

Lecture 5

Cezar Ionescu

15/06/2019

DEPARTMENT FOR
CONTINUING
EDUCATION



Administrative

- Homework from 08/06/2019 due now!
- Please complete and hand in the declarations of authorship.

Questions?

Solution to homework from lecture 2

Apply Bayes' theorem to answer the following question (Elmer Mode 1966, page 53):

A class in advanced mathematics contains 10 juniors, 30 seniors, and 10 graduate students. Three of the juniors, 10 of the seniors, and 5 of the graduate students received an A in the course. If a student is chosen at random from this class and is found to have earned an A, what is the probability that he is a graduate student?

Solution to homework from lecture 2

$\Omega = \{\text{student}_1, \dots, \text{student}_{50}\}$, Event = $\mathbb{P}(\Omega)$

$p(X) = \text{card}(X)/\text{card}(\Omega)$

Events of interest:

$G = \{x \mid x \in \Omega, x \text{ is a graduate student}\}$

$A = \{x \mid x \in \Omega, x \text{ has earned an } A\}$

Solution to homework from lecture 2

We want $p(G | A)$. By Bayes' theorem

$$p(G | A) = p(A | G) * p(G) / p(A)$$

$$p(A) = \text{card}(A) / \text{card}(\Omega) = 18/50$$

$$p(G) = \text{card}(G) / \text{card}(\Omega) = 10/50$$

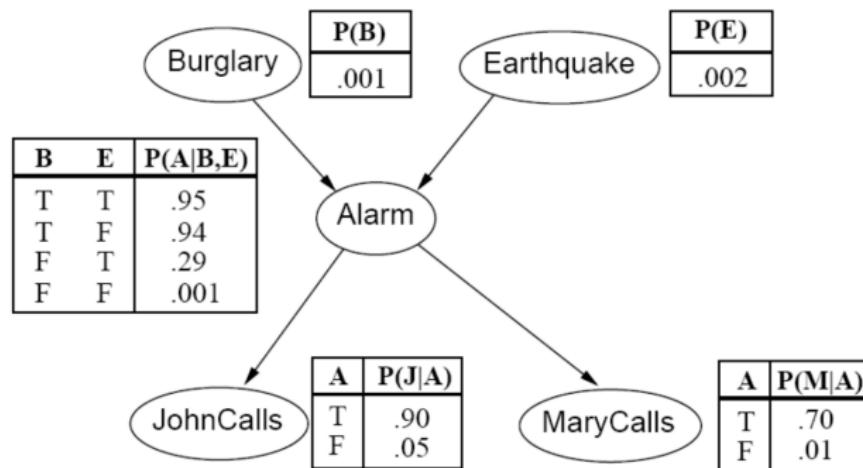
$$p(A | G) = p(A \cap G) / p(G) = \text{card}(A \cap G) / \text{card}(G) = 5/10$$

Therefore

$$p(G | A) = 5/18 = 0.278$$

Solution to homework from lecture 3

(based on Nilsson, 19.4, page 340) Consider the following Bayesian network:



Compute $p(\neg J, \neg M, A, B, E)$. This is the probability that there is both an earthquake and a burglary, the alarm rings, but neither John nor Mary call.

Solution to homework from lecture 3

$$p(\neg J, \neg M, A, B, E)$$

=

$$p(\neg J \mid \neg M \cap A \cap B \cap E) * p(\neg M \mid A \cap B \cap E) * \\ p(A \mid B \cap E) * p(B \mid E) * p(E)$$

=

$$p(\neg J \mid A) * p(\neg M \mid A) * p(A \mid B \cap E) * p(B) * p(E)$$

=

$$0.1 * 0.3 * 0.95 * 0.001 * 0.002$$

$$= 5.7 * 10^{-8}$$

McCulloch & Pitts

- McCulloch & Pitts 1943 *A Logical Calculus of the Ideas Immanent in Nervous Activity*:
“neural events and the relations among them can be treated by means of propositional logic”
- The MC & P neuron had a number of boolean inputs, some positive and others negative. The neuron was activated if the number of active positive inputs was greater than the number of active negative inputs plus a “threshold”:

```
mc_p_neuron : N -> ({0, 1}^n, {0, 1}^m) -> {0, 1}
mc_p_neuron θ (pos, neg) = if sum neg > θ then 0
                           else if sum pos ≥ θ then 1
                           else 0
```

Examples

130

LOGICAL CALCULUS FOR NERVOUS ACTIVITY

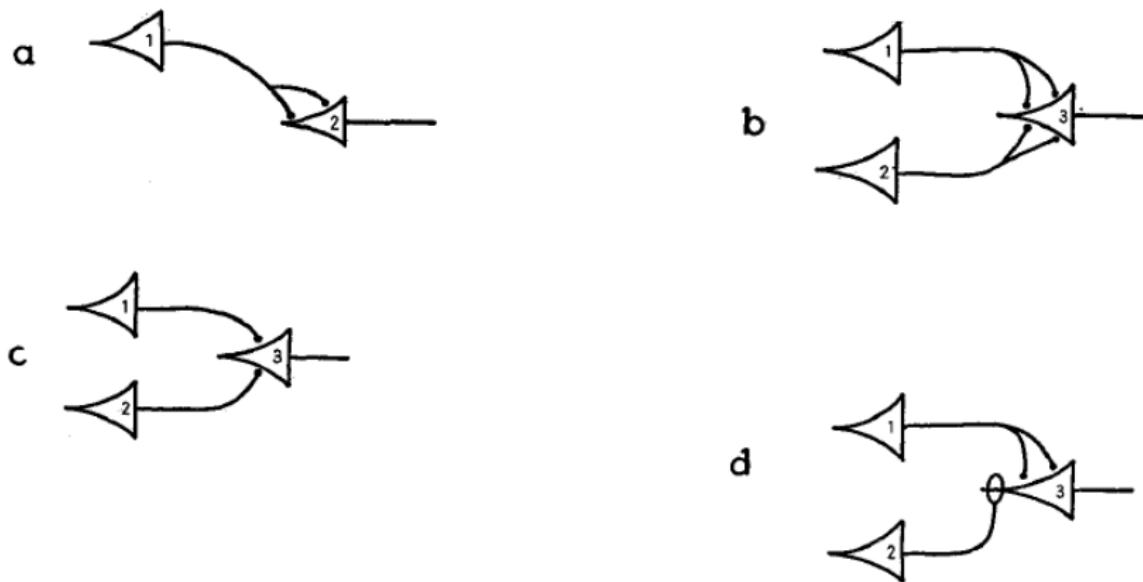


Figure 1: McCulloch & Pitts neuronal networks

Logic with McCulloch & Pitts neurons

Logical functions:

not : {0, 1} -> {0, 1}

not x = mc_p_neuron 0 ([] , [x])

and : ({0, 1}, {0, 1}) -> {0, 1}

and (x, y) = mc_p_neuron 2 ([x, y], [])

"Exclusive or" with McCulloch & Pitts neurons

```
xor : ({0, 1}, {0, 1}) -> {0, 1}
xor (x, y) = mc_p_neuron 2 ([case1, case1, case2, case2], [])
where
  case1 = mc_p_neuron 2 ([x, x], [y])
  case2 = mc_p_neuron 2 ([y, y], [x])
```

"Exclusive or" with McCulloch & Pitts neurons

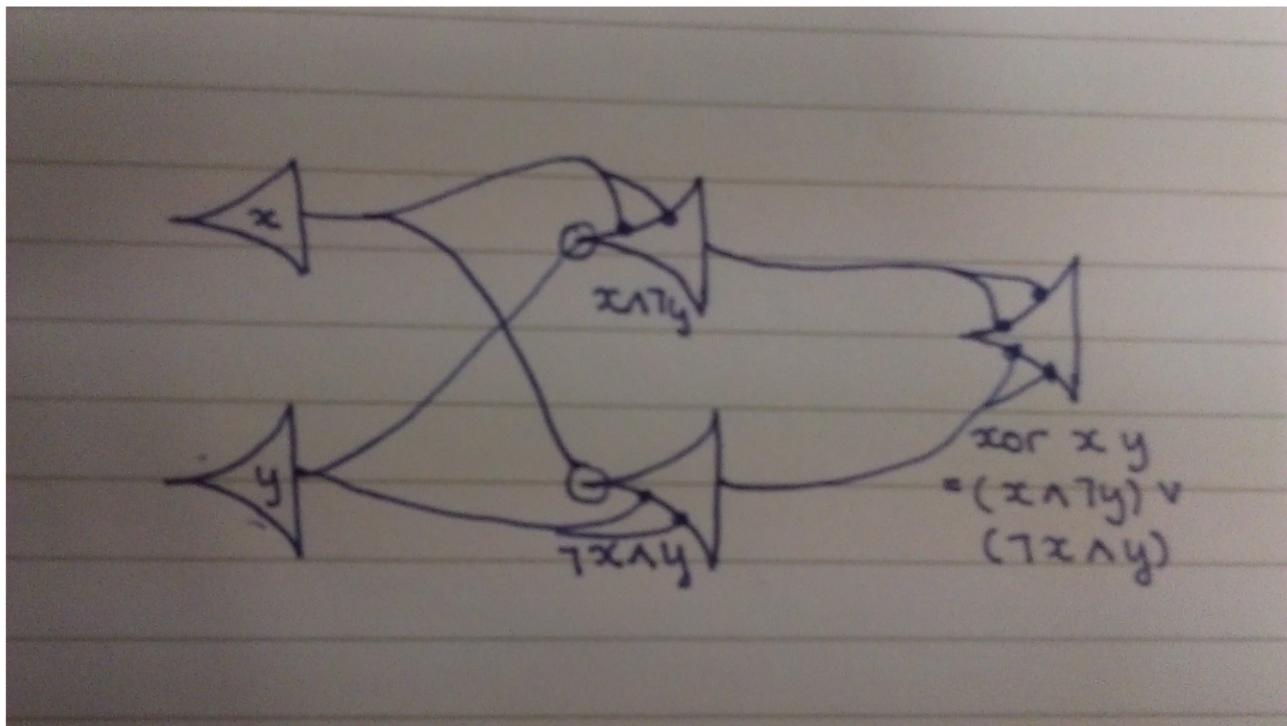


Figure 2: Logical "xor" implemented with McCulloch & Pitts neurons

Perceptrons

- Frank Rosenblatt 1957
- A perceptron had real-valued inputs and binary outputs. The output was a step function of the weighted sum of these inputs

perceptron : $(\mathbb{R}^n, \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \{0, 1\}$

perceptron ($[w_1, \dots, w_n]$, θ) $[x_1, \dots, x_n] = \text{if } s \geq \theta \text{ then } 1$
 $\text{else } -1$

where $s = w_1 * x_1 + \dots + w_n * x_n$

Perceptrons

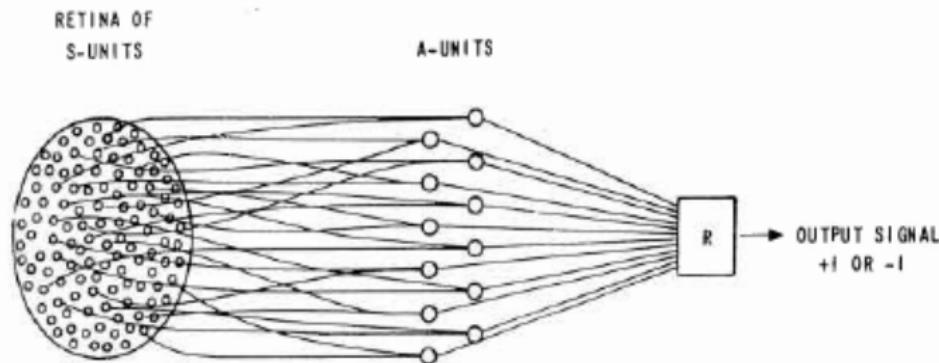


Figure 5 NETWORK ORGANIZATION OF A TYPICAL ELEMENTARY PERCEPTRON

Figure 3: Rosenblatt's perceptron (1961)

Logic with perceptrons

Logical functions:

not $x = \text{perceptron}([-1], 0) [x]$

and $(x_1, x_2) = \text{perceptron}([0.5, 0.5], 1) [x_1, x_2]$

Logic with perceptrons

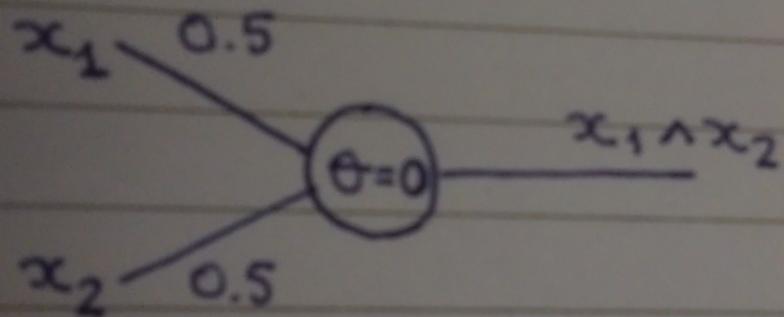


Figure 4: Logical “and” implemented with a perceptron

“Exclusive or” with perceptrons

```
xor (x1, x2) = perceptron ([1, 1], 0) [case1, case2]
```

where

```
case1 = perceptron ([0.5, -0.5], 0) [x1, x2]
```

```
case2 = perceptron ([-0.5, 0.5], 0) [x1, x2]
```

"Exclusive or" with perceptrons

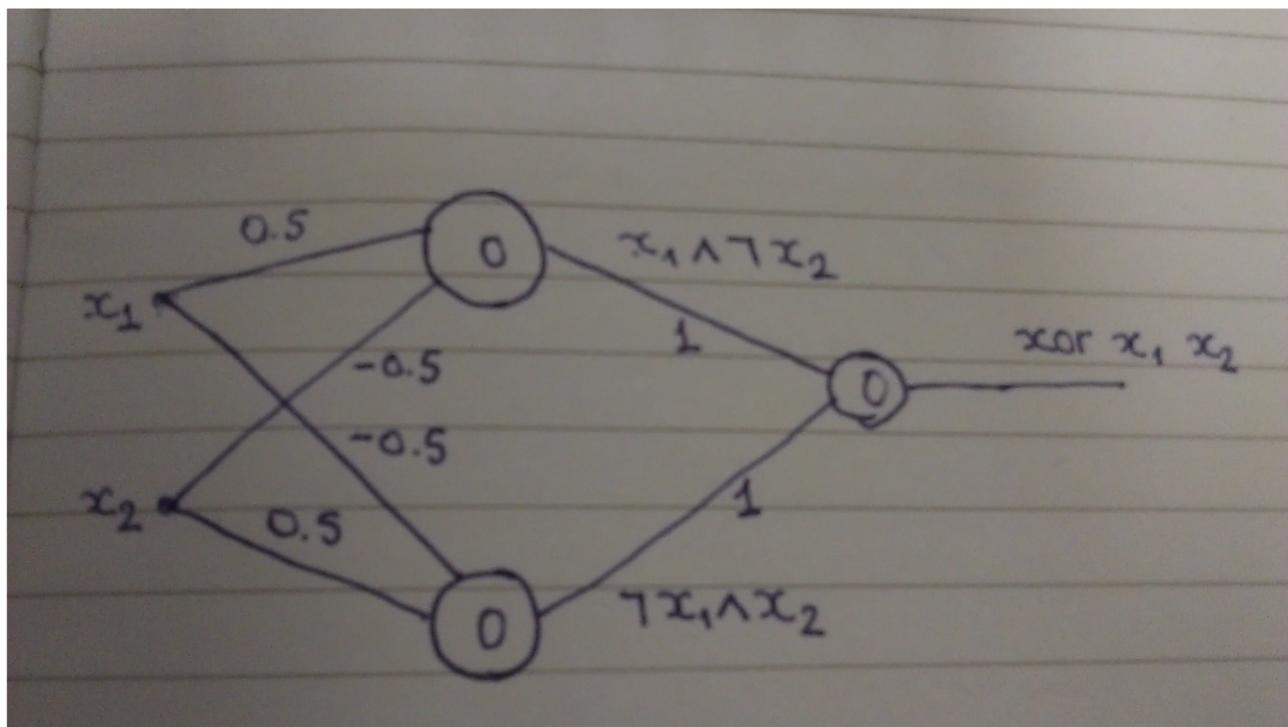


Figure 5: Logical "xor" implemented with perceptrons

Homework

Implement the “two out of three” function defined by:

```
two_of_three : ({0, 1}, {0, 1}, {0, 1}) -> {0, 1}
two_of_three (x, y, z)
| (1, 1, 0) = True
| (1, 0, 1) = True
| (0, 1, 1) = True
| otherwise = False
```

using

- a) McCulloch & Pitts neurons
- b) perceptrons

Perceptrons

The perceptron training rule:

$$w_i \leftarrow w_i + \eta * (t - o) * x_i$$

The threshold θ is treated as the weight of an always active input.

Gradient descent

Why does the perceptron training rule work?

“Naive” gradient descent:

$$x \leftarrow x - \eta * D f(x)$$

What is f in our case?

Error functions

$t - o, |t - o|, (t - o)^2, (t - o)^4, \dots$

Bayesian learning and error minimisation

data: $(x_1, t_1), \dots, (x_n, t_n)$, $x_i \in \mathbb{R}^m$, $t_i \in \mathbb{R}$

hypothesis: $w \in \mathbb{R}^m$

$$\begin{aligned} h_{\text{map}} &= \operatorname{argmax} p(h \mid d) \\ &= \operatorname{argmax} p(d \mid h) * p(h) / p(d) \\ &= \operatorname{argmax} p(d \mid h) * p(h) \\ &\quad \text{-- assume } p(h) = \text{const} \\ &= \operatorname{argmax} p(d \mid h) \\ &= h_{\text{ml}} \end{aligned}$$

Bayesian learning and error minimisation

$$\begin{aligned} p(d \mid h) &= p((x_1, t_1), \dots, (x_n, t_n) \mid w) \\ &= p(x_1, t_1 \mid w) * \dots * p(x_n, t_n \mid w) \end{aligned}$$

$$\begin{aligned} h_m &= \operatorname{argmax} p(d \mid h) \\ &= \operatorname{argmax} \ln p(d \mid h) \\ &= \operatorname{argmax} \ln p(x_1, t_1 \mid w) + \dots + \ln p(x_n, t_n \mid w) \end{aligned}$$

Bayesian learning and error minimisation

Assume the $f(x_i, w)$ are normally distributed around the t_i , with the **same** σ :

$$p(x_i, t_i | w) = \frac{1}{\sqrt{2 * \pi * \sigma^2}} \exp \left(-\frac{(t_i - f(x_i, w))^2}{2 * \sigma^2} \right)$$

Therefore

$$\begin{aligned} \ln p(x_i, t_i | w) &= \ln \frac{1}{\sqrt{2 * \pi * \sigma^2}} - \frac{(t_i - f(x_i, w))^2}{2 * \sigma^2} \\ &= k - \frac{(t_i - f(x_i, w))^2}{2 * \sigma^2} \end{aligned}$$

Bayesian learning and error minimisation

$$\begin{aligned} h_m l &= \operatorname{argmax} \ln p(x_1, t_1 | w) + \dots + \ln p(x_n, t_n | w) \\ &= \operatorname{argmax} n * k - \sum (t_i - f(x_i, w))^2 / (2 * \sigma^2) \\ &= \operatorname{argmax} - \sum (t_i - f(x_i, w))^2 \\ &= \operatorname{argmin} \sum (t_i - f(x_i, w))^2 \end{aligned}$$

Bayesian learning and error minimisation

Therefore, the correct error to choose is the sum of squared errors...

...at least if:

- all hypothesis are equally likely a-priori,
- the errors are independent, and
- the errors have identical normal distributions.

Perceptrons and learning

- Using the learning rule, we can iteratively find the weights and threshold for any perceptron (such as the one implementing `and`).
- However, the learning rule does not apply to multi-layer perceptrons, such as the one needed for `xor`.

Linear separability

- Perceptrons can only classify linearly separable sets, but `xor` is not linearly separable.
- The 1969 book *Perceptrons* by Minsky and Pappert pointed out these facts, killing funding for neural networks for a decade or more.