LEARNING ANISOTROPIC FILTERS ON PRODUCT GRAPHS

Vignac Clément, Frossard Pascal
LTS4
EPFL, Lausanne, Switzerland
{clement.vignac, pascal.frossard}@epfl.ch

ABSTRACT

The extension of convolutional neural networks to irregular domains has paved the way to promising graph data analysis methods. It has however come at the expense of a reduced representation power, as most of these new network architectures can only learn isotropic filters and therefore often underfit the training data. In this work, we propose a method for building anisotropic filters when learning representations of signals on a cartesian product graph. Instead of learning directly on the product graph, we factorize it and learn different filters for each factor, which is beneficial both in terms of computational cost and expressivity of the filters. We show experimentally that anisotropic Laplacian polynomials indeed outperform their isotropic counterpart on image classification and matrix completion tasks.

1 INTRODUCTION

Convolutional neural networks (CNNs) have become a key component of most deep learning systems thanks to their ability to learn local, shared and multiscale structures. They typically operate on grid-like domains by applying the same local filters at every location. It is however not trivial to extend the convolution operation to irregularly structured data. In order to solve this problem and analyze data on irregular domains, many iterative schemes have been proposed in which each node sums the features learned by its neighbors. These new methods have recently been grouped under the umbrella of graph neural networks (Battaglia et al., 2018).

Unfortunately, not every function on a graph can be learned using this framework. Xu et al. (2018) showed that, in the limit of an infinite number of layers, graph neural networks are only as powerful as the Weisfeiler-Lehman isomorphism test (Weisfeiler & Lehman, 1968). In practice, their expressivity is even lower, since most architectures use very few layers. Because the summation operation is invariant to permutations of its input, each graph network layer has unwanted invariance properties (Kondor et al., 2018). In particular, as observed by Levie et al. (2017), graph networks applied to grids compute filters with a spherical shape, so that they are commonly referred to as isotropic filters.

Isotropy is not necessarily a problem if it corresponds to a correct prior about the task. Unfortunately, a spherical shape is completely at odds with the filters learned by the first layers of CNNs (in images for example), which often act as edge detectors (Krizhevsky et al., 2012). In the experimental section, we show that graph neural networks indeed perform much worse than CNNs with the same number of parameters, implying that isotropy is an harmful inductive bias for some tasks.

Since there is no obvious notion of direction or orientation for arbitrary graphs, designing anisotropic or oriented filters is challenging. Fortunately, some important real-world applications involve graphs that can be represented as the cartesian product of factors. Such products usually appear when a manifold or a notion of dimension supports the graph construction, in fields such as computer vision, recommender systems and time series on networks. When dealing with such data, we propose to use the factors to introduce directions in the product graph and treat the edges differently according to them. We show that anisotropic filters indeed outperform their isotropic counterpart on image classification (MNIST and CIFAR10) and semi-supervised learning (Movielens100k), and explain why they still fall behind standard CNNs when applied to grids.

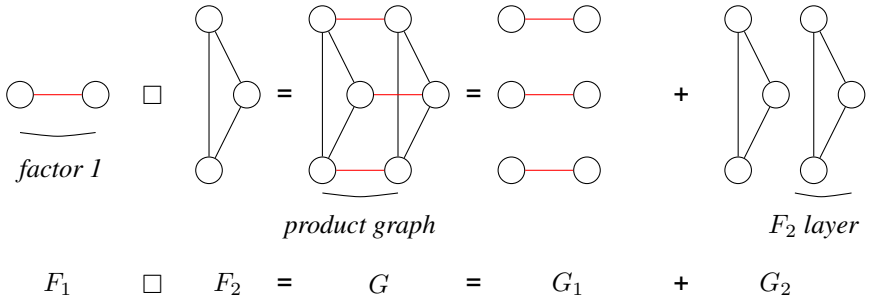

Figure 1: Product graphs are structured in layers. Because they correspond to different dimensions, the edges benefit from being treated differently according to the factor they belong to.

2 Anisotropic filters for cartesian products

Breaking the isotropy of current graph networks means trying to learn oriented filters. It is challenging because of the absence of a natural notion of direction for unstructured graphs. However, when the graph is a product of factors, we can provide a mathematical definition that captures the idea of anisotropic filters.

Although most graphs are prime, cartesian products arise when some notion of dimension is underlying the graph construction. Apart from grid graphs (images and videos), which are mostly useful to visualize and interpret graph based methods, time varying signals on fixed networks are naturally represented by cartesian products (Perraudin et al., 2017): they are used to analyze transportation networks (Li et al., 2017) (Zhang et al., 2018), brain signals or epidemic networks, to name a few. Moreover, the notion of dimension need not correspond to a physical reality: recommender systems can be modeled using the product of a similarity graph for users and another for the products to recommend, so that a signal can be associated to a node whenever a user rates an item.

2.1 Definition and properties

Definition We consider weighted, undirected graphs without self-loops or multiple edges. $\mathbb{V}$ stands for vertex sets and $\mathbb{E}$ for edge sets. The cartesian product $\mathcal{G} = \mathcal{F}_1 \,\square\, \mathcal{F}_2$ of two factors $\mathcal{F}_1 = (\mathbb{V}_1, \mathbb{E}_1)$ and $\mathcal{F}_2 = (\mathbb{V}_2, \mathbb{E}_2)$ is the graph with vertex set $\mathbb{V} = \mathbb{V}_1 \times \mathbb{V}_2$, where the vertices (u_1, u_2) and (v_1, v_2) are adjacent if $(u_1, v_1) \in \mathbb{E}_1$ and $u_2 = v_2$, or $(u_2, v_2) \in \mathbb{E}_2$ and $u_1 = v_1$. The cartesian factors of a weighted graph can be recovered in a time $O(|\mathbb{E}| + |\mathbb{V}| \log |\mathbb{V}|)$ using the work of Imrich & Peterin (2007), by replacing the breadth-first search with Dijkstra's algorithm.

Coordinates and layers By associativity, we can write a product of p graphs as $\mathcal{G} = \mathcal{F}_1 \,\square\, ... \,\square\, \mathcal{F}_p$. The vertices of such a graph are the tuples $v = (v_1, ..., v_p)$ where each $v_i \in \mathbb{V}_i$ is called the i-th *coordinate* of v. By grouping together the vertices that differ from a given vertex v by only the i-th coordinate, we define the *layer* of factor $\mathcal{F}_i$ through v. What makes cartesian products prone to the construction of anisotropic filters is that these layers give a structure to the graphs. They can be seen as directions that are consistent globally, so that sharing parameters between edges that belong to the same factor is meaningful. Finally, we define the subgraphs $\mathcal{G}_i$ as the disconnected graphs with vertex set $\mathbb{V}$ that are the sum of all the $\mathcal{F}_i$ layers, as in Figure 1.

Tool for efficient computations For the sake of simplicity, we restrict the presentation to the product of two factors. The adjacency matrix (as well as the combinatorial Laplacian) of a cartesian product and its powers can be written in a condensed way using the Kronecker product $\otimes$:

$$\boldsymbol{A}_{\mathcal{G}} = \underbrace{\boldsymbol{A}_1 \otimes \boldsymbol{I}_2}_{\boldsymbol{A}_{\mathcal{G}_1}} + \underbrace{\boldsymbol{I}_1 \otimes \boldsymbol{A}_2}_{\boldsymbol{A}_{\mathcal{G}_2}} \quad \text{and} \quad \boldsymbol{A}_{\mathcal{G}}^k = \sum_{i=0}^{k} \binom{k}{i} \boldsymbol{A}_1^i \otimes \boldsymbol{A}_2^{k-i}$$

where $\boldsymbol{A}_i$ is the adjacency matrix of $\mathcal{F}_i$, and $\boldsymbol{I}_i = \boldsymbol{A}_i^0$ the identity matrix of size $|\mathbb{V}_i| \times |\mathbb{V}_i|$.

Table 1: Different ways to compute filters with receptive fields of size 2 along each factor.

Summation	Composition	Tree exploration
$(f_1 + f_2)(f_1 + f_2)$	$f_1 f_2 f_1 f_2$	*concatenate*$(f_1^2 f_2^2,\ f_1 f_2 f_1 f_2,\ f_1 f_2^2 f_1, ...)$

2.2 ANISOTROPIC FILTERS

Since cartesian product graphs can be factorized, we propose to use this information to break the isotropy of graph networks. Instead of learning a network acting on $\mathcal{G}$, we learn two networks f_1 and f_2 acting on $\mathcal{G}_1$ and $\mathcal{G}_2$. This way, edges belonging to different factor are treated differently. This is important in cartesian products because the factors are often structurally very different: anisotropic filters can for example permit to decouple the time and spatial dimensions in a time series, or the social network and an item similarity graph in a recommender system.

How to organize the computations on the different subgraphs? In order to obtain graph networks of depth d, we need to decide along which subgraph to apply each of the d message passing steps. When the filters are non linear, the output will depend not only on how many times each subgraph is chosen, but also on the order of application of the filters. Many choices are possible: the feature maps learned by the different networks can be summed up, applied alternatively, or several combinations can be concatenated to obtain richer filters, as shown in Table 1. Note that this additional complexity disappears when the filters are linear. In this case, the two graph networks commute, which makes the choice of architecture simpler. For this reason, we make the construction of anisotropic filters explicit for the popular class of Laplacian polynomials.

Classically, when learning polynomials of the Laplacian of $\mathcal{G}$, we learn the coefficients $(c_i)_{i=1}^k$ of the function:

$$f(s) = \sum_{i=1}^{k} c_i \, \boldsymbol{L}_G^k \, s = \sum_{i=1}^{k} c_i \sum_{j=0}^{i} \binom{i}{j} \boldsymbol{L}_1^j \otimes \boldsymbol{L}_2^{i-j} \, s$$

Instead, our proposition amounts to learning a matrix (c_{ij}) of filter coefficients corresponding to the function:

$$f(s) = \sum_{0 \leq i+j \leq k} c_{ij} \, \boldsymbol{L}_{\mathcal{G}_1}^i \boldsymbol{L}_{\mathcal{G}_2}^j \, s = \sum_{0 \leq i+j \leq k} c_{ij} \, (\boldsymbol{L}_1^i \otimes \boldsymbol{L}_2^j) \, s$$

Thanks to the commutativity of $\boldsymbol{L}_{\mathcal{G}_1}$ and $\boldsymbol{L}_{\mathcal{G}_2}$, the filter $\boldsymbol{L}_1^i \otimes \boldsymbol{L}_2^j$ can be interpreted as the composition of i message passing steps along the graph $\mathcal{G}_1$ and j steps along the graph $\mathcal{G}_2$, regardless of the order in which these steps are taken.

2.3 COMPUTATIONAL ASPECTS

The structure of cartesian products can be leveraged to reduce the number of message passing operations in graph networks. Consider an edge $e \in \mathbb{E}$: it belongs to a layer along a factor F_i. During a message passing step, when a message is passed through e, we know that a similar operation will be performed in all the other layers corresponding to the same factor, so that these computations can be done in parallel. As a result, we can reshape the node attributes s as a matrix of size $\mathbb{V}_1 \times \mathbb{V}_2$. When passing messages along the edges of $\mathcal{G}_1$, we view s as a signal of dimension $|\mathbb{V}_2|$ on the graph $\mathbb{F}_1$, and when operating along $\mathcal{G}_2$, we view it as a signal of dimension $|\mathbb{V}_1|$ on the graph $\mathbb{F}_2$. This way, the number of message passing operations is reduced from $O(|\mathbb{E}|)$ to $O(|\mathbb{E}_1| + |\mathbb{E}_2|)$, and the computations can be more easily parallelized on GPUs.

Similarly, when using Laplacian polynomials, the input benefits from being reshaped as a matrix. When sparse semantics are not used, convolutions computed this way have a cost of $O(n_1^2 n_2 + n_1 n_2^2)$ instead of $O(n_1^2 \, n_2^2)$ for the naive formulation, as was observed by Sandryhaila & Moura (2014) for isotropic filters. In this case, the convolution operation can be written using only matrix products:

$$f(s) = \sum_{0 \leq i+j \leq k} c_{ij} \, \boldsymbol{L}_1^i \, s \, \boldsymbol{L}_2^j$$

When written this way, the filters are actually the same that were used by Monti et al. (2017) for recommender systems. Their proposition of learning "multi-graph" filters on two similarity graphs

(one for the movies and one for the users) actually amounts to performing semi-supervised learning on the cartesian product of these graphs. In this work, we exhibit this connection to existing methods from signal processing for cartesian products, extend it to more general graph networks and relate the experimental results to the representation power of the different filters.

3 EXPERIMENTAL RESULTS

We compare the performance of Laplacian polynomials, our method and standard CNNs on MNIST and CIFAR10 datasets [1]. A simple architecture was kept fixed so that only the convolution operation differs across settings. CNNs and Laplacian polynomials are compared to anisotropic filters by keeping constant either the kernel size or the number of parameters. Details on the architecture used can be found in Appendix A. An advantage of Laplacian polynomials is that they can easily be visualized: some filters of the first layer are displayed in Appendix 3. Although the filters learned are not isotropic, they still have two symmetry axes, which is a downside of using undirected graphs. In the limit of infinitely large receptive fields, anisotropic filters can learn 4 times less parameters than CNNs of the same size. Overall, the respective representation power of the different filters for a given size of receptive fields accurately explains the gap in performance in Table 2.

Table 2: Average final accuracy and 95% confidence interval on 10 runs.

Method	k-hop neigh.	kernel size	parameters	CIFAR10	MNIST
Isotropic	2	-	3	62.30 ± 0.34	98.65 ± 0.05
	8	-	9	62.03 ± 0.17	98.52 ± 0.08
Anisotropic	2	-	9	70.00 ± 0.40	99.22 ± 0.05
Image convolution	-	3	9	74.25 ± 0.37	99.33 ± 0.06
	-	5	25	74.26 ± 0.41	99.33 ± 0.06

To demonstrate the benefits of anisotropic filters for more complex graphs, we also compare products of Laplacian polynomials to their isotropic counterpart on Movielens100k (Harper & Konstan, 2016). The matrix completion problem is formulated as semi-supervised learning on the product of a user similarity graph and an item similarity graph. A low-rank structure is imposed by learning low-dimensional filters on each similarity graph. The setting and code of Monti et al. (2017) are used. Table 3 shows that performance deteriorates significantly when isotropic filters are used.

Table 3: Average root mean square error and 95% confidence interval on 10 runs on Movielens 100k.

Method	test RMSE
Isotropic	0.9393 ± 0.0016
Anisotropic	0.9345 ± 0.0022

CONCLUSION

Graph network layers learn functions that are invariant to permutation of the neighbors, which is a strong limit to their expressivity. Fortunately, when dealing with cartesian product graphs, the factors can be used to learn a richer class of filters and speed up the computations. The gain in performance observed experimentally calls for the development of anisotropic filters for less structured graphs as well, such as graphs sampled from manifolds.

[1] Code for the experiments is available at `https://github.com/cvignac/anisotropic_filters_cartesian`

REFERENCES

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

Wilfried Imrich and Iztok Peterin. Recognizing cartesian products in linear time. *Discrete Mathematics*, 307(3-5):472–483, 2007.

Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2017.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 3697–3707, 2017.

Nathanaël Perraudin, Andreas Loukas, Francesco Grassi, and Pierre Vandergheynst. Towards stationary time-vertex signal processing. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3914–3918. Ieee, 2017.

Aliaksei Sandryhaila and Jose MF Moura. Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure. *IEEE Signal Processing Magazine*, 31(5):80–90, 2014.

Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

Appendices

A NEURAL NETWORK ARCHITECTURE

A.1 FOR IMAGE CLASSIFICATION

The network architecture was chosen from a sample code online and never tuned. It consists of 3 convolutional layers followed by 2 fully connected layers (Figure 2). A max pooling of size 2 and stride 2 is intertwined between the convolutional layers. Note that we kept the standard pooling operation for all methods, without performing graph pooling: the different settings only differ by the

convolution operation. Zero padding is used in standard CNNs to keep the graph size constant. The architecture is the same for CIFAR10 and MNIST, except for the difference in input size, which has a knock-on effect on the number of features in each layer. Following a common practice, we used the symmetric normalized Laplacian, and shifted its eigenvalues to $[-1, 1]$ to avoid instabilities:

$$\tilde{L} = L_{sym} - I_n = -D^{-1/2} A D^{-1/2}$$

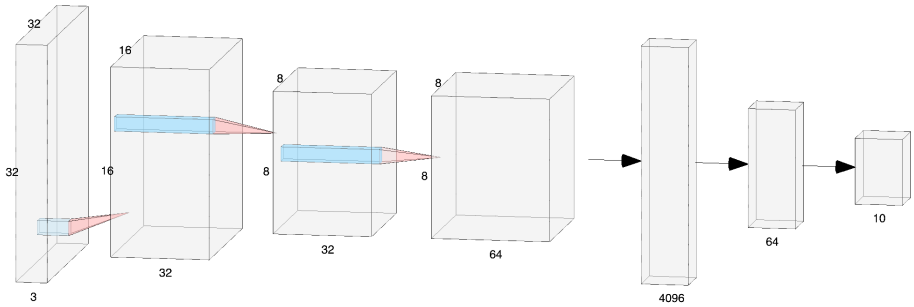

Figure 2: Neural network architecture for CIFAR10. The architecture on MNIST is the same except for the input size.

B FILTERS VISUALIZATION

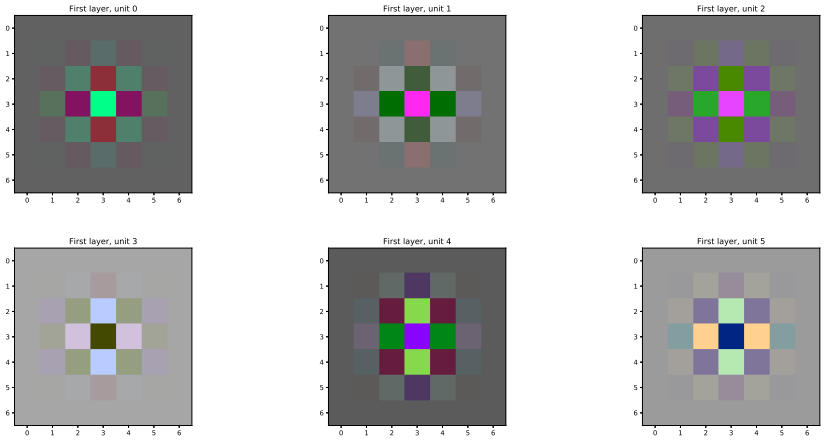

Figure 3: 6 of the 32 filters learned by the first layer on CIFAR10. The colors correspond to the different channels of the RGB image. Most behave very differently along the horizontal and vertical axes. However, they all have two symmetry axes, which comes from the fact that they were built on undirected graphs.