社群網路與推薦系統

HW 1

賴廷瑋 F44054045

Implementation Detail

這次作業的 NN 模型即依照 DrBC 論文中提及的方法和架構實作，使用套件爲 Pytorch Geometric

模型建立

初始化

1. 建立名爲 DrBC 的類別(CLASS)，並繼承 Pytorch Geometric 中 MessagePassing 類別，以便後續的 Neighbor Aggregation 步驟
2. 利用 Pyorch 中的 torch.nn.Linear 線性轉換函數設定此 DrBC 物件的參數，分別爲論文中提及之 W0, W4, W5，與論文不同的部分在於，函數中增加了 BIAS，以利資料的平移轉換。
3. 亦利用 Pytorch 的 torch.nn.GRUCell 創建 GRUcell，並設定其 input size 及 hidden size 爲 out_channels = 128。

Forward Function

1. Input 分別爲各個 node 的初始化特徵向量 (num_nodes, 3)，及圖形中所有 edge 的組合 (2, num_edges)。

2. 首先將結點的特徵向量和參數 W0 作矩陣相乘，並通過 Pytorch 中的 ReLU 函式作數值轉換，在將各點的特徵向量依照做 Norm2 正歸化。

3. 計算每個節點的 degree 值，並得到每兩點之間的正規化係數(即論文中將倫居節點與自身節點做 Neighbor Aggregation 時所需的正規化係數，如下)

$$h_{N(v)}^{(l)} = \sum_{j \in N(v)} \frac{1}{\sqrt{d_v + 1} \cdot \sqrt{d_j + 1}} h_j^{(l-1)}$$

此部分調用 Pytorch Geometric 中 Data 與 Degree 類別及函數。

4. 接著針對每層的 Neighbor Aggregation，調用 MessagePassing 中的 propagate 函數，Input 爲所有 edge 的組合、每個節點的特徵，以及上述提到之兩節點之間的正規化係數矩陣。Propagate 函數會自動調用 DrBC 中自定義的 message 函數及 update 函數，在 message 函數中，對於每個節點，將其鄰居之特徵向量與兩節點之間的正規化係數做相乘，並將所有鄰居做完正規化的結果相加，而 Propagte 函數則 output 出經過一層 aggregation 後所有節點更新後的特徵向量。

5. 得到各節點的 Neighbor Aggregation 特徵向量後，再一併將結點在上一層的特向量輸入至 GRUcell 中得到新的特徵向量，並再將其做 Norm2 正歸化。

6. 以上步驟重複 L=5 次。

7. 得到各點分別在 L=5 層中的特徵向量後，再依照論文進行 Max Pooling，這裡使用 nn.AdaptiveMaxpool2d 函數。

8. 將得到的特徵向量與 W4 參數進行矩陣相乘後再輸入至 ReLU 函數

9. 再將得到的向量與 W5 參數相乘，得到最終每個節點的 BC 值

資料訓練

1. 利用 python 中 Networkx 中的 random_graphs.powerlaw_cluster_graph 函數生成隨機網路圖
2. 利用 Networkx 中 betweenness_centrality 函數計算每個生成圖中各節點的 BC 值(Ground Truth)
3. 接著依照論文所述，調用 numpy 的 random.choice 函數，分別從圖形中的 source nodes 及 target nodes 各隨機取 5*V 個節點，並透過模型預測得到 source node 及 target node 之間 BC 值的差，亦計算其 ground truth 的差
4. 將預測的差值以及 ground truth 的差值輸入至 loss function，其中 loss function 調用 pytorch 中 BCEWithLogitsLoss 函數。即得 loss，並做訓練。
5. 爲達到批次訓練的效果，以及減少 GPU 的使用量，設定在每隔 16 次的 iteration 才進行一次的參數係數更新，即使用梯度累積的方法，達到批次訓練的效果 (Batch size 設爲 16)。

Metrics

1. 爲實踐 top k accuracy 的計算，使用 torch.topk 函式分別將所有節點的 BC 值預測結果及其 Ground Truth 篩選出各自前百分之 K 的節點，並再利用 Numpy 中的 intersect1d 函式取出其交集後，即可算得 top K accuracy。
2. 爲實踐 kendall tau 的計算，使用 scipy.stats 中的 rankdata 函數將結點預測值與 ground truth 分別做排名排序後，再利用 stats.kendalltau 函數計算其值，便得到預測值與 ground turth 之間的 kendall tau 係數。

實驗結果

參數設定

參數	值
Learning rate	1e-4
Embedding dimension (out_channels)	128
Batch size	8
Average node sampling times	5
Epochs (episodes)	200
Layer iterations	5
Training graph n, m , p	1000, 4, 0.05

測試資料

因本次作業僅提供 n= 5000 的 graph data，爲了達到論文中以不同 graph scale 進行模型表現比較，本次從論文作者 Github 下載其測試資料，其中生成原理即利用 Networkx 中的 powerlaw_cluster_graph 進行隨機的 graph 生成，參數 m= 4, p = 0.05，n 分別爲 10000, 20000, 50000, 100000 的隨機 graph data 各 30 個。眞實資料的部分也使用作者提供的測試資料。

實驗細節

爲節省訓練時間，本次作業利用 n=1000 (節點數)的隨機生成圖形作爲 table 3 與 table 4 資料的模型訓練，生成方式亦利用 Networkx 中 powerlaw_cluster_graph 函數生成，(若以 n=500 做訓練發現收斂效果並不好，多次 iteration 後發現訓練資料的 Top 1% accuracy 收斂於 0.8，kendall tau 收斂於 0.7 左右)，並以 batch size 8 進行梯度累積(實驗後發現若 batch size 與論文相同提高至 16 的話對於訓練並不會有顯著的影響，反而增加許多訓練時間)，而透過 200 次的 iteration 後，可以觀察到模型訓練已經逐漸收斂，

Table 3 Top-N% accuracy (×0.01) on synthetic graphs of different scales.

scale	Top-1%	Top-5%	Top-10%
5000	92.1±2.6	92.7±1	91.4±1
10000	92.6±1.8	92.6±0.9	90.4±1
20000	93.3±1.5	91.4±0.8	88.3±0.9
50000	93.1±0.7	88.6±0.8	85.4±0.7
100000	92.7±0.5	86.6±0.5	83.3±0.6

Table 4 Kendall tau distance (×0.01) on synthetic graphs

	DrBC
5000	81.3±0.6
10000	81.7±0.4
20000	81±0.4
50000	79.5±0.3
100000	78.5±0.3

Table 5 Running time comparison on synthetic networks (in seconds)

	DrBC
5000	0.003±0.013
10000	0.045±0.008
20000	0.09±0
50000	0.204±0.017
100000	0.408±0.029

Table 6 DrBC's generalization results on different scales

(Top-1% accuracy, ×0.01).

	5000	10000	20000	50000	100000
100_200	89.1±2.9	88.2±2.2	89.1±1.3	89.5±1	89.7±0.7
200_300	90.3±2.5	90±2.1	91.2±1.5	91.4±0.8	91.5±0.5
1000_1200	92.1±2.6	92.6±1.8	93.3±1.5	93.1±0.7	92.7±0.5
2000_3000	90±2.5	89.7±2.2	90.8±1.5	91±0.9	91.2±0.5
4000_5000	86.2±3.7	85.2±2.7	86±1.5	86.2±1.1	86.6±0.8

Table 7 DrBC's generalization results on different scales

(kendall tau distance, ×0.01).

	5000	10000	20000	50000	100000
100_200	77.2±0.7	77.7±0.4	77.3±0.4	76.2±0.3	75.4±0.3
200_300	77.7±0.8	77.9±0.4	77.2±0.4	75.8±0.3	74.9±0.3
1000_1200	81.3±0.6	81.7±0.4	81±0.4	79.5±0.3	78.5±0.3
2000_3000	78.3±0.5	77.3±0.3	76.1±0.3	74.4±0.2	73.3±0.2
4000_5000	76.2±0.7	76.7±0.4	76.3±0.4	75.3±0.3	74.6±0.3

我分別選用 n= 100, 200, 1000, 2000, 4000 來建立 table

Table 8 Top-N% accuracy (×0.01) and running time on large real-world networks.

	Top-1%	Top-5%	Top-10%
Com-youtube	0.397	0.539	0.66
amazon	0.433	0.72	0.72
dblp	0.402	0.68	0.61
Cit-Patents	X	X	X
Com-lj	0.413	0.674	0.633

Table 9 Kendall tau distance (×0.01) on real-world networks.

	Kendall tau	Time/s (On GPU)
Com-youtube	0.668	0.70
amazon	0.723	1.52
dblp	0.746	3.6
Cit-Patents	X	X
Com-lj	0.73	5.2

結論

1. 進行模型訓練時，有發現利用 n=1000 以上的 graph data，通常進行 50 次以上的 iteration 後模型表現就會開始收斂，而 n 小於 1000 以下的 graph data 通常需要 200 次以上的 iteration，雖然 iteration 次數與論文中不相同，但是驗證了 n 愈大，收斂愈早發生的情況。同樣也發現當 n 愈大時，所需的 training 時間也隨之增加（時間的增加除了來自於在訓練時需要生成隨機 graph data 以及計算其 Ground Truth BC 值外，也因爲節點數量變多導致模型訓練時在 dimension 爲 0 的部分增加許多資料，也讓模型訓練時間加長，但模型主要訓練目的爲找出節點之 BC 值的”相對分數”，並不在絕對值，故在實作時，我利用 networkx 中 betweeness_centrality 中的參數 k 設爲 0.5*n，利用 approximation 的方式接近 ground truth 值，以減少訓練時間。

2. 這次實作結果中，從 table 6 及 table 7 也可以觀察到，訓練時利用 n 較大的 graph data 能有較佳的泛化能力，這點也和論文的結果相符，且可以觀察到 n 愈大時，測試資料的標準差也跟著變小，再次證明較佳的泛化能力，唯一特別的地方是，這次實驗

結果中會發現當用 n=1000 時訓練的模型在 test data 上有最好的表現，其原因可能爲 iteration 的次數不夠充足，因爲這次訓練皆以 iteration 爲 200 進行訓練 (若 iteration 過大將花費非常久的時間)，或者因爲在使用 networkx 的 betweeness_centrality 函數時使用 k 參數而減少節點之 BC 值的估計準確性，導致在 iteration 爲 200 時，n=1000 訓練的模型有最佳的表現。

3. 從實驗結果中也可以發現，當 test graph 的 n 愈大時，利用訓練出的模型(固定)所做的預測結果的標準差愈小，等於說在預測 n 愈大的 test graph 時，預測的結果更加穩定。這個現象在論文中並沒有顯現，而這次實作會有這樣的結果可能的其中原因就是 batch size 與論文所設定的 16 不同，本次實驗利用 batch size 爲 8，而 batch size 愈小通常模型對於預測結果的穩定性愈高，也提升模型的泛化能力，因對於較高的 batch size，模型較容易 overfit 所接受的 data。另外的原因可能是，這次時做的模型在參數設定上皆允許有 bias 的參數，而論文中的模型則無，新增 bias 參數可能讓模型更能找出 feature 與 ground truth 之間的線性平移關係。

4. 從 table 5 可以看到利用本次訓練出的模型對大型資料做預測僅需不到一秒的時間，這部分和論文的數值有些差異，我在計算時僅考慮給入 test data 後至模型計算出 BC 預測值之間的時間差距，但重點是可以很清楚的看到，利用 DrBC 模型做預測時可以將時間壓縮非常多，可以在很短時間內對大量節點做運算，即 DrBC 最大的好處。

5. 對於眞實資料，可以發現和論文不同的地方是，針對 AMAZON、YOUTUBE、Dblp 的資料，論文由 top1%至 top10%的分數是依序遞減的，但我的實驗卻是依序增加，並且整體效果上仍不及論文的實驗結果，但原因可能是因爲我的實驗是由較少 node 的資料作爲訓練，且 epoch 數較少，模型在訓練的完整度不夠。但可以發現，透過 GPU 計算後模型對於大型資料(幾百萬個 node)的圖形僅需不到 10 秒即可計算完成，雖說犧牲了準確性，但就如論文所提及，效率上增加許多。另外注意的是，實驗結果發現 kendall tau 的分數大多較論文實驗結果高，意味本次實驗模型針對大多數 node 的分數排名整體是較準確和穩定的，但針對前 10%的 node 可能精準度沒有像論文所做那麼高。