Build version as of 2018-08-08 12:49:57
Revision number 233

Contents

Introduction

TODO:

Chapter 1
Presentation of the field

Introduction

In this chapter, we present notions related to our domains of interest. In particular, for tensors we give original definitions that are more appropriate for our study. In the neural network's section, we present the concepts necessary to understand the evolution of the state of the art research in this field. In the last section, we present graphs for their usage in deep learning.
Vector spaces considered in what follows are assumed to be finite-dimensional and over the field of real numbers $\mathbb{R}$.

Contents

1.1 Tensors

Intuitively, tensors in the field of deep learning are defined as a generalization of vectors and matrices, as if vectors were tensors of rank 1 and matrices were tensors of rank 2. That is, they are objects in a vector space and their dimensions are indexed using as many indices as their rank, so that they can be represented by multidimensional arrays. In mathematics, a tensor can be defined as a special type of multilinear function (Bass, 1968; Marcus, 1975; Williamson, 2015), which image of a basis can be represented by a multidimensional array. Alternatively, Hackbush propose a mathematical construction of a tensor space as a quotient set of the span of an appropriately defined tensor product (Hackbusch, 2012), which coordinates in a basis can also be represented by a multidimensional array. In particular in the field of mathematics, tensors enjoy an intrinsic definition that neither depend on a representation nor would change the underlying object after a change of basis, whereas in our domain, tensors are confounded with their representation.

1.1.1 Definition

Our definition of tensors is such that they are a bit more than multidimensional arrays but not as much as mathematical tensors, for that they are embedded in a vector space so that deep learning objects can be later defined rigorously.
Given canonical bases, we first define a tensor space, then we relate it to the definition of the tensor product of vector spaces.

Definition 1. Tensor space
We define a *tensor space* $\mathbb{T}$ of rank r as a vector space such that its canonical basis is a cartesian product of the canonical bases of r other vector spaces. Its shape is denoted $n_1 \times n_2 \times \cdots \times n_r$, where the $\{n_k\}$ are the dimensions of the vector spaces.

Definition 2. Tensor product of vector spaces
Given r vector spaces $\mathbb{V}_1, \mathbb{V}_2, \ldots, \mathbb{V}_r$, their *tensor product* is the tensor space $\mathbb{T}$ spanned by the cartesian product of their canonical bases under coordinate-wise sum and outer product.
We use the notation $\mathbb{T} = \bigotimes_{k=1}^{r} \mathbb{V}_k$.

Remark. This simpler definition is indeed equivalent with the definition of the tensor product given in (Hackbusch, 2012, p. 51). The drawback of our definition is that it depends on the canonical bases, which at first can seem limiting as being canon implies that they are bounded to a certain system of coordinates. However this is not a concern in our domain as we need not distinguish tensors from their representation.

Naming convention
For naming convenience, from now on, we will distinguish between the terms *linear space* and *vector space i.e.* we will abusively use the term *vector space* only to refer to a linear space that is seen as a tensor space of rank 1. If we don't know its rank, we rather use the term *linear space*. We also make a clear distinction between the terms *dimension* (that is, for a tensor space it is equal to $\prod_{k=1}^{r} n_k$) and the term *rank* (equal to r). Note that some authors use the term *order* instead of *rank* (*e.g.* Hackbusch, 2012) as the latter is affected to another notion.

Definition 3. Tensor
A *tensor* t is an object of a tensor space. The *shape* of t, which is the same as the shape of the tensor space it belongs to, is denoted $n_1^{(t)} \times n_2^{(t)} \times \cdots \times n_r^{(t)}$.

1.1.2 Manipulation

In this subsection, we describe notations and operators used to manipulate data stored in tensors.

Definition 4. Indexing
An *entry* of a tensor $t \in \mathbb{T}$ is one of its scalar coordinates in the canonical basis, denoted $t[i_1, i_2, \ldots, i_r]$.
More precisely, if $\mathbb{T} = \bigotimes_{k=1}^{r} \mathbb{V}_k$, with bases $((e_k^i)_{i=1,\ldots,n_k})_{k=1,\ldots,r}$, then we have

$$t = \sum_{i_1=1}^{n_1} \cdots \sum_{i_r=1}^{n_r} t[i_1, i_2, \ldots, i_r](e_1^{i_1}, \ldots, e_r^{i_r})$$

The cartesian product $\mathbb{I} = \prod_{k=1}^{r} [\![1, n_k]\!]$ is called the *index space* of $\mathbb{T}$

Remark. When using an index i_k for an entry of a tensor t, we implicitly assume that $i_k \in [\![1, n_k^{(t)}]\!]$ if nothing is specified.

Definition 5. Subtensor
A *subtensor* t' is a tensor of same rank composed of entries of t that are contiguous in the indexing, with at least one entry per rank. We denote $t' = t[l_1{:}u_1, l_2{:}u_2, \ldots, l_r{:}u_r]$, where the $\{l_k\}$ and the $\{u_k\}$ are the lower and upper bounds of the indices used by the entries that compose t'.

Remark. We don't necessarily write the lower bound index if it is equal to 1, neither the upper bound index if it is equal to $n_k^{(t)}$.

Definition 6. Slicing

A *slice* operation, along the last ranks $\{r_1, r_2, \ldots, r_s\}$, and indexed by $(i_{r_1}, i_{r_2}, \ldots, i_{r_s})$, is a morphism $s : \mathbb{T} = \bigotimes_{k=1}^{r} \mathbb{V}_k \to \bigotimes_{k=1}^{r-s} \mathbb{V}_k$, such that:

$$s(t)[i'_1, i'_2, \ldots, i'_{r-s}] = t[i'_1, i'_2, \ldots, i'_{r-s}, i_{r_1}, i_{r_2}, \ldots, i_{r_s}]$$
$$i.e. \quad s(t) := t[:, :, \ldots, :, i_{r_1}, i_{r_2}, \ldots, i_{r_s}]$$

where $:=$ means that entries of the right operand are assigned to the left operand. We denote $t_{i_{r_1}, i_{r_2}, \ldots i_{r_s}}$ and call it the *slice* of t. Slicing along a subset of ranks that are not the lasts is defined similarly. $s(\mathbb{T})$ is called a *slice subspace*.

Definition 7. Flattening

A *flatten* operation is an isomorphism $f : \mathbb{T} \to \mathbb{V}$, between a tensor space $\mathbb{T}$ of rank r and an n-dimensional vector space $\mathbb{V}$, where $n = \prod_{k=1}^{r} n_k$. It is characterized by a bijection in the index spaces $g : \prod_{k=1}^{r} [\![1, n_k]\!] \to [\![1, n]\!]$ such that

$$\forall t \in \mathbb{T}, f(t)[g(i_1, i_2, \ldots, i_r)] = f(t[i_1, i_2, \ldots, i_r])$$

We call an inverse operation a *de-flatten* operation.

Row major ordering

The choice of g determines in which order the indexing is made. g is reminiscent of how data of multidimensional arrays or tensors are stored internally by programming languages. In most tensor manipulation languages, incrementing the memory address (*i.e.* the output of g) will first increment the last index i_r if $i_r < n_r$ (and if else $i_r = n_r$, then $i_r := 1$ and ranks are ordered in reverse lexicographic order to decide what indices are incremented). This

is called *row major ordering*, as opposed to *column major ordering*. That is, in row major, g is defined as

$$g(i_1, i_2, \ldots, i_r) = \sum_{p=1}^{r} \left(\prod_{k=p+1}^{r} n_k \right) i_p \tag{1}$$

Definition 8. Reshaping

A *reshape* operation is an isomorphism defined on a tensor space $\mathbb{T} = \bigotimes_{k=1}^{r} \mathbb{V}_k$ such that some of its basis vector spaces $\{\mathbb{V}_k\}$ are de-flattened and some of its slice subspaces are flattened.

1.1.3 Binary operations

We define binary operations on tensors that we'll later have use for. In particular, we define *tensor contraction* which is sometimes called *tensor multiplication*, *tensor product* or *tensor dotproduct* by other sources. We also define *convolution* and *pooling* which serve as the common building blocks of convolution neural network architectures (see Section **??**).

Definition 9. Contraction

A *tensor contraction* between two tensors, along ranks of same dimensions, is defined by natural extension of the dot product operation to tensors.
More precisely, let $\mathbb{T}_1$ a tensor space of shape $n_1^{(1)} \times n_2^{(1)} \times \cdots \times n_{r_1}^{(1)}$, and $\mathbb{T}_2$ a tensor space of shape $n_1^{(2)} \times n_2^{(2)} \times \cdots \times n_{r_2}^{(2)}$, such that $\forall k \in [\![1, s]\!], n_{r_1-(s-k)}^{(1)} = n_k^{(2)}$, then the tensor contraction between $t_1 \in \mathbb{T}_1$ and $t_2 \in \mathbb{T}_2$ is defined as:

$$\begin{cases} t_1 \otimes t_2 = t_3 \in \mathbb{T}_3 \text{ of shape } n_1^{(1)} \times \cdots \times n_{r_1-s}^{(1)} \times n_{s+1}^{(2)} \times \cdots \times n_{r_2}^{(2)} \text{ where} \\ t_3[i_1^{(1)}, \ldots, i_{r_1-s}^{(1)}, i_{s+1}^{(2)}, \ldots, i_{r_2}^{(2)}] = \\ \displaystyle\sum_{k_1=1}^{n_1^{(2)}} \cdots \sum_{k_s=1}^{n_s^{(2)}} t_1[i_1^{(1)}, \ldots, i_{r_1-s}^{(1)}, k_1, \ldots, k_s]\, t_2[k_1, \ldots, k_s, i_{s+1}^{(2)}, \ldots, i_{r_2}^{(2)}] \end{cases}$$

For the sake of simplicity, we omit the case where the contracted ranks are not the last ones for t_1 and the first ones for t_2. But this definition still holds in the general case subject to a permutation of the indices.

Definition 10. Covariant and contravariant indices
Given a tensor contraction $t_1 \otimes t_2$, indices of the left hand operand t_1 that are not contracted are called *covariant* indices. Those that are contracted are called *contravariant* indices. For the right operand t_2, the naming convention is the opposite. The set of covariant and contravariant indices of both operands are called the *transformation laws* of the tensor contraction.

Remark. Contrary to most mathematical definitions, tensors in deep learning are independent of any transformation law, so that they must be specified for tensor contractions.

Einstein summation convention
The Einstein summation convention is a notational convention to write a sum-product expression as a product expression. The summation indices are those that appear simultaneously in the superscript of the left operand and in the subscript of the right one, if subscripts precede superscripts in the notation, or else vice-versa. For example, a dot product is written $u_k v^k = \lambda$ and a matrix product is written $A_i{}^k B_k{}^j = C_i{}^j$.
The tensor contraction of Definition 9 can be rewritten using this convention:

$$t_{1_{i_1^{(1)} \cdots i_{r_1-s}^{(1)}}}{}^{k_1 \cdots k_s} t_{2_{k_1 \cdots k_s}}{}^{i_{s+1}^{(2)} \cdots i_{r_2}^{(2)}} = t_{3_{i_1^{(1)} \cdots i_{r_1-s}^{(1)}}}{}^{i_{s+1}^{(2)} \cdots i_{r_2}^{(2)}} \tag{2}$$

Proposition 11. A contraction can be rewritten as a matrix product.

Proof. Using notation of (2), with the reshapings $t_1 \mapsto T_1$, $t_2 \mapsto T_2$ and $t_3 \mapsto T_3$ defined by grouping all covariant indices into a single index and all

contravariant indices into another single index, we can rewrite

$$T_{1_{g_i(i_1^{(1)},\dots,i_{r_1-s}^{(1)})}}{}^{g_k(k_1,\dots,k_s)} T_{2_{g_k(k_1,\dots,k_s)}}{}^{g_j(i_{s+1}^{(2)},\dots,i_{r_2}^{(2)})} = T_{3_{g_i(i_1^{(1)},\dots,i_{r_1-s}^{(1)})}}{}^{g_j(i_{s+1}^{(2)},\dots,i_{r_2}^{(2)})}$$

where g_i, g_k and g_j are bijections defined similarly as in (1). □

Definition 12. Convolution

The *n-dimensional convolution*, denoted $*^n$, between $t_1 \in \mathbb{T}_1$ and $t_2 \in \mathbb{T}_2$, where $\mathbb{T}_1$ and $\mathbb{T}_2$ are of the same rank n such that $\forall p \in [\![1, n]\!], n_p^{(1)} \geq n_p^{(2)}$, is defined as:

$$\begin{cases} t_1 *^n t_2 = t_3 \in \mathbb{T}_3 \text{ of shape } n_1^{(3)} \times \cdots \times n_n^{(3)} \text{ where} \\ \forall p \in [\![1, n]\!], n_p^{(3)} = n_p^{(1)} - n_p^{(2)} + 1 \\ t_3[i_1, \dots, i_n] = \displaystyle\sum_{k_1=1}^{n_1^{(2)}} \cdots \sum_{k_n=1}^{n_n^{(2)}} t_1[i_1 + n_1^{(2)} - k_1, \dots, i_n + n_n^{(2)} - k_n]\, t_2[k_1, \dots, k_n] \end{cases}$$

Proposition 13. A convolution can be rewritten as a matrix product.

Proof. Let $t_1 *^n t_2 = t_3$ defined as previously with $\mathbb{T}_1 = \bigotimes_{k=1}^{r} \mathbb{V}_k^{(1)}$, $\mathbb{T}_2 = \bigotimes_{k=1}^{r} \mathbb{V}_k^{(2)}$. Let $t_1' \in \bigotimes_{k=1}^{r} \mathbb{V}_k^{(1)} \otimes \bigotimes_{k=1}^{r} \mathbb{V}_k^{(2)}$ such that $t_1'[i_1, \dots, i_n, k_1, \dots, k_n] = t_1[i_1 + n_1^{(2)} - k_1, \dots, i_n + n_n^{(2)} - k_n]$, then

$$t_3[i_1, \dots, i_n] = \sum_{k_1=1}^{n_1^{(2)}} \cdots \sum_{k_n=1}^{n_n^{(2)}} t_1'[i_1, \dots, i_n, k_1, \dots, k_n]\, t_2[k_1, \dots, k_n]$$

where we recognize a tensor contraction. Proposition 11 concludes. □

The two following operations are meant to further decrease the shape of the resulting output.

Definition 14. Strided convolution
The n-dimensional *strided* convolution, with strides $s = (s_1, s_2, \ldots, s_n)$, denoted $*_s^n$, between $t_1 \in \mathbb{T}_1$ and $t_2 \in \mathbb{T}_2$, where $\mathbb{T}_1$ and $\mathbb{T}_2$ are of the same rank n such that $\forall p \in [\![1, n]\!], n_p^{(1)} \geq n_p^{(2)}$, is defined as:

$$\begin{cases} t_1 *_s^n t_2 = t_4 \in \mathbb{T}_4 \text{ of shape } n_1^{(4)} \times \cdots \times n_n^{(4)} \text{ where} \\ \forall p \in [\![1, n]\!], n_p^{(4)} = \lfloor \frac{n_p^{(1)} - n_p^{(2)} + 1}{s_p} \rfloor \\ t_4[i_1, \ldots, i_n] = (t_1 *^n t_2)[(i_1 - 1)s_n + 1, \ldots, (i_n - 1)s_n + 1] \end{cases}$$

Remark. Informally, a strided convolution is defined as if it were a regular subsampling of a convolution. They match if $s = (1, 1, \ldots, 1)$.

Definition 15. Pooling
Let a real-valued function f defined on all tensor spaces of any shape, *e.g.* the *max* or *average* function. An f-pooling operation is a mapping $t \mapsto t'$ such that each entry of t' is an image by f of a subtensor of t.

Remark. Usually, the set of subtensors that are reduced by f into entries of t' are defined by a regular partition of the entries of t.

1.2 Neural Networks

1.2.1 Formalizations

A feed-forward neural network could originally be formalized as a composite function chaining linear and non-linear functions (Rumelhart et al., 1985; LeCun et al., 1989; LeCun, Bengio, et al., 1995). That whas still the case in 2012 when important breakthroughs regenerated a surge of interest in the field (Hinton et al., 2012; Krizhevsky et al., 2012; Simonyan and Zisserman, 2014). However, in more recent years, more complex architectures have emerged (Szegedy et al., 2015; He et al., 2016a; Zoph and Le, 2016; Huang et al., 2017), such that the former formalization does not suffice. We provide a definition for the first kind of neural networks (Definition 16) and use it to present its related concepts. Then we give a more generic definition (Definition 20).
Note that in this manuscript, we only consider neural networks that are *feed-forward* (Zell, 1994; Wikipedia, 2018a).
We denote by I_f the *domain of definition* of a function f ("I" stands for "input") and by $O_f = f(I_f)$ its *image* ("O" stands for "output"), and we represent it as $I_f \xrightarrow{f} O_f$ or $f : I_f \to O_f$.

Definition 16. Neural network (simply connected)
Let f be a function such that I_f and O_f are vector or tensor spaces.
f is a *(simply connected) neural network function* if there are a series of affine functions $(g_k)_{k=1,2,..,L}$ and a series of non-linear derivable univariate functions $(h_k)_{k=1,2,..,L}$ such that:

$$\begin{cases} \forall k \in [\![1, L]\!], f_k = h_k \circ g_k, \\ I_f = I_{f_1} \xrightarrow{f_1} O_{f_1} \cong I_{f_2} \xrightarrow{f_2} \dots \xrightarrow{f_L} O_{f_L} = O_f, \\ f = f_L \circ ... \circ f_2 \circ f_1 \end{cases}$$

The couple (g_k, h_k) is called the *k-th layer* of the neural network. L is its

depth. For $x \in I_f$, we denote by $x_k = f_k \circ ... \circ f_2 \circ f_1(x)$ the *activations* of the k-th layer. We denote by $\mathcal{N}$ the set of neural network functions.

Definition 17. Activation function
An *activation function* h is a real-valued univariate function that is non-linear and derivable, that is also defined by extension with the functional notation $h(v)[i] = h(v[i])$.

Definition 18. Layer
A layer is a couple $\mathcal{L} = (g, h) : I \to O$, where $g : I \to O$ is a linear function, and $h : O \to O$ is an activation function. It computes the function

$$y = h(g(x) + b)$$

where b is a constant called *bias*.

That is, in the simple formalization, a neural network is just a sequence of layers.

Remark. The bias augments the expressivity of the layers. For notational convenience, we may sometimes omit to write it down.

The most common activation function is the *rectified linear unit* (ReLU) (Glorot et al., 2011), used for its better practical performances and faster computation times. It implements the *rectifier* function $h : x \mapsto max(0, x)$ (with convention $h'(0) = 0$), as depicted on Figure 1.

Examples
Let $f : x \to y$ be a neural network. For example, if f is used to classify its input x in one of c classes, then its output y would be a vector of dimension c, and each dimension corresponds to a class. The prediction of f for the class of x is the dimension of y where it has the bigger value. Typically, f is terminated by a softmax activation (Wikipedia, 2018b), so that values of the

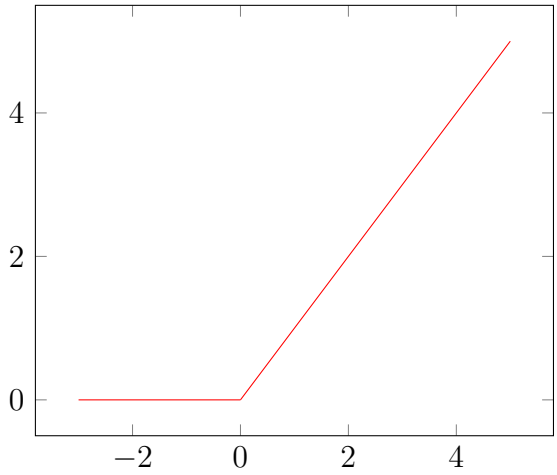


Figure 1: ReLU activation function

output y fall in the range $[0, 1]$, and so that y tends to have a dimension with a much bigger weight as to facilitates discrimination.

A neural network that comprises convolutional layers, *i.e.* layers *s.t.* g is expressed with a convolution, is called a Convolutional Neural Network (CNN). A common example is the LeNet-5 architecture (LeCun et al., 1989) as depicted in Figure 2. It implements a function

$$f = h_4 \circ g_4 \circ \cdots \circ h_1 \circ g_1$$

where g_1 and g_2 are linear functions that applies 5x5 convolutions followed by subsampling, h_1, h_2 and h_3 are ReLU activations, and h_4 is a softmax activation. It was originally applied to the task of handwritten digit classifications (for example for automatically reading postal ZIP codes).

Another example is the VGG architecture, a very deep CNN, and was state-of-the-art in image classification in 2014 (Simonyan and Zisserman). It is depicted on Figure 3

In more recent years, state-of-the-art architecture can no longer be described with a simple formalization.

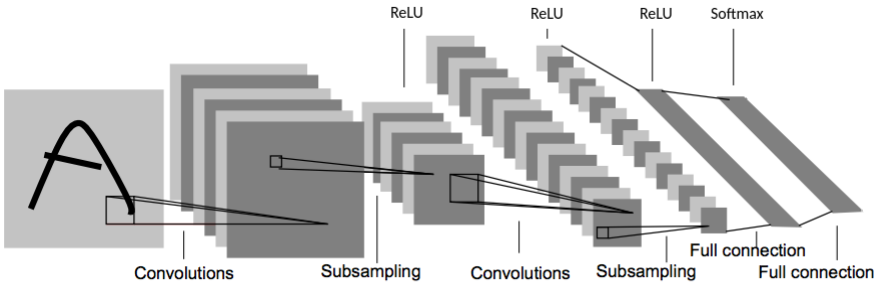


Figure 2: LeNet-5 (LeCun et al., 1989)

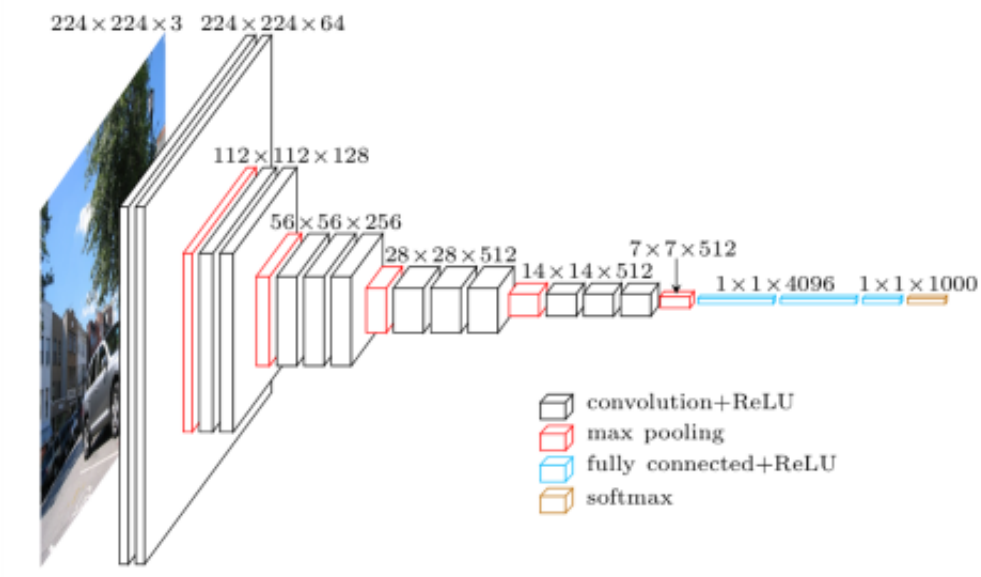


Figure 3: VGG-16 (Simonyan and Zisserman, 2014, figure from Cord, 2016)

The former neural networks are said to be *simply connected* because each layer only takes as input the output of the previous one. We'll give a more general definition after first defining branching operations.

Definition 19. Branching
A *binary branching operation* between two tensors, $x_{k_1} \bowtie x_{k_2}$, outputs, subject to shape compatibility, either their addition, either their concatenation along a rank, or their concatenation as a list.
A *branching operation* between n tensors, $x_{k_1} \bowtie x_{k_2} \bowtie \cdots \bowtie x_{k_n}$, is a composition of binary branching operations, or is the identity function Id if $n = 1$.
Branching operations are also naturally defined on tensor-valued functions.

Definition 20. Neural network (generic definition)
The set of *neural network* functions $\mathcal{N}$ is defined inductively as follows

1. $Id \in \mathcal{N}$
2. $f \in \mathcal{N} \wedge (g, h)$ is a layer $\wedge\, O_f \subset I_g \Rightarrow h \circ g \circ f \in \mathcal{N}$
3. for all shape compatible branching operations:
   $f_1, f_2, \ldots, f_n \in \mathcal{N} \Rightarrow f_1 \bowtie f_2 \bowtie \cdots \bowtie f_n \in \mathcal{N}$

Examples
The neural network proposed in (Szegedy et al., 2015), called *Inception*, use depth-wise concatenation of feature maps. Residual networks (ResNets, He et al., 2016a) make use of *residual connections*, also called *skip connections*, *i.e.* an activation that is used as input in a lower level is added to another activation at an upper level, as depicted on Figure 4. Densely connected networks (DenseNets, Huang et al., 2017) have their activations concatenated with all lower level activations. These neural networks had demonstrated state of the art performances on the imagenet classification challenge (Deng et al., 2009), outperforming simply connected neural networks. For example, DenseNet is depicted on Figure 5.

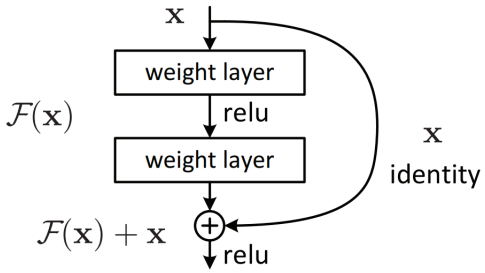


Figure 4: Module with a residual connection (He et al., 2016a)

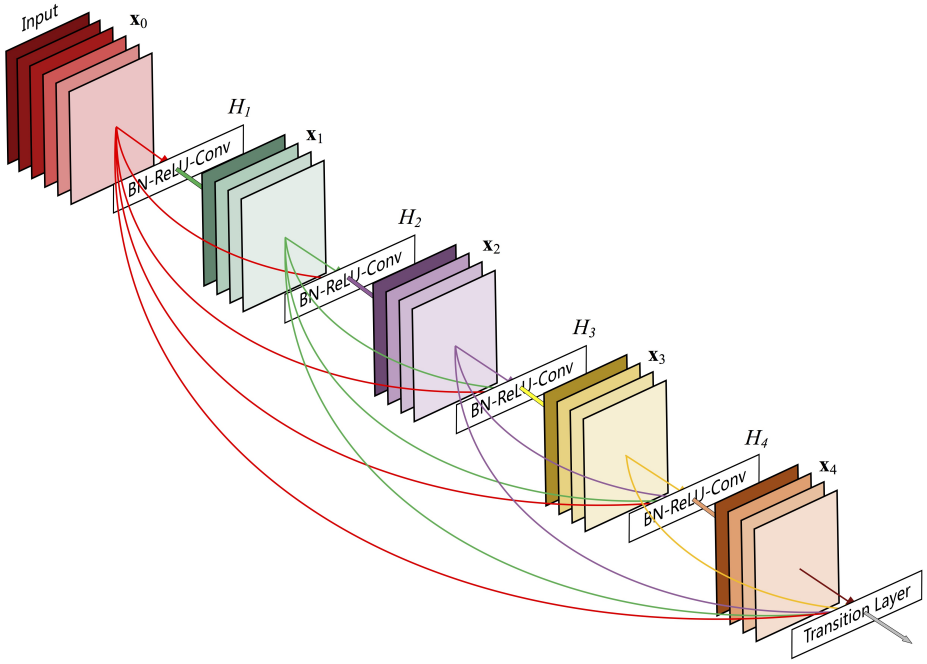


Figure 5: DenseNet (Huang et al., 2017)

Remark. For layer indexing convenience, we still use the simple formalization in the subsequent subsections, even though the presentation would be similar with the generic formalization.

1.2.2 Interpretation

Until now, we have formally introduced a neural network as a mathematical function. As its name suggests, such function can be indeed interpreted from a connectivity perspective (LeCun, 1987).

Definition 21. Connectivity matrix
Let g a linear function. Without loss of generality subject to a flattening, let's suppose I_g and O_g are vector spaces. Then there exists a *connectivity matrix* W_g, such that:

$$\forall x \in I_g, g(x) = W_g x$$

We denote W_k the connectivity matrix of the k-th layer.

Biological inspiration
A *neuron* is defined as a computational unit that is biologically inspired (McCulloch and Pitts, 1943). Each neuron is capable of:

1. receiving modulated signals from other neurons and aggregate them,
2. applying to the result an activation function,
3. passing the signal to other neurons.

That is to say, each domain $\{I_{f_k}\}$ and O_f can be interpreted as a layer of neurons, with one neuron for each dimension. The connectivity matrices $\{W_k\}$ describe the connections between each successive layers. A neuron is illustrated on Figure 6.

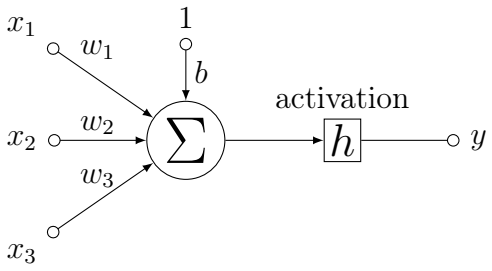


Figure 6: A neuron

1.2.3 Training

Given an objective function F, training is the process of incrementally modifying a neural network f upon obtaining a better approximation of F. The most used training algorithms are based on gradient descent, as proposed in (Widrow and Hoff, 1960). These algorithms became popular since (Rumelhart et al., 1985). Informally, f is parameterized with initial weights that characterize its linear parts. These weights are modified step by step. At each step, a batch of samples are fed to the network, and their approximation errors sum to a loss. The weights of the network are updated in the opposite direction to their gradient with respect to that loss. If the samples are shuffled and grouped in batches, this is called *Stochastic* gradient descent (SGD). Stochastic approximation (Robbins and Monro, 1985) tends to minimize effects of outliers on the training and is agnostic of the order in which the samples are fed.

Definition 22. Weights
Let consider the k-th layer of a neural network f. We define its weights as coordinates of a vector θ_k, called the *weight kernel*, such that:

$$\forall (i,j), \begin{cases} \exists p, W_k[i,j] := \theta_k[p] \\ \text{or } W_k[i,j] = 0 \end{cases}$$

A weight p that appears multiple times in W_k is said to be *shared*. Two

parameters of W_k that share a same weight p are said to be *tied*. The number of weights of the k-th layer is $n_1^{(\theta_k)}$.

Learning

A *loss* function $\mathcal{L}$ penalizes the output $x_L = f(x)$ relatively to the approximation error $|f(x) - F(x)|$. Gradient w.r.t. θ_k, denoted $\vec{\nabla}_{\theta_k}$, is used to update the weights via an optimization algorithm based on gradient descent and a learning rate α, that is:

$$\theta_k^{(\text{new})} = \theta_k^{(\text{old})} - \alpha \cdot \vec{\nabla}_{\theta_k} \left(\mathcal{L}\left(x_L, \theta_k^{(\text{old})}\right) + \mathcal{R}\left(\theta_k^{(\text{old})}\right) \right) \tag{3}$$

where $\mathcal{R}$ is a regularizer, and where α can be a scalar or a vector and $\cdot$ can denote outer or coodrinate-wise product, depending on the optimization algorithm that is used.

Linear complexity

Without loss of generality, we assume that the neural network is simply connected. Thanks to the chain rule, $\vec{\nabla}_{\theta_k}$ can be computed using gradients that are w.r.t. x_k, denoted $\vec{\nabla}_{x_k}$, which in turn can be computed using gradients w.r.t. outputs of the next layer $k+1$, up to the gradients given on the output layer.

That is:

$$\vec{\nabla}_{\theta_k} = J_{\theta_k}(x_k) \vec{\nabla}_{x_k} \tag{4}$$

$$\begin{aligned} \vec{\nabla}_{x_k} &= J_{x_k}(x_{k+1}) \vec{\nabla}_{x_{k+1}} \\ \vec{\nabla}_{x_{k+1}} &= J_{x_{k+1}}(x_{k+2}) \vec{\nabla}_{x_{k+2}} \\ &\cdots \\ \vec{\nabla}_{x_{L-1}} &= J_{x_{L-1}}(x_L) \vec{\nabla}_{x_L} \end{aligned} \tag{5}$$

Obtaining,

$$\vec{\nabla}_{\theta_k} = J_{\theta_k}(x_k)(\prod_{p=k}^{L-1} J_{x_p}(x_{p+1}))\vec{\nabla}_{x_L} \tag{6}$$

where $J_{\text{wrt}}(.)$ are the respective jacobians which can be determined with the layer's expressions and the $\{x_k\}$; and $\vec{\nabla}_{x_L}$ can be determined using $\mathcal{L}$, $\mathcal{R}$ and x_L. This allows to compute the gradients with a complexity that is linear with the number of weights (only one computation of the activations), instead of being quadratic if it were done with the difference quotient expression of the derivatives (one more computation of the activations for each weight).

Backpropagation

We can remark that (5) rewrites as

$$\begin{aligned} \vec{\nabla}_{x_k} &= J_{x_k}(x_{k+1})\vec{\nabla}_{x_{k+1}} \\ &= J_{x'_k}(h(x'_k))J_{x_k}(W_k x_k)\vec{\nabla}_{x_{k+1}} \end{aligned} \tag{7}$$

where $x'_k = W_k x_k$, and these jacobians can be expressed as:

$$J_{x'_k}(h(x'_k))[i,j] = \delta_i^j h'(x'_k[i]) \tag{8}$$

$$J_{x'_k}(h(x'_k)) = I\, h'(x'_k)$$

$$J_{x_k}(W_k x_k) = W_k^T \tag{9}$$

That means that we can write $\vec{\nabla}_{x_k} = (\widetilde{h}_k \circ \widetilde{g}_k)(\vec{\nabla}_{x_{k+1}})$ such that the connectivity matrix $\widetilde{W}_k$ is obtained by transposition. This can be interpreted as gradient calculation being a *back-propagation* on the same neural network, in opposition of the *forward-propagation* done to compute the output.

1.2.4 Historical advances

Universal approximation

Early researches have shown that neural networks with one level of depth can approximate any real-valued function defined on a compact subset of $\mathbb{R}^n$. This result was first proved for sigmoidal activations (Cybenko, 1989), and then it was shown it did not depend on the sigmoidal activations (Hornik et al., 1989; Hornik, 1991).

For example, this result brings theoretical justification that objective functions exists (even though it doesn't inform whether an algorithm to approach it exists or is efficient).

Computational difficulty

However, reaching such objective is a computationally difficult problem, which drove back interest from the field. Thanks to better hardware and to using better initialization schemes that speed up learning, researchers started to report more successes with deep neural networks (Hinton et al., 2006; Glorot and Bengio, 2010) ; see (Bengio, 2009) for a review of this period. It ultimately came to a surge of interest in the field after a significant breakthrough on the imagenet dataset (Deng et al., 2009) with deep CNNs (Krizhevsky et al., 2012). The use of the fast ReLU activation function (Glorot et al., 2011) as well as leveraging graphical processing units with CUDA (Nickolls et al., 2008) were also key factors in overcoming this computational difficulty.

Adoption of ReLU activations

Historically, sigmoidal and tanh activations were mostly used (Cybenko, 1989; LeCun et al., 1989). However in recent practice, the ReLU activation (first introduced as the *positive part*, Jarrett et al., 2009), become the most used activation, as it was demonstrated to be faster and to obtain better results (Glorot et al., 2011). ReLU originated numerous variants

e.g. leaky rectified linear unit (Maas et al., 2013), *parametric rectified linear unit* (PReLU, He et al., 2015), *exponential linear unit* (ELU, Clevert et al., 2015), *scaled exponential linear unit* (SELU, Klambauer et al., 2017), each one having particular advantages in some applications.

Adoption of dropout

Neural networks, like any other machine learning technique, may overfit. That is, a model may behave well on the training set but fails to generalize well on unseen examples. The introduction of dropout (Srivastava et al., 2014) have helped models with more parameters to be less prone to overfitting, as dropout consists in hiding some parts of the training samples and their intermediate activations.

Expressivity and expressive efficiency

The study of the *expressivity* (also called *representational power*) of families of neural networks is the field that is interested in the range of functions that can be realized or approximated by this family (Håstad and Goldmann, 1991; Pascanu et al., 2013). In general, given a maximal error ϵ and an objective F, the more expressive is a family $N \subset \mathcal{N}$, the more likely it is to contain an approximation $f \in N$ such that $d(f, F) < \epsilon$. However, if we consider the approximation $f_{min} \in N$ that have the lowest number of neurons, it is possible that f_{min} is still too large and may be unpractical. For this reason, expressivity is often studied along the related notion of *expressive efficiency* (Delalleau and Bengio, 2011; Cohen et al., 2018).

Rectifier neural netowrks

Of particular interest for the intuition is a result stating that a simply connected neural networks with only ReLU activations (a rectifier neural network) is a piecewise linear function (Pascanu et al., 2013; Montufar et al., 2014), and that conversely any piecewise linear function is also a rectifier

neural network such that an upper bound of its depth is logarithmically related to the input dimension (Arora et al., 2018, th. 2.1.). Their expressive efficiency have also been demonstrated compared to neural networks using threshold or sigmoid activations (Pan and Srikumar, 2016).

Benefits of depth

Expressive efficiency analysis have demonstrated the benefits of depth, *i.e.* a shallow neural network would need an unfeasible large number of neurons to approximate the function of a deep neural network (*e.g.* Delalleau and Bengio, 2011; Bianchini and Scarselli, 2014; Poggio et al., 2015; Eldan and Shamir, 2016; Poole et al., 2016; Raghu et al., 2016; Cohen and Shashua, 2016; Mhaskar et al., 2016; Lin et al., 2017; Arora et al., 2018). This field seeks to give theoretical grounds to the practical observation that state-of-the-art architectures are getting deeper.

Benefits of branching operations

Recent works have provided rationales supporting benefits of using branching operations, thus giving justifications for architectures obtained with the generic formalization. In particular, (Cohen et al., 2018) have analyzed the impact of residual connections used in Wavenet-like architectures (Van Den Oord et al., 2016) in terms of expressive efficiency, using tools from the field of tensor analysis ; (Orhan and Pitkow, 2018) have empirically demonstrated that skip connections can resolve some inefficiency problems inherent of fully-connected networks (dead activations, activations that are always equal, linearly dependent sets of activations).

1.2.5 Common layers

Definition 23. Connections
The set of *connections* of a layer (g, h), denoted C_g, is defined as:

$$C_g = \{(i, j), \exists p, W_g[i, j] := \theta_g[p]\}$$

We have $0 \leq |C_g| \leq n_1^{(W_g)} n_2^{(W_g)}$.

Definition 24. Dense layer
A *dense layer* (g, h) is a layer such that $|C_g| = n_1^{(W_g)} n_2^{(W_g)}$, *i.e.* all possible connections exist. The map $(i, j) \mapsto p$ is usually a bijection, meaning that there is no weight sharing.

A neural network made only of dense layers is called a Multi-Layer Perceptron (MLP, Hornik et al., 1989).

Definition 25. Partially connected layer
A *partially connected layer* (g, h) is a layer such that $|C_g| < n_1^{(W_g)} n_2^{(W_g)}$.
A *sparsely connected layer* (g, h) is a layer such that $|C_g| \ll n_1^{(W_g)} n_2^{(W_g)}$.

Definition 26. Convolutional layer
A *n-dimensional convolutional layer* (g, h) is such that the weight kernel θ_g can be reshaped into a tensor w of rank $n + 2$, and such that

$$\begin{cases} I_g \text{ and } O_g \text{ are tensor spaces of rank } n + 1 \\ \forall x \in I_g, g(x) = (g(x)_q = \sum_p x_p *^n w_{p,q})_{\forall q} \end{cases}$$

where p and q index slices along the last ranks.

A neural network that contains convolutional layers is called convolutional neural network (CNN).

Definition 27. Feature maps and input channels
The slices $g(x)_q$ are typically called *feature maps*, and the slices x_p are called *input channels*. Let's denote by $n_o = n_{n+1}^{(O_g)}$ and $n_i = n_{n+1}^{(I_g)}$ the number of feature maps and input channels. In other words, Definition 26 means that for each feature maps, a convolution layer computes n_i convolutions and sums them, computing a total if $n_i \times n_o$ convolutions.

Remark. Note that because they are simply summed, entries of two different input channels that have the same coordinates are assumed to share some sort of relationship. For instance on images, entries of each input channel (typically corresponding to Red, Green and Blue channels) that have the same coordinates share the same pixel location.

Benefits of convolutional layers
Comparatively with dense layers, convolution layers enjoy a significant decrease in the number of weights. For example, an input 2×2 convolution on images with 3-color input channels, would breed only 12 weights per feature maps, independently of the numbers of input neurons. On image datasets, their usage also breeds a significant boost in performance compared with dense layers (Krizhevsky et al., 2012), for they allow to take advantage of the topology of the inputs while dense layers don't (LeCun, Bengio, et al., 1995). A more thorough comparison and explanation of their assets will be discussed in Section 2.1.3.

Decrease of spatial dimensions
Given a tensor input x, the n-dimensional convolutions between the inputs channels x_p and slices of a weight tensor $w_{p,q}$ would result in outputs y_q of shape $n_1^{(x)} - n_1^{(w)} + 1 \times \ldots \times n_n^{(x)} - n_n^{(w)} + 1$. So, in order to preserve shapes, a padding operation must pad x with $n_1^{(w)} - 1 \times \ldots \times n_n^{(w)} - 1$ zeros beforehand. For example, the padding function of the library *tensorflow* (Abadi et al.,

2015) pads each rank with a balanced number of zeros on the left and right indices (except if $n_t^{(w)} - 1$ is odd then there is one more zero on the left).

Definition 28. Padding
A convolutional layer with *padding* (g, h) is such that g can be decomposed as $g = g_{\text{pad}} \circ g'$, where g' is the linear part of a convolution layer as in Definition 26, and g_{pad} is an operation that pads zeros to its inputs such that g preserves tensor shapes.

Remark. One asset of padding operations is that they limit the possible loss of information on the borders of the subsequent convolutions, as well as preventing a decrease in size. Moreover, preserving shape is needed to build some neural network architectures, especially for ones with branching operations *e.g.* examples in Section 1.2.1. On the other hand, they increase memory and computational footprints.

Definition 29. Stride
A convolutional layer with *stride* is a convolutional layer that computes strided convolutions (with stride > 1) instead of convolutions.

Definition 30. Pooling
A layer with *pooling* (g, h) is such that g can be decomposed as $g = g' \circ g_{\text{pool}}$, where g_{pool} is a pooling operation.

Layers with stride or pooling downscale the signals that passes through the layer. These types of layers allows to compute features at a coarser level, giving the intuition that the deeper a layer is in the network, the more abstract is the information captured by the weights of the layer.

A simple result
In two dimensions, convolutional operations can be rewritten as a matrix-vector multiplication where the matrix is Toeplitz. We show below that it is still the case in n dimensions.

Proposition 31. Connectivity matrix of a convolution with padding A convolutional layer with padding (g, h) is equivalently defined as its connectivity matrix W_g being a $n_i \times n_o$ block matrix such that its blocks are Toeplitz matrices, and where each block corresponds to a couple (p, q) of input channel p and feature map q.

Proof. Let's consider the slices indexed by p and q, and to simplify the notations, let's drop the subscripts $_{p,q}$. We recall from Definition 12 that

$$
\begin{aligned}
y &= (x *^n w)[j_1, \dots, j_n] \\
&= \sum_{k_1=1}^{n_1^{(w)}} \cdots \sum_{k_n=1}^{n_n^{(w)}} x[j_1 + n_1^{(w)} - k_1, \dots, j_n + n_n^{(w)} - k_n]\, w[k_1, \dots, k_n] \\
&= \sum_{i_1=j_1}^{j_1+n_1^{(w)}-1} \cdots \sum_{i_n=j_n}^{j_n+n_n^{(w)}-1} x[i_1, \dots, i_n]\, w[j_1 + n_1^{(w)} - i_1, \dots, j_n + n_n^{(w)} - i_n] \\
&= \sum_{i_1=1}^{n_1^{(x)}} \cdots \sum_{i_n=1}^{n_n^{(x)}} x[i_1, \dots, i_n]\, \widetilde{w}[i_1, j_1, \dots, i_n, j_n] \\
&\text{where } \widetilde{w}[i_1, j_1, \dots, i_n, j_n] = \\
&\qquad \begin{cases} w[j_1 + n_1^{(w)} - i_1, \dots, j_n + n_n^{(w)} - i_n] & \text{if } \forall t, 0 \leq i_t - j_t \leq n_t^{(w)} - 1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Using Einstein summation convention as in (2) and permuting indices, we recognize the following tensor contraction

$$y_{j_1 \cdots j_n} = x_{i_1 \cdots i_n} \widetilde{w}^{i_1 \cdots i_n}{}_{j_1 \cdots j_n} \tag{10}$$

Following Proposition 11, we reshape (10) as a matrix product. To reshape $y \mapsto Y$, we use the row major order bijections g_j as in (1) defined onto $\{(j_1, \dots, j_n), \forall t, 1 \leq j_t \leq n_t^{(y)}\}$. To reshape $x \mapsto X$, we use the same row major order bijection g_j, however defined on the indices that support non

zero-padded values, so that zero-padded values are lost after reshaping. That is, we use a bijection g_i such that $g_i(i_1, i_2, \ldots, i_n) = g_j(i_1 - o_1, i_2 - o_2, \ldots, i_n - o_n)$ defined if and only if $\forall t, 1 + o_t \leq i_t \leq n_t^{(y)}$, where the $\{o_t\}$ are the starting offsets of the non zero-padded values. $\widetilde{w} \mapsto W$ is reshaped by using g_j for its covariant indices, and g_i for its contravariant indices. The entries lost by using g_i do not matter because they would have been nullified by the resulting matrix product. We remark that W is exactly the block (p, q) of W_g (and not of $W_{g'}$). Now let's prove that it is a Toeplitz matrix.
Thanks to the linearity of the expression (1) of g_j, by denoting $i'_t = i_t - o_t$, we obtain

$$g_i(i_1, i_2, \ldots, i_n) - g_j(j_1, j_2, \ldots, j_n) = g_j(i'_1 - j_1, i'_2 - j_2, \ldots, i'_n - j_n) \quad (11)$$

To simplify the notations, let's drop the arguments of g_i and g_j. By bijectivity of g_j, (11) tells us that $g_i - g_j$ remains constant if and only if $i'_t - j_t$ remains constant for all t. Recall that

$$W[g_i, g_j] = \begin{cases} w[j_1 + n_1^{(w)} - i'_1, \ldots, j_n + n_n^{(w)} - i'_n] & \text{if } \forall t, 0 \leq i'_t - j_t \leq n_t^{(w)} - 1 \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

Hence, on a diagonal of W, $g_i - g_j$ remaining constant means that $W[g_i, g_j]$ also remains constants. So W is a Toeplitz matrix.
The converse is also true as we used invertible functions in the index spaces through the proof. □

Remark. Note that the proof doesn't hold in case there is no padding. This is due to border effects when the index of the n^{th} rank resets in the definition of the row-major ordering function g_j that would be used. Indeed, under appropriate definitions, the matrices could be seen as almost Toeplitz.

This proposition provides an equivalent-characterization of convolutional lay-

ers by their connectivity matrix. Therefore, a first avenue to define convolutions on graph signals could be to define them with the connectivity matrix being as in this characterization. However, the Toeplitz property implies that the dimensions have a specific order, which is not possible when dimensions correspond to vertices of a graph. This is because permuting the order of the vertices wouldn't change the graph, but would change the connectivity matrix (which cannot be Toeplitz for every ordering).

1.3 Graphs and signals

1.3.1 Basic definitions

1.3.1.1 Graphs

Definition 32. Graph
A *graph* G is defined as a couple of sets $\langle V, E \rangle$ where V is the set of *vertices*, also called *nodes*, and $E \subseteq \binom{V}{2}$ is the set of *edges*. For all $u, v \in V$ we define the relation $u \sim v \Leftrightarrow \{u, v\} \in E$. Unless stated otherwise, we will consider only *weighted* graphs *i.e.* each graph G is associated with a weight mapping $w : E \to \mathbb{R}^*$.

Figure 7 illustrates an example of a graph. Note that we employ interchangeably the terms *vertex* and *node*.

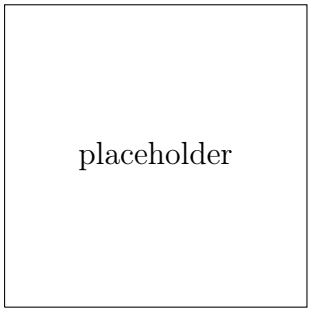


Figure 7: Example of a graph

Remark. According to this definition, we consider simple graphs *i.e.* no two edges share the same set of vertices and there is no self-loop.

Definition 33. Path
A *path* of length $n \in \mathbb{N}$ in a graph $G = \langle V, E \rangle$ is a sequence $(v_1 v_2 \cdots v_n)$ in V such that $\forall i, v_i \sim v_{i+1}$.

Remark. Our definition of graphs admit no self-loop so $\forall i, v_i \neq v_{i+1}$

Definition 34. Order
The order of a graph $G = \langle V, E \rangle$ is define as $\text{order}(G) = |V| \in \mathbb{N} \cup \{+\infty\}$

Definition 35. Adjacency matrix
The *adjacency matrix* of a finite graph $G = \langle V, E \rangle$ of order n, is a $n \times n$ real-valued matrix A associated to an indexing of $V = \{v_1, v_2, \dots v_n\}$, such that

$$A[i,j] = \begin{cases} w\left(\{v_i, v_j\}\right) & \text{if } v_i \sim v_j \\ 0 & \text{otherwise} \end{cases}$$

Definition 36. Degree
The *degree* of a vertex $v \in V$ of a graph $G = \langle V, E \rangle$ is defined as $\deg(v) = |\{u \in V, u \sim v\}| \in \mathbb{N} \cup \{+\infty\}$.
The *degree* of the graph G is defined as $\deg(G) = \max_{v \in V} \{\deg(v)\}$.
A graph is said to be *regular* if deg is constant on the vertices.

Definition 37. Degree matrix
The *degree matrix* of a finite graph $G = \langle V, E \rangle$ of order n, is the diagonal matrix D, associated to an indexing of $V = \{v_1, v_2, \dots v_n\}$, such that $D = \text{diag}(\deg(v_1), \deg(v_2), \dots, \deg(v_n))$.

Definition 38. Laplacian matrix
The *laplacian matrix* of a graph $G = \langle V, E \rangle$ of order n, associated to an indexing of $V = \{v_1, v_2, \dots v_n\}$, is defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix.

Definition 39. Digraph
A digraph is an oriented graph *i.e.* $E \subseteq V \times V - \{(v, v), v \in V\}$. Contrary to a graph, the weight mapping w, the relation $\sim$, the adjacency matrix A, and the laplacian matrix L are not symmetric. Notions defined on graphs naturally extends to digraphs where possible.

Definition 40. Bipartite graph
A *bipartite graph* is a triplet of sets $\langle V^{(1)}, V^{(2)}, E \rangle$, where $V^{(1)}$ and $V^{(2)}$ are sets of vertices, $V^{(1)} \cap V^{(2)} \neq \emptyset$, and $E \subseteq V^{(1)} \times V^{(2)}$. It is associated with a weight mapping $w : E \to \mathbb{R}^*$. Its adjacency matrix A is associated to indexings of $V^{(1)} = \{v_1^{(1)}, v_2^{(1)}, \dots v_n^{(1)}\}$ and $V^{(2)} = \{v_1^{(2)}, v_2^{(2)}, \dots v_n^{(2)}\}$, such that

$$A[i,j] = \begin{cases} w\left((v_i^{(1)}, v_j^{(2)})\right) & \text{if } (v_i^{(1)}, v_j^{(2)}) \in E \\ 0 & \text{otherwise} \end{cases}$$

Definition 41. Induced subgraph
The *subgraph* $\widetilde{G} = \langle \widetilde{V}, \widetilde{E} \rangle$ of a graph $G = \langle V, E \rangle$, *induced* by $\widetilde{V} \subseteq V$, is such that $\forall (u, v) \in \widetilde{V}^2, u \overset{\widetilde{G}}{\sim} v \Leftrightarrow u \overset{G}{\sim} v$.

TODO: subgraph (including direct or undirected)

Definition 42. Grid graph
A *grid graph* $G = \langle V, E \rangle$ is such that $V \cong \mathbb{Z}^2$, $v_1 \sim v_2 \Rightarrow \|v_2 - v_1\|_\infty \in \{0, 1\}$ and either one of the following is true:

$$\begin{cases} (i_1, j_1) \sim (i_2, j_2) \Leftrightarrow |i_2 - i_1| \operatorname{XOR} |j_2 - j_1| & \text{(4 neighbours)} \\ (i_1, j_1) \sim (i_2, j_2) \Leftrightarrow |i_2 - i_1| \operatorname{AND} |j_2 - j_1| & \text{(4 neighbours)} \\ (i_1, j_1) \sim (i_2, j_2) \Leftrightarrow |i_2 - i_1| \operatorname{OR} |j_2 - j_1| & \text{(8 neighbours)} \end{cases}$$

A *(rectangular) grid graph* of size $n \times m$ is the subgraph of a grid graph induced by $[\![1, n]\!] \times [\![1, m]\!]$.
A *square grid graph* is a rectangular grid graph of square size.

1.3.1.2 (Real-valued) Signals

Definition 43. Signal space
The *signal space* $\mathcal{S}(V)$, over the set V, is the linear space of real-valued functions defined on V.

We have $\dim(\mathcal{S}(V)) = |V| \in \mathbb{N} \cup \{+\infty\}$.

Remark. In particular, a vector space, and more generally a tensor space, are finite-dimensional signal spaces over any of their bases.

Definition 44. Signal
A *signal* over V, $s \in \mathcal{S}(V)$, is a function $s : V \to \mathbb{R}$.
An *entry* of a signal s is an image by s of some $v \in V$ and we denote $s[v]$.
If v is represented by a n-tuple, we can also write $s[v_1, v_2, \ldots, v_n]$.
The *support* of a signal $s \in \mathcal{S}(V)$ is $\operatorname{supp}(s) = \{v \in V, s[v] \neq 0\}$.

Definition 45. Graph signal
A *graph signal* over G is a signal over its vertex set. We denote by $\mathcal{S}(G)$ the graph signal space.

We have $\dim(\mathcal{S}(G)) = \text{order}(G) \in \mathbb{N} \cup \{+\infty\}$.

Definition 46. Underlying structure
An *(underlying) structure* of a signal s over a set V, is a graph G with vertex set V.

Examples
When there is a unique clear underlying structure, we say that it is *the* underlying structure. For example, images are compactly supported signals over $\mathbb{Z}^2$ and their underlying structure is a rectangular grid graph. Time series are signals over $\mathbb{N}$ and their underlying structure is a line graph. The underlying structure of a graph signal is obviously the graph itself.

1.3.2 Graphs in deep learning

TODO: below

We come across the notion of graphs several times in deep learning:

- Connections between two layers of a deep learning model can be represented as a bipartite graph, the *connectivity graph*. It encodes how the information is propagated through a layer to another. See Section 1.3.2.1.
- Neural architectures can be represented by a graph. In particular, a computation graph is used by deep learning programming languages to keep track of the dependencies between layers of a deep learning model, in order to compute forward and back-propagation. See Section 1.3.2.2.

- A graph can represent the underlying structure of an object (often a vector or a signal). The nodes represent its features, and the edges represent some structural property. See Section 1.3.2.3.
- Datasets can also be graph-structured. The nodes represent the objects of the dataset, and its edge represent some sort of relation between them. See Section 1.3.2.4.

1.3.2.1 Connectivity graph

A Connectivity graph is the bipartite graph whose adjacency matrix is the connectivity matrix of a layer of neurons. Formally, given a linear part of a layer, let $\mathbf{x}$ and $\mathbf{y}$ be the input and output signals, n the size of the set of input neurons $N = \{u_1, u_2, \dots, u_n\}$, and m the size of the set of output neurons $M = \{v_1, v_2, \dots, v_m\}$. This layer implements the equation $y = \Theta x$ where Θ is a $n \times m$ matrix.

Definition 47. The *connectivity graph* $G = (V, E)$ is defined such that $V = N \cup M$ and $E = \{(u_i, v_j) \in N \times M, \Theta_{ij} \neq 0\}$.

I.e. the connectivity graph is obtained by drawing an edge between neurons for which $\Theta_{ij} \neq 0$. For instance, in the special case of a complete bipartite graph, we would obtain a dense layer. Connectivity graphs are especially useful to represent partially connected layers, for which most of the Θ_{ij} are 0. For example, in the case of layers characterized by a small local receptive field, the connectivity graph would be sparse, and output neurons would be connected to a set of input neurons that corresponds to features that are close together in the input space. Figure 8 depicts some examples.
TODO: Figure 8. It's just a placeholder right now

Connectivity graphs also allow to graphically modelize how weights are tied in a neural layer. Let's suppose the $\Theta_i j$ are taking their values only into the finite set $K = \{w_1, w_2, \dots, w_\kappa\}$ of size κ, which we will refer to as the *kernel*

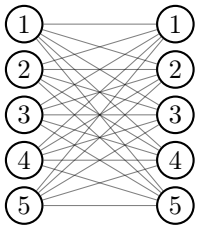


Figure 8: Examples

of *weights*. Then we can define a labelling of the edges $s : E \to K$. s is called the *weight sharing scheme* of the layer. This layer can then be formulated as $\forall v \in M, y_v = \sum_{u \in N, (u,v) \in E} w_{s(u,v)} x_u$. Figure 9 depicts the connectivity graph of a 1-d convolution layer and its weight sharing scheme.

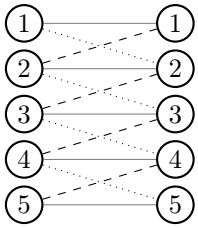


Figure 9: Depiction of a 1D-convolutional layer and its weight sharing scheme.

TODO: Add weight sharing scheme in Figure 9

1.3.2.2 Computation graph

1.3.2.3 Underlying graph structure and signals

1.3.2.4 Graph-structured dataset

Chapter 2
Convolutions on graph domains

Introduction

Defining a convolution of signals over graph domains is a challenging problem. If the graph is not a grid graph, there exists no natural extension of the euclidean convolution.

In Section 2.1, we analyze the reasons why the euclidean convolution operator is useful in deep learning, and give a characterization. Then we will search for domains onto which a convolution with these properties can be naturally obtained.

This will lead us to put our interest on representation theory and convolutions defined on groups in Section 2.2. As the euclidean convolution is just a particular case of the group convolution, it makes perfect sense to steer our construction in this direction. Hence, we will aim at transferring its representation on the vertex domain.

Then, in Section 2.3, we will introduce the role of the edge set and see how it should influence it. This will provide us with some particular classes of graphs for which we will obtain a natural construction with the wanted characteristics that we exposed in the first place.

Finally, we will relax some aspect of the construction to adapt it to general graphs in Section 2.4. The obtained construction is a set of general expressions that describes convolutions on graph domains and preserves some key properties.

We summarize our constructions in a conclusive Section 2.5.

Contents

2.1 Analysis of the classical convolution

In this section, we are exposing a few properties of the classical convolution that a generalization to graphs would likely try to preserve. For now let's consider a graph G agnostically of its edges *i.e.* $G \cong V$ is just the set of its vertices.

2.1.1 Properties of the convolution

Consider an edge-less grid graph *i.e.* $G \cong \mathbb{Z}^2$. By restriction to compactly supported signals, this case encompass the case of images.

Definition 48. Convolution on $\mathcal{S}(\mathbb{Z}^2)$
Recall that the (discrete) convolution between two signals s_1 and s_2 over $\mathbb{Z}^2$ is a binary operation in $\mathcal{S}(\mathbb{Z}^2)$ defined as:

$$\forall (a,b) \in \mathbb{Z}^2, (s_1 * s_2)[a,b] = \sum_i \sum_j s_1[i,j]\, s_2[a-i, b-j]$$

Definition 49. Convolution operator
A *convolution operator* is a function of the form $f_w : x \mapsto x * w$, where x and w are signals of domains for which the convolution $*$ is defined. When $*$ is not commutative, we differentiate the *right-action* operator $x \mapsto x * w$ from the *left-action* one $x \mapsto w * x$.

The following properties of the convolution on $\mathbb{Z}^2$ are of particular interest for our study.

Linearity
Operators produced by the convolution are linear. So they can be used as linear parts of layers of neural networks.

Locality and weight sharing

When w is compactly supported on K, an impulse response $f_w(x)[a, b]$ amounts to a w-weighted aggregation of entries of x in a neighbourhood of (a, b), called the *local receptive field*.

Commutativity

The convolution is commutative. However, it won't necessarily be the case on other domains.

Equivariance to translations

Convolution operators are equivariant to translations. Below, we show that the converse of this result also holds with Proposition 53.

2.1.2 Characterization on grid graphs

Let's recall first what is a transformation, and how it acts on signals.

Definition 50. Transformation

A *transformation* $f : V \to V$ is a function with same domain and codomain. The set of transformations is denoted $\Phi(V)$. The set of bijective transformations is denoted $\Phi^*(V) \subset \Phi(V)$.

In particular, $\Phi^*(V)$ forms the symmetric group of V and can move signals of $\mathcal{S}(V)$ by linear extension of its group action.

Lemma 51. Extension to $\mathcal{S}(V)$ by group action

A bijective transformation $f \in \Phi^*(V)$ can be extended linearly to the signal space $\mathcal{S}(V)$, and we have:

$$\forall s \in \mathcal{S}(V), \forall v \in V, f(s)[v] = s[f^{-1}(v)]$$

Proof. Let $s \in \mathcal{S}(V), f \in \Phi^*(V), L_f \in \mathcal{L}(\mathcal{S}(V))$ *s.t.* $\forall v \in V, L_f(\delta_v) = \delta_{f(v)}$. Then, we have:

$$\begin{aligned} L_f(s) &= \sum_{v \in V} s[v]\, L_f(\delta_v) \\ &= \sum_{v \in V} s[v]\, \delta_{f(v)} \\ \text{So, } \forall v \in V, L_f(s)[v] &= s[f^{-1}(v)] \end{aligned}$$

□

We also recall the formalism of translations.

Definition 52. Translation on $\mathcal{S}(\mathbb{Z}^2)$
A translation on $\mathbb{Z}^2$ is defined as a transformation $t \in \Phi^*(\mathbb{Z}^2)$ such that

$$\exists (a, b) \in \mathbb{Z}^2, \forall (x, y) \in \mathbb{Z}^2, t(x, y) = (x + a, y + b)$$

It also acts on $\mathcal{S}(\mathbb{Z}^2)$ with the notation $t_{a,b}$ *i.e.*

$$\forall s \in \mathcal{S}(\mathbb{Z}^2), \forall (x, y) \in \mathbb{Z}^2, t_{a,b}(s)[x, y] = s[x - a, y - b]$$

For any set E, we denote by $\mathcal{T}(E)$ its translations if they are defined.

The next proposition fully characterizes convolution operators with their translational equivariance property. This can be seen as a discretization of a classic result from the theory of distributions (Schwartz, 1957).

Proposition 53. Characterization of convolution operators on $\mathcal{S}(\mathbb{Z}^2)$
On real-valued signals over $\mathbb{Z}^2$, the class of linear transformations that are equivariant to translations is exactly the class of convolutive operations *i.e.*

$$\exists w \in \mathcal{S}(\mathbb{Z}^2), f = . * w \Leftrightarrow \begin{cases} f \in \mathcal{L}(\mathcal{S}(\mathbb{Z}^2)) \\ \forall t \in \mathcal{T}(\mathcal{S}(\mathbb{Z}^2)), f \circ t = t \circ f \end{cases}$$

Proof. The result from left to right is a direct consequence of the definitions:

$$\begin{aligned}
&\forall s \in \mathcal{S}(\mathbb{Z}^2), \forall s' \in \mathcal{S}(\mathbb{Z}^2), \forall (\alpha, \beta) \in \mathbb{R}^2, \forall (a, b) \in \mathbb{Z}^2, \\
&\qquad f_w(\alpha s + \beta s')[a, b] = \sum_i \sum_j (\alpha s + \beta s')[i, j]\, w[a - i, b - j] \\
&\qquad\qquad = \alpha f_w(s)[a, b] + \beta f_w(s')[a, b] \qquad \text{(linearity)} \\
&\forall s \in \mathcal{S}(\mathbb{Z}^2), \forall (\alpha, \beta) \in \mathbb{Z}^2, \forall (a, b) \in \mathbb{Z}^2, \\
&\qquad f_w \circ t_{\alpha,\beta}(s)[a, b] = \sum_i \sum_j t_{\alpha,\beta}(s)[i, j]\, w[a - i, b - j] \\
&\qquad\qquad = \sum_i \sum_j s[i - \alpha, j - \beta]\, w[a - i, b - j] \\
&\qquad\qquad = \sum_{i'} \sum_{j'} s[i', j']\, w[a - i' - \alpha, b - j' - \beta] \qquad (13) \\
&\qquad\qquad = f_w(s)[a - \alpha, b - \beta] \\
&\qquad\qquad = t_{\alpha,\beta} \circ f_w(s)[a, b] \qquad \text{(equivariance)}
\end{aligned}$$

Now let's prove the result from right to left.
Let $f \in \mathcal{L}(\mathcal{S}(\mathbb{Z}^2))$, $s \in \mathcal{S}(\mathbb{Z}^2)$. We suppose that f commutes with translations. Recall that s can be linearly decomposed on the infinite family of dirac signals:

$$s = \sum_i \sum_j s[i, j]\, \delta_{i,j}, \text{ where } \delta_{i,j}[x, y] = \begin{cases} 1 & \text{if } (x, y) = (i, j) \\ 0 & \text{otherwise} \end{cases}$$

By linearity of f and then equivariance to translations:

$$\begin{aligned}
f(s) &= \sum_i \sum_j s[i, j]\, f(\delta_{i,j}) \\
&= \sum_i \sum_j s[i, j]\, f \circ t_{i,j}(\delta_{0,0})
\end{aligned}$$

$$= \sum_i \sum_j s[i,j]\, t_{i,j} \circ f(\delta_{0,0})$$

By denoting $w = f(\delta_{0,0}) \in \mathcal{S}(\mathbb{Z}^2)$, we obtain:

$$\begin{aligned} \forall (a,b) \in \mathbb{Z}^2, f(s)[a,b] &= \sum_i \sum_j s[i,j]\, t_{i,j}(w)[a,b] \\ &= \sum_i \sum_j s[i,j]\, w[a-i, b-j] \\ \textit{i.e. } f(s) &= s * w \end{aligned} \tag{14}$$

□

2.1.3 Usefulness of convolutions in deep learning

Equivariance property of CNNs

In deep learning, an important argument in favor of CNNs is that convolutional layers are equivariant to translations. Intuitively, that means that a detail of an object in an image should produce the same features independently of its position in the image.

Lossless superiority of CNNs over MLPs

The converse result, as a consequence of Proposition 53, is never mentioned in deep learning literature. However it is also a strong one. For example, let's consider a linear function that is equivariant to translations. Thanks to the converse result, we know that this function is a convolution operator parameterized by a weight vector w, $f_w : . * w$. If the domain is compactly supported, as in the case of images, we can break down the information of w in a finite number n_q of kernels w_q with small compact supports of same size (for instance of size 2×2), such that we have $f_w = \sum_{q \in \{1,2,\dots,n_q\}} f_{w_q}$. The convolution operators f_{w_q} are all in the search space of 2×2 convolutional layers. In other words, every translational equivariant linear function can

have its information parameterized by these layers. So that means that the reduction of parameters from an MLP to a CNN is done with strictly no loss of expressivity (provided the objective function is known to bear this property). Besides, it also helps the training to search in a much more confined space.

Methodology for extending to general graphs

Hence, in our construction, we will try to preserve the characterization from Proposition 53 as it is mostly the reason why they are successful in deep learning. Note that the reduction of parameters compared to a dense layer is also a consequence of this characterization.

2.2 Construction from the vertex set

As Proposition 53 is a complete characterization of convolutions, it can be used to define them *i.e.* convolution operators can be constructed as the set of linear transformations that are equivariant to translations. However, in the general case where G is not a grid graph, translations are not defined, so that construction needs to be generalized beyond translational equivariances. In mathematics, convolutions are more generally defined for signals defined over a group structure. The classical convolution that is used in deep learning is just a narrow case where the domain group is an euclidean space. Therefore, constructing a convolution on graphs should start from the more general definition of convolution on groups rather than convolution on euclidean domains.
Our construction is motivated by the following questions:

- Does the equivariance property holds ? Does the characterization from Proposition 53 still holds ?
- Is it possible to extend the construction on non-group domains, or at least on mixed domains ? (*i.e.* one signal is defined over a set, and the other is defined over a subgroup of the transformations of this set).
- Can a group domain draw an underlying graph structure ? Is the group convolution naturally defined on this class of graphs ?

We first recall the notion of group and group convolution.

Definition 54. Group
A group Γ is a set equipped with a closed, associative and invertible composition law that admits a unique left-right identity element.

The group convolution extends the notion of the classical discrete convolution.

Definition 55. Group convolution I
Let a group Γ, the group convolution I between two signals s_1 and $s_2 \in \mathcal{S}(\Gamma)$ is defined as:

$$\forall h \in \Gamma, (s_1 *_{\text{I}} s_2)[h] = \sum_{g \in \Gamma} s_1[g]\, s_2[g^{-1}h]$$

provided at least one of the signals has finite support if Γ is not finite.

2.2.1 Steered construction from groups

For a graph $G = \langle V, E \rangle$ and a subgroup $\Gamma \subset \Phi^*(V)$ or its invertible transformations, Definition 55 is applicable for $\mathcal{S}(\Gamma)$, but not for $\mathcal{S}(V)$ as V is not a group. Nonetheless, our point here is that we will use the group convolution on $\mathcal{S}(\Gamma)$ to construct the convolutions on $\mathcal{S}(V)$.
For now, let's assume Γ is in one-to-one correspondence with V, and let's define a bijective map φ from Γ to V. We denote $\Gamma \overset{\varphi}{\cong} V$ and $g_v \overset{\varphi}{\mapsto} v$.
Then, the linear morphism $\widetilde{\varphi}$ from $\mathcal{S}(\Gamma)$ to $\mathcal{S}(V)$ defined on the Dirac bases by $\widetilde{\varphi}(\delta_g) = \delta_{\varphi(g)}$ is a linear isomorphism. Hence, $\mathcal{S}(V)$ would inherit the same inherent structural properties as $\mathcal{S}(\Gamma)$. For the sake of notational simplicity, we will use the same symbol φ for both φ and $\widetilde{\varphi}$ (as done between f and L_f). A commutative diagram between the sets is depicted on Figure 10.

$$\begin{array}{ccc} \Gamma & \xrightarrow{\varphi} & V \\ {\scriptstyle s}\downarrow & & \downarrow{\scriptstyle s} \\ \mathcal{S}(\Gamma) & \xrightarrow[\varphi]{} & \mathcal{S}(V) \end{array}$$

Figure 10: Commutative diagram between sets

We naturally obtain the following relation, which put in simpler words means that signals on $\mathcal{S}(\Gamma)$ are mapped to $\mathcal{S}(V)$ when φ is simultaneously applied on both the signal space and its domain.

Lemma 56. Relation between $\mathcal{S}(\Gamma)$ and $\mathcal{S}(V)$
$\forall s \in \mathcal{S}(\Gamma), \forall u \in V, \varphi(s)[u] = s[\varphi^{-1}(u)] = s[g_u]$

Proof.

$$\begin{aligned}\forall s \in \mathcal{S}(\Gamma), \varphi(s) &= \varphi(\sum_{g\in\Gamma} s[g]\,\delta_g) = \sum_{g\in\Gamma} s[g]\,\varphi(\delta_g) = \sum_{g\in\Gamma} s[g]\,\delta_{\varphi(g)} \\ &= \sum_{v\in V} s[g_v]\,\delta_v \\ \text{So } \forall v \in V, \varphi(s)[u] &= s[g_u]\end{aligned}$$

□

Hence, we can steer the definition of the group convolution from $\mathcal{S}(\Gamma)$ to $\mathcal{S}(V)$ as follows:

Definition 57. Group convolution II
Let a subgroup $\Gamma \subset \Phi^*(V)$ such that $\Gamma \overset{\varphi}{\cong} V$. The group convolution II between two signals s_1 and $s_2 \in \mathcal{S}(V)$ is defined as:

$$\forall u \in V, (s_1 *_{\text{II}} s_2)[u] = \sum_{v\in V} s_1[v]\, s_2[\varphi(g_v^{-1} g_u)]$$

Lemma 58. Relation between group convolution I and II
Let a subgroup $\Gamma \subset \Phi^*(V)$ such that $\Gamma \overset{\varphi}{\cong} V$,

$$\forall s_1, s_2 \in \mathcal{S}(\Gamma), \forall u \in V, (\varphi(s_1) *_{\text{II}} \varphi(s_2))[u] = (s_1 *_{\text{I}} s_2)[g_u]$$

Proof. Using Lemma 56,

$$\begin{aligned}(\varphi(s_1) *_{\text{II}} \varphi(s_2))[u] &= \sum_{v \in V} \varphi(s_1)[v]\, \varphi(s_2)[\varphi(g_v^{-1} g_u)] \\ &= \sum_{v \in V} s_1[g_v]\, s_2[g_v^{-1} g_u] \\ &= \sum_{g \in \Gamma} s_1[g]\, s_2[g^{-1} g_u] \\ &= (s_1 *_{\text{I}} s_2)[g_u]\end{aligned}$$

□

For convolution II, we only obtain a weak version of Proposition 53.

Proposition 59. Equivariance to $\varphi(\Gamma)$
If φ is a homomorphism, convolution operators acting on the right of $\mathcal{S}(V)$ are equivariant to $\varphi(\Gamma)$ *i.e.*

$$\begin{aligned}&\text{if } \varphi \in \text{ISO}(\Gamma, V), \\ &\exists w \in \mathcal{S}(V), f = . *_{\text{II}} w \Rightarrow \forall v \in V, f \circ \varphi(g_v) = \varphi(g_v) \circ f\end{aligned}$$

Proof.

$$\begin{aligned}&\forall s \in \mathcal{S}(V), \forall u \in V, \forall v \in V, \\ &\quad (f_w \circ \varphi(g_u))(s)[v] = \sum_{v \in V} \varphi(g_u)(s)[v]\, w[\varphi(g_v^{-1} g_u)] \\ &\qquad\qquad = \sum_{\substack{(a,b) \in V^2 \\ s.t.\ \ g_a g_b = g_v}} \varphi(g_u)(s)[a]\, w[b] \\ &\qquad\qquad = \sum_{\substack{(a,b) \in V^2 \\ s.t.\ \ g_a g_b = g_v}} s[\varphi(g_u)^{-1}(a)]\, w[b]\end{aligned}$$

$$= \sum_{\substack{(a,b)\in V^2 \\ s.t.\ g_{\varphi(g_u)(a)}g_b = g_v}} s[a]\, w[b]$$

Because φ is an isomorphism, its inverse $c \mapsto g_c$ is also an isomorphism and so $g_{\varphi(g_u)(a)}g_b = g_v \Leftrightarrow g_a g_b = g_{\varphi(g_u)^{-1}(v)}$. So we have both:

$$\begin{aligned}(f_w \circ \varphi(g_u))(s)[v] &= \sum_{\substack{(a,b)\in V^2 \\ s.t.\ g_a g_b = g_{\varphi(g_u)^{-1}(v)}}} s[a]\, w[b] \\ &= s *_{\text{II}} w[\varphi(g_u)^{-1}(v)] \\ &= (\varphi(g_u) \circ f_w)(s)[v]\end{aligned}$$

□

Remark. Note that convolution operators of the form $f_w = . *_{\text{I}} w$ are also equivariant to Γ, but the proposition and the proof are omitted as they are similar to the latter.

In fact, both group convolutions are the same as the latter one borrows the algebraic structure of the first one. Thus we only obtain equivariance to $\varphi(\Gamma)$ when φ also transfer the group structure from Γ to V, and the converse does not hold. To obtain equivariance to Γ (and its converse), we will drop the direct homomorphism condition, and instead we will take into account the fact that it contains invertible transformations of V.

2.2.2 Construction under group actions

Definition 60. Group action
An *action* of a group Γ on a set V is a function $L : \Gamma \times V \to V, (g, v) \mapsto L_g(v)$, such that the map $g \mapsto L_g$ is a homomorphism.
Given $g \in \Gamma$, the transformation L_g is called the action of g by L on V.

Remark. When there is no ambiguity, we use the same symbol for g and L_g.

Hence, note that $g \in \Gamma$ can act on both Γ through the left multiplication and on V as being an object of $\Phi^*(V)$. This ambivalence can be seen on a commutative diagram, see Figure 11.

$$\begin{array}{ccc} g_u & \xrightarrow{g_v} & g_v g_u \\ {\scriptstyle\varphi}\downarrow & & \downarrow{\scriptstyle\varphi} \\ u & \xrightarrow[g_v]{\text{(P)}} & \varphi(g_v g_u) \end{array}$$

Figure 11: Commutative diagram. All arrows except for the one labeled with (P) are always True.

For (P) to be true means that φ is an equivariant map *i.e.* whether the mapping is done before or after the action of Γ has no impact on the result. When such φ exists, Γ and V are said to be equivalent and we denote $\Gamma \equiv V$.

Definition 61. Equivariant map
A map φ from a group Γ acting on the destination set V and itself, is said to be an *equivariant map* if

$$\forall g, h \in \Gamma, g(\varphi(h)) = \varphi(g(h))$$

Remark. Here g acts on Γ through left multiplication so $g(h) = gh$.

Suppose we have $\Gamma \overset{\varphi}{\cong} V$. If we also have that $\Gamma \equiv V$, we are interested to know if then φ exhibits the equivalence.

Definition 62. φ-Equivalence
A subgroup $\Gamma \subset \Phi^*(V)$ such that $\Gamma \overset{\varphi}{\cong} V$, is said to be *$\varphi$-equivalent* if φ is a bijective equivariant map *i.e.* if it verifies the property:

$$\forall v, u \in V, g_v(u) = \varphi(g_v g_u) \tag{P}$$

In that case we denote $\Gamma \overset{\varphi}{\equiv} V$.

Remark. For example, translations on the grid graph, with $\varphi(t_{i,j}) = (i, j)$, are φ-equivalent as $t_{i,j}(a, b) = \varphi(t_{i,j} \circ t_{a,b})$. However, with $\varphi(t_{i,j}) = (-i, -j)$, they would not be φ-equivalent.

Definition 63. Group convolution III
Let a subgroup $\Gamma \subset \Phi^*(V)$ such that $\Gamma \overset{\varphi}{\cong} V$. The group convolution III between two signals s_1 and $s_2 \in \mathcal{S}(V)$ is defined as:

$$s_1 *_{\text{III}} s_2 = \sum_{v \in V} s_1[v]\, g_v(s_2) \tag{15}$$

$$= \sum_{g \in \Gamma} s_1[\varphi(g)]\, g(s_2) \tag{16}$$

The two expressions differ on the domain upon which the summation is done. The expression (15) put the emphasis on each vertex and its action, whereas the expression (16) emphasizes on each object of Γ.

Lemma 64. Relation with group convolution II
$\Gamma \overset{\varphi}{\equiv} V \Leftrightarrow *_{\text{II}} = *_{\text{III}}$

Proof.

$$\forall s_1, s_2 \in \mathcal{S}(V),$$
$$s_1 *_{\text{II}} s_2 = s_1 *_{\text{III}} s_2$$
$$\Leftrightarrow \forall u \in V, \sum_{v \in V} s_1[v]\, s_2[\varphi(g_v^{-1} g_u)] = \sum_{v \in V} s_1[v]\, s_2[g_v^{-1}(u)] \tag{17}$$

Hence, the direct sense is obtained by applying (P).
For the converse, given $u, v \in V$, we first realize (17) for $s_1 := \delta_v$, obtaining $s_2[\varphi(g_v^{-1} g_u)] = s_2[g_v^{-1}(u)]$, which we then realize for a real signal s_2 having no two equal entries, obtaining $\varphi(g_v^{-1} g_u) = g_v^{-1}(u)$. From the latter we finally obtain (P) with the one-to-one correspondence $g_{v'} := g_v^{-1}$. □

We can then coin the term φ-convolution.

Definition 65. φ-convolution
Let $\Gamma \overset{\varphi}{\equiv} V$, the *$\varphi$-convolution* between two signals s_1 and $s_2 \in \mathcal{S}(V)$ is defined as:

$$s_1 *_{\varphi} s_2 = s_1 *_{\text{II}} s_2 = s_1 *_{\text{III}} s_2$$

This time, we do obtain equivariance to Γ as expected, and the full characterization as well.

Proposition 66. Characterization by right-action equivariance to Γ
If Γ is φ-equivalent, the class of linear transformations of $\mathcal{S}(V)$ that are equivariant to Γ is exactly the class of φ-convolution operators acting on the

right of $\mathcal{S}(V)$ *i.e.*

$$\text{If } \Gamma \overset{\varphi}{\equiv} V,$$
$$\exists w \in \mathcal{S}(V), f = .*_{\varphi} w \Leftrightarrow \begin{cases} f \in \mathcal{L}(\mathcal{S}(V)) \\ \forall g \in \Gamma, f \circ g = g \circ f \end{cases}$$

Proof. 1. From left to right:

In the following equations, (18) is obtained by definition, (19) is obtained because left multiplication in a group is bijective, and (20) is obtained because of (P).

$$\begin{aligned} &\forall g \in \Gamma, \forall s \in \mathcal{S}(V), \\ &\qquad f_w \circ g(s) = \sum_{h \in \Gamma} g(s)[\varphi(h)]\, h(w) && (18) \\ &\qquad\qquad = \sum_{h \in \Gamma} g(s)[\varphi(gh)]\, gh(w) && (19) \\ &\qquad\qquad = \sum_{h \in \Gamma} g(s)[g(\varphi(h))]\, gh(w) && (20) \\ &\qquad\qquad = \sum_{h \in \Gamma} s[\varphi(h)]\, gh(w) \\ &\qquad\qquad = \sum_{h \in \Gamma} s[\varphi(h)]\, h(w)[g^{-1}(.)] \\ &\qquad\qquad = f_w(s)[g^{-1}(.)] \\ &\qquad\qquad = g \circ f_w(s) \end{aligned}$$

Of course, we also have that f_w is linear.

2. From right to left:

Let $f \in \mathcal{L}(\mathcal{S}(V)), s \in \mathcal{S}(V)$. By linearity of f, we distribute $f(s)$ on

the family of dirac signals:

$$f(s) = \sum_{v \in V} s[v] f(\delta_v) \tag{21}$$

Thanks to (P), we have that:

$$\begin{aligned} & g_v(\varphi(\mathrm{Id})) = \varphi(g_v \,\mathrm{Id}) = v \\ & \text{So, } v = u \Leftrightarrow \varphi(\mathrm{Id}) = g_v^{-1}(u) \\ & \text{So, } \delta_v = g_v(\delta_{\varphi(\mathrm{Id})}) \end{aligned}$$

By denoting $w = f(\delta_{\varphi(\mathrm{Id})})$, and using the hypothesis of equivariance, we obtain from (21) that:

$$\begin{aligned} f(s) &= \sum_{v \in V} s[v]\, f \circ g_v(\delta_{\varphi(\mathrm{Id})}) \\ &= \sum_{v \in V} s[v]\, g_v \circ f(\delta_{\varphi(\mathrm{Id})}) \\ &= \sum_{v \in V} s[v]\, g_v(w) \\ &= s *_\varphi w \end{aligned}$$

□

Construction of φ-convolutions on vertex domains

Proposition 66 tells us that in order to define a convolution on the vertex domain of a graph $G = \langle V, E \rangle$, all we need is a subgroup Γ of invertible transformations of V, that is equivalent to V. The choice of Γ can be done with respect to E. This is discussed in more details in Section 2.3, where we will see that in fact, we only need a generating set of Γ.

Exposure of φ
This construction relies on exposing a bijective equivariant map φ between Γ and V. In the next subsection, we show that in cases where Γ is abelian, we even need not expose φ and the characterization still holds.

2.2.3 Mixed domain formulation

From (16), we can define a mixed domain convolution *i.e.* that is defined for $r \in \mathcal{S}(\Gamma)$ and $s \in \mathcal{S}(V)$, without the need of expliciting φ.

Definition 67. Mixed domain convolution
Let a subgroup $\Gamma \subset \Phi^*(V)$ such that $V \cong \Gamma$. The *mixed domain convolution* between two signals $r \in \mathcal{S}(\Gamma)$ and $s \in \mathcal{S}(V)$ results in a signal $r *_{\text{M}} s \in \mathcal{S}(V)$ and is defined as:

$$r *_{\text{M}} s = \sum_{g \in \Gamma} r[g]\, g(s)$$

We coin it M-convolution. From a practical point of view, this expression of the convolution is useful because it relegates φ as an underpinning object.

Lemma 68. Relation with group convolution III
$\forall \varphi \in \text{BIJ}(\Gamma, V), \forall (r, s) \in \mathcal{S}(\Gamma) \times \mathcal{S}(V)$,

$$r *_{\text{M}} s = \varphi(r) *_{\text{III}} s$$

Proof. Let $\varphi \in \text{BIJ}(\Gamma, V), (r, s) \in \mathcal{S}(\Gamma) \times \mathcal{S}(V)$,

$$\begin{aligned} r *_{\text{M}} s = \sum_{g \in \Gamma} r[g]\, g(s) = \sum_{v \in V} r[g_v]\, g_v(s) &\overset{(\diamond)}{=} \sum_{v \in V} \varphi(r)[v]\, g_v(s) \\ &= \varphi(r) *_{\text{III}} s \end{aligned}$$

Where $\overset{(\diamond)}{=}$ comes from Lemma 56. □

In other words, $*_{\text{M}}$ is a convenient reformulation of $*_{\text{III}}$ which does not depend on a particular φ.

Lemma 69. Relation with group convolution I, II and φ-convolution
Let $\varphi \in \text{BIJ}(\Gamma, V), (r, s) \in \mathcal{S}(\Gamma) \times \mathcal{S}(V)$, we have:

$$\begin{aligned}\Gamma \overset{\varphi}{\equiv} V &\Leftrightarrow \forall v \in V, (r *_{\text{M}} s)[v] = (r *_{\text{I}} \varphi^{-1}(s))[g_v] \\ &\Leftrightarrow r *_{\text{M}} s = \varphi(r) *_{\text{II}} s \\ &\Leftrightarrow r *_{\text{M}} s = \varphi(r) *_{\varphi} s\end{aligned}$$

Proof. On one hand, Lemma 68 gives $r *_{\text{M}} s = \varphi(r) *_{\text{III}} s$. On the other hand, Lemma 58 gives $\forall v \in V, (r *_{\text{I}} \varphi^{-1}(s))[g_v] = (\varphi(r) *_{\text{II}} s)[v]$. Then Lemma 64 concludes. □

Remark. The converse sense is meaningful because it justifies that when the M-convolution is employed, the property $\Gamma \equiv V$ underlies, without the need of expliciting φ.

From M-convolution, we can derive operators acting on the left of $\mathcal{S}(V)$, of the form $s \mapsto w *_{\text{M}} s$, parameterized by $w \in \mathcal{S}(\Gamma)$. In particular, these operators would be relevant as layers of neural networks. On the contrary, derived operators acting on the right such as $r \mapsto r *_{\text{M}} w$ wouldn't make sense with this formulation as they would make φ resurface. However, the equivariance to Γ incurring from Lemma 68 and Proposition 66 only holds for operators acting on the right. So we need to intertwine an abelian condition as follows. This is also a good excuse to see the influence of abelianity.

Proposition 70. Equivariance to Γ through left action
Let a subgroup $\Gamma \subset \Phi^*(V)$ such that $\Gamma \cong V$. Γ is abelian, if and only if, M-convolution operators acting on the left of $\mathcal{S}(V)$ are equivariant to it *i.e.*

$$\forall g, h \in \Gamma, gh = hg \Leftrightarrow \forall w, g \in \Gamma, w *_{\text{M}} g(.) = g \circ (w *_{\text{M}} .)$$

Proof. Let $w, g \in \Gamma$, and define $f_w : s \mapsto w *_{\text{M}} s$. In the following expressions, Γ is abelian if and only if (22) and (23) are equal (the converse is obtained by particularizing on well chosen signals):

$$\begin{aligned} f_w \circ g(s) &= \sum_{h \in \Gamma} w[h]\, hg(s) && (22)\\ &= \sum_{h \in \Gamma} w[h]\, gh(s) && (23)\\ &= \sum_{h \in \Gamma} w[h]\, h(s)[g^{-1}(.)] \\ &= (w *_{\text{M}} s)[g^{-1}(.)] \\ &= g \circ f_w(s) \end{aligned}$$

□

Remark. Similarly, $*_\varphi$ is also equivariant to Γ through left action if and only if Γ is abelian, as a consequence of being commutative if and only if Γ is abelian. On the contrary, note that commutativity of $*_{\text{M}}$ doesn't make sense.

Corrolary 71. Characterization by left-action equivariance to Γ
Let $\Gamma \cong V$. If Γ is abelian, the class of linear transformations of $\mathcal{S}(V)$ that are equivariant to Γ is exactly the class of M-convolution operators acting on

the left of $\mathcal{S}(V)$ *i.e.*

$$\text{If } \Gamma \cong V \text{ and } \Gamma \text{ is abelian,}$$
$$\exists w \in \mathcal{S}(\Gamma), f = w *_{\text{M}} . \Leftrightarrow \begin{cases} f \in \mathcal{L}(\mathcal{S}(V)) \\ \forall g \in \Gamma, f \circ g = g \circ f \end{cases}$$

Proof. By picking φ such that $\Gamma \overset{\varphi}{\equiv} V$ with Lemma 69 and using the relation between $*_{\text{M}}$ and $*_{\varphi}$. □

Depending on the applications, we will build upon either $*_{\varphi}$ or $*_{\text{M}}$ when the abelian condition is satisfied.

2.3 Inclusion of the edge set in the construction

The constructions from the previous section involve the vertex set V and depend on Γ, a subgroup of the set of invertible transformations on V. Therefore, it looks natural to try to relate the edge set and Γ.
There are two approaches. Either Γ describes an underlying graph structure $G = \langle V, E \rangle$, either G can be used to define a relevant subgroup Γ to which the produced convolutive operators will be equivariant. Both approaches will help characterize classes of graphs that can support natural definitions of convolutions.

2.3.1 Edge-constrained convolutions

In this subsection, we are trying to answser the following question:

- What graphs admit a φ-convolution, or an M-convolution (in the sense that they can be defined with the characterization), under the condition that Γ is generated by a set of edge-constrained transformations ?

Definition 72. Edge-constrained transformation
An *edge-constrained* (EC) transformation on a graph $G = \langle V, E \rangle$ is a transformation $f : V \mapsto V$ such that

$$\forall u, v \in V, f(u) = v \Rightarrow u \overset{E}{\sim} v$$

We denote $\Phi_{\text{EC}}(G)$ and $\Phi^*_{\text{EC}}(G)$ the sets of (EC) and invertible (EC) transformations. When a convolution is defined as a sum over a set that is in one-to-one correspondence with a group that is generated from a set of (EC) transformations, we call it an (EC) convolution.

Remark. Note that $\Phi^*_{\text{EC}}(G)$ is not a group, thus why we are interested in groups and their generating sets.

This leads us to consider Cayley graphs (Cayley, 1878).

Definition 73. Cayley graph
Let a group Γ and one of its generating set $\mathcal{U}$. The *Cayley graph* generated by $\mathcal{U}$, is the digraph $\vec{G} = \langle V, E \rangle$ such that $V = \Gamma$ and E is such that:

$$a \to b \Leftrightarrow \exists g \in \mathcal{U}, ga = b$$

Also, if Γ is abelian, we call it an *abelian Cayley graph*. We call *Cayley subgraph*, a subgraph that is isomorph to a Cayley graph.

Remark. Note that for compatibility with transformations that are left actions, we define Cayley graphs with $ga = b$ instead of $ag = b$.

Convolution on Cayley graphs
In the case of Cayley graphs, it is clear that $\mathcal{U} \subseteq \Phi^*_{\text{EC}}$ and $\Phi^* \supseteq \langle \mathcal{U} \rangle \equiv V$. So that they admit (EC) φ-convolutions, and (EC) M-convolutions in the abelian case.
More precisely, we obtain the following characterization:

Proposition 74. Characterization by Cayley subgraph isomorphism
Let a graph $G = \langle V, E \rangle$, then:

(i) G admits an (EC) φ-convolution if and only if it contains a subgraph isomorph to a Cayley graph

(ii) G admits an (EC) M-convolution if and only if it contains a subgraph isomorph to an abelian Cayley graph

Proof. We show the result only in the general case as the proof for the abelian case is similar.

1. From left to right: as a direct application of the definitions.

2. From right to left:
   Let a graph $G = \langle V, E \rangle$. We suppose it contains a subgraph $\vec{G_s} = \langle V_s, E_s \rangle$ that is graph-isomorph to a Cayley graph $\vec{G_c} = \langle V_c, E_c \rangle$, generated by $\mathcal{U}$. Let ψ be a graph isomorphism from G_s to G_c. To obtain the proof, we need to find a group of invertible transformations Γ of V_s generated by a set of (EC) transformations, such that $\Gamma \equiv V_s$.

   Let's define the group action $L : V_c \times V_s \to V_s$ inductively as follows:

   (a) $\forall g \in \mathcal{U}, L_g(u) = v \Leftrightarrow g\psi(u) = \psi(v)$

   (b) Whenever L_g and L_h are defined, the action of gh is defined by homomorphism as $L_{gh} = L_g \circ L_h$

   (c) Whenever L_g is defined, the action of g^{-1} is defined by homomorphism as $L_{g^{-1}} = L_g^{-1}$ *i.e.* $L_{g^{-1}}(u) = v \Leftrightarrow \psi(u) = g\psi(v)$

   Note that the induction transfers the property (a) to all $g \in V_c$ in a transitive manner because

$$L_{gh}(u) = L_g(L_h(u)) = w \Leftrightarrow \exists v \in V_s \begin{cases} L_h(u) = v \\ L_g(v) = w \end{cases}$$

   and

$$\exists v \in V_s \begin{cases} h\psi(u) = \psi(v) \\ g\psi(v) = \psi(w) \end{cases} \Leftrightarrow gh\psi(u) = \psi(w)$$

   We must also verify that this construction is well-defined, *i.e.* whenever we define an action with (b) or (c), if the action was already defined, then they must be equal. This is the case because the homomorphism

$g \mapsto L_g$ on V_c is in fact an isomorphism as

$$\begin{aligned} L_g = L_h &\Leftrightarrow \forall u \in V, L_g(u) = L_h(u) \\ &\Leftrightarrow \forall u \in V, g\psi(u) = h\psi(u) \\ &\Leftrightarrow g = h \end{aligned}$$

Also note that (c) is needed only in case that V_c is infinite.

Denote the set $L_{\mathcal{U}} = \{L_g, g \in \mathcal{U}\}$ and $\Gamma = \langle L_{\mathcal{U}} \rangle \cong V_c$. Let's define the map φ as:

$$\begin{aligned} \Gamma &\to V_s \\ \varphi : L_g &\mapsto L_g(\psi^{-1}(\mathrm{Id})) \end{aligned}$$

φ is bijective because $\forall g \in V_c, \varphi(L_g) = \psi^{-1}(g)$ thanks to (a).

Additionally, we have:

$$\begin{aligned} L_h(\varphi(L_g) &= L_h(L_g(\psi^{-1}(\mathrm{Id}))) \\ &= L_h \circ L_g(\psi^{-1}(\mathrm{Id})) \\ &= L_{hg}(\psi^{-1}(\mathrm{Id})) \\ &= \varphi(L_{hg}) \\ &= \varphi(L_h \circ L_g) \end{aligned}$$

That is, φ is a bijective equivariant map and $\langle L_{\mathcal{U}} \rangle = \Gamma \overset{\varphi}{\equiv} V_s$. Moreover, $L_{\mathcal{U}}$ is a set of (EC) transformations thanks to (a). Therefore, G admits an (EC) φ-convolution.

□

Corrolary 75. Characterization by φ
Let a graph $G = \langle V, E \rangle$, and a set $\mathcal{U} \subset \Phi^*_{\text{EC}}(G)$ *s.t.*

$$\langle \mathcal{U} \rangle \cong \Gamma \equiv V' \subset V$$

G admits an (EC) φ-convolution, if and only if, φ is a graph isomorphism between the Cayley graph generated by $\mathcal{U}$ and the subgraph induced by V'.

The proof is omitted as it would be highly similar to the previous one.

2.3.2 Intrinsic properties

- Obviously the constructed convolutions are linear. But do they also preserve the locality and weight sharing properties ?

Let $\vec{G} = \langle V, E \rangle$ be a Cayley subgraph, generated by $\mathcal{U}$, of some graph G. Recall that its (EC) φ-convolution operator is a right operator, and can be expressed as

$$\begin{aligned} &\forall s \in \mathcal{S}(V), \forall u \in V, \\ &\quad f_w(s)[u] = (s *_\varphi w)[u] \\ &\qquad\qquad = \sum_{v \in V} s[v]\, w[g_v^{-1}(u)] \end{aligned} \tag{24}$$

From this expression, it is not obvious that f_w is a local operator. To see this, we can show for example the following proposition.

Proposition 76. Locality
When the support of w is a compact (in the sense that its induced subgraph in G is connected), of diameter d, the same holds for the support of the sum Σ in (24). More precisely, the subgraph induced by the support of Σ is isomorphic to the transpose of the subgraph induced by the support of w.

Proof. Without loss of generality subject to growing $\mathcal{U}$, let's suppose that w has a support $\mathcal{M} = \varphi(\mathcal{N})$, such that $\mathcal{N} \subset \mathcal{U}$. $\mathcal{N}$ and $\mathcal{M}$ are obviously compacts of diameter 2. Thanks to (P), we have

$$\begin{aligned} g_v^{-1}(u) \in \mathcal{M} &\Leftrightarrow u \in g_v(\mathcal{M}) = g_v(\varphi(\mathcal{N})) = \varphi(g_v\mathcal{N}) \\ &\Leftrightarrow g_u \in g_v\mathcal{N} \\ &\Leftrightarrow g_v^{-1} \in \mathcal{N}g_u^{-1} \\ &\Leftrightarrow g_v \in g_u\mathcal{N}^{-1} \\ &\Leftrightarrow v \in g_u(\varphi(\mathcal{N}^{-1})) \end{aligned}$$

where $\mathcal{N}^{-1}$ reverses the edges of $\mathcal{N}$. Let's denote $\mathcal{K}_u = g_u(\varphi(\mathcal{N}^{-1})) \subset V$.
TODO: FALSE, consider $g_u(v) = g_v(u)$

TODO: In reality we have: local conv iff SNP iff simple transitive Aut iff Cayley graph
TODO: EC local conv iff Abelian Cayley graph

By composing edge reversal and graph isomorphisms (as φ and its inverse are graph isomorphisms by Proposition 75), the compactness and diameter of $\mathcal{M}$ is preserved for $\mathcal{K}_u$. More precisely, the transposed subgraph structure is also preserved. □

Let's define $\mathcal{M}$, $\mathcal{N}$ and $\mathcal{K}_u$ as in the previous proof.

Definition 77. Supporting set
The *supporting set* of an (EC) convolution operator f_w, is a set $\mathcal{N} \subset \Phi^*_{\text{EC}}$, such that

(i) when $*$ is $*_\varphi$: $0 \notin w[\mathcal{M}]$, where $\mathcal{M} = \varphi(\mathcal{N})$

(ii) when $*$ is $*_{\text{M}}$: $0 \notin w[\mathcal{N}]$

Definition 78. Local patch for $*_\varphi$
The *local patch* at $u \in V$ of an (EC) φ-convolution operator f_w is defined as $\mathcal{K}_u = g_u(\varphi(\mathcal{N}^{-1}))$.

Remark. In other terms, $\mathcal{K}_{\mathrm{Id}} = \varphi(\mathcal{N}^{-1})$ is the *initial local patch*, which is composed of all vertices that are connected in direction to $\varphi(\mathrm{Id})$; and $\mathcal{K}_u$ is obtained by moving $\mathcal{K}_{\mathrm{Id}}$ on the Cayley subgraph via the edges corresponding to the decomposition of g_u on the generating set $\mathcal{U}$.

To see that the weights are tied in the general case (i), we can show the following proposition.

Proposition 79. Weight sharing
$\forall a, \alpha \in V, \forall b \in \mathcal{K}_a : \exists \beta \in \mathcal{K}_\alpha \Leftrightarrow g_\beta^{-1}(\alpha) = g_b^{-1}(a)$

Proof. By using (P),

$$\begin{aligned} g_{\mathcal{K}_\alpha}^{-1}(\alpha) = g_{\mathcal{K}_a}^{-1}(a) &\Leftrightarrow g_\alpha^{-1} g_{\mathcal{K}_\alpha} = g_a^{-1} g_{\mathcal{K}_a} \\ &\Leftrightarrow \mathcal{K}_\alpha = g_\alpha g_a^{-1}(\mathcal{K}_a) = g_\alpha g_a^{-1} g_a(\varphi(\mathcal{N}^{-1})) \\ &\Leftrightarrow \mathcal{K}_\alpha = g_\alpha(\varphi(\mathcal{N}^{-1})) \end{aligned}$$

□

2.3.3 Stricly edge-constrained convolutions

We make the disctinction between general (EC) convolution operators and those for which the weight kernel w is smaller and is supported only on (EC) transformations of $\mathcal{U}$.

Definition 80. Strictly (EC) convolution operator
A *strictly* edge-constrained (EC*) convolution operator f_w, is an (EC) convolution operator such that its supporting set $\mathcal{N} \subset \mathcal{U}$.

Remark. (EC*) convolution operators are simpler to obtain as we can construct them just with $\mathcal{U} \subset \Phi^*_{\text{EC}}(G)$ without composing the transformations.

Let f_w be an (EC*) convolutional operator. In the general case (i), $w \in \mathcal{S}(V)$, so its support is $\mathcal{M} = \varphi(\mathcal{N})$ such that $\mathcal{N} \subseteq \mathcal{U}$. In the abelian case (ii), we use instead $w \in \mathcal{S}(\Gamma)$, and thus its support is directly $\mathcal{N}$. Therefore, we can rewrite the expressions of the convolution operator as:

(i) $\forall s \in \mathcal{S}(V), \forall u \in V, f_w(s)[u] \overset{(\varphi)}{=} \sum_{v \in \mathcal{K}_u} s[v]\, w[g_v^{-1}(u)]$

(ii) $\forall s \in \mathcal{S}(V), f_w(s) \overset{(\text{M})}{=} \sum_{g \in \mathcal{N}} w[g]\, g(s)$

Remark. Note that in the abelian case, we can see from (ii) that a definition of a local patch would coincide with the supporting set, so that locality and weight sharing is straightforward.

Construction
From these expressions, it is clear that Γ needs not to be fully determined to calculate $f_w(s)[u]$. The case (ii) is the simplest as the only requirement is a supporting set $\mathcal{N}$ of (EC) invertible transformations. In the case (i), we also need to determine $\mathcal{K}_u$.

Strict locality
Note that $f_w(s)[u]$ is a weighted aggregation of entries $s[v]$ for $v \in \mathcal{K}_u$. As $\mathcal{K}_{\text{Id}} = \varphi(\mathcal{N}^{-1}) = \mathcal{N}^{-1}(\varphi(\text{Id}))$, $\mathcal{K}_{\text{Id}}$ contains only neighbors of $\varphi(\text{Id})$, and so $\mathcal{K}_u = g_u(\mathcal{K}_{\text{Id}})$ contains only neighbors of u. Therefore, in both cases $f_w(s)[u]$ is a weighted aggregation of entries located in the neighborhood of u.

Complexity

Another merit is that (EC*) convolutions have a complexity of $\mathcal{O}(kn)$, where $n = |V|$ is the degree of the graph, and $k = |\mathcal{N}|$ is the size of the weight kernel. In comparison, (EC) convolutions have complexity up to $\mathcal{O}(n^2)$.

2.4 From groups to groupoids

2.4.1 Motivation

One possible limitation coming from searching for Cayley subgraphs is that they are degree-regular *i.e.* the in- and the out-degree $d = |\mathcal{U}|$ of each vertex is the same. That is, for a general graph G, the size of the weight kernel w of an (EC*) convolution operator f_w supported on $\mathcal{U}$ is bounded by d, which in turn is bounded by twice the mininimal degree of G (twice because G is undirected and $\mathcal{U}$ can contain every inverse).
There are a lot of possible strategies to overcome this limitation. For example:

1. connecting each vertex with its k-hop neighbors, with $k > 1$,
2. artificially creating new connections for less connected vertices,
3. ignoring less connected vertices,
4. allowing the supporting set $\mathcal{N}$ to exceed $\mathcal{U}$ *i.e.* dropping * in (EC*).

These strategies require to concede that the topological structure supported by G is not the best one to support an (EC*) convolution on it, which breeds the following question:

- What can we relax in the previous (EC*) contruction in order to unbound the supporting set, and still preserve the equivariance characterization?

The latter constraint is a consequence that every vertex of the Cayley subgraph $\vec{G}$ must be composable with every generator from $\mathcal{U}$. Therefore, an answer consists in considering groupoids (Brandt, 1927) instead of groups. Roughly speaking, a groupoid is almost a group except that its composition law needs not be defined everywhere. Weinstein, 1996, unveiled the benefits to base convolutions on groupoids instead of groups in order to exploit partial symmetries.

2.4.2 Definition of notions related to groupoids

Definition 81. Groupoid

A *groupoid* Υ is a set equipped with a partial composition law with domain $\mathcal{D} \subset \Upsilon \times \Upsilon$, called *composition rule*, that is

1. closed into Υ *i.e.* $\forall (g,h) \in \mathcal{D}, gh \in \Upsilon$
2. associative *i.e.* $\forall f,g,h \in \Upsilon$, $\begin{cases} (f,g),(g,h) \in \mathcal{D} \Leftrightarrow (fg,h),(f,gh) \in \mathcal{D} \\ (f,g),(fg,h) \in \mathcal{D} \Leftrightarrow (g,h),(f,gh) \in \mathcal{D} \\ \text{when defined, } (fg)h = f(gh) \end{cases}$
3. invertible *i.e.* $\forall g \in \Upsilon, \exists! g^{-1} \in \Upsilon \quad s.t. \begin{cases} (g,g^{-1}),(g^{-1},g) \in \mathcal{D} \\ (g,h) \in \mathcal{D} \Rightarrow g^{-1}gh = h \\ (h,g) \in \mathcal{D} \Rightarrow hgg^{-1} = h \end{cases}$

Optionally, it can be *domain-symmetric i.e.* $(g,h) \in \mathcal{D} \Leftrightarrow (h,g) \in \mathcal{D}$, and *abelian i.e.* domain-symmetric with $gh = hg$.

Remark. Note that left and right inverses are necessarily equal (because $(gg^{-1})g = g(g^{-1}g)$). Also note we can define a right identity element $e_g^r = g^{-1}g$, and a left one $e_g^l = gg^{-1}$, but they are not necessarily equal and depend on g.

Most definitions related to groups can be adapted to groupoids. In particular, let's adapt a few notions.

Definition 82. Groupoid partial action

A partial *action* of a groupoid Υ on a set V, is a function L, with domain $\mathcal{D}_L \subset \Upsilon \times V$ and valued in V, such that the map $g \mapsto L_g$ is a groupoid homomorphism.

Remark. As usual, we will confound L_g and g when there is no possible confusion, and we denote $\mathcal{D}_{L_g} = \mathcal{D}_g = \{v \in V, (g, v) \in \mathcal{D}_L\}$.

Definition 83. Partial equivariant map
A map φ from a groupoid Υ partially acting on the destination set V is said to be a *partial equivariant map* if

$$\forall g, h \in \Upsilon, \begin{cases} \varphi(h) \in \mathcal{D}_g \Leftrightarrow (g, h) \in \mathcal{D} \\ g(\varphi(h)) = \varphi(gh) \end{cases}$$

Also, φ-equivalence between a subgroupoid and a set is defined similarly with φ being a bijective *partial* equivariant map between them.

Definition 84. Partial transformations groupoid
The *partial transformations groupoid* $\Psi^*(V)$, is the set of invertible partial transformations, equipped with the functional composition law with domain $\mathcal{D}$ such that

$$\begin{cases} \mathcal{D}_{gh} = h(\mathcal{D}_h) \cap \mathcal{D}_g \\ (g, h) \in \mathcal{D} \Leftrightarrow \mathcal{D}_{gh} \neq \emptyset \end{cases}$$

Remark. Note that a subgroupoid $\Upsilon \subset \Psi^*(V)$ is domain-symmetric when $\exists v \in V, g(v) \in \mathcal{D}_h \Leftrightarrow \exists u \in V, h(u) \in \mathcal{D}_g$

2.4.3 Construction of partial convolutions

The expression of the convolution we constructed in the previous section cannot be applied as is. We first need to extend the algebraic objects we work with. Extending a partial transformation g on the signal space $\mathcal{S}(V)$ (and thus the convolutions) is a bit tricky, because only the signal entries corresponding to $\mathcal{D}_g$ are moved. A convenient way to do this is to consider the groupoid closure obtained with the addition of an absorbing element.

Definition 85. Zero-closure
The *zero-closure* of a groupoid Υ, denoted Υ^0, is the set $\Upsilon \cup 0$, such that the groupoid axioms 1, 2 and 3, and the domain $\mathcal{D}$ are left unchanged, and

4. the composition law is extended to $\Upsilon^0 \times \Upsilon^0$ with $\forall (g,h) \notin \mathcal{D}, gh = 0$

Remark. Note that this is coherent as the properties 2 and 3 are still partially defined on the original domain $\mathcal{D}$.

Now, we will also extend every other algebraic object used in the expression of the φ-convolution and the M-convolution, so that we can directly apply our previous constructions.

Lemma 86. Extension of φ on V^0
Let a partial equivariant map $\varphi : \Upsilon \to V$. It can be extended to a (total) equivariant map $\varphi : \Upsilon^0 \to V^0 = V \cup \varphi(0)$, such that $\varphi(0) \notin V$, that we denote $0_V = \varphi(0)$, and such that

$$\forall g \in \Upsilon^0, \forall v \in V^0, g(v) = \begin{cases} \varphi(g g_v) & \text{if } g_v \in \mathcal{D}_g \\ 0_V & \text{else} \end{cases}$$

Proof. We have $\varphi(0) \notin V$ because φ is bijective. Additionaly, we must have $\forall (g,h) \notin \mathcal{D}, g(\varphi(h)) = \varphi(gh) = \varphi(0) = 0_V$. □

Remark. Note that for notational conveniency, we may use the same symbol 0 for 0_Υ, 0_V and $0_\mathbb{R}$.

Similarly to $\Phi^*(V)$, $\Psi^*(V)$ can also move signals of $\mathcal{S}(V)$.

Lemma 87. Extension of injective partial transformations to $\mathcal{S}(V)$
Let $g \in \Psi^*(V)$. Its extension is done in two steps:

1. g is extended to $V^0 = V \cup \{0_V\}$ as $g(v) = 0_V \Leftrightarrow v \notin \mathcal{D}_g$.

2. Under the convention $\forall s \in \mathcal{S}(V), s[0_V] = 0_{\mathbb{R}}$, g is extended via linear extension to $\mathcal{S}(V)$, and we have

$$\forall s \in \mathcal{S}(V), \forall v \in V, g(s)[v] = s[g^{-1}(v)]$$

Proof. Straightforward. □

With these extensions, we can obtain the partial φ- and M-convolutions related to Υ almost by substituting Υ^0 to Γ in Definition 65 and Definition 67.

Definition 88. Partial convolution
Let a subgroupoid $\Upsilon \subset \Psi^*(V)$, such that $\Upsilon \overset{\varphi}{\equiv} V$. The partial φ- and M-convolutions, based on Υ, are defined on its zero-closure, with the same expression as if Υ^0 were a subgroup, and by extension of φ and of the groupoid partial actions *i.e.*

(i) $\forall s, w \in \mathcal{S}(V), s *_\varphi w = \sum_{v \in V} s[v]\, g_v(w) = \sum_{g \in \Upsilon} s[\varphi(g)]\, g(w)$

(ii) $\forall (w, s) \in \mathcal{S}(\Upsilon) \times \mathcal{S}(V), w *_{\mathrm{M}} s = \sum_{g \in \Upsilon} w[g]\, g(s)$

Symmetrical expressions
Note that, as $\forall r, r[0] = 0$, the partial convolutions can also be expressed on the domain $\mathcal{D}$ with a convenient symmetrical expression:

(i) $\forall u \in V, (s *_\varphi w)[u] = \sum_{\substack{(g_a, g_b) \in \mathcal{D} \\ s.t.\ g_a g_b = g_u}} s[a]\, w[b]$

(ii) $\forall u \in V, (w *_{\mathrm{M}} s)[u] = \sum_{\substack{v \in \mathcal{D}_g \\ s.t.\ g(v) = u}} w[g]\, s[v]$

We obtain an equivariance characterization similar to Proposition 66 and Corrolary 71.

Proposition 89. Characterization by equivariance to Υ
Let a subgroupoid $\Upsilon \subset \Psi^*(V)$, such that $\Upsilon \overset{\varphi}{\rightrightarrows} V$, with $*$ based on Υ.

1. Then,
   (i) partial φ-convolution right-operators are equivariant to Υ,
   (ii) if Υ is abelian, partial M-convolution left-operators are equiv to Υ.
2. Conversely,
   (i) if Υ is domain-symmetric, linear transformations of $\mathcal{S}(V)$ that are equivariant to Υ are partial φ-convolution right-operators,
   (ii) if Υ is abelian, they are also partial M-convolution left-operators.

Proof. (i) (a) Direct sense:
Using the symmetrical expressions, and the fact that $\forall r, r[0] = 0$, we have

$$\begin{aligned}
(f_w \circ g(s))[u] &= \sum_{\substack{(g_a,g_b)\in\mathcal{D} \\ s.t.\ g_a g_b = g_u}} g(s)[a]\, w[b] \\
&= \sum_{\substack{(g_a,g_b)\in\mathcal{D} \\ s.t.\ g_a g_b = g_u}} s[g^{-1}(a)]\, w[b] \\
&= \sum_{\substack{(g_a,g_b)\in\mathcal{D} \\ s.t.\ (g,g_a)\in\mathcal{D} \\ s.t.\ g g_a g_b = g_u}} s[a]\, w[b] \\
&= \sum_{\substack{(g_a,g_b)\in\mathcal{D} \\ s.t.\ (g,g_a)\in\mathcal{D} \\ s.t.\ g_a g_b = g^{-1} g_u = g_{\varphi(g^{-1} g_u)} = g_{g^{-1}(u)}}} s[a]\, w[b] \\
&= f_w(s)[g^{-1}(u)] \\
&= (g \circ f_w(s))[u]
\end{aligned}$$

(b) Converse:
Let $v \in V$. Denote $e^r_{g_v} = g_v^{-1} g_v$ the right identity element of g_v, and $e^r_v = \varphi(e^r_{g_v})$. We have that

$$g_v(e^r_v) = v$$
$$\text{So, } \delta_v = g_v(\delta_{e^r_v})$$

Let $f \in \mathcal{L}(\mathcal{S}(V))$ that is equivariant to Υ, and $s \in \mathcal{S}(V)$. Thanks to the previous remark we obtain that

$$\begin{aligned} f(s) &= \sum_{v \in V} s[v]\, f(\delta_v) \\ &= \sum_{v \in V} s[v]\, f(g_v(\delta_{e^r_v})) \\ &= \sum_{v \in V} s[v]\, g_v(f(\delta_{e^r_v})) \\ &= \sum_{v \in V} s[v]\, g_v(w_v) \end{aligned} \tag{25}$$

where $w_v = f(\delta_{e^r_v})$. In order to finish the proof, we need to find w such that $\forall v \in V, g_v(w) = g_v(w_v)$.

Let's consider the equivalence relation $\mathcal{R}$ defined on $V \times V$ such that:

$$\begin{aligned} a \mathcal{R} b &\Leftrightarrow w_a = w_b \\ &\Leftrightarrow e^r_a = e^r_b \\ &\Leftrightarrow g_a^{-1} g_a = g_b^{-1} g_b \\ &\Leftrightarrow (g_b, g_a^{-1}) \in \mathcal{D} \\ &\Leftrightarrow (g_a^{-1}, g_b) \in \mathcal{D} \end{aligned} \tag{26}$$

with (26) owing to the fact that Υ is domain-symmetric.

Given $x \in V$, denote its equivalence class $\mathcal{R}(x)$. Under the hypothesis of the axiom of choice (Zermelo, 1904) (if V is infinite), define the set $\aleph$ that contains exactly one representative per equivalence class. Let $w = \sum_{n\in\aleph} w_n$. Then V is the disjoint union $V = \cup_{n\in\aleph}\mathcal{R}(n)$ and (25) rewrites:

$$\begin{aligned}\forall u \in V, f(s)[u] &= \sum_{n\in\aleph}\sum_{v\in\mathcal{R}(n)} s[v]\, g_v(w_n)[u] \\ &= \sum_{n\in\aleph}\sum_{v\in\mathcal{R}(n)} s[v]\, w_n[g_v^{-1}(u)] \\ &= \sum_{n\in\aleph}\sum_{v\in\mathcal{R}(n)} s[v]\, w[g_v^{-1}(u)] \qquad (27) \\ &= (s *_\varphi w)[u]\end{aligned}$$

where (27) is obtained thanks to (26).

(ii) With symmetrical expressions, it is clear that the convolution is abelian, if and only if, Υ is abelian. Then (i) concludes.

□

Inclusion of (EC)

Similarly to the construction in Section 2.3, partial convolutions can define (EC) and (EC*) counterparts with a characterization of admissibility by groupoid Cayley subgraph isomorphism, and similar intrinsic properties.

Limitation of partial convolutions

However, because of the groupoid associativity, if $g \in \Psi^*_{\text{EC}}(G)$, then, any $v \in V$ *s.t.* $g(u) = v$ would be constrained to allow to be acted by every h *s.t.* $(h, g) \in \mathcal{D}$, which fails at unbounding the supporting set of a partial (EC*) convolutions.

2.4.4 Construction of path convolutions

To answer the limitation of partial convolutions, given $g \in \langle \mathcal{U} \rangle$ where $\mathcal{U} \subset \Psi^*_{\text{EC}}(G)$, the idea is to proceed with a foliation of g into pieces, each corresponding to an edge $e \in E$, and together generating another groupoid with a different associativity law, as follows.

Definition 90. Path groupoid
Let $\mathcal{U} \subset \Psi^*_{\text{EC}}(G)$. The *path groupoid* generated from $\mathcal{U}$, denoted $\mathcal{U} \ltimes G$, with composition rule $\mathcal{D}_{\ltimes}$, is the groupoid obtained inductively with:

1. $\mathcal{U} \ltimes_1 G = \{(g, v) \in \mathcal{U} \times V, v \in \mathcal{D}_g\} \subset \mathcal{U} \ltimes G$
2. $((g_n, v_n) \cdots (g_1, v_1), (h_m, u_m) \cdots (h_1, u_1)) \in \mathcal{D}_{\ltimes} \Leftrightarrow h_m(u_m) = v_1$
3. $((g_n, v_n) \cdots (g_1, v_1))^{-1} = (g_1^{-1}, g_1(v_1)) \cdots (g_n^{-1}, g_n(v_n))$

Call path its objects. Given a length $l \in \mathbb{N}$, denote $\mathcal{U} \ltimes_l G$ the subset composed of the paths that are the composition of exactly l paths of $\mathcal{U} \ltimes_1 G$.

Remark. This groupoid construction is inspired from the field of operator algebra where partial action groupoids have been extensively studied, *e.g.* Nica, 1994; Exel, 1998; Li, 2016.

Such groupoids usually come equipped with source and target maps. We also define the path map.

Definition 91. Source, target and path maps
Let a path groupoid $\mathcal{U} \ltimes G$. We define on it the *source map* α the *target map* β and the *path map* γ as:

$$\begin{cases} \alpha : (g_n, v_n) \cdots (g_1, v_1) \mapsto v_1 \in V \\ \beta : (g_n, v_n) \cdots (g_1, v_1) \mapsto g_n(v_n) \in V \\ \gamma : (g_n, v_n) \cdots (g_1, v_1) \mapsto g_n g_{n-1} \dots g_1 \in \Psi^*(V^0) \end{cases}$$

Remark. Note that the path groupoid can also be obtained by derivation of the partial transformation groupoid (*i.e.* $p \in \mathcal{U} \ltimes G$ can be seen as a derivative of $\gamma(p)$ *w.r.t.* $\alpha(p)$), and can thus be seen as the local structure of it.

Lemma 92.
Note the following properties:

1. $(p, q) \in \mathcal{D}_{\ltimes} \Leftrightarrow \alpha(p) = \beta(q)$
2. $\alpha(p) = \beta(p^{-1})$
3. $e_p^l = pp^{-1} = (\mathrm{Id}, \beta(p))$ and $e_p^r = p^{-1}p = (\mathrm{Id}, \alpha(p))$
4. γ is a groupoid partial action. We will denote γ_p instead of $\gamma(p)$.

Remark. Note that this time we won't use the notation $p(v)$ for $\gamma_p(v)$ for clarity.

One of the key object of our contruction is the use of φ-equivalence in order to transform a sum over a group(oid) of (partial) transformations, into a sum over the vertex set. With the current notion of path groupoid, searching for something similar amounts to searching for a graph traversal.

Definition 93. Traversal set
Let a graph $G = \langle V, E \rangle$ that is connected. A *traversal set* is a pair $(\mathcal{U}, \mathcal{T})$ of (EC) partial transformations subsets $\subset \Psi^*_{\mathrm{EC}}(G)$, such that

1. $\mathcal{U}$ is *edge-deterministic*, in the sense that an edge can only correspond to a unique g, *i.e.* $\forall g, h \in \mathcal{U} : \exists v \in V, g(v) = h(v) \Rightarrow g = h$
2. The (EC) partial transformations of $\mathcal{T}$ are restrictions of those of $\mathcal{U}$, *i.e.* $\forall g \in \mathcal{U}, \exists! h \in \mathcal{T}, \begin{cases} \mathcal{D}_h \subset \mathcal{D}_g \\ \forall v \in \mathcal{D}_h, h(v) = g(v) \end{cases}$
   (equivalently, $\mathcal{T} \ltimes G$ is a path subgroupoid of $\mathcal{U} \ltimes G$ *s.t.* $|\mathcal{T}| = |\mathcal{U}|$)
3. The subgraph $G_{\mathcal{T}} = \langle V, \mathcal{T} \ltimes_1 G \rangle$ is a spanning tree of G.

We denote $(\mathcal{U}, \mathcal{T}) \in \text{trav}(G)$, and denote by r the root of $G_{\mathcal{T}}$.
For $p \in \mathcal{T} \ltimes G \subset \mathcal{U} \ltimes G$, we denote $\gamma_p^{\mathcal{T} \ltimes G}$ and $\gamma_p^{\mathcal{U} \ltimes G}$ its path maps.

Remark. The assumption that the graph G is connected doesn't lose generality as the construction can be replicated to each connected component in the general case.

A traversal set $(\mathcal{U}, \mathcal{T})$ defines a φ-equivalence between the α-fiber of the root r and the vertex set V as follows.

Lemma 94. Path φ-Equivalence
Let $(\mathcal{U}, \mathcal{T}) \in \text{trav}(G)$. Given $v \in V$, there exists a unique $p_v \in \mathcal{T} \ltimes G$ such that $\alpha(p_v) = r$ and $\beta(p_v) = v$. Denote $\mathcal{T} \ltimes^r G = \alpha^{-1}_{\mathcal{T} \ltimes G}\{r\}$. We can do the following construction:

1. Define $\varphi : p_v \mapsto v$.
2. Define $(p_v, p_u) \mapsto p_v^u \in \mathcal{U}^0 \ltimes^r G$ such that the sequence of partial transformations of p_v^u and p_v are the same (*i.e.* $\gamma_{p_v^u}^{\mathcal{U}^0 \ltimes G} = \gamma_{p_v}^{\mathcal{U} \ltimes G}$), and the source of p_v^u is the target of p_u (*i.e.* $\alpha(p_v^u) = \beta(p_u) = u$)
3. Define the external composition $p_v p_u = p_v^u p_u \in \mathcal{U}^0 \ltimes^r G$.

Then $\varphi : \alpha^{-1}_{\mathcal{T} \ltimes G}\{r\} \to V$ is a bijective partial equivariant map.

TODO: check domain of φ and bijectivity

Proof. Bijectivity is a consequence of the spanning tree structure of $\mathcal{T}$. Equivariance because $\gamma_{p_v}(u) = \gamma_{p_v}\gamma_{p_u}(r) = \gamma_{p_v p_u}(r) = \varphi(p_v p_u)$. □

We can now define the convolution that is based on a path groupoid.

Definition 95. Path convolution
Let $(\mathcal{U}, \mathcal{T}) \in \mathrm{trav}(G)$. The *path convolution* is the partial convolution based on the path subgroupoid $\mathcal{T} \ltimes G$, which uses the groupoid partial action $\gamma := \gamma^{\mathcal{U}^0 \ltimes G}$ of the embedding groupoid zero-closure $\mathcal{U}^0 \ltimes G$.

(i) In what follows are the three expressions of the path φ-convolution for signals $s_1, s_2 \in \mathcal{S}(V)$, and $u \in V$:

$$\begin{aligned}(s *_\varphi w) &= \sum_{v \in V} s[v]\, \gamma_{p_v}(w) \\ &= \sum_{\substack{p \in \mathcal{T} \ltimes G \\ s.t.\ \alpha(p) = r}} s[\varphi(p)]\, \gamma_p(w) \\ (s *_\varphi w)[u] &= \sum_{\substack{(a,b) \in V \\ s.t.\ \gamma_{p_a}(b) = u}} s[a]\, w[b]\end{aligned}$$

(ii) The mixed formulations with $w \in \mathcal{S}(\mathcal{T} \ltimes G)$ are:

$$\begin{aligned}(w *_{\mathrm{M}} s) &= \sum_{\substack{p \in \mathcal{T} \ltimes G \\ s.t.\ \alpha(p) = r}} w[p]\, \gamma_p(s) \\ (w *_{\mathrm{M}} s)[u] &= \sum_{\substack{(p,v) \in \mathcal{T} \ltimes G \times V \\ s.t.\ \alpha(p) = r \\ s.t.\ \gamma_p(v) = u}} w[p]\, s[v]\end{aligned}$$

Remark. The role of $\mathcal{T}$ is to provide a φ-equivalence. The role of $\mathcal{U}$ is to extend every partial transformation $\gamma_g^{\mathcal{T} \ltimes G}$ to the domain of its unrestricted counterpart $\gamma_g^{\mathcal{U} \ltimes G}$.

Proposition 89 also holds for path groupoids, except that the domain-symmetric condition of 2.(i) is not needed.

Proposition 96. Characterization by equivariance to $\mathcal{U} \ltimes G$'s action Let $(\mathcal{U}, \mathcal{T}) \in \mathrm{trav}(G)$.

(i) The class of linear transformations of $\mathcal{S}(V)$ that are equivariant to the path actions of $\mathcal{U} \ltimes G$ is exactly the path φ-convolution right-operators;

(ii) in the abelian case, they are also exactly the M-convolution left-operators.

Proof. Instead of the domain-symmetric condition that was used in the proof of the converse of Proposition 89 (2.(i)), we use the fact that any vertex can be reached with an action from the root of the spanning tree of the traversal set. Indeed, given $v \in V$, as we have $\gamma_{p_v}(r) = v$, then $\gamma_{p_v}(\delta_r) = \delta_v$. Therefore, by developping a linear transformation $f(s)$ on the dirac family, and commuting f with γ_{p_v}, we obtain that $f(s) = s *_\varphi w$, where $w = f(\delta_r)$. The rest of the proof is similar to that of Proposition 89. □

Remark. Note that $\mathcal{U} \ltimes V$'s action is almost the same as the groupoid partial action of $\Upsilon = \langle \mathcal{U} \rangle$ (only "almost" because not all combinations of partial transformations might exist in the paths). However $\mathcal{U} \ltimes V$ associativy law doesn't have the limitation of Υ's.

Edge convolution operators

The counterparts of strictly edge-constrained (EC*) convolution operators for path convolutions, are indeed path convolution operators obtained with supporting set $\mathcal{N} \subset \mathcal{T} \ltimes_1 G$ which any graph can admit. By extrapolation, we can coin them *edge convolution operators*. As shown by this section, to construct one, all we need is a traversal set of partial transformations $(\mathcal{U}, \mathcal{T})$.

2.5 Conclusion

In this chapter, we constructed the convolution on graph domains.

1. We first saw that classical convolutions are in fact the class of linear transformations of the signal space that are equivariant to translations. For signals defined on graph domains, there is no natural definition of translations.

2. Therefore, we adopted a more abstract standpoint and considered in the first place any kind of transformation of the vertex set V. Hence, given a subgroup of transformation Γ, we constructed the class of linear transformations of the signal space that are equivariant to it. This provided us with an expression of a convolution based on this subgroup, and a bijective equivariant map between Γ and V, in order to transport a sum over Γ into a sum over V. We also proposed a simpler expression in the abelian case.

3. Then, we introduced the role of the edge set E, and we constrained Γ by it. This allows us to obtain a characterization of admissibility of convolutions by Cayley subgraph isomorphism, and to analyze intrinsic properties of the constructed convolution operator, namely locality and weight sharing. We also discussed operators with a smaller kernel, in particular those that are strictly edge-constrained (EC*), as they are simpler to construct.

4. Finally, we overcame the limitation that some graphs only have trivials or low degree Cayley subgraphs. In this case, we rebased our construction on groupoids of partial transformations Υ as a first iteration, but this one didn't overcome fully the above-mentioned limitation. As a last iteration, we broke down the previous construction into elementary partial actions onto the edges, recomposed into path groupoids $\mathcal{U} \ltimes G$.

Similarly, equivariance characterization and intrinsic properties hold, and the simpler (EC*) construction is also possible.

Summary of practical (EC*) convolution operators

3. For graphs that are quite regular, in the sense that they contain an above-low-degree Cayley subgraph (degree $d \geq 4$), we saw in Section 2.3.3 that all we need to construct an (EC*) convolution operator is a generating set $\mathcal{U}$ of transformations, without the need of composing its elements, and optionally (in the non-abelian case) to move a local patch $\mathcal{K}_{\mathrm{Id}}$ over the graph domain.

4. For a general graph, we saw in Section 2.4.4 that all we need to construct an (EC*) path convolution operator is a traversal set $(\mathcal{U}, \mathcal{T})$ of partial transformations, without the need to compose the paths.

In the next chapter, we will encounter examples of (EC) and (EC*) convolution operators defined on graphs, that can be expressed under group representations or under path groupoid representations.

Chapter 3
Neural networks on graph domains

Introduction

TODO:

Contents

3.1 Layer representations

Let $\mathcal{L} = (g, h)$ a neural network layer, where $g : I \to O$ is its linear part, $h : O \to O$ is its activation function, I and O are its input and output spaces, which are tensor spaces.

3.1.1 Neural interpretation of tensor spaces

Recall from Definition 1 that a tensor space has been defined such that its canonical basis is a cartesian product of canonical bases of vector spaces. Let $I = \bigotimes_{k=1}^{p} \mathbb{V}_k$ and $O = \bigotimes_{l=1}^{q} \mathbb{U}_l$. Their canonical bases are denoted $\mathbb{v}_k = (\mathbb{v}_k^1, \ldots, \mathbb{v}_k^{n_k})$ and $\mathbb{u}_l = (\mathbb{u}_l^1, \ldots, \mathbb{u}_l^{n_l})$.

Remark. Note that a tensor space is isomorph to the signal space defined over its canonical basis.

More precisely, we have the following relation.

Lemma 97. Relation between tensor and signal spaces
Let $\mathbb{V}$, $\mathbb{U}$ vector spaces, and $\mathbb{v}$, $\mathbb{u}$ their canonical bases. Let $\mathbb{T}$ a tensor space. $\otimes$ and $\times$ denote tensor and cartesian products. Then,

(i) $\mathbb{V} \cong \mathcal{S}(\mathbb{v})$

(ii) $\mathbb{V} \otimes \mathbb{U} \cong \mathcal{S}(\mathbb{v} \times \mathbb{u})$

(iii) $\mathbb{V} \otimes \mathbb{T} \cong \mathcal{S}_{\mathbb{T}}(\mathbb{v})$

where $\mathcal{S}_{\mathbb{T}}$ are signals taking values in $\mathbb{T}$ (and $\mathcal{S}$ are real-valued signals).

Proof. (i) Given $x \in \mathbb{V}$, define $\widetilde{x} \in \mathcal{S}(\mathbb{v})$ such that $\forall i, \widetilde{x}[\mathbb{v}^i] = x[i]$. The mapping $x \mapsto \widetilde{x}$ is a linear isomorphism.

(ii) $\widetilde{x}[\mathbb{v}^i, \mathbb{u}^j] = x[i, j]$

(iii) $\widetilde{x}[\mathbb{v}^i] = x[i, :, \ldots, :]$

□

Let $d \leq n_k$ and $e \leq n_l$. Define V and U as the cartesian products $V = \bigtimes_{k=1}^{d} \mathbb{v}_k$ and $U = \bigtimes_{l=1}^{e} \mathbb{u}_l$. Thanks to Lemma 97, we can identify the input and output spaces as $I = \mathcal{S}(V) \otimes \bigotimes_{k=d+1}^{p} \mathbb{V}_k$ and $O = \mathcal{S}(U) \otimes \bigotimes_{l=e+1}^{q} \mathbb{U}_l$. As $\mathcal{S}(\mathbb{v}) \otimes \mathbb{T} = \mathcal{S}_{\mathbb{T}}(\mathbb{v})$, an object of V or U can be interpreted as the representation of a *neuron* which can take multiple values.

In what follows, without loss of generality, we will make the simplification that a neuron can only take a single value (we don't consider input channels and feature maps yet). We'll thus consider that $I = \mathcal{S}(V)$ and $O = \mathcal{S}(U)$, where V is the set of *input neurons*, and U is the set of *output neurons*.

3.1.2 Propagational interpretation

Let $\mathcal{L} = (g, h)$, recall that $g : \mathcal{S}(V) \to \mathcal{S}(U)$ is characterized by a connectivity matrix W such that, $g(x) = Wx$.

Remark. Using the mapping defined in the proof of Lemma 97, for notational conveniency, we'll abusively consider x as a vector (enventually reshaped from a tensor), and W as an object of a binary tensor product for its indexing (*i.e.* $W[u, v] := W[i, j]$ where $u = \mathbb{u}^i$ and $v = \mathbb{v}^j$).

Definition 98. Propagation graph
The *propagation graph* $P = \langle (V, U), E_P \rangle$ of a layer $\mathcal{L} = (W, h)$ is the bipartite graph that has the connectivity matrix W for bipartite adjacency matrix.

An example is depicted on Figure 12.

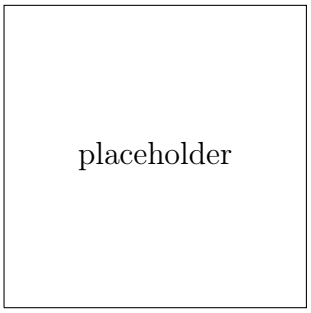


Figure 12: A propagation graph

The propagation graph defines an input topological space $\mathcal{T}_V$, and an output topological space $\mathcal{T}_U$.

Definition 99. Topological space
A *topological space* is a pair $\mathcal{T} = (X, \mathcal{O})$, where X is a set of points, $\mathcal{O}$ is a set of sets that is closed under intersection (the *open sets*), and such that every point $x \in X$ is associated with a set $\mathcal{N}_x \in \mathcal{O}$, called its *neighborhood*.

Hence, the *neural topologies* $\mathcal{T}_V$ and $\mathcal{T}_U$ are defined as

1. $\mathcal{T}_V = (V, \mathcal{O}(U))$, with $\forall v \in V, \mathcal{N}_v = \{u \in U, v \overset{P}{\sim} u\}$
2. $\mathcal{T}_U = (U, \mathcal{O}(V))$, with $\forall u \in U, \mathcal{N}_u = \{v \in V, v \overset{P}{\sim} u\}$

In particular, given an output neuron $u \in U$, a neighborhood $\mathcal{N}_u$ is also called a *receptive field*, that we denote $\mathcal{R}_u$.

3.1.3 Graph representation of the input space

Let's consider that the input neurons V have a (possibly edge-less) graph structure $G = \langle V, E \rangle$. We define an edge-constrained layer as follows.

Definition 100. Edge-constrained layer
A layer $\mathcal{L} : G = \langle V, E \rangle \to U$, is said to be *edge-constrained* (EC) if:

1. There is a one-to-one correspondence $\pi : V_\pi \to U$, where $V_\pi \subset V$.

2. Given an output neuron u, an input neuron v is in its receptive field, if and only if, v and the π-fiber of u are connected in G,
$$\textit{i.e. } \forall u \in U, v \in \mathcal{R}_u \Leftrightarrow v \overset{E}{\sim} \pi^{-1}(u)$$

Note that (EC) convolutions are (EC) layers. We have the following characterization.

Proposition 101. (EC) Characterization with receptive fields
Let a layer $\mathcal{L} : V \to U$, $V_\pi \subset V$, and a one-to-one correspondence $\pi : V_\pi \to U$. There exists a graph $G = \langle V, E \rangle$ for which $\mathcal{L}$ is (EC), if and only if, the receptive fields are *intertwined* (*i.e.* $\forall a, b \in V_\pi, a \in \mathcal{R}_{\pi(b)} \Leftrightarrow b \in \mathcal{R}_{\pi(a)}$).

Proof. ($\Rightarrow$) Thanks to $a \in \mathcal{R}_{\pi(b)} \Leftrightarrow a \overset{E}{\sim} b \Leftrightarrow b \in \mathcal{R}_{\pi(a)}$

($\Leftarrow$) If the receptive fields are intertwined, then the relation defined as $a \sim b \Leftrightarrow a \in \mathcal{R}_{\pi(b)}$ is symmetric, and thus can define an edge set.

□

Therefore, any layer that has its receptive fields intertwined, admits an *underlying* graph structure. For example, a 2-d convolution operator can be rewritten as an (EC*) convolution on a lattice graph, and as an (EC) convolution on a grid graph.
Figure 13 depicts an underlying graph and its corresponding propagation graph.

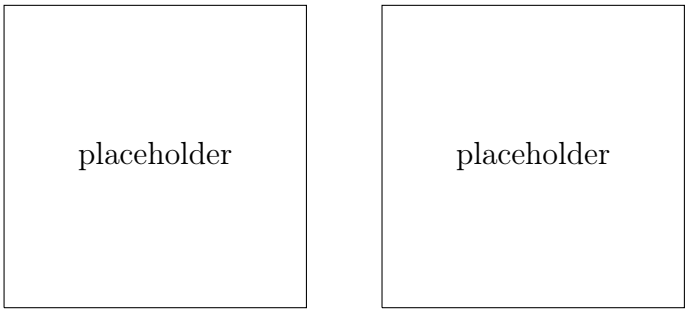


Figure 13: Underlying graph Vs Prop graph

3.1.4 General representation with weight sharing

Weight sharing refers to the fact that some parameters of the connectivity matrix W are equal, and stay equal after each learning iteration. In other words they are tied together. From a propagational point of view, this amounts to label the edges of the propagation graph P with weights, where weights can be used multiple times to label these edges. Supposed W is of shape $m \times n$ and there are ω weights in the kernel used to label the edges. Given a input neuron i and an output neuron j, the edge labelling can be expressed as:

$$W[j, i] = \theta[h] = \theta^T a \tag{28}$$

where θ, the weight kernel, is a vector of size ω and a is a one-hot vector (full of 0s except for one 1) of size ω with a 1 at the index h corresponding to the weight that labels the edge $i \sim j$; or a is the *zero* vector in case $i \nsim j$.

This equation (28) can be rewritten as a tensor contraction under Einstein summation convention, by noticing that a depends on i, j and by defining a tensor S such that $a = S[:, i, j]$, as follows:

$$W_{ij} = \theta_k S^k{}_{ij} \tag{29}$$

Therefore, the linear part of $\mathcal{L}$ can be rewritten as:

$$g(x)_j = \theta_k S^k{}_j{}^i x_i \tag{30}$$

If we consider that the layer $\mathcal{L}$ is duplicated with input channels and feature maps, then θ, x and $g(x)$ become tensors, denoted Θ, X and $g(X)$. Usually, for stochastic gradient descent, X and $g(X)$ are also expanded with a further rank corresponding to the batch size and we obtain:

$$g(X) = \Theta S X \text{ where } \begin{cases} W_{pq}{}^{ij} = \Theta_{pq}{}^k S_k{}^{ij} \\ g(X)_{jq}{}^b = W_{jq}{}^{ip} X_{ip}{}^b \end{cases} \tag{31}$$

index	size	description
i	n	input neuron
j	m	output neuron
p	N	input channel
q	M	feature map
k	ω	kernel weight
b	B	batch instance

Table 1: Table of indices

Remark. Note that the expression $\Theta S X$ is written regardless of the ordering of the tensors ranks and is defined by index juggling.

Also, note that it is associative and commutative. This can be seen by the index symmetry of (32), which rewrites (31), and where the sum symbols Σ and scalar values commute:

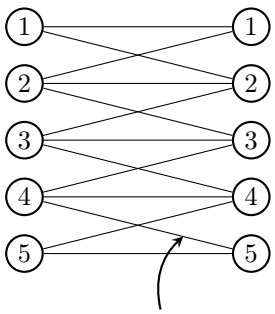
$$\Theta S X[j,q,b] = \sum_{k=1}^{\omega} \sum_{p=1}^{P} \sum_{i=1}^{n} \Theta[k,p,q]\, S[k,i,j]\, X[i,p,b] \tag{32}$$

We call S the *weight sharing scheme* (Vialatte et al., 2017) of the layer $\mathcal{L}$.

Definition 102. Ternary representation
The *ternary representation* of a layer $\mathcal{L} : X \mapsto Y$, with activation function h, is the equation $Y = h(\Theta S X)$, as defined in (31), where Θ is the *weight kernel*, and S is called the *weight sharing scheme*.

Remark. In (28), we defined $a = S[:, i, j]$ as a one-hot vector when $i \sim j$, as its role is to select a weight in $\theta = \Theta[p, q, :]$. However, a can also do this selection *linearly*, so in fact it is not necessarily a one-hot vector.

Figure 14 depicts an example of how the equation (31) labels the edges of P.



$$W[p, q, i = 4, j = 5] = \sum_{k=1}^{\omega} \Theta[p, q, k] S[k, 4, 5]$$

Figure 14: Example of a propagation graph P for a given input channel p and feature map q. The edge $4 \sim 5$ is labelled with a linear combination of kernel weights from $\Theta[p, q, :]$. In the usual case, $S[k, 4, 5]$ is a one-hot vector that selects a single kernel weight: $\exists h, W[p, q, 4, 5] = \Theta[p, q, h]$.

The ternary representation uncouples the roles of Θ and S in W, and is the most general way of representing any kind of partially connected layer with weight sharing.

3.2 Study of the ternary representation

In this section, we study the ternary representation, which is the general representation with weight sharing we obtained above.

3.2.1 Genericity

The ternary representation can represent any kind of layer. For example,

- To obtain a fully connected layer, one can choose ω to be of size nm and S the matrix of vectors that contains all possible one-hot vectors.

- To obtain a convolutional layer, one can choose ω to be the size of the kernel. S would contain one-hot vectors. A stride > 1 can be obtained by removing the corresponding dimensions. If the convolution is a classical convolution, or is supported by a Cayley subgraph (see Chapter 2), then S would be circulant along the input neurons rank in the canonical basis.

- Any partially connected layer with (or without) weight sharing can be obtained with appropriate construction of S.

3.2.2 Efficient implementation under sparse priors

What is the fastest way to compute ΘSX ?
As the equation (31) is associative and commutative, there are three ways to start to calculate it: with ΘS, SX, or ΘX, which we will call *middle-stage tensors*. The computation of a middle-stage tensor is the bottleneck to compute (31) as it imposes to compute significantly more entries than for the second tensor contraction. In Table 2, we compare their shapes. We refer the reader to Table 1 for the denomination of the indices.

tensor	shape
Θ	$\omega \times P \times Q$
S	$\omega \times n \times m$
X	$n \times P \times B$
ΘS	$n \times m \times P \times Q$
SX	$\omega \times m \times P \times B$
ΘX	$\omega \times n \times Q \times B$
ΘSX	$m \times Q \times B$

Table 2: Table of shapes

In usual settings, we want to have $\omega \ll n$ and $\omega \ll m$, which means that we have weight kernels of small sizes (for example in the case of images, convolutional kernel are of size significantly smaller than that of the images). Also, the number of input channels P and of feature maps Q are roughly in the same order, with $P < Q$ more often than the contrary. So in practice, the size of ΘS is significantly bigger than the size of SX and of ΘX, and the size of SX is usually the smallest.

How to exploit S sparsity ?

Also, in usual settings, S is sparse as $S[:, i, j]$ are one-hot vectors. So computing SX should be faster than computing ΘX, provided we exploit the sparsity. Although S is very sparse as it contains at most a fraction $\frac{1}{w}$-th of non-zero values, it is only sparse along the first rank, which makes implementation with sparse classes of common deep learning libraries not optimized. However, we can benefit from two other sparse priors: the local receptive fields (LRF) are usualy of the same size (or at least of around the same order), and each weight is associated to one *and only one* neuron of each LRF.

So we proceed differently. The idea is to use a non-sparse tensor X_{LRF} that has a rank that indexes the LRF, and another rank that indexes elements of these LRF, in order to lower the computation to a dense matrix multiplication (or a dense tensor contraction) which is already well optimized. This approach

is similar to that proposed in Chellapilla et al., 2006, which is also exploited in the cudnn primitives (Chetlur et al., 2014) to efficiently implement the classical convolution.

The LRF representation

In our case, it turns out that X_{LRF} can be exactly SX, as given fixed b, p, and j, $SX[:, j, p, b]$ corresponds to entries of the input signal $X[:, p, b]$ restrained to a LRF $\mathcal{R}_j$ of size ω. Therefore,

$$\exists\, \text{LRF}_j = [i_1, \ldots, i_\omega] \ s.t. \ SX[:, j, p, b] = X[\text{LRF}_j, p, b] \tag{33}$$

The elements of LRF_j can be found by doing a lookup in the one-hot vectors of S, provided each kernel weight occurs exactly once in each LRF. We have:

$$R_j[k] = i_k \ s.t. \ S[:, i_k, j][k] = 1 \tag{34}$$

This lookup needs not be computed each time and can be done beforehand. Finally, if we define $\text{LRF} = [\text{LRF}_1, \ldots, \text{LRF}_m]$, (33) gives:

$$SX = X[\text{LRF}, :, :] \tag{35}$$

The equation (35) is computed with only $\omega \times m$ assignations and can be simply implemented with automatic differentiation in commonly used deep learning libraries.

Benchmarks

To support our theoretical analysis, we benchmark three methods for computing the tensor contraction SX:

- naively using dense multiplication,
- using sparse classes of deep learning libraries,

- using the LRF based method we described above.

We run the benchmarks under the assumptions that $S[:, i, j]$ are one-hot vectors, and that a weight occur exactly once in each LRF (as it is the case for convolutions supported by a Cayley subgraph). For each method, we make 100 runs of computations of SX, with S and X being randomly generated according to the assumptions. In Table 3, we report the mean time and standard deviation. The values of the hyperparameters were each time $n = m = N = M = B = 100$, and $\omega = 10$. The computations were done on graphical processing units (GPU).

Method	Time
Naive	$todo\mu s \pm todo$
Sparse	$todo\mu s \pm todo$
LRF	$\boldsymbol{todo}\mu s \pm \boldsymbol{todo}$

Table 3: Benchmark results

As expected, the LRF method is faster.

3.2.3 Influence of symmetries

In the case of images, or other signals over a grid, the grid structure of the domain defines the weight sharing scheme S of the convolution operation. For example, for a layer $\mathcal{L} : X \mapsto Y = h(\Theta SX)$, and given fixed b, p, q, the classical convolution can be rewritten as

$$y[j] = h\left(\sum_{k=1}^{\omega} \theta[k] \sum_{i=1}^{n} S[k, i, j]\, x[i]\right) \tag{36}$$

$$= h\left(\sum_{k=1}^{\omega} \theta[k]\, x_{\text{LRF}}[k, j]\right) \tag{37}$$

Where $x_{\text{LRF}}[k, j]$ can be obtained by matching (37) with the expression given by the definition (see Definition 48). So, $S[k, :, j]$ is a one-hot vector that

specifies which input neuron in the LRF of y is associated with the k-th kernel weight, and the index of the 1 is determined by $x_{\text{LRF}}[k, j]$.
That is to say, in the ternary representation, W captures the information the layer has learnt, whereas S captures how the layer exploits the symmetries of the input domain.

Visual construction

Visually, constructing S amounts to move a rectangular grid over the pixel domain, as depicted on Figure 15 where each point represents the center of a pixel, and the moving rectangle represents the LRF of its center j. Each of its squares represents a kernel weight $\theta[k]$ which are associated with the pixel $x_{\text{LRF}}[k, j]$ that falls in it.

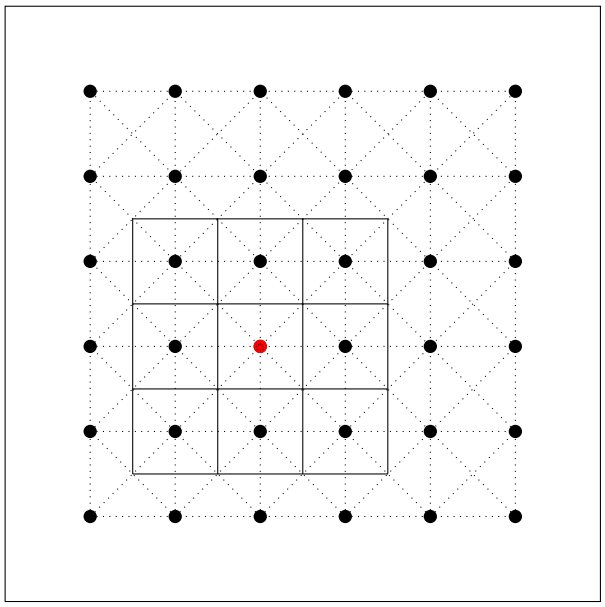


Figure 15: Weight assignement of convolutions on pixel domains

The case of images is very regular, in the sense that every pixels are regularly spaced out, so that obtained S is circulant along its last two ranks. This is a consequence of the translational symmetries of the input domain, which underly the definition of the convolution, as seen in Chapter 2.

- What happens if we loose these symmetries?

To answer this question, we make the following experiment (Vialatte et al., 2016):

1. We distort the domain by moving the pixels randomly. The radial displacement is uniformly random with the angle, and its radius follows a gaussian distribution $\mathcal{N}(0, \sigma)$.

2. Then we compare performances of shallow CNNs expressed under the ternary representation, for which S is constructed similarly than with the above visual construction, for different values of σ.

The visual construction of S on distorded domain is depicted by Figure 16.

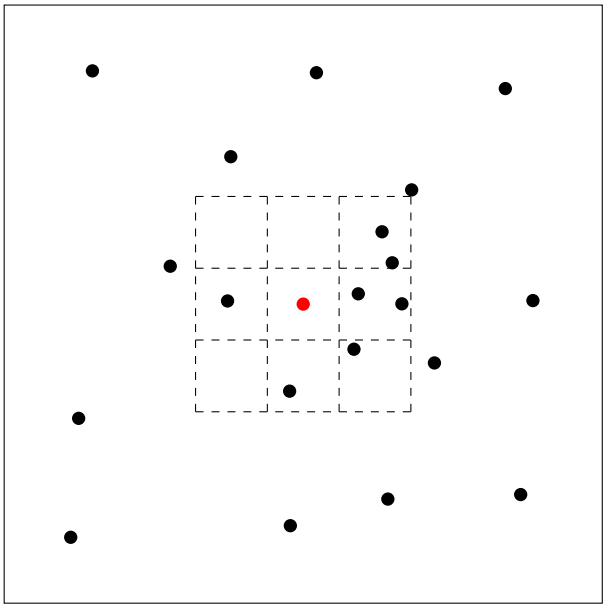


Figure 16: Weight assignement in generalized convolution on distorded domains

We run a classification task with standard hyperparameters on a toy dataset (we used MNIST, LeCun et al., 1998). The results are reported in Figure 17

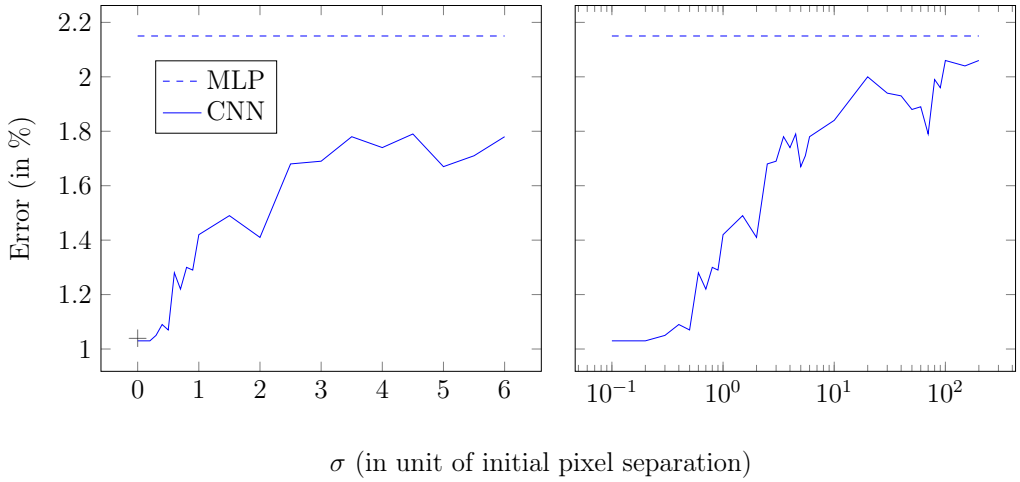


Figure 17: Error in function of the standard deviation σ, for generalized CNNs and an MLP, each with 500 weights.

The bigger is σ, the less accurate are the symmetries of the input domain, up to a point where the ternary representation becomes almost equivalent to a dense layer. The results illustrate nicely this evolution, and stress out the importance of trying to leverage symmetries when defining new convolutions. Moreover, they indicate that the ternary representation also allows to improve performances compared to using dense layers, providing we are able to prototype a relevant weight sharing scheme S to exploit symmetries.

3.2.4 Semi-supervised node classification

TODO: transpose, w=1

3.3 Learning the weight sharing scheme

3.3.1 Discussion

In the ternary representation $Y = h(\Theta SX)$ of a layer $\mathcal{L}$, the weight kernel Θ is usually the only operand that is learned, and the role of S is to label the edges of the propagation graph P with these weights. Recall that, as noted after Definition 102, S needs not be sparse and composed of one-hot vectors. In that case, the labelling is done linearly as depicted previously in Figure 14. Therefore, the weight sharing scheme S can also be updated during the learning phase. This can be interpreted as learning a convolution-like operator on the underlying graph G (Vialatte et al., 2017).

Remark. When S has no sparse priors, we must then not have more parameters in S and Θ than in W, so that the weight sharing still makes sense. If we call l the number of edges in the resulting propagation graph P, then the former assumption requires $l\omega + \omega NM \leq lNM$ or equivalently $\frac{1}{\omega} \geq \frac{1}{NM} + \frac{1}{l}$. It implies that the number of weights per filter ω must be lower than the total number of filters NM and than the number of edges l, which is always the case in practice.

However, as learning S will require in most cases to lose the sparse priors, this can lead to memory issues and increased time execution. Hence, our first experiments focus on shallow architectures to study feasability. Despite these limitations, the observations could still be useful to understand better the extent of the ternary representation. Nonetheless, these experiments can also see application in deep settings: an approach would be to pick a weight sharing scheme S learned in shallow settings and to share it among ternary representations of convolutional layers of a deep architecture.
TODO: drop conditional if experiment is done

3.3.2 Training settings

We learn S and Θ simultaneously. We also use a fine-tuning step, which consists in freezing S in the last epochs.

Constraints

Because of our inspiration from CNNs, we propose constraints on the parameters of S. Namely, we impose them to be between 0 and 1, and to sum to 1 along the rank that choose a weight label.

$$\forall(k,i,j),\ S[k,i,j] \in [0,1] \tag{38}$$

$$\forall(i,j),\ \sum_{k=1}^{\omega} S[k,i,j] = 1 \tag{39}$$

Therefore, the vectors on the third rank of S can be interpreted as performing a positive weighted average of the parameters in Θ.
We also imposes the layers to be edge-constrained. Hence, their input space admits an underlying graph G, whether exposed or not. This implies that if we call A its adjacency matrix, then we must have

$$A[i,j] = 0 \Rightarrow \forall k, S[k,i,j] = 0 \tag{40}$$

In other terms, we say that S is *masked by* A.

Initialization

We introduce three types of initialization for S. The first two have the sparse priors mentioned earlier:

1. uniform random: parameters of S are simply initialized with a uniform random distribution with limits as described by Glorot and Bengio, 2010.

2. random one-hot: one-hot vectors are distributed randomly on the $S[:, i, j]$, with the constraint that for each LRF, a particular one-hot vector can only be distributed at most once more than any other.

3. circulant one-hot: one-hot vectors are distributed in a circulant fashion on the $S[:, i, j]$, so that on euclidean domains, the initial state of S correspond exactly to the weight sharing scheme of a standard convolution.

3.3.3 Experiments with grid graphs

We experiment (EC) layers on the MNIST dataset (LeCun et al., 1998). We use a shallow architecture made of a single ternary layer with 50 feature maps, without pooling, followed by a fully connected layer of 300 neurons, and terminated by a softmax layer of 10 neurons. Rectified Linear Units Glorot et al., 2011 are used for the activations and a dropout Srivastava et al., 2014 of 0.5 is applied on the fully-connected layer. Input layers are regularized by a factor weight of 10^{-5} Ng, 2004. We optimize with ADAM Kingma and Ba, 2014 up to 100 epochs and fine-tune (while S is frozen) for up to 50 additional epochs.

For the underlying graph strucure, we consider a grid graph that connects each pixel to itself and its 4 nearest neighbors (or less on the borders). We also consider the square of this graph (pixels are connected to their 13 nearest neighbors, including themselves), the cube of this graph (25 nearest neighbors), up to 10 powers (211 nearest neighbors). We test the model under two setups: either the ordering of the node is unknown, and then we use random one-hot intialization for S; either an ordering of the node is known, and then we use circulant one-hot for S which is freezed in this state. We use the number of nearest neighbors as for the dimension of the first rank of S. We also compare with a convolutional layer of size 5x5, thus containing as many weights as the cube of the grid graph. On MNIST, training architectures that

learn S took about twice longer. Table 4 summarizes the obtained results. The ordering is unknown for the first result given, and known for the second result between parenthesis.

Conv5x5	Grid^1	Grid^2	Grid^3
(0.87%)	1.24% (1.21%)	1.02% (0.91%)	0.93% (0.91%)
Grid^4	Grid^5	Grid^6	Grid^{10}
0.90% (0.87%)	0.93% (0.80%)	1.00% (0.74%)	0.93% (0.84%)

Table 4: Error rates on powers of the grid graphs on MNIST.

We observe that even without knowledge of the underlying euclidean structure, grid (EC) layers obtain comparable performances as convolutional ones, and when the ordering is known, they match convolutions. We also noticed that after training, even though the one-hot vectors used for initialization had changed to floating point values, their most significant dimension was almost always the same. That suggests there is room to improve the initialization and the optimization.

In Figure 18, we plot the test error rate for various normalizations when using the square of the grid graph, as a function of the number of epochs of training, only to find that they have little influence on the performance but sometimes improve it a bit. Thus, we will treat them as optional hyperparameters.

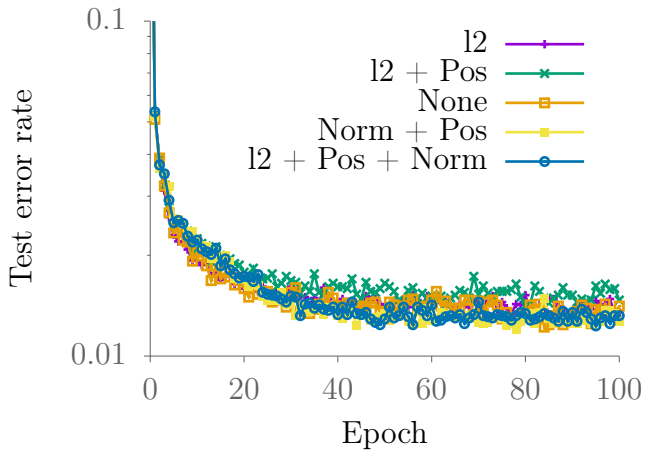


Figure 18: Evolution of the test error rate when learning MNIST using the square of a grid graph and for various normalizations, as a function of the epoch of training. The legend reads: "l2" means ℓ_2 normalization of weights is used (with weights 10^{-5}), "Pos" means parameters in S are forced to being positive, and "Norm" means that the ℓ_1 norm of each vector in the third dimension of S is forced to 1.

3.3.4 Experiments with covariance graphs

As underlying graph structure, we use a thresholded covariance matrix obtained by using all the training examples. We choose the threshold so that the number of remaining edges corresponds to a certain density p (5x5 convolutions correspond approximately to a density of $p = 3\%$). We also infer a graph based on the k nearest neighbors of the inverse of the values of this covariance matrix (k-NN). The latter two are using no prior about the signal underlying structure. The pixels of the input images are shuffled and the same re-ordering of the pixels is used for every image. Dimension of the first rank of S is chosen equal to k and its weights are initialized random uniformly Glorot and Bengio, 2010. The receptive graph layers are also compared with models obtained when replacing the first layer by a fully connected or convolutional one. Architecture used is the same as in the previous

section. Results are reported on table 5.

MLP	Conv5x5	Thresholded ($p = 3\%$)	k-NN ($k = 25$)
1.44%	1.39%	1.06%	0.96%

Table 5: Error rates when topology is unknown on scrambled MNIST.

We observe that the receptive graph layers outperforms the CNN and the MLP on scrambled MNIST. This is remarkable because that suggests it has been able to exploit information about the underlying structure thanks to its underlying graph.

3.3.5 Improving S for standard convolutions

On CIFAR-10, a dataset of tiny images (Krizhevsky, 2009), we made experiments on shallow CNN architectures and replaced convolutions by receptive graphs. We report results on a variant of AlexNet Krizhevsky et al., 2012 using little distortion on the input that we borrowed from a tutorial of tensorflow Abadi et al., 2015. It is composed of two 5x5 convolutional layers of 64 feature maps, with max pooling and local response normalization, followed by two fully connected layers of 384 and 192 neurons. On CIFAR-10, training architectures that learn S took about 2.5 times longer. We compare two different graph supports: the one obtained by using the underlying graph of a regular 5x5 convolution, and the support of the square of the grid graph. Optimization is done with stochastic gradient descent on 375 epochs where S is freezed on the 125 last ones. Circulant one-hot intialization is used. These are weak classifiers for CIFAR-10 but they are enough to analyse the usefulness of the proposed layer. Experiments are run five times each. Means and standard deviations of accuracies are reported in table 6. "Pos" means parameters in S are forced to being positive, "Norm" means that the ℓ_1 norm

of each vector in the third dimension of S is forced to 1, "Both" means both constraints are applied, and "None" means none are used.

Support	Learn S	None	Pos	Norm	Both
Conv5x5	No	/	/	/	86.8 ± 0.2
Conv5x5	Yes	87.4 ± 0.1	87.1 ± 0.2	87.1 ± 0.2	87.2 ± 0.3
Grid^2	Yes	87.3 ± 0.2	87.3 ± 0.1	87.5 ± 0.1	87.4 ± 0.1

Table 6: Accuracies (in %) of shallow networks on CIFAR-10.

The (EC) layers are able to outperform the corresponding CNNs by a small amount in the tested configurations, opening the way for more complex architectures.

TODO: take S, and reuse it on resnet

TODO: shallow on Pines

TODO: incorporate MCnets results (see github) on 20news

TODO: learning A on kipf

3.4 Translation-convolutional neural networks

3.4.1 Methodology

As we saw in Section 3.2.3 (Figure 15), the classical convolution on grid graphs can be obtained visually by translating a rectangular window over the pixel domain. The idea of this section is to define similarly a convolution on graph domains, using a notion of translation defined on graphs (Pasdeloup et al., 2017b). The translations we use match euclidean translations on 2D grid graphs (Grelier et al., 2016), and extend them on general ones, by preserving three simple key properties: injectivity, edge-constraint, and neighborhood-preservation, which we will detail later.

Given a set of graph translations $T = \{t_i, i \in \{1, 2, \dots, \kappa - 1\}\} \cup \{t_0 = \mathrm{Id}\}$, the corresponding convolution can be expressed using the same formalism we used in Chapter 2, as:

$$w * x = \sum_{t \in T} w[t]\, t(x) \tag{41}$$

where x is a signal over vertices of a graph $G = \langle V, E \rangle$, and w is a signal defined on $T \subset \Phi_{\mathrm{EC}}(G)$. Depending on the algebraic nature of T (that we will discuss later), the theoretical results obtained in Chapter 2 hold.

By denoting $w_i = w[t_i]$, we obtain the more familiar expression:

$$\forall u \in V, w * x[u] = \sum_{i=0}^{\kappa-1} w_i\, x[t_i^{-1}(u)] \tag{42}$$

This expression extends the definition of convolutions from grid graphs to general graphs, with the use of graph translations. We use it to obtain extended CNNs that can be applied on graphs. In this manuscript, we coin the term *Translation-convolutional neural network* (TCNN) to refer to them, but they are the same as the extended convolutions we defined in Pasdeloup

et al., 2017a.
Besides defining translation-convolution on general graphs, designing a TCNN also implies extending the other building blocks of classical CNNs. As a counterpart for pooling operations, we define a graph subsampling and apply strided translation-convolution. The translations of the subsampled graph are derived from those of the base graph to define translation-convolution at the downscaled level. We will also use a weak form of data augmentation using these translations. Figure 19 depicts the proposed methodology (Lassance et al., 2018).

3.4.2 Background

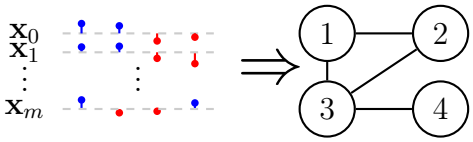
Let a graph $G = \langle V, E \rangle$. We suppose the graph is connected, as conversely the process can be applied to each connected component of G. We denote by d the max degree of the graph and $n = |V|$ the number of vertices.

Definition 103. Candidate-translation
A *candidate-translation* is a function $\phi : U \to V$, where $U \subset V$ and such that:

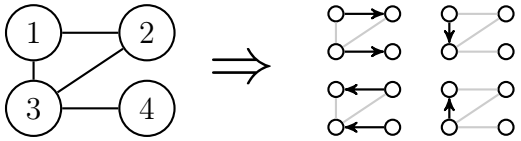
- ϕ is *injective*:
  $\forall v, v' \in U, \phi(v) = \phi(v') \Rightarrow v = v'$,
- ϕ is *edge-constrained*:
  $\forall v \in U, (v, \phi(v)) \in E$,
- ϕ is *strongly neighborhood-preserving*:
  $\forall v, v' \in U, (v, v') \in E \Leftrightarrow (\phi(v), \phi(v')) \in E$.

The cardinal $|V - U|$ is called the *loss* of ϕ. A translation for which $V = U$ is called a *lossless* translation. Two candidate-translations ϕ and ϕ' are said to be *aligned* if $\exists v \in U, \phi(v) = \phi'(v)$. We define $N_r(v)$ as the set of vertices that are at most r-hop away from a vertex $v \in V$.

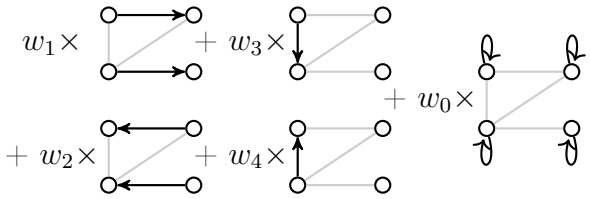
Step 0 (optional): infer a graph (see Pasdeloup et al., 2017a)



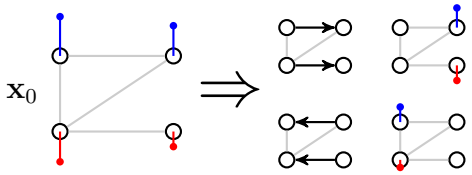
Step 1: infer translations



Step 2: design convolution weight-sharing



Step 3: design data augmentation



Step 4: design subsampling and convolution weight-sharing

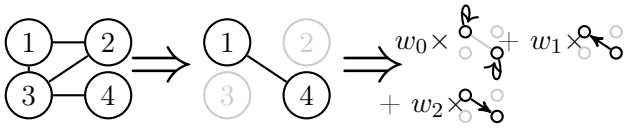


Figure 19: Outline of the proposed method

Definition 104. Translation
A *translation* in a graph G is a candidate-translation such that there is no aligned translation with a strictly smaller loss, or is the identity function.

Because translations and candidate-translations need not be surjective on V, we introduce a zero vertex, denoted $\perp$, and sometimes called *black hole* (Grelier et al., 2016), such that any candidate translation $\phi : U \to V$ is extended as a function $\phi : V \to V \cup \{\perp\}$

$$\forall v \notin U, \phi(v) = \perp \tag{43}$$

Finding translations is an NP-complete problem (Pasdeloup et al., 2017b). So in practice, we will first search locally. For this reason, we define local translations:

Definition 105. Local translation
A *local translation* of center $v \in V$ is a translation in the subgraph of G induced by $N_2(v)$, that has v in its definition domain.

With the help of local translations, we can construct proxies to global translations.

Definition 106. Proxy-translations
A family of *proxy-translations* $(\psi_p)_{p=0,..\kappa-1}$ initialized by $v_0 \in V$ is defined algorithmically as follows:

1. We place an indexing kernel on $N_1(v_0)$ i.e.
   $N_1(v_0) = \{v_0, v_1, ..., v_{\kappa-1}\}$ with $\forall p, \psi_p(v_0) = v_p$,
2. We move this kernel using each local translation ϕ of center v_0:
   $\forall p, \psi_p(\phi(v_0)) = \phi(v_p)$,
3. We repeat 2) from each new center reached until saturation. If a center is being reached again, we keep the indexing that minimizes the sum of losses of the local translations that has lead to it.

We explain this algorithm in more details in the next Section 3.4.4. A family of proxy-translations defines a translation-convolution as follows:

Definition 107. Translation-convolution layer
Let $(\psi_p)_{p=0,..,\kappa-1}$ be a family of proxy-translations identified on G *s.t.* $\psi_0 = \mathrm{Id}$. The *translation-convolution layer* $\mathcal{L} : x \mapsto y$ is defined as:

$$\forall v \in V, y[v] = h\left(\sum_{p=0}^{\kappa-1} w_p \, x[\psi_p(v)] + b\right) \tag{44}$$

where h is the activation function, b is the bias term, and with the convention that $x[\perp] = 0$.

3.4.3 Connection with action convolutions

TODO: See chapter 2

- lossless translations = group action convolution
- translations = groupoid action convolution
- proxy translations = path convolution

TODO: SNP == abelian?

TODO: state the properties obtained from results of chapter 2

3.4.4 Finding proxy-translations

We describe in three steps how we efficiently find proxy-translations.

First step: finding local translations
For each vertex $v \in G$, we identify all local translations using a bruteforce algorithm. This process requires finding all translations in all induced subgraphs. There are n such subgraphs, each one contains at most d local translations. Finding a translation can be performed by looking at all possible

injections from 1-hop vertices around the central vertex to any vertex that is at most 2-hops away. We conclude that it requires at most $\mathcal{O}(ndd^{2(d+1)})$ elementary operations and is thus linear with the order of the graph. On the other hand, it suggests that sparsity of the graph is a key criterion in order to maintain the complexity reasonable.

Figure 20 depicts an example of a grid graph and the induced subgraph around vertex v_0. Figure 21 depicts all obtained translations in the induced subgraph.

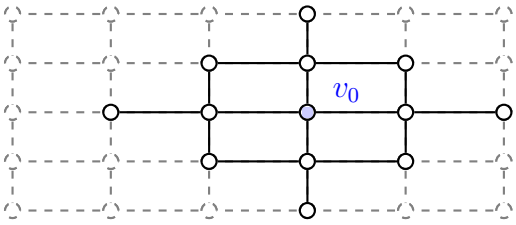


Figure 20: Grid graph (in dashed grey) and the subgraph induced by $N_2(v_0)$ (in black).

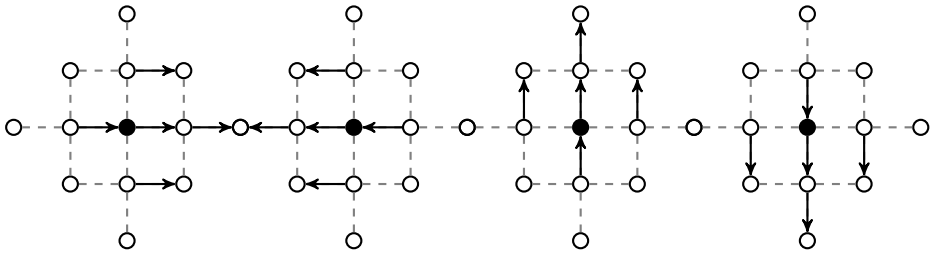


Figure 21: Translations (black arrows) in the induced subgraph (dashed grey) around v_0 (filled in black) that contains v_0 and only some of its neighbors.

Second step: move a small localized kernel

Given an arbitrary[1] vertex $v_0 \in V$, we place an indexing kernel on $N_1(v_0)$ i.e. $N_1(v_0) = \{v_0, v_1, ..., v_{\kappa-1}\}$. Then we move it using every local translations of center v_0, repeating this process for each center that is reached for the first time. We stop when the kernel has been moved everywhere in the graph. In case of multiple paths leading to the same destination, we keep the indexing that minimizes the sum of loss of the series of local translations. We henceforth obtain an indexing of at most κ objects of $N_1(v)$ for every $v \in V$.

This process is depicted in Figure 22. Since it requires moving the kernel everywhere, its complexity is $\mathcal{O}(nd^2)$.

Final step: identifying proxy-translations

Finally, by looking at the indexings obtained in the previous step, we obtain a family of proxy-translations defined globally on G. More precisely, each index defines its own proxy-translation. Note that they are not translations because only the local properties have been propagated through the second step, so there can exist aligned candidates with smaller losses. Because of the constraint to keep the paths with the minimum sum of losses, they are good proxies to translations on G.

An illustration on a grid graph is given in Figure 23. The complexity is $\mathcal{O}(nd)$. Overall, all three steps are linear in n.

[1] In practice we run several experiments while changing the initial vertex and keep the best obtained result.

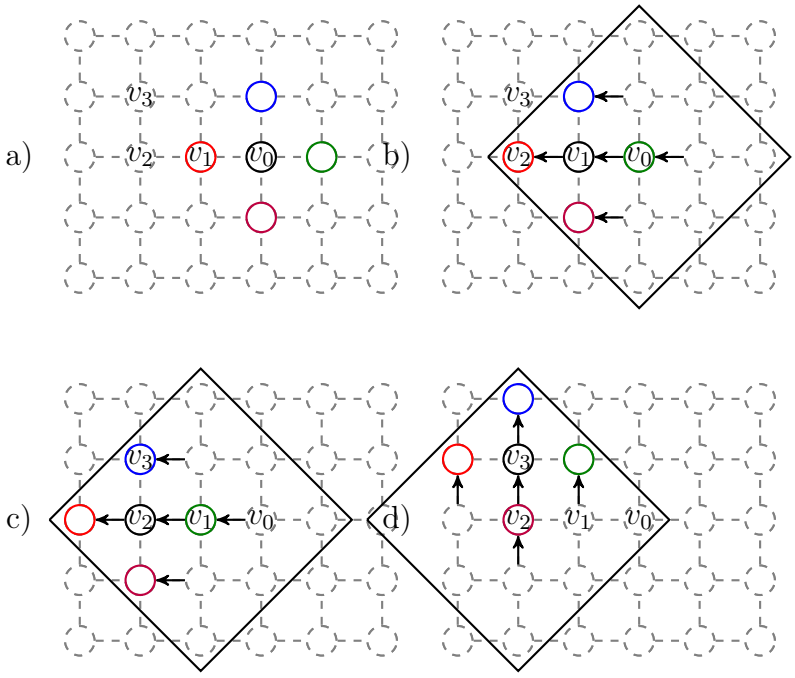


Figure 22: Illustration of the translation of a small indexing kernel using translations in each induced subgraph. Kernel is initialized around v_0 (a), then moved left around v_1 (b) using the induced subgraph around v_0, then moved left again around v_2 (c) using the induced subgraph around v_1 then moved up around v_3 (d) using the induced subgraph around v_2. At the end of the process, the kernel has been localized around each vertex in the graph.

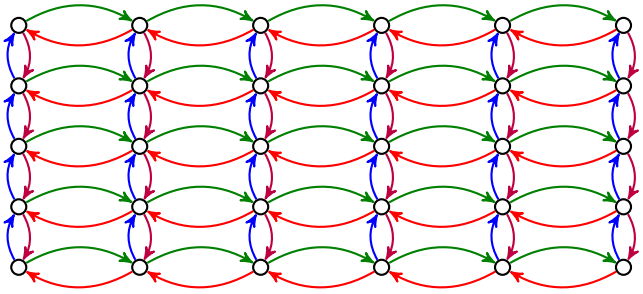


Figure 23: Proxy-translations in G obtained after moving the small kernel around each vertex. Each color corresponds to one translation.

3.4.5 Subsampling

Downscaling is a tricky part of the process because it supposes one can somehow regularly sample vectors. As a matter of fact, a nonregular sampling is likely to produce a highly irregular downscaled graph, on which looking for translations irremediably leads to poor accuracy, as we noticed in our experiments.
We rather define the translations of the strided graph using the previously found proxy-translations on G.

First step: extended convolution with stride r
Given an arbitrary initial vertex $v_0 \in V$, the set of kept vertices $V_{\downarrow r}$ is defined inductively as follows:

- $V_{\downarrow r}^0 = \{v_0\}$,
- $\forall t \in \mathbb{N}, V_{\downarrow r}^{t+1} = V_{\downarrow r}^t \cup \{v \in V, \forall v' \in V_{\downarrow}^t, v \notin N_{r-1}(v') \wedge \exists v' \in V_{\downarrow r}^t, v \in N_r(v')\}$.

This sequence is nondecreasing and bounded by V, so it eventually becomes stationary and we obtain $V_{\downarrow r} = \lim_t V_{\downarrow r}^t$. Figure 24 illustrate the first downscaling $V_{\downarrow 2}$ on a grid graph.
The output neurons of the extended convolution layer with stride r are $V_{\downarrow r}$.

Second step: convolutions for the strided graph
Using the proxy-translations on G, we move a localized r-hop indexing kernel over G. At each location, we associate the vertices of $V_{\downarrow r}$ with indices of the kernel, thus obtaining what we define as induced $_{\downarrow r}$-translations on the set $V_{\downarrow r}$. In other words, when the kernel is centered on v_0, if $v_1 \in V_{\downarrow r}$ is associated with the index p_0, we obtain $\phi_{p_0}^{\downarrow r}(v_0) = v_1$. Subsequent convolutions at lower scales are defined using these induced $_{\downarrow r}$-translations.

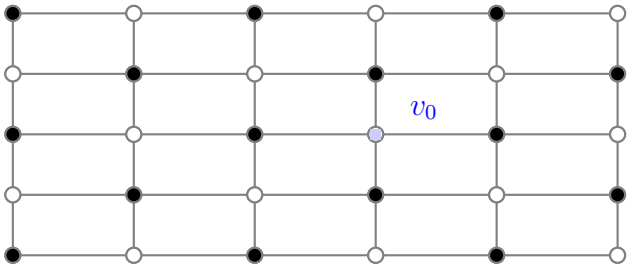


Figure 24: Downscaling of the grid graph. Disregarded vertices are filled in.

3.4.6 Data augmentation

Once proxy-translations are obtained on G, we use them to move training singals, artificially creating new ones. Note that this type of data-augmentation is poorer than for images since no flipping, scaling or rotations are used.

3.4.7 Experiments

Matching CNNs on an image classification task

On the CIFAR-10 dataset (Krizhevsky, 2009), our models are based on a variant of a deep residual network, namely PreActResNet18 (He et al., 2016b). We tested different combinations of graph support and data augmentation. For the graph support, we use either a regular 2D grid or either an inferred graph obtained by keeping the four neighbours that covary the most. In the second case, no structure prior have been fed to the process, so the results can be compared with those of Lin et al., 2015. We also report results obtained with ChebNet (Defferrard et al., 2016), where only convolutional layers have been replaced for comparison.

Table 7 summarizes our results. In particular, it is interesting to note that results obtained without any structure prior (91.07%) are only 2.7% away from the baseline using classical CNNs on images (93.80%). This gap is even smaller (less than 1%) when using the grid prior. Also, without priors our method significantly outperforms the others.

Support	MLP	CNN	Grid Graph		Covariance Graph
			ChebNet[c]	Proposed	Proposed
Full Data Augmentation	78.62%[a,b]	**93.80%**	85.13%	93.94%	92.57%
Data Augmentation - Flip	——	92.73%	84.41%	92.94%	91.29%
Graph Data Augmentation	——	92.10%[d]	——	92.81%	**91.07%**[a]
None	69.62%	87.78%	——	88.83%	85.88%[a]

[a] No priors about the structure
[b] Lin et al., 2015
[c] Defferrard et al., 2016
[d] Data augmentation done with covariance graph

Table 7: CIFAR-10 result comparison table.

Experiments on a fMRI dataset

The PINES dataset consists of functional Magnetic Resonance Imaging (fMRI) scans on 182 subjects, during an emotional picture rating task (Chang et al., 2015). We fetched individual first-level statistical maps (beta images) for the minimal and maximal ratings from `https://neurovault.org/collections/1964/`, to generate the dataset. Full brain data was masked on the MNI template and resampled to a 16mm cubic grid, in order to reduce dimensionality of the dataset while keeping a regular geometrical structure. Final volumes used for classification contain 369 signals for each subject and rating.
We used a shallow network. The results on Table 8 show that our method was able to improve over CNNs, MLPs and other graph-based extended convolutional neural networks.

Support	None		Neighborhood Graph	
Method	MLP	CNN (1x1 kernels)	ChebNet[c]	Proposed
Accuracy	82.62%	84.30%	82.80%	**85.08%**

Table 8: PINES fMRI dataset accuracy comparison table.

3.5 Conclusion

TODO:

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `http://tensorflow.org/` (cit. on pp. 33, 112).

Arora, Raman, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee (2018). "Understanding Deep Neural Networks with Rectified Linear Units". In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=B1J_rgWRW` (cit. on p. 31).

Bass, Jean (1968). "Cours de mathématiques". In: (cit. on p. 11).

Bengio, Yoshua (2009). "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1, pp. 1–127 (cit. on p. 29).

Bianchini, Monica and Franco Scarselli (2014). "On the complexity of neural network classifiers: A comparison between shallow and deep architec-

tures". In: *IEEE transactions on neural networks and learning systems* 25.8, pp. 1553–1565 (cit. on p. 31).

Brandt, Heinrich (1927). "Über eine Verallgemeinerung des Gruppenbegriffes". In: *Mathematische Annalen* 96.1, pp. 360–366 (cit. on p. 76).

Cayley, Professor (1878). "Desiderata and Suggestions: No. 2. The Theory of Groups: Graphical Representation". In: *American Journal of Mathematics* 1.2, pp. 174–176. ISSN: 00029327, 10806377. URL: `http://www.jstor.org/stable/2369306` (cit. on p. 68).

Chang, Luke J, Peter J Gianaros, Stephen B Manuck, Anjali Krishnan, and Tor D Wager (2015). "A sensitive and specific neural signature for picture-induced negative affect". In: *PLoS biology* 13.6, e1002180 (cit. on p. 124).

Chellapilla, Kumar, Sidd Puri, and Patrice Simard (2006). "High performance convolutional neural networks for document processing". In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft (cit. on p. 102).

Chetlur, Sharan, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer (2014). "cudnn: Efficient primitives for deep learning". In: *arXiv preprint arXiv:1410.0759* (cit. on p. 102).

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2015). "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (cit. on p. 30).

Cohen, Nadav and Amnon Shashua (2016). "Convolutional rectifier networks as generalized tensor decompositions". In: *International Conference on Machine Learning*, pp. 955–963 (cit. on p. 31).

Cohen, Nadav, Ronen Tamari, and Amnon Shashua (2018). "Boosting Dilated Convolutional Networks with Mixed Tensor Decompositions". In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=S1JHhv6TW` (cit. on pp. 30, 31).

Cord, Matthieu (2016). *Deep learning an weak supervision for image classification*. [Online; accessed April-2018]. URL: `http://webia.lip6.fr/~cord/pdfs/news/TalkDeepCordI3S.pdf` (cit. on p. 22).

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4, pp. 303–314 (cit. on p. 29).

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in Neural Information Processing Systems*, pp. 3837–3845 (cit. on pp. 123, 124).

Delalleau, Olivier and Yoshua Bengio (2011). "Shallow vs. deep sum-product networks". In: *Advances in Neural Information Processing Systems*, pp. 666–674 (cit. on pp. 30, 31).

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, pp. 248–255 (cit. on pp. 23, 29).

Eldan, Ronen and Ohad Shamir (2016). "The power of depth for feedforward neural networks". In: *Conference on Learning Theory*, pp. 907–940 (cit. on p. 31).

Exel, Ruy (1998). "Partial actions of groups and actions of inverse semigroups". In: *Proceedings of the American Mathematical Society* 126.12, pp. 3481–3494 (cit. on p. 84).

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256 (cit. on pp. 29, 108, 111).

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *International Conference on Artificial Intelligence and Statistics*, pp. 315–323 (cit. on pp. 20, 29, 109).

Grelier, Nicolas, Bastien Pasdeloup, Jean-Charles Vialatte, and Vincent Gripon (2016). "Neighborhood-Preserving Translations on Graphs". In: *Proceedings of IEEE GlobalSIP*, pp. 410–414 (cit. on pp. 114, 117).

Hackbusch, Wolfgang (2012). *Tensor spaces and numerical tensor calculus*. Vol. 42. Springer Science & Business Media (cit. on pp. 11, 12).

Håstad, Johan and Mikael Goldmann (1991). "On the power of small-depth threshold circuits". In: *Computational Complexity* 1.2, pp. 113–129 (cit. on p. 30).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034 (cit. on p. 30).

— (2016a). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on pp. 19, 23, 24).

— (2016b). "Identity mappings in deep residual networks". In: *European Conference on Computer Vision*. Springer, pp. 630–645 (cit. on p. 123).

Hinton, Geoffrey, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. (2012). "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal Processing Magazine* 29.6, pp. 82–97 (cit. on p. 19).

Hinton, Geoffrey E, Simon Osindero, and Yee-Whye Teh (2006). "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7, pp. 1527–1554 (cit. on p. 29).

Hornik, Kurt (1991). "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2, pp. 251–257 (cit. on p. 29).

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5, pp. 359–366 (cit. on pp. 29, 32).

Huang, Gao, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten (2017). "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2, p. 3 (cit. on pp. 19, 23, 24).

Jarrett, Kevin, Koray Kavukcuoglu, Yann LeCun, et al. (2009). "What is the best multi-stage architecture for object recognition?" In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, pp. 2146–2153 (cit. on p. 29).

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (cit. on p. 109).

Klambauer, Günter, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter (2017). "Self-Normalizing Neural Networks". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 971–980. URL: `http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf` (cit. on p. 30).

Krizhevsky, Alex (2009). "Learning multiple layers of features from tiny images". In: (cit. on pp. 112, 123).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems*, pp. 1097–1105 (cit. on pp. 19, 29, 33, 112).

Lassance, Carlos Eduardo Rosar Kos, Jean-Charles Vialatte, and Vincent Gripon (2018). "Matching Convolutional Neural Networks without Priors about Data". In: (cit. on p. 115).

LeCun, Y. (1987). "Modeles connexionnistes de l'apprentissage (connectionist learning models)". PhD thesis. Université P. et M. Curie (Paris 6) (cit. on p. 25).

LeCun, Yann, Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10, p. 1995 (cit. on pp. 19, 33).

LeCun, Yann, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551 (cit. on pp. 19, 21, 22, 29).

LeCun, Yann, Corinna Cortes, and Christopher JC Burges (1998). *The MNIST database of handwritten digits* (cit. on pp. 105, 109).

Li, Xin (2016). "Partial transformation groupoids attached to graphs and semigroups". In: *International Mathematics Research Notices* 2017.17, pp. 5233–5259 (cit. on p. 84).

Lin, Henry W, Max Tegmark, and David Rolnick (2017). "Why does deep and cheap learning work so well?" In: *Journal of Statistical Physics* 168.6, pp. 1223–1247 (cit. on p. 31).

Lin, Zhouhan, Roland Memisevic, and Kishore Reddy Konda (2015). "How far can we go without convolution: Improving fully-connected networks". In: *CoRR* abs/1511.02580. arXiv: `1511.02580`. URL: `http://arxiv.org/abs/1511.02580` (cit. on pp. 123, 124).

Maas, Andrew L, Awni Y Hannun, and Andrew Y Ng (2013). "Rectifier nonlinearities improve neural network acoustic models". In: *Proceedings of the 30th international conference on machine learning* (cit. on p. 30).

Marcus, Marvin (1975). "Finite dimensional multilinear algebra". In: (cit. on p. 11).

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133 (cit. on p. 25).

Mhaskar, Hrushikesh, Qianli Liao, and Tomaso Poggio (2016). "Learning functions: when is deep better than shallow". In: *arXiv preprint arXiv:1603.00988* (cit. on p. 31).

Montufar, Guido F, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio (2014). "On the number of linear regions of deep neural networks". In: *Advances in neural information processing systems*, pp. 2924–2932 (cit. on p. 30).

Ng, Andrew Y (2004). "Feature selection, L 1 vs. L 2 regularization, and rotational invariance". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 78 (cit. on p. 109).

Nica, Alexandru (1994). "On a groupoid construction for actions of certain inverse semigroups". In: *International Journal of Mathematics* 5.03, pp. 349–372 (cit. on p. 84).

Nickolls, John, Ian Buck, Michael Garland, and Kevin Skadron (2008). "Scalable parallel programming with CUDA". In: *ACM SIGGRAPH 2008 classes*. ACM, p. 16 (cit. on p. 29).

Orhan, Emin and Xaq Pitkow (2018). "Skip Connections Eliminate Singularities". In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=HkwBEMWCZ` (cit. on p. 31).

Pan, Xingyuan and Vivek Srikumar (2016). "Expressiveness of rectifier networks". In: *International Conference on Machine Learning*, pp. 2427–2435 (cit. on p. 31).

Pascanu, Razvan, Guido Montufar, and Yoshua Bengio (2013). "On the number of response regions of deep feed forward networks with piece-wise linear activations". In: *arXiv preprint arXiv:1312.6098* (cit. on p. 30).

Pasdeloup, Bastien, Vincent Gripon, Jean-Charles Vialatte, and Dominique Pastor (2017a). "Convolutional neural networks on irregular domains through approximate translations on inferred graphs". In: *arXiv preprint arXiv:1710.10035* (cit. on pp. 114, 116).

Pasdeloup, Bastien, Vincent Gripon, Nicolas Grelier, Jean-Charles Vialatte, and Dominique Pastor (2017b). "Translations on graphs with neighborhood preservation". In: *arXiv preprint arXiv:1709.03859* (cit. on pp. 114, 117).

Poggio, Tomaso, Fabio Anselmi, and Lorenzo Rosasco (2015). *I-theory on depth vs width: hierarchical function composition*. Tech. rep. Center for Brains, Minds and Machines (CBMM) (cit. on p. 31).

Poole, Ben, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli (2016). "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in neural information processing systems*, pp. 3360–3368 (cit. on p. 31).

Raghu, Maithra, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein (2016). "On the expressive power of deep neural networks". In: *arXiv preprint arXiv:1606.05336* (cit. on p. 31).

Robbins, Herbert and Sutton Monro (1985). "A stochastic approximation method". In: *Herbert Robbins Selected Papers*. Springer, pp. 102–109 (cit. on p. 26).

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science (cit. on pp. 19, 26).

Schwartz, Laurent (1957). *Théorie des distributions*. Vol. 2. Hermann Paris (cit. on p. 49).

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (cit. on pp. 19, 21, 22).

Srivastava, Nitish, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958 (cit. on pp. 30, 109).

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. (2015). "Going deeper with convolutions". In: *Conference on Computer Vision and Pattern Recognition* (cit. on pp. 19, 23).

Van Den Oord, Aaron, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (cit. on p. 31).

Vialatte, Jean-Charles, Vincent Gripon, and Grégoire Mercier (2016). "Generalizing the convolution operator to extend cnns to irregular domains". In: *arXiv preprint arXiv:1606.01166* (cit. on p. 105).

Vialatte, Jean-Charles, Vincent Gripon, and Gilles Coppin (2017). "Learning Local Receptive Fields and their Weight Sharing Scheme on Graphs". In: *arXiv preprint arXiv:1706.02684* (cit. on pp. 98, 107).

Weinstein, Alan (1996). "Groupoids: unifying internal and external symmetry". In: *Notices of the AMS* 43.7, pp. 744–752 (cit. on p. 76).

Widrow, Bernard and Marcian E Hoff (1960). *Adaptive switching circuits*. Tech. rep. STANFORD UNIV CA STANFORD ELECTRONICS LABS (cit. on p. 26).

Wikipedia, contributors (2018a). *Feedforward neural network — Wikipedia, The Free Encyclopedia*. [Online; accessed April-2018]. URL: `https://en.wikipedia.org/wiki/Feedforward_neural_network` (cit. on p. 19).

— (2018b). *Softmax function — Wikipedia, The Free Encyclopedia*. [Online; accessed April-2018]. URL: `https://en.wikipedia.org/wiki/Softmax_function` (cit. on p. 20).

Williamson, S Gill (2015). "Tensor spaces-the basics". In: *arXiv preprint arXiv:1510.02428* (cit. on p. 11).

Zell, Andreas (1994). *Simulation neuronaler netze*. Vol. 1. Addison-Wesley Bonn (cit. on p. 19).

Zermelo, Ernst (1904). "Beweis, daß jede Menge wohlgeordnet werden kann". In: *Mathematische Annalen* 59.4, pp. 514–516 (cit. on p. 83).

Zoph, Barret and Quoc V Le (2016). "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (cit. on p. 19).