

# CS472 – Homework 5 – Solutions

December 10, 1999

## Problem 1

(a)

We just need to use the formulas for Section 18.4 from R&N.

$$\begin{aligned} \text{Gain(EQ)} &= I\left(\frac{9}{14}, \frac{5}{14}\right) - \text{Remainder(EQ)} \\ &= 0.9402 - (0.970 * 0.357 + 0.970 * 0.357 + 0) = 0.2467 \end{aligned} \quad (1)$$

$$\begin{aligned} \text{Gain(good hacker)} &= I\left(\frac{3}{5}, \frac{2}{5}\right) - \text{Remainder(good hacker)} \\ &= 0.970 - (0 * 1 + 0 * 1) = 0.970 \end{aligned} \quad (2)$$

$$\begin{aligned} \text{Gain(publications)} &= I\left(\frac{2}{5}, \frac{3}{5}\right) - \text{Remainder(publications)} \\ &= 0.970 - (0 * 1 + 0 * 1) = 0.970 \end{aligned} \quad (3)$$

(b)

---

Class	EQ	good hacker	has publications	hobby	reason
positive	190	?	no	sci-fi	decision tree
positive	130	?	no	music	data, yes more often than no
positive	130	yes	yes	?	form data, class does not depend on hobby
unclasi- fiable	160	no	?	tennis	no example matches the pattern

---

(c)

For any decision tree a decision list can be build in the following manner: for every path in the decision tree build a node for the decision list that has the query of the form:  $A_1 = v_1 \wedge A_2 = v_2 \wedge \dots \wedge A_k = v_k$  where  $A_i$  is the  $i$ -th attribute on the path and  $v_i$  is it's value along the path. If the path ends in a positive classification the YES branch of the query will answer *positive*, if the classification is *negative* the YES branch will answer *negative*. The NO branch always goes into the next query except for the last query where is not required.

Since the examples are learnable by a decision tree the above method can be used to build from this a decision tree so the examples are learnable by a decision list.

## Problem 2

(a)

The network is depicted in the Figure 1.

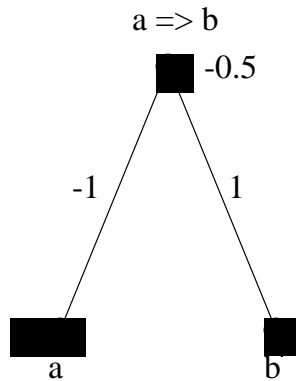


Figure 1: Neural network for  $a \Rightarrow b$

(b)

Figure 2 is the modified network.

(c)

Let  $l_1 \wedge l_2 \wedge \dots \wedge l_k$  be an arbitrary conjunction where  $l_i$  is an arbitrary literal (proposition or negation of a proposition). Let  $S(l_i)$  be 1 if  $l_i$  is a proposition and -1 if it is the negation of a proposition. The network in the Figure 3 implements the conjunction.

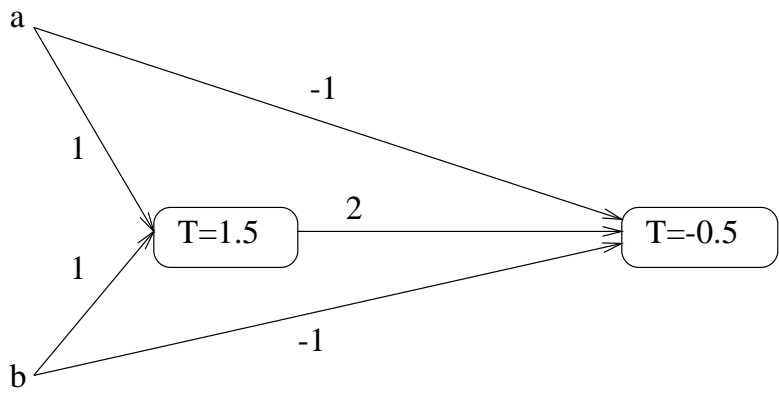


Figure 2: Neural network to represent  $a \equiv b$

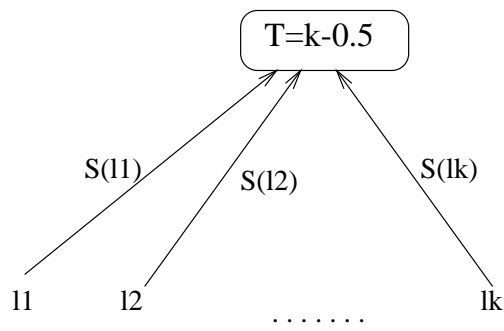


Figure 3: Neural network to represent a conjunction

(d)

Every propositional boolean formula can be put in Disjunctive Normal Form (disjunction of conjunctions). We can use the network in point (c) as a building block to build conjunctions. We just need a similar network for disjunctions. Figure 4 is a two layer network that implements a formula in DNF. Conjunction blocks are instantiations of the networks like the one described in point (c).

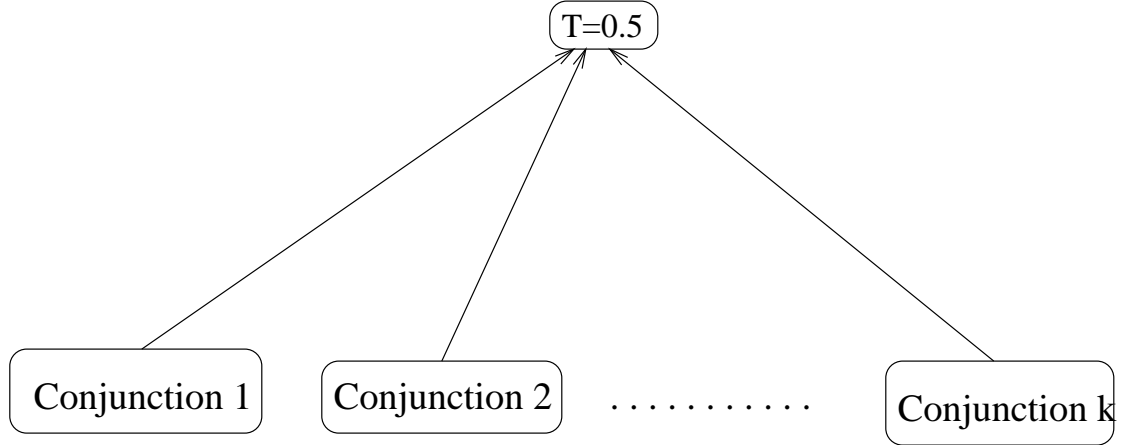


Figure 4: Neural network to emulate a propositional formula in DNF

### Problem 3

Because each literal can be positive or negative in a clause,  
Number of clauses with  $k$  literals =  $2^k \cdot C(n, k) \leq 2^k n^k$

Each clause can appear or can be absent in the k-CNF formula,

$$|\text{k-CNF}| = 2^{2^k C(n,k)}$$

Hence, the sample complexity  $m$  for k-CNF is:

$$\begin{aligned} m &\geq \frac{1}{\epsilon} \left( \ln \frac{1}{\delta} + \ln |\text{k-CNF}| \right) \\ &= O(n^k) \end{aligned}$$

The following algorithm can return a k-CNF formula that is consistent with the training examples, and it can be easily seen running in polynomial time.

**begin**

**H** = conjunction of all ternary clauses

**foreach** positive example  $e$

```

foreach clause  $c$  in  $\mathbf{H}$ 
  if  $e$  falsify  $c$ , delete  $c$  from  $\mathbf{H}$ 
return  $\mathbf{H}$ 
end

```

Therefore,  $k$ -CNF is PAC learnable by i. and ii.

### Problem 4

The smallest decision tree is given in the figure.

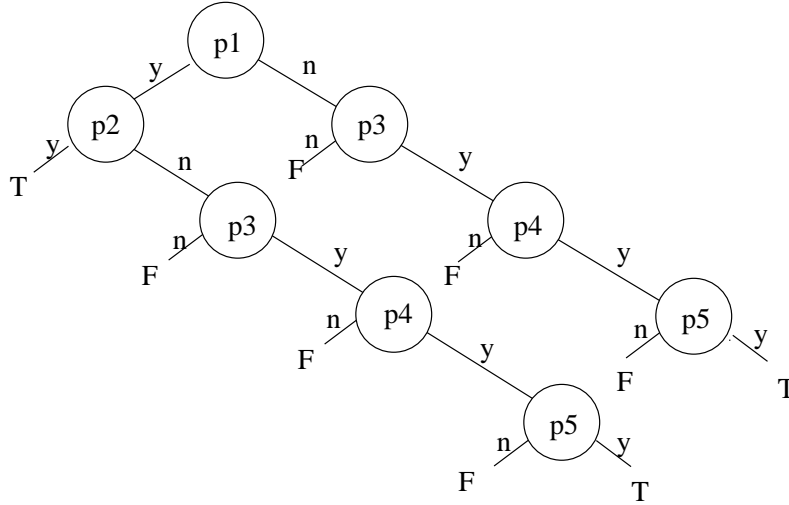


Figure 5: Decision tree to represent the formula

The smallest decision tree for this concept contains a replicated subtree. We cannot remove the redundancy by using pointers instead of copying the subtree because that would lead to a DAG instead of a tree. Another way to look at it is that in the evaluation of the expression tree (see figure 2), we can do something similar to backtracking if we don't find a solution in the first subtree, whereas in decision trees, we descend deeper down the tree at each step and never go back.

### Problem 5

(a)

If the multiplicative constant is positive the behavior will not change. To see this let's consider a general neural network with step threshold activation function.

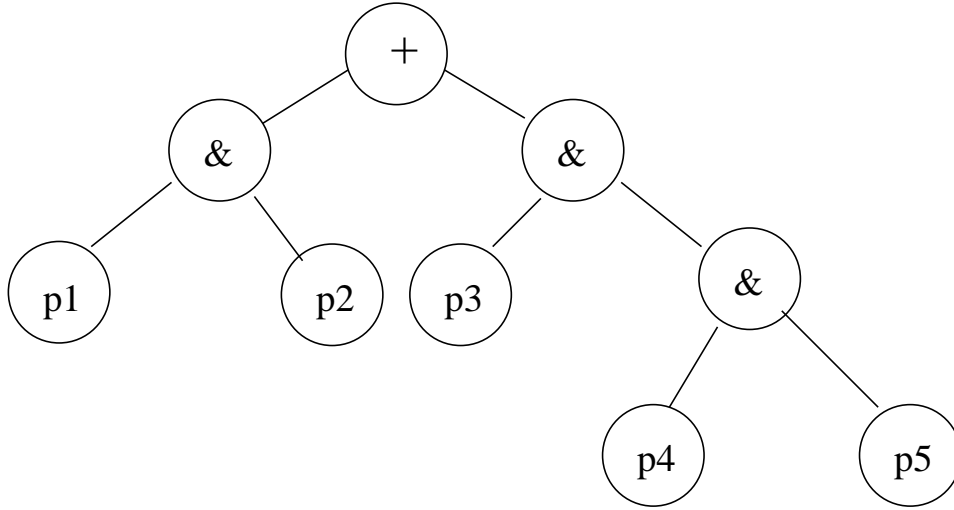


Figure 6: Expression tree

The output of a neuron as a function of inputs and weights is:

$$o = g\left(\sum_{i=1}^n w_i a_i + t\right) \quad (4)$$

where  $o$  is the output of the neuron,  $g$  is the step function,  $a_i$  are the inputs,  $t$  is the threshold and  $w_i$  are the weights. If we multiply by a constant  $c$  both weights and the threshold we get:

$$o' = g\left(\sum_{i=1}^n c w_i a_i + ct\right) = g\left(c\left(\sum_{i=1}^n w_i a_i + t\right)\right) \quad (5)$$

Since for the step function  $g(x) = g(cx)$  if  $c > 0$ , we conclude that  $o' = o$ . Because the neuron we considered was arbitrary and we proved that by multiplying the weights and the threshold with a constant leaves the behavior the same, we conclude that the behavior of the whole network is the same; it doesn't matter how neurons are connected.

If the multiplicative constant is negative the behavior changes.

**(b)**

Yes. Let's add a constant  $c$  to the weights and the threshold of the equation 4.

$$o' = g\left(\sum_{i=1}^n (w_i + c)a_i + t + c\right) \quad (6)$$

In general (with the exception of a single case when  $a_i$  are chosen such that the equality is preserved)  $o' \neq o$  and in any case  $o'$  and  $o$  as functions of  $a_i$  are different. This means that the behavior of this neuron is different from the original one, which means that in general the behavior of the neural network is different. (There is at least an input such that the output of the old and new neural network differ).