Particles and Cosmology
16th Baksan School on Astroparticle Physics

Machine Learning in Astroparticle Physics

Oleg Kalashev
Institute for Nuclear Research, RAS

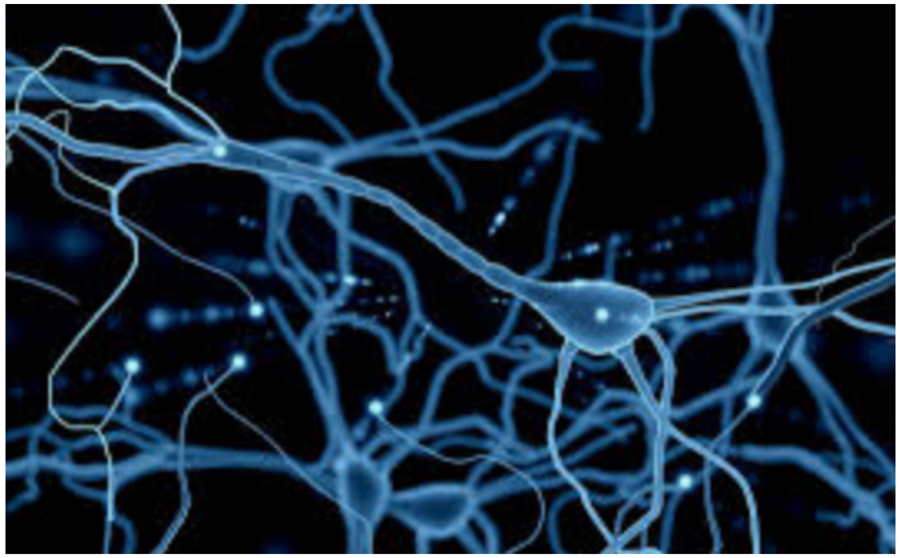
Lecture 2

April 10-18, 2019

Biological Neural Networks

- *Herbert Spencer, Principles of Psychology (1872)*
- *Alexander Bain, Mind and Body: The Theories of their Relation (1873)*
- *Theodor Meynert Psychiatry (1884)*

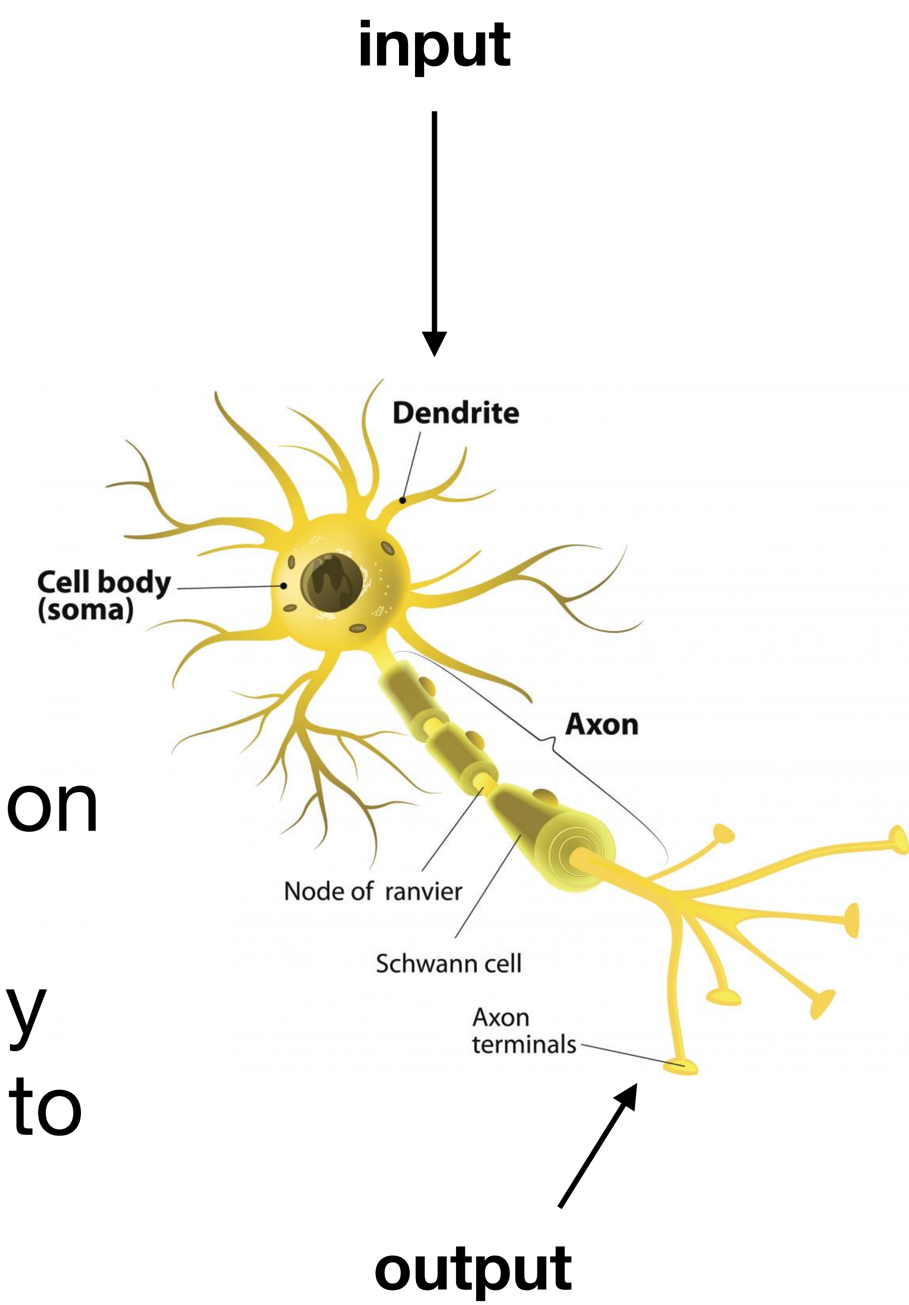
- Neurons are responsible for carrying information throughout the human body



- Neural circuits is a population of neurons interconnected by synapses to carry out a specific function when activated.
- Neural circuits interconnect to one another to form large scale brain networks.

Biological Neural Networks

- **Soma (cell body)** — this portion of the neuron receives information. It contains the cell's nucleus.
- **Dendrites** — these thin filaments carry information from other neurons to the soma. They are the "input" part of the cell.
- **Axon** — this long projection carries information from the soma and sends it off to other cells. This is the "output" part of the cell. It normally ends with a number of synapses connecting to the dendrites of other neurons.



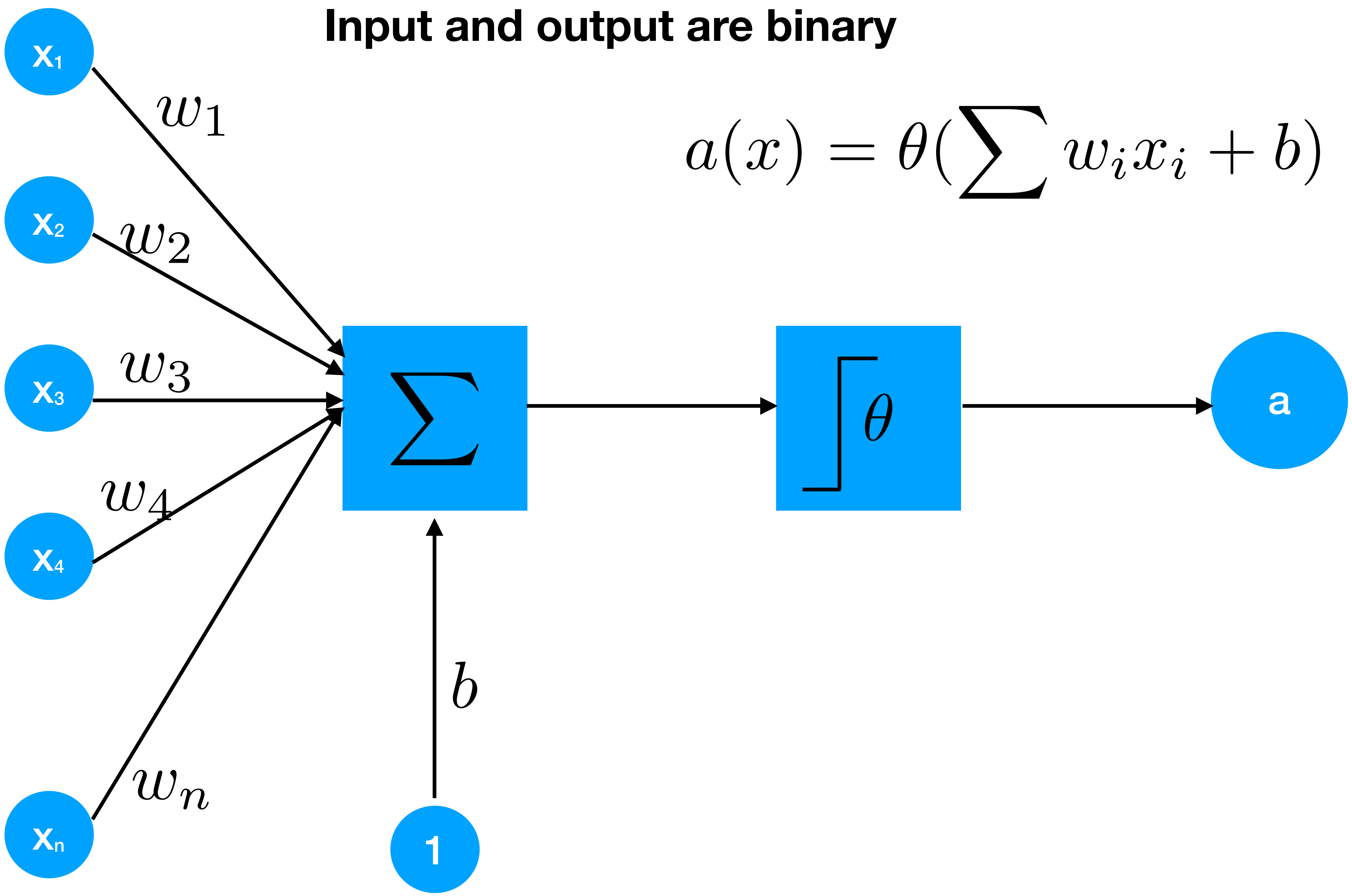
Biological Neural Networks

- Perceptron model.

Walter Pitts, Warren McCulloch (1943)

Input and output are binary

$$a(x) = \theta(\sum w_i x_i + b)$$



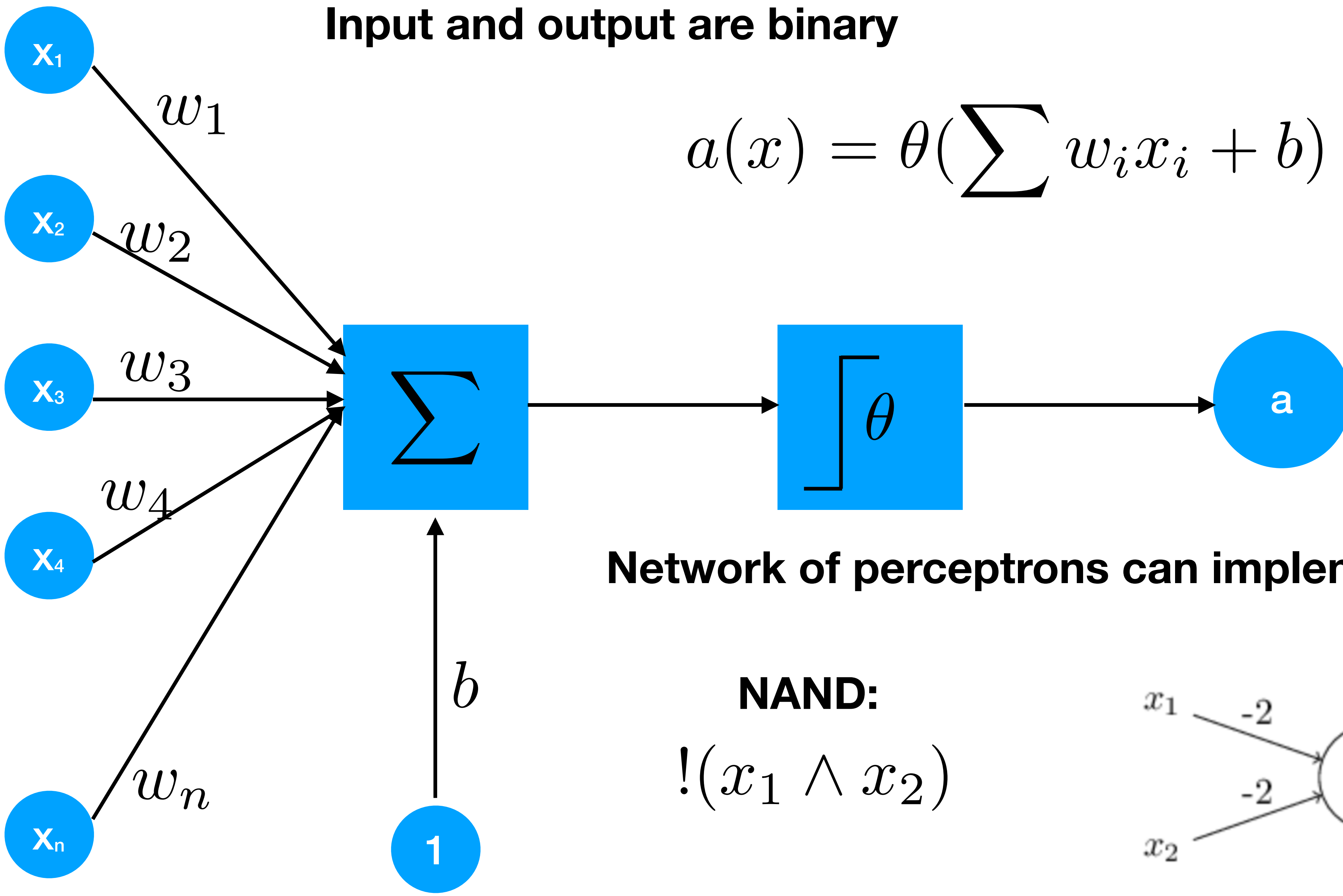
Biological Neural Networks

- Perceptron model.

Walter Pitts, Warren McCulloch (1943)

Input and output are binary

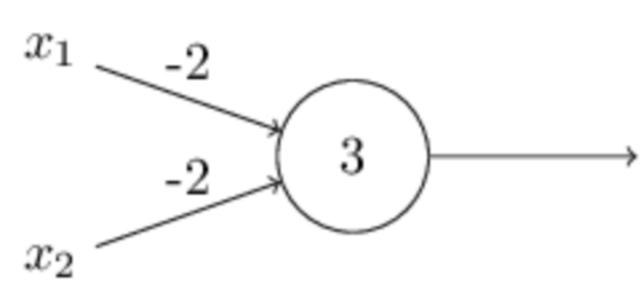
$$a(x) = \theta(\sum w_i x_i + b)$$



Network of perceptrons can implement binary logic

NAND:

$$!(x_1 \land x_2)$$



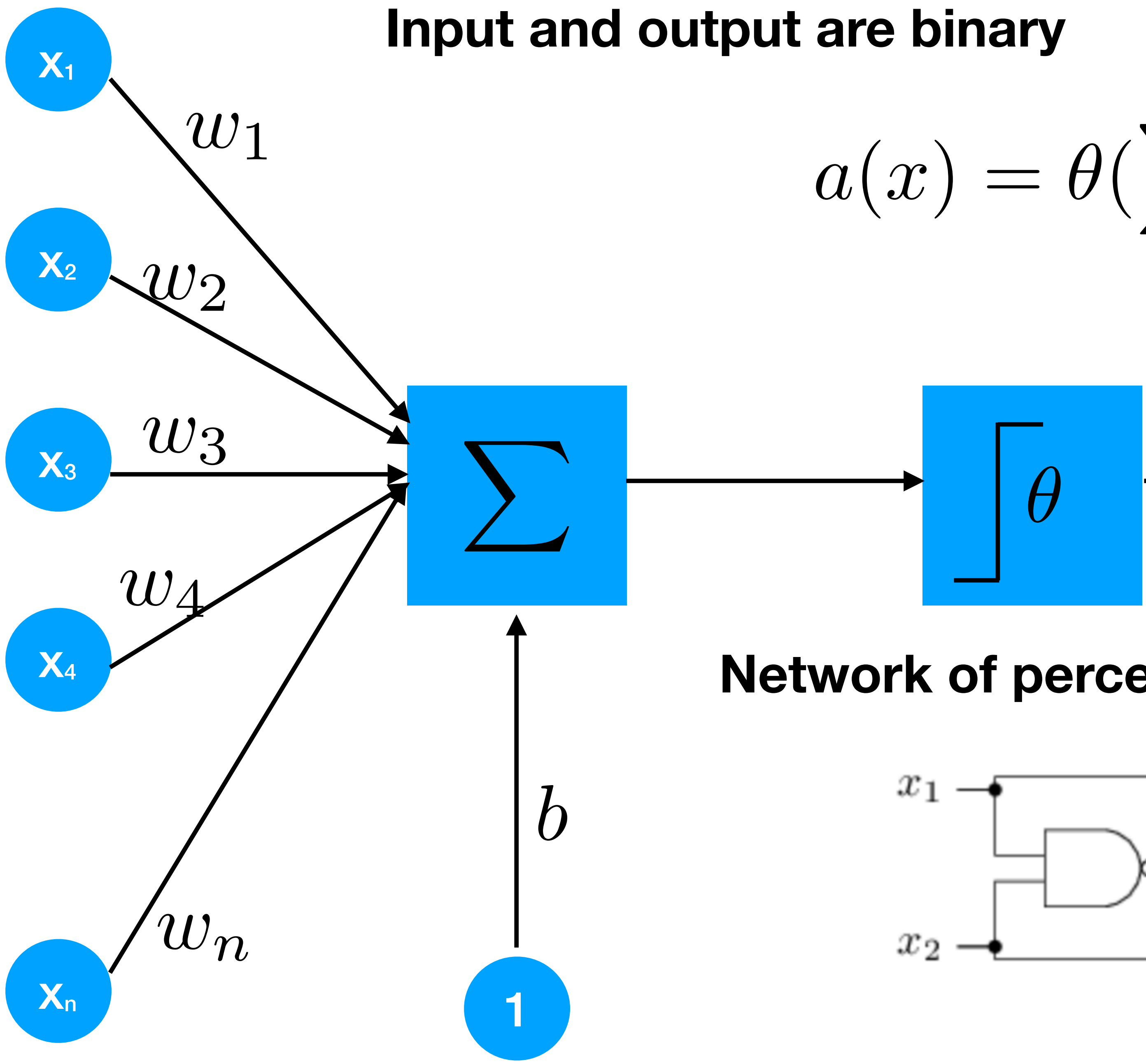
Biological Neural Networks

- Perceptron model.

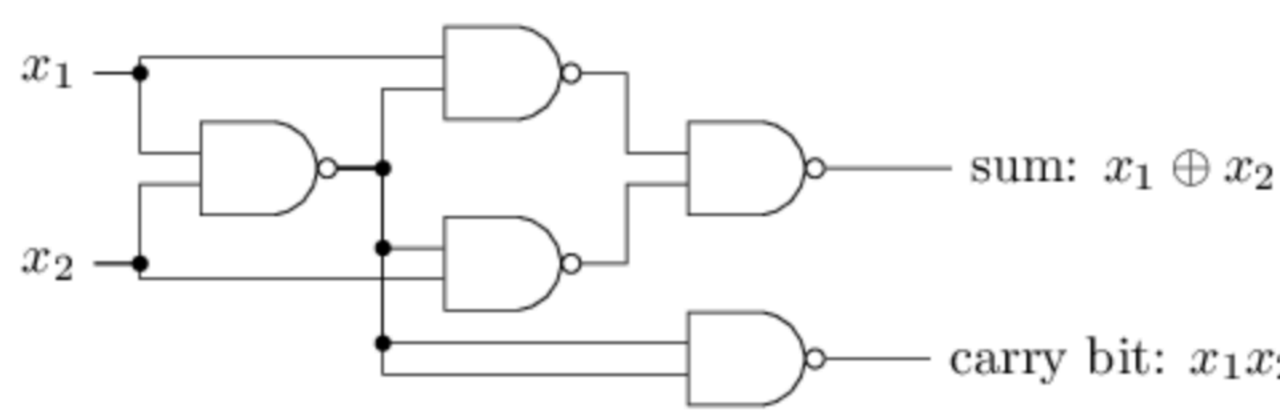
Walter Pitts, Warren McCulloch (1943)

Input and output are binary

$$a(x) = \theta(\sum w_i x_i + b)$$



Network of perceptrons can implement binary logic



Artificial Neural Networks

- Perceptron model in machine learning

$$a(x) = \theta(\sum w_i x_i + b)$$

Does it have anything to do with machine learning?

Artificial Neural Networks

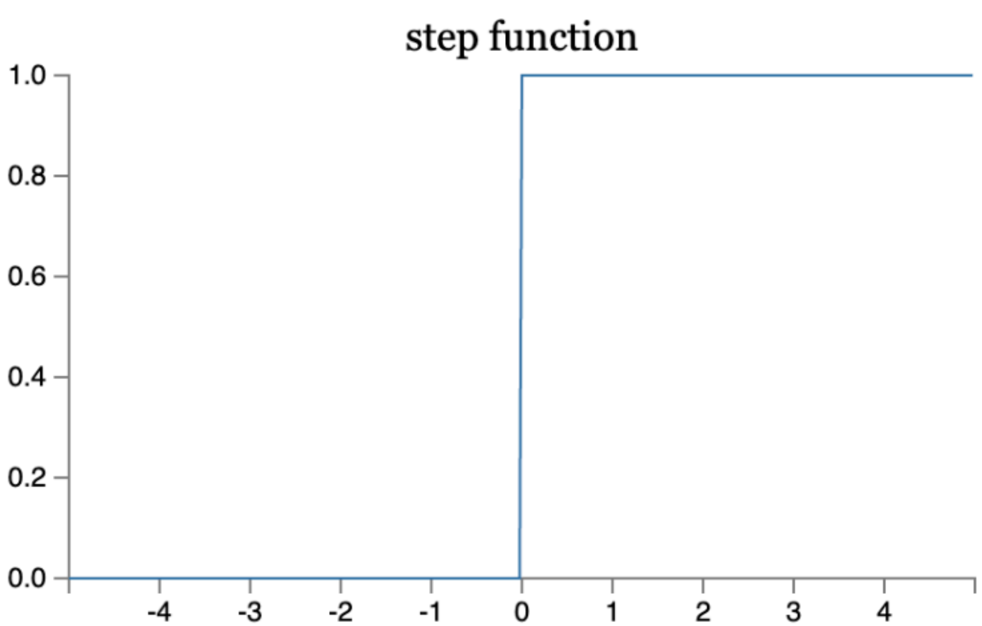
- Sigmoid neuron

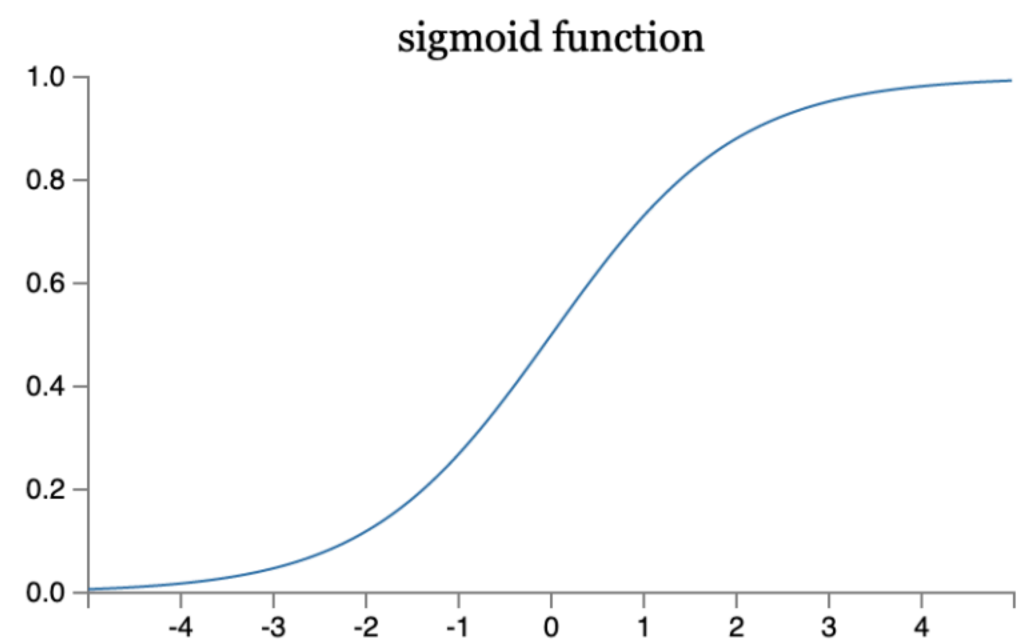
$$a(x) = \theta(\sum w_i x_i + b)$$

We want to train the weights.

- **continuous input/output instead of binary**
- **replace step by a smooth function**

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$





- **define lost (cost) function**

$$C(w,b) = \frac{1}{2n}\sum_x (y(x) - a)^2$$

note: if we used $\sigma(z) = z$
we would get linear regression

Artificial Neural Networks

- Sigmoid neuron training

$$a(x) = \sigma(wx + b)$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

$$C(w, b) = \frac{1}{2n} \sum_x (y(x) - a)^2$$

- Gradient calculation

$$\frac{\partial C}{\partial w_i} = \frac{1}{n} \sum_x (a(x) - y) \sigma'(wx + b) x_i$$

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (a(x) - y) \sigma'(wx + b)$$

$$\sigma' = \frac{e^{-z}}{(1 + e^{-z})^2} = \sigma(z)(1 - \sigma(z))$$

Artificial Neural Networks

- Sigmoid neuron training

$$a(x) = \sigma(wx + b)$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

$$C(w, b) = \frac{1}{2n} \sum_x (y(x) - a)^2$$

- Gradient calculation

$$\frac{\partial C}{\partial w_i} = \frac{1}{n} \sum_x (a(x) - y) \sigma'(wx + b) x_i$$

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (a(x) - y) \sigma'(wx + b)$$

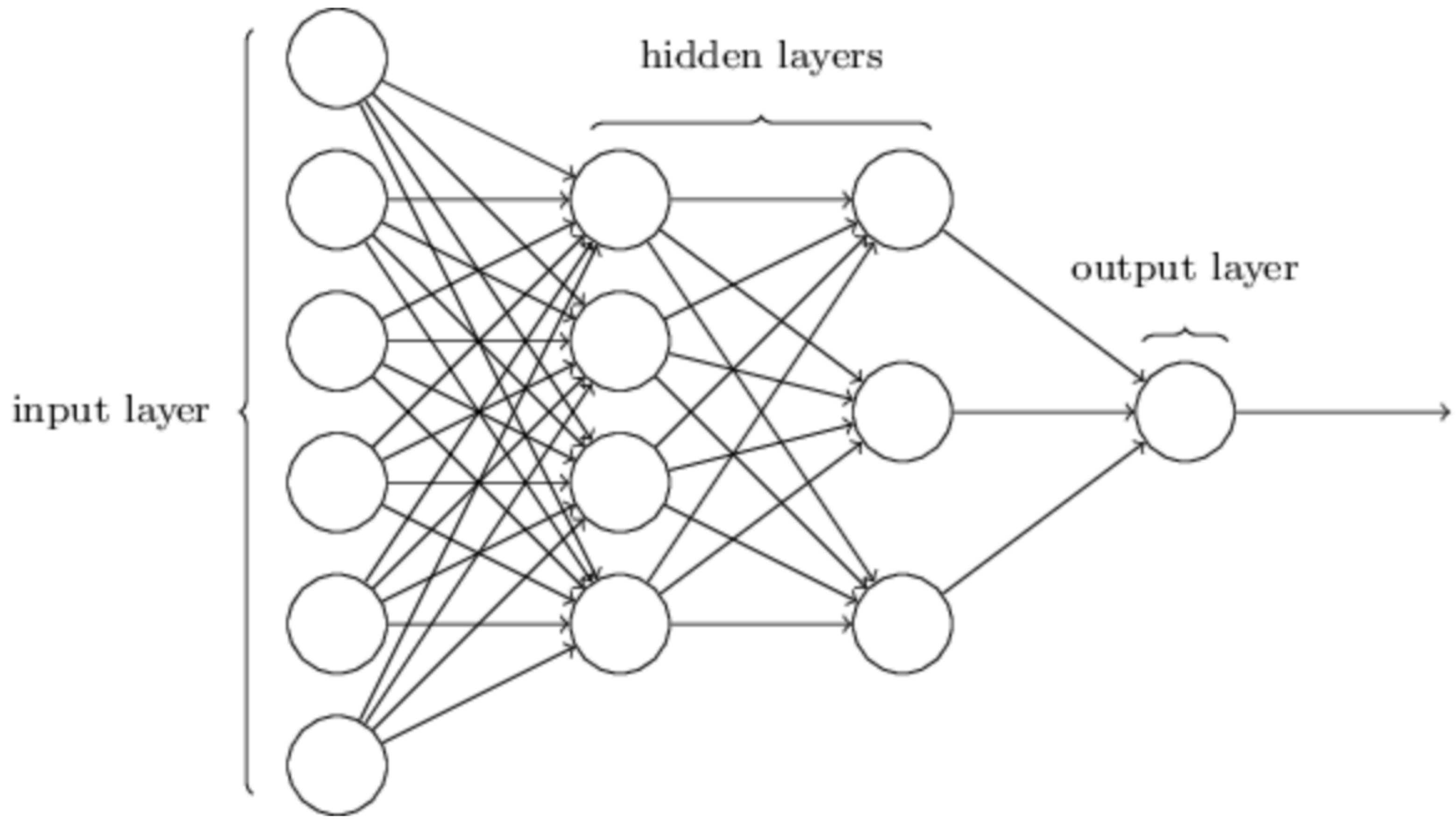
$$\sigma' = \frac{e^{-z}}{(1 + e^{-z})^2} = \sigma(z)(1 - \sigma(z))$$

exercise:

Train sigmoid neuron to predict sin(x) for x in [0,1]

Artificial Neural Networks

- Multilayer Perceptron



Signal propagation: $$a_j^l = \sigma(z_j^l), \;\; z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

Vector form: $$a^l = \sigma(w^l a^{l-1} + b^l)$$

Theorem (K. Hornik, 1991): ANY continuous function can be approximated with ANY precision by MLP

Time to open jupyter notebook

Loss function gradient calculation

- finite difference approximation:

$\delta C / dw^l_{jk} \simeq C(w + \delta w^l_{jk}) / \delta w^l_{jk}$ requires calculation of loss $C(w)$

N+1 times, where *N* is number of free model parameters (weights and biases)

- Backpropagation algorithm - fast loss function derivative calculation.
  - known since early 60s

The main idea: user '*error at layer l*' $\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$ **where** $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$

to calculate gradients $\partial C / \partial w^l_{jk}$ **and** $\partial C / \partial b^l_j$ **iteratively**

Back-propagation algorithm

Forward pass:

$$a^l_j = \sigma(z^l_j), \;\; z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$$

Cost:

$$C(w,b) = \frac{1}{2n}\sum_x (y(x) - a^L)^2$$

1. calculate the error in the last layer δ^L_j using the chain rule

$$\delta^L_j = \sum_k \frac{\partial C}{\partial a^L_k}\frac{\partial a^L_k}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j),$$

2. calculate the error in the intermediate layer δ^l_j using the chain rule

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} = \sum_k \frac{\partial C}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_j} = \sum_k \frac{\partial z^{l+1}_k}{\partial z^l_j}\delta^{l+1}_k$$

$$z^{l+1}_k = \sum_j w^{l+1}_{kj}\sigma(z^l_j) + b^{l+1}_k$$

$$\frac{\partial z^{l+1}_k}{\partial z^l_j} = w^{l+1}_{kj}\sigma'(z^l_j)$$

$$\delta^l_j = \sum_k w^{l+1}_{kj}\delta^{l+1}_k\sigma'(z^l_j)$$ - recurrent expression

3. express $\delta C/\delta b^l_j$ and $\delta C/\delta w^l_{jk}$ via δ^l_j:

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_i \frac{\partial C}{\partial z^l_i}\frac{\partial z^l_i}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j}a^{l-1}_k = \delta^l_j a^{l-1}_k$$

$$\frac{\partial C}{\partial b^l_j} = \sum_k \frac{\partial C}{\partial z^l_k}\frac{\partial z^l_k}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j} = \delta^l_j$$

Time to open jupyter notebook

What we have learned

Cross-entropy loss:

$$C = -\frac{1}{n}\sum_x \left[y \ln a^L + (1-y)\ln(1-a^L)\right]$$

helps to avoid gradient vanishing for sigmoid neurons

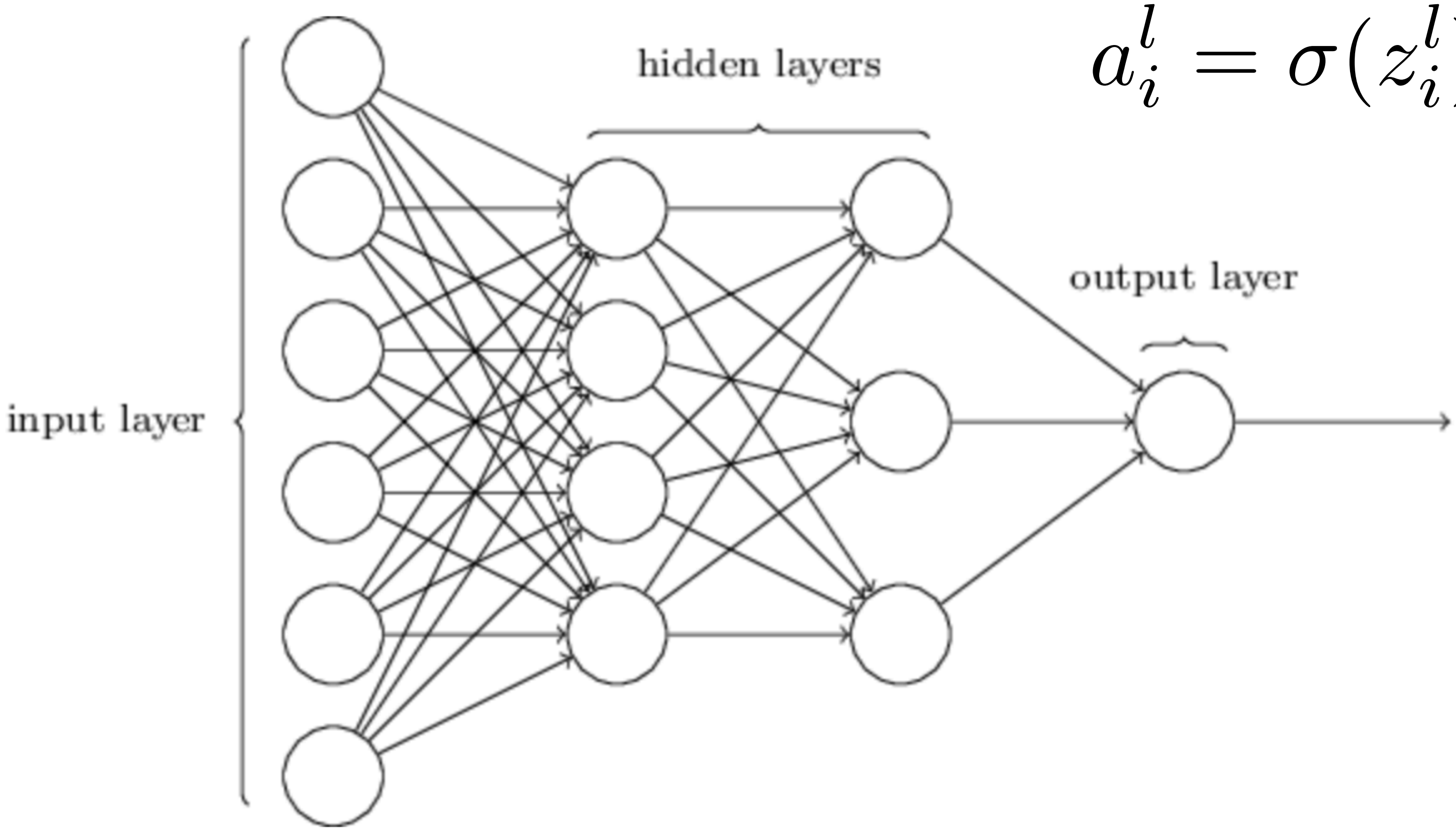
Classification

- Two classes: $y = \begin{cases} 0 & \text{for class } 0 \\ 1 & \text{for class } 1 \end{cases}$
  - $a^L = p_1$, interpreted as probability of sample belonging to class 1
- N classes: $y_i = \begin{cases} 0 & \text{for class other than } i \\ 1 & \text{for class } i \end{cases}$
  - use N output neurons
  - $C = -\frac{1}{n} \sum_x \sum_i \left[y_i \ln a_i^L + (1 - y_i) \ln(1 - a_i^L) \right]$ $\qquad \sum_i a_i^L \neq 1$

Softmax layer:

$$a_i^L = \frac{e^{z_i^L}}{\sum_k e^{z_k^L}} \qquad \sum_i a_i^L = 1 \qquad a_i^L = p_i$$

Initializing weights



$$a_i^l = \sigma(z_i^l), \;\; z_i^l = \sum_k w_{ik}^l a_k^{l-1} + b_i^l$$

Assume $|a_i^{l-1}| \simeq 1$ N^{l-1} **- number of neurons in layer** $l-1$

If weights $w_{ik}^l = \pm 1$ **then** $|z_i^l| \simeq \sqrt{N^{l-1}}$

$$|w_{ik}^l| \simeq \frac{1}{\sqrt{N^l}}\,, b^l = 0$$ **- good starting point**

Practical Task
Classification of gamma-ray sources

Source: **FERMI LAT 3FGL catalog**
Task: **use source features to predict source class (discriminate between blazars and pulsars)**
Data: **~3000 objects ~1000 of which are not identified**

column description:

1. **object name (3FGL prefix omitted)**
2. **equatorial coordinates: Right Ascencion, deg**
3. **equatorial coordinates: Declination, deg**

4-29. spectral and variability parameters

30. Source type code or NULL for unidentified sources.
Blazars: bll,BLL,bcu,BCU,fsrq,FSRQ.
Pulsars: psr, PSR.