self, dataset_name, transform=None)
    fit (self)
class cogdl.layers.strategies_layers.Discriminator (hidden_size)
    Bases: torch.nn.Module
    reset_parameters (self)
    forward (self, x, summary)
class cogdl.layers.strategies_layers.InfoMaxTrainer (args)
    Bases: cogdl.layers.strategies_layers.Pretrainer
    static add_args (parser)
    _train_step (self)
class cogdl.layers.strategies_layers.ContextPredictTrainer (args)
    Bases: cogdl.layers.strategies_layers.Pretrainer
    static add_args (parser)
    _train_step (self)
    get_cbow_pred (self, overlapped_rep, overlapped_context, neighbor_rep)
    get_skipgram_pred (self, overlapped_rep, overlapped_context_size, neighbor_rep)
class cogdl.layers.strategies_layers.MaskTrainer (args)
    Bases: cogdl.layers.strategies_layers.Pretrainer
    static add_args (parser)
    _train_step (self)
class cogdl.layers.strategies_layers.SupervisedTrainer (args)
    Bases: cogdl.layers.strategies_layers.Pretrainer
    static add_args (parser)
    split_data (self)
    _train_step (self)
class cogdl.layers.strategies_layers.Finetuner (args)
    Bases: cogdl.layers.strategies_layers.Pretrainer
    static add_args (parser)
    build_model (self, args)
    split_data (self)
    _train_step (self)
    _test_step (self, split='val')
    fit (self)
```

## Package Contents

### Classes

---

*MeanAggregator*

---

*SELayer*

Squeeze-and-excitation networks

---

*MixHopLayer*

---

**class** cogdl.layers.**MeanAggregator** (*in\_channels, out\_channels, improved=False, cached=False, bias=True*)

Bases: torch.nn.Module

**static norm** (*x, edge\_index*)

**forward** (*self, x, edge\_index, edge\_weight=None, bias=True*)

**update** (*self, aggr\_out*)

**\_\_repr\_\_** (*self*)

**class** cogdl.layers.**SELayer** (*in\_channels, se\_channels*)

Bases: torch.nn.Module

Squeeze-and-excitation networks

**forward** (*self, x*)

**class** cogdl.layers.**MixHopLayer** (*num\_features, adj\_pows, dim\_per\_pow*)

Bases: torch.nn.Module

**reset\_parameters** (*self*)

**adj\_pow\_x** (*self, x, adj, p*)

**forward** (*self, x, edge\_index*)

cogdl.models

### Subpackages

cogdl.models.emb

### Submodules

cogdl.models.emb.complex

### Module Contents

#### Classes

---

*Complex*

the implementation of ComplEx model from the paper “Complex Embeddings for Simple Link Prediction” <<http://proceedings.mlr.press/v48/trouillon16.pdf>>

---

```
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>>

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

**cogdl.models.emb.deepwalk**

## Module Contents

### Classes

---

*DeepWalk*

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

---

```
class cogdl.models.emb.deepwalk.DeepWalk (dimension, walk_length, walk_num, window_size,
                                           worker, iteration)
```

Bases: *cogdl.models.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*)

Add model-specific arguments to the parser.

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

Build a new model instance.

**train** (*self*, *G*)

**\_walk** (*self*, *start\_node*, *walk\_length*)

**\_simulate\_walks** (*self*, *walk\_length*, *num\_walks*)

`cogdl.models.emb.dgk`

## Module Contents

### Classes

---

|                              |   |
|------------------------------|---|
| <code>DeepGraphKernel</code> | The Hin2vec model from the “Deep Graph Kernels” |
|------------------------------|---|

---

```
class cogdl.models.emb.dgk.DeepGraphKernel (hidden_dim, min_count, window_size,
                                             sampling_rate, rounds, epoch, alpha,
                                             n_workers=4)
```

Bases: `cogdl.models.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** (*cls, args*)  
Build a new model instance.

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

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

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

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

`cogdl.models.emb.distmult`

## Module Contents

### Classes

---

|                       |  |
|-----------------------|--|
| <code>DistMult</code> | The DistMult model from the ICLR 2015 paper “EMBEDDING ENTITIES AND RELATIONS FOR LEARNING AND INFERENCE IN KNOWLEDGE BASES” |
|-----------------------|--|

---

```
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>>

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

`cogdl.models.emb.dngr`

## Module Contents

### Classes

---

*DNGR\_layer*

---

*DNGR*

---

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

---

**class** `cogdl.models.emb.dngr.DNGR_layer` (*num\_node, hidden\_size1, hidden\_size2*)

Bases: `torch.nn.Module`

**forward** (*self, x*)

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

Bases: `cogdl.models.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** (*cls, args*)

Build a new model instance.

**scale\_matrix** (*self, mat*)

**random\_surfing** (*self, adj\_matrix*)

**get\_ppmi\_matrix** (*self, mat*)

**get\_denoised\_matrix** (*self, mat*)

**get\_emb** (*self, matrix*)

**train** (*self, G*)

`cogdl.models.emb.gatne`

## Module Contents

### Classes

---

*GATNE*

---

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

---

continues on next page

Table 49 – continued from previous page

---

*GATNEModel*

---

*NSLoss*

---

*RWGraph*

---

## Functions

---

*get\_G\_from\_edges*(edges)

---

*generate\_pairs*(all\_walks, vocab, window\_size=5)

---

*generate\_vocab*(all\_walks)

---

*get\_batches*(pairs, neighbors, batch\_size)

---

*generate\_walks*(network\_data, num\_walks, walk\_length, schema=None)

---

**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.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** (*cls, args*)

Build a new model instance.

**train** (*self, network\_data*)

**class** cogdl.models.emb.gatne.**GATNEModel** (*num\_nodes, embedding\_size, embedding\_u\_size, edge\_type\_count, dim\_a*)

Bases: torch.nn.Module

**reset\_parameters** (*self*)

**forward** (*self, train\_inputs, train\_types, node\_neigh*)

**class** cogdl.models.emb.gatne.**NSLoss** (*num\_nodes, num\_sampled, embedding\_size*)

Bases: torch.nn.Module

**reset\_parameters** (*self*)

**forward** (*self*, *input*, *embs*, *label*)

**class** cogdl.models.emb.gatne.RWGraph (*nx\_G*, *node\_type=None*)

**walk** (*self*, *walk\_length*, *start*, *schema=None*)

**simulate\_walks** (*self*, *num\_walks*, *walk\_length*, *schema=None*)

cogdl.models.emb.gatne.**get\_G\_from\_edges** (*edges*)

cogdl.models.emb.gatne.**generate\_pairs** (*all\_walks*, *vocab*, *window\_size=5*)

cogdl.models.emb.gatne.**generate\_vocab** (*all\_walks*)

cogdl.models.emb.gatne.**get\_batches** (*pairs*, *neighbors*, *batch\_size*)

cogdl.models.emb.gatne.**generate\_walks** (*network\_data*, *num\_walks*, *walk\_length*,  
*schema=None*)

**cogdl.models.emb.graph2vec**

## Module Contents

### Classes

---

*Graph2Vec*

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

---

**class** cogdl.models.emb.graph2vec.Graph2Vec (*dimension*, *min\_count*, *window\_size*, *dm*, *sampling\_rate*, *rounds*, *epoch*, *lr*, *worker=4*)

Bases: *cogdl.models.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** (*cls*, *args*)

Build a new model instance.

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

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

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

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



`cogdl.models.emb.grarep`

## Module Contents

### Classes

---

|                     |   |
|---------------------|---|
| <code>GraRep</code> | The GraRep model from the “GraRep: Learning graph representations with global structural information” |
|---------------------|---|

---

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

Bases: `cogdl.models.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** (*cls, args*)

Build a new model instance.

**train** (*self, G*)

**\_get\_embedding** (*self, matrix, dimension*)

`cogdl.models.emb.hin2vec`

## Module Contents

### Classes

---

|                            |  |
|----------------------------|--|
| <code>Hin2vec_layer</code> |  |
| <code>RWgraph</code>       |  |
| <code>Hin2vec</code>       | The Hin2vec model from the “HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning” |

---

**class** `cogdl.models.emb.hin2vec.Hin2vec_layer` (*num\_node, num\_relation, hidden\_size, cpu*)

Bases: `torch.nn.Module`

**regular\_tion** (*self, embr*)

**forward** (*self, x, y, r, l*)

**get\_emb** (*self*)

**class** `cogdl.models.emb.hin2vec.RWgraph` (*nx\_G, node\_type=None*)

**\_walk** (*self, start\_node, walk\_length*)

`_simulate_walks` (*self*, *walk\_length*, *num\_walks*)

`data_preparation` (*self*, *walks*, *hop*, *negative*)

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

Bases: `cogdl.models.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 negative 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** (*cls*, *args*)  
Build a new model instance.

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

`cogdl.models.emb.hope`

## Module Contents

### Classes

---

`HOPE`

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

---

**class** `cogdl.models.emb.hope.HOPE` (*dimension*, *beta*)

Bases: `cogdl.models.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** (*cls*, *args*)  
Build a new model instance.

**train** (*self*, *G*)

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

**\_get\_embedding** (*self*, *matrix*, *dimension*)

`cogdl.models.emb.knowledge_base`

## Module Contents

### Classes

---

*KGEModel*

---

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

Bases: `cogdl.models.BaseModel`

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

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

**forward** (*self, sample, mode='single'*)  
Forward function that calculate the score of a batch of triples. In the ‘single’ mode, sample is a batch of triple. In the ‘head-batch’ or ‘tail-batch’ mode, sample consists two part. The first part is usually the positive sample. And the second part is the entities in the negative samples. Because negative samples and positive samples usually share two elements in their triple ((head, relation) or (relation, tail)).

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

**static train\_step** (*model, optimizer, train\_iterator, args*)  
A single train step. Apply back-propation and return the loss

**static test\_step** (*model, test\_triples, all\_true\_triples, args*)  
Evaluate the model on test or valid datasets

`cogdl.models.emb.line`

## Module Contents

### Classes

---

*LINE*

The LINE model from the [”Line: Large-scale information network embedding”](#)

---

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

Bases: `cogdl.models.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 (cls, args)
    Build a new model instance.

train (self, G)

_update (self, vec_u, vec_v, vec_error, label)

_train_line (self, order)
    
```

`cogdl.models.emb.metapath2vec`

## Module Contents

### Classes

---

|                     |   |
|---------------------|---|
| <i>Metapath2vec</i> | The Metapath2vec model from the <a href="#">“metapath2vec: Scalable Representation Learning for Heterogeneous Networks”</a> paper |
|---------------------|---|

---

```

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

Bases: `cogdl.models.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 (cls, args)
    Build a new model instance.

train (self, G, node_type)

_walk (self, start_node, walk_length, schema=None)

_simulate_walks (self, walk_length, num_walks, schema='No')
    
```

`cogdl.models.emb.netmf`

## Module Contents

### Classes

---

|              |  |
|--------------|--|
| <i>NetMF</i> | The NetMF model from the “Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec” |
|--------------|--|

---

**class** `cogdl.models.emb.netmf.NetMF` (*dimension, window\_size, rank, negative, is\_large=False*)

Bases: `cogdl.models.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** (*cls, args*)

Build a new model instance.

**train** (*self, G*)

**\_compute\_deepwalk\_matrix** (*self, A, window, b*)

**\_approximate\_normalized\_laplacian** (*self, A, rank, which='LA'*)

**\_deepwalk\_filter** (*self, evals, window*)

**\_approximate\_deepwalk\_matrix** (*self, evals, D\_rt\_invU, window, vol, b*)

`cogdl.models.emb.netsmf`

## Module Contents

### Classes

---

|               |  |
|---------------|--|
| <i>NetSMF</i> | The NetSMF model from the “NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization” |
|---------------|--|

---

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

Bases: `cogdl.models.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 nagative 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** (*cls, args*)

Build a new model instance.

**train** (*self, G*)

**\_get\_embedding\_rand** (*self, matrix*)

**\_path\_sampling** (*self, u, v, r*)

**\_random\_walk\_matrix** (*self, pid*)

`cogdl.models.emb.node2vec`

### Module Contents

#### Classes

---

`Node2vec`

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

---

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

Bases: `cogdl.models.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** (*cls, args*)

Build a new model instance.

**train** (*self, G*)

**\_node2vec\_walk** (*self, walk\_length, start\_node*)

**\_simulate\_walks** (*self, num\_walks, walk\_length*)

**\_get\_alias\_edge** (*self, src, dst*)

**\_preprocess\_transition\_probs** (*self*)

`cogdl.models.emb.prone`

## Module Contents

### Classes

---

|              |   |
|--------------|---|
| <i>ProNE</i> | The ProNE model from the “ProNE: Fast and Scalable Network Representation Learning” |
|--------------|---|

---

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

Bases: `cogdl.models.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** (*cls, args*)

Build a new model instance.

**train** (*self, G*)

**\_get\_embedding\_rand** (*self, matrix*)

**\_get\_embedding\_dense** (*self, matrix, dimension*)

**\_pre\_factorization** (*self, tran, mask*)

**\_chebyshev\_gaussian** (*self, A, a, order=5, mu=0.5, s=0.2, plus=False, nn=False*)

`cogdl.models.emb.pte`

## Module Contents

### Classes

---

|            |   |
|------------|---|
| <i>PTE</i> | The PTE model from the “PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks” |
|------------|---|

---

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

Bases: `cogdl.models.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 (cls, args)
    Build a new model instance.

train (self, G, node_type)

_update (self, vec_u, vec_v, vec_error, label)

_train_line (self)
```

`cogdl.models.emb.rotate`

## Module Contents

### Classes

---

|               |   |
|---------------|---|
| <i>RotatE</i> | Implementation of RotatE model from the paper “RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space” |
|---------------|---|

---

```
class cogdl.models.emb.rotate.RotatE (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>>

```
score (self, head, relation, tail, mode)
```

`cogdl.models.emb.sdne`

## Module Contents

### Classes

---

|                   |  |
|-------------------|--|
| <i>SDNE_layer</i> |  |
|-------------------|--|

---

|             |   |
|-------------|---|
| <i>SDNE</i> | The SDNE model from the “Structural Deep Network Embedding” |
|-------------|---|

---

```
class cogdl.models.emb.sdne.SDNE_layer (num_node, hidden_size1, hidden_size2, dropout,
                                       alpha, beta, nu1, nu2)
    Bases: torch.nn.Module
```

```
forward (self, adj_mat, l_mat)
```

```
get_emb (self, adj)
```

```
class cogdl.models.emb.sdne.SDNE (hidden_size1, hidden_size2, dropout, alpha, beta, nu1, nu2,
                                   max_epoch, lr, cpu)
    Bases: cogdl.models.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** (*cls, args*)

Build a new model instance.

**train** (*self, G*)

`cogdl.models.emb.spectral`

## Module Contents

### Classes

---

*Spectral*

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

---

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

Bases: `cogdl.models.BaseModel`

The Spectral clustering model from the “Leveraging social media networks for classication” 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** (*cls, args*)

Build a new model instance.

**train** (*self, G*)

`cogdl.models.emb.transe`

## Module Contents

### Classes

---

*TransE*

The TransE model from paper “Translating Embeddings for Modeling Multi-relational Data”

---

**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)” <<http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf>> borrowed from `KnowledgeGraphEmbedding` <<https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding>>

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

`cogdl.models.nn`

### Submodules

`cogdl.models.nn.asgcn`

### Module Contents

#### Classes

---

|                               |  |
|-------------------------------|--|
| <code>GraphConvolution</code> | Simple GCN layer, similar to <a href="https://arxiv.org/abs/1609.02907">https://arxiv.org/abs/1609.02907</a> |
| <code>ASGCN</code>            |  |

---

**class** `cogdl.models.nn.asgcn.GraphConvolution` (*in\_features*, *out\_features*, *bias=True*)

Bases: `torch.nn.Module`

Simple GCN layer, similar to <https://arxiv.org/abs/1609.02907>

**reset\_parameters** (*self*)

**forward** (*self*, *input*, *adj*)

**\_\_repr\_\_** (*self*)

**class** `cogdl.models.nn.asgcn.ASGCN` (*num\_features*, *num\_classes*, *hidden\_size*, *num\_layers*, *dropout*, *sample\_size*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**reset\_parameters** (*self*)

**set\_adj** (*self*, *edge\_index*, *num\_nodes*)

**compute\_adjlist** (*self*, *sp\_adj*, *max\_degree=32*)

Transfer sparse adjacent matrix to adj-list format

**from\_adjlist** (*self*, *adj*)

Transfer adj-list format to sparsensor

**\_sample\_one\_layer** (*self*, *x*, *adj*, *v*, *sample\_size*)

**sampling** (*self*, *x*, *v*)

**forward** (*self*, *x*, *adj*)

`cogdl.models.nn.compvcn`

## Module Contents

### Classes

---

*BasesRelEmbLayer*

---

*CompGCNLayer*

---

*CompGCN*

---

*LinkPredictCompGCN*

---

### Functions

|                             |               |   |
|-----------------------------|---------------|---|
| <code>com_mult(a, b)</code> | Borrowed from | <a href="https://github.com/malllabiisc/CompGCN">https://github.com/malllabiisc/CompGCN</a> |
| <code>conj(a)</code>        | Borrowed from | <a href="https://github.com/malllabiisc/CompGCN">https://github.com/malllabiisc/CompGCN</a> |
| <code>ccorr(a, b)</code>    | Borrowed from | <a href="https://github.com/malllabiisc/CompGCN">https://github.com/malllabiisc/CompGCN</a> |

`cogdl.models.nn.compvcn.com_mult(a, b)`  
 Borrowed from <https://github.com/malllabiisc/CompGCN>

`cogdl.models.nn.compvcn.conj(a)`  
 Borrowed from <https://github.com/malllabiisc/CompGCN>

`cogdl.models.nn.compvcn.ccorr(a, b)`  
 Borrowed from <https://github.com/malllabiisc/CompGCN>

**class** `cogdl.models.nn.compvcn.BasesRelEmbLayer(num_bases, num_rels, in_feats)`

Bases: `torch.nn.Module`

**reset\_parameters**(*self*)

**forward**(*self*)

**class** `cogdl.models.nn.compvcn.CompGCNLayer(in_feats, out_feats, num_rels, opn='mult', num_bases=None, activation=lambda x: ..., dropout=0.0, bias=True)`

Bases: `torch.nn.Module`

**get\_param**(*self*, num\_in, num\_out)

**forward**(*self*, x, edge\_index, edge\_type, rel\_embed=None)

**message\_passing**(*self*, x, rel\_embed, edge\_index, edge\_types, mode, edge\_weight=None)

**rel\_transform**(*self*, ent\_embed, rel\_embed)

**class** `cogdl.models.nn.compvcn.CompGCN(num_entities, num_rels, num_bases, in_feats, hidden_size, out_feats, layers, dropout, activation)`

Bases: `torch.nn.Module`

**forward** (*self*, *x*, *edge\_index*, *edge\_types*)

**class** cogdl.models.nn.compgcn.**LinkPredictCompGCN** (*num\_entities*, *num\_rels*, *hidden\_size*,  
*num\_bases*=0, *layers*=1, *sampling\_rate*=0.01, *score\_func*='conve',  
*penalty*=0.001, *dropout*=0.0,  
*lbl\_smooth*=0.1)

Bases: `cogdl.layers.link_prediction_module.GNNLinkPredict`, `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**add\_reverse\_edges** (*self*, *edge\_index*, *edge\_types*)

**forward** (*self*, *edge\_index*, *edge\_types*)

**loss** (*self*, *data*, *split*='train')

**predict** (*self*, *edge\_index*, *edge\_types*)

`cogdl.models.nn.dgi`

## Module Contents

### Classes

---

*GCN*

---

*AvgReadout*

---

*Discriminator*

---

*LogReg*

---

*LogRegTrainer*

---

*DGIModel*

---

*DGI*

---

## Functions

|  |  |
|--|--|
| <code>preprocess_features(features)</code>               | Row-normalize feature matrix and convert to tuple representation |
| <code>normalize_adj(adj)</code>                          | Symmetrically normalize adjacency matrix.                        |
| <code>sparse_mx_to_torch_sparse_tensor(sparse_mx)</code> | Convert a scipy sparse matrix to a torch sparse tensor.          |

```

class cogdl.models.nn.dgi.GCN (in_ft, out_ft, act, bias=True)
    Bases: torch.nn.Module

    weights_init (self, m)

    forward (self, seq, adj, sparse=False)

class cogdl.models.nn.dgi.AvgReadout
    Bases: torch.nn.Module

    forward (self, seq, msk)

class cogdl.models.nn.dgi.Discriminator (n_h)
    Bases: torch.nn.Module

    weights_init (self, m)

    forward (self, c, h_pl, h_mi, s_bias1=None, s_bias2=None)

class cogdl.models.nn.dgi.LogReg (ft_in, nb_classes)
    Bases: torch.nn.Module

    weights_init (self, m)

    forward (self, seq)

class cogdl.models.nn.dgi.LogRegTrainer
    Bases: object

    train (self, data, labels, opt)

class cogdl.models.nn.dgi.DGIModel (n_in, n_h, activation)
    Bases: torch.nn.Module

    forward (self, seq1, seq2, adj, sparse, msk, samp_bias1, samp_bias2)

    embed (self, seq, adj, sparse, msk)

cogdl.models.nn.dgi.preprocess_features (features)
    Row-normalize feature matrix and convert to tuple representation

cogdl.models.nn.dgi.normalize_adj (adj)
    Symmetrically normalize adjacency matrix.

cogdl.models.nn.dgi.sparse_mx_to_torch_sparse_tensor (sparse_mx)
    Convert a scipy sparse matrix to a torch sparse tensor.

class cogdl.models.nn.dgi.DGI (nfeat, nhid, nclass, max_epochs)
    Bases: cogdl.models.BaseModel

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

    classmethod build_model_from_args (cls, args)
        Build a new model instance.

```

`train (self, data)`

`cogdl.models.nn.dgl_gcc`

## Module Contents

### Classes

---

`NodeClassificationDataset`

---

`GraphClassificationDataset`

---

`GCC`

---

### Functions

---

`batcher()`

---

`test_moco(train_loader, model, opt)` one epoch training for moco

---

`eigen_decomposition(n, k, laplacian, hidden_size, retry)`

---

`_add_undirected_graph_positional_embedding(g, hidden_size, retry=10)`

---

`_rwr_trace_to_dgl_graph(g, seed, trace, positional_embedding_size, entire_graph=False)`

---

`cogdl.models.nn.dgl_gcc.batcher()`

`cogdl.models.nn.dgl_gcc.test_moco(train_loader, model, opt)`  
one epoch training for moco

`cogdl.models.nn.dgl_gcc.eigen_decomposition(n, k, laplacian, hidden_size, retry)`

`cogdl.models.nn.dgl_gcc._add_undirected_graph_positional_embedding(g, hidden_size, retry=10)`

`cogdl.models.nn.dgl_gcc._rwr_trace_to_dgl_graph(g, seed, trace, positional_embedding_size, entire_graph=False)`

**class** `cogdl.models.nn.dgl_gcc.NodeClassificationDataset` (`data`, `rw_hops=64`, `subgraph_size=64`, `restart_prob=0.8`, `positional_embedding_size=32`, `step_dist=[1.0, 0.0, 0.0]`)

Bases: `object`

`_create_dgl_graph(self, data)`

`__len__(self)`

`_convert_idx` (*self*, *idx*)

`__getitem__` (*self*, *idx*)

```
class cogdl.models.nn.dgl_gcc.GraphClassificationDataset (data, rw_hops=64,
                                                    subgraph_size=64,
                                                    restart_prob=0.8, posi-
                                                    tional_embedding_size=32,
                                                    step_dist=[1.0, 0.0, 0.0])

Bases: cogdl.models.nn.dgl_gcc.NodeClassificationDataset
```

`_convert_idx` (*self*, *idx*)

```
class cogdl.models.nn.dgl_gcc.GCC (load_path)

Bases: cogdl.models.BaseModel
```

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**train** (*self*, *data*)

`cogdl.models.nn.disengcn`

## Module Contents

### Classes

---

*DisenGCNLayer*

Implementation of “Disentangled Graph Convolutional Networks” <<http://proceedings.mlr.press/v97/ma19a.html>>.

---

*DisenGCN*

---

```
class cogdl.models.nn.disengcn.DisenGCNLayer (in_feats, out_feats, K, iterations, tau=1.0,
                                                    activation='leaky_relu')
```

Bases: `torch.nn.Module`

Implementation of “Disentangled Graph Convolutional Networks” <<http://proceedings.mlr.press/v97/ma19a.html>>.

**reset\_parameters** (*self*)

**forward** (*self*, *x*, *edge\_index*)

```
class cogdl.models.nn.disengcn.DisenGCN (in_feats, hidden_size, num_classes, K, iterations,
                                                    tau, dropout, activation)
```

Bases: *cogdl.models.BaseModel*

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**reset\_parameters** (*self*)

**forward** (*self*, *x*, *edge\_index*)

**loss** (*self*, *data*)

**predict** (*self*, *data*)

`cogdl.models.nn.fastgcn`

### Module Contents

#### Classes

---

|                         |  |
|-------------------------|--|
| <i>GraphConvolution</i> | Simple GCN layer, similar to <a href="https://arxiv.org/abs/1609.02907">https://arxiv.org/abs/1609.02907</a> |
| <i>FastGCN</i>          |  |

---

**class** `cogdl.models.nn.fastgcn.GraphConvolution` (*in\_features*, *out\_features*, *bias=True*)

Bases: `torch.nn.Module`

Simple GCN layer, similar to <https://arxiv.org/abs/1609.02907>

**reset\_parameters** (*self*)

**forward** (*self*, *input*, *adj*)

**\_\_repr\_\_** (*self*)

**class** `cogdl.models.nn.fastgcn.FastGCN` (*num\_features*, *num\_classes*, *hidden\_size*, *num\_layers*, *dropout*, *sample\_size*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**set\_adj** (*self*, *edge\_index*, *num\_nodes*)

**\_sample\_one\_layer** (*self*, *sampled*, *sample\_size*)

**\_generate\_adj** (*self*, *sample1*, *sample2*)

**sampling** (*self*, *x*, *v*)

**forward** (*self*, *x*, *adj*)



`cogdl.models.nn.gat`

## Module Contents

### Classes

|                              |  |
|------------------------------|--|
| <i>SpecialSpmFunction</i>    | Special function for only sparse region backpropataion layer.  |
| <i>SpecialSpm</i>            |  |
| <i>SpGraphAttentionLayer</i> | Sparse version GAT layer, similar to <a href="https://arxiv.org/abs/1710.10903">https://arxiv.org/abs/1710.10903</a> |
| <i>PetarVSpGAT</i>           | The GAT model from the “Graph Attention Networks”  |

**class** `cogdl.models.nn.gat.SpecialSpmFunction`

Bases: `torch.autograd.Function`

Special function for only sparse region backpropataion layer.

**static forward** (*ctx, indices, values, shape, b*)

**static backward** (*ctx, grad\_output*)

**class** `cogdl.models.nn.gat.SpecialSpm`

Bases: `torch.nn.Module`

**forward** (*self, indices, values, shape, b*)

**class** `cogdl.models.nn.gat.SpGraphAttentionLayer` (*in\_features, out\_features, dropout, alpha, concat=True*)

Bases: `torch.nn.Module`

Sparse version GAT layer, similar to <https://arxiv.org/abs/1710.10903>

**forward** (*self, input, edge*)

**\_\_repr\_\_** (*self*)

**class** `cogdl.models.nn.gat.PetarVSpGAT` (*nfeat, nhid, nclass, dropout, alpha, nheads*)

Bases: `cogdl.models.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** (*cls, args*)

Build a new model instance.

**forward** (*self, x, edge\_index*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.gcn`

## Module Contents

### Classes

---

|                               |  |
|-------------------------------|--|
| <code>GraphConvolution</code> | Simple GCN layer, similar to <a href="https://arxiv.org/abs/1609.02907">https://arxiv.org/abs/1609.02907</a> |
| <code>TKipfGCN</code>         | The GCN model from the “Semi-Supervised Classification with Graph Convolutional Networks”                    |

---

**class** `cogdl.models.nn.gcn.GraphConvolution` (*in\_features, out\_features, bias=True*)

Bases: `torch.nn.Module`

Simple GCN layer, similar to <https://arxiv.org/abs/1609.02907>

**reset\_parameters** (*self*)

**forward** (*self, input, edge\_index, edge\_attr=None*)

**\_\_repr\_\_** (*self*)

**class** `cogdl.models.nn.gcn.TKipfGCN` (*nfeat, nhid, nclass, dropout*)

Bases: `cogdl.models.BaseModel`

The GCN model from the “Semi-Supervised Classification with Graph Convolutional 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.

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**forward** (*self, x, adj*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.gcnii`

## Module Contents

### Classes

---

|                         |
|-------------------------|
| <code>GCNIILayer</code> |
| <code>GCNII</code>      |

---

**class** `cogdl.models.nn.gcnii.GCNIILayer` (*n\_channels, alpha=0.1, beta=1, residual=False*)

Bases: `torch.nn.Module`

```

    reset_parameters (self)
    forward (self, x, edge_index, edge_attr, init_x)
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)
    Bases: cogdl.models.BaseModel
    static add_args (parser)
        Add model-specific arguments to the parser.
    classmethod build_model_from_args (cls, args)
        Build a new model instance.
    forward (self, x, edge_index, edge_attr=None)
    loss (self, data)
    predict (self, data)
    get_optimizer (self, args)

```

`cogdl.models.nn.graphsage`

## Module Contents

### Classes

---

*GraphSAGELayer*

---

*Graphsage*

---

### Functions

---

*sage\_sampler*(*adjlist*, *edge\_index*, *num\_sample*)

---

`cogdl.models.nn.graphsage.sage_sampler` (*adjlist*, *edge\_index*, *num\_sample*)

**class** `cogdl.models.nn.graphsage.GraphSAGELayer` (*in\_feats*, *out\_feats*)  
 Bases: `torch.nn.Module`

**forward** (*self*, *x*, *edge\_index*)

**class** `cogdl.models.nn.graphsage.Graphsage` (*num\_features*, *num\_classes*, *hidden\_size*,  
*num\_layers*, *sample\_size*, *dropout*)

Bases: *cogdl.models.BaseModel*

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

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

**sampling** (*self*, *edge\_index*, *num\_sample*)

```
forward (self, x, edge_index)  
loss (self, data)  
predict (self, data)
```

`cogdl.models.nn.mixhop`

### Module Contents

#### Classes

---

*MixHop*

---

```
class cogdl.models.nn.mixhop.MixHop (num_features, num_classes, dropout, layer1_pows,  
                                         layer2_pows)  
    Bases: cogdl.models.BaseModel  
  
    static add_args (parser)  
        Add model-specific arguments to the parser.  
  
    classmethod build_model_from_args (cls, args)  
        Build a new model instance.  
  
    forward (self, x, edge_index)  
  
    loss (self, data)  
  
    predict (self, data)
```

`cogdl.models.nn.mlp`

### Module Contents

#### Classes

---

*MLP*

---

```
class cogdl.models.nn.mlp.MLP (num_features, num_classes, hidden_size, num_layers, dropout)  
    Bases: cogdl.models.BaseModel  
  
    static add_args (parser)  
        Add model-specific arguments to the parser.  
  
    classmethod build_model_from_args (cls, args)  
        Build a new model instance.  
  
    forward (self, x, edge_index)  
  
    loss (self, data)  
  
    predict (self, data)
```

`cogdl.models.nn.mvgrl`

## Module Contents

### Classes

---

*Discriminator*

---

---

*Model*

---

---

*MVGRL*

---

### Functions

---

*preprocess\_features*(features) Row-normalize feature matrix and convert to tuple representation

---

*normalize\_adj*(adj) Symmetrically normalize adjacency matrix.

---

*sparse\_mx\_to\_torch\_sparse\_tensor*(sparse\_mx) Convert a scipy sparse matrix to a torch sparse tensor.

---

*compute\_ppr*(graph: networkx.Graph, alpha=0.2, self\_loop=True)

---

**class** `cogdl.models.nn.mvgrl.Discriminator` (*n\_h*)Bases: `torch.nn.Module`**weights\_init** (*self, m*)**forward** (*self, c1, c2, h1, h2, h3, h4, s\_bias1=None, s\_bias2=None*)**class** `cogdl.models.nn.mvgrl.Model` (*n\_in, n\_h*)Bases: `torch.nn.Module`**forward** (*self, seq1, seq2, adj, diff, sparse, msk, samp\_bias1, samp\_bias2*)**embed** (*self, seq, adj, diff, sparse, msk*)`cogdl.models.nn.mvgrl.preprocess_features` (*features*)

Row-normalize feature matrix and convert to tuple representation

`cogdl.models.nn.mvgrl.normalize_adj` (*adj*)

Symmetrically normalize adjacency matrix.

`cogdl.models.nn.mvgrl.sparse_mx_to_torch_sparse_tensor` (*sparse\_mx*)

Convert a scipy sparse matrix to a torch sparse tensor.

`cogdl.models.nn.mvgrl.compute_ppr` (*graph: networkx.Graph, alpha=0.2, self\_loop=True*)**class** `cogdl.models.nn.mvgrl.MVGRL` (*nfeat, nhid, nclass, max\_epochs*)Bases: `cogdl.models.BaseModel`**static add\_args** (*parser*)

Add model-specific arguments to the parser.

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

Build a new model instance.

`train` (*self*, *data*, *dataset\_name*)

`cogdl.models.nn.patchy_san`

## Module Contents

### Classes

---

|                        |   |
|------------------------|---|
| <code>PatchySAN</code> | The Patchy-SAN model from the “Learning Convolutional Neural Networks for Graphs” |
|------------------------|---|

---

### Functions

---

|  |   |
|--|---|
| <code>assemble_neighbor</code> ( <i>G</i> , <i>node</i> , <i>num_neighbor</i> , <i>sorted_nodes</i> )  | assemble neighbors for node with BFS strategy                           |
| <code>cmp</code> ( <i>s1</i> , <i>s2</i> )   |   |
| <code>one_dim_wl</code> ( <i>graph_list</i> , <i>init_labels</i> , <i>iteration=5</i> )  | 1-dimension WL method used for node normalization for all the subgraphs |
| <code>node_selection_with_1d_wl</code> ( <i>G</i> , <i>features</i> , <i>num_channel</i> , <i>num_sample</i> , <i>num_neighbor</i> , <i>stride</i> )   | construct features for cnn  |
| <code>get_single_feature</code> ( <i>data</i> , <i>num_features</i> , <i>num_classes</i> , <i>num_sample</i> , <i>num_neighbor</i> , <i>stride=1</i> ) | construct features  |

---

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

Bases: `cogdl.models.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.

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

Build a new model instance.

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

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

**forward** (*self*, *batch*)

`cogdl.models.nn.patchy_san.assemble_neighbor` (*G*, *node*, *num\_neighbor*, *sorted\_nodes*)  
assemble neighbors for node with BFS strategy

`cogdl.models.nn.patchy_san.cmp` (*s1*, *s2*)

`cogdl.models.nn.patchy_san.one_dim_wl` (*graph\_list*, *init\_labels*, *iteration=5*)  
1-dimension WL method used for node normalization for all the subgraphs

---

```
cogdl.models.nn.patchy_san.node_selection_with_1d_wl(G, features, num_channel,
                                                    num_sample, num_neighbor,
                                                    stride)
```

construct features for cnn

```
cogdl.models.nn.patchy_san.get_single_feature(data, num_features, num_classes,
                                              num_sample, num_neighbor, stride=1)
```

construct features

**cogdl.models.nn.pyg\_cheb**

## Module Contents

### Classes

---

*Chebyshev*

---

```
class cogdl.models.nn.pyg_cheb.Chebyshev(num_features, num_classes, hidden_size,
                                         num_layers, dropout, filter_size)
```

Bases: *cogdl.models.BaseModel*

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**forward** (*self, x, edge\_index*)

**loss** (*self, data*)

**predict** (*self, data*)

**cogdl.models.nn.pyg\_deepergcn**

## Module Contents

### Classes

---

*GENConv*

---

*DeepGCNLayer*

---

*DeeperGCN*

---

```
class cogdl.models.nn.pyg_deepergcn.GENConv(in_feat, out_feat, aggr='softmax_sg',
                                             beta=1.0, p=1.0, learn_beta=False,
                                             learn_p=False, use_msg_norm=False,
                                             learn_msg_scale=True)
```

Bases: *torch.nn.Module*

**message\_norm** (*self*, *x*, *msg*)

**forward** (*self*, *x*, *edge\_index*, *edge\_attr=None*)

**class** cogdl.models.nn.pyg\_deepergcn.**DeepGCNLayer** (*in\_feat*, *out\_feat*, *conv*,  
*connection='res'*, *activation='relu'*, *dropout=0.0*, *checkpoint\_grad=False*)

Bases: torch.nn.Module

**forward** (*self*, *x*, *edge\_index*)

**class** cogdl.models.nn.pyg\_deepergcn.**DeeperGCN** (*in\_feat*, *hidden\_size*, *out\_feat*,  
*num\_layers*, *connection='res+'*, *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*)

Bases: cogdl.models.BaseModel

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

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

**forward** (*self*, *x*, *edge\_index*, *edge\_attr=None*)

**loss** (*self*, *x*, *edge\_index*, *y*, *x\_mask*)

**predict** (*self*, *x*, *edge\_index*)

**static get\_trainer** (*taskType: Any*, *args*)

cogdl.models.nn.pyg\_dgcnn

## Module Contents

### Classes

---

*DGCNN*

EdgeConv and DynamicGraph in paper [“Dynamic Graph CNN for Learning on](#)

---

**class** cogdl.models.nn.pyg\_dgcnn.**DGCNN** (*in\_feats*, *hidden\_dim*, *out\_feats*, *k=20*, *dropout=0.5*)

Bases: cogdl.models.BaseModel

EdgeConv and DynamicGraph in paper [“Dynamic Graph CNN for Learning on Point Clouds”](https://arxiv.org/pdf/1801.07829.pdf)  
 <<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 (cls, args)
    Build a new model instance.

classmethod split_dataset (cls, dataset, args)

forward (self, batch)

```

`cogdl.models.nn.pyg_diffpool`

## Module Contents

### Classes

|                             |   |
|-----------------------------|---|
| <i>EntropyLoss</i>          |   |
| <i>LinkPredLoss</i>         |   |
| <i>GraphSAGE</i>            | GraphSAGE from “Inductive Representation Learning on Large Graphs”. |
| <i>BatchedGraphSAGE</i>     | GraphSAGE with mini-batch   |
| <i>BatchedDiffPoolLayer</i> | DIFFPOOL from paper “Hierarchical Graph Representation Learning     |
| <i>BatchedDiffPool</i>      | DIFFPOOL layer with batch forward                                   |
| <i>DiffPool</i>             | DIFFPOOL from paper “Hierarchical Graph Representation Learning     |

### Functions

|  |
|--|
| <i>toBatchedGraph</i> ( <i>batch_adj,</i> <i>batch_feat,</i><br><i>node_per_pool_graph</i> ) |
|--|

```

class cogdl.models.nn.pyg_diffpool.EntropyLoss

```

```

    Bases: torch.nn.Module

```

```

    forward (self, adj, anext, s_l)

```

```

class cogdl.models.nn.pyg_diffpool.LinkPredLoss

```

```

    Bases: torch.nn.Module

```

```

    forward (self, adj, anext, s_l)

```

```

class cogdl.models.nn.pyg_diffpool.GraphSAGE (in_feats, hidden_dim, out_feats,  
num_layers, dropout=0.5, normalize=False, concat=False, use_bn=False)

```

```

    Bases: torch.nn.Module

```

GraphSAGE from “Inductive Representation Learning on Large Graphs”.

```

..math::  $h^{i+1}_{\mathcal{N}(v)} = \text{AGGREGATE}_{\{k\}}(h_{\mathcal{U}}^k) \oplus h^{k+1}_{\mathcal{V}}$ 
            $= \text{sigma}(\mathbf{W}^k \cdot \text{CONCAT}(h_{\mathcal{V}}^k, h_{\mathcal{N}(v)}))$ 

```

**Args:** *in\_feats* (int) : Size of each input sample. *hidden\_dim* (int) : Size of hidden layer dimension. *out\_feats* (int) : Size of each output sample. *num\_layers* (int) : Number of GraphSAGE Layers. *dropout* (float,

optional) : Size of dropout, default: 0.5. normalize (bool, optional) : Normalize features after each layer if True, default: True.

**forward** (*self*, *x*, *edge\_index*, *edge\_weight=None*)

**class** cogdl.models.nn.pyg\_diffpool.**BatchedGraphSAGE** (*in\_feats*, *out\_feats*,  
*use\_bn=True*, *self\_loop=True*)

Bases: torch.nn.Module

GraphSAGE with mini-batch

**Args:** *in\_feats* (int) : Size of each input sample. *out\_feats* (int) : Size of each output sample. *use\_bn* (bool) : Apply batch normalization if True, default: True. *self\_loop* (bool) : Add self loop if True, default: True.

**forward** (*self*, *x*, *adj*)

**class** cogdl.models.nn.pyg\_diffpool.**BatchedDiffPoolLayer** (*in\_feats*, *out\_feats*, *assign\_dim*, *batch\_size*,  
*dropout=0.5*,  
*link\_pred\_loss=True*,  
*entropy\_loss=True*)

Bases: torch.nn.Module

DIFFPOOL from paper “Hierarchical Graph Representation Learning with Differentiable Pooling”.

$$X^{(l+1)} = S^{(l)T} Z^{(l)} A^{(l+1)} = S^{(l)T} A^{(l)} S^{(l)} Z^{(l)} = GNN_{l,embed}(A^{(l)}, X^{(l)}) S^{(l)} = softmax(GNN_{l,pool}(A^{(l)}, X^{(l)}))$$

**in\_feats** [int] Size of each input sample.

**out\_feats** [int] Size of each output sample.

**assign\_dim** [int] Size of next adjacency matrix.

**batch\_size** [int] Size of each mini-batch.

**dropout** [float, optional] Size of dropout, default: 0.5.

**link\_pred\_loss** [bool, optional] Use link prediction loss if True, default: True.

**forward** (*self*, *x*, *edge\_index*, *batch*, *edge\_weight=None*)

**get\_loss** (*self*)

**class** cogdl.models.nn.pyg\_diffpool.**BatchedDiffPool** (*in\_feats*, *next\_size*, *emb\_size*,  
*use\_bn=True*, *self\_loop=True*,  
*use\_link\_loss=False*,  
*use\_entropy=True*)

Bases: torch.nn.Module

DIFFPOOL layer with batch forward

**in\_feats** [int] Size of each input sample.

**next\_size** [int] Size of next adjacency matrix.

**emb\_size** [int] Dimension of next node feature matrix.

**use\_bn** [bool, optional] Apply batch normalization if True, default: True.

**self\_loop** [bool, optional] Add self loop if True, default: True.

**use\_link\_loss** [bool, optional] Use link prediction loss if True, default: True.

**use\_entropy** [bool, optional] Use entropy prediction loss if True, default: True.

**forward** (*self*, *x*, *adj*)

**get\_loss** (*self*)

`cogdl.models.nn.pyg_diffpool.toBatchedGraph` (*batch\_adj*, *batch\_feat*,  
*node\_per\_pool\_graph*)

**class** `cogdl.models.nn.pyg_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.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.

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

Build a new model instance.

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

**reset\_parameters** (*self*)

**after\_pooling\_forward** (*self*, *gnn\_layers*, *adj*, *x*, *concat=False*)

**forward** (*self*, *batch*)

**loss** (*self*, *prediction*, *label*)

`cogdl.models.nn.pyg_drgat`

## Module Contents

### Classes

---

*DrGAT*

---

**class** `cogdl.models.nn.pyg_drgat.DrGAT` (*num\_features, num\_classes, hidden\_size, num\_heads, dropout*)

Bases: `cogdl.models.BaseModel`

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

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

**forward** (*self, x, edge\_index*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.pyg_drgcn`

### Module Contents

#### Classes

---

*DrGCN*

---

**class** `cogdl.models.nn.pyg_drgcn.DrGCN` (*num\_features, num\_classes, hidden\_size, num\_layers, dropout*)

Bases: `cogdl.models.BaseModel`

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

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

**forward** (*self, x, edge\_index*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.pyg_gat`

### Module Contents

#### Classes

---

*GAT*

---

**class** `cogdl.models.nn.pyg_gat.GAT` (*num\_features, num\_classes, hidden\_size, num\_heads, dropout*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*cls, args*)

Build a new model instance.

**forward** (*self, x, edge\_index*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.pyg_gcn`

## Module Contents

### Classes

---

*GCN*

---

**class** `cogdl.models.nn.pyg_gcn.GCN` (*num\_features, num\_classes, hidden\_size, num\_layers, dropout*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

**classmethod** `build_model_from_args` (*cls, args*)

Build a new model instance.

**get\_trainer** (*self, task, args*)

**forward** (*self, x, edge\_index, weight=None*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.pyg_gcnmix`

## Module Contents

### Classes

---

*GCNConv*

---

*BaseGNNMix*

---

*GCNMix*

---

## Functions

---

*mix\_hidden\_state*(feat, target, train\_index, alpha)

---

*sharpen*(prob, temperature)

---

*get\_one\_hot\_label*(labels, index)

---

*get\_current\_consistency\_weight*(final\_consistency\_weight, rampup\_starts, rampup\_ends, epoch)

---

`cogdl.models.nn.pyg_gcnmix.mix_hidden_state` (*feat, target, train\_index, alpha*)

`cogdl.models.nn.pyg_gcnmix.sharpen` (*prob, temperature*)

`cogdl.models.nn.pyg_gcnmix.get_one_hot_label` (*labels, index*)

`cogdl.models.nn.pyg_gcnmix.get_current_consistency_weight` (*final\_consistency\_weight, rampup\_starts, rampup\_ends, epoch*)

**class** `cogdl.models.nn.pyg_gcnmix.GCNConv` (*in\_feats, out\_feats*)  
 Bases: `torch.nn.Module`

**forward** (*self, x, edge\_index, edge\_attr=None*)

**forward\_aux** (*self, x*)

**class** `cogdl.models.nn.pyg_gcnmix.BaseGNNMix` (*in\_feat, hidden\_size, num\_classes, k, temperature, alpha, dropout*)

Bases: `cogdl.models.BaseModel`

**forward** (*self, x, edge\_index*)

**forward\_aux** (*self, x, label, train\_index, mix\_hidden=True, layer\_mix=1*)

**update\_aux** (*self, data, vector\_labels, train\_index, opt*)

**update\_soft** (*self, data, labels, train\_index*)

**loss** (*self, data, opt*)

**predict\_noise** (*self, data, tau=1*)

**class** `cogdl.models.nn.pyg_gcnmix.GCNMix` (*in\_feat, hidden\_size, num\_classes, k, temperature, alpha, rampup\_starts, rampup\_ends, final\_consistency\_weight, ema\_decay, dropout*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**forward** (*self, x, edge\_index*)

**forward\_ema** (*self, x, edge\_index*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.pyg_gin`

## Module Contents

### Classes

|                       |  |
|-----------------------|--|
| <code>GINLayer</code> | Graph Isomorphism Network layer from paper “How Powerful are Graph |
| <code>GINMLP</code>   | Multilayer perception with batch normalization                     |
| <code>GIN</code>      | Graph Isomorphism Network from paper “How Powerful are Graph       |

**class** `cogdl.models.nn.pyg_gin.GINLayer` (*apply\_func=None, eps=0, train\_eps=True*)  
 Bases: `torch.nn.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** (*self, x, edge\_index, edge\_weight=None*)

**class** `cogdl.models.nn.pyg_gin.GINMLP` (*in\_feats, out\_feats, hidden\_dim, num\_layers, use\_bn=True, activation=None*)

Bases: `torch.nn.Module`

Multilayer perception with batch 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** (*self, x*)

**class** `cogdl.models.nn.pyg_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.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** (*cls, args*)  
Build a new model instance.

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

**forward** (*self, batch*)

**loss** (*self, output, label=None*)

`cogdl.models.nn.pyg_gpt_gnn`

## Module Contents

### Classes

---

*GPT\_GNN*

Helper class that provides a standard way to create an ABC using

---

**class** `cogdl.models.nn.pyg_gpt_gnn.GPT_GNN`

Bases: `cogdl.models.supervised_model.SupervisedHomogeneousNodeClassificationModel`,  
`cogdl.models.supervised_model.SupervisedHeterogeneousNodeClassificationModel`

Helper class that provides a standard way to create an ABC using inheritance.

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

Add task-specific arguments to the parser.

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

Build a new model instance.

**loss** (*self, data: Any*) → Any

**predict** (*self, data: Any*) → Any

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

**static get\_trainer** (*taskType: Any, args*) → Optional[Type[Union[GPT\_GNNHomogeneousTrainer,  
GPT\_GNNHeterogeneousTrainer]]]



`cogdl.models.nn.pyg_grand`

## Module Contents

### Classes

---

*MLP*Layer

---

*Grand*

---

**class** `cogdl.models.nn.pyg_grand.MLP`Layer(*in\_features, out\_features, bias=True*)

Bases: `torch.nn.Module`

**reset\_parameters**(*self*)

**forward**(*self, x*)

**\_\_repr\_\_**(*self*)

**class** `cogdl.models.nn.pyg_grand.Grand`(*nfeat, nhid, nclass, input\_dropout, hidden\_dropout, use\_bn, dropout\_rate, tem, lam, order, sample, alpha*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**dropNode**(*self, x*)

**normalize\_adj**(*self, edge\_index, edge\_weight, num\_nodes*)

**rand\_prop**(*self, x, edge\_index, edge\_weight*)

**consis\_loss**(*self, logps, train\_mask*)

**normalize\_x**(*self, x*)

**forward**(*self, x, edge\_index*)

**adj** = `torch.sparse_coo_tensor`(*edge\_index, torch.ones(edge\_index.shape[1]).float(), (x.shape[0], x.shape[0]),*

*)*.to(*x.device*)

**loss**(*self, data*)

**predict**(*self, data*)

`cogdl.models.nn.pyg_gtn`

### Module Contents

#### Classes

---

*GTConv*

---

*GTLayer*

---

*GTN*

---

**class** `cogdl.models.nn.pyg_gtn.GTConv` (*in\_channels, out\_channels, num\_nodes*)

Bases: `torch.nn.Module`

**reset\_parameters** (*self*)

**forward** (*self, A*)

**class** `cogdl.models.nn.pyg_gtn.GTLayer` (*in\_channels, out\_channels, num\_nodes, first=True*)

Bases: `torch.nn.Module`

**forward** (*self, A, H=None*)

**class** `cogdl.models.nn.pyg_gtn.GTN` (*num\_edge, num\_channels, w\_in, w\_out, num\_class, num\_nodes, num\_layers*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**normalization** (*self, H*)

**norm** (*self, edge\_index, num\_nodes, edge\_weight, improved=False, dtype=None*)

**forward** (*self, A, X, target\_x, target*)

**loss** (*self, data*)

**evaluate** (*self, data, nodes, targets*)

`cogdl.models.nn.pyg_han`

### Module Contents

#### Classes

---

*AttentionLayer*

---

*HANLayer*

---

continues on next page

Table 102 – continued from previous page

HAN

---

```

class cogdl.models.nn.pyg_han.AttentionLayer (num_features)
    Bases: torch.nn.Module

    forward (self, x)

class cogdl.models.nn.pyg_han.HANLayer (num_edge, w_in, w_out)
    Bases: torch.nn.Module

    forward (self, x, adj)

class cogdl.models.nn.pyg_han.HAN (num_edge, w_in, w_out, num_class, num_nodes,
                                     num_layers)
    Bases: cogdl.models.BaseModel

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

    classmethod build_model_from_args (cls, args)
        Build a new model instance.

    forward (self, A, X, target_x, target)

    loss (self, data)

    evaluate (self, data, nodes, targets)

```

**cogdl.models.nn.pyg\_infograph**

## Module Contents

### Classes

|                   |   |
|-------------------|---|
| <i>SUPEncoder</i> | Encoder used in supervised model with Set2set in paper <a href="#">“Order Matters: Sequence to sequence for sets”</a>         |
| <i>Encoder</i>    | Encoder stacked with GIN layers   |
| <i>FF</i>         | Residual MLP layers.  |
| <i>InfoGraph</i>  | Implimentation of Infograph in paper <a href="#">“InfoGraph: Un-supervised and Semi-supervised Graph-Level Representation</a> |

```

class cogdl.models.nn.pyg_infograph.SUPEncoder (num_features, dim, num_layers=1)
    Bases: torch.nn.Module

    Encoder used in supervised model with Set2set in paper “Order Matters: Sequence to sequence for sets”
    <https://arxiv.org/abs/1511.06391> and NNConv in paper “Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs”
    <https://arxiv.org/abs/1704.02901>

    forward (self, x, edge_index, batch, edge_attr)

class cogdl.models.nn.pyg_infograph.Encoder (in_feats, hidden_dim, num_layers=3,
                                             num_mlp_layers=2, pooling='sum')
    Bases: torch.nn.Module

    Encoder stacked with GIN layers

```

**in\_feats** [int] Size of each input sample.

**hidden\_feats** [int] Size of output embedding.

**num\_layers** [int, optional] Number of GIN layers, default: 3.

**num\_mlp\_layers** [int, optional] Number of MLP layers for each GIN layer, default: 2.

**pooling** [str, optional] Aggragation type, default : sum.

**forward** (*self*, *x*, *edge\_index*, *batch*, \**args*)

**class** cogdl.models.nn.pyg\_infograph.**FF** (*in\_feats*, *out\_feats*)

Bases: torch.nn.Module

Residual MLP layers.

**..math::**  $out = \text{mathbf{MLP}}(x) + \text{mathbf{Linear}}(x)$

**in\_feats** [int] Size of each input sample

**out\_feats** [int] Size of each output sample

**forward** (*self*, *x*)

**class** cogdl.models.nn.pyg\_infograph.**InfoGraph** (*in\_feats*, *hidden\_dim*, *out\_feats*,  
*num\_layers=3*, *sup=False*)

Bases: *cogdl.models.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)** <<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** (*cls*, *args*)

Build a new model instance.

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

**reset\_parameters** (*self*)

**forward** (*self*, *batch*)

**sup\_forward** (*self*, *x*, *edge\_index=None*, *batch=None*, *label=None*, *edge\_attr=None*)

**unsup\_forward** (*self*, *x*, *edge\_index=None*, *batch=None*)

**sup\_loss** (*self*, *prediction*, *label=None*)

**unsup\_loss** (*self*, *x*, *edge\_index=None*, *batch=None*)

**unsup\_sup\_loss** (*self*, *x*, *edge\_index*, *batch*)

**static mi\_loss** (*pos\_mask*, *neg\_mask*, *mi*, *pos\_div*, *neg\_div*)

---

`cogdl.models.nn.pyg_infomax`

## Module Contents

### Classes

---

*Encoder*

---

*Infomax*

---

### Functions

---

*corruption*(*x*, *edge\_index*)

---

**class** `cogdl.models.nn.pyg_infomax.Encoder` (*in\_channels*, *hidden\_channels*)

Bases: `torch.nn.Module`

**forward** (*self*, *x*, *edge\_index*)

`cogdl.models.nn.pyg_infomax.corruption` (*x*, *edge\_index*)

**class** `cogdl.models.nn.pyg_infomax.Infomax` (*num\_features*, *num\_classes*, *hidden\_size*)

Bases: `cogdl.models.BaseModel`

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

Add model-specific arguments to the parser.

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

Build a new model instance.

**forward** (*self*, *x*, *edge\_index*)

**loss** (*self*, *data*)

**predict** (*self*, *data*)

`cogdl.models.nn.pyg_sortpool`

## Module Contents

### Classes

---

*SortPool*

Implimentation of sortpooling in paper ["An End-to-End Deep Learning](#)

---

### Functions

---

`scatter_sum(src, index, dim, dim_size)`

---

`spare2dense_batch(x, batch=None, fill_value=0)`

---

`cogdl.models.nn.pyg_sortpool.scatter_sum(src, index, dim, dim_size)`

`cogdl.models.nn.pyg_sortpool.spare2dense_batch(x, batch=None, fill_value=0)`

**class** `cogdl.models.nn.pyg_sortpool.SortPool` (*in\_feats*, *hidden\_dim*, *num\_classes*,  
*num\_layers*, *out\_channel*, *kernel\_size*,  
*k=30*, *dropout=0.5*)

Bases: `cogdl.models.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** (*cls*, *args*)

Build a new model instance.

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

**forward** (*self*, *batch*)

`cogdl.models.nn.pyg_srgcn`

### Module Contents

#### Classes

---

`NodeAdaptiveEncoder`

---

`SrgcnHead`

---

continues on next page

Table 108 – continued from previous page

---

*SrgcnSoftmaxHead*

---

*SRGCN*

---

```

class cogdl.models.nn.pyg_srgcn.NodeAdaptiveEncoder (num_features, dropout=0.5)
    Bases: cogdl.layers.srgcn_module.nn.Module

    forward (self, x)

class cogdl.models.nn.pyg_srgcn.SrgcnHead (num_features, out_feats, attention, activation,
                                             normalization, nhop, subheads=2, dropout=0.5,
                                             node_dropout=0.5, alpha=0.2, concat=True)
    Bases: cogdl.layers.srgcn_module.nn.Module

    forward (self, x, edge_index, edge_attr)

class cogdl.models.nn.pyg_srgcn.SrgcnSoftmaxHead (num_features, out_feats, attention,
                                                    activation, nhop, normalization,
                                                    dropout=0.5, node_dropout=0.5,
                                                    alpha=0.2)
    Bases: cogdl.layers.srgcn_module.nn.Module

    forward (self, x, edge_index, edge_attr)

class cogdl.models.nn.pyg_srgcn.SRGCN (num_features, hidden_size, num_classes, atten-
                                          tion, activation, nhop, normalization, dropout,
                                          node_dropout, alpha, nhead, subheads)
    Bases: cogdl.models.BaseModel

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

    classmethod build_model_from_args (cls, args)
        Build a new model instance.

    forward (self, batch)

    loss (self, data)

    predict (self, data)

```

**cogdl.models.nn.pyg\_stpgnn**

## Module Contents

### Classes

---

*stpgnn*

---

```

class cogdl.models.nn.pyg_stpgnn.stpgnn (args)
    Bases: cogdl.models.BaseModel

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

```

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

`cogdl.models.nn.pyg_unet`

### Module Contents

#### Classes

---

*UNet*

---

**class** `cogdl.models.nn.pyg_unet.UNet` (*num\_features, num\_classes, hidden\_size, num\_layers, dropout, num\_nodes*)

Bases: `cogdl.models.BaseModel`

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

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

**forward** (*self, x, edge\_index*)

**loss** (*self, data*)

**predict** (*self, data*)

`cogdl.models.nn.pyg_unsup_graphsage`

### Module Contents

#### Classes

---

*SAGE*

---

*Graphsage*

---

**class** `cogdl.models.nn.pyg_unsup_graphsage.SAGE` (*num\_features, hidden\_size, num\_layers, sample\_size, dropout, walk\_length, negative\_samples*)

Bases: `torch.nn.Module`

**sampling** (*self, edge\_index, num\_sample*)

**forward** (*self, x, edge\_index*)

**loss** (*self, data*)

**embed** (*self, data*)



```
class cogdl.models.nn.pyg_unsup_graphsage.Graphsage (num_features, hidden_size,
                                                num_classes, num_layers, sample_size, dropout, walk_length,
                                                negative_samples, lr, epochs,
                                                patience)
```

Bases: `cogdl.models.BaseModel`

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

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

```
train (self, data)
```

`cogdl.models.nn.rgcn`

## Module Contents

### Classes

---

`RGCNLayer`

---

`RGCN`

---

`LinkPredictRGCN`

---

```
class cogdl.models.nn.rgcn.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.Module`

```
reset_parameters (self)
```

```
forward (self, x, edge_index, edge_type)
```

```
basis_forward (self, x, edge_index, edge_type)
```

```
bdd_forward (self, x, edge_index, edge_type)
```

```
class cogdl.models.nn.rgcn.RGCN (in_feats, out_feats, num_layers, num_rels, regularizer='basis',
                                  num_bases=None, self_loop=True, dropout=0.0,
                                  self_dropout=0.0)
```

Bases: `torch.nn.Module`

```
forward (self, x, edge_index, edge_type)
```

```
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.layers.link_prediction_module.GNNLinkPredict`, `cogdl.models.BaseModel`

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

classmethod build_model_from_args (cls, args)
    Build a new model instance.

forward (self, edge_index, edge_type)

loss (self, data, split='train')

predict (self, edge_index, edge_type)
```

### Submodules

`cogdl.models.base_model`

### Module Contents

#### Classes

---

*BaseModel*

---

```
class cogdl.models.base_model.BaseModel
    Bases: torch.nn.Module

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

    abstract classmethod build_model_from_args (cls, args)
        Build a new model instance.

    _forward_unimplemented (self, *input: Any) → None

    static get_trainer (taskType: Any, args: Any) → Optional[Type[BaseTrainer]]
```

`cogdl.models.supervised_model`

### Module Contents

#### Classes

---

|  |  |
|--|--|
| <i>SupervisedModel</i>                           | Helper class that provides a standard way to create an ABC using |
| <i>SupervisedHeterogeneousNodeClassification</i> | Helper class that provides a standard way to create an ABC using |
| <i>SupervisedHomogeneousNodeClassification</i>   | Helper class that provides a standard way to create an ABC using |

---

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

Helper class that provides a standard way to create an ABC using inheritance.

```
abstract loss (self, data: Any) → Any
```

```
class cogdl.models.supervised_model.SupervisedHeterogeneousNodeClassificationModel
```

Bases: `cogdl.models.BaseModel`, `abc.ABC`

Helper class that provides a standard way to create an ABC using inheritance.

```
abstract loss (self, data: Any) → Any
```

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

```
static get_trainer (taskType: Any, args: Any) → Optional[Type[SupervisedHeterogeneousNodeClassificationTrainer]]
```

```
class cogdl.models.supervised_model.SupervisedHomogeneousNodeClassificationModel
```

Bases: `cogdl.models.BaseModel`, `abc.ABC`

Helper class that provides a standard way to create an ABC using inheritance.

```
abstract loss (self, data: Any) → Any
```

```
abstract predict (self, data: Any) → Any
```

```
static get_trainer (taskType: Any, args: Any) → Optional[Type[SupervisedHomogeneousNodeClassificationTrainer]]
```

## Package Contents

### Classes

---

`BaseModel`

---

### Functions

|                                   |  |
|-----------------------------------|--|
| <code>register_model(name)</code> | New model types can be added to cogdl with the <code>register_model()</code> |
| <code>alias_setup(probs)</code>   | Compute utility lists for non-uniform sampling from discrete distributions.  |
| <code>alias_draw(J, q)</code>     | Draw sample from a non-uniform discrete distribution using alias sampling.   |
| <code>build_model(args)</code>    |  |

---

```
class cogdl.models.BaseModel
```

Bases: `torch.nn.Module`

```
static add_args (parser)
```

Add model-specific arguments to the parser.

```
abstract classmethod build_model_from_args (cls, args)
```

Build a new model instance.

```
_forward_unimplemented (self, *input: Any) → None
```

**static** `get_trainer` (*taskType: Any, args: Any*) → Optional[Type[BaseTrainer]]

`cogdl.models.pyg = False`

`cogdl.models.dgl_import = False`

`cogdl.models.MODEL_REGISTRY`

`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.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.models.alias_draw` (*J, q*)

Draw sample from a non-uniform discrete distribution using alias sampling.

`cogdl.models.model_name`

`cogdl.models.build_model` (*args*)

**cogdl.tasks**

**Submodules**

`cogdl.tasks.base_task`

**Module Contents**

**Classes**

---

*BaseTask*

---

**class** `cogdl.tasks.base_task.BaseTask` (*args*)

Bases: `object`

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

Add task-specific arguments to the parser.

**abstract** `train` (*self, num\_epoch*)

`cogdl.tasks.graph_classification`

## Module Contents

## Classes

---

|                                  |                                       |
|----------------------------------|---------------------------------------|
| <code>GraphClassification</code> | Supervised graph classification task. |
|----------------------------------|---------------------------------------|

---

## Functions

---

|   |   |
|---|---|
| <code>node_degree_as_feature(data)</code> | Set each node feature as one-hot encoding of degree |
| <code>uniform_node_feature(data)</code>   | Set each node feature to the same                   |

---

`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

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

Bases: `cogdl.tasks.BaseTask`

Supervised graph classification task.

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

**train** (*self*)

**\_train** (*self*)

**\_train\_step** (*self*)

**\_test\_step** (*self*, *split='val'*)

**\_kfold\_train** (*self*)

**generate\_data** (*self*, *dataset*, *args*)

`cogdl.tasks.heterogeneous_node_classification`

## Module Contents

## Classes

---

|  |   |
|--|---|
| <code>HeterogeneousNodeClassification</code> | Heterogeneous Node classification task. |
|--|---|

---

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

Bases: `cogdl.tasks.BaseTask`

Heterogeneous Node classification task.

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

train (self)
_train_step (self)
_test_step (self, split='val')
```

`cogdl.tasks.link_prediction`

### Module Contents

#### Classes

---

*HomoLinkPrediction*

---

*TripleLinkPrediction*

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

---

*KGLinkPrediction*

---

*LinkPrediction*

---

#### Functions

---

*save\_model*(*model*, *optimizer*, *save\_variable\_list*, *args*) Save the parameters of the model and the optimizer,

---

*set\_logger*(*args*) Write logs to checkpoint and console

---

*log\_metrics*(*mode*, *step*, *metrics*) Print the evaluation logs

---

*divide\_data*(*input\_list*, *division\_rate*)

---

*randomly\_choose\_false\_edges*(*nodes*,  
*true\_edges*, *num*)

---

*gen\_node\_pairs*(*train\_data*, *test\_data*, *nega-*  
*tive\_ratio=5*)

---

*get\_score*(*embs*, *node1*, *node2*)

---

*evaluate*(*embs*, *true\_edges*, *false\_edges*)

---

*select\_task*(*model\_name=None*, *model=None*)

---

`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.set_logger` (*args*)

Write logs to checkpoint and console

```
cogdl.tasks.link_prediction.log_metrics (mode, step, metrics)
```

Print the evaluation logs

```
cogdl.tasks.link_prediction.divide_data (input_list, division_rate)
```

```
cogdl.tasks.link_prediction.randomly_choose_false_edges (nodes, true_edges, num)
```

```
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.evaluate (embs, true_edges, false_edges)
```

```
cogdl.tasks.link_prediction.select_task (model_name=None, model=None)
```

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

Bases: torch.nn.Module

```
train (self)
```

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

Bases: torch.nn.Module

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

```
train (self)
```

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

Bases: torch.nn.Module

```
train (self)
```

```
_train_step (self, split='train')
```

```
_test_step (self, split='val')
```

```
class cogdl.tasks.link_prediction.LinkPrediction (args, dataset=None, model=None)
Bases: cogdl.tasks.BaseTask
```

```
static add_args (parser)
```

```
train (self)
```

```
cogdl.tasks.multiplex_link_prediction
```

## Module Contents

### Classes

---

*MultiplexLinkPrediction*

---

### Functions

---

`get_score(embs, node1, node2)`

---

`evaluate(embs, true_edges, false_edges)`

---

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

`cogdl.tasks.multiplex_link_prediction.evaluate(embs, true_edges, false_edges)`

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

Bases: `cogdl.tasks.BaseTask`

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

**train** (*self*)

`cogdl.tasks.multiplex_node_classification`

### Module Contents

#### Classes

---

`MultiplexNodeClassification` Node classification task.

---

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

Bases: `cogdl.tasks.BaseTask`

Node classification task.

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

**train** (*self*)

`cogdl.tasks.node_classification`

### Module Contents

#### Classes

---

`NodeClassification` Node classification task.

---



---

```
class cogdl.tasks.node_classification.NodeClassification (args, dataset=None,
                                                         model: Optional[SupervisedHomogeneousNodeClassification]
                                                         = None)
```

Bases: *cogdl.tasks.BaseTask*

Node classification task.

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

train (self)
_train_step (self)
_test_step (self, split='val', logits=None)
```

**cogdl.tasks.node\_classification\_sampling**

## Module Contents

### Classes

---

|                                   |   |
|-----------------------------------|---|
| <i>NodeClassificationSampling</i> | Node classification task with sampling. |
|-----------------------------------|---|

---

### Functions

---

|   |               |
|---|---------------|
| <i>get_batches</i> (train_nodes, batch_size=64, shuffle=True) | train_labels, |
|---|---------------|

---

```
cogdl.tasks.node_classification_sampling.get_batches (train_nodes, train_labels,
                                                         batch_size=64, shuffle=True)
```

```
class cogdl.tasks.node_classification_sampling.NodeClassificationSampling (args,
                                                                              dataset=None,
                                                                              model=None)
```

Bases: *cogdl.tasks.BaseTask*

Node classification task with sampling.

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

train (self)
_train_step (self)
_test_step (self, split='val')
```

`cogdl.tasks.pretrain`

### Module Contents

#### Classes

---

*PretrainTask*

---

**class** `cogdl.tasks.pretrain.PretrainTask` (*args*)

Bases: `cogdl.tasks.BaseTask`

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

Add task-specific arguments to the parser.

**train** (*self*)

`cogdl.tasks.unsupervised_graph_classification`

### Module Contents

#### Classes

---

*UnsupervisedGraphClassification*

Unsupervised graph classification

---

**class** `cogdl.tasks.unsupervised_graph_classification.UnsupervisedGraphClassification` (*args*,  
*dataset=*

*model=N*

Bases: `cogdl.tasks.BaseTask`

Unsupervised graph classification

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

Add task-specific arguments to the parser.

**train** (*self*)

**save\_emb** (*self*, *embs*)

**\_evaluate** (*self*, *embeddings*, *labels*)

`cogdl.tasks.unsupervised_node_classification`

### Module Contents

#### Classes

---

*UnsupervisedNodeClassification*

Node classification task.

---

*TopKRanker*

---

```
cogdl.tasks.unsupervised_node_classification.pyg = False
```

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

Bases: `cogdl.tasks.BaseTask`

Node classification task.

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

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

```
save_emb (self, embs)
```

```
train (self)
```

```
_evaluate (self, features_matrix, label_matrix, num_shuffle)
```

```
class cogdl.tasks.unsupervised_node_classification.TopKRanker
```

Bases: `sklearn.multiclass.OneVsRestClassifier`

```
predict (self, X, top_k_list)
```

## Package Contents

### Classes

---

`BaseTask`

---

### Functions

---

|                                  |  |
|----------------------------------|--|
| <code>register_task(name)</code> | New task types can be added to cogdl with the <code>register_task()</code> |
|----------------------------------|--|

---

|   |
|---|
| <code>build_task(args, dataset=None, model=None)</code> |
|---|

---

```
class cogdl.tasks.BaseTask(args)
```

Bases: `object`

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

```
abstract train (self, num_epoch)
```

```
cogdl.tasks.TASK_REGISTRY
```

```
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.task_name`

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

`cogdl.trainers`

### Submodules

`cogdl.trainers.base_trainer`

### Module Contents

#### Classes

---

|                    |  |
|--------------------|--|
| <i>BaseTrainer</i> | Helper class that provides a standard way to create an ABC using |
|--------------------|--|

---

**class** `cogdl.trainers.base_trainer.BaseTrainer`

Bases: `abc.ABC`

Helper class that provides a standard way to create an ABC using inheritance.

**abstract classmethod** `build_trainer_from_args` (*cls*, *args*)

Build a new trainer instance.

`cogdl.trainers.deepergcn_trainer`

### Module Contents

#### Classes

---

|                         |  |
|-------------------------|--|
| <i>DeeperGCNTrainer</i> | Helper class that provides a standard way to create an ABC using |
|-------------------------|--|

---

#### Functions

---

|   |
|---|
| <i>random_partition_graph</i> ( <i>num_nodes</i> , <i>cluster_number=10</i> ) |
|---|

---

|   |
|---|
| <i>generate_subgraphs</i> ( <i>edge_index</i> , <i>parts</i> , <i>cluster_number=10</i> , <i>batch_size=1</i> ) |
|---|

---

`cogdl.trainers.deepergcn_trainer.random_partition_graph` (*num\_nodes*, *cluster\_number=10*)

`cogdl.trainers.deepergcn_trainer.generate_subgraphs` (*edge\_index*, *parts*, *cluster\_number=10*, *batch\_size=1*)

---

```

class cogdl.trainers.deepergcn_trainer.DeeperGCNTrainer (args)
    Bases: cogdl.trainers.base_trainer.BaseTrainer

    Helper class that provides a standard way to create an ABC using inheritance.

    fit (self, model, data)

    test_gpu_volume (self)

    _train_step (self)

    _test_step (self, split='val')

    loss (self, data)

    predict (self, data)

    classmethod build_trainer_from_args (cls, args)
        Build a new trainer instance.

```

```
cogdl.trainers.gpt_gnn_trainer
```

## Module Contents

### Classes

---

```
GPT_GNNHomogeneousTrainer
```

---

```
GPT_GNNHeterogeneousTrainer
```

---

### Functions

---

|   |      |   |
|---|------|---|
| <i>node_classification_sample</i> ( <i>args, target_type, seed, nodes, time_range</i> )               | tar- | sub-graph sampling and label preparation for node classification:                   |
| <i>prepare_data</i> ( <i>args, graph, target_type, train_target_nodes, valid_target_nodes, pool</i> ) |      | Sampled and prepare training and validation data using multi-process parallization. |

---

```
cogdl.trainers.gpt_gnn_trainer.graph_pool
```

```
cogdl.trainers.gpt_gnn_trainer.node_classification_sample (args, target_type, seed, nodes, time_range)
    sub-graph sampling and label preparation for node classification: (1) Sample batch_size number of output nodes (papers) and their time.
```

```
cogdl.trainers.gpt_gnn_trainer.prepare_data (args, graph, target_type, train_target_nodes, valid_target_nodes, pool)
    Sampled and prepare training and validation data using multi-process parallization.
```

```

class cogdl.trainers.gpt_gnn_trainer.GPT_GNNHomogeneousTrainer (args)
    Bases: cogdl.trainers.supervised_trainer.SupervisedHomogeneousNodeClassificationTrainer

    fit (self, model: cogdl.models.supervised_model.SupervisedHeterogeneousNodeClassificationModel, dataset: cogdl.data.Dataset) → None

```

**classmethod** `build_trainer_from_args` (*cls, args*)

**class** `cogdl.trainers.gpt_gnn_trainer.GPT_GNNHeterogeneousTrainer` (*model, dataset*)

Bases: `cogdl.trainers.supervised_trainer.SupervisedHeterogeneousNodeClassificationTrainer`

**fit** (*self*) → None

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

`cogdl.trainers.sampled_trainer`

### Module Contents

#### Classes

---

`SampledTrainer`

---

`SAINTTrainer`

---

**class** `cogdl.trainers.sampled_trainer.SampledTrainer`

Bases: `cogdl.trainers.supervised_trainer.SupervisedHeterogeneousNodeClassificationTrainer`

**abstract fit** (*self, model: cogdl.models.supervised\_model.SupervisedHeterogeneousNodeClassificationModel, dataset: cogdl.data.Dataset*)

**class** `cogdl.trainers.sampled_trainer.SAINTTrainer` (*args*)

Bases: `cogdl.trainers.sampled_trainer.SampledTrainer`

**static build\_trainer\_from\_args** (*args*)

**sampler\_from\_args** (*self, args*)

**fit** (*self, model: cogdl.models.supervised\_model.SupervisedHeterogeneousNodeClassificationModel, dataset: cogdl.data.Dataset*)

**\_train\_step** (*self*)

**\_test\_step** (*self, split='val'*)

`cogdl.trainers.supervised_trainer`

### Module Contents

#### Classes

---

`SupervisedTrainer`

Helper class that provides a standard way to create an ABC using

---

`SupervisedHeterogeneousNodeClassificationTrainer`

Helper class that provides a standard way to create an ABC using

---

continues on next page

Table 140 – continued from previous page

---

*SupervisedHomogeneousNodeClassificationTrainer* Helper class that provides a standard way to create an ABC using

---

**class** `cogdl.trainers.supervised_trainer.SupervisedTrainer`

Bases: `cogdl.trainers.base_trainer.BaseTrainer`, `abc.ABC`

Helper class that provides a standard way to create an ABC using inheritance.

**abstract fit** (*self*) → None

**abstract predict** (*self*) → Any

**class** `cogdl.trainers.supervised_trainer.SupervisedHeterogeneousNodeClassificationTrainer`

Bases: `cogdl.trainers.base_trainer.BaseTrainer`, `abc.ABC`

Helper class that provides a standard way to create an ABC using inheritance.

**abstract fit** (*self*, *model*: `cogdl.models.supervised_model.SupervisedHeterogeneousNodeClassificationModel`,  
*dataset*: `cogdl.data.Dataset`) → None

**class** `cogdl.trainers.supervised_trainer.SupervisedHomogeneousNodeClassificationTrainer`

Bases: `cogdl.trainers.base_trainer.BaseTrainer`, `abc.ABC`

Helper class that provides a standard way to create an ABC using inheritance.

**abstract fit** (*self*, *model*: `cogdl.models.supervised_model.SupervisedHomogeneousNodeClassificationModel`,  
*dataset*: `cogdl.data.Dataset`) → None

`cogdl.trainers.unsupervised_trainer`

## Module Contents

### Classes

---

*UnsupervisedTrainer*

---

**class** `cogdl.trainers.unsupervised_trainer.UnsupervisedTrainer`

Bases: `cogdl.trainers.base_trainer.BaseTrainer`

**abstract get\_embedding** (*self*)

## 6.1.2 Submodules

`cogdl.options`

### Module Contents

### Functions

---

`get_parser()`

---

`add_task_args(parser)`

---

`add_dataset_args(parser)`

---

`add_model_args(parser)`

---

`get_training_parser()`

---

`get_display_data_parser()`

---

`get_download_data_parser()`

---

`parse_args_and_arch(parser, args)`

The parser doesn't know about model-specific args, so we parse twice.

---

`cogdl.options.get_parser()`

`cogdl.options.add_task_args(parser)`

`cogdl.options.add_dataset_args(parser)`

`cogdl.options.add_model_args(parser)`

`cogdl.options.get_training_parser()`

`cogdl.options.get_display_data_parser()`

`cogdl.options.get_download_data_parser()`

`cogdl.options.parse_args_and_arch(parser, args)`

The parser doesn't know about model-specific args, so we parse twice.

### `cogdl.utils`

## Module Contents

### Classes

---

`ArgClass`

---

### Functions

---

`build_args_from_dict(dic)`

---

`add_self_loops(edge_index, edge_weight=None, fill_value=1, num_nodes=None)`

---

`add_remaining_self_loops(edge_index, edge_weight=None, fill_value=1, num_nodes=None)`

---

continues on next page



Table 144 – continued from previous page

|   |       |
|---|-------|
| <code>row_normalization(num_nodes, edge_index, edge_weight=None)</code>       |       |
| <code>symmetric_normalization(num_nodes, edge_index, edge_weight=None)</code> |       |
| <code>sppmm(indices, values, b)</code>  | Args: |
| <code>sppmm_adj(indices, values, shape, b)</code>                             |       |
| <code>get_degrees(indices, num_nodes=None)</code>                             |       |
| <code>edge_softmax(indices, values, shape)</code>                             | Args: |
| <code>mul_edge_softmax(indices, values, shape)</code>                         | Args: |
| <code>remove_self_loops(indices)</code>                                       |       |
| <code>get_activation(act)</code>  |       |
| <code>cycle_index(num, shift)</code>  |       |
| <code>batch_sum_pooling(x, batch)</code>                                      |       |
| <code>batch_mean_pooling(x, batch)</code>                                     |       |
| <code>tabulate_results(results_dict)</code>                                   |       |
| <code>print_result(results, datasets, model_name)</code>                      |       |
| <code>set_random_seed(seed)</code>  |       |

**class cogdl.utils.ArgClass**

Bases: object

`cogdl.utils.build_args_from_dict(dic)``cogdl.utils.add_self_loops(edge_index, edge_weight=None, fill_value=1, num_nodes=None)``cogdl.utils.add_remaining_self_loops(edge_index, edge_weight=None, fill_value=1, num_nodes=None)``cogdl.utils.row_normalization(num_nodes, edge_index, edge_weight=None)``cogdl.utils.symmetric_normalization(num_nodes, edge_index, edge_weight=None)``cogdl.utils.sppmm(indices, values, b)`

Args: indices : Tensor, shape=(2, E) values : Tensor, shape=(E,) shape : tuple(int, int) b : Tensor, shape=(N, )

`cogdl.utils.sppmm_adj(indices, values, shape, b)``cogdl.utils.get_degrees(indices, num_nodes=None)``cogdl.utils.edge_softmax(indices, values, shape)`

Args: indices: Tensor, shape=(2, E) values: Tensor, shape=(N,) shape: tuple(int, int)

Returns: Softmax values of edge values for nodes

`cogdl.utils.mul_edge_softmax(indices, values, shape)`

Args: indices: Tensor, shape=(2, E) values: Tensor, shape=(E, d) shape: tuple(int, int)

Returns: Softmax values of multi-dimension edge values for nodes

`cogdl.utils.remove_self_loops` (*indices*)  
`cogdl.utils.get_activation` (*act*)  
`cogdl.utils.cycle_index` (*num, shift*)  
`cogdl.utils.batch_sum_pooling` (*x, batch*)  
`cogdl.utils.batch_mean_pooling` (*x, batch*)  
`cogdl.utils.tabulate_results` (*results\_dict*)  
`cogdl.utils.print_result` (*results, datasets, model\_name*)  
`cogdl.utils.set_random_seed` (*seed*)  
`cogdl.utils.args`

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### C

- cogdl, 23
- cogdl.data, 23
  - cogdl.data.batch, 23
  - cogdl.data.data, 24
  - cogdl.data.dataloader, 26
  - cogdl.data.dataset, 27
  - cogdl.data.download, 28
  - cogdl.data.extract, 28
  - cogdl.data.makedirs, 29
  - cogdl.data.sampler, 29
- cogdl.datasets, 35
  - cogdl.datasets.dgl\_data, 35
  - cogdl.datasets.gatne, 36
  - cogdl.datasets.gcc\_data, 37
  - cogdl.datasets.gtn\_data, 38
  - cogdl.datasets.han\_data, 40
  - cogdl.datasets.kg\_data, 41
  - cogdl.datasets.matlab\_matrix, 45
  - cogdl.datasets.pyg, 46
  - cogdl.datasets.pyg\_ogb, 48
  - cogdl.datasets.pyg\_strategies\_data, 50
- cogdl.layers, 57
  - cogdl.layers.gcc\_module, 57
  - cogdl.layers.gpt\_gnn\_module, 59
  - cogdl.layers.link\_prediction\_module, 63
  - cogdl.layers.maggregator, 64
  - cogdl.layers.mixhop\_layer, 65
  - cogdl.layers.prone\_module, 65
  - cogdl.layers.se\_layer, 67
  - cogdl.layers.srgcn\_module, 67
  - cogdl.layers.strategies\_layers, 69
- cogdl.models, 71
  - cogdl.models.base\_model, 118
  - cogdl.models.emb, 71
    - cogdl.models.emb.complex, 71
    - cogdl.models.emb.deepwalk, 72
    - cogdl.models.emb.dgk, 73
    - cogdl.models.emb.distmult, 73
    - cogdl.models.emb.dngr, 74
    - cogdl.models.emb.gatne, 74
    - cogdl.models.emb.graph2vec, 76
    - cogdl.models.emb.grarep, 77
    - cogdl.models.emb.hin2vec, 77
    - cogdl.models.emb.hope, 78
    - cogdl.models.emb.knowledge\_base, 79
    - cogdl.models.emb.line, 79
    - cogdl.models.emb.metapath2vec, 80
    - cogdl.models.emb.netmf, 81
    - cogdl.models.emb.netsmf, 81
    - cogdl.models.emb.node2vec, 82
    - cogdl.models.emb.prone, 83
    - cogdl.models.emb.pte, 83
    - cogdl.models.emb.rotate, 84
    - cogdl.models.emb.sdne, 84
    - cogdl.models.emb.spectral, 85
    - cogdl.models.emb.transe, 85
  - cogdl.models.nn, 86
    - cogdl.models.nn.asgcn, 86
    - cogdl.models.nn.compgcn, 87
    - cogdl.models.nn.dgi, 88
    - cogdl.models.nn.dgl\_gcc, 90
    - cogdl.models.nn.disengcn, 91
    - cogdl.models.nn.fastgcn, 92
    - cogdl.models.nn.gat, 93
    - cogdl.models.nn.gcn, 94
    - cogdl.models.nn.gcnii, 94
    - cogdl.models.nn.graphsage, 95
    - cogdl.models.nn.mixhop, 96
    - cogdl.models.nn.mlp, 96
    - cogdl.models.nn.mvgrl, 97
    - cogdl.models.nn.patchy\_san, 98
    - cogdl.models.nn.pyg\_cheb, 99
    - cogdl.models.nn.pyg\_deepergcn, 99
    - cogdl.models.nn.pyg\_dgcnn, 100
    - cogdl.models.nn.pyg\_diffpool, 101
    - cogdl.models.nn.pyg\_drgat, 103
    - cogdl.models.nn.pyg\_drgcn, 104
    - cogdl.models.nn.pyg\_gat, 104
    - cogdl.models.nn.pyg\_gcn, 105
    - cogdl.models.nn.pyg\_gcnmix, 105
    - cogdl.models.nn.pyg\_gin, 107
    - cogdl.models.nn.pyg\_gpt\_gnn, 108
    - cogdl.models.nn.pyg\_grand, 109

- `cogdl.models.nn.pyg_gtn`, 110
- `cogdl.models.nn.pyg_han`, 110
- `cogdl.models.nn.pyg_infograph`, 111
- `cogdl.models.nn.pyg_infomax`, 113
- `cogdl.models.nn.pyg_sortpool`, 113
- `cogdl.models.nn.pyg_srgcn`, 114
- `cogdl.models.nn.pyg_stpgnn`, 115
- `cogdl.models.nn.pyg_unet`, 116
- `cogdl.models.nn.pyg_unsup_graphsage`, 116
- `cogdl.models.nn.rgcn`, 117
- `cogdl.models.supervised_model`, 118
- `cogdl.options`, 131
- `cogdl.tasks`, 120
  - `base_task`, 120
  - `graph_classification`, 121
  - `heterogeneous_node_classification`, 121
  - `link_prediction`, 122
  - `multiplex_link_prediction`, 123
  - `multiplex_node_classification`, 124
  - `node_classification`, 124
  - `node_classification_sampling`, 125
  - `pretrain`, 126
  - `unsupervised_graph_classification`, 126
  - `unsupervised_node_classification`, 126
- `cogdl.trainers`, 128
  - `base_trainer`, 128
  - `deepergcn_trainer`, 128
  - `gpt_gnn_trainer`, 129
  - `sampled_trainer`, 130
  - `supervised_trainer`, 130
  - `unsupervised_trainer`, 131
- `cogdl.utils`, 132

## Symbols

`__call__()` (*cogdl.data.Data method*), 31  
`__call__()` (*cogdl.data.data.Data method*), 25  
`__call__()` (*cogdl.datasets.pyg\_strategies\_data.ChemExtractSubstructureContextPair method*), 52  
`__call__()` (*cogdl.datasets.pyg\_strategies\_data.ExtractSubstructureContextPair method*), 52  
`__call__()` (*cogdl.datasets.pyg\_strategies\_data.MaskAtom method*), 51  
`__call__()` (*cogdl.datasets.pyg\_strategies\_data.MaskEdge method*), 51  
`__call__()` (*cogdl.datasets.pyg\_strategies\_data.NegativeEdge method*), 51  
`__call__()` (*cogdl.layers.prone\_module.ProNE method*), 66  
`__contains__()` (*cogdl.data.Data method*), 31  
`__contains__()` (*cogdl.data.data.Data method*), 24  
`__getitem__()` (*cogdl.data.Data method*), 31  
`__getitem__()` (*cogdl.data.Dataset method*), 34  
`__getitem__()` (*cogdl.data.data.Data method*), 24  
`__getitem__()` (*cogdl.data.dataset.Dataset method*), 28  
`__getitem__()` (*cogdl.datasets.Dataset method*), 56  
`__getitem__()` (*cogdl.datasets.kg\_data.TestDataset method*), 42  
`__getitem__()` (*cogdl.datasets.kg\_data.TrainDataset method*), 42  
`__getitem__()` (*cogdl.datasets.pyg.ENZYMES method*), 48  
`__getitem__()` (*cogdl.models.nn.dgl\_gcc.NodeClassificationDataset method*), 91  
`__inc__()` (*cogdl.data.Data method*), 32  
`__inc__()` (*cogdl.data.data.Data method*), 25  
`__iter__()` (*cogdl.data.Data method*), 31  
`__iter__()` (*cogdl.data.data.Data method*), 24  
`__len__()` (*cogdl.data.Data method*), 31  
`__len__()` (*cogdl.data.Dataset method*), 33  
`__len__()` (*cogdl.data.data.Data method*), 24  
`__len__()` (*cogdl.data.dataset.Dataset method*), 27  
`__len__()` (*cogdl.datasets.Dataset method*), 56  
`__len__()` (*cogdl.datasets.kg\_data.TestDataset method*), 42  
`__len__()` (*cogdl.datasets.kg\_data.TrainDataset method*), 42  
`__len__()` (*cogdl.models.nn.dgl\_gcc.NodeClassificationDataset method*), 90  
`__next__()` (*cogdl.datasets.kg\_data.BidirectionalOneShotIterator method*), 42  
`__repr__()` (*cogdl.data.Data method*), 32  
`__repr__()` (*cogdl.data.Dataset method*), 34  
`__repr__()` (*cogdl.data.data.Data method*), 25  
`__repr__()` (*cogdl.data.dataset.Dataset method*), 28  
`__repr__()` (*cogdl.datasets.Dataset method*), 56  
`__repr__()` (*cogdl.datasets.gatne.GatneDataset method*), 36  
`__repr__()` (*cogdl.datasets.gtn\_data.GTNDataset method*), 39  
`__repr__()` (*cogdl.datasets.han\_data.HANDataset method*), 41  
`__repr__()` (*cogdl.datasets.pyg\_strategies\_data.ChemExtractSubstructureContextPair method*), 52  
`__repr__()` (*cogdl.datasets.pyg\_strategies\_data.ExtractSubstructureContextPair method*), 52  
`__repr__()` (*cogdl.datasets.pyg\_strategies\_data.MaskAtom method*), 52  
`__repr__()` (*cogdl.layers.MeanAggregator method*), 71  
`__repr__()` (*cogdl.layers.gpt\_gnn\_module.Classifier method*), 62  
`__repr__()` (*cogdl.layers.gpt\_gnn\_module.HGTConv method*), 61  
`__repr__()` (*cogdl.layers.gpt\_gnn\_module.Matcher method*), 62  
`__repr__()` (*cogdl.layers.magggregator.MeanAggregator method*), 64  
`__repr__()` (*cogdl.models.nn.asgcn.GraphConvolution method*), 86  
`__repr__()` (*cogdl.models.nn.fastgcn.GraphConvolution method*), 92  
`__repr__()` (*cogdl.models.nn.gat.SpGraphAttentionLayer method*), 93  
`__repr__()` (*cogdl.models.nn.gcn.GraphConvolution method*), 94  
`__repr__()` (*cogdl.models.nn.pyg\_grand.MLPLayer method*), 94

*method*), 109  
 \_\_getitem\_\_ () (*cogdl.data.Data method*), 31  
 \_\_getitem\_\_ () (*cogdl.data.data.Data method*), 24  
 \_add\_undirected\_graph\_positional\_embeddings () (*in module cogdl.models.nn.dgl\_gcc*), 90  
 \_approximate\_deepwalk\_matrix () (*cogdl.models.emb.netmf.NetMF method*), 81  
 \_approximate\_normalized\_laplacian () (*cogdl.models.emb.netmf.NetMF method*), 81  
 \_chebyshev\_gaussian () (*cogdl.models.emb.prone.ProNE method*), 83  
 \_compute\_deepwalk\_matrix () (*cogdl.models.emb.netmf.NetMF method*), 81  
 \_convert\_idx () (*cogdl.models.nn.dgl\_gcc.GraphClassificationDataset method*), 91  
 \_convert\_idx () (*cogdl.models.nn.dgl\_gcc.NodeClassificationDataset method*), 90  
 \_create\_dgl\_graph () (*cogdl.models.nn.dgl\_gcc.NodeClassificationDataset method*), 90  
 \_deepwalk\_filter () (*cogdl.models.emb.netmf.NetMF method*), 81  
 \_download () (*cogdl.data.Dataset method*), 34  
 \_download () (*cogdl.data.dataset.Dataset method*), 28  
 \_download () (*cogdl.datasets.Dataset method*), 56  
 \_evaluate () (*cogdl.tasks.unsupervised\_graph\_classification.UnsupervisedGraphClassification method*), 126  
 \_evaluate () (*cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification method*), 127  
 \_forward\_unimplemented () (*cogdl.models.BaseModel method*), 119  
 \_forward\_unimplemented () (*cogdl.models.base\_model.BaseModel method*), 118  
 \_generate\_adj () (*cogdl.models.nn.fastgcn.FastGCN method*), 92  
 \_get\_alias\_edge () (*cogdl.models.emb.node2vec.Node2vec method*), 82  
 \_get\_embedding () (*cogdl.models.emb.grarep.GraRep method*), 77  
 \_get\_embedding () (*cogdl.models.emb.hope.HOPE method*), 78  
 \_get\_embedding\_dense () (*cogdl.models.emb.prone.ProNE method*), 83  
 \_get\_embedding\_rand () (*cogdl.models.emb.netmf.NetSMF method*), 82  
 \_get\_embedding\_rand () (*cogdl.models.emb.prone.ProNE method*), 83  
 \_kfold\_train () (*cogdl.tasks.graph\_classification.GraphClassification method*), 121  
 \_loss () (*cogdl.layers.link\_prediction\_module.GNNLinkPredict method*), 64  
 \_node2vec\_walk () (*cogdl.models.emb.node2vec.Node2vec method*), 82  
 \_path\_sampling () (*cogdl.models.emb.netmf.NetSMF method*), 82  
 \_pre\_factorization () (*cogdl.models.emb.prone.ProNE method*), 83  
 \_preprocess\_transition\_probs () (*cogdl.models.emb.node2vec.Node2vec method*), 82  
 \_process () (*cogdl.data.Dataset method*), 34  
 \_process () (*cogdl.data.dataset.Dataset method*), 28  
 \_process () (*cogdl.datasets.Dataset method*), 56  
 \_regularization () (*cogdl.layers.link\_prediction\_module.GNNLinkPredict method*), 64  
 \_rwr\_trace\_to\_dgl\_graph () (*in module cogdl.models.nn.dgl\_gcc*), 90  
 \_sample\_one\_layer () (*cogdl.models.nn.asgcn.ASGCN method*), 86  
 \_sample\_one\_layer () (*cogdl.models.nn.fastgcn.FastGCN method*), 92  
 \_simulate\_walks () (*cogdl.models.emb.deepwalk.DeepWalk method*), 72  
 \_simulate\_walks () (*cogdl.models.emb.hin2vec.RWgraph method*), 77  
 \_simulate\_walks () (*cogdl.models.emb.metapath2vec.Metapath2vec method*), 80  
 \_simulate\_walks () (*cogdl.models.emb.node2vec.Node2vec method*), 82  
 \_test\_step () (*cogdl.layers.strategies\_layers.Finetuner method*), 70  
 \_test\_step () (*cogdl.tasks.graph\_classification.GraphClassification method*), 121  
 \_test\_step () (*cogdl.tasks.heterogeneous\_node\_classification.HeterogeneousNodeClassification method*), 122  
 \_test\_step () (*cogdl.tasks.link\_prediction.KGLinkPrediction method*), 123  
 \_test\_step () (*cogdl.tasks.node\_classification.NodeClassification method*), 125  
 \_test\_step () (*cogdl.tasks.node\_classification\_sampling.NodeClassificationSampling method*), 125



*method*), 125  
 \_test\_step() (*cogdl.trainers.deepergcn\_trainer.DeeperGCNTrainer* *static method*), 70  
*method*), 129  
 \_test\_step() (*cogdl.trainers.sampled\_trainer.SAINTTrainer* *static method*), 70  
*method*), 130  
 \_train() (*cogdl.tasks.graph\_classification.GraphClassification* *static method*), 70  
*method*), 121  
 \_train\_line() (*cogdl.models.emb.line.LINE* *static method*), 70  
*method*), 80  
 \_train\_line() (*cogdl.models.emb.pte.PTE* *method*), 84  
 \_train\_step() (*cogdl.layers.strategies\_layers.ContextPredictTrainer* *static method*), 118  
*method*), 70  
 \_train\_step() (*cogdl.layers.strategies\_layers.Finetuner* *static method*), 119  
*method*), 70  
 \_train\_step() (*cogdl.layers.strategies\_layers.InfoMaxTrainer* *static method*), 72  
*method*), 70  
 \_train\_step() (*cogdl.layers.strategies\_layers.MaskTrainer* *static method*), 73  
*method*), 70  
 \_train\_step() (*cogdl.layers.strategies\_layers.SupervisedTrainer* *method*), 74  
*method*), 70  
 \_train\_step() (*cogdl.tasks.graph\_classification.GraphClassification* *static method*), 75  
*method*), 121  
 \_train\_step() (*cogdl.tasks.heterogeneous\_node\_classification.HeterogeneousNodeClassification* *static method*), 76  
*method*), 122  
 \_train\_step() (*cogdl.tasks.link\_prediction.KGLinkPrediction* *method*), 77  
*method*), 123  
 \_train\_step() (*cogdl.tasks.node\_classification.NodeClassification* *static method*), 78  
*method*), 125  
 \_train\_step() (*cogdl.tasks.node\_classification\_sampling.NodeClassificationSampling* *static method*), 79  
*method*), 125  
 \_train\_step() (*cogdl.trainers.deepergcn\_trainer.DeeperGCNTrainer* *static method*), 79  
*method*), 129  
 \_train\_step() (*cogdl.trainers.sampled\_trainer.SAINTTrainer* *method*), 80  
*method*), 130  
 \_update() (*cogdl.models.emb.line.LINE* *method*), 80  
 \_update() (*cogdl.models.emb.pte.PTE* *method*), 84  
 \_walk() (*cogdl.models.emb.deepwalk.DeepWalk* *method*), 72  
 \_walk() (*cogdl.models.emb.hin2vec.RWgraph* *method*), 77  
 \_walk() (*cogdl.models.emb.metapath2vec.Metapath2vec* *method*), 80  
  
**A**  
 ACM\_GTNDataset (*class in cogdl.datasets.gtn\_data*), 39  
 ACM\_HANDataset (*class in cogdl.datasets.han\_data*), 41  
 act\_attention() (*in module cogdl.layers.srgcn\_module*), 68  
 act\_map() (*in module cogdl.layers.srgcn\_module*), 68  
 act\_normalization() (*in module cogdl.layers.srgcn\_module*), 68  
 add\_args() (*cogdl.layers.strategies\_layers.ContextPredictTrainer* *static method*), 70  
 add\_args() (*cogdl.layers.strategies\_layers.Finetuner* *static method*), 70  
 add\_args() (*cogdl.layers.strategies\_layers.InfoMaxTrainer* *static method*), 70  
 add\_args() (*cogdl.layers.strategies\_layers.MaskTrainer* *static method*), 70  
 add\_args() (*cogdl.layers.strategies\_layers.SupervisedTrainer* *static method*), 70  
 add\_args() (*cogdl.models.base\_model.BaseModel* *static method*), 118  
 add\_args() (*cogdl.models.BaseModel* *static method*), 119  
 add\_args() (*cogdl.models.emb.deepwalk.DeepWalk* *method*), 72  
 add\_args() (*cogdl.models.emb.dgk.DeepGraphKernel* *method*), 73  
 add\_args() (*cogdl.models.emb.dngr.DNGR* *static method*), 74  
 add\_args() (*cogdl.models.emb.gatne.GATNE* *static method*), 75  
 add\_args() (*cogdl.models.emb.graph2vec.Graph2Vec* *static method*), 76  
 add\_args() (*cogdl.models.emb.grarep.GraRep* *static method*), 77  
 add\_args() (*cogdl.models.emb.hin2vec.Hin2vec* *static method*), 78  
 add\_args() (*cogdl.models.emb.hope.HOPE* *static method*), 79  
 add\_args() (*cogdl.models.emb.knowledge\_base.KGEModel* *static method*), 80  
 add\_args() (*cogdl.models.emb.line.LINE* *static method*), 80  
 add\_args() (*cogdl.models.emb.metapath2vec.Metapath2vec* *static method*), 80  
 add\_args() (*cogdl.models.emb.netmf.NetMF* *static method*), 81  
 add\_args() (*cogdl.models.emb.netsmf.NetSMF* *static method*), 82  
 add\_args() (*cogdl.models.emb.node2vec.Node2vec* *static method*), 82  
 add\_args() (*cogdl.models.emb.prone.ProNE* *static method*), 83  
 add\_args() (*cogdl.models.emb.pte.PTE* *static method*), 83  
 add\_args() (*cogdl.models.emb.sдне.SDNE* *static method*), 85  
 add\_args() (*cogdl.models.emb.spectral.Spectral* *static method*), 85  
 add\_args() (*cogdl.models.nn.asgcn.ASGCN* *static method*), 86  
 add\_args() (*cogdl.models.nn.compvcn.LinkPredictCompGCN* *static method*), 88

- `add_args()` (*cogdl.models.nn.dgi.DGI static method*), 89
- `add_args()` (*cogdl.models.nn.dgl\_gcc.GCC static method*), 91
- `add_args()` (*cogdl.models.nn.disengcn.DisenGCN static method*), 91
- `add_args()` (*cogdl.models.nn.fastgcn.FastGCN static method*), 92
- `add_args()` (*cogdl.models.nn.gat.PetarVSpGAT static method*), 93
- `add_args()` (*cogdl.models.nn.gcn.TKipfGCN static method*), 94
- `add_args()` (*cogdl.models.nn.gcnii.GCNII static method*), 95
- `add_args()` (*cogdl.models.nn.graphsage.Graphsage static method*), 95
- `add_args()` (*cogdl.models.nn.mixhop.MixHop static method*), 96
- `add_args()` (*cogdl.models.nn.mlp.MLP static method*), 96
- `add_args()` (*cogdl.models.nn.mvgrl.MVGRL static method*), 97
- `add_args()` (*cogdl.models.nn.patchy\_san.PatchySAN static method*), 98
- `add_args()` (*cogdl.models.nn.pyg\_cheb.Chebyshev static method*), 99
- `add_args()` (*cogdl.models.nn.pyg\_deepergcn.DeeperGCN static method*), 100
- `add_args()` (*cogdl.models.nn.pyg\_dgcnn.DGCNN static method*), 100
- `add_args()` (*cogdl.models.nn.pyg\_diffpool.DiffPool static method*), 103
- `add_args()` (*cogdl.models.nn.pyg\_drgat.DrGAT static method*), 104
- `add_args()` (*cogdl.models.nn.pyg\_drgcn.DrGCN static method*), 104
- `add_args()` (*cogdl.models.nn.pyg\_gat.GAT static method*), 104
- `add_args()` (*cogdl.models.nn.pyg\_gcn.GCN static method*), 105
- `add_args()` (*cogdl.models.nn.pyg\_gcnmix.GCNMix static method*), 106
- `add_args()` (*cogdl.models.nn.pyg\_gin.GIN static method*), 108
- `add_args()` (*cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN static method*), 108
- `add_args()` (*cogdl.models.nn.pyg\_grand.Grand static method*), 109
- `add_args()` (*cogdl.models.nn.pyg\_gtn.GTN static method*), 110
- `add_args()` (*cogdl.models.nn.pyg\_han.HAN static method*), 111
- `add_args()` (*cogdl.models.nn.pyg\_infograph.InfoGraph static method*), 112
- `add_args()` (*cogdl.models.nn.pyg\_infomax.Infomax static method*), 113
- `add_args()` (*cogdl.models.nn.pyg\_sortpool.SortPool static method*), 114
- `add_args()` (*cogdl.models.nn.pyg\_srgcn.SRGCN static method*), 115
- `add_args()` (*cogdl.models.nn.pyg\_stpgnn.stpgnn static method*), 115
- `add_args()` (*cogdl.models.nn.pyg\_unet.UNet static method*), 116
- `add_args()` (*cogdl.models.nn.pyg\_unsup\_graphsage.Graphsage static method*), 117
- `add_args()` (*cogdl.models.nn.rgcn.LinkPredictRGCN static method*), 117
- `add_args()` (*cogdl.tasks.base\_task.BaseTask static method*), 120
- `add_args()` (*cogdl.tasks.BaseTask static method*), 127
- `add_args()` (*cogdl.tasks.graph\_classification.GraphClassification static method*), 121
- `add_args()` (*cogdl.tasks.heterogeneous\_node\_classification.HeterogeneousNodeClassification static method*), 122
- `add_args()` (*cogdl.tasks.link\_prediction.LinkPrediction static method*), 123
- `add_args()` (*cogdl.tasks.multiplex\_link\_prediction.MultiplexLinkPrediction static method*), 124
- `add_args()` (*cogdl.tasks.multiplex\_node\_classification.MultiplexNodeClassification static method*), 124
- `add_args()` (*cogdl.tasks.node\_classification.NodeClassification static method*), 125
- `add_args()` (*cogdl.tasks.node\_classification\_sampling.NodeClassificationSampling static method*), 125
- `add_args()` (*cogdl.tasks.pretrain.PretrainTask static method*), 126
- `add_args()` (*cogdl.tasks.unsupervised\_graph\_classification.UnsupervisedGraphClassification static method*), 126
- `add_args()` (*cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification static method*), 127
- `add_dataset_args()` (*in module cogdl.options*), 132
- `add_edge()` (*cogdl.layers.gpt\_gnn\_module.Graph static method*), 61
- `add_model_args()` (*in module cogdl.options*), 132
- `add_node()` (*cogdl.layers.gpt\_gnn\_module.Graph static method*), 61
- `add_remaining_self_loops()` (*in module cogdl.utils*), 133
- `add_reverse_edges()` (*cogdl.models.nn.comp\_gcn.LinkPredictCompGCN static method*), 88
- `add_self_loops()` (*in module cogdl.utils*), 133
- `add_task_args()` (*in module cogdl.options*), 132
- `adj_pow_x()` (*cogdl.layers.mixhop\_layer.MixHopLayer static method*), 65
- `adj_pow_x()` (*cogdl.layers.MixHopLayer static method*), 71

`after_pooling_forward()` (*cogdl.models.nn.pyg\_diffpool.DiffPool method*), 103  
`aggr()` (*cogdl.layers.strategies\_layers.GINConv method*), 69  
`alias_draw()` (*in module cogdl.models*), 120  
`alias_setup()` (*in module cogdl.models*), 120  
`AmazonDataset` (*class in cogdl.datasets.gatne*), 36  
`apply()` (*cogdl.data.Data method*), 32  
`apply()` (*cogdl.data.data.Data method*), 25  
`apply_to_device()` (*cogdl.datasets.gtn\_data.GTNDataset method*), 39  
`apply_to_device()` (*cogdl.datasets.han\_data.HANDataset method*), 40  
`ApplyNodeFunc` (*class in cogdl.layers.gcc\_module*), 57  
`ArgClass` (*class in cogdl.utils*), 133  
`args` (*in module cogdl.utils*), 134  
`args_print()` (*in module cogdl.layers.gpt\_gnn\_module*), 60  
`ASGCN` (*class in cogdl.models.nn.asgcn*), 86  
`assemble_neighbor()` (*in module cogdl.models.nn.patchy\_san*), 98  
`AttentionLayer` (*class in cogdl.models.nn.pyg\_han*), 111  
`AvgReadout` (*class in cogdl.models.nn.dgi*), 89

## B

`BACEDataset` (*class in cogdl.datasets.pyg\_strategies\_data*), 54  
`backward()` (*cogdl.models.nn.gat.SpecialSpmFunction static method*), 93  
`BaseGNNMix` (*class in cogdl.models.nn.pyg\_gcnmix*), 106  
`BaseModel` (*class in cogdl.models*), 119  
`BaseModel` (*class in cogdl.models.base\_model*), 118  
`BasesRelEmbLayer` (*class in cogdl.models.nn.compgcn*), 87  
`BaseTask` (*class in cogdl.tasks*), 127  
`BaseTask` (*class in cogdl.tasks.base\_task*), 120  
`BaseTrainer` (*class in cogdl.trainers.base\_trainer*), 128  
`basis_forward()` (*cogdl.models.nn.rgcn.RGCNLayer method*), 117  
`Batch` (*class in cogdl.data*), 32  
`Batch` (*class in cogdl.data.batch*), 23  
`batch_mean_pooling()` (*in module cogdl.utils*), 134  
`batch_sum_pooling()` (*in module cogdl.utils*), 134  
`BatchAE` (*class in cogdl.datasets.pyg\_strategies\_data*), 53  
`BatchedDiffPool` (*class in cogdl.models.nn.pyg\_diffpool*), 102  
`BatchedDiffPoolLayer` (*class in cogdl.models.nn.pyg\_diffpool*), 102  
`BatchedGraphSAGE` (*class in cogdl.models.nn.pyg\_diffpool*), 102  
`batcher()` (*in module cogdl.models.nn.dgl\_gcc*), 90  
`BatchFinetune` (*class in cogdl.datasets.pyg\_strategies\_data*), 52  
`BatchMasking` (*class in cogdl.datasets.pyg\_strategies\_data*), 52  
`BatchSubstructContext` (*class in cogdl.datasets.pyg\_strategies\_data*), 53  
`BBBPDataset` (*class in cogdl.datasets.pyg\_strategies\_data*), 54  
`bdd_forward()` (*cogdl.models.nn.rgcn.RGCNLayer method*), 117  
`BidirectionalOneShotIterator` (*class in cogdl.datasets.kg\_data*), 42  
`BioDataset` (*class in cogdl.datasets.pyg\_strategies\_data*), 54  
`BlogcatalogDataset` (*class in cogdl.datasets.matlab\_matrix*), 46  
`build_args_from_dict()` (*in module cogdl.utils*), 133  
`build_dataset()` (*in module cogdl.datasets*), 56  
`build_dataset_from_name()` (*in module cogdl.datasets*), 56  
`build_model()` (*cogdl.layers.strategies\_layers.Finetuner method*), 70  
`build_model()` (*cogdl.models.nn.patchy\_san.PatchySAN method*), 98  
`build_model()` (*in module cogdl.models*), 120  
`build_model_from_args()` (*cogdl.models.base\_model.BaseModel class method*), 118  
`build_model_from_args()` (*cogdl.models.BaseModel class method*), 119  
`build_model_from_args()` (*cogdl.models.emb.deepwalk.DeepWalk class method*), 72  
`build_model_from_args()` (*cogdl.models.emb.dgk.DeepGraphKernel class method*), 73  
`build_model_from_args()` (*cogdl.models.emb.dngr.DNGR class method*), 74  
`build_model_from_args()` (*cogdl.models.emb.gatne.GATNE class method*), 75  
`build_model_from_args()` (*cogdl.models.emb.graph2vec.Graph2Vec class method*), 76

|  |              |  |              |
|--|--------------|--|--------------|
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.grarep.GraRep class method</i> ), 77             | <i>class</i> | ( <i>cogdl.models.nn.fastgcn.FastGCN class method</i> ), 92  | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.hin2vec.Hin2vec class method</i> ), 78           | <i>class</i> | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.gat.PetarVSpGAT class method</i> ), 93          | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.hope.HOPE class method</i> ), 78                 |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.gcn.TKipfGCN class method</i> ), 94             | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.knowledge_base.KGEModel class method</i> ), 79   |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.gcnii.GCNII class method</i> ), 95              |              |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.line.LINE class method</i> ), 80                 |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.graphsage.Graphsage class method</i> ), 95      | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.metapath2vec.Metapath2vec class method</i> ), 80 |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.mixhop.MixHop class method</i> ), 96            | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.netmf.NetMF class method</i> ), 81               |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.mlp.MLP class method</i> ), 96                  |              |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.netsmf.NetSMF class method</i> ), 82             | <i>class</i> | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.mvgrl.MVGRL class method</i> ), 97              |              |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.node2vec.Node2vec class method</i> ), 82         | <i>class</i> | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.patchy_san.PatchySAN class method</i> ), 98     |              |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.prone.ProNE class method</i> ), 83               |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_cheb.Chebyshev class method</i> ), 99       | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.pte.PTE class method</i> ), 84                   |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_deepergcn.DeeperGCN class method</i> ), 100 |              |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.sdne.SDNE class method</i> ), 85                 |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_dgcnn.DGCNN class method</i> ), 100         | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.emb.spectral.Spectral class method</i> ), 85         | <i>class</i> | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_diffpool.DiffPool class method</i> ), 103   | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.asgcn.ASGCN class method</i> ), 86                |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_drgat.DrGAT class method</i> ), 104         | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.compvcn.LinkPredictCompGCN class method</i> ), 88 |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_drgcn.DrGCN class method</i> ), 104         | <i>class</i> |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.dgi.DGI class method</i> ), 89                    | <i>class</i> | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_gat.GAT class method</i> ), 105             |              |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.dgl_gcc.GCC class method</i> ), 91                |              | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_gcn.GCN class method</i> ), 105             |              |
| <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.disengcn.DisenGCN class method</i> ), 91          | <i>class</i> | <code>build_model_from_args()</code><br>( <i>cogdl.models.nn.pyg_gcnmix.GCNMix class method</i> ), 106       | <i>class</i> |
| <code>build_model_from_args()</code>   |              | <code>build_model_from_args()</code>   |              |

(*cogdl.models.nn.pyg\_gin.GIN class method*), 108  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN class method*), 108  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_grand.Grand class method*), 109  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_gtn.GTN class method*), 110  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_han.HAN class method*), 111  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_infograph.InfoGraph class method*), 112  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_infomax.Infomax class method*), 113  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_sortpool.SortPool class method*), 114  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_srgcn.SRGCN class method*), 115  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_stpgnn.stpgnn class method*), 115  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_unet.UNet class method*), 116  
 build\_model\_from\_args() (*cogdl.models.nn.pyg\_unsup\_graphsage.Graphsage class method*), 117  
 build\_model\_from\_args() (*cogdl.models.nn.rgcn.LinkPredictRGCN class method*), 118  
 build\_task() (*in module cogdl.tasks*), 128  
 build\_trainer\_from\_args() (*cogdl.trainers.base\_trainer.BaseTrainer class method*), 128  
 build\_trainer\_from\_args() (*cogdl.trainers.deepergcn\_trainer.DeeperGCNTrainer class method*), 129  
 build\_trainer\_from\_args() (*cogdl.trainers.gpt\_gnn\_trainer.GPT\_GNNHomogeneousTrainer class method*), 129  
 build\_trainer\_from\_args() (*cogdl.trainers.sampled\_trainer.SAINTrainer static method*), 130  
 cal\_mrr() (*in module cogdl.layers.link\_prediction\_module*), 63  
 cat\_dim() (*cogdl.data.Data method*), 32  
 cat\_dim() (*cogdl.data.data.Data method*), 25  
 cat\_dim() (*cogdl.datasets.pyg\_strategies\_data.BatchAE method*), 53  
 cat\_dim() (*cogdl.datasets.pyg\_strategies\_data.BatchSubstructContext method*), 53  
 ccorr() (*in module cogdl.models.nn.compgcn*), 87  
 Chebyshev (*class in cogdl.models.nn.pyg\_cheb*), 99  
 chebyshev() (*cogdl.layers.prone\_module.HeatKernelApproximation method*), 66  
 ChemExtractSubstructureContextPair (*class in cogdl.datasets.pyg\_strategies\_data*), 52  
 CiteSeerDataset (*class in cogdl.datasets.pyg*), 47  
 Classifier (*class in cogdl.layers.gpt\_gnn\_module*), 62  
 clone() (*cogdl.data.Data method*), 32  
 clone() (*cogdl.data.data.Data method*), 25  
 cmp() (*in module cogdl.models.nn.patchy\_san*), 98  
 cogdl  
   module, 23  
 cogdl.data  
   module, 23  
   cogdl.data.batch  
     module, 23  
   cogdl.data.data  
     module, 24  
   cogdl.data.data\_loader  
     module, 26  
   cogdl.data.dataset  
     module, 27  
   cogdl.data.download  
     module, 28  
   cogdl.data.extract  
     module, 28  
   cogdl.data.makedirs  
     module, 29  
   cogdl.data.sampler  
     module, 29  
 cogdl.datasets  
   module, 35  
   cogdl.datasets.dgl\_data  
     module, 35  
   cogdl.datasets.gatne  
     module, 36  
   cogdl.datasets.gcc\_data  
     module, 37  
   cogdl.datasets.gtn\_data  
     module, 38  
   cogdl.datasets.han\_data  
     module, 40  
   cogdl.datasets.kg\_data  
     module, 41  
   cogdl.datasets.matlab\_matrix

## C

cal\_mrr() (*in module*)



- module, 45
- cogdl.datasets.pyg
  - module, 46
- cogdl.datasets.pyg\_ogb
  - module, 48
- cogdl.datasets.pyg\_strategies\_data
  - module, 50
- cogdl.layers
  - module, 57
- cogdl.layers.gcc\_module
  - module, 57
- cogdl.layers.gpt\_gnn\_module
  - module, 59
- cogdl.layers.link\_prediction\_module
  - module, 63
- cogdl.layers.maggregator
  - module, 64
- cogdl.layers.mixhop\_layer
  - module, 65
- cogdl.layers.prone\_module
  - module, 65
- cogdl.layers.se\_layer
  - module, 67
- cogdl.layers.srgcn\_module
  - module, 67
- cogdl.layers.strategies\_layers
  - module, 69
- cogdl.models
  - module, 71
- cogdl.models.base\_model
  - module, 118
- cogdl.models.emb
  - module, 71
- cogdl.models.emb.complex
  - module, 71
- cogdl.models.emb.deepwalk
  - module, 72
- cogdl.models.emb.dgk
  - module, 73
- cogdl.models.emb.distmult
  - module, 73
- cogdl.models.emb.dngr
  - module, 74
- cogdl.models.emb.gatne
  - module, 74
- cogdl.models.emb.graph2vec
  - module, 76
- cogdl.models.emb.grarep
  - module, 77
- cogdl.models.emb.hin2vec
  - module, 77
- cogdl.models.emb.hope
  - module, 78
- cogdl.models.emb.knowledge\_base
  - module, 79
- cogdl.models.emb.line
  - module, 79
- cogdl.models.emb.metapath2vec
  - module, 80
- cogdl.models.emb.netmf
  - module, 81
- cogdl.models.emb.netsmf
  - module, 81
- cogdl.models.emb.node2vec
  - module, 82
- cogdl.models.emb.prone
  - module, 83
- cogdl.models.emb.pte
  - module, 83
- cogdl.models.emb.rotate
  - module, 84
- cogdl.models.emb.sdne
  - module, 84
- cogdl.models.emb.spectral
  - module, 85
- cogdl.models.emb.transe
  - module, 85
- cogdl.models.nn
  - module, 86
- cogdl.models.nn.asgcn
  - module, 86
- cogdl.models.nn.compvcn
  - module, 87
- cogdl.models.nn.dgi
  - module, 88
- cogdl.models.nn.dgl\_gcc
  - module, 90
- cogdl.models.nn.disengcn
  - module, 91
- cogdl.models.nn.fastgcn
  - module, 92
- cogdl.models.nn.gat
  - module, 93
- cogdl.models.nn.gcn
  - module, 94
- cogdl.models.nn.gcnii
  - module, 94
- cogdl.models.nn.graphsage
  - module, 95
- cogdl.models.nn.mixhop
  - module, 96
- cogdl.models.nn.mlp
  - module, 96
- cogdl.models.nn.mvgrl
  - module, 97
- cogdl.models.nn.patchy\_san
  - module, 98
- cogdl.models.nn.pyg\_cheb

module, 99  
 cogdl.models.nn.pyg\_deepergcn  
   module, 99  
 cogdl.models.nn.pyg\_dgcnn  
   module, 100  
 cogdl.models.nn.pyg\_diffpool  
   module, 101  
 cogdl.models.nn.pyg\_drgat  
   module, 103  
 cogdl.models.nn.pyg\_drgcn  
   module, 104  
 cogdl.models.nn.pyg\_gat  
   module, 104  
 cogdl.models.nn.pyg\_gcn  
   module, 105  
 cogdl.models.nn.pyg\_gcnmix  
   module, 105  
 cogdl.models.nn.pyg\_gin  
   module, 107  
 cogdl.models.nn.pyg\_gpt\_gnn  
   module, 108  
 cogdl.models.nn.pyg\_grand  
   module, 109  
 cogdl.models.nn.pyg\_gtn  
   module, 110  
 cogdl.models.nn.pyg\_han  
   module, 110  
 cogdl.models.nn.pyg\_infograph  
   module, 111  
 cogdl.models.nn.pyg\_infomax  
   module, 113  
 cogdl.models.nn.pyg\_sortpool  
   module, 113  
 cogdl.models.nn.pyg\_srgcn  
   module, 114  
 cogdl.models.nn.pyg\_stpgnn  
   module, 115  
 cogdl.models.nn.pyg\_unet  
   module, 116  
 cogdl.models.nn.pyg\_unsup\_graphsage  
   module, 116  
 cogdl.models.nn.rgcn  
   module, 117  
 cogdl.models.supervised\_model  
   module, 118  
 cogdl.options  
   module, 131  
 cogdl.tasks  
   module, 120  
 cogdl.tasks.base\_task  
   module, 120  
 cogdl.tasks.graph\_classification  
   module, 121  
 cogdl.tasks.heterogeneous\_node\_classification  
   module, 121  
 cogdl.tasks.link\_prediction  
   module, 122  
 cogdl.tasks.multiplex\_link\_prediction  
   module, 123  
 cogdl.tasks.multiplex\_node\_classification  
   module, 124  
 cogdl.tasks.node\_classification  
   module, 124  
 cogdl.tasks.node\_classification\_sampling  
   module, 125  
 cogdl.tasks.pretrain  
   module, 126  
 cogdl.tasks.unsupervised\_graph\_classification  
   module, 126  
 cogdl.tasks.unsupervised\_node\_classification  
   module, 126  
 cogdl.trainers  
   module, 128  
 cogdl.trainers.base\_trainer  
   module, 128  
 cogdl.trainers.deepergcn\_trainer  
   module, 128  
 cogdl.trainers.gpt\_gnn\_trainer  
   module, 129  
 cogdl.trainers.sampled\_trainer  
   module, 130  
 cogdl.trainers.supervised\_trainer  
   module, 130  
 cogdl.trainers.unsupervised\_trainer  
   module, 131  
 cogdl.utils  
   module, 132  
 CollabDataset (class in cogdl.datasets.dgl\_data), 35  
 CollabDataset (class in cogdl.datasets.pyg), 47  
 collate\_fn() (cogdl.datasets.kg\_data.TestDataset  
   static method), 42  
 collate\_fn() (cogdl.datasets.kg\_data.TrainDataset  
   static method), 42  
 ColumnUniform (class in cogdl.layers.srgcn\_module),  
   68  
 com\_mult() (in module cogdl.models.nn.compvcn), 87  
 CompGCN (class in cogdl.models.nn.compvcn), 87  
 CompGCNLayer (class in cogdl.models.nn.compvcn),  
   87  
 Complex (class in cogdl.models.emb.complex), 72  
 compute\_adjlist()  
   (cogdl.models.nn.asgcn.ASGCN method),  
   86  
 compute\_ppr() (in module cogdl.models.nn.mvgrl),  
   97  
 concat() (cogdl.layers.link\_prediction\_module.ConvELayer  
   method), 63  
 concat() (in module cogdl.models.nn.compvcn), 87

`consis_loss()` (*cogdl.models.nn.pyg\_grand.Grand method*), 109  
`ContextPredictTrainer` (*class in cogdl.layers.strategies\_layers*), 70  
`contiguous()` (*cogdl.data.Data method*), 32  
`contiguous()` (*cogdl.data.data.Data method*), 25  
`ConvELayer` (*class in cogdl.layers.link\_prediction\_module*), 63  
`CoraDataset` (*class in cogdl.datasets.pyg*), 47  
`corruption()` (*in module cogdl.models.nn.pyg\_infomax*), 113  
`count_frequency()` (*cogdl.datasets.kg\_data.TrainDataset static method*), 42  
`cuda()` (*cogdl.data.Data method*), 32  
`cuda()` (*cogdl.data.data.Data method*), 25  
`cumsum()` (*cogdl.data.Batch method*), 33  
`cumsum()` (*cogdl.data.batch.Batch method*), 23  
`cumsum()` (*cogdl.datasets.pyg\_strategies\_data.BatchMasking method*), 52  
`cumsum()` (*cogdl.datasets.pyg\_strategies\_data.BatchSubstructContext method*), 53  
`cycle_index()` (*in module cogdl.utils*), 134

## D

`Data` (*class in cogdl.data*), 31  
`Data` (*class in cogdl.data.data*), 24  
`data_preparation()` (*cogdl.models.emb.hin2vec.RWgraph method*), 78  
`DataListLoader` (*class in cogdl.data*), 34  
`DataListLoader` (*class in cogdl.data.data\_loader*), 26  
`DataLoader` (*class in cogdl.data*), 34  
`DataLoader` (*class in cogdl.data.data\_loader*), 26  
`DataLoaderAE` (*class in cogdl.datasets.pyg\_strategies\_data*), 53  
`DataLoaderFinetune` (*class in cogdl.datasets.pyg\_strategies\_data*), 53  
`DataLoaderMasking` (*class in cogdl.datasets.pyg\_strategies\_data*), 53  
`DataLoaderSubstructContext` (*class in cogdl.datasets.pyg\_strategies\_data*), 53  
`Dataset` (*class in cogdl.data*), 33  
`Dataset` (*class in cogdl.data.dataset*), 27  
`Dataset` (*class in cogdl.datasets*), 55  
`dataset_name` (*in module cogdl.datasets*), 56  
`DATASET_REGISTRY` (*in module cogdl.datasets*), 56  
`DBLP_GTNDataset` (*class in cogdl.datasets.gtn\_data*), 39  
`DBLP_HANDataset` (*class in cogdl.datasets.han\_data*), 41  
`dcg_at_k()` (*in module cogdl.layers.gpt\_gnn\_module*), 60  
`DeeperGCN` (*class in cogdl.models.nn.pyg\_deepergcn*), 100  
`DeeperGCNTrainer` (*class in cogdl.trainers.deepergcn\_trainer*), 128  
`DeepGCNLayer` (*class in cogdl.models.nn.pyg\_deepergcn*), 100  
`DeepGraphKernel` (*class in cogdl.models.emb.dgk*), 73  
`DeepWalk` (*class in cogdl.models.emb.deepwalk*), 72  
`defaultDictDict` (*in module cogdl.layers.gpt\_gnn\_module*), 61  
`defaultDictDictDictDictDictInt` (*in module cogdl.layers.gpt\_gnn\_module*), 61  
`defaultDictDictDictDictInt` (*in module cogdl.layers.gpt\_gnn\_module*), 61  
`defaultDictDictDictInt` (*in module cogdl.layers.gpt\_gnn\_module*), 61  
`defaultDictDictInt` (*in module cogdl.layers.gpt\_gnn\_module*), 61  
`defaultDictInt` (*in module cogdl.layers.gpt\_gnn\_module*), 61  
`defaultDictList` (*in module cogdl.layers.gpt\_gnn\_module*), 61  
`DenseDataLoader` (*class in cogdl.data*), 34  
`DenseDataLoader` (*class in cogdl.data.data\_loader*), 26  
`DGCNN` (*class in cogdl.models.nn.pyg\_dgcnn*), 100  
`DGI` (*class in cogdl.models.nn.dgi*), 89  
`DGIModel` (*class in cogdl.models.nn.dgi*), 89  
`dgl_import` (*in module cogdl.datasets*), 56  
`dgl_import` (*in module cogdl.models*), 120  
`DiffPool` (*class in cogdl.models.nn.pyg\_diffpool*), 103  
`Discriminator` (*class in cogdl.layers.strategies\_layers*), 70  
`Discriminator` (*class in cogdl.models.nn.dgi*), 89  
`Discriminator` (*class in cogdl.models.nn.mvgrl*), 97  
`DisenGCN` (*class in cogdl.models.nn.disengcn*), 91  
`DisenGCNLayer` (*class in cogdl.models.nn.disengcn*), 91  
`DistMult` (*class in cogdl.models.emb.distmult*), 73  
`DistMultLayer` (*class in cogdl.layers.link\_prediction\_module*), 63  
`divide_data()` (*in module cogdl.tasks.link\_prediction*), 123  
`DNGR` (*class in cogdl.models.emb.dngr*), 74  
`DNGR_layer` (*class in cogdl.models.emb.dngr*), 74  
`download()` (*cogdl.data.Dataset method*), 33  
`download()` (*cogdl.data.dataset.Dataset method*), 27  
`download()` (*cogdl.datasets.Dataset method*), 55  
`download()` (*cogdl.datasets.gatne.GatneDataset method*), 36  
`download()` (*cogdl.datasets.gcc\_data.Edgelist method*), 38  
`download()` (*cogdl.datasets.gtn\_data.GTNDataset method*), 39



- method*), 39
- `download()` (*cogdl.datasets.han\_data.HANDataset method*), 40
- `download()` (*cogdl.datasets.kg\_data.KnowledgeGraphDataset method*), 43
- `download()` (*cogdl.datasets.matlab\_matrix.MatlabMatrix method*), 45
- `download()` (*cogdl.datasets.pyg\_strategies\_data.BACEDataset method*), 54
- `download()` (*cogdl.datasets.pyg\_strategies\_data.BBBPDataset method*), 55
- `download()` (*cogdl.datasets.pyg\_strategies\_data.BioDataset method*), 54
- `download()` (*cogdl.datasets.pyg\_strategies\_data.MoleculeDataset method*), 54
- `download_url()` (*in module cogdl.data*), 34
- `download_url()` (*in module cogdl.data.download*), 28
- DrGAT (*class in cogdl.models.nn.pyg\_drgat*), 103
- DrGCN (*class in cogdl.models.nn.pyg\_drgcn*), 104
- `dropNode()` (*cogdl.models.nn.pyg\_grand.Grand method*), 109
- ## E
- `edge_attention()` (*cogdl.layers.gcc\_module.GATLayer method*), 57
- `edge_softmax()` (*in module cogdl.utils*), 133
- EdgeAttention (*class in cogdl.layers.srgcn\_module*), 68
- Edgelist (*class in cogdl.datasets.gcc\_data*), 37
- EdgeSampler (*class in cogdl.data.sampler*), 30
- `eigen_decomposition()` (*in module cogdl.models.nn.dgl\_gcc*), 90
- `embed()` (*cogdl.models.nn.dgi.DGIModel method*), 89
- `embed()` (*cogdl.models.nn.mvgrl.Model method*), 97
- `embed()` (*cogdl.models.nn.pyg\_unsup\_graphsage.SAGE method*), 116
- Encoder (*class in cogdl.models.nn.pyg\_infograph*), 111
- Encoder (*class in cogdl.models.nn.pyg\_infomax*), 113
- `enhance_emb()` (*cogdl.tasks.unsupervised\_node\_classification.cogdl.tasks.unsupervised\_node\_classification method*), 127
- EntropyLoss (*class in cogdl.models.nn.pyg\_diffpool*), 101
- ENZYMES (*class in cogdl.datasets.pyg*), 48
- `estimate()` (*cogdl.data.sampler.SAINTSampler method*), 30
- `evaluate()` (*cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN method*), 108
- `evaluate()` (*cogdl.models.nn.pyg\_gtn.GTN method*), 110
- `evaluate()` (*cogdl.models.nn.pyg\_han.HAN method*), 111
- `evaluate()` (*cogdl.models.supervised\_model.SupervisedHeterogeneousModelClassificationMethod method*), 119
- `evaluate()` (*cogdl.trainers.gpt\_gnn\_trainer.GPT\_GNNHeterogeneousTrainer method*), 130
- `evaluate()` (*in module cogdl.tasks.link\_prediction*), 123
- `evaluate()` (*in module cogdl.tasks.multiplex\_link\_prediction*), 124
- `extract_bz2()` (*in module cogdl.data*), 35
- `extract_bz2()` (*in module cogdl.data.extract*), 29
- `extract_gz()` (*in module cogdl.data*), 35
- `extract_gz()` (*in module cogdl.data.extract*), 29
- `extract_subgraph()` (*cogdl.data.sampler.SAINTSampler method*), 30
- `extract_tar()` (*in module cogdl.data*), 35
- `extract_tar()` (*in module cogdl.data.extract*), 28
- `extract_zip()` (*in module cogdl.data*), 35
- `extract_zip()` (*in module cogdl.data.extract*), 29
- ExtractSubstructureContextPair (*class in cogdl.datasets.pyg\_strategies\_data*), 52
- ## F
- FastGCN (*class in cogdl.models.nn.fastgcn*), 92
- FB13Dataset (*class in cogdl.datasets.kg\_data*), 43
- FB13SDataset (*class in cogdl.datasets.kg\_data*), 44
- FB15k237Dataset (*class in cogdl.datasets.kg\_data*), 43
- FB15kDataset (*class in cogdl.datasets.kg\_data*), 43
- `feat_loss()` (*cogdl.layers.gpt\_gnn\_module.GPT\_GNN method*), 62
- `feature_extractor()` (*cogdl.models.emb.dgk.DeepGraphKernel static method*), 73
- `feature_extractor()` (*cogdl.models.emb.graph2vec.Graph2Vec static method*), 76
- `feature_OAG()` (*in module cogdl.layers.gpt\_gnn\_module*), 61
- `feature_reddit()` (*in module cogdl.layers.gpt\_gnn\_module*), 61
- FeatureLoss (*class in cogdl.tasks.unsupervised\_node\_classification.cogdl.tasks.unsupervised\_node\_classification method*), 112
- `files_exist()` (*in module cogdl.data.dataset*), 27
- Finetuner (*class in cogdl.layers.strategies\_layers*), 70
- `fit()` (*cogdl.layers.strategies\_layers.Finetuner method*), 70
- `fit()` (*cogdl.layers.strategies\_layers.Pretrainer method*), 70
- `fit()` (*cogdl.trainers.deepergcn\_trainer.DeeperGCNTrainer method*), 129
- `fit()` (*cogdl.trainers.gpt\_gnn\_trainer.GPT\_GNNHeterogeneousTrainer method*), 130
- `fit()` (*cogdl.trainers.gpt\_gnn\_trainer.GPT\_GNNHomogeneousTrainer method*), 129
- `fit()` (*cogdl.trainers.gpt\_gnn\_trainer.GPT\_GNNHeterogeneousTrainer.SAINTTrainer method*), 130

`fit()` (`cogdl.trainers.sampled_trainer.SampledTrainer` method), 130  
`fit()` (`cogdl.trainers.supervised_trainer.SupervisedHeterogeneousNodeClassificationTrainer` method), 131  
`fit()` (`cogdl.trainers.supervised_trainer.SupervisedHomogeneousNodeClassificationTrainer` method), 131  
`fit()` (`cogdl.trainers.supervised_trainer.SupervisedTrainer` method), 131  
`FlickrDataset` (class in `cogdl.datasets.matlab_matrix`), 46  
`forward()` (`cogdl.layers.gcc_module.ApplyNodeFunc` method), 57  
`forward()` (`cogdl.layers.gcc_module.GATLayer` method), 57  
`forward()` (`cogdl.layers.gcc_module.GraphEncoder` method), 59  
`forward()` (`cogdl.layers.gcc_module.MLP` method), 57  
`forward()` (`cogdl.layers.gcc_module.SELayer` method), 57  
`forward()` (`cogdl.layers.gcc_module.UnsupervisedGAT` method), 57  
`forward()` (`cogdl.layers.gcc_module.UnsupervisedGIN` method), 58  
`forward()` (`cogdl.layers.gcc_module.UnsupervisedMPNN` method), 58  
`forward()` (`cogdl.layers.gpt_gnn_module.Classifier` method), 62  
`forward()` (`cogdl.layers.gpt_gnn_module.GeneralConv` method), 62  
`forward()` (`cogdl.layers.gpt_gnn_module.GNN` method), 62  
`forward()` (`cogdl.layers.gpt_gnn_module.GPT_GNN` method), 62  
`forward()` (`cogdl.layers.gpt_gnn_module.HGTConv` method), 61  
`forward()` (`cogdl.layers.gpt_gnn_module.Matcher` method), 62  
`forward()` (`cogdl.layers.gpt_gnn_module.RelTemporalEncoding` method), 62  
`forward()` (`cogdl.layers.gpt_gnn_module.RNNModel` method), 62  
`forward()` (`cogdl.layers.link_prediction_module.ConvELayer` method), 63  
`forward()` (`cogdl.layers.link_prediction_module.DistMultLayer` method), 63  
`forward()` (`cogdl.layers.link_prediction_module.GNNLinkPredict` method), 63  
`forward()` (`cogdl.layers.maggregator.MeanAggregator` method), 64  
`forward()` (`cogdl.layers.MeanAggregator` method), 71  
`forward()` (`cogdl.layers.mixhop_layer.MixHopLayer` method), 65  
`forward()` (`cogdl.layers.MixHopLayer` method), 71  
`forward()` (`cogdl.layers.se_layer.SELayer` method), 67  
`forward()` (`cogdl.layers.SELayer` method), 71  
`forward()` (`cogdl.layers.srgcn_module.ColumnUniform` method), 68  
`forward()` (`cogdl.layers.srgcn_module.EdgeAttention` method), 68  
`forward()` (`cogdl.layers.srgcn_module.HeatKernel` method), 68  
`forward()` (`cogdl.layers.srgcn_module.Identity` method), 68  
`forward()` (`cogdl.layers.srgcn_module.NodeAttention` method), 68  
`forward()` (`cogdl.layers.srgcn_module.NormIdentity` method), 68  
`forward()` (`cogdl.layers.srgcn_module.PPR` method), 68  
`forward()` (`cogdl.layers.srgcn_module.RowSoftmax` method), 68  
`forward()` (`cogdl.layers.srgcn_module.RowUniform` method), 68  
`forward()` (`cogdl.layers.srgcn_module.SymmetryNorm` method), 68  
`forward()` (`cogdl.layers.strategies_layers.Discriminator` method), 70  
`forward()` (`cogdl.layers.strategies_layers.GINConv` method), 69  
`forward()` (`cogdl.layers.strategies_layers.GNN` method), 69  
`forward()` (`cogdl.layers.strategies_layers.GNNPred` method), 70  
`forward()` (`cogdl.models.emb.dgk.DeepGraphKernel` method), 73  
`forward()` (`cogdl.models.emb.dngr.DNGR_layer` method), 74  
`forward()` (`cogdl.models.emb.gatne.GATNEModel` method), 75  
`forward()` (`cogdl.models.emb.gatne.NSLoss` method), 75  
`forward()` (`cogdl.models.emb.graph2vec.Graph2Vec` method), 76  
`forward()` (`cogdl.models.emb.hin2vec.Hin2vec_layer` method), 77  
`forward()` (`cogdl.models.emb.knowledge_base.KGEModel` method), 79  
`forward()` (`cogdl.models.emb.s dne.SDNE_layer` method), 84  
`forward()` (`cogdl.models.nn.asgcn.ASGCN` method), 86  
`forward()` (`cogdl.models.nn.asgcn.GraphConvolution` method), 86  
`forward()` (`cogdl.models.nn.compgcn.BasesRelEmbLayer` method), 87  
`forward()` (`cogdl.models.nn.compgcn.CompGCN` method), 87

`forward()` (*cogdl.models.nn.compgcn.CompGCNLayer method*), 87  
`forward()` (*cogdl.models.nn.compgcn.LinkPredictCompGCN method*), 88  
`forward()` (*cogdl.models.nn.dgi.AvgReadout method*), 89  
`forward()` (*cogdl.models.nn.dgi.DGIModel method*), 89  
`forward()` (*cogdl.models.nn.dgi.Discriminator method*), 89  
`forward()` (*cogdl.models.nn.dgi.GCN method*), 89  
`forward()` (*cogdl.models.nn.dgi.LogReg method*), 89  
`forward()` (*cogdl.models.nn.disengcn.DisenGCN method*), 91  
`forward()` (*cogdl.models.nn.disengcn.DisenGCNLayer method*), 91  
`forward()` (*cogdl.models.nn.fastgcn.FastGCN method*), 92  
`forward()` (*cogdl.models.nn.fastgcn.GraphConvolution method*), 92  
`forward()` (*cogdl.models.nn.gat.PetarVSpGAT method*), 93  
`forward()` (*cogdl.models.nn.gat.SpecialSpm method*), 93  
`forward()` (*cogdl.models.nn.gat.SpecialSpmFunction static method*), 93  
`forward()` (*cogdl.models.nn.gat.SpGraphAttentionLayer method*), 93  
`forward()` (*cogdl.models.nn.gcn.GraphConvolution method*), 94  
`forward()` (*cogdl.models.nn.gcn.TKipfGCN method*), 94  
`forward()` (*cogdl.models.nn.gcnii.GCNII method*), 95  
`forward()` (*cogdl.models.nn.gcnii.GCNIILayer method*), 95  
`forward()` (*cogdl.models.nn.graphsage.Graphsage method*), 95  
`forward()` (*cogdl.models.nn.graphsage.GraphSAGELayer method*), 95  
`forward()` (*cogdl.models.nn.mixhop.MixHop method*), 96  
`forward()` (*cogdl.models.nn.mlp.MLP method*), 96  
`forward()` (*cogdl.models.nn.mvgrl.Discriminator method*), 97  
`forward()` (*cogdl.models.nn.mvgrl.Model method*), 97  
`forward()` (*cogdl.models.nn.patchy\_san.PatchySAN method*), 98  
`forward()` (*cogdl.models.nn.pyg\_cheb.Chebyshev method*), 99  
`forward()` (*cogdl.models.nn.pyg\_deepergcn.DeeperGCN method*), 100  
`forward()` (*cogdl.models.nn.pyg\_deepergcn.DeepGCNLayer method*), 100  
`forward()` (*cogdl.models.nn.pyg\_deepergcn.GENConv method*), 100  
`forward()` (*cogdl.models.nn.pyg\_dgcnn.DGCNN method*), 101  
`forward()` (*cogdl.models.nn.pyg\_diffpool.BatchedDiffPool method*), 102  
`forward()` (*cogdl.models.nn.pyg\_diffpool.BatchedDiffPoolLayer method*), 102  
`forward()` (*cogdl.models.nn.pyg\_diffpool.BatchedGraphSAGE method*), 102  
`forward()` (*cogdl.models.nn.pyg\_diffpool.DiffPool method*), 103  
`forward()` (*cogdl.models.nn.pyg\_diffpool.EntropyLoss method*), 101  
`forward()` (*cogdl.models.nn.pyg\_diffpool.GraphSAGE method*), 102  
`forward()` (*cogdl.models.nn.pyg\_diffpool.LinkPredLoss method*), 101  
`forward()` (*cogdl.models.nn.pyg\_drgat.DrGAT method*), 104  
`forward()` (*cogdl.models.nn.pyg\_drgcn.DrGCN method*), 104  
`forward()` (*cogdl.models.nn.pyg\_gat.GAT method*), 105  
`forward()` (*cogdl.models.nn.pyg\_gcn.GCN method*), 105  
`forward()` (*cogdl.models.nn.pyg\_gcnmix.BaseGNNMix method*), 106  
`forward()` (*cogdl.models.nn.pyg\_gcnmix.GCNConv method*), 106  
`forward()` (*cogdl.models.nn.pyg\_gcnmix.GCNMix method*), 106  
`forward()` (*cogdl.models.nn.pyg\_gin.GIN method*), 108  
`forward()` (*cogdl.models.nn.pyg\_gin.GINLayer method*), 107  
`forward()` (*cogdl.models.nn.pyg\_gin.GINMLP method*), 107  
`forward()` (*cogdl.models.nn.pyg\_grand.Grand method*), 109  
`forward()` (*cogdl.models.nn.pyg\_grand.MLPLayer method*), 109  
`forward()` (*cogdl.models.nn.pyg\_gtn.GTConv method*), 110  
`forward()` (*cogdl.models.nn.pyg\_gtn.GTLayer method*), 110  
`forward()` (*cogdl.models.nn.pyg\_gtn.GTN method*), 110  
`forward()` (*cogdl.models.nn.pyg\_han.AttentionLayer method*), 111  
`forward()` (*cogdl.models.nn.pyg\_han.HAN method*), 111  
`forward()` (*cogdl.models.nn.pyg\_han.HANLayer method*), 111  
`forward()` (*cogdl.models.nn.pyg\_infograph.Encoder method*), 100

- method*), 112
- `forward()` (*cogdl.models.nn.pyg\_infograph.FF method*), 112
- `forward()` (*cogdl.models.nn.pyg\_infograph.InfoGraph method*), 112
- `forward()` (*cogdl.models.nn.pyg\_infograph.SUPEncoder method*), 111
- `forward()` (*cogdl.models.nn.pyg\_infomax.Encoder method*), 113
- `forward()` (*cogdl.models.nn.pyg\_infomax.Infomax method*), 113
- `forward()` (*cogdl.models.nn.pyg\_sortpool.SortPool method*), 114
- `forward()` (*cogdl.models.nn.pyg\_srgcn.NodeAdaptiveEncoder method*), 115
- `forward()` (*cogdl.models.nn.pyg\_srgcn.SRGCN method*), 115
- `forward()` (*cogdl.models.nn.pyg\_srgcn.SrgcnHead method*), 115
- `forward()` (*cogdl.models.nn.pyg\_srgcn.SrgcnSoftmaxHead method*), 115
- `forward()` (*cogdl.models.nn.pyg\_unet.UNet method*), 116
- `forward()` (*cogdl.models.nn.pyg\_unsup\_graphsage.SAGE method*), 116
- `forward()` (*cogdl.models.nn.rgcn.LinkPredictRGCN method*), 118
- `forward()` (*cogdl.models.nn.rgcn.RGCN method*), 117
- `forward()` (*cogdl.models.nn.rgcn.RGCNLayer method*), 117
- `forward_aux()` (*cogdl.models.nn.pyg\_gcnmix.BaseGNNMix method*), 106
- `forward_aux()` (*cogdl.models.nn.pyg\_gcnmix.GCNConv method*), 106
- `forward_ema()` (*cogdl.models.nn.pyg\_gcnmix.GCNMix method*), 106
- `from_adjlist()` (*cogdl.models.nn.asgcn.ASGCN method*), 86
- `from_data_list()` (*cogdl.data.Batch static method*), 32
- `from_data_list()` (*cogdl.data.batch.Batch static method*), 23
- `from_data_list()` (*cogdl.datasets.pyg\_strategies\_data.BatchAE static method*), 53
- `from_data_list()` (*cogdl.datasets.pyg\_strategies\_data.BatchFinetune static method*), 52
- `from_data_list()` (*cogdl.datasets.pyg\_strategies\_data.BatchMasking static method*), 52
- `from_data_list()` (*cogdl.datasets.pyg\_strategies\_data.BatchSubstructContext static method*), 53
- `from_dict()` (*cogdl.data.Data static method*), 31
- `from_dict()` (*cogdl.data.data.Data static method*), 24
- `from_w2v()` (*cogdl.layers.gpt\_gnn\_module.RNNModel method*), 62
- ## G
- `GAT` (*class in cogdl.models.nn.pyg\_gat*), 104
- `GATLayer` (*class in cogdl.layers.gcc\_module*), 57
- `GATNE` (*class in cogdl.models.emb.gatne*), 75
- `GatneDataset` (*class in cogdl.datasets.gatne*), 36
- `GATNEModel` (*class in cogdl.models.emb.gatne*), 75
- `Gaussian` (*class in cogdl.layers.prone\_module*), 66
- `GCC` (*class in cogdl.models.nn.dgl\_gcc*), 91
- `GCN` (*class in cogdl.models.nn.dgi*), 89
- `GCN` (*class in cogdl.models.nn.pyg\_gcn*), 105
- `GCNConv` (*class in cogdl.models.nn.pyg\_gcnmix*), 106
- `GCNII` (*class in cogdl.models.nn.gcnii*), 95
- `GCNIILayer` (*class in cogdl.models.nn.gcnii*), 94
- `GCNMix` (*class in cogdl.models.nn.pyg\_gcnmix*), 106
- `gen_node_pairs()` (*in cogdl.tasks.link\_prediction*), 123
- `gen_subgraph()` (*cogdl.data.sampler.SAINTSampler method*), 30
- `GENConv` (*class in cogdl.models.nn.pyg\_deepergcn*), 99
- `GeneralConv` (*class in cogdl.layers.gpt\_gnn\_module*), 62
- `generate_data()` (*cogdl.tasks.graph\_classification.GraphClassification method*), 121
- `generate_pairs()` (*in cogdl.models.emb.gatne*), 76
- `generate_subgraphs()` (*in cogdl.trainers.deepergcn\_trainer*), 128
- `generate_vocab()` (*in cogdl.models.emb.gatne*), 76
- `generate_walks()` (*in cogdl.models.emb.gatne*), 76
- `get()` (*cogdl.data.Dataset method*), 33
- `get()` (*cogdl.data.dataset.Dataset method*), 27
- `get()` (*cogdl.datasets.Dataset method*), 56
- `get()` (*cogdl.datasets.gatne.GatneDataset method*), 36
- `get()` (*cogdl.datasets.gcc\_data.Edgelist method*), 38
- `get()` (*cogdl.datasets.gtn\_data.GTNDataset method*), 39
- `get()` (*cogdl.datasets.han\_data.HANDataset method*), 40
- `get()` (*cogdl.datasets.kg\_data.KnowledgeGraphDataset method*), 43
- `get()` (*cogdl.datasets.matlab\_matrix.MatlabMatrix method*), 46
- `get()` (*cogdl.datasets.pyg\_ogb.OGBGDataset method*), 49
- `get()` (*cogdl.datasets.pyg\_ogb.OGBNDataset method*), 49
- `get()` (*cogdl.datasets.pyg\_strategies\_data.BACEDataset method*), 54
- `get()` (*cogdl.datasets.pyg\_strategies\_data.BBBPDataset method*), 55
- `get()` (*cogdl.datasets.pyg\_strategies\_data.MoleculeDataset method*), 54



`get()` (*cogdl.datasets.pyg\_strategies\_data.TestChemDataset* method), 54  
`get_activation()` (*in module cogdl.utils*), 134  
`get_batches()` (*cogdl.data.sampler.LayerSampler* method), 30  
`get_batches()` (*in module cogdl.models.emb.gatne*), 76  
`get_batches()` (*in module cogdl.tasks.node\_classification\_sampling*), 125  
`get_cbow_pred()` (*cogdl.layers.strategies\_layers.ContextPredictTrainer* method), 70  
`get_current_consistency_weight()` (*in module cogdl.models.nn.pyg\_gcnmix*), 106  
`get_dataset()` (*cogdl.layers.strategies\_layers.Pretrainer* method), 70  
`get_degrees()` (*in module cogdl.utils*), 133  
`get_denoised_matrix()` (*cogdl.models.emb.dngr.DNGR* method), 74  
`get_display_data_parser()` (*in module cogdl.options*), 132  
`get_download_data_parser()` (*in module cogdl.options*), 132  
`get_edge_set()` (*cogdl.layers.link\_prediction\_module.GNNLinkPredict* method), 64  
`get_emb()` (*cogdl.models.emb.dngr.DNGR* method), 74  
`get_emb()` (*cogdl.models.emb.hin2vec.Hin2vec\_layer* method), 77  
`get_emb()` (*cogdl.models.emb.sdne.SDNE\_layer* method), 84  
`get_embedding()` (*cogdl.trainers.unsupervised\_trainer.UnsupervisedTrainer* method), 131  
`get_embedding_dense()` (*in module cogdl.layers.prone\_module*), 66  
`get_filtered_rank()` (*in module cogdl.layers.link\_prediction\_module*), 64  
`get_G_from_edges()` (*in module cogdl.models.emb.gatne*), 76  
`get_loader()` (*cogdl.datasets.pyg\_ogb.OGBGDataset* method), 49  
`get_loss()` (*cogdl.models.nn.pyg\_diffpool.BatchedDiffPool* method), 102  
`get_loss()` (*cogdl.models.nn.pyg\_diffpool.BatchedDiffPool* method), 102  
`get_meta_graph()` (*cogdl.layers.gpt\_gnn\_module.GraphGNN* method), 61  
`get_one_hot_label()` (*in module cogdl.models.nn.pyg\_gcnmix*), 106  
`get_optimizer()` (*cogdl.models.nn.gcnii.GCNII* method), 95  
`get_param()` (*cogdl.models.nn.comp\_gcn.CompGCNLayer* method), 87  
`get_parser()` (*in module cogdl.options*), 132  
`get_ppmi_matrix()` (*cogdl.models.emb.dngr.DNGR* method), 74  
`get_rank()` (*in module cogdl.layers.link\_prediction\_module*), 64  
`get_raw_rank()` (*in module cogdl.layers.link\_prediction\_module*), 64  
`get_score()` (*cogdl.layers.link\_prediction\_module.GNNLinkPredict* method), 63  
`get_score()` (*in module cogdl.tasks.link\_prediction*), 72  
`get_score()` (*in module cogdl.tasks.multiplex\_link\_prediction*), 124  
`get_single_feature()` (*in module cogdl.models.nn.patchy\_san*), 99  
`get_skipgram_pred()` (*cogdl.layers.strategies\_layers.ContextPredictTrainer* method), 70  
`get_subgraph()` (*cogdl.data.sampler.SAINTSampler* method), 30  
`get_trainer()` (*cogdl.models.base\_model.BaseModel* static method), 118  
`get_trainer()` (*cogdl.models.BaseModel* static method), 119  
`get_trainer()` (*cogdl.models.nn.pyg\_deepergcn.DeeperGCN* static method), 100  
`get_trainer()` (*cogdl.models.nn.pyg\_gcn.GCN* method), 105  
`get_trainer()` (*cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN* static method), 108  
`get_trainer()` (*cogdl.models.supervised\_model.SupervisedHeterogeneousTrainer* static method), 119  
`get_trainer()` (*cogdl.models.supervised\_model.SupervisedHomogeneousTrainer* static method), 119  
`get_training_parser()` (*in module cogdl.options*), 132  
`get_true_head_and_tail()` (*cogdl.datasets.kg\_data.TrainDataset* static method), 42  
`get_types()` (*cogdl.layers.gpt\_gnn\_module.GraphGNN* method), 61  
GIN (*class in cogdl.models.nn.pyg\_gin*), 107  
GINConv (*class in cogdl.layers.strategies\_layers*), 69  
GINLayer (*class in cogdl.models.nn.pyg\_gin*), 107  
GINLayer (*class in cogdl.models.nn.pyg\_gin*), 107  
GNN (*class in cogdl.layers.gpt\_gnn\_module*), 62  
GNN (*class in cogdl.layers.strategies\_layers*), 69  
GNNLinkPredict (*class in cogdl.layers.link\_prediction\_module*), 63  
GNNPred (*class in cogdl.layers.strategies\_layers*), 69  
GPT\_GNN (*class in cogdl.layers.gpt\_gnn\_module*), 62  
GPT\_GNN (*class in cogdl.models.nn.pyg\_gpt\_gnn*), 108  
GPT\_GNNHeterogeneousTrainer (*class in cogdl.trainers.gpt\_gnn\_trainer*), 130

GPT\_GNNHomogeneousTrainer (class in *cogdl.trainers.gpt\_gnn\_trainer*), 129  
 Grand (class in *cogdl.models.nn.pyg\_grand*), 109  
 Graph (class in *cogdl.layers.gpt\_gnn\_module*), 61  
 Graph2Vec (class in *cogdl.models.emb.graph2vec*), 76  
 graph\_data\_obj\_to\_nx() (in module *cogdl.datasets.pyg\_strategies\_data*), 51  
 graph\_data\_obj\_to\_nx\_simple() (in module *cogdl.datasets.pyg\_strategies\_data*), 51  
 graph\_pool (in module *cogdl.trainers.gpt\_gnn\_trainer*), 129  
 GraphClassification (class in *cogdl.tasks.graph\_classification*), 121  
 GraphClassificationDataset (class in *cogdl.models.nn.dgl\_gcc*), 91  
 GraphConvolution (class in *cogdl.models.nn.asgcn*), 86  
 GraphConvolution (class in *cogdl.models.nn.fastgcn*), 92  
 GraphConvolution (class in *cogdl.models.nn.gcn*), 94  
 GraphEncoder (class in *cogdl.layers.gcc\_module*), 58  
 Graphsage (class in *cogdl.models.nn.graphsage*), 95  
 GraphSAGE (class in *cogdl.models.nn.pyg\_diffpool*), 101  
 Graphsage (class in *cogdl.models.nn.pyg\_unsup\_graphsage*), 116  
 GraphSAGELayer (class in *cogdl.models.nn.graphsage*), 95  
 GraRep (class in *cogdl.models.emb.grarep*), 77  
 GTConv (class in *cogdl.models.nn.pyg\_gtn*), 110  
 GTLayer (class in *cogdl.models.nn.pyg\_gtn*), 110  
 GTN (class in *cogdl.models.nn.pyg\_gtn*), 110  
 GTNDataset (class in *cogdl.datasets.gtn\_data*), 39

## H

HAN (class in *cogdl.models.nn.pyg\_han*), 111  
 HANDataset (class in *cogdl.datasets.han\_data*), 40  
 HANLayer (class in *cogdl.models.nn.pyg\_han*), 111  
 HeatKernel (class in *cogdl.layers.prone\_module*), 66  
 HeatKernel (class in *cogdl.layers.srgcn\_module*), 68  
 HeatKernelApproximation (class in *cogdl.layers.prone\_module*), 66  
 HeterogeneousNodeClassification (class in *cogdl.tasks.heterogeneous\_node\_classification*), 121  
 HGTCConv (class in *cogdl.layers.gpt\_gnn\_module*), 61  
 Hin2vec (class in *cogdl.models.emb.hin2vec*), 78  
 Hin2vec\_layer (class in *cogdl.models.emb.hin2vec*), 77  
 HomoLinkPrediction (class in *cogdl.tasks.link\_prediction*), 123  
 HOPE (class in *cogdl.models.emb.hope*), 78

## I

Identity (class in *cogdl.layers.srgcn\_module*), 68  
 IMDB\_GTNDataset (class in *cogdl.datasets.gtn\_data*), 39  
 IMDB\_HANDataset (class in *cogdl.datasets.han\_data*), 41  
 ImdbBinaryDataset (class in *cogdl.datasets.dgl\_data*), 35  
 ImdbBinaryDataset (class in *cogdl.datasets.pyg*), 47  
 ImdbMultiDataset (class in *cogdl.datasets.dgl\_data*), 35  
 ImdbMultiDataset (class in *cogdl.datasets.pyg*), 47  
 InfoGraph (class in *cogdl.models.nn.pyg\_infograph*), 112  
 Infomax (class in *cogdl.models.nn.pyg\_infomax*), 113  
 InfoMaxTrainer (class in *cogdl.layers.strategies\_layers*), 70  
 is\_coalesced() (*cogdl.data.Data* method), 32  
 is\_coalesced() (*cogdl.data.data.Data* method), 25

## K

keys() (*cogdl.data.Data* property), 31  
 keys() (*cogdl.data.data.Data* property), 24  
 KGEModel (class in *cogdl.models.emb.knowledge\_base*), 79  
 KGLinkPrediction (class in *cogdl.tasks.link\_prediction*), 123  
 KnowledgeGraphDataset (class in *cogdl.datasets.kg\_data*), 42

## L

layer (in module *cogdl.layers.mixhop\_layer*), 65  
 LayerSampler (class in *cogdl.data.sampler*), 30  
 LINE (class in *cogdl.models.emb.line*), 79  
 link\_loss() (*cogdl.layers.gpt\_gnn\_module.GPT\_GNN* method), 62  
 LinkPredictCompGCN (class in *cogdl.models.nn.comp\_gcn*), 88  
 LinkPrediction (class in *cogdl.tasks.link\_prediction*), 123  
 LinkPredictRGCN (class in *cogdl.models.nn.rgcn*), 117  
 LinkPredLoss (class in *cogdl.models.nn.pyg\_diffpool*), 101  
 load\_from\_pretrained() (*cogdl.layers.strategies\_layers.GNNPred* method), 69  
 load\_gnn() (in module *cogdl.layers.gpt\_gnn\_module*), 61  
 log\_metrics() (in module *cogdl.tasks.link\_prediction*), 123  
 LogReg (class in *cogdl.models.nn.dgi*), 89

- LogRegTrainer (class in cogdl.models.nn.dgi), 89
- loss () (cogdl.models.nn.compgcn.LinkPredictCompGCN method), 88
- loss () (cogdl.models.nn.disengcn.DisenGCN method), 92
- loss () (cogdl.models.nn.gat.PetarVSpGAT method), 93
- loss () (cogdl.models.nn.gcn.TKipfGCN method), 94
- loss () (cogdl.models.nn.gcnii.GCNII method), 95
- loss () (cogdl.models.nn.graphsage.Graphsage method), 96
- loss () (cogdl.models.nn.mixhop.MixHop method), 96
- loss () (cogdl.models.nn.mlp.MLP method), 96
- loss () (cogdl.models.nn.pyg\_cheb.Chebyshev method), 99
- loss () (cogdl.models.nn.pyg\_deepergcn.DeeperGCN method), 100
- loss () (cogdl.models.nn.pyg\_diffpool.DiffPool method), 103
- loss () (cogdl.models.nn.pyg\_drgat.DrGAT method), 104
- loss () (cogdl.models.nn.pyg\_drgcn.DrGCN method), 104
- loss () (cogdl.models.nn.pyg\_gat.GAT method), 105
- loss () (cogdl.models.nn.pyg\_gcn.GCN method), 105
- loss () (cogdl.models.nn.pyg\_gcnmix.BaseGNNMix method), 106
- loss () (cogdl.models.nn.pyg\_gcnmix.GCNMix method), 106
- loss () (cogdl.models.nn.pyg\_gin.GIN method), 108
- loss () (cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN method), 108
- loss () (cogdl.models.nn.pyg\_grand.Grand method), 109
- loss () (cogdl.models.nn.pyg\_gtn.GTN method), 110
- loss () (cogdl.models.nn.pyg\_han.HAN method), 111
- loss () (cogdl.models.nn.pyg\_infomax.Infomax method), 113
- loss () (cogdl.models.nn.pyg\_srgcn.SRGCN method), 115
- loss () (cogdl.models.nn.pyg\_unet.UNet method), 116
- loss () (cogdl.models.nn.pyg\_unsup\_graphsage.SAGE method), 116
- loss () (cogdl.models.nn.rgcn.LinkPredictRGCN method), 118
- loss () (cogdl.models.supervised\_model.SupervisedHeterogeneousNodeClassificationModel method), 119
- loss () (cogdl.models.supervised\_model.SupervisedHomogeneousNodeClassificationModel method), 119
- loss () (cogdl.models.supervised\_model.SupervisedModel method), 119
- loss () (cogdl.trainers.deepergcn\_trainer.DeeperGCNTrainer method), 129
- ## M
- makedirs () (in module cogdl.data.makedirs), 29
- MaskAtom (class in cogdl.datasets.pyg\_strategies\_data), 51
- MaskEdge (class in cogdl.datasets.pyg\_strategies\_data), 51
- MaskTrainer (class in cogdl.layers.strategies\_layers), 70
- Matcher (class in cogdl.layers.gpt\_gnn\_module), 62
- MatlabMatrix (class in cogdl.datasets.matlab\_matrix), 45
- maybe\_log () (in module cogdl.data.extract), 28
- mean\_reciprocal\_rank () (in module cogdl.layers.gpt\_gnn\_module), 60
- MeanAggregator (class in cogdl.layers), 71
- MeanAggregator (class in cogdl.layers.maggregator), 64
- message () (cogdl.layers.gpt\_gnn\_module.HGTConv method), 61
- message\_func () (cogdl.layers.gcc\_module.GATLayer method), 57
- message\_norm () (cogdl.models.nn.pyg\_deepergcn.GENConv method), 99
- message\_passing () (cogdl.models.nn.compgcn.CompGCNLayer method), 87
- Metapath2vec (class in cogdl.models.emb.metapath2vec), 80
- mi\_loss () (cogdl.models.nn.pyg\_infograph.InfoGraph static method), 112
- mix\_hidden\_state () (in module cogdl.models.nn.pyg\_gcnmix), 106
- MixHop (class in cogdl.models.nn.mixhop), 96
- MixHopLayer (class in cogdl.layers), 71
- MixHopLayer (class in cogdl.layers.mixhop\_layer), 65
- MLP (class in cogdl.layers.gcc\_module), 57
- MLP (class in cogdl.models.nn.mlp), 96
- MLPLayer (class in cogdl.models.nn.pyg\_grand), 109
- Model (class in cogdl.models.nn.mvgrl), 97
- model\_name (in module cogdl.models), 120
- MODEL\_REGISTRY (in module cogdl.models), 120
- module
- cogdl, 23
  - cogdl.data, 23
  - cogdl.data.batch, 23
  - cogdl.data.data, 24
  - cogdl.data.data\_loader, 26
  - cogdl.data.dataset, 27
  - cogdl.data.download, 28
  - cogdl.data.extract, 28
  - cogdl.data.makedirs, 29
  - cogdl.data.sampler, 29
  - cogdl.datasets, 35
  - cogdl.datasets.dgl\_data, 35

- `cogdl.datasets.gatne`, 36
- `cogdl.datasets.gcc_data`, 37
- `cogdl.datasets.gtn_data`, 38
- `cogdl.datasets.han_data`, 40
- `cogdl.datasets.kg_data`, 41
- `cogdl.datasets.matlab_matrix`, 45
- `cogdl.datasets.pyg`, 46
- `cogdl.datasets.pyg_ogb`, 48
- `cogdl.datasets.pyg_strategies_data`, 50
- `cogdl.layers`, 57
- `cogdl.layers.gcc_module`, 57
- `cogdl.layers.gpt_gnn_module`, 59
- `cogdl.layers.link_prediction_module`, 63
- `cogdl.layers.maggregator`, 64
- `cogdl.layers.mixhop_layer`, 65
- `cogdl.layers.prone_module`, 65
- `cogdl.layers.se_layer`, 67
- `cogdl.layers.srgcn_module`, 67
- `cogdl.layers.strategies_layers`, 69
- `cogdl.models`, 71
- `cogdl.models.base_model`, 118
- `cogdl.models.emb`, 71
- `cogdl.models.emb.complex`, 71
- `cogdl.models.emb.deepwalk`, 72
- `cogdl.models.emb.dgk`, 73
- `cogdl.models.emb.distmult`, 73
- `cogdl.models.emb.dngr`, 74
- `cogdl.models.emb.gatne`, 74
- `cogdl.models.emb.graph2vec`, 76
- `cogdl.models.emb.grarep`, 77
- `cogdl.models.emb.hin2vec`, 77
- `cogdl.models.emb.hope`, 78
- `cogdl.models.emb.knowledge_base`, 79
- `cogdl.models.emb.line`, 79
- `cogdl.models.emb.metapath2vec`, 80
- `cogdl.models.emb.netmf`, 81
- `cogdl.models.emb.netsmf`, 81
- `cogdl.models.emb.node2vec`, 82
- `cogdl.models.emb.prone`, 83
- `cogdl.models.emb.pte`, 83
- `cogdl.models.emb.rotate`, 84
- `cogdl.models.emb.sdne`, 84
- `cogdl.models.emb.spectral`, 85
- `cogdl.models.emb.transe`, 85
- `cogdl.models.nn`, 86
- `cogdl.models.nn.asgcn`, 86
- `cogdl.models.nn.compvcn`, 87
- `cogdl.models.nn.dgi`, 88
- `cogdl.models.nn.dgl_gcc`, 90
- `cogdl.models.nn.disengcn`, 91
- `cogdl.models.nn.fastgcn`, 92
- `cogdl.models.nn.gat`, 93
- `cogdl.models.nn.gcn`, 94
- `cogdl.models.nn.gcnii`, 94
- `cogdl.models.nn.graphsage`, 95
- `cogdl.models.nn.mixhop`, 96
- `cogdl.models.nn.mlp`, 96
- `cogdl.models.nn.mvgrl`, 97
- `cogdl.models.nn.patchy_san`, 98
- `cogdl.models.nn.pyg_cheb`, 99
- `cogdl.models.nn.pyg_deepergcn`, 99
- `cogdl.models.nn.pyg_dgcnn`, 100
- `cogdl.models.nn.pyg_diffpool`, 101
- `cogdl.models.nn.pyg_drgat`, 103
- `cogdl.models.nn.pyg_drgcn`, 104
- `cogdl.models.nn.pyg_gat`, 104
- `cogdl.models.nn.pyg_gcn`, 105
- `cogdl.models.nn.pyg_gcnmix`, 105
- `cogdl.models.nn.pyg_gin`, 107
- `cogdl.models.nn.pyg_gpt_gnn`, 108
- `cogdl.models.nn.pyg_grand`, 109
- `cogdl.models.nn.pyg_gtn`, 110
- `cogdl.models.nn.pyg_han`, 110
- `cogdl.models.nn.pyg_infograph`, 111
- `cogdl.models.nn.pyg_infomax`, 113
- `cogdl.models.nn.pyg_sortpool`, 113
- `cogdl.models.nn.pyg_srgcn`, 114
- `cogdl.models.nn.pyg_stpgnn`, 115
- `cogdl.models.nn.pyg_unet`, 116
- `cogdl.models.nn.pyg_unsup_graphsage`, 116
- `cogdl.models.nn.rgcn`, 117
- `cogdl.models.supervised_model`, 118
- `cogdl.options`, 131
- `cogdl.tasks`, 120
- `cogdl.tasks.base_task`, 120
- `cogdl.tasks.graph_classification`, 121
- `cogdl.tasks.heterogeneous_node_classification`, 121
- `cogdl.tasks.link_prediction`, 122
- `cogdl.tasks.multiplex_link_prediction`, 123
- `cogdl.tasks.multiplex_node_classification`, 124
- `cogdl.tasks.node_classification`, 124
- `cogdl.tasks.node_classification_sampling`, 125
- `cogdl.tasks.pretrain`, 126
- `cogdl.tasks.unsupervised_graph_classification`, 126
- `cogdl.tasks.unsupervised_node_classification`, 126
- `cogdl.trainers`, 128
- `cogdl.trainers.base_trainer`, 128



- cogdl.trainers.deepergcn\_trainer, 128  
 cogdl.trainers.gpt\_gnn\_trainer, 129  
 cogdl.trainers.sampled\_trainer, 130  
 cogdl.trainers.supervised\_trainer, 130  
 cogdl.trainers.unsupervised\_trainer, 131  
 cogdl.utils, 132  
 MoleculeDataset (class in cogdl.datasets.pyg\_strategies\_data), 54  
 MRWSampler (class in cogdl.data.sampler), 30  
 mul\_edge\_softmax() (in module cogdl.utils), 133  
 MultiplexLinkPrediction (class in cogdl.tasks.multiplex\_link\_prediction), 124  
 MultiplexNodeClassification (class in cogdl.tasks.multiplex\_node\_classification), 124  
 MUTAGDataset (class in cogdl.datasets.dgl\_data), 35  
 MUTAGDataset (class in cogdl.datasets.pyg), 47  
 MVGRL (class in cogdl.models.nn.mvgrl), 97
- ## N
- NCT109Dataset (class in cogdl.datasets.pyg), 48  
 NCT1Dataset (class in cogdl.datasets.pyg), 48  
 ndcg\_at\_k() (in module cogdl.layers.gpt\_gnn\_module), 60  
 neg\_sample() (cogdl.layers.gpt\_gnn\_module.GPT\_GNN method), 62  
 NegativeEdge (class in cogdl.datasets.pyg\_strategies\_data), 51  
 NetMF (class in cogdl.models.emb.netmf), 81  
 NetSMF (class in cogdl.models.emb.netsmf), 81  
 Node2vec (class in cogdl.models.emb.node2vec), 82  
 node\_classification\_sample() (in module cogdl.trainers.gpt\_gnn\_trainer), 129  
 node\_degree\_as\_feature() (in module cogdl.tasks.graph\_classification), 121  
 node\_feature (cogdl.layers.gpt\_gnn\_module.Graph attribute), 61  
 node\_selection\_with\_ld\_wl() (in module cogdl.models.nn.patchy\_san), 98  
 NodeAdaptiveEncoder (class in cogdl.layers.prone\_module), 66  
 NodeAdaptiveEncoder (class in cogdl.models.nn.pyg\_srgcn), 115  
 NodeAttention (class in cogdl.layers.srgcn\_module), 68  
 NodeClassification (class in cogdl.tasks.node\_classification), 124  
 NodeClassificationDataset (class in cogdl.models.nn.dgl\_gcc), 90  
 NodeClassificationSampling (class in cogdl.tasks.node\_classification\_sampling), 125  
 NodeSampler (class in cogdl.data.sampler), 30  
 norm() (cogdl.layers.maggregator.MeanAggregator static method), 64  
 norm() (cogdl.layers.MeanAggregator static method), 71  
 norm() (cogdl.models.nn.pyg\_gtn.GTN method), 110  
 normalization() (cogdl.models.nn.pyg\_gtn.GTN method), 110  
 normalize() (in module cogdl.layers.gpt\_gnn\_module), 60  
 normalize\_adj() (cogdl.models.nn.pyg\_grand.Grand method), 109  
 normalize\_adj() (in module cogdl.models.nn.dgi), 89  
 normalize\_adj() (in module cogdl.models.nn.mvgrl), 97  
 normalize\_feature() (in module cogdl.datasets.pyg), 47  
 normalize\_x() (cogdl.models.nn.pyg\_grand.Grand method), 109  
 NormIdentity (class in cogdl.layers.srgcn\_module), 68  
 NSLoss (class in cogdl.models.emb.gatne), 75  
 num\_edges() (cogdl.data.Data property), 32  
 num\_edges() (cogdl.data.data.Data property), 25  
 num\_entities() (cogdl.datasets.kg\_data.KnowledgeGraphDataset property), 43  
 num\_features() (cogdl.data.Data property), 32  
 num\_features() (cogdl.data.data.Data property), 25  
 num\_features() (cogdl.data.Dataset property), 33  
 num\_features() (cogdl.data.dataset.Dataset property), 28  
 num\_features() (cogdl.datasets.Dataset property), 56  
 num\_graphs() (cogdl.data.Batch property), 33  
 num\_graphs() (cogdl.data.batch.Batch property), 24  
 num\_graphs() (cogdl.datasets.pyg\_strategies\_data.BatchAE property), 53  
 num\_graphs() (cogdl.datasets.pyg\_strategies\_data.BatchFinetune property), 52  
 num\_graphs() (cogdl.datasets.pyg\_strategies\_data.BatchMasking property), 53  
 num\_graphs() (cogdl.datasets.pyg\_strategies\_data.BatchSubstructCont property), 53  
 num\_nodes() (cogdl.data.Data property), 32  
 num\_nodes() (cogdl.data.data.Data property), 25  
 num\_relations() (cogdl.datasets.kg\_data.KnowledgeGraphDataset property), 43  
 nx\_to\_graph\_data\_obj() (in module cogdl.datasets.pyg\_strategies\_data), 51  
 nx\_to\_graph\_data\_obj\_simple() (in module cogdl.datasets.pyg\_strategies\_data), 51

## O

- OGBArxivDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBCodeDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBGDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBMolbaseDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBMolhivDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBMolpcbaDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBNDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBPapers100MDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBPpaDataset (class in cogdl.datasets.pyg\_ogb), 49  
 OGBProductsDataset (class in cogdl.datasets.pyg\_ogb), 49  
 one\_dim\_wl () (in module cogdl.models.nn.patchy\_san), 98  
 one\_shot\_iterator () (cogdl.datasets.kg\_data.BidirectionalOneShotIterator static method), 42
- P**  
 parse\_args\_and\_arch () (in module cogdl.options), 132  
 PatchySAN (class in cogdl.models.nn.patchy\_san), 98  
 PetarVSpGAT (class in cogdl.models.nn.gat), 93  
 pool () (cogdl.layers.strategies\_layers.GNNPred method), 70  
 PPIDataset (class in cogdl.datasets.matlab\_matrix), 46  
 PPR (class in cogdl.layers.prone\_module), 66  
 PPR (class in cogdl.layers.srgcn\_module), 68  
 predict () (cogdl.layers.link\_prediction\_module.ConvELayer method), 63  
 predict () (cogdl.layers.link\_prediction\_module.DistMultiLayer method), 63  
 predict () (cogdl.models.nn.compgcn.LinkPredictCompGCN method), 88  
 predict () (cogdl.models.nn.disengcn.DisenGCN method), 92  
 predict () (cogdl.models.nn.gat.PetarVSpGAT method), 93  
 predict () (cogdl.models.nn.gcn.TKipfGCN method), 94  
 predict () (cogdl.models.nn.gcnii.GCNII method), 95  
 predict () (cogdl.models.nn.graphsage.Graphsage method), 96  
 predict () (cogdl.models.nn.mixhop.MixHop method), 96  
 predict () (cogdl.models.nn.mlp.MLP method), 96  
 predict () (cogdl.models.nn.pyg\_cheb.Chebyshev method), 99  
 predict () (cogdl.models.nn.pyg\_deepergcn.DeeperGCN method), 100  
 predict () (cogdl.models.nn.pyg\_drgat.DrGAT method), 104  
 predict () (cogdl.models.nn.pyg\_drgcn.DrGCN method), 104  
 predict () (cogdl.models.nn.pyg\_gat.GAT method), 105  
 predict () (cogdl.models.nn.pyg\_gcn.GCN method), 105  
 predict () (cogdl.models.nn.pyg\_gcnmix.GCNMix method), 106  
 predict () (cogdl.models.nn.pyg\_gpt\_gnn.GPT\_GNN method), 108  
 predict () (cogdl.models.nn.pyg\_grand.Grand method), 109  
 predict () (cogdl.models.nn.pyg\_infomax.Infomax method), 113  
 predict () (cogdl.models.nn.pyg\_srgcn.SRGCN method), 115  
 predict () (cogdl.models.nn.pyg\_unet.UNet method), 116  
 predict () (cogdl.models.nn.rgcn.LinkPredictRGCN method), 118  
 predict () (cogdl.models.supervised\_model.SupervisedHomogeneousNo method), 119  
 predict () (cogdl.tasks.unsupervised\_node\_classification.TopKRanker method), 127  
 predict () (cogdl.trainers.deepergcn\_trainer.DeeperGCNTrainer method), 129  
 predict () (cogdl.trainers.supervised\_trainer.SupervisedTrainer method), 131  
 predict\_noise () (cogdl.models.nn.pyg\_gcnmix.BaseGNNMix method), 106  
 prepare\_data () (in module cogdl.trainers.gpt\_gnn\_trainer), 129  
 preprocess\_dataset () (in module cogdl.layers.gpt\_gnn\_module), 62  
 preprocess\_features () (in module cogdl.models.nn.dgi), 89  
 preprocess\_features () (in module cogdl.models.nn.mvgrl), 97  
 Pretrainer (class in cogdl.layers.strategies\_layers), 70  
 PretrainTask (class in cogdl.tasks.pretrain), 126  
 print\_result () (in module cogdl.utils), 134  
 process () (cogdl.data.Dataset method), 33  
 process () (cogdl.data.dataset.Dataset method), 27  
 process () (cogdl.datasets.Dataset method), 56  
 process () (cogdl.datasets.gatne.GatneDataset method), 36  
 process () (cogdl.datasets.gcc\_data.Edgelist method),

38  
 process () (cogdl.datasets.gtn\_data.GTNDataset method), 39  
 process () (cogdl.datasets.han\_data.HANDataset method), 41  
 process () (cogdl.datasets.kg\_data.KnowledgeGraphDataset method), 43  
 process () (cogdl.datasets.matlab\_matrix.MatlabMatrix method), 46  
 process () (cogdl.datasets.pyg\_strategies\_data.BACEDataset method), 54  
 process () (cogdl.datasets.pyg\_strategies\_data.BBBPDataset method), 55  
 process () (cogdl.datasets.pyg\_strategies\_data.BioDataset method), 54  
 process () (cogdl.datasets.pyg\_strategies\_data.MoleculeDataset method), 54  
 processed\_file\_names () (cogdl.data.Dataset property), 33  
 processed\_file\_names () (cogdl.data.dataset.Dataset property), 27  
 processed\_file\_names () (cogdl.datasets.Dataset property), 55  
 processed\_file\_names () (cogdl.datasets.gatne.GatneDataset property), 36  
 processed\_file\_names () (cogdl.datasets.gcc\_data.Edgelist property), 38  
 processed\_file\_names () (cogdl.datasets.gtn\_data.GTNDataset property), 39  
 processed\_file\_names () (cogdl.datasets.han\_data.HANDataset property), 40  
 processed\_file\_names () (cogdl.datasets.kg\_data.KnowledgeGraphDataset property), 43  
 processed\_file\_names () (cogdl.datasets.matlab\_matrix.MatlabMatrix property), 45  
 processed\_file\_names () (cogdl.datasets.pyg\_strategies\_data.BACEDataset property), 54  
 processed\_file\_names () (cogdl.datasets.pyg\_strategies\_data.BBBPDataset property), 55  
 processed\_file\_names () (cogdl.datasets.pyg\_strategies\_data.BioDataset property), 54  
 processed\_file\_names () (cogdl.datasets.pyg\_strategies\_data.MoleculeDataset property), 54  
 processed\_paths () (cogdl.data.Dataset property), 33  
 processed\_paths () (cogdl.data.dataset.Dataset property), 28  
 processed\_paths () (cogdl.datasets.Dataset property), 56  
 PRONE (class in cogdl.layers.prone\_module), 66  
 PRONE (class in cogdl.models.emb.prone), 83  
 prop () (cogdl.layers.prone\_module.Gaussian method), 66  
 prop () (cogdl.layers.prone\_module.HeatKernel method), 66  
 prop () (cogdl.layers.prone\_module.HeatKernelApproximation method), 66  
 prop () (cogdl.layers.prone\_module.NodeAdaptiveEncoder static method), 66  
 prop () (cogdl.layers.prone\_module.PPR method), 66  
 prop () (cogdl.layers.prone\_module.SignalRescaling method), 66  
 prop\_adjacency () (cogdl.layers.prone\_module.HeatKernel method), 66  
 propagate () (in module cogdl.layers.prone\_module), 66  
 ProtainsDataset (class in cogdl.datasets.dgl\_data), 35  
 ProtainsDataset (class in cogdl.datasets.pyg), 48  
 PTCMRDataset (class in cogdl.datasets.pyg), 48  
 PTE (class in cogdl.models.emb.pte), 83  
 PubMedDataset (class in cogdl.datasets.pyg), 47  
 pyg (in module cogdl.datasets), 56  
 pyg (in module cogdl.models), 120  
 pyg (in module cogdl.tasks.unsupervised\_node\_classification), 126

## Q

QM9Dataset (class in cogdl.datasets.pyg), 48

## R

rand\_prop () (cogdl.models.nn.pyg\_grand.Grand method), 109  
 randint () (in module cogdl.layers.gpt\_gnn\_module), 60  
 random\_partition\_graph () (in module cogdl.trainers.deepergcn\_trainer), 128  
 random\_surfing () (cogdl.models.emb.dngr.DNGR method), 74  
 randomly\_choose\_false\_edges () (in module cogdl.tasks.link\_prediction), 123  
 raw\_file\_names () (cogdl.data.Dataset property), 33  
 raw\_file\_names () (cogdl.data.dataset.Dataset property), 27  
 raw\_file\_names () (cogdl.datasets.Dataset property), 55  
 raw\_file\_names () (cogdl.datasets.gatne.GatneDataset property), 36

`raw_file_names()` (*cogdl.datasets.gcc\_data.Edgelist* *property*), 38  
`raw_file_names()` (*cogdl.layers.strategies\_layers.Discriminator* *method*), 70  
`raw_file_names()` (*cogdl.datasets.gtn\_data.GTNDataset* *property*), 39  
`reset_parameters()` (*cogdl.models.emb.gatne.GATNEModel* *method*), 75  
`raw_file_names()` (*cogdl.datasets.han\_data.HANDataset* *property*), 40  
`reset_parameters()` (*cogdl.models.emb.gatne.NSLoss* *method*), 75  
`raw_file_names()` (*cogdl.datasets.kg\_data.KnowledgeGraphDataset* *property*), 42  
`reset_parameters()` (*cogdl.models.nn.asgcn.ASGCN* *method*), 86  
`raw_file_names()` (*cogdl.datasets.matlab\_matrix.MatlabMatrix* *property*), 45  
`reset_parameters()` (*cogdl.models.nn.asgcn.ASGCN* *method*), 86  
`raw_file_names()` (*cogdl.datasets.pyg\_strategies\_data.BACEDataset* *property*), 54  
`reset_parameters()` (*cogdl.models.nn.asgcn.GraphConvolution* *method*), 86  
`raw_file_names()` (*cogdl.datasets.pyg\_strategies\_data.BBBPDataset* *property*), 55  
`reset_parameters()` (*cogdl.models.nn.compvcn.BasesRelEmbLayer* *method*), 87  
`raw_file_names()` (*cogdl.datasets.pyg\_strategies\_data.BioDataset* *property*), 54  
`reset_parameters()` (*cogdl.models.nn.compvcn.BasesRelEmbLayer* *method*), 87  
`raw_file_names()` (*cogdl.datasets.pyg\_strategies\_data.MolecularDataset* *property*), 54  
`reset_parameters()` (*cogdl.models.nn.disengcn.DisenGCN* *method*), 91  
`raw_paths()` (*cogdl.data.Dataset* *property*), 33  
`reset_parameters()` (*cogdl.models.nn.disengcn.DisenGCNLayer* *method*), 91  
`raw_paths()` (*cogdl.data.dataset.Dataset* *property*), 28  
`reset_parameters()` (*cogdl.models.nn.disengcn.DisenGCNLayer* *method*), 91  
`read_gatne_data()` (*in module cogdl.datasets.gatne*), 36  
`reset_parameters()` (*cogdl.models.nn.fastgcn.GraphConvolution* *method*), 92  
`read_gtn_data()` (*cogdl.datasets.gtn\_data.GTNDataset* *method*), 39  
`reset_parameters()` (*cogdl.models.nn.gcn.GraphConvolution* *method*), 94  
`read_gtn_data()` (*cogdl.datasets.han\_data.HANDataset* *method*), 40  
`reset_parameters()` (*cogdl.models.nn.gcnii.GCNIIlayer* *method*), 94  
`read_triplet_data()` (*in module cogdl.datasets.kg\_data*), 42  
`reset_parameters()` (*cogdl.models.nn.pyg\_diffpool.DiffPool* *method*), 103  
`RedditBinary` (*class in cogdl.datasets.pyg*), 48  
`RedditDataset` (*class in cogdl.datasets.pyg*), 47  
`RedditMulti12K` (*class in cogdl.datasets.pyg*), 48  
`RedditMulti5K` (*class in cogdl.datasets.pyg*), 48  
`reduce_func()` (*cogdl.layers.gcc\_module.GATLayer* *method*), 57  
`reset_parameters()` (*cogdl.models.nn.pyg\_grand.MLPLayer* *method*), 109  
`register_dataset()` (*in module cogdl.datasets*), 56  
`register_model()` (*in module cogdl.models*), 120  
`register_task()` (*in module cogdl.tasks*), 127  
`reset_parameters()` (*cogdl.models.nn.pyg\_gtn.GTConv* *method*), 110  
`regularation()` (*cogdl.models.emb.hin2vec.Hin2vec\_layer* *method*), 77  
`rel_transform()` (*cogdl.models.nn.compvcn.CompGCNLayer* *method*), 87  
`reset_parameters()` (*cogdl.models.nn.pyg\_infograph.InfoGraph* *method*), 112  
`RelTemporalEncoding` (*class in cogdl.layers.gpt\_gnn\_module*), 61  
`reset_parameters()` (*cogdl.models.nn.rgcn.RGCNLayer* *method*), 117  
`remove_self_loops()` (*in module cogdl.utils*), 134  
`reset_idxes()` (*in module cogdl.datasets.pyg\_strategies\_data*), 52  
`RGCN` (*class in cogdl.models.nn.rgcn*), 117  
`RGCNLayer` (*class in cogdl.models.nn.rgcn*), 117  
`RNNModel` (*class in cogdl.layers.gpt\_gnn\_module*), 62  
`RotateE` (*class in cogdl.models.emb.rotate*), 84  
`reset_parameters()` (*cogdl.layers.MixHopLayer* *method*), 65  
`reset_parameters()` (*cogdl.layers.MixHopLayer* *method*), 71  
`reset_parameters()` (*RowSoftmax* *class in cogdl.layers.srgcn\_module*), 68  
`reset_parameters()` (*RowUniform* *class in cogdl.layers.srgcn\_module*), 68



- RWGraph (class in *cogdl.models.emb.gatne*), 76
- RWgraph (class in *cogdl.models.emb.hin2vec*), 77
- RWSampler (class in *cogdl.data.sampler*), 30
- ## S
- SAGE (class in *cogdl.models.nn.pyg\_unsup\_graphsage*), 116
- sage\_sampler() (in module *cogdl.models.nn.graphsage*), 95
- SAINTSampler (class in *cogdl.data.sampler*), 30
- SAINTTrainer (class in *cogdl.trainers.sampled\_trainer*), 130
- sample() (*cogdl.data.sampler.EdgeSampler* method), 30
- sample() (*cogdl.data.sampler.MRWSampler* method), 30
- sample() (*cogdl.data.sampler.NodeSampler* method), 30
- sample() (*cogdl.data.sampler.RWSampler* method), 30
- sample() (*cogdl.data.sampler.SAINTSampler* method), 30
- sample() (*cogdl.data.sampler.Sampler* method), 30
- sample\_mask() (in module *cogdl.datasets.han\_data*), 40
- sample\_subgraph() (in module *cogdl.layers.gpt\_gnn\_module*), 61
- SampledTrainer (class in *cogdl.trainers.sampled\_trainer*), 130
- Sampler (class in *cogdl.data.sampler*), 29
- sampler\_from\_args() (*cogdl.trainers.sampled\_trainer.SAINTTrainer* method), 130
- sampling() (*cogdl.models.nn.asgcn.ASGCN* method), 86
- sampling() (*cogdl.models.nn.fastgcn.FastGCN* method), 92
- sampling() (*cogdl.models.nn.graphsage.Graphsage* method), 95
- sampling() (*cogdl.models.nn.pyg\_unsup\_graphsage.SAGE* method), 116
- sampling\_edge\_uniform() (in module *cogdl.layers.link\_prediction\_module*), 64
- save\_emb() (*cogdl.tasks.unsupervised\_graph\_classification.UnsupervisedGraphClassification* method), 126
- save\_emb() (*cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification* method), 127
- save\_embedding() (*cogdl.models.emb.dgk.DeepGraphKernel* method), 73
- save\_embedding() (*cogdl.models.emb.graph2vec.Graph2Vec* method), 76
- save\_model() (in module *cogdl.tasks.link\_prediction*), 122
- scale\_matrix() (*cogdl.models.emb.dngr.DNGR* method), 74
- scatter\_sum() (in module *cogdl.models.nn.pyg\_sortpool*), 114
- score() (*cogdl.models.emb.complex.Complex* method), 72
- score() (*cogdl.models.emb.distmult.DistMult* method), 73
- score() (*cogdl.models.emb.knowledge\_base.KGEModel* method), 79
- score() (*cogdl.models.emb.rotate.RotatE* method), 84
- score() (*cogdl.models.emb.transe.TransE* method), 86
- SDNE (class in *cogdl.models.emb.sdne*), 84
- SDNE\_layer (class in *cogdl.models.emb.sdne*), 84
- SELayer (class in *cogdl.layers*), 71
- SELayer (class in *cogdl.layers.gcc\_module*), 57
- SELayer (class in *cogdl.layers.se\_layer*), 67
- select\_task() (in module *cogdl.tasks.link\_prediction*), 123
- set\_adj() (*cogdl.models.nn.asgcn.ASGCN* method), 86
- set\_adj() (*cogdl.models.nn.fastgcn.FastGCN* method), 92
- set\_logger() (in module *cogdl.tasks.link\_prediction*), 122
- set\_random\_seed() (in module *cogdl.utils*), 134
- sharpen() (in module *cogdl.models.nn.pyg\_gcnmix*), 106
- SignalRescaling (class in *cogdl.layers.prone\_module*), 66
- simulate\_walks() (*cogdl.models.emb.gatne.RWGraph* method), 76
- SortPool (class in *cogdl.models.nn.pyg\_sortpool*), 114
- spare2dense\_batch() (in module *cogdl.models.nn.pyg\_sortpool*), 114
- sparse\_mx\_to\_torch\_sparse\_tensor() (in module *cogdl.layers.gpt\_gnn\_module*), 60
- sparse\_mx\_to\_torch\_sparse\_tensor() (in module *cogdl.models.nn.dgi*), 89
- sparse\_mx\_to\_torch\_sparse\_tensor() (in module *cogdl.models.nn.mvgrl*), 97
- SpecialSpmm (class in *cogdl.models.nn.gat*), 93
- SpecialSpmmFunction (class in *cogdl.models.nn.gat*), 93
- Spectral (class in *cogdl.models.emb.spectral*), 85
- SpGraphAttentionLayer (class in *cogdl.models.nn.gat*), 93
- split\_data() (*cogdl.layers.strategies\_layers.Finetuner* method), 70
- split\_data() (*cogdl.layers.strategies\_layers.SupervisedTrainer* method), 70
- split\_dataset() (*cogdl.models.nn.patchy\_san.PatchySAN* class method), 98
- split\_dataset() (*cogdl.models.nn.pyg\_dgcnn.DGCNN* class method), 101
- split\_dataset() (*cogdl.models.nn.pyg\_diffpool.DiffPool*

*class method*), 103  
 split\_dataset() (*cogdl.models.nn.pyg\_gin.GIN class method*), 108  
 split\_dataset() (*cogdl.models.nn.pyg\_infograph.InfoGraph class method*), 112  
 split\_dataset() (*cogdl.models.nn.pyg\_sortpool.SortPool class method*), 114  
 spmm() (*in module cogdl.utils*), 133  
 spmm\_adj() (*in module cogdl.utils*), 133  
 SRGCN (*class in cogdl.models.nn.pyg\_srgcn*), 115  
 SrgcnHead (*class in cogdl.models.nn.pyg\_srgcn*), 115  
 SrgcnSoftmaxHead (*class in cogdl.models.nn.pyg\_srgcn*), 115  
 stpgnn (*class in cogdl.models.nn.pyg\_stpgnn*), 115  
 sup\_forward() (*cogdl.models.nn.pyg\_infograph.InfoGraph method*), 112  
 sup\_loss() (*cogdl.models.nn.pyg\_infograph.InfoGraph method*), 112  
 SUPEncoder (*class in cogdl.models.nn.pyg\_infograph*), 111  
 SupervisedHeterogeneousNodeClassificationModel (*class in cogdl.models.supervised\_model*), 119  
 SupervisedHeterogeneousNodeClassificationTrainer (*class in cogdl.trainers.supervised\_trainer*), 131  
 SupervisedHomogeneousNodeClassificationModel (*class in cogdl.models.supervised\_model*), 119  
 SupervisedHomogeneousNodeClassificationTrainer (*class in cogdl.trainers.supervised\_trainer*), 131  
 SupervisedModel (*class in cogdl.models.supervised\_model*), 118  
 SupervisedTrainer (*class in cogdl.layers.strategies\_layers*), 70  
 SupervisedTrainer (*class in cogdl.trainers.supervised\_trainer*), 131  
 symmetric\_normalization() (*in module cogdl.utils*), 133  
 SymmetryNorm (*class in cogdl.layers.srgcn\_module*), 68

**T**

tabulate\_results() (*in module cogdl.utils*), 134  
 task\_name (*in module cogdl.tasks*), 128  
 TASK\_REGISTRY (*in module cogdl.tasks*), 127  
 taylor() (*cogdl.layers.prone\_module.HeatKernelApproximation method*), 66  
 test\_gpu\_volume() (*cogdl.trainers.deepergcn\_trainer.DeeperGCNTrainer method*), 129  
 test\_moco() (*in module cogdl.models.nn.dgl\_gcc*), 90  
 test\_start\_idx() (*cogdl.datasets.kg\_data.KnowledgeGraphDataset property*), 43  
 test\_step() (*cogdl.models.emb.knowledge\_base.KGEModel static method*), 79  
 TestBioDataset (*class in cogdl.datasets.pyg\_strategies\_data*), 53  
 TestChemDataset (*class in cogdl.datasets.pyg\_strategies\_data*), 54  
 TestDataset (*class in cogdl.datasets.kg\_data*), 42  
 text\_loss() (*cogdl.layers.gpt\_gnn\_module.GPT\_GNN method*), 62  
 TKipfGCN (*class in cogdl.models.nn.gcn*), 94  
 to() (*cogdl.data.Data method*), 32  
 to() (*cogdl.data.data.Data method*), 25  
 to\_data\_list() (*cogdl.data.Batch method*), 33  
 to\_data\_list() (*cogdl.data.batch.Batch method*), 24  
 to\_list() (*in module cogdl.data.dataset*), 27  
 to\_torch() (*in cogdl.layers.gpt\_gnn\_module*), 61  
 toBatchedGraph() (*in cogdl.models.nn.pyg\_diffpool*), 103  
 TopKRanker (*class in cogdl.tasks.unsupervised\_node\_classification*), 127  
 train() (*cogdl.models.emb.deepwalk.DeepWalk method*), 72  
 train() (*cogdl.models.emb.dngr.DNGR method*), 74  
 train() (*cogdl.models.emb.gatne.GATNE method*), 75  
 train() (*cogdl.models.emb.grarep.GraRep method*), 77  
 train() (*cogdl.models.emb.hin2vec.Hin2vec method*), 78  
 train() (*cogdl.models.emb.hope.HOPE method*), 78  
 train() (*cogdl.models.emb.line.LINE method*), 80  
 train() (*cogdl.models.emb.metapath2vec.Metapath2vec method*), 80  
 train() (*cogdl.models.emb.netmf.NetMF method*), 81  
 train() (*cogdl.models.emb.netsmf.NetSMF method*), 82  
 train() (*cogdl.models.emb.node2vec.Node2vec method*), 82  
 train() (*cogdl.models.emb.prone.ProNE method*), 83  
 train() (*cogdl.models.emb.pte.PTE method*), 84  
 train() (*cogdl.models.emb.sдне.SDNE method*), 85  
 train() (*cogdl.models.emb.spectral.Spectral method*), 85  
 train() (*cogdl.models.nn.dgi.DGI method*), 89  
 train() (*cogdl.models.nn.dgi.LogRegTrainer method*), 89  
 train() (*cogdl.models.nn.dgl\_gcc.GCC method*), 91  
 train() (*cogdl.models.nn.mvgrl.MVGRL method*), 97  
 train() (*cogdl.models.nn.pyg\_unsup\_graphsage.Graphsage method*), 117  
 train() (*cogdl.tasks.base\_task.BaseTask method*), 120  
 train() (*cogdl.tasks.BaseTask method*), 127

train() (*cogdl.tasks.graph\_classification.GraphClassification* method), 121  
 train() (*cogdl.tasks.heterogeneous\_node\_classification.HeterogeneousNodeClassification* method), 122  
 train() (*cogdl.tasks.link\_prediction.HomoLinkPrediction* method), 123  
 train() (*cogdl.tasks.link\_prediction.KGLinkPrediction* method), 123  
 train() (*cogdl.tasks.link\_prediction.LinkPrediction* method), 123  
 train() (*cogdl.tasks.link\_prediction.TripleLinkPrediction* method), 123  
 train() (*cogdl.tasks.multiplex\_link\_prediction.MultiplexLinkPrediction* method), 124  
 train() (*cogdl.tasks.multiplex\_node\_classification.MultiplexNodeClassification* method), 124  
 train() (*cogdl.tasks.node\_classification.NodeClassification* method), 125  
 train() (*cogdl.tasks.node\_classification\_sampling.NodeClassificationSampling* method), 125  
 train() (*cogdl.tasks.pretrain.PretrainTask* method), 126  
 train() (*cogdl.tasks.unsupervised\_graph\_classification.UnsupervisedGraphClassification* method), 126  
 train() (*cogdl.tasks.unsupervised\_node\_classification.UnsupervisedNodeClassification* method), 127  
 train\_start\_idx() (*cogdl.datasets.kg\_data.KnowledgeGraphDataset* property), 43  
 train\_step() (*cogdl.models.emb.knowledge\_base.KGEMModel* static method), 79  
 TrainDataset (class in *cogdl.datasets.kg\_data*), 42  
 TransE (class in *cogdl.models.emb.transe*), 85  
 TripleLinkPrediction (class in *cogdl.tasks.link\_prediction*), 123  
 TwitterDataset (class in *cogdl.datasets.gatne*), 37

## U

UNet (class in *cogdl.models.nn.pyg\_unet*), 116  
 uniform\_node\_feature() (in module *cogdl.tasks.graph\_classification*), 121  
 unsup\_forward() (*cogdl.models.nn.pyg\_infograph.InfoGraph* method), 112  
 unsup\_loss() (*cogdl.models.nn.pyg\_infograph.InfoGraph* method), 112  
 unsup\_sup\_loss() (*cogdl.models.nn.pyg\_infograph.InfoGraph* method), 112  
 UnsupervisedGAT (class in *cogdl.layers.gcc\_module*), 57  
 UnsupervisedGIN (class in *cogdl.layers.gcc\_module*), 58  
 UnsupervisedGraphClassification (class in *cogdl.tasks.unsupervised\_graph\_classification*), 126

## V

url (*cogdl.datasets.gatne.GatneDataset* attribute), 36  
 url (*cogdl.datasets.gcc\_data.EdgeList* attribute), 38  
 url (*cogdl.datasets.kg\_data.KnowledgeGraphDataset* attribute), 45  
 url (*cogdl.datasets.kg\_data.KnowledgeGraphDataset* attribute), 45  
 USAAirportDataset (class in *cogdl.datasets.gcc\_data*), 38

## W

walk() (*cogdl.models.emb.gatne.RWGraph* method), 76  
 weights\_init() (*cogdl.models.nn.dgi.Discriminator* method), 89  
 weights\_init() (*cogdl.models.nn.dgi.GCN* method), 89  
 weights\_init() (*cogdl.models.nn.dgi.LogReg* method), 89  
 weights\_init() (*cogdl.models.nn.mvgrl.Discriminator* method), 97  
 WikipediaDataset (class in *cogdl.datasets.matlab\_matrix*), 46  
 wIterations() (*cogdl.models.emb.dgk.DeepGraphKernel* static method), 73  
 wl\_iterations() (*cogdl.models.emb.graph2vec.Graph2Vec* static method), 76  
 WN18Datset (class in *cogdl.datasets.kg\_data*), 44  
 WN18RRDataset (class in *cogdl.datasets.kg\_data*), 44

## Y

YouTubeDataset (class in *cogdl.datasets.gatne*), 37