Community Detection on NIPS12

January 22, 2019

1 Community Detection on NIPS12 dataset

1.1 We will visualize and detect the overlapping communities from the latent structure learned by our framework

```
In [1]: %matplotlib inline

        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        from IPython.display import Markdown, display
        from utils import plota_fn

        def printmd(string):
            display(Markdown(string))

        import scipy.io as sio

        from input_data import load_data
```

```
/usr/local/lib/python2.7/dist-packages/h5py/__init__.py:34: FutureWarning: Conversion of the sec
  from ._conv import register_converters as _register_converters
```

1.1.1 Embedding learned using the following command! The embedding learned is also attached for quick reference. For ease of visualization, we have taken K=10 and alpha=2.

```
python train.py --dataset nips12 --hidden 64_10 --alpha0 2 --split_idx 0
--reconstruct_x 0 --early_stopping 0 --deep_decoder 0 --split_idx 0 --model dglfrm
--epochs 1000 --weighted_ce 1 --dropout 0
```

The above command trains the DGLFRM model on 85% of the adjacency matrix.

```
In [2]: nips12 = np.load('data/qual_nips12_dglfrm.npz')
        # nips12_vae = np.load('data/qual_nips12_gcn_vae.npz')
        print nips12.keys()

        nips_author = sio.loadmat('data/nips12authors.mat')
        nips_author_names = nips_author['anames']
```

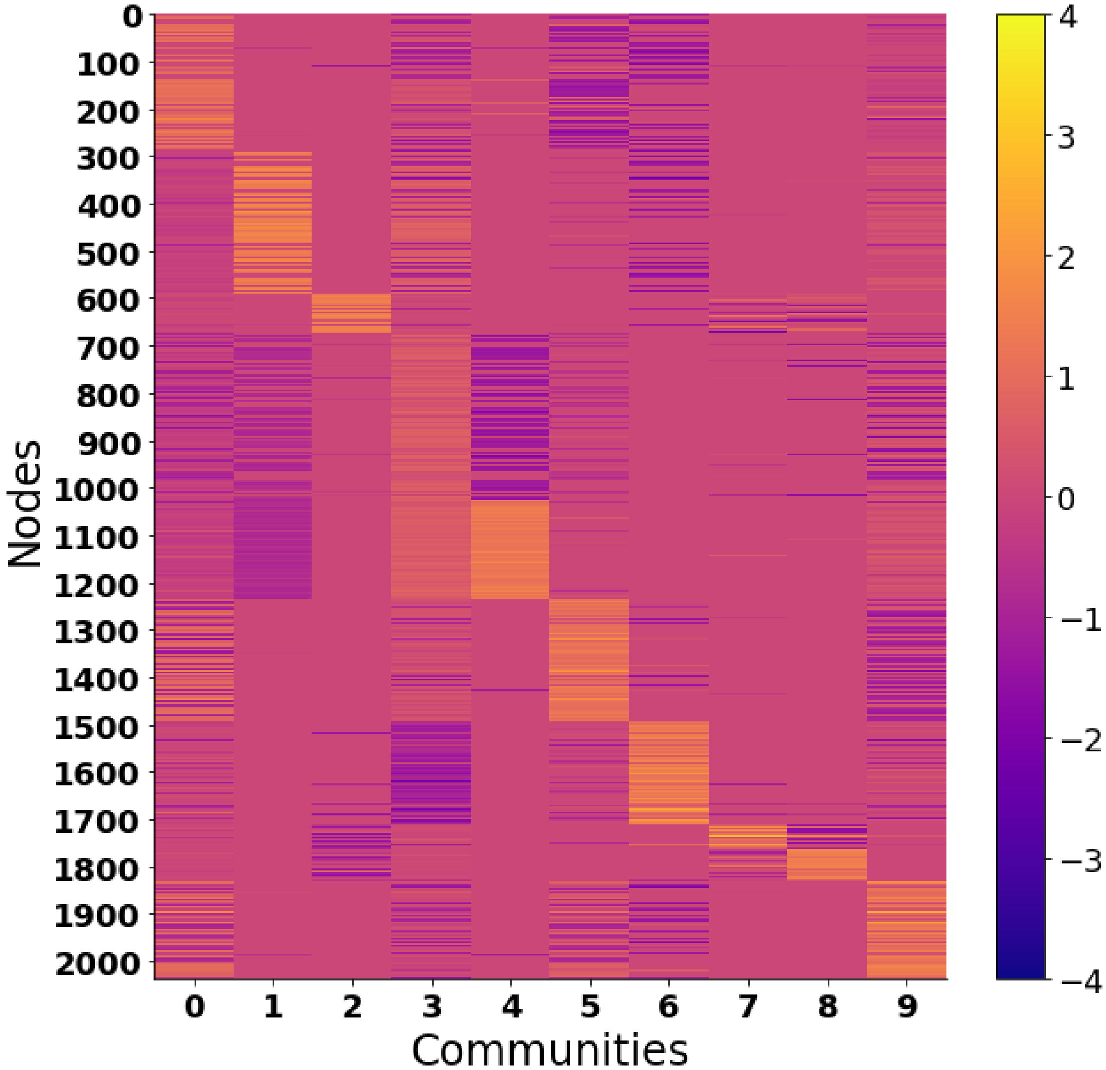
```
['z_out', 'z_real', 'z_discrete', 'adj_rec']
```

1.2 Creating the node embedding z_n = b_n * r_n

where " * " is the element wise product.

```
In [3]: size = (10, 10)

        b = np.round(nips12['z_discrete'])
        r = nips12['z_real']

        avg_activated_communities = np.sum(b) / b.shape[0]
        z = np.multiply(r, b)

        # Reordering the communities (Columns of Z) for visualization.
        total_community_strength = np.sum(z, axis=0)
        new_community_idx = np.argsort(-total_community_strength)
        z = z[:, new_community_idx]

        # Reordering the nodes (Rows of Z) such that nodes having
        # high strength for communities with lower indices are on top.
        max_node_community_idx = np.argmax(z, axis=1)
        new_node_idx = np.argsort(max_node_community_idx)
        z = z[new_node_idx, :]

        # Since we re-ordered the nodes, we should re-arrange the author_names as well
        author_names = nips_author_names[new_node_idx, :]

        printmd ("**Average activated communities: {}**".format(avg_activated_communities))
        plota_fn(z, size, 'plasma', f_name="", x_step=1, y_step=100, vmin=-4, vmax=4, x_label='C
```

Average activated communities: 4.16200294551



1.3 Print authors in communities

```
In [4]: # Sort all authors based on their strength
        sorted_d = np.argsort(-z, axis=0) # Sort all authors and get indices
        num_authors = len(sorted_d)

        M = 50 # Numbers of authors to print per community
        author_positive = sorted_d[0:M,:] # Get top M for all comm
        author_negative = sorted_d[num_authors-M:,:] # Get bottom 10 authors for all comm
        # Note: Both high negative and high positive strengths can form communities.

        def flat (author_names, col='blue'):
            author_list = ''
```

```
    for arr_name in author_names:
        author_list += arr_name[0][0] + ', '
    return author_list

print ('Ordering is based on strength of the membership in a community. We are printing

K = 10 # Number of communities to print.
for i in np.arange(K):
    printmd ('**Community {}**'.format(2*i))
    print (flat(np.flip(author_names[author_negative[:,i]], 0)))
    printmd ('**Community {}**'.format(2*i+1))
    print (flat(author_names[author_positive[:,i]]))
```

```
Ordering is based on strength of the membership in a community. We are printing top 50 authors.
```

Community 0

```
Koch_C, Bower_J, Moore_A, Coolen_A, Moody_J, Horiuchi_T, DeWeerth_S, Bair_W, Tishby_N, Saad_D, H
```

Community 1

```
Sejnowski_T, Spence_C, Barto_A, Muller_K, Dayan_P, Platt_J, Scholkopf_B, Mozer_M, Smola_A, Gelfa
```

Community 2

```
Cowan_J, Chiang_Y, Hanson_S, Shavlik_J, Wang_D, Munro_P, Schreiner_C, Roychowdhury_V, Potter_D,
```

Community 3

```
Edelman_S, Ruppin_E, Horn_D, Cooper_L, Weinshall_D, Intrator_N, Meilijson_I, Eeckman_F, Bert_J,
```

Community 4

```
Anderson_C, Yuan_J, Latham_P, Malik_J, Schuster_M, Vu_V, Phillips_P, Malkoff_D, Lang_K, Annaswam
```

Community 5

```
Walter_J, Leong_H, Berthold_M, Waskiewicz_J, Lewis_M, Caprile_B, Siegel_R, Santos_E, Grove_A, We
```

Community 6

```
Hinton_G, Guyon_I, Jordan_M, LeCun_Y, Giles_C, Simard_P, Schapire_R, Opper_M, Bengio_Y, Personna
```

Community 7

Shavlik_J, Blair_A, Roychowdhury_V, Cowan_J, Peper_F, Maass_W, Horn_D, Toomarian_N, Hanson_S, Co

Community 8

Cowan_J, Shavlik_J, Munro_P, Tsitsiklis_J, Hancock_E, Hanson_S, Thakoor_A, Paugam-Moisy_H, Ullma

Community 9

Roychowdhury_V, Baluja_S, van-Schaik_A, Poggio_T, Kailath_T, Linden_A, Xiang_D, Maass_W, Thrun_S

Community 10

Barto_A, Smola_A, Bartlett_P, Scholkopf_B, Meir_R, Sutton_R, Shawe-Taylor_J, Andreou_A, Alspecto

Community 11

Sejnowski_T, Goodman_R, Moody_J, Yang_H, Chauvin_Y, Coolen_A, Krogh_A, Henkle_V, Obermayer_K, Ba

Community 12

Jordan_M, Kawato_M, Murray_A, Bishop_C, Atlas_L, Wolpert_D, Opper_M, Abbott_L, Ghahramani_Z, Col

Community 13

Giles_C, Cottrell_G, Bengio_Y, Graf_H, Morgan_N, Lippmann_R, Jabri_M, Mjolsness_E, Waibel_A, Plu

Community 14

Sundararajan_S, Bengio_Y, Monaco_J, Metcalfe_J, Margaritis_D, Fukumizu_K, Metz_C, Kamimura_R, Gh

Community 15

Kremer_S, Wejchert_J, Haffner_P, Page_E, Brand_M, Murphy_K, Lambert_R, Simmons_J, Smith_L, Tanak

Community 16

Marts_A, Sereno_M, Burgess_A, Stolorz_P, Asogawa_M, Levemon_M, Freeman_D, Pareigis_S, Ring_M, Gl

Community 17

```
Zeitouni_O, Carley_L, Duda_R, Hormel_M, Kremer_S, Lemmon_M, Slaney_M, Tam_D, Cole_C, Hartstein_A
```

Community 18

```
Koch_C, Sejnowski_T, Ruderman_D, Bower_J, Dayan_P, Mel_B, Obermayer_K, Touretzky_D, Harris_J, Ba
```

Community 19

```
Mozer_M, Stork_D, Tishby_N, Tresp_V, Wolff_G, Ohmi_T, McNaughton_B, Yamashita_T, Nakashima_M, Bu
```

1.4 Example of communities inferred on a random split:

Community 6: Hinton_G, Jordan_M, LeCun_Y, Bengio_Y, Bishop_C, Williams_C ...
Community 10: Barto_A, Sutton_R, Singh_S ...
Community 12: Jordan_M, Bishop_C, ..
Community 18: Sejnowski_T, Pearlmutter_B, Abu-Mostafa_Y, Tang_A

```
In [ ]:
```