6.858/18.428 Machine Learning	Lecture 17: November 14, 1994
Lecturer: Ron Rivest	*Scribe: Milos Hauskrecht*

Outline

1. Complexity of training neural nets
   - Perceptrons/Linear programming
   - Judd's result
   - Blum/Rivest result
2. Approximation properties of neural nets
   - Cybenko's result

17.1 Complexity of training neural nets

So far in this course we have studied neural nets more from the practical side. Now we will discuss properties of neural nets and address theoretical issues like:

- *computational complexity*: Given a training set of examples and a neural network, how hard is it to learn the weights of the net such that good performance on the training set is achieved?
- *sample complexity*: How many samples do we need to see in the training phase in order to achieve good performance in the testing phase?

First we will focus on the issues related to computational complexity of neural nets and present some results achieved in recent years.

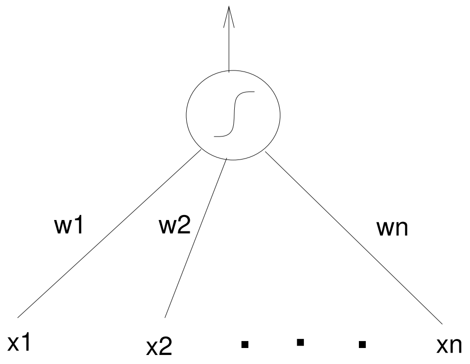

Figure 17.1: One node neural net

17.1.1 Complexity of training a one-node neural net

Here we will assume the easiest neural net topology — a neural net with one n-input node (i.e., a perceptron). The question we would like to answer is: Is it possible to train a one node neural net in a reasonable time? That is, is there an algorithm that assigns weights to the network such that it correctly classifies the given examples and runs in time polynomial in the size of the input.

In the case of a one node neural net (figure 17.1), we are looking for a set of weights $\bar{w}$ such that
$\bar{w}\bar{x} \geq 0$ for positive examples and
$\bar{w}\bar{x} < 0$ for negative examples.
In other words, we are looking for a hyperplane that separates the set of positive and negative examples.

We have already seen the perceptron algorithm. However, this algorithm gives no bound on the number of examples needed to do the training and thus cannot be used here.

An alternate approach is based on the idea of viewing every example as a constraint in the hyperspace of weights (figure 17.2). Then the problem of finding weights can be translated to the problem of finding a non-empty region in the weights space that satisfies all the constraints. But this problem is solvable in time polynomial in the size of the input using linear programming. Thus a one-node neural net can be trained in polynomial time.

The question that arises is whether there is an algorithm that can achieve the same

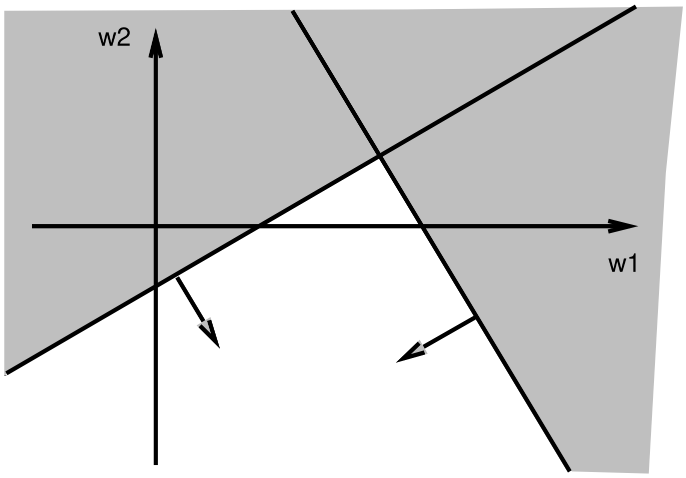

Figure 17.2: Problem of finding a feasible region of weights

result for larger neural networks. This problem was addressed by the works of Judd [3], and Blum and Rivest [1].

17.1.2 Judd's result

The main result of the Judd's work is showing that there are some intrinsic computational difficulties associated with training neural nets. This result is embodied in the following theorem.

Theorem 1 *(Judd) It is NP-complete to determine if a given neural net architecture can be trained to correctly classify a given set of examples*

Idea of the proof:
The reduction is from 3SAT. The following are conditions and properties of Judd's reduction:

1. The reduction produces multiple output networks (i.e., the neural network represents a mapping from inputs to a vector of outputs).
2. The architecture of the network varies with 3-SAT instance.
3. The network has "don't cares" in the outputs of the examples (partially specified examples).

4. The reduction uses only 3 training examples.

Judd also showed that the problem remains NP-complete even in the case when only two thirds of the training examples are required to be correctly classified. This implies that approximate training of neural nets is also intrinsically hard.

Note that the theorem and the result for approximate training do not mean that for some specific network and/or some specific set of training examples the answer to the problem cannot be found in the polynomial time. It merely states that there are networks and examples for which the decision about the existence of weights that correctly classify the examples in the training set is difficult.

Also note that the search problem (i.e., the problem of actually finding correct weights), is at least as hard as a decision problem.

17.1.3 Blum/Rivest result

The theorem proven by Judd does not specify any particular network or class of networks that is hard to train. This was done by Blum and Rivest[1], when they showed that training very simple networks can in fact be NP-complete. The network they used is shown in figure 17.3. It is a completely connected 3-node neural net, with 2 nodes in a hidden layer, binary inputs and sharp threshold functions.

Theorem 2 *(Blum/Rivest) Training the 3-node neural network is NP-complete.*

Proof: The standard approach to prove NP-completeness will be followed. First we will argue that the 3-node network is in NP. For this we will use the following lemma (given without proof).

Lemma 1 *The number of bits needed to encode any weight (and threshold) is at most* $O(n \lg n)$.

Since $O(n \lg n)$ bits are needed per weight and the number of weights is polynomial in n, one can write down all bits needed to encode the weights and thresholds in time polynomial in n. Then since it is possible to verify in polynomial time that the network with assigned weights correctly classifies all the examples, the 3-node network training problem is in NP.

Now we need to show that some known NP-complete problem is reducible to the problem of training the 3-node network. Before we show the reduction we will give

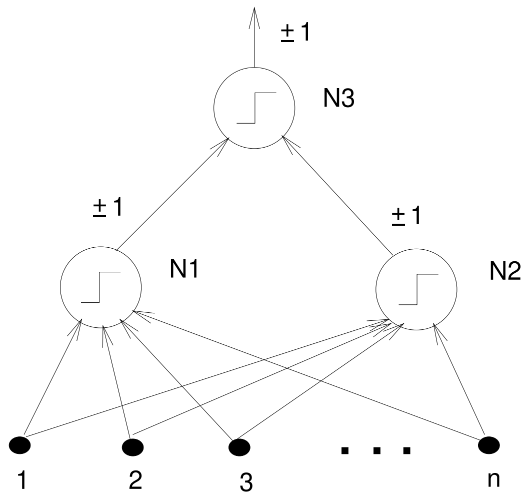

Figure 17.3: Three node neural net used by Blum and Rivest

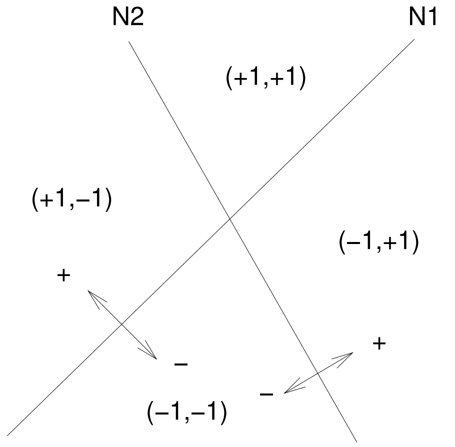

Figure 17.4: Division of hyperspace to 4 quadrants by nodes N_1 and N_2.

the geometric interpretation of the neural net training problem. Here the training example can be considered to be a labelled point in n-dimensional Boolean space $\{0,1\}^n$, and the linear functions thresholded by nodes N_1 and N_2 can be thought of as $n-1$ dimensional hyperplanes in this space. These hyperplanes define 4 quandrants of the space based on four possible combinations of outputs for N_1 and N_2 (see figure 17.4).

Note that the structure of the network excludes some combinations of the point labellings. This is because N_3 is a linear function of outputs from N_1 and N_2, and therefore it cannot output the same result for inputs $(+1+1)$ and $(-1-1)$ and the opposite result for $(+1-1)$ and $(-1+1)$.

Using the above geometric interpretation, the 3-node network training problem can be equivalently described as:
Given a collection of points $\{0,1\}^n$, where each point is labelled as positive or negative, does there exist either:

1. a single plane that separates positive and negative points, or

2. two planes that partition the space such that any one quadrant contains all the positive points and no negative points, or all the negative points and no positive points.

In the following we will base the reduction on the restricted version of the above geometric problem: *Given points from $\{0,1\}^n$, where each point is labelled positive or negative, do there exist 2 hyperplanes that partition the points such that one quadrant contains all positive points and no negative points?* Later, we will show how the general case can be transformed to this restricted case.

First, the Set-Splitting problem will be defined. This problem belongs to the family of NP-complete problems (we won't prove this here) and it will be used in the reduction.

Definition 1 *(Set-Splitting problem)*
Given a set of points $S = \{s_1, s_2, \cdots, s_n\}$ and family $C = \{c_1, c_2, \cdots, c_k\}$ of subsets of S, does there exist sets S_1 and S_2, satisfying $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$ such that:

$$\forall j \quad c_j \cap S_1 \neq \emptyset \text{ and } c_j \cap S_2 \neq \emptyset$$

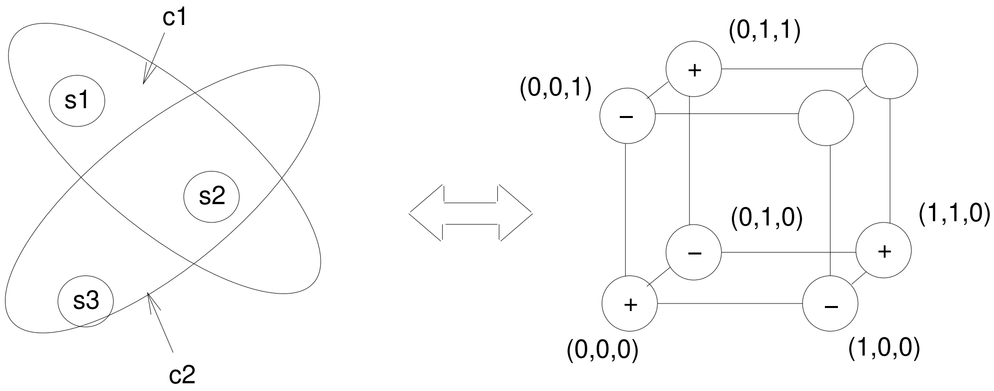

Figure 17.5: An example of the transformation of the Set-Splitting problem to the geometric problem.

Assume the following reduction from the Set-Splitting problem to the restricted version of the geometric problem. Given an instance of the Set-Splitting problem $S = \{s_1, s_2, \cdots, s_k\}$ and $C = \{c_1, c_2, \cdots, c_l\}$ where $c_j \subset S$ for $1 \leq j \leq l$, create the following points in $\{0,1\}^n$:

1. The origin 0^n is labelled positive.
2. For each s_i, point $p_i = 0^{i-1}10^{n-i}$ (i.e., $i-1$ $0's$ followed by 1, followed by $n-i$ $0's$) is labelled negative.
3. For each $c_j = \{s_{j_1}, s_{j_2}, \cdots, s_{j_m}\}$, there is a point labeled positive with bits set to 1 at positions $j_1, j_2, \cdots, j_m$ (i.e., $p_{j_1} + p_{j_2} + \cdots + p_{j_m}$).

Example of the transformation (figure 17.5)
Let $S = \{s_1, s_2, s_3\}$, $C = \{c_1, c_2\}$, $c_1 = \{s_1, s_2\}$, $c_2 = \{s_2, s_3\}$. Then the points created are:
positive points : (000), (110), (011);
negative points : (001), (010), (100).

Claim 1 *An instance of the Set-Splitting problem has a solution iff there exist a quadrant of* $\{0,1\}^n$ *with all the positive points and no negative points.*

($\Rightarrow$)
Assume that Set-Splitting has a solution S_1, S_2. Let planes P_1 and P_2 be two hyperplanes such that:

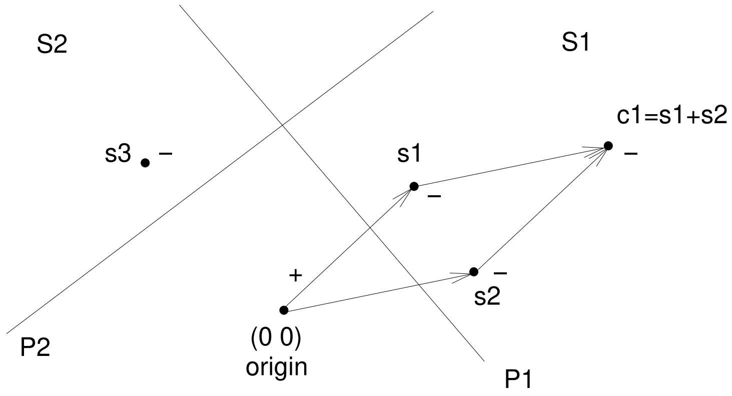

Figure 17.6:

P_1: $a_1x_1 + a_2x_2 + \cdots + a_nx_n = -\frac{1}{2}$ where:
$a_i = -1$ if $s_i \in S_1$
$a_i = n$ if $s_i \notin S_1$;
P_2: $b_1x_1 + b_2x_2 + \cdots + b_nx_n = -\frac{1}{2}$ where:
$b_i = -1$ if $s_i \in S_2$
$b_i = n$ if $s_i \notin S_2$.

Then plane P_1 separates all points $\{p_i\}$ corresponding to $s_i \in S_1$, and does not separate any positive point from the origin. Similarly P_2 separates all points corresponding to $s_i \in S_2$, and does not separate any positive point from the origin. Therefore the quadrant for which $\bar{a}\bar{x} > -\frac{1}{2}$ and $\bar{b}\bar{x} > -\frac{1}{2}$ contains all the positive points and no negative points.

($\Leftarrow$)
Assume that there exist hyperplanes P_1 and P_2 separating all positive points from negative points in $\{0,1\}^n$. Let S_1 be the set of all points separated from the origin by P_1 and S_2 those separated by P_2 (break ties arbitrarily). Note that the points of S_1 and S_2 cover all points corresponding to S.

Now for the purpose of contradiction assume that all negative points $p_{j1}, p_{j2}, \cdots, p_{jm}$ corresponding to members of $c_j = \{s_{j1}, s_{j2}, \cdots, s_{jm}\}$ are separated from the origin by P_1 (i.e., that $c_j \subset S_1$). Then also $p_{j1} + p_{j2} + \cdots + p_{jm}$ must be separated from the origin by P_1 (see figure 17.6). As $p_{j1} + p_{j2} + \cdots + p_{jm}$ is exactly a point corresponding to c_j, it is labelled positive. But this contradicts with the fact that P_1 and P_2 separate positive and negative points. Therefore $c_j \subset S_1$ does not hold. Similarly, $c_j \subset S_2$ does not hold.

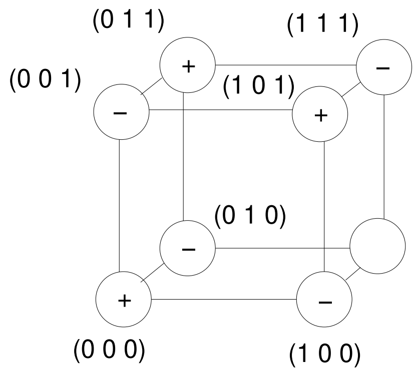

Figure 17.7: The extension enforcing positive points to one quadrant

This concludes the proof of the validity of the reduction for the case when positive points are confined to one quadrant. However the above reduction does not prove the general case.

In order to use the above idea for general case we need to enforce that all positive points be in one quadrant. This can be done by adding three new dimensions $x_{n+1}, x_{n+2}, x_{n+3}$ and the following set of additional points (see figure 17.7):

- positive points : $(\underbrace{00\cdots0}_{n}\ \ 101)$, $(00\cdots0\ \ 011)$;
- negative points : $(00\cdots0\ \ 100)$, $(00\cdots0\ \ 010)$, $(00\cdots0\ \ 001)$, $(00\cdots0\ \ 111)$.

Using examples created in this manner, the only way positive and negative points can be separated by 2 hyperplanes is by confining all positive points to one quadrant. But this allows us to use the reduction described above and transform Set-Splitting problem to the problem of finding weights of the 3-node neural network. Therefore training the 3-node network is NP-complete.

■

17.2 Approximation properties of Neural Nets

In this section we will focus on the problem "Can neural network approximate any bounded continuous function?". This problem was explored by Cybenko [2] who showed that networks with 2-hidden layers are sufficient to achieve that. In the following a slightly modified version of Cybenko's theorem will be presented.

17.2.1 Cybenko's result

Theorem 3 *(Cybenko) Let f be a bounded continuous function mapping $[-1,1]^n \to R$ and σ be any "sigmoid" function such that:*

$$\sigma(x) \to 1 \ \ as \ x \to +\infty$$

$$\sigma(x) \to 0 \ \ as \ x \to -\infty.$$

Then $\forall \epsilon > 0$ there exists a neural network (NN) with nonlinearity σ and two hidden layers such that:

$$\forall x \in [-1,1]^n \qquad\qquad | f(x) - NN(x) | < \epsilon$$

Proof sketch : A direct approach to approximate the function is to partition the input space to regions and approximate values of the function in every region by constant. In the proof the same "partitioning" idea will be used.

Let R be a set of regions that cover the input space. Then function f can be approximated as :

$$f(x) = \sum_{r \in R} I_r(x) f(x_r)$$

where :

$I_r(x)$ is function that is 1 on any point in region r and 0 elsewhere;
$f(x_r)$ is an approximation of f on region r (i.e., x_r is a point in region r).

Assuming that we know how to compute $I_r(x)$ for all regions, then the function can be approximated by the network with the structure in figure 17.8, where weights correspond to values of $f(x_r)$ and the output node has no squashing function.

Now we need to show how $I_r(x)$ can be computed. For simplicity assume that the number of input variables is 2 and that input regions are squares patches (see 17.9). The idea of how to compute $I_r(x)$ is to:

- use two threshold functions per dimension such that both of them are 1 in the area of patch and less than 1 ouside;

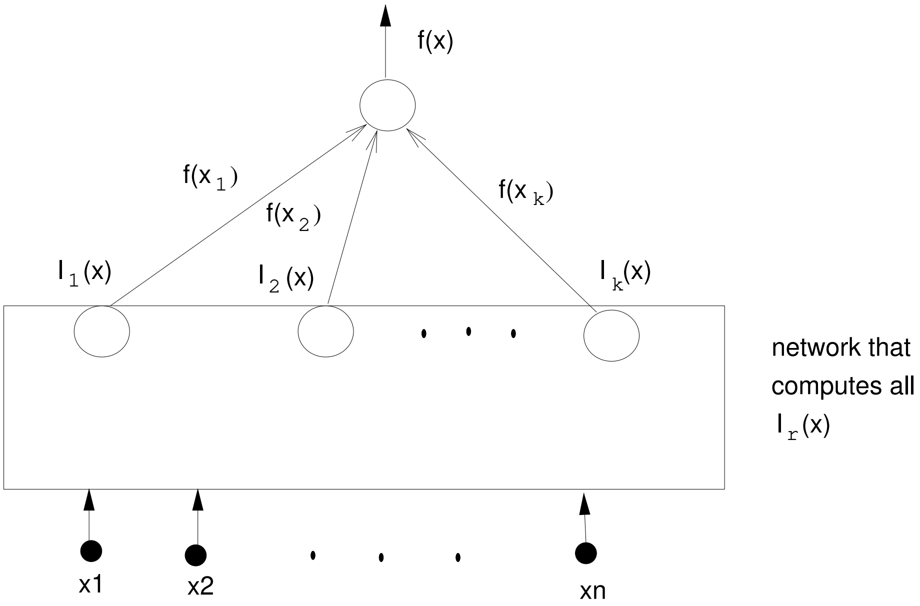

Figure 17.8: Structure of the network approximating continuous function

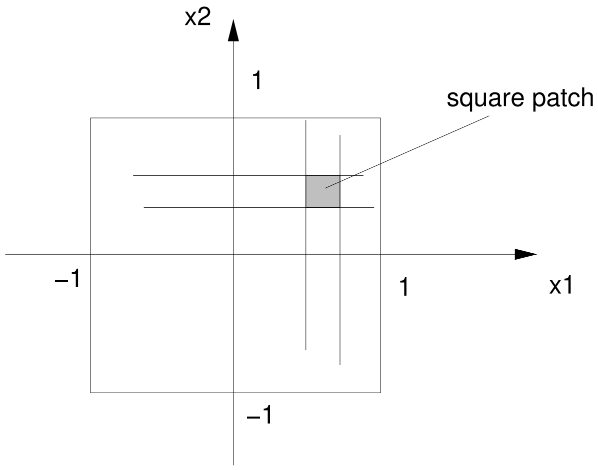

Figure 17.9: Input space partitioned to square patches ($n = 2$)

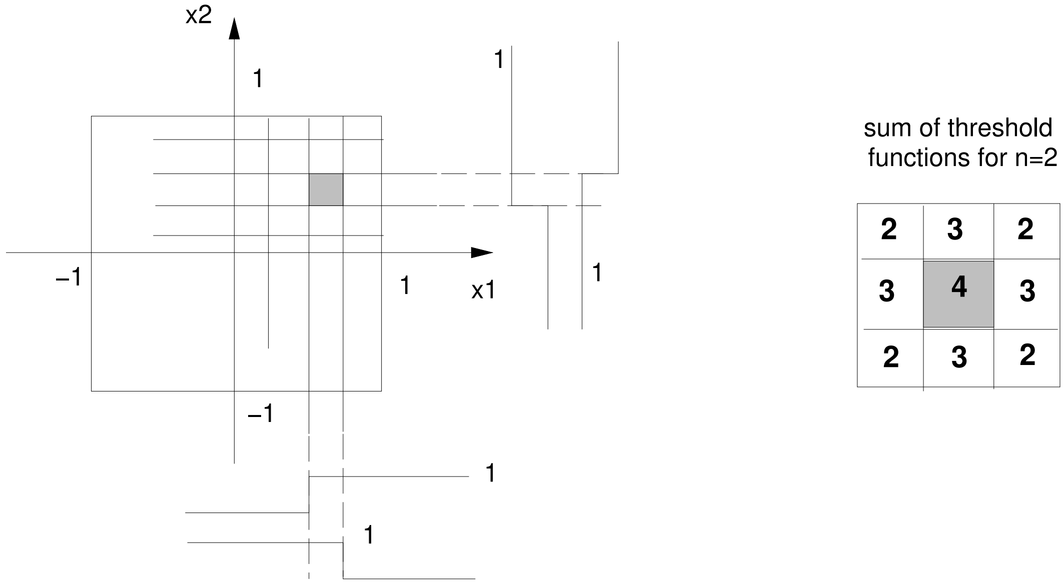

Figure 17.10: An example of computation of $I_r(x)$ using threshold functions

- add all such threshold functions and run the result through threshold of a $2n - 0.5$, where n is the number of dimensions

The idea for $n = 2$ is illustrated in figure 17.10, where the threshold of 3.5 filters all $x's$ outside the patch.

Note that the construction of $I_r(x)$ as described above can be realized by two layer network with sigmoid nonlinearities. As function to be approximated is continuous the required error bound can be always satisfied by altering the the size of patches. Therefore it is possible to construct a neural net with 2 hidden layers that approximates any bounded continuous function with error at most ϵ, which concludes the proof.

Note that no assumption on the number of nodes used was made by the theorem. This is because the number of nodes in neural net grows with decreasing ϵ (i.e., smaller ϵ values increase the number of patches that cover the input space).

References

[1] A. Blum, R. Rivest. Training a 3-node nerural net is NP-complete. In D.S. Touretzky, editor, *Advances in Neural Information Processin Systems I*, Morgan Kaufmann, pp 494-501, 1989.

[2] G. Cybenko. Continuous-valued neural networks with two hidden layers are sufficient. (unpublished), 1988

[3] J.S. Judd. Neural network design and the complexity of learning. MIT Press, 1990