Neural Networks

1949 Hebb   Bryson & Ho 1969 — mid 1980 — 1989 LeCun   learn concepts

A feedforward neural network is described by a directed acyclic graph $G = (V, E)$, and a weight function over the edges $w: E \to \mathbb{R}$

connectionism

computational units

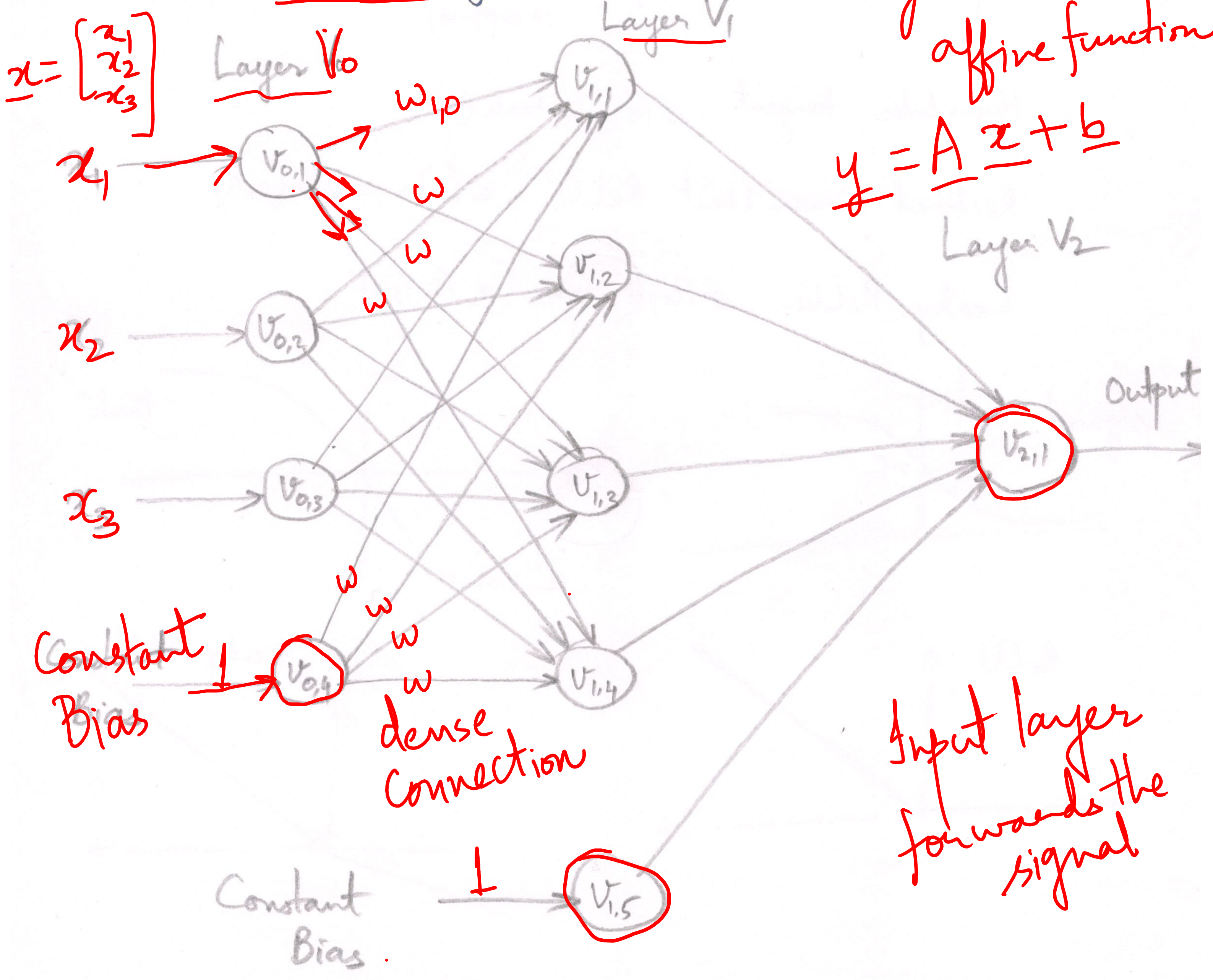
Nodes of the graph correspond to neurons.

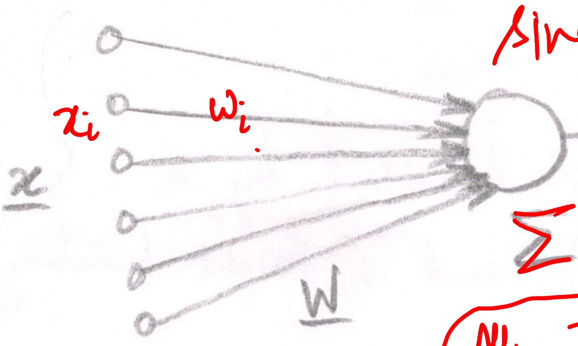
Each single neuron is modelled as a computational unit which performs affine transformation of the input and applies a scalar non-linearity $\sigma: \mathbb{R} \to \mathbb{R}$.

$y = mx + c$

affine function

$\underline{y} = \underline{A}\,\underline{x} + \underline{b}$





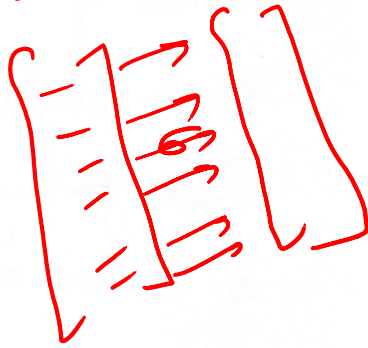
single neuron/unit

$\rightarrow \langle \underline{w}, \underline{x} \rangle$

Affine $\underline{W} \cdot \underline{x} = \sum_{i=1}^{n} w_i x_i$

$\boxed{\text{Affine Tr} + \text{Element wise non-linearity}}$ $\rightarrow$ non linear σ

element-wise



Possible non-linearities can be

Not differentiable $\left\{ \begin{array}{l} \text{sign function } \sigma(a) = \text{sign}(a) \\ \text{threshold function } \sigma(a) = \mathbb{1}_{[a>0]} \end{array} \right.$

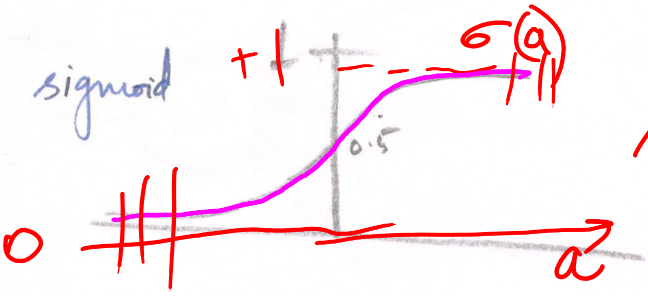
sigmoid function $\sigma(a) = \dfrac{1}{1+\exp(-a)}$

Hyperbolic tangent $\sigma(a) = \tanh(a)$

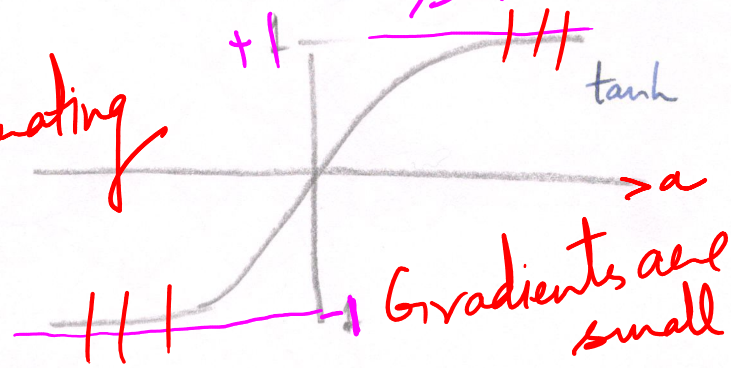
Rectified Linear Unit ReLU $\sigma(a) = \max(0, a)$

Leaky ReLU $\sigma(a) = \max(a, (0.1)a)$.

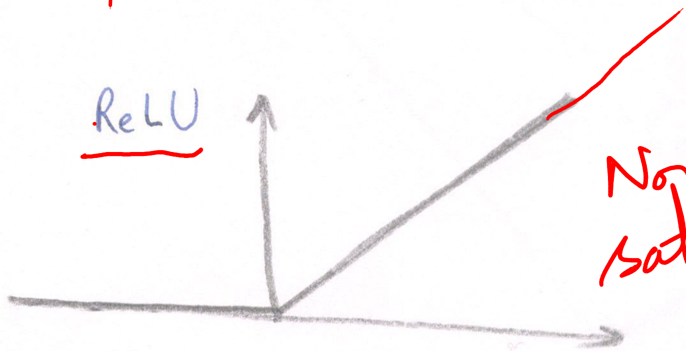
Smooth threshold



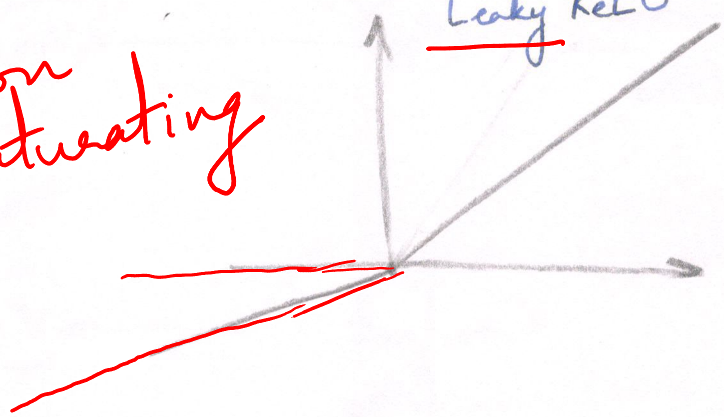
saturating

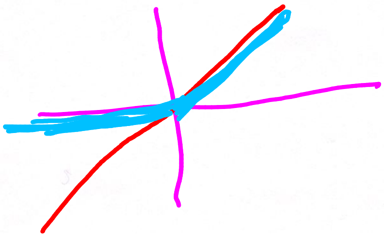


Gradients are small



Non saturating





$$\begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix} \times 5 = \begin{bmatrix}5\\10\\15\\20\\25\end{bmatrix}$$

element-wise

$$\begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix} \rightarrow \mathbb{1}[a>3] = \begin{bmatrix}0\\0\\0\\1\\1\end{bmatrix}$$

$$\underset{3\times3}{\begin{bmatrix}1_A & 2_A & \\ & & \\ & & \end{bmatrix}} \times \underset{3\times3}{\begin{bmatrix}1_B & 2_B & \\ & & \\ & & \end{bmatrix}} = \underset{3\times3.}{\begin{bmatrix}1_A1_B & 2_A2_B & \cdot \\ & \cdot & \cdot \\ & & \end{bmatrix}}$$

Matrix product

element-wise product

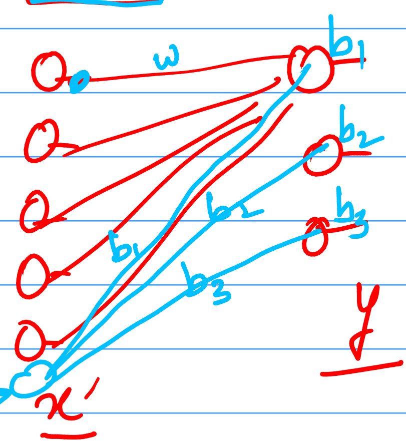
$(x, 1)$

$$\begin{bmatrix}x_1\\x_2\\x_3\\1\end{bmatrix}$$

$$y = \underbrace{A\,\underline{x}} + o.\underline{b}$$

A → weights to be learned

b → bias to be learned



$1 \times b_1$

$1 \times b_2$

$1 \times b_3$ Constant 1

bias & weights are learned in a uniform manner.

bias has been treated equivalent to edge weights

Each edge in the graph links the output of some neuron to the input of another neuron.

Affine Transformation

The input of a neuron is obtained by taking a weighted sum of the outputs of all the neurons connected to it, where the weighing is according to $\underline{w}$ $\langle \underline{w}, \underline{x} \rangle$

The network is organized in layers.

The set of nodes can be decomposed into a union of (non-empty) subsets $V = \bigcup_{t=0}^{T} V_t$ such that every edge in E connects some node in V_{t-1} to some node in V_t for some $t \in [T]$

V_0 is the input layer.

It contains $n+1$ neurons, where n is the dimensionality of the input space.

For every $i \in [n]$ the output neuron i in V_0 is simply x_i

The last neuron in V_0 is the constant neuron, which always outputs 1.

bias

We denote by $v_{t,i}$ the i^{th} neuron of the t^{th} layer.

$v_{\text{layer, neuron id}}$

output value $o_{t,i}(\underline{x})$ the output of $v_{t,i}$ when the network is fed with the input vector $\underline{x}$

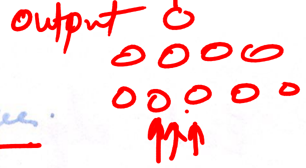
V_0 Input layer
$$\begin{cases} o_{0,i}(\underline{x}) = x_i \\ o_{0,n+1}(\underline{x}) = \underset{\text{constant bias}}{1} \quad \text{i.e. for } i = n+1 \text{ (last node)} \end{cases}$$

Layers $\underbrace{V_1 \ldots V_{T-1}}$ are called the Hidden layers.

T layer

V_0: input

V_T: output layer



The top layer V_T is called the output layer.

In simple prediction problems, the output layer contains a single neuron whose output is the output of the network.

$h(x) \in \mathbb{R}$

We refer to T as the number of layers/depth of the network. The size of the network is $|V|$. #nodes

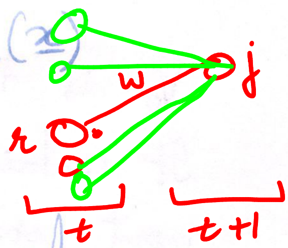
The width of the network is $\max_t |V_t|$

Suppose we have calculated the outputs of the neurons at V_t. Layer $t+1$ ~~as~~ then computes the activation value for every neuron j as

layer, neuron id.

Affine transform

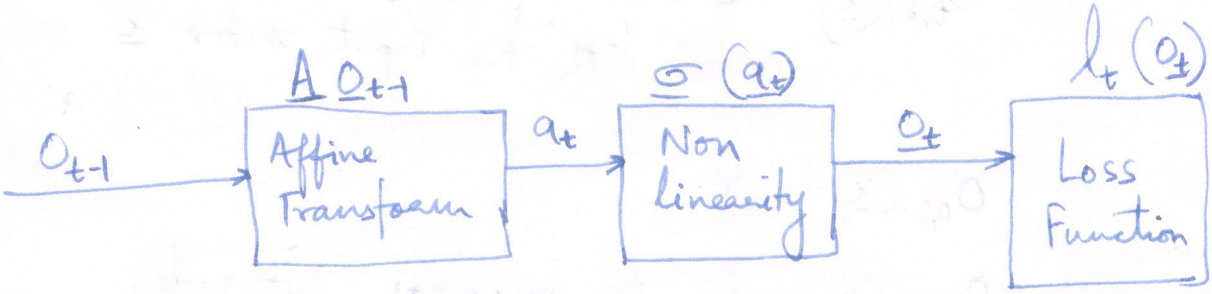
$$a_{t+1,j}(x) = \sum_{r:\,(v_{t,r},\,v_{t+1,j}) \in E} \underbrace{w(v_{t,r},\, v_{t+1,j})}\, o_{t,r}(x)$$



Applying the non-linearity to the activation value:

$$o_{t+1,j}(x) = \sigma(a_{t+1,j}(x))$$

Forward Propagation



Matrix A is formulated using layer weights W_{t-1}

Activations $a_t = A\,o_{t-1} = B\,w_{t-1}$ where matrix B is formulated using o_{t-1}