

# 1 Definitions

In this section we recall and define the notions we will be using throughout this manuscript.

## 1.1 Tensors

TODO: reformat

Vector spaces considered in this thesis are always assumed to be finite-dimensional.

### Definition 1.1. Tensor space

We define a *tensor space of rank  $r$*  as a cartesian product of  $r$  vector spaces, equipped with the coordinate-wise sum and the monilinear outer product. 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. For notational conveniency, we will often abusively confound vector spaces with tensor spaces of rank 1, and matrix spaces with tensor spaces of rank 2.

### Definition 1.2. Tensor

A *tensor  $t$*  is an object of a tensor space. An *entry* of  $t$  is denoted  $t[i_1, i_2, \dots, i_r]$ . The *shape* of the tensor  $t$  is denoted  $n_1(t) \times n_2(t) \times \cdots \times n_r(t)$

### Definition 1.3. Reshaping

### Definition 1.4. Slice

A *slice* of a tensor  $t$ , along the last ranks  $\{r_1, r_2, \dots, r_s\}$ , and indexed by  $\{i_{r_1}, i_{r_2}, \dots, i_{r_s}\}$ , is the tensor  $t'$  of rank  $r - s$  such that

$$\begin{aligned} t'[i'_1, i'_2, \dots, i'_{r-s}] &= t[i'_1, i'_2, \dots, i'_{r-s}, i_{r_1}, i_{r_2}, \dots, i_{r_s}] \\ \text{i.e. } t' &:= t[:, :, \dots, :, i'_{r-s}, i_{r_1}, i_{r_2}, \dots, i_{r_s}] \end{aligned}$$

It is denoted  $t_{i_{r_1} i_{r_2} \dots i_{r_s}}$ .

### Definition 1.5. Slice

A *slice* of a tensor  $t$ , along the ranks  $\{r_1, r_2, \dots, r_s\}$ , and indexed by  $\{i_{r_1}, i_{r_2}, \dots, i_{r_s}\}$ , is the tensor  $t'$  of rank  $r - s$  such that

$$t'[i'_1, i'_2, \dots, i'_{r-s}] = t[i_1, i_2, \dots, i_r] \tag{1}$$

### Definition 1.6. Contraction

### Remark 1. Covariant and contravariant indices

### Definition 1.7. Convolution

## 1.2 Neural Networks

We denote by  $I_f$  the *domain of definition* of a function  $f$  ("I" for "input") and by  $O_f = f(I_f)$  its *image* ("O" for "output"), and we represent it as  $I_f \xrightarrow{f} O_f$ . An activation function  $h$  defined from a tensor space to itself is a 1-d function applied dimension-wise and we use the functional notation  $h(v)[i_1, i_2, \dots, i_r] = h(v[i_1, i_2, \dots, i_r])$ .

### Definition 1.8. Neural network

Let  $F$  be a function such that  $I_f$  and  $O_f$  are vector or tensor spaces.  $F$  is a *functional formulation* of a *neural network* if there are a series of linear or affine functions  $(g_k)_{k=1,2,\dots,L}$  and a series of non-linear derivable activation functions  $(h_k)_{k=1,2,\dots,L}$  such that:

$$\begin{cases} \forall k \in \{1, 2, \dots, L\}, f_k = h_k \circ g_k, \\ I_F = I_{f_1} \xrightarrow{f_1} O_{f_1} \cong I_{f_2} \xrightarrow{f_2} \dots \xrightarrow{f_L} O_{f_L} = O_F, \\ F = f_L \circ \dots \circ f_2 \circ f_1 \end{cases} \quad (2)$$

The couple  $(g_k, h_k)$  is called the *k-th layer* of the neural network. For  $x \in I_f$ , we denote by  $x_k = f_k \circ \dots \circ f_2 \circ f_1(x)$  the *activations* of the *k-th* layer.

### Remark 2. Bias

Affine functions  $\tilde{g}$  can be written as a sum between a linear function  $g$  and a constant vector  $b$ . For notational conveniency, we only consider linear functions in this section.  $b$  is called the *bias*. Its role is to augment the expressivity of the neural network's family of functions.

### Remark 3. Connectivity matrix

Any linear function  $g$  is characterized by a *set of parameters*  $\Theta_g$ , that we order arbitrarily in the dimensions of a vector  $\theta_g$ . Without loss of generality, let's suppose  $I_g$  and  $O_g$  are vector spaces<sup>1</sup>. Then there exists a *connectivity matrix*  $W_g$ , for which:

$$\begin{cases} \forall x \in I_g, g(x) = W_g x \\ \forall (i, j), \begin{cases} \exists w \in \Theta_g, W_g[i, j] := w \\ \text{or } W_g[i, j] = 0 \end{cases} \end{cases} \quad (3)$$

### Remark 4. Biological inspiration

A (*computational*) *neuron* is a computational unit that is biologically inspired. Each neuron should be capable of:

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

---

<sup>1</sup>for instance if they are tensor spaces, they can be reshaped to vector spaces

That is to say, each domain  $\{I_{f_k}\}$ ,  $O_F$  can be interpreted as a layer of neurons, with one neuron for each dimension. The connectivity matrices  $\{W_k\}$  describe the connexions between each successive layers. A neuron is illustrated on Figure 1.

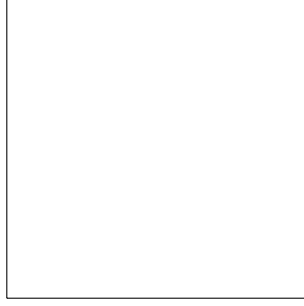


Figure 1: A neuron (placeholder)

**Definition 1.9. Weights**

The *weights* of the  $k$ -th layer of a neural network, denoted  $\Theta_k$ , are defined as the set of parameters of its linear part. A weight that appears multiple times in  $W_k$  is said to be *shared*. Two parameters of  $W_k$  that share a weight are said to be *tied*.

**Remark 5. Training**

Usually, a *loss* function  $\mathcal{L}$  penalizes the output  $x_L = F(x)$ . Its gradient w.r.t.  $\theta_k$ , denoted  $\vec{\nabla}_{\theta_k}$ , is used to update the weights via an optimization algorithm based on gradient descent and a learning rate  $\alpha$ , that is:

$$\theta_k^{\text{new}} = \theta_k^{\text{old}} - \alpha \cdot \vec{\nabla}_{\theta_k} \quad (4)$$

where  $\alpha$  depends on training variables and can be a scalar or a vector, and so  $\cdot$  denotes here outer or pointwise product. **TODO: Give some ref of optimization algorithms**

**Remark 6. Linear complexity**

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:

$$\vec{\nabla}_{\theta_k} = J(x_k)_{\theta_k} \vec{\nabla}_{x_k} \quad (5)$$

$$\begin{aligned} \vec{\nabla}_{x_k} &= J(x_{k+1})_{x_k} \vec{\nabla}_{x_{k+1}} \\ \vec{\nabla}_{x_{k+1}} &= J(x_{k+2})_{x_{k+1}} \vec{\nabla}_{x_{k+2}} \\ &\dots \\ \vec{\nabla}_{x_{L-1}} &= J(x_L)_{x_{L-1}} \vec{\nabla}_{x_L} \end{aligned} \quad (6)$$

Obtaining,

$$\vec{\nabla}_{\theta_k} = J(x_k)_{\theta_k} \left( \prod_{p=k}^{L-1} J(x_{p+1})_{x_p} \right) \vec{\nabla}_{x_L} \quad (7)$$

where  $J(\cdot)_{\text{wrt}}$  are the respective jacobians which can be determined with the layer's expressions and the  $\{x_k\}$ .

This allows to compute the gradients with a complexity that is linear with the number of weights, instead of being quadratic if it were done with the difference quotient expression of the derivatives.

#### Remark 7. Neural interpretation

We can remark that (6) rewrites as

$$\begin{aligned} \vec{\nabla}_{x_k} &= J(x_{k+1})_{x_k} \vec{\nabla}_{x_{k+1}} \\ &= J(h(x'_k))_{x'_k} J(W_k x_k)_{x_k} \vec{\nabla}_{x_{k+1}} \end{aligned} \quad (8)$$

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

$$J(h(x'_k))_{x'_k} [i, j] = \delta_i^j h'(x'_k[i]) \quad (9)$$

$$\begin{aligned} J(h(x'_k))_{x'_k} &= I h'(x'_k) \\ J(W_k x_k)_{x_k} &= W_k^T \end{aligned} \quad (10)$$

That means that we can write  $\vec{\nabla}_{x_k} = (\tilde{h}_k \circ \tilde{g}_k)(\vec{\nabla}_{x_{k+1}})$  such that the connectivity matrix  $\tilde{W}_k$  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.

#### Definition 1.10. Dense layer

A *dense layer*  $(g, h)$  is a layer such that

$$\left\{ \begin{array}{l} I_g \text{ and } O_g \text{ are vector spaces} \\ \forall w \in \Theta_g, \exists!(i, j), W_g[i, j] := w \\ |\Theta_g| = n_1(W_g) n_2(W_g) \end{array} \right.$$

**Definition 1.11. Partially connected layer**

A *partially connected layer* is a layer such that

$$\begin{cases} I_g \text{ and } O_g \text{ are vector spaces} \\ \exists(i, j), \forall w \in \Theta_g, W_g[i, j] \neq w \end{cases}$$

**Definition 1.12. Convolutional layer**

A *n-dimensional convolutional layer*  $(g, h)$  is a layer such that there is a *weight tensor*  $W$  of rank  $n + 2$  for which

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

where  $p$  and  $q$  index the last ranks and  $*_n$  denotes the  $n$ -d convolution. The tensor slices indexed by  $p$  and  $q$  are typically called *feature maps*. Note that if  $I_g$  and  $O_g$  are reshaped to vector spaces,  $W_g$  would be a Toeplitz matrix.

**Definition 1.13. Pooling**

A layer with *pooling*  $(g, h)$  is such that  $g = g_1 \circ g_2$ , where  $(g_1, h)$  is a layer and  $g_2$  is a pooling operation.

**Definition 1.14. Reshaping**

A layer with *dropout*  $(g, h)$  is such that  $h = h_1 \circ h_2$ , where  $(g, h_2)$  is a layer and  $h_1$  is a dropout operation [?]. When dropout is used, a certain number of neurons are randomly set to zero during the training phase, compensated at test time by scaling down the whole layer. This is done to prevent overfitting.

**TODO: neuron interpretation**

A multilayer perceptron (MLP) [?] is a neural network composed of only dense layers. A convolutional neural network (CNN) [?] is a neural network composed of convolutional layers.

Neural networks are commonly used for machine learning tasks. For example, to perform supervised classification, we usually add a dense output layer  $s = (g_{L+1}, h_{L+1})$  with as many neurons as classes. We measure the error between an output and its expected output with a discriminative loss function  $\mathcal{L}$ . During the training phase, the weights of the network are adapted for the classification task based on the errors that are back-propagated [?] via the chain rule and according to a chosen optimization algorithm (e.g. [?]).

### 1.3 Graphs

A graph  $G$  is defined as a couple  $(V, E)$  where  $V$  represents the set of nodes and  $E \subseteq \binom{V}{2}$  is the set of edges connecting these nodes.

**TODO: Example of figure**

We encounter 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, coined *connectivity graph*. It encodes how the information is propagated through a layer to another. See section 1.3.1.
- A computation graph is used by deep learning frameworks to keep track of the dependencies between layers of a deep learning models, in order to compute forward and back-propagation. See section 1.3.2.
- A graph can represent the underlying structure of an object (often a vector), whose nodes represent its features. See section 1.3.3.
- Datasets can also be graph-structured, where the nodes represent the objects of the dataset. See section 1.3.4.

### 1.3.1 Connectivity graph

A Connectivity graph is the bipartite graph whose adjacency matrix is the connectivity graph 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  $\Theta$  is a  $n \times m$  matrix.

**Definition 1.15.** 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  $\Theta_{ij}$  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 2 depicts some examples.

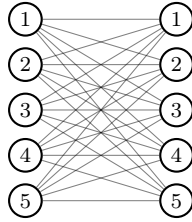


Figure 2: Examples

TODO: Figure 2. 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  $\kappa$ , which we will refer to as the *kernel of weights*. Then we can define a labelling of the edges  $s : E \rightarrow K$ .  $s$  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 3 depicts the connectivity graph of a 1-d convolution layer and its weight sharing scheme.

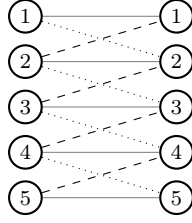


Figure 3: Depiction of a 1D-convolutional layer and its weight sharing scheme.

TODO: Add weight sharing scheme in Figure 3

### 1.3.2 Computation graph

### 1.3.3 Underlying graph structure

### 1.3.4 Graph-structured dataset

transductive vs inductive

## 1.4 Geometric grids

## 1.5 Grid graphs

## 1.6 Spatial graphs

## 1.7 Projections of spatial graphs