Introduction

Welcome to this lab! At this lab, we will learn:

1. How to build a graph from a file or create a simple graph by ourself
2. Implement DeepWalk in the simplest way based on the paper DeepWalk. You can implement your own code or study some packages already made. I recommend reading Karateclub framwork, which is well-known for Unsupervised Learning on Graphs

Exercise

Download data and install packages

In [ ]:

```
!gdown --id "1vqsjGzGZnpCEgHliEsVmAvzm9_1h3L-Y&export=download"
!unrar x -Y "/content/lab1.rar" -d "/content/"
```

```
Downloading...
From: https://drive.google.com/uc?id=1vqsjGzGZnpCEgHliEsVmAvzm9_1h3L-Y&export=
download
To: /content/lab1.rar
100% 50.0k/50.0k [00:00<00:00, 30.9MB/s]

UNRAR 5.50 freeware      Copyright (c) 1993-2017 Alexander Roshal


Extracting from /content/lab1.rar

Extracting  /content/lab1_big_edgelist.txt                            99%
OK
Extracting  /content/lab1_small_edgelist.txt                          99%
OK
All OK
```

Build a graph

In [ ]:

```
import networkx as nx
import numpy as np
import torch
```

TO DO: Create graphs

Create a graph from the file: "lab1_small_edgelist.txt"

In [ ]:

```
# TO DO: Create a graph from the file: "lab1_small_edgelist.txt"
```

Create a graph by generating nodes and edges.

In [ ]:

```
# TO DO: Create a graph by generating nodes and edges.
```

Implement DeepWalk

Packages: Import necessary packages

In [ ]:

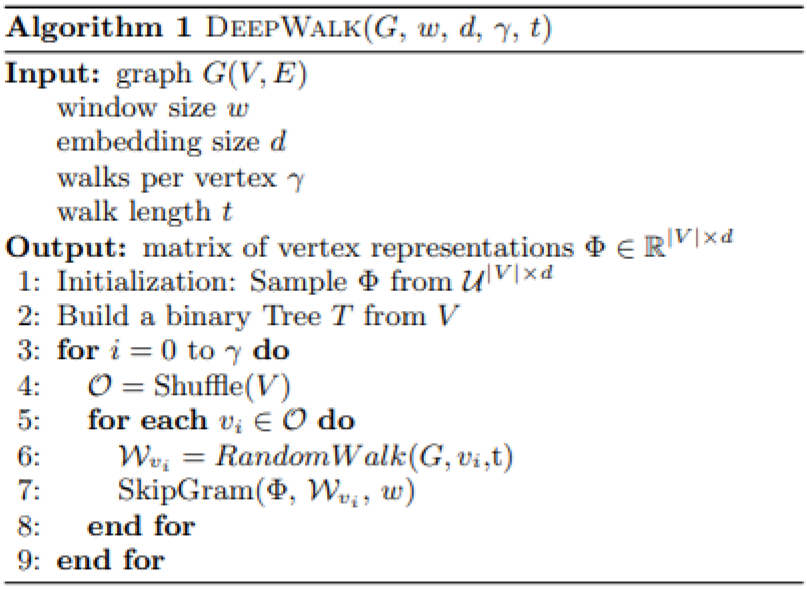
```
import networkx as nx
from joblib import Parallel, delayed
import random
import itertools
import numpy as np
from gensim.models import Word2Vec
```

Utils: Processing data

In [ ]:

```
def partition_num(num, workers):
    if num % workers == 0:
        return [num//workers]*workers
    else:
        return [num//workers]*workers + [num % workers]
```

TO DO: Implement DeepWalk



Algorithm 1 DEEPWALK(G, w, d, γ, t)

```
Input: graph $G(V, E)$
    window size $w$
    embedding size $d$
    walks per vertex $\gamma$
    walk length $t$
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$
Initialization: Sample $\Phi$ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree $T$ from $V$
for $i = 0$ to $\gamma$ do
    $\mathcal{O} = \text{Shuffle}(V)$
    for each $v_i \in \mathcal{O}$ do
        $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
        SkipGram($\Phi$, $\mathcal{W}_{v_i}$, $w$)
    end for
end for
```

In [ ]:

```
# Please implement your code here.
# 1. Try to build your own code
# 2. Try to use class RandomWalk built in the KarateClub framework (used in h
class RandomWalker:
  def __init__(self, G, num_walks, walk_length):
      """
      :param G: Graph
      :param num_walks: a number of walks per vertex
      :param walk_length: Length of a walk. Each walk is considered as a sent
      """
      self.G = G
      self.num_walks = num_walks
      self.walk_length = walk_length

  # INFORMATION EXTRACTOR
  def simulate_walks(self):
      # TO DO: Create walks
      pass
```

In [ ]:

```
class DeepWalk:
    def __init__(self, graph, walk_length, num_walks):

        self.graph = graph
        self.w2v_model = None
        self._embeddings = {}

        self.walker = RandomWalker(graph, num_walks=num_walks, walk_length=wa
        self.walks = self.walker.simulate_walks()

    def train(self, embed_size=128, window_size=5, workers=1, iter=5, **kwarg

        kwargs["sentences"] = self.walks
        kwargs["min_count"] = kwargs.get("min_count", 0)
        kwargs["size"] = embed_size
        kwargs["sg"] = 1  # skip gram
        kwargs["hs"] = 1  # deepwalk use Hierarchical Softmax
        kwargs["workers"] = workers
        kwargs["window"] = window_size
        kwargs["iter"] = iter

        print("Learning embedding vectors...")
        model = Word2Vec(**kwargs) # Pay attention here
        print("Learning embedding vectors done!")

        self.w2v_model = model
        return model

    def get_embeddings(self,):
        if self.w2v_model is None:
            print("model not train")
            return {}

        self._embeddings = {}
        for word in self.graph.nodes():
            self._embeddings[word] = self.w2v_model.wv[word]

        return self._embeddings
```

Run graph embedding

In [ ]:

```
G = nx.read_edgelist('lab1_big_edgelist.txt',create_using=nx.DiGraph(),nodety
model = DeepWalk(G,walk_length=10,num_walks=80)#init model
model.train(window_size=5,iter=3)# train model
embeddings = model.get_embeddings()# get embedding vectors
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    4.8s finished
Learning embedding vectors...
Learning embedding vectors done!
```

Questions

Did you see that we use the function "Word2vec" as the primary function to implement the DeepWalk algorithm?

The reason is that DeepWalk is based on the idea of Word2vec. As a result, all we need is packed in the implementation of Word2vec. Within a short amount of time, we couldn't go through all the code.

This is your homework. The details will be shown in the file "Lab3 - Homeworks".

Please take a look at this file for more details.