Machine Learning II: Deep Learning and Applications

Homework 3

Due date: Apr 20

Instructions

- Make a copy of this notebook in your own Colab and complete the questions there.
- You can add more cells if necessary. You may also add descriptions to your code, though it is not mandatory.
- Make sure the notebook can run through by *Runtime -> Run all*. **Keep all cell outputs** for grading.
- Submit the link of your notebook here. Please **enable editing or comments** so that you can receive feedback from TAs.

Install GraphVite and PyTorch. This may take a while.

```
!wget -c https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
!chmod +x Miniconda3-latest-Linux-x86_64.sh
!./Miniconda3-latest-Linux-x86_64.sh -b -p /usr/local -f

!conda install -y -c milagraph -c conda-forge graphvite \
  python=3.6 cudatoolkit=$(nvcc -V | grep -Po "(?<=V)\d+\.\d+")
!conda install -y wurlitzer ipykernel
```

```
import site
site.addsitedir("/usr/local/lib/python3.6/site-packages")
%reload_ext wurlitzer
```

```
import numpy as np
from matplotlib import pyplot as plt
import sklearn
import torch
from torch import nn
%matplotlib inline
import graphvite as gv
import graphvite.application as gap
```

1. Node embedding and visualization (50 points)

1) Node embedding

In this part, we will implement unsupervised node embeddings, and evaluate the learned embeddings on some downstream tasks.

Common packages for implementing node embeddings include

- GraphVite: Website, Tutorial, Example config
- PyTorch BigGraph: Website, Document, Example config

- Open NE: Website, Tutorial

The following scaffold is based on GraphVite. However, you can override it with any implementation.

We carry out the experiments on BlogCatalog dataset, where each node corresponds to a blog user and each edge corresponds to their friendship. Some node has labels which indicate the user's interests.

The dataset can be found in GraphVite.

The train file contains edge list of format `[head] [tail]`.

The test file contains node labels of format `[node] [label]`. Note one node may have multiple labels.

```
print(gv.dataset.blogcatalog.train)
print(gv.dataset.blogcatalog.label)
```

Now train the node embeddings. For GraphVite, the following steps are needed.

- Create a GraphApplication instance
- Load the training file to the application
- Build the application
- Train the application with hyperparameters

Implement the missing steps in the following cell.

Note: Due to the implementation of GraphVite, `batch_size` should always be divisible by `augmentation_step`, otherwise Colab would crash.

```
# TODO: Implement the training and evaluation steps
```

Evaluate the learned embeddings on node classification task. Try to use different portions of labeled nodes to train the node classifier.

How much macro-F1 and micro-F1 do you get for different portions?

	10%	20%	30%	40%
macro-F1				
micro-F1				

2) Visualization

In this part, we will visualize the embeddings learned in the previous problem.

Common packages for visualization include

- GraphVite (LargeVis): Website, Tutorial
- tSNE-CUDA (t-SNE): Website, Document
- scikit-learn (t-SNE): Website, Document

Implement the training of the visualization algorithm.

```
# TODO: Implement the training of visualization
```

Visualize the coordinates as a scatter plot.

You may need to tune the `perplexity` in LargeVis / t-SNE to get a better structure.

TODO: Visualize the coordinates

Use the node classification labels to pick two classes of nodes. Color these nodes in the plot.

Hint: For those belonging to both classes, you can randomly assign it to one class, or use a third color to denote them.

Are the node embeddings learned from BlogCatalog good? Why or why not?

Answer:

2. Graph Convolutional Networks (50 points)

For this part, we are going to implement the GCN model for node classification, where the Cora dataset is used for testing the model.

The Cora dataset is avaible here. You could find several files there. Among those files, net.txt provides the edges between different nodes, and the three columns correspond to source nodes, target nodes and edge weights respectively. For feature.txt, it gives the features of nodes. For label.txt, it provides the node labels. train.txt, valid.txt, test.txt provide the training nodes, validation nodes and test nodes respectively.

The goal is to train a model on the training nodes, and further apply the model for classifying test nodes. In this process, you may use the validation nodes for hyper-parameter tuning and early stopping.

1) Implement the GCN model

In the first step, please implement a GCN model in the following code block.

```
# TODO: Implement GCN here
```

2) Performance w.r.t. the number of layers

Most GNN models only use a few layers for information propagation. Otherwise, they may suffer from the over-smoothing problem. To look into that, please fill in the following table to show the performance of your GNN model with respect to the number of propagation layers.

	1	2	3	4	5	6	7	8	9	10
Test Accuracy										

Given the above result, please write down your observation and the potential reason based on your understanding.

Observations:

Reasons:

3) Performance w.r.t. the hidden dimension

Another important hyperparameter for graph neural networks is the dimension of hidden layers. In this part, you need to try different dimensions for the hidden layers, and further fill in the table below.

	4	**8**	**16**	**32**	**64**	**128**
Test Accuracy						

Given the above result, please write down your observation and the potential reason based on your understanding.

Observations:

Reasons: