Graph Recurrent Networks for node classifications

Wonbong (Wayne) Jang *
66-72 Gower Street
Department of Computer Science
University College London
London, WC2A 2AE
United Kingdom
won.jang1108@gmail.com

Abstract

GRN(Graph Recurrent Networks) based on the simple RNN with two dense networks achieves similar performances with the existing models for classifying the nodes on graphs. GRN uses the random-walk transition matrix to generate the sequence of the matrices, and feeds the output tensor into the classification and the attention dense networks respectively, and computes the score for the labels. The contributions of this paper are that the random-walk transitioned data on graphs clusters as the number of jumps increases, and GRN takes advantage of these transition matrices with the simple architecture. GRN optimizes the parameters for different nodes and time-steps of the attention dense network, and the attention improves the performance of the model by a slight margin.

1 Introduction

Applying the neural networks with the data on graphs may not be direct. The metrics are usually not defined, so finding the neighbours of the specific vertex is not straight forward. Spectral method is one way to deal with the problem, as the scalar multiplications in the frequency domain corresponds to the convolutions in the spatial domains. Then, the problem is how to change the basis without complicating the computations.

The existing methods, such as Chebyshev Networks(ChebNet) proposed by Defferrard et al. [2016] and Graph Convolutional Network(GCN) come up with by Kipf and Welling [2016], apply Chebyshev polynomials to transform the basis to the spectral domains in the order of nodes($O(N)$). GCN assumes the second order polynomials(α_0 and α_1) and further tightens the condition by $\alpha_0 = -\alpha_1$. With the above restrictions and the renormalization trick, GCN comes up with the simple layer propagation. GCN achieved state-of-the-art performance on the semi-supervised learning problems, where the frameworks of the problems are proposed by Yang et al. [2016] (the Planetoid split). Graph Attention Networks(GAT) by Veličković et al. [2017] further develops the spectral methods by computing the attention matrix between nodes with the multi-headed self-attentions.

On the Planetoid split, the adjacency matrix is given and fixed, and the players on the graphs move to neighbouring nodes by the adjacency matrix. GRN is based on the idea that the neighbouring nodes are more similar than non-neighbouring vertices on graphs. It implements the simple Recurrent Neural Networks(RNN) which receives the sequence of matrices jumping from the initial data points. The one distinct feature of RNN is that it has the shared parameters for the matrices in different time-steps, and RNN assumes that the data has the translational invariant properties over the time. This applies well with the data on graphs.

*This project was done while he worked as a visitor at the Department of Statistics, LSE.

Rather than having two consecutive RNNs for the classification tasks, GRN uses the two dense networks which have shared parameters for different time-steps. The classification dense network reduces the feature dimensions to the number of classes, and the attention dense network assigns the scalar value depending upon the node and the time-step. The attention improves the performance by a slight margin.

2 Proposed Techniques

GRN composed of four steps: 1) GRN receives the input X and the random-walk transition matrix P, and it generates the chain of random-walk transitioned tensor from X to $P^{(t-1)}X$. 2) The simple RNN cell of GRN receives the sequence of matrices $X : P^{(t-1)}X$ and computes the output tensor of (time-steps, nodes, output dimensions). 3) The classification and the attention dense layers with the shared parameters for time-steps reduce the dimension of the outputs of RNN to the number of class and the scalar value separately. 4) Do scalar multiplication to get the score for each class.

2.1 Random-walk transitioned matrix

In a connected graph, the Markov chain converges to the stationary distribution over the multiple iterations if the chain is irreducible. Computing $P = D^{-1}A$ as a random-walk transition matrix where D, A are degree and adjacency matrix respectively, the specific player on the node moves to the adjacent node according to P in every time-step. If P, which is the stochastic matrix, is irreducible, it would reach to the stationary distributions regardless of the player's initial points. They have the property that the points cluster by themselves.

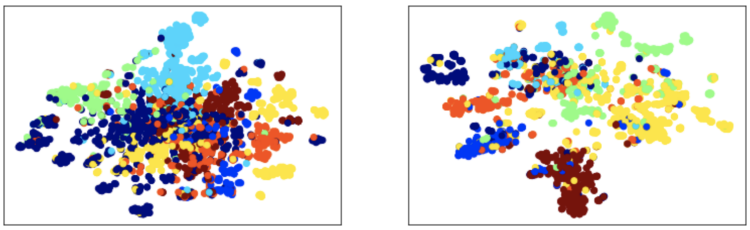

Figure 1: Left: Cora data after 1 step -Px, Right: after 10 steps-$P^{10}x$, colored with labels

As we can see from the above visualization by t-SNE proposed by Maaten and Hinton [2008] in the figure 1, the nodes on the graphs seems to aggregate over the centre points even with just one jump from initial positions as on the left-above figure, and they cluster quite well within ten iterations. These aggregates imply that they gathered the information of graphs coming from the neighbours by P, and they automatically form clusters. It is one of the reasons why the existing models based on graphs perform better than simple multi-layer perceptrons containing only the information of features.

2.2 Overall model architecture

The overall architecture of GRN composes of two parts. First, GRN receives the random-walk transition matrix P and the initial input X and samples the sequence of matrices with the simple RNN - $H^{(1)}$, .. $H^{(t)}$. Secondly, the output tensor from RNN is fed into the two separate dense networks with the shared parameters over the time-steps t, followed by the softmax activation function. The outputs from two dense networks, a and u, which are attention and the score respectively, and the logit is calculated based on $a \times u$. Then the cross-entropy loss is computed.

There are various ways to compute the score. Allocating the same attention(weight) over the time-steps may be the one way. The modified GRN which takes the last output of RNN and computes the logit without implementing the attention is the one way, or applying the hardcoded attention such as the summation of the output tensor of the RNN may be the other way.

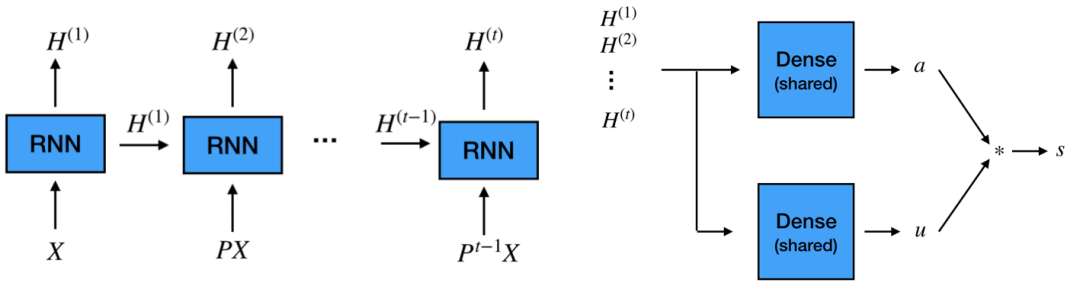

Figure 2: The architecture of GRN

2.3 Simple RNN and Convolutions

The convolution is the same as computing the weighted sum of neighbours for the given point. Convolutional Neural Networks(CNN) works well in images because it can exploit the locality of the data in the grid-like structures. By stacking multiple convolutional layers, CNN learns the hierarchical representations of features. However, on graphs, the neighbours of the given vertex are not defined naturally. One way to do the convolutions in the non-Euclidean domains is to convert the inputs to the spectral domains, to do scalar multiplications in the frequency domains, and to transform back to the spatial domains. It had the issue of finding the eigenvalues and the eigenvectors, with the help of Chebyshev polynomials on the rescaled eigenvalues, the existing models such as GCN performs fastly and efficiently.

The convolution also could be viewed as learning the filters in the spatial domains. As Shuman et al. [2012] suggested, the filtering operation on the specific node i is as equation 1 below, where j are within K-hop neighbours of i and $(\Gamma^K)_{i,j}$ is 1 when the shortest path between i and j within K steps and 0 otherwise. It means that it is possible to do the convolutions in the spatial domain if the adjacency matrix is known.

$$f_{out}(i) = \sum_{j=1}^{N} f_{in}(j) \sum_{k=0}^{K} a_k (\Gamma^k)_{i,j} \tag{1}$$

Inspired by the above equation 1, the convolutional operation can be expressed in the spatial domain as the below equation 2 given the time-step t. The recursive relations in the equation 2 can be implemented with the framework of the simple RNN cell with the learnable parameters W_H, W_X, W_b and the non-linear activation $\sigma(x) = \max(0, x)$

$$\begin{aligned} H^{(t)} &= \sigma(P^{t-1} X W_X + H^{(t-1)} W_H + W_b) \\ &<= \sigma(\sum_{k=0}^{t} (P^{t-k} X W_X W_H^k + W_b W_H^k)) \end{aligned} \tag{2}$$

As the equation 2 implies, GRN implements convolutional operations in spatial domains on graphs with RNN cell, without having to change the basis, if the adjacency matrix is fixed.

2.4 Dense layers for the attention and the classification

GRN generates the sequence of matrices and feeds them to the simple RNN as described above, and the output of RNN composes of the tensors with (number of time steps, number of nodes, dimensions of features). For classification tasks, the dense layer aggregate the outputs of RNN by extractung the features of which the dimension is the same as the number of classes, by applying the same parameters over the time-steps. GRN computes the reduced features(u) and the attention(a), and computes the scores(s) for every i.

GRN computes the attention $a_{i,t}$ for the given node i and the time-step t, and it uses the dense layer with the shared parameters for different t, followed by softmax activation for the fixed node i as in equation 3.

$$
\begin{aligned}
l_{i,t} &= H_i^{(t)} W_a + W_b \\
a_{i,t} &= \frac{\exp(l_{i,t})}{\sum_{t'} \exp(l_{i,t'})}
\end{aligned}
\tag{3}
$$

There may be different ways to compute the score such as the modified GRN which only incorporates the classification dense layer and takes the last output of RNN, or to use the hard-coded summation attention to get the score.

The attention dense layer has almost the same structures as the classification dense layer except that its output dimension is just the scalar value. The score s_i of the node i is given as the below, and W_s, W_b are learnable parameters shared over different time-steps t.

$$
\begin{aligned}
u_{i,t} &= H_i^{(t)} W_s + W_b \\
k_i &= \sum_{t=1}^{T} a_{i,t} u_{i,t} \\
s_i &= \frac{\exp(k_i)}{\sum_j \exp(k_j)}
\end{aligned}
$$

Dropout, Layer normalization and early stopping are also implemented to prevent the overfitting and to stabilize the training.

3 Related Works

There have been researches applying the neural networks on graphs. One of the problem was that the data does not lie on the Euclidean space, where CNN and RNN have exploited the stationary, locality, translational invariant properties of the data well, but these do not hold in non-Euclidean areas. The problems with applying the neural networks on graphs are well analyzed in Bronstein et al. [2017].

3.1 Spectral approaches and ChebNet

Spectral approaches for convolutions on graphs have been researched as convolutions in spatial domains are same as multiplications in spectral areas. Therefore, a signal y filtered by g_θ can be written as below equation where L is the Laplacian and Λ is the set of eigenvalues λs within L.

$$
\begin{aligned}
y &= g_\theta(L)x = U g_\theta(\Lambda) U^T x = U \text{diag}(\lambda) U^T x \\
&= \sum_j \alpha_j U T_j(\tilde{\lambda}) U^T x
\end{aligned}
$$

The problem is how to find the basis to compute Λ and U to transform them into the frequency domains, and it has the scalability issue - $O(n^2)$. ChebNet solves this basis dependent issue by assuming the Chebyshev polynomials $T_k(x)$ with the rescaled eigenvalues $\tilde{\lambda} = \frac{2}{\lambda_{max}} - 1$. Chebyshev polynomials have this recursive relations $T_{k+2}(x) = 2xT_{k+1}(x) - T_k(x)$, where $T_0(x) = 1$ and $T_1(x) = x$.

By optimizing α_js through the neural networks, the convolutional filtering on graphs can be learned. Also, as we increase the order r of the Chebyshev polynomials, it further calculates based on r-hop neighbours.

3.2 Graph Convolutional Networks

GCN further assumed the second-order Chebyshev polynomial with $\alpha_0 = -\alpha_1$. It comes up with the simpler equations as the below equation where $\tilde{A} = A + I$ and A is the adjacency matrix, $\tilde{D}$ is the degree matrix for $\tilde{A}$, and $\tilde{D} = D + I$. Unlike ChebNet, GCN does not need to compute the Chebyshev polynomials recusively, so GCN becomes the faster and more efficient.

$$H^{(t+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(t)}W^{(t)})$$

GCN achieved higher accuracy in the semi-supervised learning setting when it was first released. However, the performance of GCN does not necessarily increase as more than three layers are stacked on top, which may be due to the tight assumptions of GCN itself.

3.3 Graph Attention Networks

GAT computes the attention between nodes by self-attention. The reduced features are first used to calculate the attention by concatenating and repeating, and the attention does dot-product with the reduced features. The attention matrix can be sparse because the number of parameters on the attention matrix is the same as the number of edges in the adjacency matrix. The multi-headed attention may be applied to prevent the over-fitting.

$$\begin{aligned} \alpha_{i,j} &= \frac{\hat{a}^T\sigma(h'_i||h'_j)}{\sum_{k\in N_i}\hat{a}^T\sigma(h'_i||h'_k)} \\ h'_i &= \sigma(\sum_{j\in N(i)}\alpha_{ij}h'_j) \end{aligned} \tag{4}$$

As in the above equation 4, h_i, h'_i are inputs and outputs respectively for the given node i, and $\hat{a}$ is the learnable parameters to compute the attention between two nodes. $N(i)$ represents the neighbourhoods of the node i.

4 Numerical Experiments

GRN is compared with GCN and GAT under the Planetoid split proposed by Yang et al. [2016]. The modified GRN, which uses the last output from the RNN cell works achieves around 82% with Cora data set, which is slightly lower than GRN (approximately 1%). GRN is also compared with GCN by tightening the number of iterations from the inputs up to 2 times as GCN receives the input up to 2-hop neighbours, and GRN achieves the similar performance with GCN both in the accuracy and the execution time.

4.1 Test results

The accuracy rates for GCN and GAT are from their papers, respectively, Kipf and Welling [2016] and Veličković et al. [2017]. GRN performs comparably with two widely used models. The hyperparameter searches of GCN and GAT have been well studied in Shchur et al. [2018].

For the hyper-parameters of GRN, the Adam optimizer with the learning rate of 0.01 was used. The number of hidden layers was 112, and that of the iterations was 8, the patience for early stopping was 5, the dropout rates for RNN, attention, and classification are 0.2, 0.2 and 0.4 respectively, and the accuracy was the averaged value for the randomized index for 5 times. Layer normalization was implemented for RNN, as it increases the training speed of the model and stabilizes the training process.

4.2 The number of iterations vs the accuracy rate

The number of r iterations in GRN means that it takes the input from the 0 to r-hop neighbours. For the Cora data set, the performance of the model dramatically increases as t increases from 0 to 2, but it does not necessarily increase after the number of iterations becomes 3 or more significant.

Models	**Cora**	**Citeseer**	**PubMed**
GCN	81.4	70.3	79.0
GAT	83.0	72.5	79.0
GRN	83.7	70.4	78.4
# of Nodes	2708	3327	19717
# of Edges	5429	4732	44338
# of Classes	7	6	3
# of Features	1433	3703	500
# of Training	140	119	60
# of Validation	500	500	500
# of Test	1000	1000	1000

Table 1: The accuracy rates for GCN, GAT and GRN

H\T	0	1	2	3	4	5	6	7	8	9
64	53.6	74.5	77.7	80.9	80.3	81.8	82.2	81.2	83.4	82.1
112	55.7	74.9	79.8	79.6	82.0	82.0	81.1	80.7	83.7	82.4

Table 2: The accuracy rates for different time-steps with the dimension of 64 and 112 with Cora

The 0 iteration means that GRN is the same as Multi-Layer Perceptrons(MLP) which uses the feature information only. GRN improves the prediction by using the information on graphs. It also implies that the neural networks could perform better if the relationship between nodes is available.

4.3 Different types of attention

GRN can be implemented with different ways of applying the weights such as the modified GRN or the hard-coded attention for the output of the simple RNN.

4.3.1 Taking the last output - The modified GRN

There are different ways to deal with the outputs of RNN. As explained in the earlier section, the random-walk transitioned matrix clusters over centre points after a few jumps. (even they started to gather around a few points after one jump) Therefore, it is worth trying with the last output of the RNN without applying the attention.

H\T	0	1	2	3	4	5	6	7	8	9
64	54.1	76.3	79.9	79.8	82.0	79.8	81.4	80.7	81.0	80.8
112	53.2	75.8	80.1	81.0	81.6	82.1	82.2	80.2	81.3	80.6

Table 3: The accuracy rates for taking the last step with the dimension of 64 and 112 on Cora

The accuracy rates of the modified GRN also reaches around 82% in Cora dataset. The modified GRN does not incorporate the attention, but its performance is higher than just MLP. It clearly shows that as in figure 1, the clusters by random-walk transitions help GRN optimize the parameters faster. The information of the earlier hops is well aggregated within GRN.

4.3.2 Taking the summation

Rather than using the attention as the learnable parameters, another way to assign the attention is to take the sum over the sequence of output matrices from the simple RNN.

As clear from the table 4, the accuracy rates of the above model are lower than GRN and the modified GRN. It implies that all random-walk transitioned matrices are not equally relevant, and also inspires to apply the attention on the time-steps and the nodes.

H\T	0	1	2	3	4	5	6	7	8	9
64	55.5	74.3	78.2	80.3	77.8	80.4	80.1	76.0	79.8	78.4
112	53.1	73.2	78.9	78.4	76.4	71.5	77.4	76.3	76.6	77.3

Table 4: The accuracy rates for taking the summation with the dimensions of 64 and 112 on Cora

To apply the attention, the same attention may be applied regardless of the nodes. It may be good in semi-supervised learning with the high dimensional data(as Cora) by reducing the number of parameters in the dense network. However, it turned out that assigning the same attention within the same time-step was not effective and sometimes made the process of the training unstable.

4.4 Comparing with GCN

GCN imposes the tighter restrictions on the second-order Chebyshev polynomials with the rescaled eigenvalues, so GCN uses from 0 to 2-hop inputs by stacking two graph convolutional layers of which the parameters are not shared. As explained in Section 2, GRN does convolutional operations with the simple RNN of which the parameters are shared, and it uses two dense layers to compute the attention and the scores.

Models	**GRN(T=2)**	**GCN**
Accuracy (%)	80.9	80.3
Time (sec)	0.91	0.95
# of Parameters	96K	92K
# of hidden layers	64	64
Layer Norm	Used for the 1st layer	Used for the 1st layer
Learning rate	1e-2	5e-3
Early Stopping	5	50
Dropout rate	0.2	0.2

Table 5: The comparisons GRN(T=2) and GCN on Cora

The test of GCN was based on the Pytorch version of GCN(link), and the GCN used here was slightly modified by adding the Layer Normalization layer on the output of the first graph convolutional layer, and the window for early stopping was 50. The testing environment was Google Colabatory with Tesla K80 GPU, and both GRN and GCN are implemented in Pytorch.

The number of parameters for GRN is larger than those for GCN by 4,083. Since GRN has the first layer as the simple RNN, so it has the additional learnable parameters for $H^{(t)}$ to $H^{(t+1)}$ which is 64×64 matrix. GRN has the shared parameters for the input data randon-walk transitioned up to t times, but GRN has additional parameters to extract the features between the time-steps.

The performances of GRN and GCN are comparable, and both achieve similar accuracy results. The accuracy values of GRN and GCN are the average values for five randomized index tests. The split is the same as in table 1.

The execution times are both less than 1 second. GRN is based on RNN so that it may be slower than other convolutional models. However, GRN optimizes the parameters quite fast, so with the patience of early stopping 5, GRN finishes the training quite well (less than 30 epochs). The execution time per epoch of GCN is faster than that of GRN, but GCN needs the higher patience for early stopping to get the better accuracy (around 50). To sum up, GCN is known to be fast and efficient, but GRN also performs quite fast and efficient, even based on the RNN structure.

5 Conclusions

In this paper, we have introduced GRN on node classification tasks. GRN is based on the simple RNN with two dense layers, along with the layer normalization, dropout and early stopping. In the

connected graph, the stationary distribution of the Markov chain tends to cluster over a few centres, and GRN fully exploits this characteristic well by implementing the RNN. It does the convolutional operations in the spatial domains within the simple RNN architecture when the adjacency matrix is fixed. Also, GRN improves the performance by assigning specific attention over node and time-steps.

GRN achieves the comparable performances with the existing models such as GCN and GAT on the widely used semi-supervised learning tasks. GRN is also compared with GCN by setting the number of iterations equals two, so both use the data from up to 2-hop neighbours. They achieve similar results, and both perform less than 1 second, so GRN works comparable with GCN, but it is based on the simpler structure and it has more flexibility in designing the overall architecture.

References

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *arXiv preprint arXiv:1211.0053*, 2012.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.