TP DI Graph Letter Classification Student-Correction

January 15, 2020

1 A Simple Graph Neural Network from Scratch

2 Goals :

1. An illustration of GCN for graph classification
2. A toy application to Letter classification
3. A lecture on Structured Machine Learning

3 Author: Romain Raveaux (romain.raveaux@univ-tours.fr)

4 Acknowledgement

1. Datasets and functions to read data are taken from http://gmprdia.univ-lr.fr/

4.1 Install requirements

- Network
- Pytorch

5 The lecture

The content of the is notebook is based on the following lectures : Supervised Machine Learning for structured input/output: Polytech, Tours

- 1. Introduction to supervised Machine Learning: A probabilistic introduction PDF
- 2. Connecting local models : The case of chains PDF slides
- 3. Connecting local models : Beyond chains and trees.PDF slides
- 4. Machine Learning and Graphs : Introduction and problems PDF slides
- 5. **Graph Neural Networks.** PDF slides
- 6. Graph Kernels. PDF slides

```
In [1]: #!pip install networkx
        #!pip install torch
        #!pip install scipy
        #!pip install matplotlib
```

5.1 Download Data

5.1.1 Letter Database

Graphs that represent distorted letter drawings. They consider the 15 capital letters of the Roman alphabet that consist of straight lines only (A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z). Each node is labeled with a two-dimensional attribute giving its position relative to a reference coordinate system. Edges are unlabeled. The graph database consists of a training set, a validation set, and a test set of size 750 each. Also, three levels of distortions are provided.

This dataset is part of IAM Graph Database Repository and it is also linked in the IAPR TC15 resources.

It can be considered as a **TOY EXAMPLE** for graph classification.

> Riesen, K. and Bunke, H.: IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In: da Vitora Lobo, N. et al. (Eds.), SSPR&SPR 2008, LNCS, vol. 5342, pp. 287-297, 2008.

```
In [2]: #!wget https://iapr-tc15.greyc.fr/IAM/Letter.zip
        #!unzip Letter.zip
```

5.2 Prepare data reader

IAM graphs are provided as a GXL file:

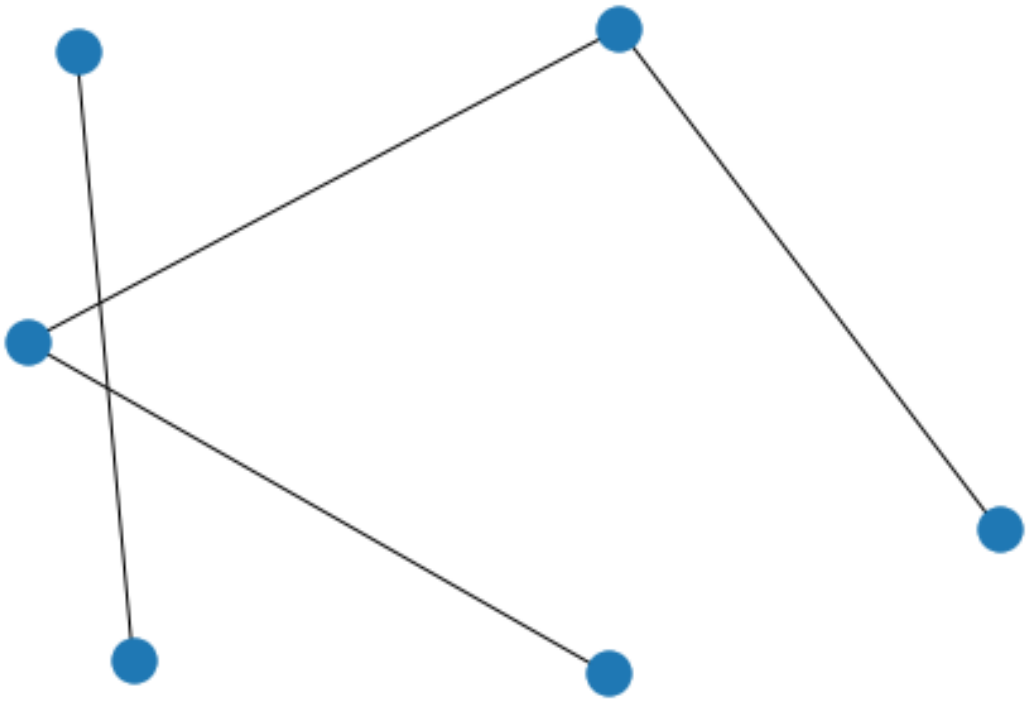
```
<gxl>
  <graph id="GRAPH_ID" edgeids="false" edgemode="undirected">
    <node id="_0">
      <attr name="x">
        <float>0.812867</float>
      </attr>
      <attr name="y">
        <float>0.630453</float>
      </attr>
    </node>
    ...
    <node id="_N">
      ...
    </node>
    <edge from="_0" to="_1"/>
    ...
    <edge from="_M" to="_N"/>
  </graph>
</gxl>
```

In [3]:
```
import numpy as np
import xml.etree.ElementTree as ET
import networkx as nx
import torch


def read_letters(file):
  """Parse GXL file and returns a networkx graph
  """

  tree_gxl = ET.parse(file)
  root_gxl = tree_gxl.getroot()
  node_label = {}
  node_id = []

  # Parse nodes
  for i, node in enumerate(root_gxl.iter('node')):
    node_id += [node.get('id')]
    for attr in node.iter('attr'):
      if (attr.get('name') == 'x'):
        x = float(attr.find('float').text)
      elif (attr.get('name') == 'y'):
        y = float(attr.find('float').text)
    node_label[i] = [x, y]

  node_id = np.array(node_id)

  # Create adjacency matrix
  am = np.zeros((len(node_id), len(node_id)))
  for edge in root_gxl.iter('edge'):
    s = np.where(node_id==edge.get('from'))[0][0]
    t = np.where(node_id==edge.get('to'))[0][0]

    # Undirected Graph
    am[s,t] = 1
    am[t,s] = 1

  # Create the networkx graph
  G = nx.from_numpy_matrix(am)
  nx.set_node_attributes(G, node_label, 'position')

  return G
```

5.3 Load Data with NetworkX

In [4]:
```
import os

# Select distortion [LOW, MED, HIGH]
```

```
distortion = 'LOW'

# Select letter [A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z]
letter = 'K'

# Select id [0-149]
id=100

# Read the graph and draw it using networkx tools
G = read_letters(os.path.join('Letter', distortion, letter+'P1_'+ str(id).zfill(4) +'.g
nx.draw(G, pos=dict(G.nodes(data='position')))
```



5.3.1 Define dataset

Dataset Division The dataset is divided by means of CXL files in *train, validation* and *test* with the correspondance filename and class:

```
<GraphCollection>
  <fingerprints base="/scratch/mneuhaus/progs/letter-database/automatic/0.1" classmodel="henry5
    <print file="AP1_0100.gxl" class="A"/>
    ...
    <print file="ZP1_0149.gxl" class="Z"/>
  </fingerprints>
</GraphCollection>
```

```
In [5]: def getFileList(file_path):
          """Parse CXL file and returns the corresponding file list and class
          """

          elements, classes = [], []
          tree = ET.parse(file_path)
          root = tree.getroot()

          for child in root:
            for sec_child in child:
              if sec_child.tag == 'print':
                elements += [sec_child.attrib['file']]
                classes += sec_child.attrib['class']

          return elements, classes
```

Define Dataset Class Pytorch provides an abstract class representig a dataset, `torch.utils.data.Dataset`. We need to override two methods:

- `__len__` so that `len(dataset)` returns the size of the dataset.
- `__getitem__` to support the indexing such that `dataset[i]` can be used to get i-th sample

```
In [6]: import torch.utils.data as data
        from torch.utils.data import DataLoader

        class Letters(data.Dataset):
          def __init__(self, root_path, file_list):
            self.root = root_path
            self.file_list = file_list

            # List of files and corresponding labels
            self.graphs, self.labels = getFileList(os.path.join(self.root, self.file_list))

            # Labels to numeric value
            self.unique_labels = np.unique(self.labels)
            self.num_classes = len(self.unique_labels)

            self.labels = [np.where(target == self.unique_labels)[0][0]
                           for target in self.labels]


          def __getitem__(self, index):
            # Read the graph and label
            g = read_letters(os.path.join(self.root, self.graphs[index]))
            target = self.labels[index]

            nodelist, nodes = map(list, zip(*g.nodes(data='position')))
```

```
            nodes = np.array(nodes)
            edges = np.array(nx.adjacency_matrix(g, nodelist=nodelist).todense())

            return nodes, edges, target

          def label2class(self, label):
            # Converts the numeric label to the corresponding string
            return self.unique_labels[label]

          def __len__(self):
            # Subset length
            return len(self.labels)


        # Define the corresponding subsets for train, validation and test.
        trainset = Letters(os.path.join('Letter', distortion), 'train.cxl')
        validset = Letters(os.path.join('Letter', distortion), 'validation.cxl')
        testset = Letters(os.path.join('Letter', distortion), 'test.cxl')

        print(len(trainset.labels))
        print((trainset.labels[100]))
        print(len(trainset.graphs))
        print((trainset.graphs[0]))
        print((trainset.unique_labels))
        print((trainset.num_classes))

        print(len(validset.labels))
        print(len(testset.labels))

        print(trainset.__getitem__(100)[0])
        print(trainset.__getitem__(100)[1])
        print(trainset.__getitem__(100)[2])
```

```
750
2
750
AP1_0000.gxl
['A' 'E' 'F' 'H' 'I' 'K' 'L' 'M' 'N' 'T' 'V' 'W' 'X' 'Y' 'Z']
15
750
750
[[0.626163 2.85025 ]
 [0.617416 0.715336]
 [0.572522 1.65465 ]
 [1.65172  1.66186 ]
 [2.1      2.78    ]
 [0.706782 0.799407]]
[[0. 1. 0. 0. 1. 0.]
```