10-806 Foundations of Machine Learning and Data Science

Lecturer: Avrim Blum Additional notes: December 2, 2015

---

1 Brief notes on multi-layer networks

Multilayer feed-forward networks have re-emerged as a popular, powerful hypothesis class for different kinds of learning problems.

In a multi-layer network we have n inputs, and then a series of computational nodes (or gates), that each compute some function of their input. Outputs of these nodes can then be inputs to other nodes. These values travel up through the network (or circuit) until we get to the final output of the overall function. We assume no loops, so in graph-theoretic terminology, the network is a directed acyclic graph.

One of the most basic types of gates is the linear-threshold function gate: such a node stores a weight vector $\mathbf{w}$ and threshold w_0, and then outputs 0 or 1 on an input $\mathbf{x}$ depending on whether or not $\mathbf{w} \cdot \mathbf{x} \geq w_0$. We can think of a network architecture (that is, the connections in the network, and the types of gates, but not yet fixing the weights) as defining a concept class $\mathcal{C}$, where each $h \in \mathcal{C}$ is defined by some way of assigning the weights. Given such a class $\mathcal{C}$, one of the most basic computational problems is the *consistency problem for* $\mathcal{C}$: given a labeled sample S, can you find an $h \in \mathcal{C}$ with $err_S(h) = 0$ if such h exists?

1.1 NP-hardness of the consistency problem

Earlier, we showed that this computational problem is NP-hard for the class $\mathcal{C}$ consisting of all Boolean functions that can be described as an intersection of two halfspaces. Here we extend this result to show hardness for the case of a network with three computational nodes f_1, f_2, f_3, where f_1 and f_2 compute linear threshold functions over the n inputs, and then f_3 computes a linear threshold function over a 2-bit input given by the outputs of f_1 and f_2. The output of f_3 is then the output of the network.

Let us first recall the proof of hardness for an intersection of two halfspaces, which corresponds to a network of the type above but where we require f_3 to compute the AND function.

We prove that this problem is NP-hard by performing a reduction from the NP-hard hypergraph 2-coloring problem. In hypergraph 2-coloring, you are given a hypergraph, which is just a set V of n nodes (call them $1, 2, \ldots, n$) and m subsets $s_1, s_2, \ldots, s_m \subseteq V$ called "hyperedges". Your goal is to give each node a color, Red or Blue, such that no subset s_j is monochromatic. That is, every s_j has at least one Red node and at least one Blue node. This problem is NP-hard.

To prove NP-hardness of the consistency problem for the intersection of two halfspaces we need a method that given an instance of hypergraph 2-coloring (a collection of m subsets $s_i \subseteq \{1, \ldots, n\}$) constructs a labeled sample S that is consistent with an intersection of two halfspaces (i.e., an AND of two linear threshold functions) if and only if the given instance of hypergraph 2-coloring has a solution. We do this as follows.

1. First, we label the origin as positive.
2. Next, for each coordinate i, we label the unit vector e_i (having a 1 in coordinate i and a 0 in all other coordinates) as negative.
3. Finally, for each set s_j we label the indicator-vector for that set (having a 1 in coordinate i for each $i \in s_j$ and a 0 in the rest) as positive.

Claim 1 *The labeled examples produced by this reduction are consistent with an intersection of two halfspaces (i.e., an AND of two linear threshold functions) if and only if the given instance of hypergraph 2-coloring has a solution.*

(Proof given in Lecture 9.)

We now extend this result to consider the problem where f_3 can be any LTF over its 2-bit input, and not just an AND function. The reason that the above reduction does not quite work is that the dataset S created is always consistent with an OR of two LTFs: namely the LTF $x_1+x_2+\ldots+x_n \leq 1/2$ and the LTF $x_1 + x_2 + \ldots + x_n \geq 3/2$. To fix this problem, we simply add a small additional labeled sample S' to the set S given above so that (a) it is no longer possible for f_3 to be an OR function and still be consistent with the sample $S \cup S'$, and yet (2) if there exists an AND of two LTFs consistent with S then there still exists an AND of two LTFs consistent with $S \cup S'$. This is sufficient because the only functions computable by our 3-node network are those that correspond to an AND of two LTFs, an OR of two LTFs, or just a single LTF (and the single LTF is already disallowed by the original set S).

To do this reduction, we add three new coordinates $n+1, n+2, n+3$. We give all examples in S a value of 0 in these coordinates. We then let S' be the following 6 new points (which are zero in coordinates $1, \ldots, n$):

0	...	0	1	0	0	−
0	...	0	0	1	0	−
0	...	0	0	0	1	−
0	...	0	0	1	1	+
0	...	0	1	1	0	+
0	...	0	1	1	1	−

This S' is not consistent with an OR of two LTFs, but any AND of two LTFs consistent with S can be extended to be consistent with $S \cup S'$. For example, given LTFs $w_1x_1 + \ldots w_nx_n \leq 1/2$ and $w'_1x_1 + \ldots w'_nx_n \leq 1/2$ used in proving Claim 1, we simply extend them as:

$$w_1x_1 + \ldots w_nx_n + x_{n+1} + x_{n+2} - x_{n+3} \leq 1/2$$

and

$$w'_1x_1 + \ldots w'_nx_n - x_{n+1} - x_{n+2} + x_{n+3} \leq 1/2.$$

1.2 Worst-case vs realistic case

The above just shows that there exist functions that such a network can represent that can be difficult to find. The success of deep networks in practice shows that for many functions that we *want* to use complex networks to find, we can in fact find a good set of weights efficiently, often using stochastic gradient descent. That is, real problems may well not be worst-case.