CS 344 (Spring 2017): Class Test 2

Instructor: Shivaram Kalyanakrishnan

8.30 a.m. – 9.30 a.m., February 14, 2017, 101/103/105 New CSE Building

Total marks: 15

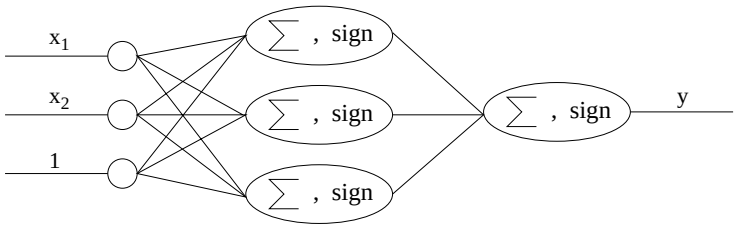
Note. Provide brief justifications and/or calculations along with each answer to illustrate how you arrived at the answer.

Question 1. This question considers representing Boolean functions using neural networks.

1a. The neural network in the figure below has an input layer with three nodes; one hidden layer with three nodes; and an output layer with a single node. The hidden nodes and the output node all compute the sign function on the weighted sum of their inputs. Recall that for $\alpha \in \mathbb{R}$,

$$\text{sign}(\alpha) = \begin{cases} 1 \text{ if } \alpha \geq 0, \\ 0 \text{ if } \alpha < 0. \end{cases}$$

In other words, each of the hidden nodes and the output node is a Perceptron. The first two inputs to the network correspond to bits x_1 and x_2: these inputs can only take values 0 and 1. The third input is always a *constant* 1. The input layer is fully connected to the hidden layer, and every hidden node is connected to the output node.



x_1	x_2	y
0	0	1
0	1	1
1	0	0
1	1	1

Your task is to determine weights for this neural network such that its output y corresponds to the Boolean function $(x_1 \wedge x_2) \vee \neg x_1$. Draw the network and label the edges with weights that implement this input-output relationship, which is also specified in the table above. You do not have to show connections to which you assign a weight of zero. [4 marks]

1b. Based on 1a, can you infer a general principle to implement an *arbitrary* Boolean function y of n Boolean variables using a neural network? The network must have one input layer with $n + 1$ inputs (each of the n Boolean variables, and also a constant 1 as input), and a single output node. For a general function y, can you manage with a single hidden layer (with a possibly large but still finite number of nodes), or do you need more than one layer? Provide a construction to justify your answer. [3 marks]

Question 2. We now consider how Boolean functions may be represented using decision trees.

2a. Draw a decision tree to compute the Boolean function in 1a: that is, $y = (x_1 \wedge x_2) \vee \neg x_1$. The internal nodes of the tree must correspond to either x_1 or x_2; the two branches from each internal node must correspond to the values that these variables can take (either 0 or 1); the leaves must correspond to the output y (also 0 or 1). [2 marks]

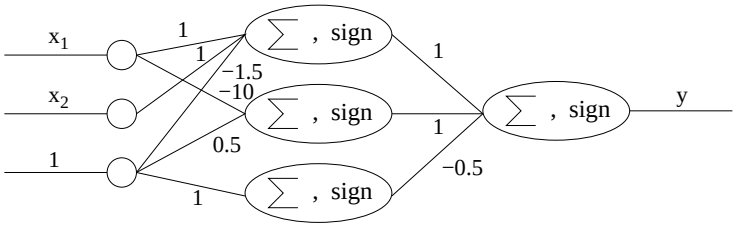
2b. Write down y', a Boolean function of x_1 and x_2, such that any decision tree implementing y' must necessarily contain three internal nodes and four leaves. Also draw a decision tree that implements y'. [2 marks]

Question 3. What is k-fold cross-validation? Describe the procedure and explain why it is performed. [3 marks]

Question 4. Modern machine learning-based methods have significantly reduced error rates in automatic speech recognition. What is the main challenge in applying these methods successfully to Indian languages? [1 mark]

Solutions

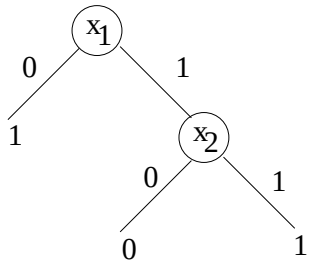
1a. Below is one assignment of weights that implements $y = (x_1 \wedge x_2) \vee \neg x_1$. The output of the first hidden node is $x_1 \wedge x_2$, and the output of the second hidden node is $\neg x_1$. The third hidden node merely carries forward a constant output of 1. The output node computes the disjunction of the outputs from the first two hidden nodes; the constant bias is used in implementing the disjunction.



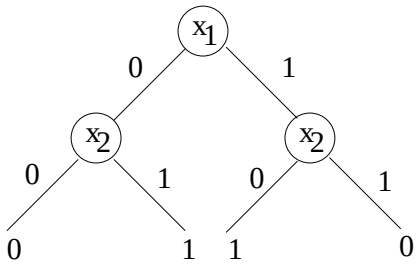
1b. It is well-known that every Boolean function can be expressed as a sum (disjunction) of products (conjunctions of variables and their negations). We can have one hidden layer, with each node in the hidden layer computing one particular product (and one additional node in the hidden layer to carry forward the constant bias). If we do so, the function will be implemented by giving a weight of 1 to each connection going from the product-computing hidden nodes to the output, and a weight of -0.5 to the output of the hidden node carrying forward the bias. This approach is consistent with the solution to 1a.

It remains to be shown how products of variables and their negations can be computed at the hidden nodes. Let a hidden node h compute one such product: for example, let the product be $x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4$. Observe that h will output this product if each non-negated variable (x_1, x_3) is given a weight of 1, each negated variable (x_2, x_4) is given a large negative weight, say $-n$, and the constant term's connection to h has a weight of $0.5 - p$, where p is the number non-negated variables (in the example, $p = 2$). Hence, a neural network with one hidden layer suffices to represent an arbitrary Boolean function.

2a. Below is one possible decision tree to represent $y = (x_1 \wedge x_2) \vee \neg x_1$.



2b. Consider $y = x_1 \oplus x_2$; that is, $y = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$. Since the function is symmetric with respect to x_1 and x_2, choosing any one of the variables as the root of the decision tree will be functionally the same as choosing the other. Without loss of generality, assume we pick x_1 for the root, as shown in the decision tree below. The function y is such that after having conditioned on x_1, each subtree must necessarily predict a different response for each values of x_2. Consequently the tree must have four leaves.



3. Learning algorithms typically have hyperparameters: such as the *number of layers* in a neural network, the *learning rate* in on-line update rules, and the *pruning threshold* in decision trees. Cross-validation is a method that uses the given training data set to tune these hyperparameters.

Under k-fold cross validation, the training data set is partitioned into k (roughly) equal sets called *folds*. The performance of a particular configuration (say, of hyperparameters) is taken as the average of the performance on k runs: in each run, the training algorithm, implementing the given configuration, is run on some $k-1$ folds, and its performance measured on the remaining fold. This test fold is distinct in each of the k runs.

Rather than a separate out a single validation set from the training data, as is also done quite often, averaging over multiple train-validate splits reduces the variance in evaluating a model. Of course, k-fold cross-validation requires k times as much computation (which can be parallelised, though).

4. Machine learning-based methods rely on the statistical strength of data. Unfortunately, there is a shortage of labeled training data in most Indian languages. These languages are termed "low resource" languages, as opposed to "high resource" languages such as English and Spanish. Understandably, the performance of machine learning-based speech recognition systems is not as high on low resource languages as it is on high resource languages.