6.858/18.428 Machine Learning	Lecture 14 : October 31, 1994
Lecturer: Ron Rivest	*Scribe: Michael Best*

14.1 Outline

- Finish weak $\Rightarrow$ strong learning and implications
- Neural nets – an introduction

14.2 Weak learning

14.2.1 Weak learning – initial bounds

Recall our weak learning algorithm, L, which gave us an error rate, $\beta \leq (1/2 - 1/p(n))$ with probability $1 - \delta$. We built a ternary tree of majority gates where the height of our tree was defined by $B = O(\log(p(n)) + \log\log(1/\varepsilon))$. The leaf nodes of our execution tree are labeled with hypotheses from L, using the oracle $EX(c, \mathcal{D}')$. Here $\mathcal{D}'$ are special distributions derived from the original distribution $\mathcal{D}$.

We need to allocate a confidence measure to all points in the execution tree which can fail. We give a bound on the total number of places where the algorithm can fail and allocate probability, $\delta' = \delta/3^B$, to each point. We now wish to arrive at some bounds on a strong learning algorithm SL, as a function of similar bounds on our weak learning algorithm L. Let the following bounds, $t(\delta, n)$, $m(\delta, n)$, and $u(\delta, n)$ be the running time, sample size required, and time to evaluate a hypotheses respectively for algorithm L.

We can arrive at the following bounds:

$$U = \text{time to evaluate hypothesis} = O(3^B \cdot u(\delta', n)) = \text{poly}(n, 1/\varepsilon, 1/\delta).$$

By poly(...) we mean a polynomial function in the given three variables.

In the last lecture we showed how to bound the total number of draws required on the original distribution $\mathcal{D}$ to arrive at the desired distribution of some leaf node, $\mathcal{D}'$.

From these bounds we arrive at,

$$\begin{aligned}
M &= \text{sample size} = O(216^B/\varepsilon^2(m(\delta', n) + p^2(n)\log(1/\delta'))) \\
&= \text{poly}(n, 1/\varepsilon, 1/\delta) \\
T &= \text{total time} = O(648^B/\varepsilon^2(m(\delta', n) + p^2(n)\log(1/\delta') \cdot B \cdot u(\delta', n)) + 3^B t(\delta', n)) \\
&= \text{poly}(n, 1/\varepsilon, 1/\delta)
\end{aligned}$$

14.2.2 Weak learning – improved bounds

The above bounds are improveable by a few techniques. In particular, it has been shown that by "damaging" a hypothesis found to be too "good" we can arrive at better asymptotic results.

$$M = \text{sample size} = 1/\varepsilon \cdot \text{poly}(n, \log(1/\varepsilon), \log(1/\delta))$$

$$T = \text{time} = 1/\varepsilon \cdot \text{poly}(n, \log(1/\varepsilon), \log(1/\delta))$$

$$\text{memory} = \text{poly}(n, \log(1/\varepsilon), \log(1/\delta))$$

Note that even though our sample size is growing with $1/\varepsilon$ the memory requirements are growing with $\log(1/\varepsilon)$. Therefore we do not have to keep all of our examples in memory at all times.

$$size(h) = \text{poly}(n, \log(1/\varepsilon))$$

We can see that this method produces a big compression factor from our set of samples $(O(1/\varepsilon))$ to our hypothesis $(O(\log(1/\varepsilon)))$.

14.2.3 On-line learning

Now we'd like to consider how our bounds will change for an on-line algorithm.

The probability of an error after m trials is $1/m \cdot \text{poly}(n, \log(m))$. So the expected number of mistakes in these first m trials is $\text{poly}(n, \log(m))$. The computation time of the first m trials is $m \cdot \text{poly}(n, \log(m))$. Finally the space used for the first m trials is $\text{poly}(n, \log(m))$.

14.2.4 An algorithm with a single majority gate

Consider learning from a single majority gate. In other words, where our output is the result of a single majority function applied to some set of k hypothesis. This technique, due to Freund [1], gives the best theoretic results known to date.

Let $\gamma = 1/p(n)$, and our committee size, $k = 1/(2\gamma^2)\log(1/\varepsilon)$.

Consider training the particular hypothesis, h_{i+1}. Let r be the number of the first i committee members which were correct on the current example. We wish to decide whether to give this particular example to our current hypothesis, h_{i+1}. We will ignore the example if $r > \lfloor k/2 \rfloor$, or $r < i - \lceil k/2 \rceil$. This is because in the first case we would already have a majority of hypothesis that are correct, and in the second case we already have a majority that are incorrect. Thus we consider training our new hypothesis only on those samples in which it is possible that an additional vote might influence the outcome. Rather then always excepting the sample in that case we accept it with the probability:

$$\binom{k-i-1}{\lfloor k/2 \rfloor - r} (1/2+\gamma)^{\lfloor k/2-r \rfloor} (1/2-\gamma)^{\lceil k/2 \rceil - i - 1 + r}$$

If not enough examples are seen by our current hypothesis (because the i previous members are too good) then report a coin flip instead. The bounds for this algorithm follow:

$$\begin{aligned} M = \text{sample size} &= 1/\varepsilon \cdot \log^2(1/\varepsilon)(\log\log(1/\varepsilon) + \log(1/\delta)) \cdot \text{poly}(n) \\ T = \text{time} &= 1/\varepsilon \cdot \log^2(1/\varepsilon)(\log\log(1/\varepsilon) + \log(1/\delta)) \cdot \text{poly}(n) \\ \text{memory} &= \log(1/\varepsilon)(\log\log(1/\varepsilon) + \log(1/\delta)) \cdot \text{poly}(n) \\ \text{size of hypothesis} &= \log(1/\varepsilon) \cdot \text{poly}(n) \\ \text{time to evaluate} &= \log(1/\varepsilon) \cdot \text{poly}(n) \end{aligned}$$

14.3 Neural networks – an introduction

We can think of a neural network as a layered directed acyclic graph. We have n layer 0 nodes of indegree 1 which we think of as our inputs. Usually there is a single node at the highest level of the graph with outdegree 1 which we think of as our output.

Associated with each directed edge is a weight, $w \epsilon \mathcal{R}$. At any node in the graph, the weighted sum of its inputs with their weights is computed, $w \cdot x$, and the result is compared against some thresholding function to determine whether the node's output is positive or negative.

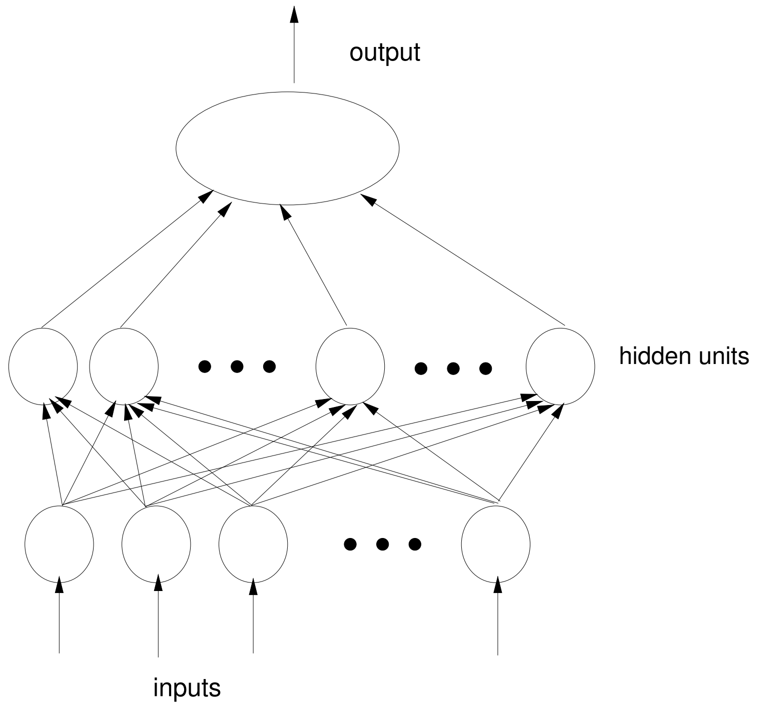

Figure 14.1: A completely connected neural net with one hidden layer.

We could use a sharp threshold function like we saw in the perceptron model. However, more success has been found when using sigmoid threshold functions. One nice property of these functions is that they, unlike a sharp threshold, are differentiable at all points.

Consider the following method generally referred to as back-propagation:

1) Define an error measure for the weights, $Error(w)$. This could be the mean-squared error for instance.

2) Compute the gradient of the errors with respect to the weights, $\bigtriangledown E$.

3) Iteratively reduce the error by adjusting the weights, $w \leftarrow w - \varepsilon \cdot \bigtriangledown E$.

Continue this process until the error is sufficiently small.

We will continue this discussion of back-propagation in our next lecture.

14.4 Bibliography

[1] Y. Freund. An improved boosting algorithm and its implications on learning complexity. *Proceedings of the 5th Workshop on Computational Learning Theory*, pages 391-398, ACM Press, New York, NY, 1992.

[2] R. E. Schapire. *The Design and Analysis of Efficient Learning Algorithms.* MIT Press, Cambridge, MA, 1992.