Project Report : CayleyNet, another spectral graph convolution

Théau Blanchard
Institut Polytechnique de Paris
Palaiseau, France
blanchard.theau@live.fr

Abstract

Graph are a powerful general representation for data that may lack global structure. This led to the extension of neural network such as CNN to graph data. Here we present and analyze a new type of convolution for graph deep learning based on the spectral representation of the graph originally presented in the paper *CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters* by Levie, Monti *et. al.* [3]. This Cayley net is based on the eponym transform of the graph laplacian. This architecture provides improved performances over other spectral approaches but at a significantlty higher computational cost.

CCS Concepts: • **Computing methodologies** → *Spectral methods*; Neural networks.

Keywords: graph neural networks, convolution, rationnal filter, polynomials, eigenvalues

1 Introduction

Data comes in all shape and size and usually does not exhibit an Euclidean structure. Social network, proteins, images, 3D shapes can all be represented as graph despite not necessarily having such a structure. Therefore all the learning operations usually defined on texts or images such as convolution are not easilly extended to graph data.

1.1 Problem formulation

Let $\mathcal{G} = (1, \dots, n, \mathcal{E}, W)$ be an undirected weighted graph, with W its symmetric adjacency matrix. Let $f \in \mathbb{R}^{n\times p}$ be a signal on the graph, ie each node is assigned a signal with p features. The objective of defining a convolution on a graph is, by analogy with regular convolutions on Euclidean data, to find a function $H : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times q}$ such that the output of H taken at a node, depends on its neighbourhood.

A rather intuitive approach is given in [2]. Here the output signal f' is given, node-wise, by

$$f'_i = \Theta^T \sum_{j\in\mathcal{N}_1(i)\cup\{i\}} \frac{w_{j,i}}{\sqrt{\hat{d}_j\hat{d}_i}} f_j \tag{1}$$

Where $\hat{d}_j = 1 + \sum_{j\in\mathcal{N}_1(i)} w_{j,i}$ and $\mathcal{N}_1(i)$ is the 1-hop neighbourhood of the node i. Here $\Theta \in \mathbb{R}^{p\times q}$ is the parameter we are learning.

We will use this approach as a simple baseline further on denoted by GCN.

This definition is rather restrictive. First the notion of localization is very strict. Because we only consider the 1-hop neighbourhood, these kind of convolution will struggle to detect long distance meaningful relationship between nodes. Then, if one were to increase the 1-hop to a k-hop, given that there is only one parameter, it is still impossible to distinguish between long and short range relationship. Yet this approach can be easily implemented so as to be GPU friendly thanks to the sparsity of natural graphs.

1.2 Spectral Graph

In Fourier analysis, we can easily show that the Fourier basis function are eigenvectors of the Laplacian operator. Then given the property of the Fourier transform $\mathcal{F}$ and the convolution operator, we have that the convolution between two signal on Euclidean domain f and g can be done instead in the Fourier domain : $f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \odot \mathcal{F}(g))$, where $\odot$ is the point-wise multiplication. Hence in spectral graph theory we aim to mimic such a domain transform so as to define a convolution operator between to signal on graph.

Let's consider the Laplacian of a graph $\Delta = D - W$ where $D = \text{diag} \sum_{i\neq j} w_{j,i}$ is the degree matrix. It admits a decomposition $\Delta = \Phi\Lambda\Phi^T$ where $\Phi = (\phi_1, \dots, \phi_n)$ are the eigenvectors and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ are the eigenvalues of the graph. Thus an analogy to Fourier transform can be defined on a signal as $\hat{f} = \Phi^T f$. Hence we can define the graph spectral convolution as

$$g * f = \Phi(\Phi^T g \odot \Phi^T f) = \Phi\text{diag}(\hat{g}_1, \dots, \hat{g}_n)\hat{f}$$

Given this definition we can construct a deep learning convolution layer. Let $F_{in} = (f_1^{in}, \dots, f_p^{in})$ be the input signal and $F_{out} = (f_1^{out}, \dots, f_q^{out})$ be the output signal such that :

$$f_i^{out} = \xi\left(\sum_{j=1}^{r} \phi_j \hat{G}_{i,j} \phi_j^T f_j^{in}\right)$$

Where ξ is an activation function (eg. ReLU),$r \in \mathbb{N}$ and the $\hat{G}_{i,j} = \text{diag}(\hat{g}_{i,j,1}, \dots, \hat{g}_{i,j,n})$ are the learnable parameters.

Yet as it is, this approach comes with significant drawback. First it requires the very expensive computation of the eigenvectors of the graph. Then it is consequently basis dependent in the sense that the parameters are learned given the basis thus making transfer learning impossible. Finally, the output signal will necessarily be of the same shape as the input signal.

1.3 Frequency based learning

Instead of using only the eigenvectors of the graph, other approaches also make use of the graph frequencies (ie eigenvalues of the Laplacian). There, all the parameters $\hat{G}_{i,j} = \text{diag}(\hat{g}_{i,j,1}, \dots, \hat{g}_{i,j,n})$ are replaced by a parameterized function of the frequencies such that $\hat{G}_{i,j} = g_\Psi(\Lambda)$.

Another key propriety we're looking for is to avoid the computation of the eigenvectors. We can show that if g is rational function – which in the matrix case amounts to addition, powers, inversions and multiplications by a scalar –, then $g_\Psi(\Delta)f_j = \phi_j g_\Psi(\Lambda)\phi_j^T f_j$. Hence, instead of diagonalizing the graph Laplacian, we can use it directly as it is.

These two properties allow to study the frequency response of a filter by only computing $g(\lambda)$ while being able to apply it to signal with a minimal computational overhead. So finally, the output signal is given by

$$f_i^{out} = \xi\left(\sum_{j=1}^{r} g_\Psi(\Delta) f_j^{in}\right)$$

And, as we will study, it requires minimal modification to be also able to reduce the feature dimension.

So to recap what we've seen so far. We are able to define convolution between two signal on graph, thus making it suitable for deep learning. The task at hand is now to design function g_ψ that are efficient and provide improved performances compared to a classic GCN approach (Eq. (1)). In particular we should be able to make use of long range information while still being discriminative.

1.4 An example : ChebNet

Such a function is introduced in [1] and make use of the Chebyshev polynomials. Their function is defined on $[-1, 1]$. It is then applied to a rescaled Laplacian $\tilde{\Delta}$ so that all eigenvalues lie in this interval. The function is given by :

$$g_\alpha(\lambda) = \sum_{k=0}^{r} \alpha_k T_k(\lambda)$$

Where r is the order considered, α is the learned parameter and $T_j(\lambda) = 2\lambda T_{j-1}(\lambda) - T_{j-2}(\lambda)$ denotes the Chebyshev polynomial of degree j defined in a recursive manner with $T_1(\lambda) = \lambda$ and $T_0(\lambda) = 1$

With notation more akin to deep learning. If $F_{in} \in \mathbb{R}^{n\times p}$ is the input signal and $F_{out} \in \mathbb{R}^{n\times q}$ is the output signal then with $\alpha_k \in \mathbb{R}^{q\times p}$:

$$F_{out} = \xi\left(\sum_{k=0}^{r} T_k(\tilde{\Delta}) F_{in} \alpha_k^T\right)$$

Because we use the Laplacian up to the power r, the filter can deal with neighbours up to r-hop.

2 CayleyNet

Because the ChebNet [1] only works with the rescaled Laplacian, it causes some frequency bands to be squished together. In some cases it makes it so that all the important frequencies for the task at hand are indistinguishable from each other. And because of the polynomial nature of the filter, you need inversely as many order (r) as the size of the band of frequency you want to distinguish.

The filters introduced in [3] builds on the same idea as the ChebNet [1] but tries to both be able to distinguish small bands of frequencies while also being able to work with large bands.

2.1 Formulation

Given an order r, and learnable parameters $(h, c_0) \in \mathbb{R}^2$ and $(c_1, \dots, c_r) \in \mathbb{C}^r$, we define a cayley polynomial as :

$$g_{h,\mathbf{c}}(\lambda) = c_0 + \sum_{k=0}^{r} c_k (h\lambda - i)^k (h\lambda + i)^{-k}$$

Where i is the imaginary complex unit.

Let $C(h\Delta) = (h\Delta - i\mathbf{I})(h\Delta + i\mathbf{I})^{-1}$

With notation more akin to deep learning, then with $c_0 \in \mathbb{R}^{q\times p}$, $c_k \in \mathbb{C}^{q\times p}$ and $h \in \mathbb{R}$, the output signal is:

$$F_{out} = F_{in} c_0^T + \sum_{k=1}^{r} C(h\Delta)^k F_{in} c_k^T$$

2.2 Properties

Because the Cayley transform $x \longrightarrow \frac{x-i}{x+i}$ is a bijection between $\mathbb{R}$ and the unit complex circle. The spectrum of the Laplacian is then in the lower half of this circle. This thus mean that each power of $C(h\Delta)$ amounts to a multiplication of the spectrum by a pure complex harmonic such that:

$$C^j(h\Delta) = \Phi \text{diag}\left(C(h\lambda_1)^j, \dots, C(h\lambda_n)^j\right)\Phi^T$$

Therefore one of the pros of using multiple order is that is allows for a diverse distribution of the frequencies on the unit circle. This way wide frequency bands can be detected because higher order polynomial will eventually cluster together far appart frequencies, as shown in Figure 1.

Nevertheless, the initial distribution of the frequency can be analogous to the Chebyshev case where they're squished. In such case the zoom-parameter h comes in handy. The greater h is, the more the spectrum of $h\Delta$ is spread apart, thus the more spaced are the small eigenvalues of $C(h\Delta)$ while the big eigenvalues are pushed toward 1. On the contrary, the smaller h is, the more spaced are the big eigenvalues of $C(h\Delta)$ while the small eigenvalues are pushed toward -1. Consequently, this parameter is trully able to zoom on the spectrum. Thus tuning it to fit the data overcomes the issue of ChebNet on discriminating specific bands of frequencies as shown in Figure2.

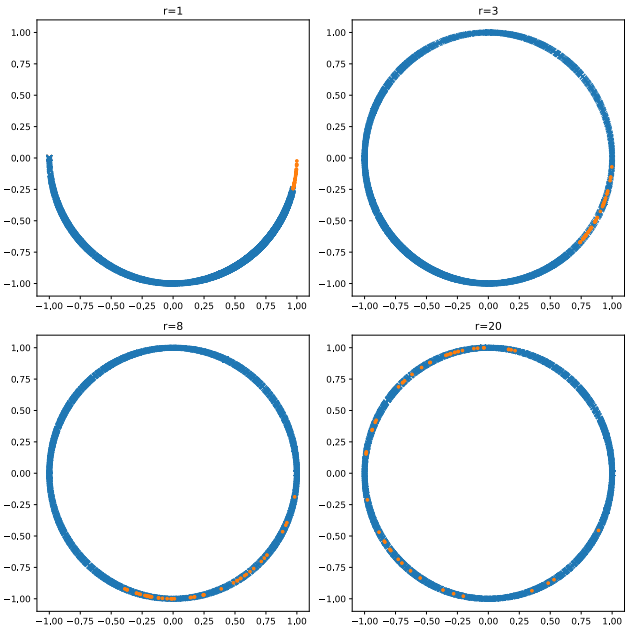

Figure 1. Eigenvalues of the unnormalized Laplacian $h\Delta$, with $h = 1$ on the Cora data set mapped by Cayley transform for different order r. $C(h\Delta)^r$. The 50 most import eigenvalues are in blue. Eventually blue and orange frequencies are mixed up.

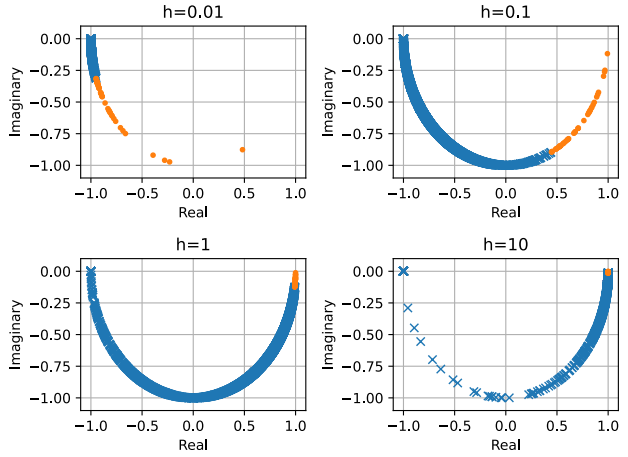

Figure 2. Eigenvalues of the unnormalized Laplacian $h\Delta$ on the Cora data set mapped by Cayley transform.The most important frequencies are marked in blue.

2.3 Computation method

At the core of the method lies the need to compute the values $C(h\Delta)^k f$ which implies the inversion of the matrix $h\Delta + i\mathbf{I}$. We do no wish to make such a calculation for two reasons. First, it requires $O(n^3)$ operations, which is a magnitude of operations we wanted to avoid in the first place by not computing the eigenvectors of the Laplacian. Then we need to consider that most natural graphs are very sparse. For example the unnormalized Laplacian of the Cora dataset is empty at 85%. Its inverse is almost 100% full. Thus actually using the exact formula using the inverse matrix would add a huge computational cost to normally really sparse operations.

To overcome this issue we solve the following linear recursive system,

$$y_0 = f; (h\Delta + i\mathbf{I})y_j = (h\Delta - i\mathbf{I})y_{j-1}; j = 1, \dots, r \qquad (2)$$

For which we approximate the solution $\tilde{y_j}$ using the Jacobi method for K iterations. Let $J = -\text{diag}(h\Delta+i\mathbf{I})^{-1}\text{Offdiag}(h\Delta+i\mathbf{I})$. Then this method yields :

$$\begin{aligned} \tilde{y}_j^{(k+1)} &= J\tilde{y}_j^{(k)} + b_j \\ b_j &= \text{diag}(h\Delta + i\mathbf{I})^{-1}(h\Delta - i\mathbf{I})\tilde{y}_{j-1} \end{aligned} \qquad (3)$$

Initialized with $\tilde{y}_j^{(0)} = b_j$, finishing after K iterations such that $\tilde{y}_j = \tilde{y}_j^{(K)}$

Thus the applied filter is given by $c_0 f + 2\mathcal{R}e\left(\sum_{k=1}^{r} c_j \tilde{y}_j\right)$

Beside allowing for a GPU friendly implementation of the layer, this method also brings strong theoretical results. Indeed, with an exact inversion each node interacts once with its neighbours for each order, thus in the end this layer make use of r-hop neighbourhoods. With the Jacobi method, there is a matrix multiplication involving the Laplacian for each iteration plus one for the initialization. So in the end a node will have interacted with $(K + 1)r$ neighbours.

So finally for the same filter order and only twice the number of parameters (because of the complex coefficient), the CayleyNet is able to theoretically overcome all of the flaws of the ChebNet which was back then State of the Art in spectral graph learning.

3 Results and proposed implementation

An important flaw of the original paper is that they did not submit the code they used to make their experiment. I therefore spent an important part of my time implementing their methods. I choose to develop their method using the popular *Pytorch Geometric* library. All code is made to be easily understood and is well documented. It is clearly suboptimal in terms of speed and memory consumption both because of my ability but also for numerical reasons detailed further on. In the process I also had to come up with a fast and scalable implementation of the Jacobi method on sparse matrices in COO format.

3.1 The Jacobi method

As mentioned previously, in order to avoid to compute the expensive inversion of the matrix $(h\Delta + i\mathbf{I})$, we make use of the Jacobi method to solve linear system. This iterative

method converges on the assumption that the matrix A of the linear system $Ax = b$ has a spectral radius smaller than 1. For a graph, because the matrix $C(h\Delta)$ is unitary, it will always verify this property. The author even provided an upper bound on the error made by the method.

Let $J = -\text{Diag}(h\Delta + i\mathbf{I})^{-1}\text{Offdiag}(h\Delta + i\mathbf{I})$ be the matrix used at each iteration, $\kappa = ||J||_2 < 1$ the spectral radius of J, K the number of iterations made, $\mathbf{Gf}$ be the signal filtered using an exact inversion, and $\tilde{\mathbf{Gf}}$ the filtered signal using the Jacobi method. Then we have the following result:

$$\frac{||\mathbf{Gf} - \tilde{\mathbf{Gf}}||_2}{||\mathbf{f}||_2} \leq 2M\kappa^K \tag{4}$$

Where $M = \sqrt{n}\sum_{j=1}^{r} j|c_j|$ in the general case and $M = \sum_{j=1}^{r} j|c_j|$ if the graph is regular.

Empirically we can see that this bound is pessimistic and that the convergence is faster, see Fig3.

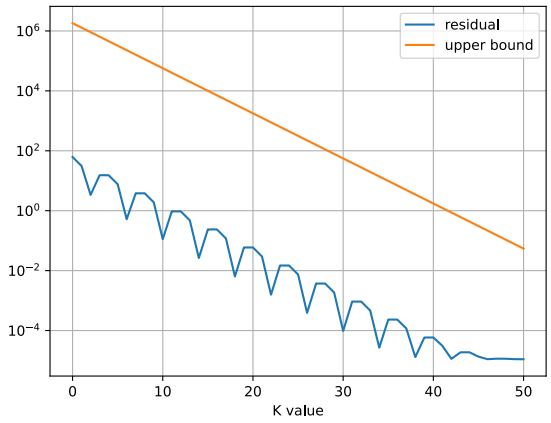

Figure 3. Convergence of the Jacobi method on the Cora dataset.

As it is, the method already offers improvement speed performances compared to a resolution requiring to inverse the matrix. But it is possible to optimize the algorithm by making use of the structure of natural graphs. Generally such graphs are really sparse in the sense that a node only has a very small degree compared to the number of nodes in the whole graph. So the adjacency matrix, and consequently the Laplacian, will be mostly filled with zeros. It is thus possible to adapt the method to such matrix, offering significant speed up, see Table1.

3.2 Learning setting

Because we make use of learnable parameters, we do not need to compute too many Jacobi iterations. The parameters will "compensate" the error, initially already small, made by not reaching full convergence off the method. This allows for significantly less operation thus and improved speed.

Table 1. Relative speed of the Jacobi methods compared to the full inversion on the Cora dataset with $K = 10$

Method used	Time	Speed ratio
Full inversion	$3.41s \pm 107ms$	1.0
Dense matrix	$14.s \pm 340ms$	0.25
Sparse matrix	$700ms \pm 46.5ms$	4.9

Another key point about using less Jacobi iteration is the memory consumption. In a learning setting with modern library such as Pytorch or Tensorflow we make use of some kind of automatic differentiation to perform the optimization. This requires to store every operations done on Tensors using the input or the parameters in a computational graph. So, because each Jacobi iteration needs both the input signal and the matrix $\mathbf{J}$ that is built on the parameter h, we need to store the operation resulting in a memory consumption that will be quickly too high for a modest GPU. Moreover, because it is adding a lot of layers to the computational graph, the backprogration method is much more prone to vanishing/exploding gradient problems that can be observed in very deep networks. On my experiments I did not face such problems with the gradient, but I was not able to achieve more than 5/6 Jacobi iterations as my GPU RAM is too small.

3.3 Node classification

The ChebNet layer is specifically designed to be able to work with information on both long and short range. In the spectral domain this amounts to being able to use all type of frequency bands.

To test these hypothesis and the performances of such a method over other methods we will evaluate its performance at node classification. We use the Cora dataset [4]. It consists of 2708 nodes representing research papers whose linked are citations between the papers. Each node is assigned a 1433 feature vector representing the content of the paper. The goal is to classify each node into one of the 7 classes in a semi-supervised fashion.

I used a similar architecture for the GCN, the ChebNet and the CayleyNet consisting of two layers of convolutions with 16 features and ReLU activation with a dropout with $p = 0.5$. Optimization is done using the standard training split of the dataset and the Adam Optimizer with a learning rate of 5×10^{-3} for 100 epochs. The order of the filter considered varies so as to either match the order or the number of parameters . Remember that the Jacobi method allows for broader interactions, thus an effective order K times higher.

The results of Table2 show that this method offers indeed significant improvement over previous spectral methods. However I was not able to reproduce the results of the original paper. The accuracy are lower but more importantly the number of parameter is significantly different, even though

I've used the already implemented ChebConv layer of torch-geometric.

3.4 Influence of the order

The choice of the order considered is a major hyper parameter to tune. It brings the possibility to explore broader neighbourhood at a higher cost. Also, because of the increased number of parameters, a high order filter is much more prone to over-fitting. Such a phenomenon must be taken into account when designing at network as learning with this type of spectral filter brings rapid convergence on the training data. My experiments showed that for the type of dataset used, ie pretty small here, it is wiser to use low-order filter as they mitigate the overfitting during training. I do not have the computational capabilities to experiment on bigger dataset and larger order. This goes to show still that the tuning of this hyper parameters is of great importance for the performance of any models built using this method.

3.5 Influence of the h zoom-parameter

As we explored earlier, the zoom parameter is helpful in order to untangle clusters of important frequencies. Hence its optimal value should depend on the graphs considered. The Fig2 suggests that for the CORA dataset, this value is around 1. But experimentally we observe that when the initial value of this parameter is far away from this value, the network is not able to bring it toward this optimal value. See 5

This shows two things. First, once again, the initialization of the h parameter should be treated an hyper parameter initialization optimization problem. Then, the zoom parameter is indeed a key element to the success of the method as Table 4 shows.

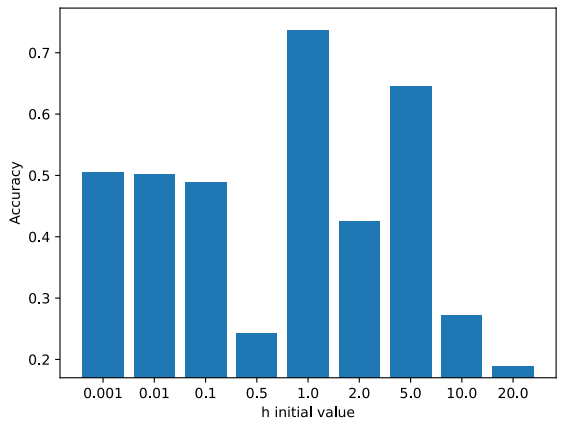

Figure 4. Influence of the initial value of h on the final accuracy. Here r=6,K=5 for 50 epochs.

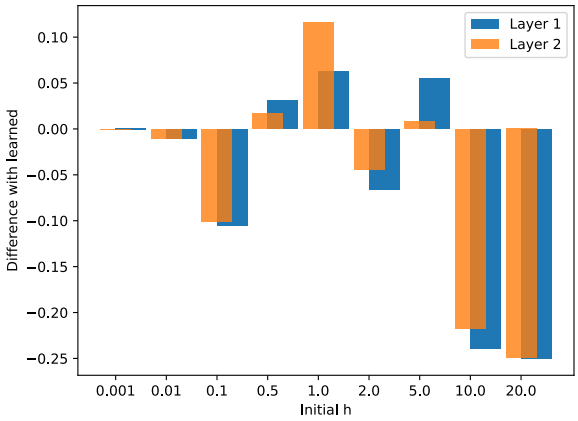

Figure 5. Difference between the final learned value of h and its initial value.Here r=6,K=5 for 50 epochs.

4 Conclusion

Finally the CayleyNet method brought new ideas to the spectral graph learning community. The authors tried to tackle the issue that the current polynomial based approach had both theoretically and complexity-wise. This approach has shown to be able to perform slightly better than other spectral method. Yet is requires a lot of hyper-parameter tuning that impact significantly the final performances. Moreover, if on paper using the inverse of a sparse matrix is not a problem, in practice it actually requires drastic changes to the implementation that the Jacobi method is not totally able to circumvent. Consequently, this new type of convolution has not been largely adopted. Hence the lack of modern implementation.

References

[1] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *CoRR* abs/1606.09375 (2016). arXiv:1606.09375 http://arxiv.org/abs/1606.09375

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

[3] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. 2017. CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters. *CoRR* abs/1705.07664 (2017). arXiv:1705.07664 http://arxiv.org/abs/1705.07664

[4] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval* 3, 2 (01 Jul 2000), 127–163. https://doi.org/10.1023/A:1009953814988

A Research Methods

All experiments were conducted using a simple gaming laptop. CPU : AMD Ryzen 5 4600h (3.00 GHz), GPU : Nvidia 1650 GTX Ti (4 Go RAM), RAM : 16Go.

Table 2. Test accuracy of node classification of the Cora dataset

Method	Number of Parameters	Order	Accuracy
GCN	23K	-	79.7%
ChebNet	253 K	11	46.9%
ChebNet	138 K	6	70.2%
$\text{CayleyNet}_{K=5}$	299 K	6	75.0%
$\text{CayleyNet}_{\text{Exact Inversion}}$	299 K	6	42.5%

It appeared during my experiments that building complex linear layer by myself rather than using the pre-existing PyTorch layer showed improved performances. I suggest that it is linked with how the backprogation is actually handled for complex number.

B Online Resources

All code is available on my github. There you will find an overview of the Jacobi method and several experiments with the CaleyNet. All results and figures presented here are easily reproductible.

Received 1 January 2023