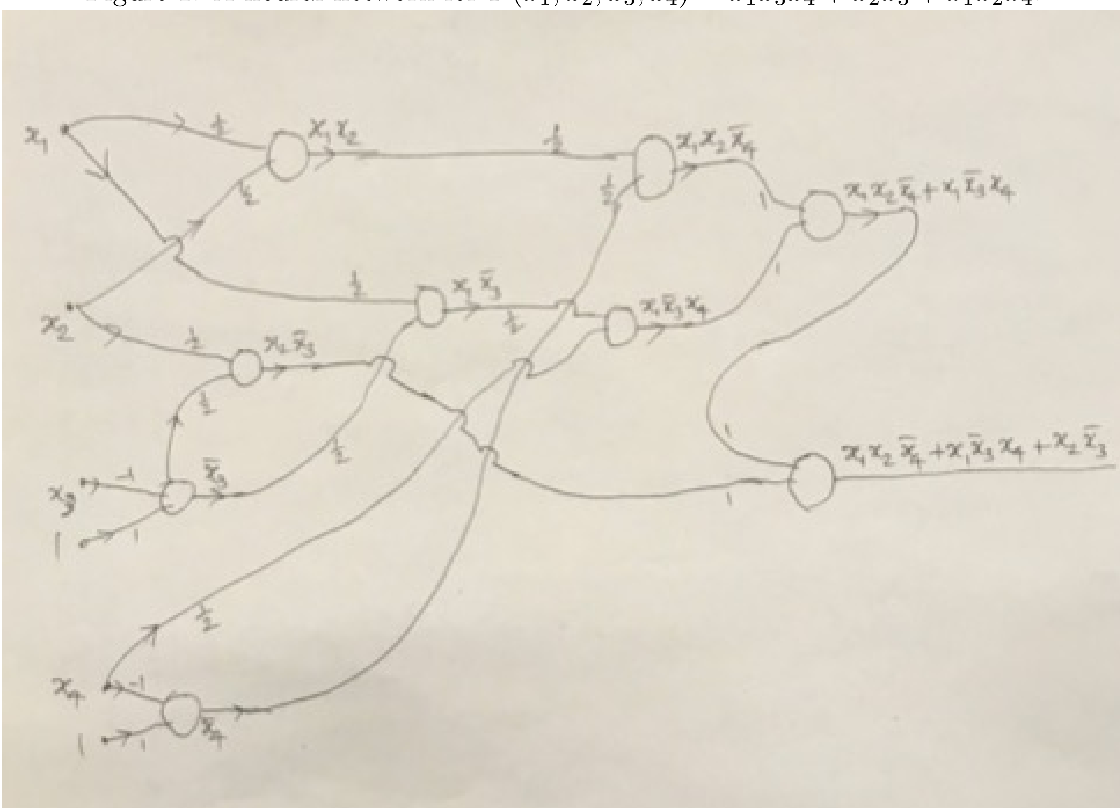
CSE 4502/5717 Big Data Analytics. Fall 2019

Exam III Solutions

1. The loss function is $L(w_1, w_2) = (w_2 - 3)^2 + (w_1 - 4)^2 + (w_1 + w_2 -)^2 + (2w_1 + w_2 - 10)^2$ $= 6w_1^2 + 3w_2^2 + 6w_1w_2 - 60w_1 - 38w_2 + 161$. We want to have: $\frac{\partial L}{\partial w_1} = 0$ and $\frac{\partial L}{\partial w_2} = 0$.
   $\frac{\partial L}{\partial w_1} = 0$ implies that $12w_1 + 6w_2 = 60$ and $\frac{\partial L}{\partial w_2} = 0$ implies that $6w_1 + 6w_2 = 38$. Solving these two equations, we get: $w_1 = \frac{11}{3}$ and $w_2 = \frac{8}{3}$.

2. Here is a multilevel perceptron for realizing the Boolean function $F(x_1, x_2, x_3, x_4) = x_1\bar{x_3}x_4 + x_2\bar{x_3} + x_1x_2\bar{x_4}$:

Figure 1: A neural network for $F(x_1, x_2, x_3, x_4) = x_1\bar{x_3}x_4 + x_2\bar{x_3} + x_1x_2\bar{x_4}$.



3. (a) Let the output vector at layer k be $\vec{a_k}$, for $k = 2, 3, \ldots, L$. Let the input vector be $\vec{a_1}$. Also, let the weight matrix and bias vector of layer k be W_k and $\vec{b_k}$, respectively, for $k = 2, 3, \ldots, L$.
   Then, we know that $\vec{a_k} = W_k\vec{a}_{k-1} + \vec{b_k}$, for $2 \leq k \leq L$.
   Note that $\vec{a_k} = W_k(W_{k-1}\vec{a}_{k-2} + \vec{b}_{k-1}) + \vec{b_k} = W_kW_{k-1}\vec{a}_{k-2} + W_k\vec{b}_{k-1} + \vec{b}_k$.

Expanding in a similar manner we see that:

$$\vec{a}_L = W_L W_{L-1} \cdots W_2 \vec{a}_1 + W_L W_{L-1} \cdots W_3 \vec{b}_2 + W_L W_{L-1} \cdots W_4 \vec{b}_3 + \cdots + W_L W_{L-1} \vec{b}_{L-2} + W_L \vec{b}_{L-1} + \vec{b}_L.$$

We can perform the above computation by first computing the following matrices: $W_L, W_L W_{L-1}, W_L W_{L-1} W_{L-2}, \ldots, W_L W_{L-1} W_{L-2} \cdots W_3 W_2$. Note that this a prefix computation where the underlying operation is that of multiplying two $n \times n$ matrices. Two $n \times n$ matrices can be multiplied in $O(\log n)$ time using $\frac{n^3}{\log n}$ CREW PRAM processors. As a result, the above prefix computation can be performed in $O(\log L \log n)$ time using $\frac{Ln^3}{\log L \log n}$ CREW PRAM processors.

Followed by the prefix computations we have perform $(L-1)$ matrix (of size $n \times n$) vector (of size $n \times 1$) multiplications. Each multiplication can be done in $O(\log n)$ time given $\frac{n^2}{\log n}$ CREW PRAM processors. We have enough processors to do all of these products in parallel.

Finally, we have to perform the addition of $L-2$ vectors (of size $n \times 1$ each). This can be done in $O(\log L)$ time using a total of $\frac{nL}{\log L}$ processors.

(b) Consider the case when all the bias vectors are zero. In this case, $\vec{a}_L = W_L W_{L-1} \cdots W_2 \vec{a}_1$. In other words, $\vec{a}_L = W \vec{a}_1$ for some matrix W. We can directly learn W with no hidden layers!

4. Let the transactions in the database be $t_1, t_2, \ldots, t_q$. Note that any item in the database can be represented as an integer in the range $[1, n^c]$.

   S is an empty sequence to begin with;
   for $i = 1$ **to** q **do**
       for every item $a \in t_i$ **do**
           Add a to the sequence S;
   Sort S using the integer sorting algorithm;
   Scan through the sorted sequence to count the support for each item and
   output those that have enough support.

   Clearly, the above algorithm runs in $O(n)$ time.

5. Sort X to get a sorted sequence: $r_1, r_2, \ldots, r_n$. This will take $O(n \log n)$ time. Let d_i stand for $|r_{i+1} - r_i|$ for $i = 1, 2, \ldots, (n-1)$. Find the $(n-k)^{\text{th}}$ smallest number q among $d_1, d_2, \ldots, d_{n-1}$. This can be done in $O(n)$ time using the BFPRT algorithm. Think of a linear graph $G(V, E)$ in which each r_i is a node (for $1 \le i \le n$), and there is an edge between r_{i+1} and r_i (for $1 \le i \le (n-1)$). The weight on the edge (r_{i+1}, r_i) is d_i (for $1 \le i \le (n-1)$). In this graph keep only edges whose weights are $\le q$. I.e., we keep $(n-k)$ edges. The connected components of this graph are the clusters that we are looking for. These connected components can be found in one pass through the sorted sequence in $O(n)$ time.

   The total run time of the algorithm is $O(n \log n)$.