Guest Lecture:
XOR Neural Nets

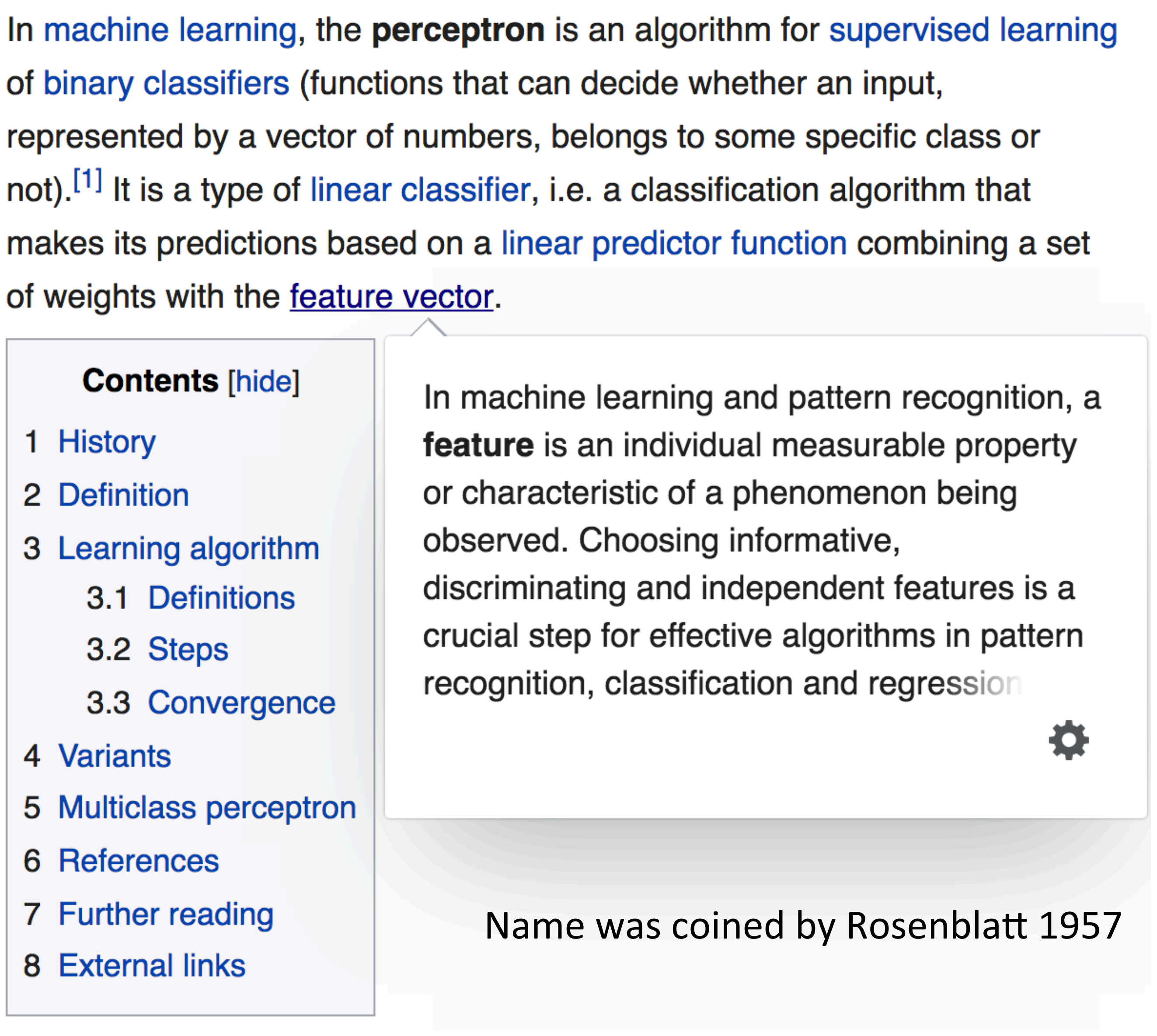
In Math 490 (Prof Kay Kirkpatrick)

“Mathematics of Machine Learning”

19 October 2018

George Francis, email: gfrancis@illinois.edu
Notes at http:new.math.uiuc.edu/TraskXOR

Perceptron (Wikipedia)



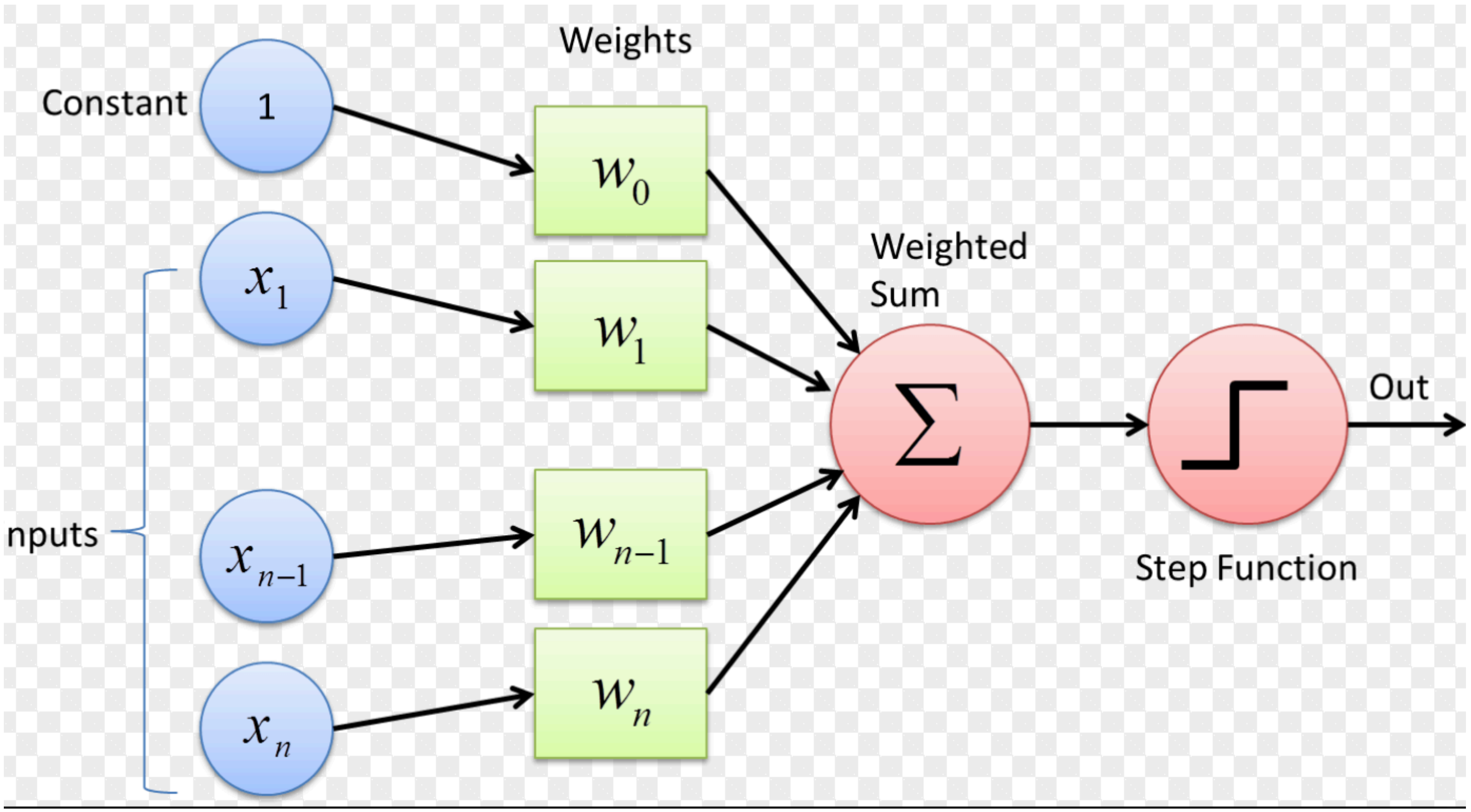
In machine learning, the **perceptron** is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not).[1] It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

Contents [hide]

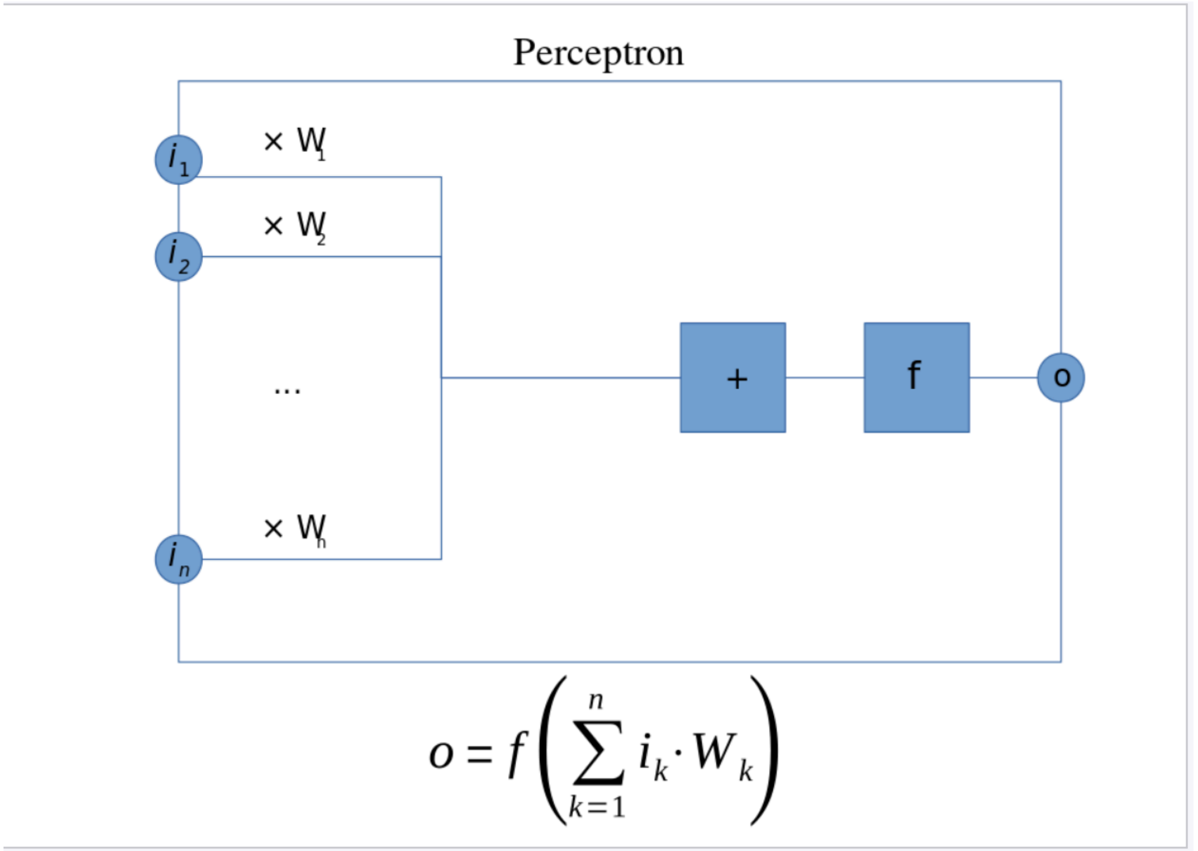
In machine learning and pattern recognition, a **feature** is an individual measurable property or characteristic of a phenomenon being observed. Choosing informative, discriminating and independent features is a crucial step for effective algorithms in pattern recognition, classification and regression

Name was coined by Rosenblatt 1957

Perceptron, a good graphic



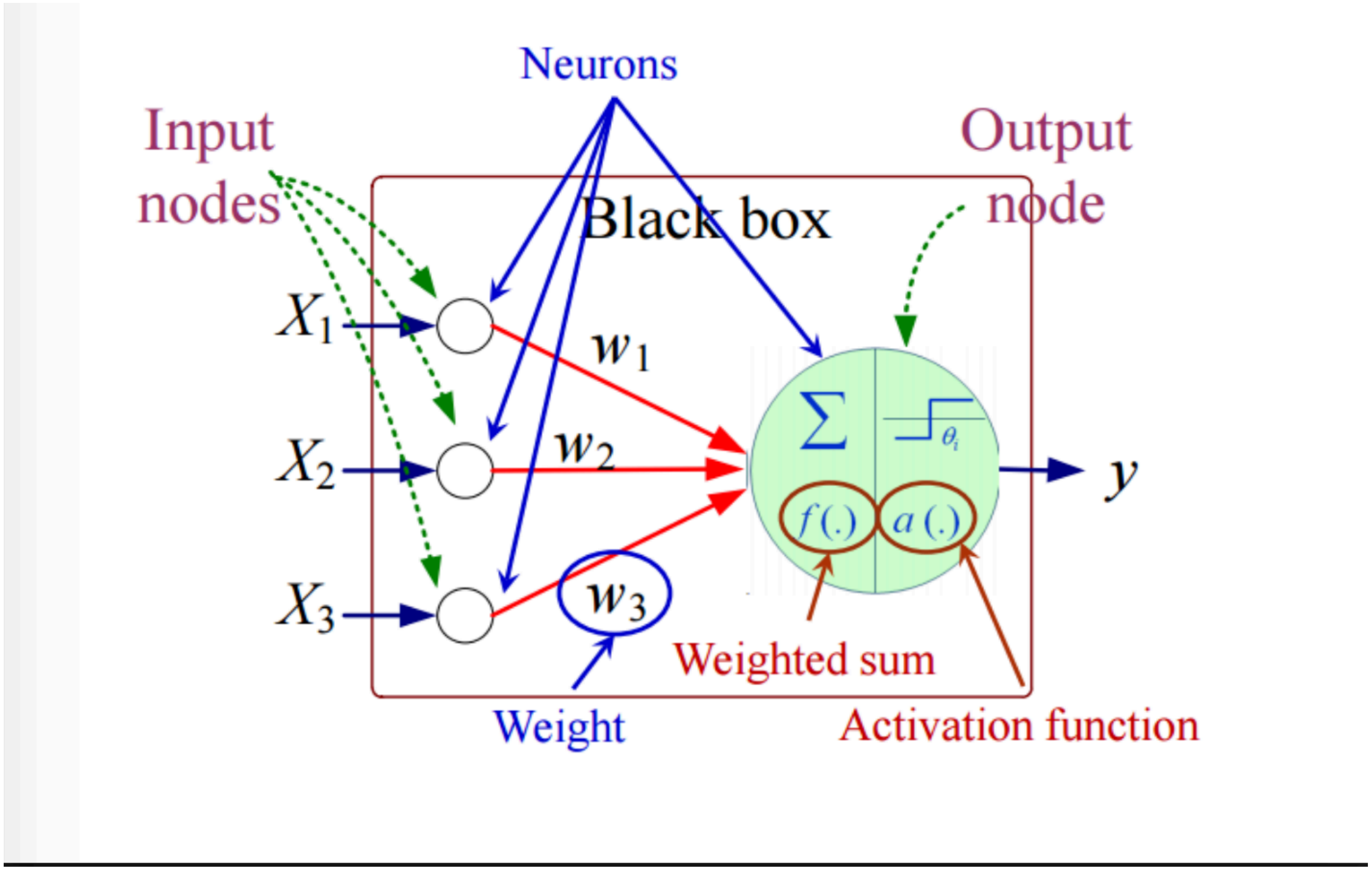
Another good figure (Wikipedia)



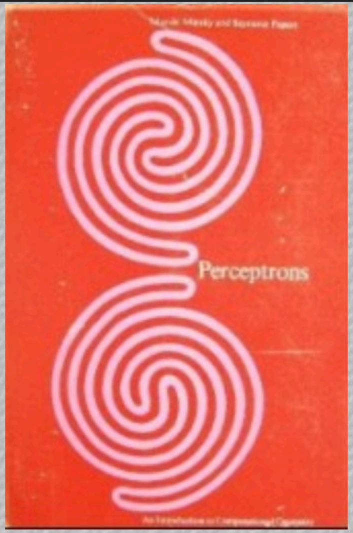
$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

The appropriate weights are applied to the inputs, and the resulting weighted sum passed to a function that produces the output o.

Perceptron, a bad figure



Cover of Minsky & Papert “Perceptrons” 1969, 1987



Cover of the 1972 Edition of *Perceptrons*
This book had a significant impact on the development of AI. Minsky and Papert 'proved' that single layer perceptrons could not distinguish on the basis of connectivit and hence could not diustinguish topologically the upper and lower figures.

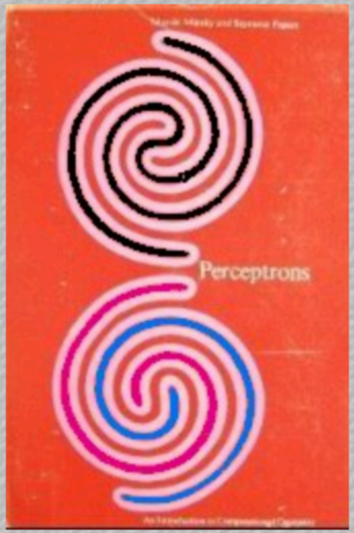
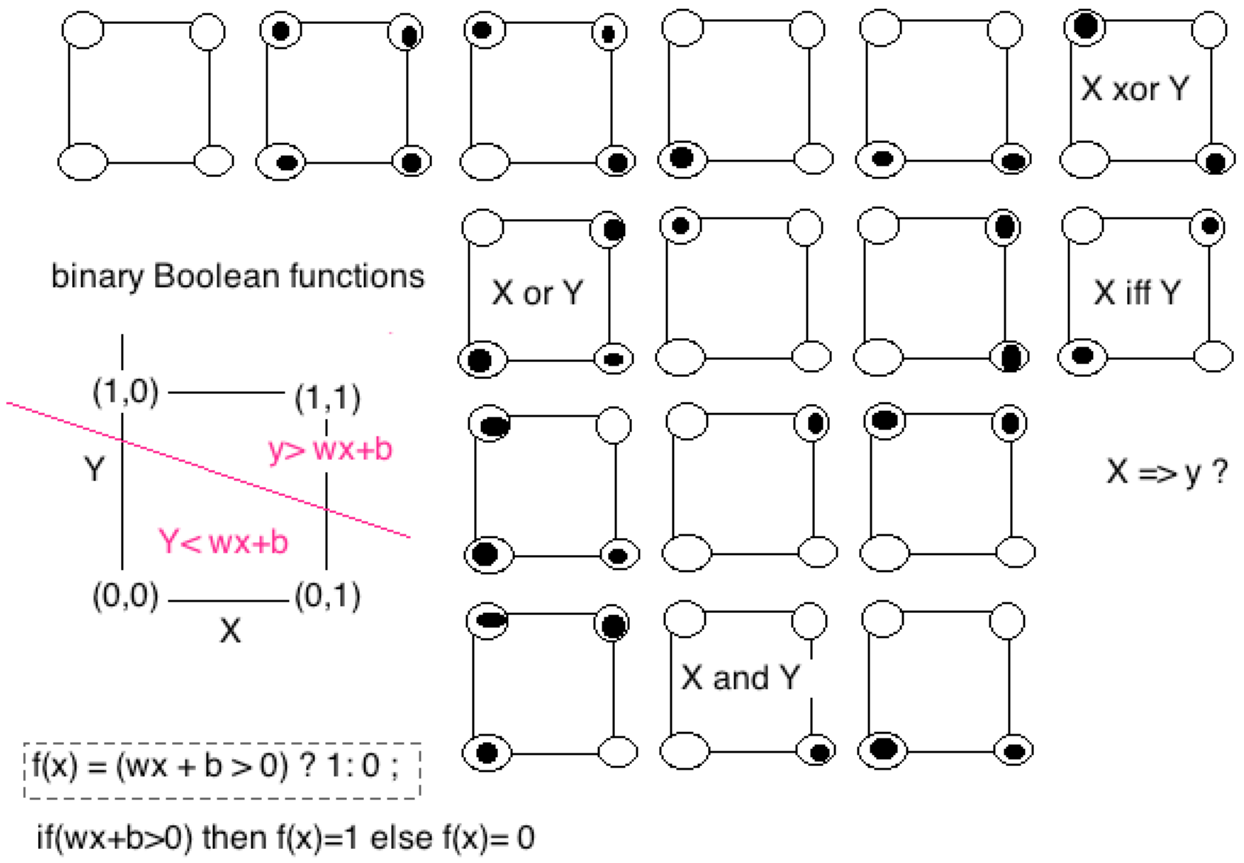


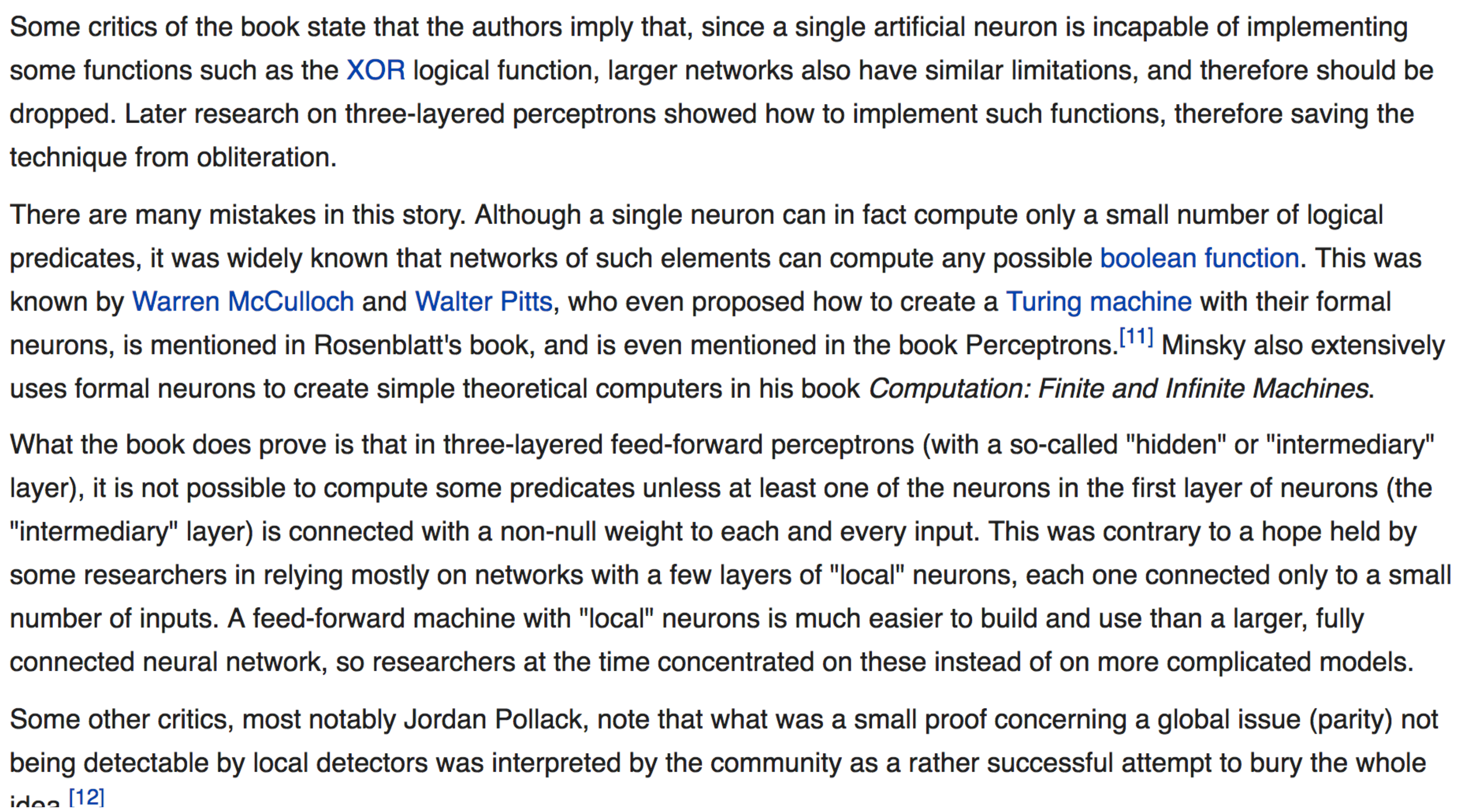
Image processed cover of the 1972 Edition of *Perceptrons*.
In the top figure the curves outline a single connected region [black pixels] while in lower figure there are two unconnected regions [red and blue pixels]

THE PERCEPTRON

Papert: Perceptrons can't learn XOR



The XOR-Affair (Wikipedia)



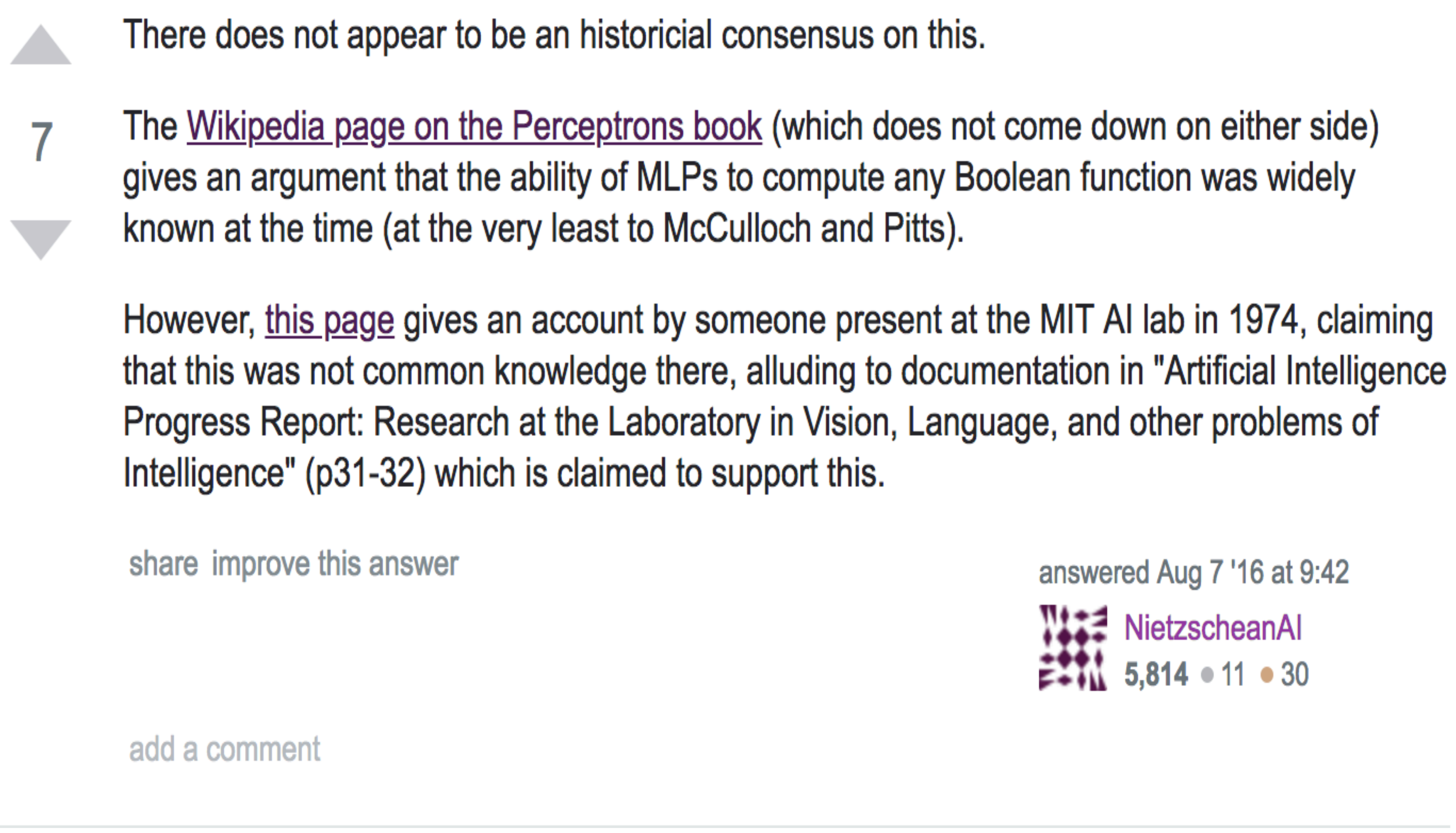
Some critics of the book state that the authors imply that, since a single artificial neuron is incapable of implementing some functions such as the XOR logical function, larger networks also have similar limitations, and therefore should be dropped. Later research on three-layered perceptrons showed how to implement such functions, therefore saving the technique from obliteration.

There are many mistakes in this story. Although a single neuron can in fact compute only a small number of logical predicates, it was widely known that networks of such elements can compute any possible boolean function. This was known by Warren McCulloch and Walter Pitts, who even proposed how to create a Turing machine with their formal neurons, is mentioned in Rosenblatt's book, and is even mentioned in the book Perceptrons.[11] Minsky also extensively uses formal neurons to create simple theoretical computers in his book *Computation: Finite and Infinite Machines*.

What the book does prove is that in three-layered feed-forward perceptrons (with a so-called "hidden" or "intermediary" layer), it is not possible to compute some predicates unless at least one of the neurons in the first layer of neurons (the "intermediary" layer) is connected with a non-null weight to each and every input. This was contrary to a hope held by some researchers in relying mostly on networks with a few layers of "local" neurons, each one connected only to a small number of inputs. A feed-forward machine with "local" neurons is much easier to build and use than a larger, fully connected neural network, so researchers at the time concentrated on these instead of on more complicated models.

Some other critics, most notably Jordan Pollack, note that what was a small proof concerning a global issue (parity) not being detectable by local detectors was interpreted by the community as a rather successful attempt to bury the whole idea.[12]

It’s never that simple (Stackoverflow)



7

There does not appear to be an historicial consensus on this.

The Wikipedia page on the Perceptrons book (which does not come down on either side) gives an argument that the ability of MLPs to compute any Boolean function was widely known at the time (at the very least to McCulloch and Pitts).

However, this page gives an account by someone present at the MIT AI lab in 1974, claiming that this was not common knowledge there, alluding to documentation in "Artificial Intelligence Progress Report: Research at the Laboratory in Vision, Language, and other problems of Intelligence" (p31-32) which is claimed to support this.

share improve this answer

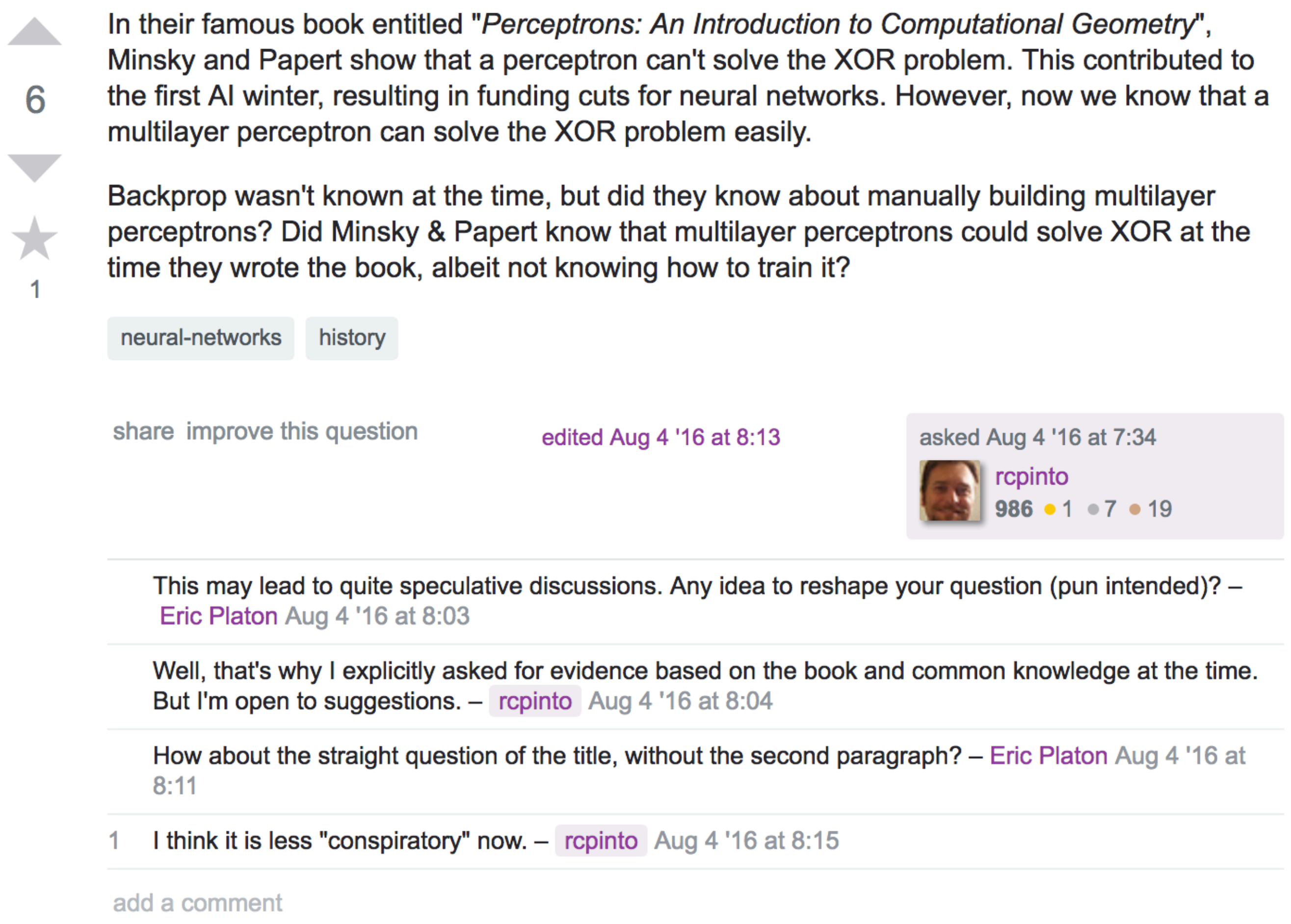
answered Aug 7 '16 at 9:42

NietzscheanAI

5,814 ● 11 ● 30

add a comment

Controversy (ai.stackexchange.com)



6

1

In their famous book entitled "*Perceptrons: An Introduction to Computational Geometry*", Minsky and Papert show that a perceptron can't solve the XOR problem. This contributed to the first AI winter, resulting in funding cuts for neural networks. However, now we know that a multilayer perceptron can solve the XOR problem easily.

Backprop wasn't known at the time, but did they know about manually building multilayer perceptrons? Did Minsky & Papert know that multilayer perceptrons could solve XOR at the time they wrote the book, albeit not knowing how to train it?

neural-networks history

share improve this question

edited Aug 4 '16 at 8:13

asked Aug 4 '16 at 7:34
rcpinto
986 ● 1 ● 7 ● 19

This may lead to quite speculative discussions. Any idea to reshape your question (pun intended)? – Eric Platon Aug 4 '16 at 8:03

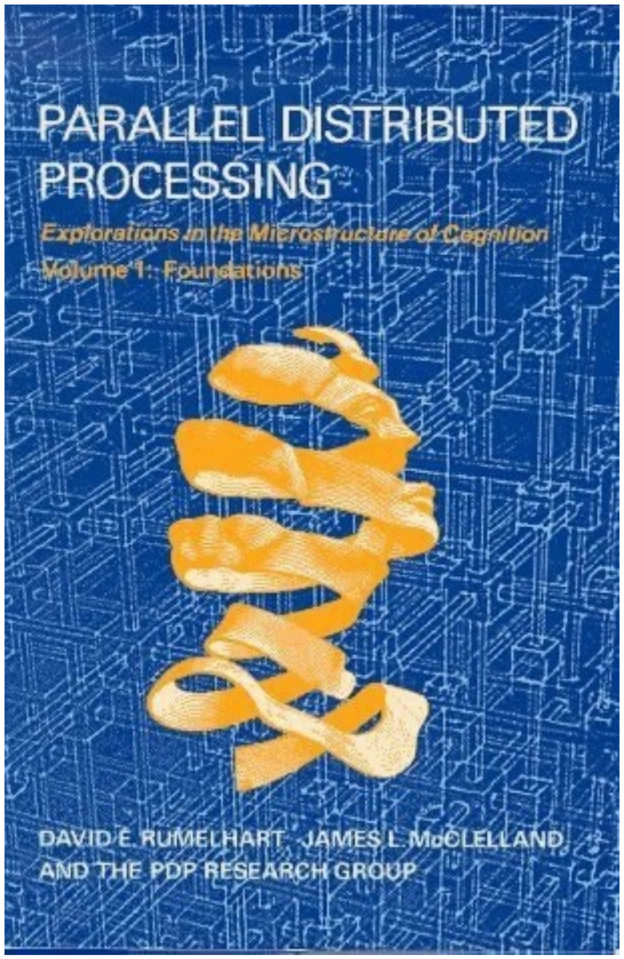
Well, that's why I explicitly asked for evidence based on the book and common knowledge at the time. But I'm open to suggestions. – rcpinto Aug 4 '16 at 8:04

How about the straight question of the title, without the second paragraph? – Eric Platon Aug 4 '16 at 8:11

1 I think it is less "conspiratory" now. – rcpinto Aug 4 '16 at 8:15

add a comment

Rumelhart, Hart, MacClelland 1986 (End of the Controversy)

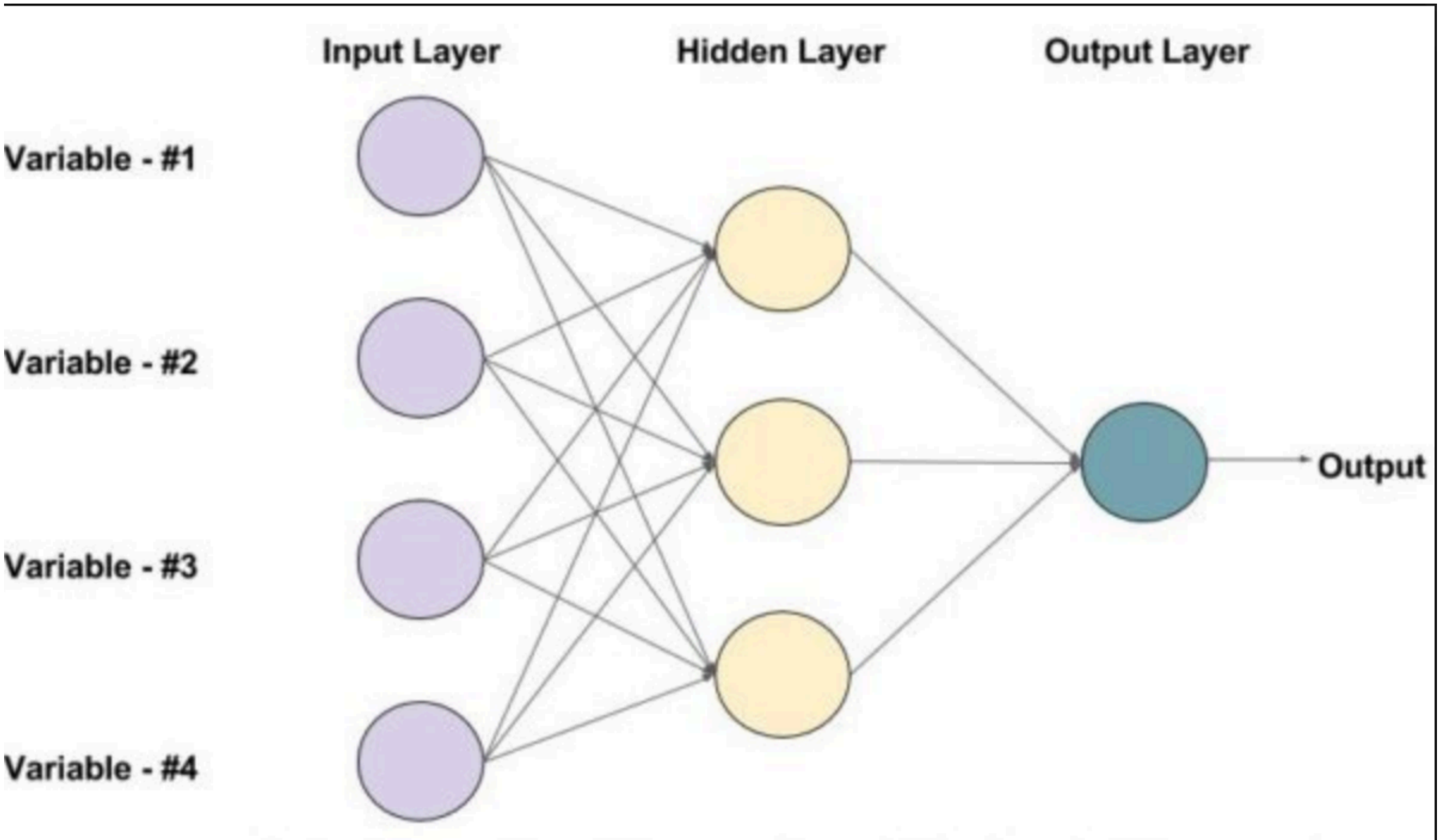

PARALLEL DISTRIBUTED PROCESSING

Explorations in the Microstructure of Cognition

Volume 1: Foundations

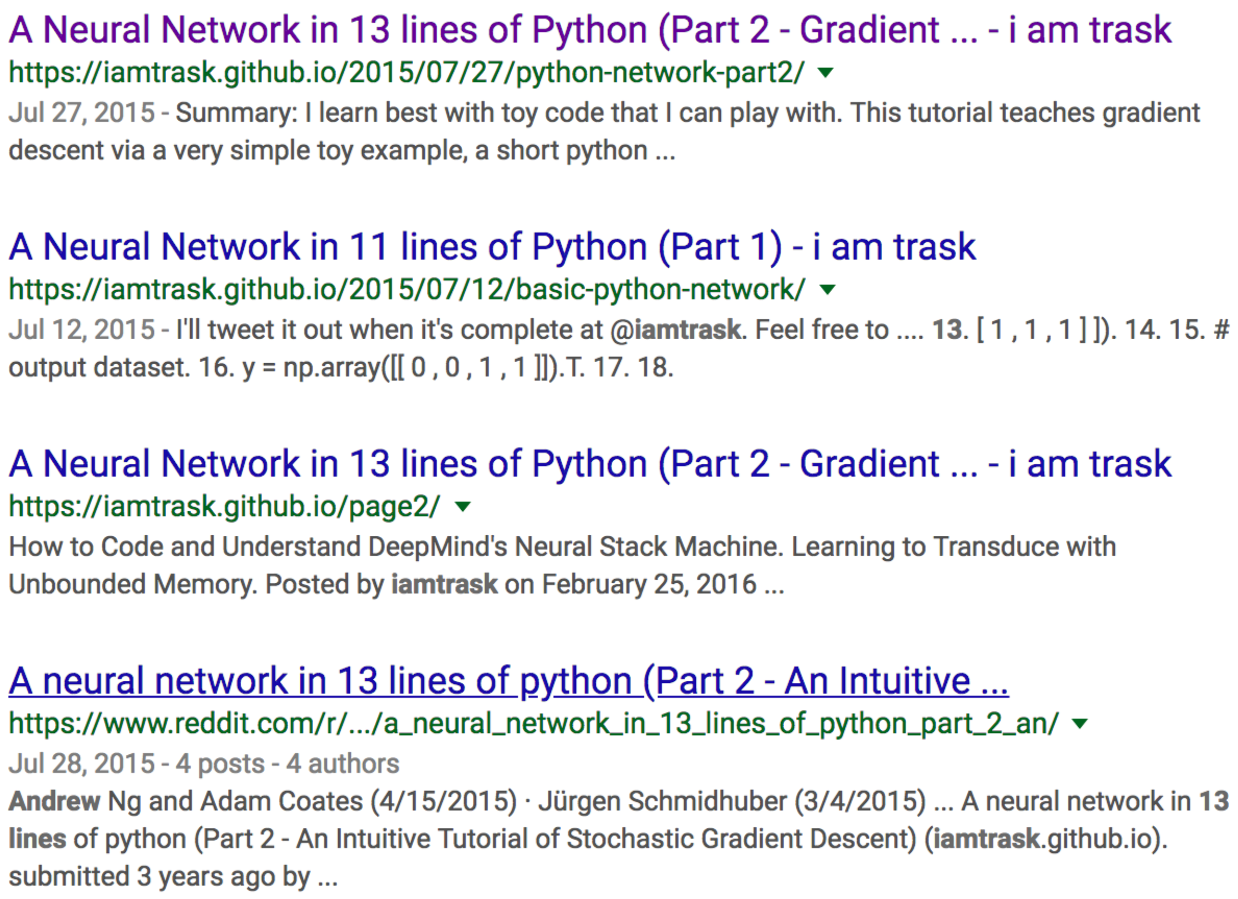
DAVID E. RUMELHART JAMES L. McCLELLAND AND THE PDP RESEARCH GROUP

Not a Perceptron



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

Andrew Trask

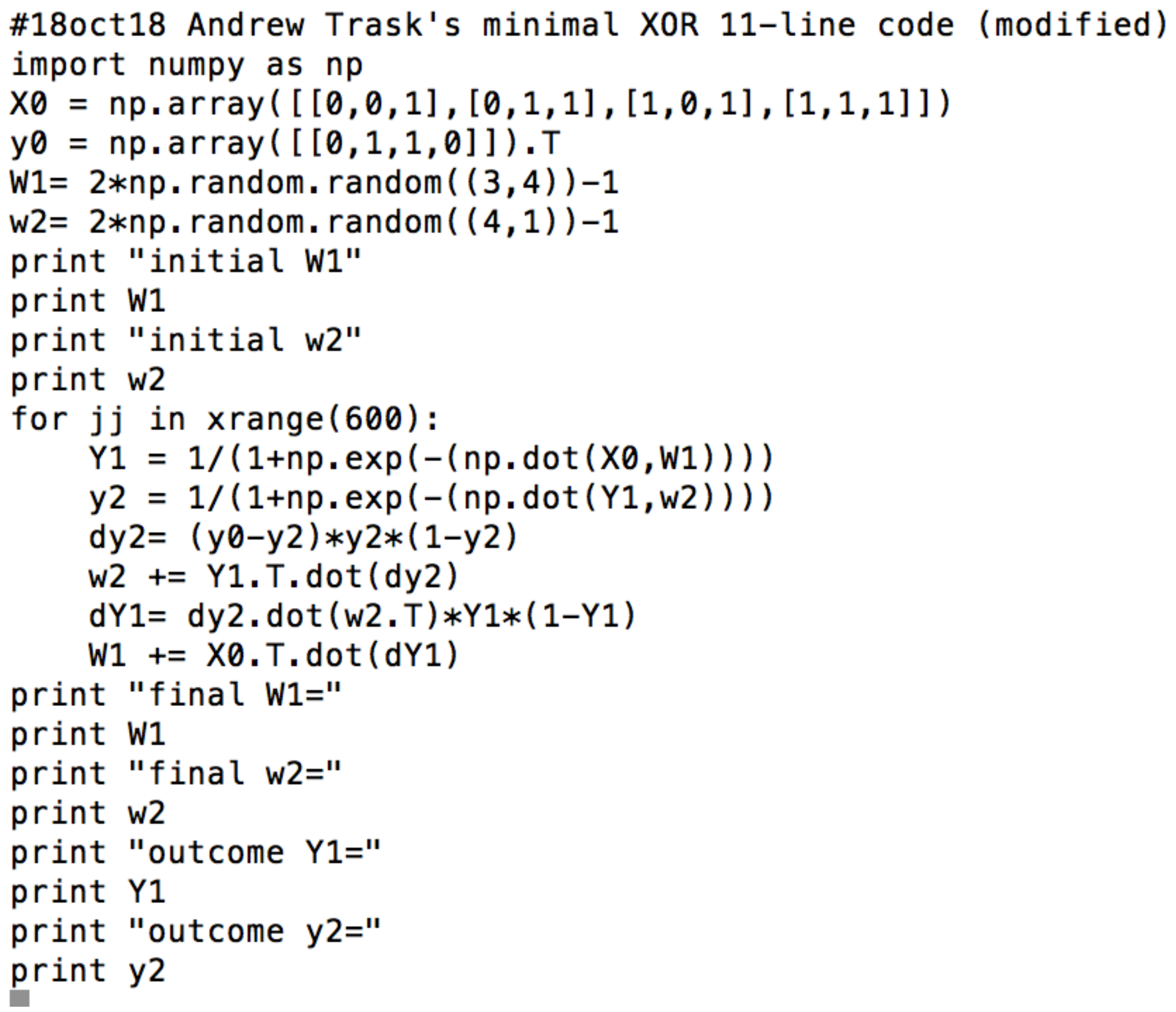


A Neural Network in 13 lines of Python (Part 2 - Gradient ... - i am trask
https://iamtrask.github.io/2015/07/27/python-network-part2/
Jul 27, 2015 - Summary: I learn best with toy code that I can play with. This tutorial teaches gradient descent via a very simple toy example, a short python ...

A Neural Network in 11 lines of Python (Part 1) - i am trask
https://iamtrask.github.io/2015/07/12/basic-python-network/
Jul 12, 2015 - I'll tweet it out when it's complete at @**iamtrask**. Feel free to .... **13**. [1 , 1 , 1]]). 14. 15. # output dataset. 16. y = np.array([[0 , 0 , 1 , 1]]).T. 17. 18.

A Neural Network in 13 lines of Python (Part 2 - Gradient ... - i am trask
https://iamtrask.github.io/page2/
How to Code and Understand DeepMind's Neural Stack Machine. Learning to Transduce with Unbounded Memory. Posted by **iamtrask** on February 25, 2016 ...

A neural network in 13 lines of python (Part 2 - An Intuitive ...
https://www.reddit.com/r/.../a_neural_network_in_13_lines_of_python_part_2_an/
Jul 28, 2015 - 4 posts - 4 authors
Andrew Ng and Adam Coates (4/15/2015) · Jürgen Schmidhuber (3/4/2015) ... A neural network in **13** **lines** of python (Part 2 - An Intuitive Tutorial of Stochastic Gradient Descent) (**iamtrask**.github.io). submitted 3 years ago by ...



```
#18oct18 Andrew Trask's minimal XOR 11-line code (modified)
import numpy as np
X0 = np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
y0 = np.array([[0,1,1,0]]).T
W1= 2*np.random.random((3,4))-1
w2= 2*np.random.random((4,1))-1
print "initial W1"
print W1
print "initial w2"
print w2
for jj in xrange(600):
    Y1 = 1/(1+np.exp(-(np.dot(X0,W1))))
    y2 = 1/(1+np.exp(-(np.dot(Y1,w2))))
    dy2= (y0-y2)*y2*(1-y2)
    w2 += Y1.T.dot(dy2)
    dY1= dy2.dot(w2.T)*Y1*(1-Y1)
    W1 += X0.T.dot(dY1)
print "final W1="
print W1
print "final w2="
print w2
print "outcome Y1="
print Y1
print "outcome y2="
print y2
```