What is the computational task a neural net learning?

How do we show that learning is hard for a 3 Node network?

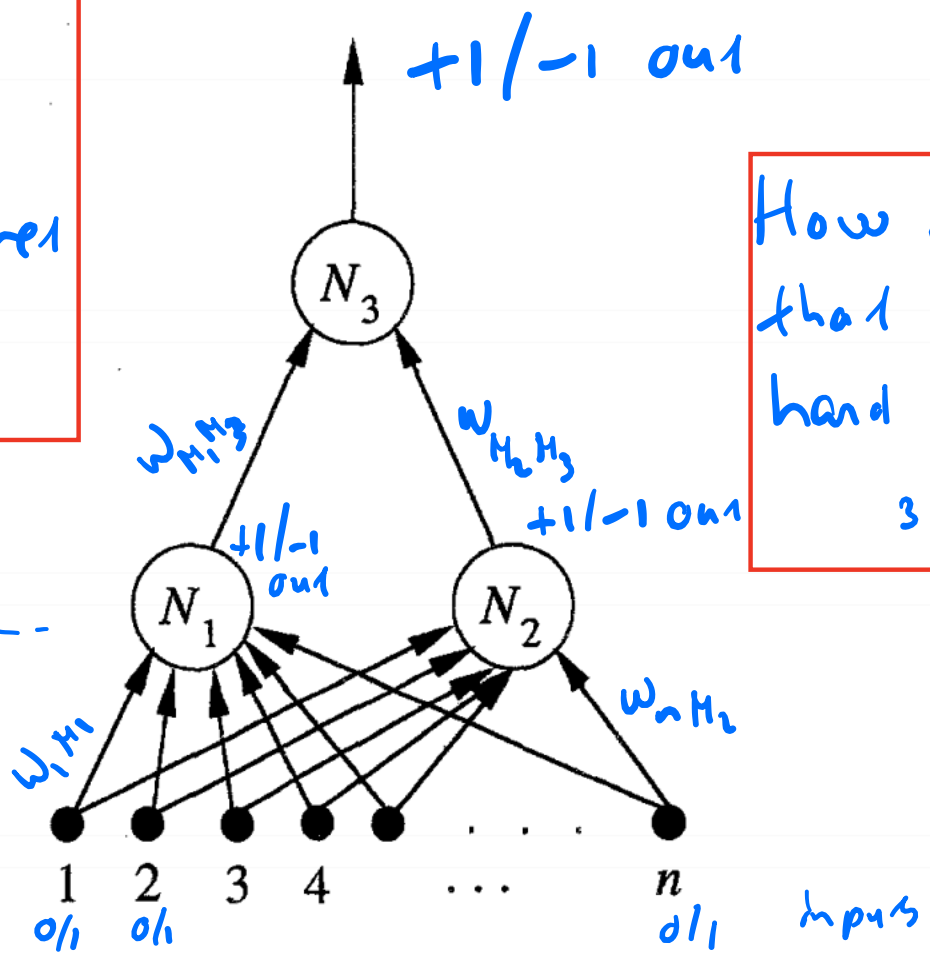


Fig. 1. The 3-Node Network.

Each node N_i computes a linear threshold function (also called N_i) on its inputs. If N_i has input $x = (x_1, \ldots, x_m)$, then for some values $a_0, \ldots, a_m$,

$$N_i(x) = \begin{cases} +1 & \text{if } a_1x_1 + a_2x_2 + \cdots + a_mx_m > a_0 \\ -1 & \text{otherwise.} \end{cases}$$

Definition 1. Given a neural network $\mathcal{N}$, let the *training problem for* $\mathcal{N}$ be the question:

> "Given a set of training examples, do there exist edge weights and thresholds for the nodes of $\mathcal{N}$ so that it produces output consistent with the training set?"

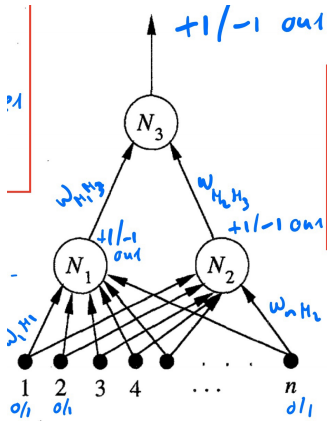
Theorem 2. *Training the 3-Node Network is NP-complete.*

i.e. believed to take time $O(2^n)$

The proof of Theorem 2 involves reducing the known NP-complete problem "Set-Splitting" to the network training problem. In order to more clearly un-

	features					label (output)
ex1	0	1	0	1	1	1
ex2	1	1	0	1	1	1
ex3	1	1	0	1	0	0
ex4	1	1	0	1	0	1
ex5	0	1	1	1	1	0
	1	2	3	...	n	

example encoding ex1



Find weights to classify all examples in training set correctly.

Aside: NP-completeness

1) Boolean satisfiability problem (SAT)
is NP-complete. (Cook 1971)

E.g. $(x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_3)$ (*)

Boolean vars. True / False

Q. Does there exist a truth assignment
s.t. (*) evaluates to True?

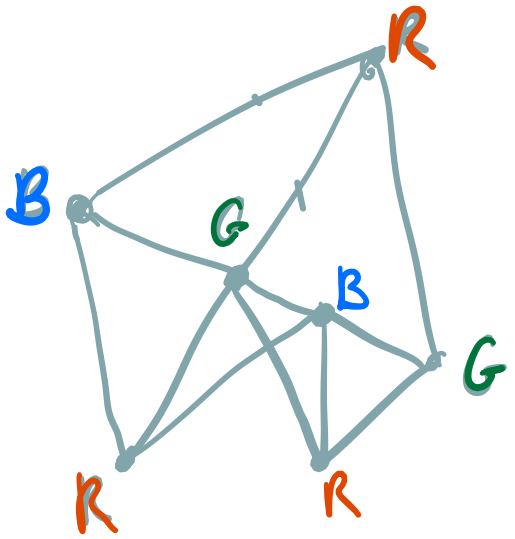
Best alg. $O(2^n)$ — n is # vars
(unless $P = NP$)

Computationally intractable.

2) Thousands of other known NP-complete problems.

E.g. graph coloring is NP-complete. (COL)

($O(2^n)$ – n nodes) & 3-COL



a valid 3-Coloring

How?

Polytime reduction / transformation:

SAT $\Rightarrow$ COL

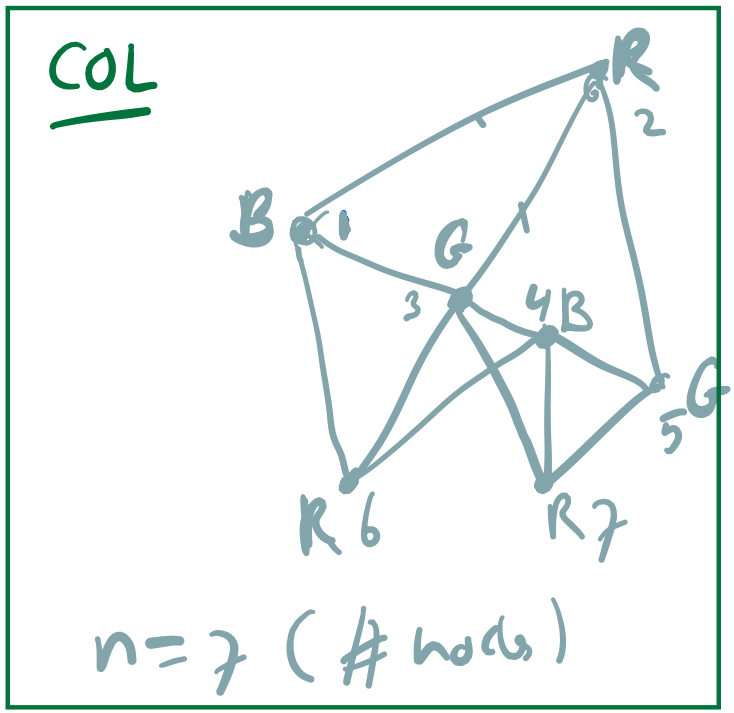
i.e.

Instance$_{SAT}$ is satisfiable iff Instance$_{COL}$ is 3-Colorable.

No efficient algorithm to find coloring.

3) All NP-complete problems are equivalent in computational terms.

E.g. SAT $\Rightarrow$ COL & COL $\Rightarrow$ SAT



For each node i:

$(x_i^B \vee x_i^R \vee x_i^G)$ SAT

also:

$x_i^R \Rightarrow \overline{x_i^G}$

$x_i^R \Rightarrow \overline{x_i^B}$

$x_i^B \Rightarrow \overline{x_i^G}$

$\equiv (\overline{x_i^R} \vee \overline{x_i^G})$

Finally, if i & j connected by edge then:

$x_i^B \Rightarrow \overline{x_j^B}$

$x_i^G \Rightarrow \overline{x_j^G}$

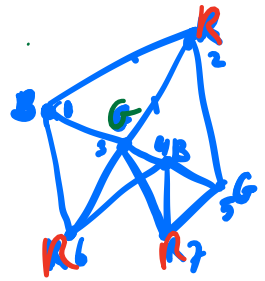
$x_i^R \Rightarrow \overline{x_j^R}$

So, we can go from a coloring problem to a logic problem & vice versa.

efficiently (poly time)

3-node neural net training is also part of this class.

COL



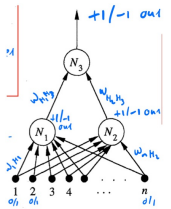
n=7 (# nodes)

$(x_1 \vee \bar{x}_1) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_3)$

i.e.

SAT $\Leftrightarrow$ COL

SAT $\Leftrightarrow$ 3MAT $\Leftrightarrow$ COL



3MAT $\hookrightarrow$ 2^n time, NP.

n is size of training

Theorem 2. *Training the 3-Node Network is NP-complete.*

Clever reduction from the Set_Splitting problem.

Ex. · Set Splitting

$S = \{1, 2, 3\}$ $n = 3$

$c_1 = \{1, 2\}$ $c_2 = \{2, 3\}$

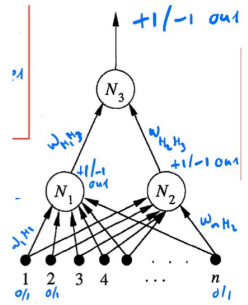
Can I 2-color elements in S, s.t. each c_i has 2 colors ?

Yes !

⇓

+ examples	− examples
000	100
110	010
011	001

NN training problem



It can be shown that

Set_Splitting is solvable iff Training set examples can be learned exactly by 3-node neural net.

one plane per color

Red: $-x_1 + 3x_2 - x_3 = -1/2$

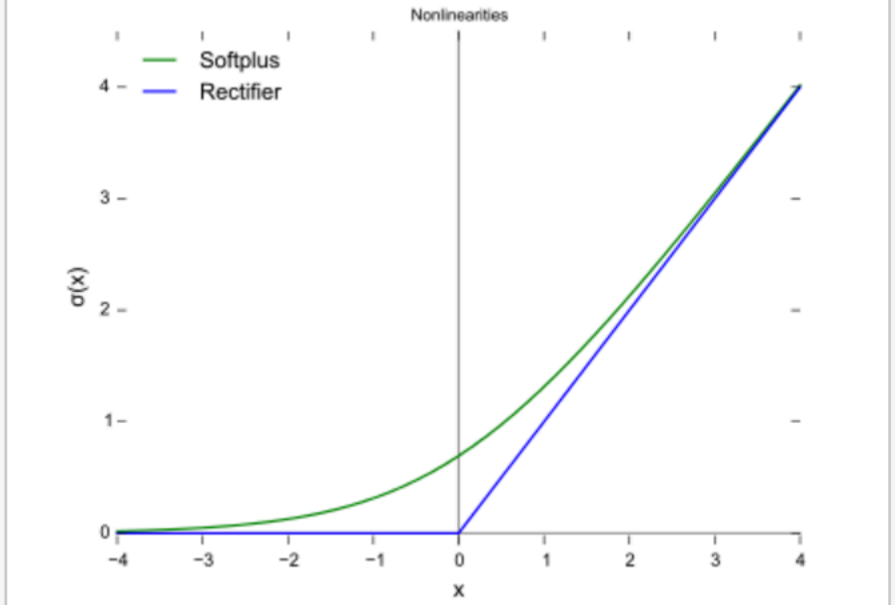
Blue: $3x_1 - x_2 + 3x_3 = -1/2$

We also show the training problem for the following networks to be NP-complete:

1. The 3-Node Network restricted so that any or all of the weights for one hidden node are required to equal the corresponding weights of the other (so possibly only the thresholds differ) and any or all of the weights are required to belong to $\{+1, -1\}$.

In addition we show that any set of positive and negative training examples classifiable by the 3-node network with XOR output node (for which training is NP-complete) can be correctly classified by a perceptron with $O(n^2)$ inputs which consist of the original n inputs and all products of pairs of the original n inputs (for which training can be done in polynomial-time using linear programming techniques).

This is a surprising & profound observation.



Plot of the rectifier (blue) and softplus (green) functions near $x = 0$.

A unit employing the rectifier is also called a **rectified linear unit** (**ReLU**).[4]

For the first time in 2011,[1] the use of the rectifier as a non-linearity has been shown to enable training deep supervised neural networks without requiring unsupervised pre-training. Rectified linear units, compared to sigmoid function or similar activation functions, allow for faster and effective training of deep neural architectures on large and complex datasets.