Course Project on GCN

Lincheng Ji, 2017012775

May 17, 2020

Abstract

This is the reprouduction of the splendid model from work of Thomas M. Kipf and Max Welling *Semi-Supervised Classification with Graph Convolutional Networks* [8]. The reproduction is based on PyTorch machine learning package. The dataset **Cora** applied in the experiment is from *Collective Classification in Network Data* [4].

1 Introduction

The primary work of the paper is based on the findings of T. M. Kipf and M. Welling, who invented measures for classification in graph network.

Graph network classification is different from original graphical classification in that graphical data is mostly pixels or matrix which lines up into Euclidean Structure. Original classification models, such as the Convolution Neutral Network, apply convolution operators on the Euclidean Structure to substract features from pixels, as shown in Figure 1.

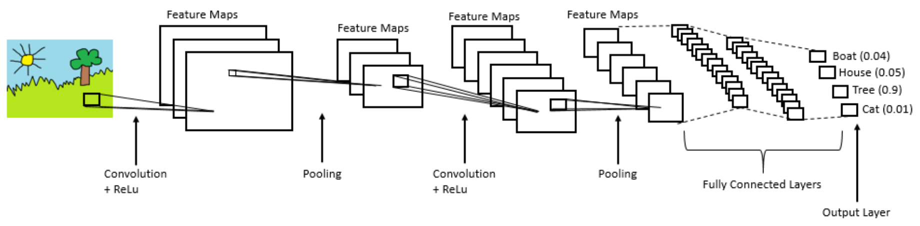

Figure 1: Convolution on pixels in CNN

2 Fourier Transformation and Graph Convolution

While graph data consists of node features and edge features, which are non-Euclidean Structure, the original convolution cannot apply on the data. In the work T. M. Kipf and M. Welling, the covolution operators are re-applied through different methods. The spectral graph theory provides fourier transformation and inverse fourier transformation, given discrete graph adjacency matrix A.

$$\hat{f} = U^T f \tag{1}$$

$$f = U\hat{f} \tag{2}$$

where U is the eigenvector matrix in normalized symmetry graph Laplacian

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T \tag{3}$$

The character of fourier tranformation brings sound transformation between convolution operators and fourier operators in non-Euclidean field.

$$\begin{aligned} \mathcal{F}(h * f) &= \mathcal{F}(h) \cdot \mathcal{F}(f) \\ (h * f)_G &= U \left(U^T(h) \odot U^T(f) \right) \end{aligned} \tag{4}$$

where $\odot$ is the Hadamard Product.

The calculation of graph convolution can be simplified throught the work of Hammond et al. on *Wavelets on Graphs via Spectral Graph Theory*[6], gaining Equation 5.

$$(h * f)_{G'} = \sum_{k=0}^{K} h'_k T_k(\tilde{L}) x \tag{5}$$

where $T_k(x)$ denotes the kth Chebyshev polynomials, h''s denote the Chebyshev coefficient vectors, $\tilde{L} = \frac{2}{\lambda_{max}} L - I_N$, λ_{max} denotes the maximum eigenvalues of L, the normalized symmetry Laplacian. The detailed exploration should refer to work of Kipf et al.[8]

3 Graph Convolution Layer and GCN model

A neutral network based on graph convolution consists of layers of graph convolution and non-linear activation function. To reproduct the work of Kipf et al.[8], a neutral network is modeled by the forward function in Equation 6.

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A} \cdot \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W(1)\right) \tag{6}$$

4 Model Implement and Experiment

The reproduced model is based on PyTorch package, which contains built-in neutral network model frameworks. The codes are attached in Appendix.

To regain the training result in the work of Kipf et al[8], the model implements the same hyperparameters, as shown in Table 1.

random seed	hidden units	dropout rate	learning rate	weight decay for L2 loss	epochs
24	16	0.5	0.01	5e-4	200

Table 1: Hyperparameters in GCN Reproduction

5 Comparison and Result

The training is carried out on **Cora** dataset[4] in both reproducable random split measures. The result performance is calculated on the basis on 20 trainings, which

The original paper performance and comparison is shown in Table 2.

model	Cora	Citeseer
ManiReg[3]	59.5	60.1
SemiEmb[5]	59.0	59.6
LP[2]	68.0	45.3
DeepWalk[7]	67.2	43.2
ICA[1]	75.1	69.1
Planetoid*[9]	75.7	64.7
GCN (paper)	81.5	70.3
GCN (paper, rand splits)	80.1±0.5	67.9±0.5
GCN (reproduced)	82.5	73.1

Table 2: Comparison of GCN and Other Model Performance

The results of reproduction generally matches the original performance. The reproduction appears to have slightly higher accuracy rate, which can be caused by sample shuffling and difference in train-valid-test ratio. The reproduction is in all satisfactory.

6 Discussion of GCN

Kipf et al.[8] modeled and trained the GCN on the whole graph set; whereas, the model reveals two shortcomings. The whole graph is stored in CPU/GPU in every epochs, which is computationally expensive and makes it hard to train on large-scale graph network. Meanwhile, mini-batch gradient descent cannot apply to the model, which should increase the stochasticity of the model.

7 Possible Improvements

Improvements on mini-batch, which is in considerable complexity and beyond the mathematical ability of the author, are revealed by reviews on other researches.

7.1 GCNSAGE

Inductive Representation Learning on Large Graph by Hamilton et al.[10] presents GCNSAGE model, which utilizes low-dimensional node embeddings and feature sampling and aggregating.

The implementation of each training iteration in GCNSAGE is in Table 3.

- Randomly select a minibatch $\upsilon_B \in \upsilon_L$ of nodes;
- Build a computation graph that only contains the activation $h_\upsilon^P(l)$ and $\bar{h}_\upsilon^{(l)}$ needed for the current minibatch;
- Get the predictions by forward propagation as $Z^{(l+1)} = \left(\hat{P}^{(l)}(H^{(l)} - \bar{H}^{(l)}) + P\bar{H}^{(l)}\right) W^{(l)}$;
- Get the gradients by backward propagation, and update the parameters by SGD;
- Update the historical activations;

Table 3: GCNSAGE Implementation

7.2 FastGCN

FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling by Chen et al.[12]

7.3 N-GCN

[11]

References

[1] Qing Lu and Lise Getoor. "Link-Based Classification". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 496–503. ISBN: 1577351894. DOI: 10.5555/3041838.3041901.

[2] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 912–919. ISBN: 1577351894. DOI: 10.5555/3041838.3041953.

[3] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. "Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples". In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 2399–2434. ISSN: 1532-4435. DOI: 10.5555/1248547.1248632.

[4] Prithviraj Sen et al. "Collective Classification in Network Data". In: *AI Magazine* 29.3 (Sept. 2008), p. 93. DOI: 10.1609/aimag.v29i3.2157.

[5] Jason Weston, Frédéric Ratle, and Ronan Collobert. "Deep Learning via Semi-Supervised Embedding". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1168–1175. ISBN: 9781605582054. DOI: 10.1145/1390156.1390303. URL: https://doi.org/10.1145/1390156.1390303.

[6] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. "Wavelets on Graphs via Spectral Graph Theory". In: *ArXiv* abs/0912.3848 (2009).

[7] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 701–710. ISBN: 9781450329569. DOI: 10.1145/2623330.2623732. URL: https://doi.org/10.1145/2623330.2623732.

[8] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[9] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. “Revisiting Semi-Supervised Learning with Graph Embeddings”. In: *CoRR* abs/1603.08861 (2016). arXiv: `1603.08861`. URL: `http://arxiv.org/abs/1603.08861`.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *CoRR* abs/1706.02216 (2017). arXiv: `1706.02216`. URL: `http://arxiv.org/abs/1706.02216`.

[11] Sami Abu-El-Haija et al. “N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification”. In: *CoRR* abs/1802.08888 (2018). arXiv: `1802.08888`. URL: `http://arxiv.org/abs/1802.08888`.

[12] Jie Chen, Tengfei Ma, and Cao Xiao. “FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling”. In: *CoRR* abs/1801.10247 (2018). arXiv: `1801.10247`. URL: `http://arxiv.org/abs/1801.10247`.

Appendix A GCN Model Code

```
import math

import torch
from torch.nn.modules.module import Module
from torch.nn.parameter import Parameter
import torch.nn.functional as F


class GCN(Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc2(x, adj)
        return F.log_softmax(x, dim=1)


class GraphConvolution(Module):
    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' (' \
               + str(self.in_features) + ' -> ' \
               + str(self.out_features) + ')'
```