(i) An *architecture* $a(\mathcal{N}) = (N, \varphi)$, where $N = (N_0, \dots, N_L) \in \mathbb{N}^{L+1}$ and $\varphi = \left(\varphi^{(l)}\right)_{l=1}^{L}$ is a tuple of differentiable functions $\varphi^{(l)} \colon \mathbb{R} \to \mathbb{R}$. Here differentiability is meant except perhaps on a set of zero measure. The number L counts the layers of the network, while N_0, N_L, and N_l for $l \in [L-1]$ are the numbers of neurons of the input, output, and l-th hidden layers, respectively, and $\varphi^{(l)}$ is the *activation function* of layer l.

(ii) A set of parameters $\theta(\mathcal{N}) = \left(\theta^{(l)}\right)_{l=1}^{L}$, where $\theta^{(l)} = \left(W^{(l)}, b^{(l)}\right)$ are the parameters of layer l, belonging to $\mathbb{R}^{N_l \times N_{l-1}} \times \mathbb{R}^{N_l}$ for $l \in [L]$ and known as *weights* and *biases*, respectively.

A FCFNN induces a directed graph and a function depending on it. The directed graph defined by the architecture, denoted by $G(\mathcal{N})$, is given by $L+1$ disjoint ordered sets $V_l = \{v_l^1, \dots, v_l^{N_l}\}$, $l = 0, \dots, L$, with the property that every vertex in V_l has an edge pointing to it from all the vertices in V_{l-1}, for $l \in [L]$. The vertices of this graph are called *neurons*. The function $\phi_{\mathcal{N}} \colon \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined on top of the directed graph is given by the recursive formula

$$
\begin{aligned}
\phi_{\mathcal{N}}(x) &= \phi_{\mathcal{N}}^{(L)}(x), \\
\phi_{\mathcal{N}}^{(l)}(x) &= \varphi^{(l)}\left(\bar{\phi}_{\mathcal{N}}^{(l)}(x)\right) \text{ for } l \in [L], \\
\bar{\phi}_{\mathcal{N}}^{(l)}(x) &= \begin{cases} W^{(1)}x + b^{(1)} & \text{if } l = 1, \\ W^{(l)}\phi_{\mathcal{N}}^{(l-1)}(x) + b^{(l)} & \text{if } l \in \{2, \dots, L\}, \end{cases}
\end{aligned}
\tag{3}
$$

where each component x_i of x is associated with the vertex v_0^i, and each intermediate function value $\phi_{\mathcal{N}}^{(l)}(x)_i$ is associated with the vertex v_l^i. Given an input value x, we say that the values $\phi_{\mathcal{N}}^{(l)}(x)_i$ are *activations* of the vertices v_l^i. A visual example of a FCFNN is shown in Figure 2.

Given a specific tuple $a = (N, \varphi)$ representing an architecture for a FCFNN and a subset $\Theta \subseteq \prod_{l \in [L]} \mathbb{R}^{\mathbb{N}_l \times \mathbb{N}_{l-1}} \times \mathbb{R}^{N_l}$, we denote the hypothesis set of all the neural networks represented by the architecture a with parameters in Θ as

$$\mathcal{F}_{a,\Theta} = \{\mathcal{N} : a(\mathcal{N}) = a,\, \theta(\mathcal{N}) \in \Theta\}\,.$$

If Θ is the space of all possible parameters, we simply write $\mathcal{F}_a$. Also, when it is clear that we speak about neural networks in a specific hypothesis set $\mathcal{F}_{a,\Theta}$, we denote the neural network with architecture a and parameters $\theta \in \Theta$ as $\mathcal{N}_\theta$.

Note that, fixing a dataset $\mathcal{D}$, an architecture a, and a set of possible parameters Θ, the composition of the empirical risk with the neural network functions in $\mathcal{F}_{a,\Theta}$ can be seen as a function $\widehat{\mathcal{R}}_{\mathcal{D}_{\text{train}}}(\theta)$ of the parameters θ. Usually, training algorithms for neural networks, given an initial set of parameters $\theta^{(0)} \in \Theta$, generate a discrete weight trajectory $\left(\theta^{(i)}\right)_{i=1}^{N}$ by minimizing the empirical risk $\widehat{\mathcal{R}}_{\mathcal{D}_{\text{train}}}(\theta)$ iteratively with respect to the parameters θ until some stopping criteria are satisfied. The output of the algorithm is one of the weights $\theta^{(i)}$ generated in the trajectory, usually the last one, $\theta^{(N)}$. In this survey, we assume that neural networks are trained in this way.

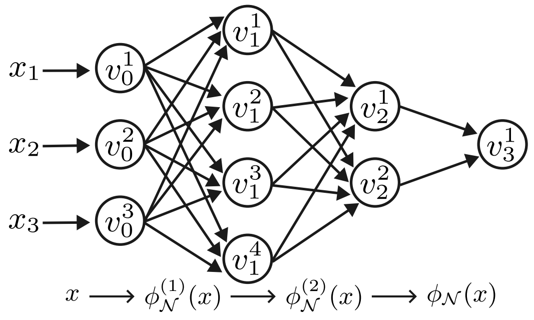
x_1
x_2
x_3
v_0^1
v_0^2
v_0^3
v_1^1
v_1^2
v_1^3
v_1^4
v_2^1
v_2^2
v_3^1
$x \longrightarrow \phi_{\mathcal{N}}^{(1)}(x) \longrightarrow \phi_{\mathcal{N}}^{(2)}(x) \longrightarrow \phi_{\mathcal{N}}(x)$

Figure 2: A graphical representation of a fully connected feedforward neural network $\mathcal{N}$ with $L = 3$ and $N = (3, 4, 2, 1)$. Each FCFNN can be represented as a sequence of sets of vertices, called layers, with vertices of the layer l connected with the vertices of the layer $l-1$, for $l \in [L]$. Given an input $x \in \mathbb{R}^{N_0} = \mathbb{R}^3$, the input values x_i are associated with the vertices v_0^i of the first (input) layer and then transformed sequentially by a set of maps. In this representation, each edge indicates that the value of the source vertex is used for the computation of the value of the target vertex. Values for vertices are computed sequentially from the first layer to the last. Each transformation from layer $l-1$ to layer l, made by the corresponding function $\phi_{\mathcal{N}}^{(l)}$, is a composite of an affine transformation $\bar{\phi}_{\mathcal{N}}^{(l)}$ and the activation function $\varphi^{(l)}$ applied elementwise to all the outputs of $\bar{\phi}_{\mathcal{N}}^{(l)}$.

A special type of FCFNN is a *convolutional neural network* (CNN), which is a FCFNN with a specific architecture designed to work with data that have a grid-like structure, especially images. A CNN is one of the most common type of neural networks used in computer vision. CNNs implement two functions between layers that impose restrictions on the set of weights and on the activation functions: convolutional layers and pooling operators. A *convolutional layer* is a layer whose outputs are the result of performing convolutions between the values associated to the neurons of its input layer rearranged as a grid and a filter tensor that imposes a set of equalities on the coefficients of the weight matrices. In one of the most typical scenarios, a convolutional layer between layer $i-1$ and i is a transformation between the activations of V_{i-1} and V_i in such a way V_{i-1} and V_i are ordered in grid structures of size $h^{(i-1)\times}w^{(i-1)} \times c^{(i-1)}$ and $h^{(i)\times}w^{(i)} \times c^{(i)}$, respectively. We say that a layer i arranged in this grid structure has *height* $h^{(i)}$, *width* $w^{(i)}$, and c^i *channels*, where each channel is defined as the subgrid obtained by fixing one value in $\left[c^i\right]$ for the third dimension. The convolutional layer transformation is produced by (discretely) convolving $c^{(i)}$ filter tensors $C_j^{(i)}$, $j \in \left[d^{(i)}\right]$ of size $h \times w \times c^{(i-1)}$, where $h \times w$ is called *convolution size*, with the activations in the grid structure of V_{i-1}, obtaining $c^{(i)}$ convolved grids of size $h^{(i)} \times w^{(i)}$, that are then concatenated to obtain the activations of V_i. *Pooling operators* are similar operations aiming to reduce the dimensionality of the input layer, which reduce or mantain the size of the input, and not necessarily all values in the input affect each output of the layer. Given a neuron, the region of the input that affects its activations is called its *receptive field.* For a thorough introduction to CNNs, we refer the reader to Goodfellow et al. (2016, Chapter 9).

Fully connected feedforward networks define functions $\phi_{\mathcal{N}} \colon \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$, which is optimal whenever $\mathcal{X}$ and $\mathcal{Y}$ are subsets of Euclidean spaces. However, there are specific tasks, mostly classification ones, in which one deals with a finite set of points in the output space, that is, there exists $k \in \mathbb{N}$ such that $\mathcal{Y} = [k]$. In these cases, one selects a projection function $\pi \colon \mathbb{R}^{N_L} \to [k]$ that projects the outputs of the neural network $\mathcal{N}$ into the finite set of points $[k]$ and one uses $\pi \circ \phi_{\mathcal{N}} \colon \mathbb{R}^{N_0} \to [k]$ as a model. In this context, the *decision region* of a label $y \in [k]$ is the set of inputs x such that $\pi(\phi_{\mathcal{N}}(x)) = y$. To train the neural network, it is common to choose surrogate loss functions $\mathcal{L}^{\text{surr}}$ that depend directly on the output of $\phi_{\mathcal{N}}$. In classification tasks, the most common configuration for a problem with $\mathcal{X}$ having $\mathbb{R}^d$ as an ambient space and $\mathcal{Y} = [k]$ is to choose neural networks with $N_0 = d$, $N_L = k$, projection $\pi(x_1, \ldots, x_k) = \operatorname{argmax}_{i \in [k]} x_i$, and surrogate loss function $\mathcal{L}^{\text{surr}}$ depending only on $\phi_{\mathcal{N}}$ like the *categorical cross entropy loss* CE, that, in its most basic form, looks like

$$\mathrm{CE}\left(\phi_{\mathcal{N}}, x, y\right) = -\frac{1}{k}\log\left(\frac{\exp\left(\phi_{\mathcal{N}}(x)_y\right)}{\sum_{i=1}^{k}\exp\left(\phi_{\mathcal{N}}(x)_i\right)}\right).$$

For neural networks with the configuration $N_L = k$ and $\pi(x_1, \ldots, x_k) = \operatorname{argmax}_{i \in [k]} x_i$, there is an ambiguity when the argmax is not unique, i.e., when there are at least two indices $i, j \in [k]$ such that $x_i = x_j$. In this case, the projection function must define a deterministic way to choose one of the equal-valued indices. Given a neural network $\mathcal{N}$ as the previous one, the points $x \in \mathcal{X}$ for which there exists an ambiguity is called *decision boundary* of $\mathcal{N}$. Formally,

$$\mathcal{B}\left(\mathcal{N}\right) = \left\{x \in \mathrm{Dom}(\phi_{\mathcal{N}}) : \exists i, j \in [N_L] \text{ such that } \phi_{\mathcal{N}}(x)_i = \phi_{\mathcal{N}}(x)_j = \max_{\iota \in [k]} \phi_{\mathcal{N}}(x)_\iota\right\}. \quad (4)$$

Sometimes, decision regions are studied without the decision boundary; in other words, $(\pi \circ \phi_{\mathcal{N}})^{-1}(y) \smallsetminus \mathcal{B}(\mathcal{N})$ is used. Figure 3 shows examples of decision boundaries and regions for classification problems with three labels.

In this context, it is usual to interpret the outputs of the function $\phi_{\mathcal{N}}$ as (unnormalized) probabilities, indicating the likelihood that the input belongs to one of the classes represented by the outputs. Specifically, $\phi_{\mathcal{N}}(x)_i$ denotes the (unnormalized) probability that x belongs to the class $i \in [k]$. Therefore, π is chosen as argmax since it selects the class with the highest probability according to the output of the neural network.

For binary classification problems, i.e., $\mathcal{Y} = [2]$, the usual neural network architectures, projection functions, and decision boundaries are slightly different. On the one hand, the number of neurons in the last layer is set to one, that is, $N_L = 1$. On the other hand, the usual projection function is given by $\pi(x) = 2$ if $x > b$ and $\pi(x) = 1$ if $x \leq b$, with the usual value of b being zero. In this configuration, the decision boundary is defined as

$$\mathcal{B}\left(\mathcal{N}\right) = \left\{x \in \mathrm{Dom}(\phi_{\mathcal{N}}) : \phi_{\mathcal{N}}(x) = b\right\}. \quad (5)$$