Lecture 7: Fundamentals of Learning Theory:Perceptorn I

Haim Sompolinsky, MCB 131, Tuesday, 16 February, 2010

1 Introduction to Learning in Neuronal Networks

1. Formalizing computation in a neuronal network:

Network is an input-output device. Its computation consists of realizing a desired function. This definition requires specification
of the domain of the inputs and their representation in the network, and the representation of the output.

2. Complexity of a neural architecture. Once we have specified the computation of a network we would like to compare the different classes of functions that that a given network architecture can realize. These are the notion of complexity, capacity of effective dimensionality.

3. Supervised Learning: Given an appropriate network architecture, how do we find the sepcific parameters for solving a paricular task?
Here we will adopt the basic framework of Supervised Learning: Training data in the form of labeled examples; and a set of rules how the synapses evolve such that the examples are correcly learned (at least approximately). To assess the performance, we adopt a measure of error or loss for each example.

4. Learning Algorithms and Synaptic Plasticity: A brief comment.

5. The simplest unit of neural computatoin - Perceptron

2 Perceptron and Neural Computation: Historical milestones

1940's

Working from the beginnings of neuroscience, Warren McCulloch and Walter Pitts in their 1943 paper, "A Logical Calculus of Ideas Immanent in Nervous

Activity," contended that neurons with a binary threshold activation function, a linear threshold device, were analogous to first order logic sentences. The McCulloch-Pitts neuron worked by inputting either a 1 or 0 for each of the inputs, where 1 represented true and 0 false. Likewise, the threshold was given a real value, say 1, which would allow for a 0 or 1 output if the threshold was met or exceeded.

One of the difficulties with the McCulloch-Pitts neuron was the lack of a notion of learning.

In 1949, Donald Hebb in his book, The Organization of Behavior, introduced a synaptic learning rule that is known as Hebb's rule. He states, "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." [1] Hebb was proposing not only that, when two neurons fire together the connection between the neurons is strengthened, but also that this activity is one of the fundamental operations necessary for learning and memory.

1950-60's

Frank Rosenblatt, using the McCulloch-Pitts neuron and the findings of Hebb, went on to develop the first perceptron. Importantly, the perceptron could learn through incremental modification of its weights. He discussed the perceptron in his 1962 book, Principles of Neurodynamics where he made the overly overblown remark:

"Given an elementary α-perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always yield a solution to C(W) in finite time..."

With these types of remarks Rosenblatt had drawn a line in the sand between those in support of perceptron styled research and the more traditional symbol manipulation projects being performed by Marvin Minsky . As a result, in 1969, Minsky co-authored with Seymour Papert , **Perceptrons: An Introduction to Computational Geometry**. In this work they attacked the limitations of the perceptron. They showed that the perceptron could only solve linearly separable functions. Of particular interest was the fact that the perceptron still could not solve the XOR and NXOR functions. Likewise, Minsky and Papert stated that the style of research being done on the perceptron was doomed to failure because of these limitations. This was, of course, Minsky's equally ill-timed remark. As a result, very little research was done in the area until about the 1980's.

1980-90's

What would come to resolve many of the difficulties was the creation of more complex neural networks and more powerful learning algorithms. Several important classes of networks and learning algorithms for them were presented in

1986 of in a widely publicized collection PDP: Parallel Distributed Processing, including the BackProp and Bolzman Machine. These networks have more complex architecture than the perceptron. As a result, the networks were able to solve more difficult problems, and they have grown considerably more complex.

Other important developments: The appearance of the John Hopfield, the theoretical developments in dynamic theory and statistical physics, and the development of powerful computational tools.

Warren McCulloch and Walter Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity", 1943, Bulletin of Mathematical Biophysics 5:115-133.

Hebb, Donald O. (1949). The Organization of Behavior. New York: Wiley, pg. 62.

By the study of neural networks such as the Perceptron, Rosenblatt hoped that "the fundamental laws of organization which are common to all information handling systems, machines and men included, may eventually be understood."

"Principles of neurodynamics: Perceptrons and the theory of brain mechanisms" (Spartan Books, 1962)

Marvin Minsky and Seymour Papert, 1972 (2nd edition with corrections, first edition 1969) Perceptrons: An Introduction to Computational Geometry, The MIT Press, Cambridge MA, ISBN: 0 262 63022 2.

J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", Proceedings of the National Academy of Sciences of the USA, vol. 79 no. 8 pp. 2554-2558, April 1982.

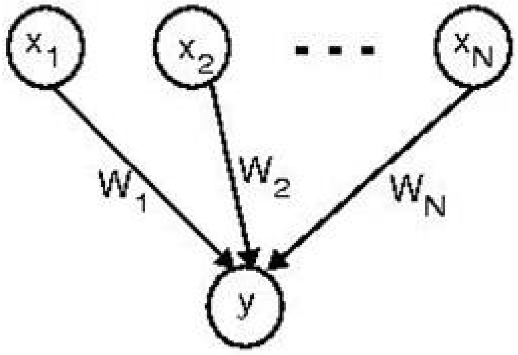
Rumelhart, D.E., J.L. McClelland and the PDP Research Group (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volumes 1-2 Cambridge, MA: MIT Press

3 Perceptron Decision Surface

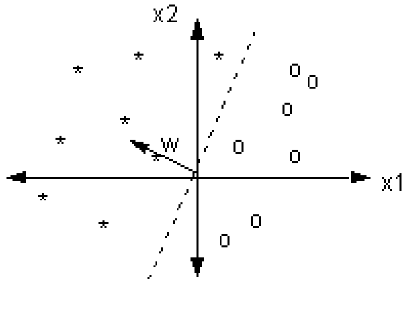
The simplest network that implements interesting input-output function is the Perceptron. It has a single layer of weights connecting the inputs and output. Formally, the perceptron is defined by

$$y = sign(\sum_{i=1}^{N} W_i x_i - \theta) \tag{1}$$

which we also write in our usual notation as $y = sign(W^T x - \theta)$. θ is the threshold parameter. Often we will ignore the threshold in the analysis of the perceptron (and other topics), because we can instead see the threshold as another synaptic weight, which is given the constant input -1. This is so because $W^T x - \theta = (W, \theta)^T (x, -1)$.

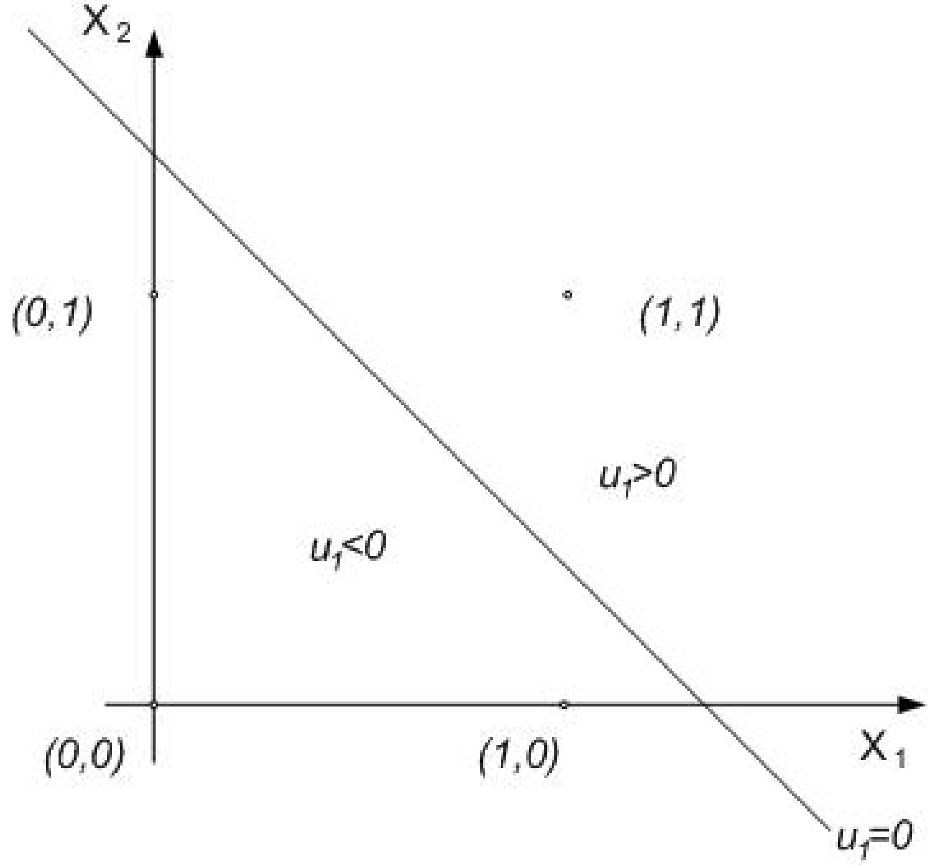


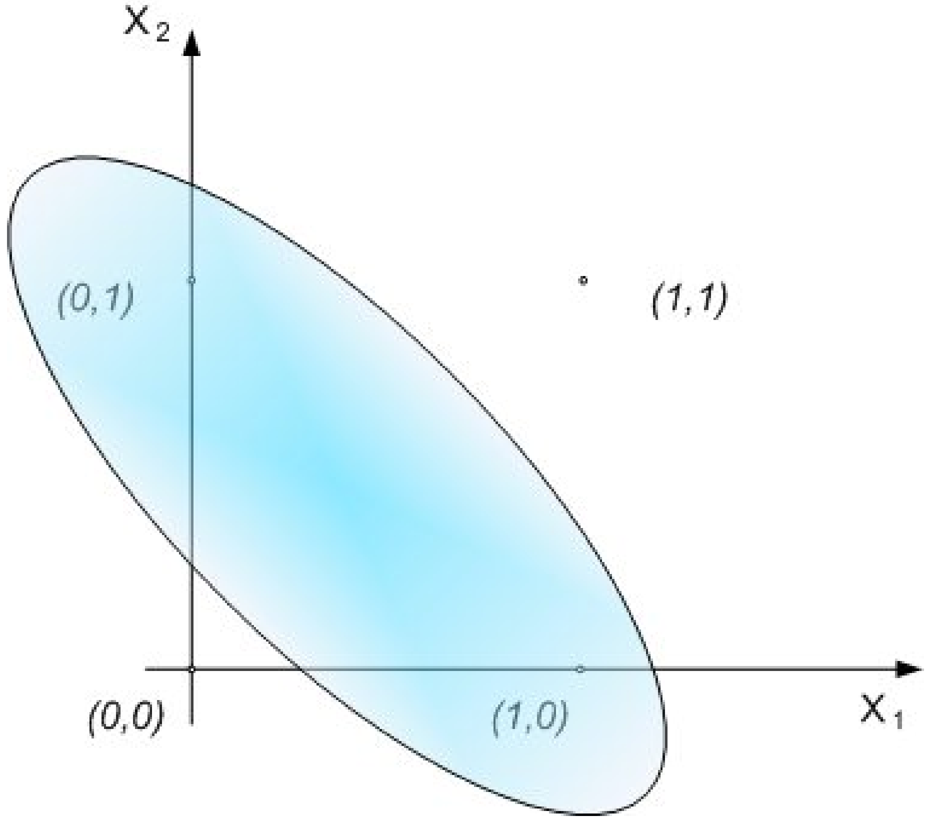
Perceptron's decision surface: Perceptron is an example of system that realizes a dichotomy, i.e., a function from R^N to $\{-1, 1\}$(or $\{0, 1\}$; the choice of the binary values is a matter of convenience only). It is easy to visualize the action of the perceptron in geometric terms because W and x have the same dimensionality, N. In the present case, the surface which divides the set of points into two classes is a hyperplane. Its defined as the set of points x which obey $W^T x = 0$ This defines a hyperplanes that passes through the origin and is perpendicular to the vector W. In the case of a non-zero threshold, the plane is defined by the equation $W^T x - \theta = 0$. It is perpendicular to W but is separated from the origin along the direction of W by a distance $\theta/|W|$. It is costumary to call the plane itself W. Thus, the set of dichotomies that are realizable by the Perceptron are those that geometrically, are linearly separable. This is of course a small subset of all possible dichotomies.



$Figure\,2:\ Perceptron\,Decision\,Surface$

An example of dichotomies in $N = 2$ is shown in Figure 3.





$Figure\,3:\ Linearly\,Separable\,(AND\,gate)\ and\,Inseperable\ (XOR)\,Dichotomies\,in\,N=2$

Here the inputs are also binary variables, so that the system can be thought of as a bolean gate. Cleraly the AND gate (separating the (1,1) corner of the unit square from the other corners) is linearly separable while XOR (separating the (1,0) and (0,1)) is not. Finally note that there is a redundancy in the parameters (W, θ), since the overall magnitude of the input to the threshold device is immaterial.

4 The Capacity of the Perceptron

It is often desirable to measure the complexity of a given neural circuit (or any computing device). From the present perspective, the difficulty in the notion of complexity is that a given network (where all its parameters fixed) implements exaclty one input-output function. It is not obvious how we compare different functions in terms of complexity or whether such comparison is relevant for learning.

So we need to discuss the complexity of architectures not of a specific network. An architecture is a set of networks defined though the common architectual constraints. For instance, all networks that have two layer of synapses, or all recurrent networks of a given size etc. In Machine Learning, we talk about the hypothesis set. We now ask how complex is a given neuronal architecure? It would be useful to quantify this notion of complexity in such a way that we can compare meaningfully different architectures or different types of neurons etc.

One way to measure the complexity of an architecture is to count the number of possible funcions that are implemented by the networks with the given architecture, i.e., to equate complexity with the appropriate measure of capacity. In our particular case, we will ask: How many dichotomies can perceptron (architecture) implement? The simple answer is: infinite. If the domain of inputs is a continuous one, then every vector W implements a different dichotomoy, since it slices the space of inputs in a different way than any other vector (except for the above mentioned redundancy in mangnitude). Therefore in order to 'save' this notion of complexity we need to restrict the input space.

A useful definition of capacity assumes that we constrain ourselves to a finite number of inputs $\{x^1, ..., x^P\}$. For now, the capacity of a network is the number of functions (dichotomies) that can be implemented by a given architecture network on this fixed set of inputs (this will be refined later).

Let us apply this measure on the perceptron. The architecture is specified by all systems of the type $y = sign(W^T x)$ where W is an arbitrary N dimensional vector. Then the set of possible dichotomies over a fixed set of inputs $\{x^1, ..., x^P\}$ is $\big(sign(W^T x^1), ..., sign(W^T x^P)\big) \mid W \in R^N\}$. The size of this set is the capacity of the network. Generally speaking, the capacity of it depends upon the specific set of inputs $\{x^1, ..., x^P\}$, hence it does not seem to be a particularly useful notion. However, it is a remarkable property of the perceptron that the capacity is independent of the identity of the set of input vectors, as long as they are *general position.*

General position is the condition that $\{x^1, ..., x^P\}$ has no subset of size N or less that is linearly dependent. It turns out that if the inputs are in general position then the capacity, as defined before, does not depend on the inputs. This allows us to write it as $C(P, N)$ - in other words, the capacity depends upon the dimension of the input (N) and the number of inputs (P) *only.*

An immediate observation is that $C(P, N) \leq 2^P$, since the total number of dichotomies over P inputs is 2^P. But we want to know exactly what $C(P, N)$ is equal to. Thankfully, there is a theorem that gives us exactly that, known as Cover's Function Counting Theorem:

Cover's Function Counting Theorem (Cover 1966):

Theorem: Let $\left\{x_1, ..\vec{x}^P\right\}$ be vectors in R^N, that are in general position. Then the number of possible dichotomies that can defined on them by means of a plane through the origin is:

$$C(P,N) = 2\sum_{k=0}^{N-1}\binom{P-1}{k} \tag{2}$$

Reminder:for $m \leq n$, $\binom{n}{m} = \frac{n!}{m!(n-m)!}$. For $m < n$, we define $\binom{n}{m} = 0$.

Note that if $P \leq N$ then it is natural to expect the vectors to not be linearly dependent – which means that they are in general position and Cover's Theorem holds for them. However, if $P > N$ then the input vectors must be linearly dependent. Even so, Cover's theorem holds if they fulfill the weaker demand that they be in general position, which is obeyed by all generic generators of inpu vectors. Thus Cover's theorem is widely applicable.

Some consequences of Cover's theorem:

1. For $P \leq N$ the above sum is limited by $P-1$, so it can be rewritten as

$$C(P,N) = 2\sum_{k=0}^{P-1}\binom{P-1}{k} = 2(1+1)^{P-1} = 2^P \tag{3}$$

(using the familiar binomial expansion). In other words, *all* possible dichotomies can be realized.

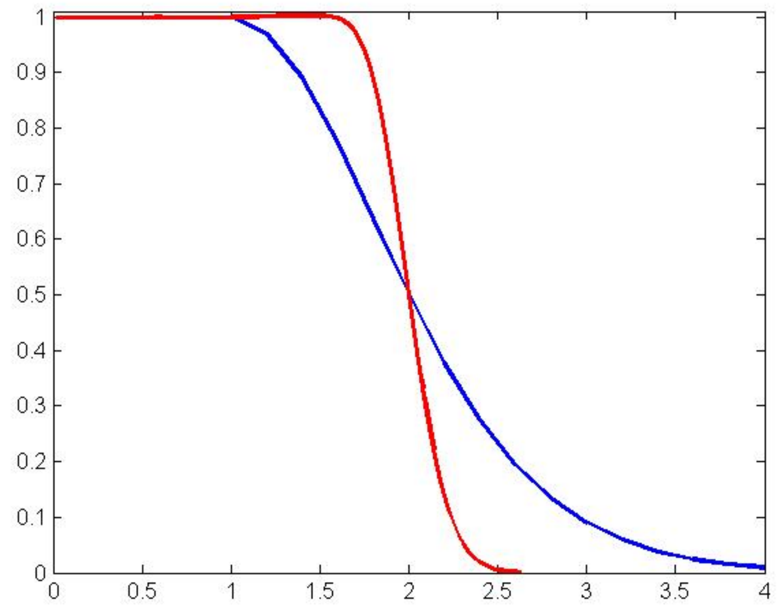
2. for $P = 2N$,

$$C(2N,N) = 2\sum_{k=0}^{N-1}\binom{2N-1}{k} = 2\frac{1}{2}2^{2N-1} = 2^{P-1} \tag{4}$$

This is so because the expression for C contains exactly one half of the full binomial expansion of $(1+1)^{2N-1}$, and we know that this expansion is symmetrical (for example, the first few (odd) binomial expansions are $(1,1)$, $(1,3,3,1)$, and $(1,5,10,10,5,1)$. We conclude that in this case exactly half of all possible dichotomies are able to be realized.

3. For $P \gg N$, it is easy to see that $C(P,N) \sim AP^N$ for some $A > 0$.

When P is large compared to N the number of dichotomies that can be implemented still grows with P, but the number of dichotomies that can be implemented in proportion to the number of possible dichotomies shrinks, since the total number of possible dichotomies is 2^P. Figure 4 shows this phenomena, in logarithmic scale: for small P, the number of dichotomies is maximal, i.e. 2^P, and thus linear in the logarithmic scale. When P becomes large, the number of possible dichotomies becomes only polynomial, and hence logarithmic in the logarithmic scale.



$Figure\,4: \; Fraction\,of\,linearly\,separable\,dichotomies\,vs.\,P/N\;for\,N = 5\,and\,65$

4. Another way to look at the behavior of $C(P, N)$ is to allow both P and N to approach ∞, but to keep them proportional, i.e. $\frac{P}{N} = \alpha$. In this case, as $N \to \infty$, $C(P, N)$ vs. $\frac{P}{N}$ becomes a step function, as shown in figure 5. Note that the shape of the step function must of course still obey the properties that we found before, in particular $\frac{C(2N,N)}{2^N} = 0.5$. This is in fact the value around which the step occurs, i.e. below the critical value of $\frac{P}{N} = 2$ we see that virtually all of the dichotomies are possible, and above that value virtually none are possible.

Proof will be explained in the next lecture:

Proof of Cover's Theorem: Start with P points in general position. We now assume that there are $C(P, N)$ dichotomies possible on them, and ask how many dichotomies are possible if another point (in general position) is added, i.e. what is the value of $C(P + 1\,()\,, N)$. In this way we will set up a recursive expression for $C(P, N)$.

Let $(b_1, ..., b_P)$ be a dichotomy realizable by the network over the set of P inputs – in other words, $b_i \in \{-1, +1\}$ for every $i = 1..P$, and there is a set of weights W so that $\big(sign(W^T x^1), ..., sign(W^T x^P)\big) = (b_1, ..., b_P)$. Using this same W, we get a dichotomy over the set of $P + 1$ inputs:

$$\big(sign(Wx^1), ..., sign(Wx^{P+1})\big) = (b_1, ..., b_P, sign(Wx^{P+1})) \qquad (5)$$

Thus for every dichotomy over P points there is a dichotomy over $P+1$ points. Note that different dichotomies over P points define different dichotomies over $P+1$ points, since they differ somewhere in the first P co-ordinates.

Now, we have seen that $(b_1, ..., b_P, sign(Wx^{P+1}))$ is possible (for every $(b_1, ..., b_P)$). Perhaps the additional dichotomy $(b_1, ..., b_P, -sign(Wx^{P+1}))$ (reversing the sign of the last co-ordinate) is also possible, by some other set of weights? In this way $C(P+1, N)$ can be higher than $C(P, N)$. Let us write

$$C(P+1, N) = C(P, N) + D \tag{6}$$

Our goal is to find D, the number of additional dichotomies, by with we will find a recursive formula for $C(P, N)$.

Let us assume that one of the weight vectors W that generates $(b_1, ..., b_P)$ passes directly through x^{P+1}, as shown in the figure. In this case, it is clear that by slight changes in the angle of the hyperplane (i.e. in the weights W which define the hyperplane) we will be able to move the hyperplane slightly to this side or the other of x^{P+1} - thus getting a value of either $+1$ or -1 on it. Thus in this case, *both* $(b_1, ..., b_P, +1)$ and $(b_1, ..., b_P, -1)$ are possible, and there is one additional possible dichotomy beyond $C(P, N)$ (that is counted in D).

$Figure\, 4:\ Geometry\, of\, Cover's\, Theorem$

On the other hand, if no hyperplane that passes through x^{P+1} (and generates $(b_1, ..., b_P)$ on the first P vectors) exists, then it means that the point lies in one side of all the planes that generate the old dichotomoy, hence we will *not* be able to achieve both dichotomies, unlike before. We have thus seen that D is the number of those dichotomies over P points that are realized by a hyperplane that *passes through a certain fixed point* x^{P+1} (which is in general position with the other points, of course). Now, by forcing the hyperplane to pass through a certain fixed point, we are in fact moving the problem to one in $N-1$ dimensions, instead of N. This can be understood if the point is on the x axis, for example - then the hyperplane has $N-1$ axes left to "work with" (if the point is not on the x axis, then rotate the axes of the space around to get the point on the x axis, and this of course has no effect on the geometry of the problem). In conclusion, $D = C(P, N-1)$, and the recursion formula is

$$C(P+1, N) = C(P, N) + C(P, N-1) \tag{7}$$

We now prove the theorem by induction. Let us assume that

$$C(P, N) = 2 \sum_{k=0}^{N-1} \binom{P-1}{k}$$

holds up to P and N [Note that it trivially holds for $P = 1$ and all N, since it gives $C(1, N) = 2$ as expected, since one point in N dimensions can be dichotomized with the two labels by a hyperplane. Then,

$$\begin{aligned} C(P+1,N) &= 2\sum_{k=0}^{N-1}\binom{P-1}{k} + 2\sum_{k=0}^{N-2}\binom{P-1}{k} \\ &= 2\sum_{k=0}^{N-1}\binom{P-1}{k} + 2\sum_{k=0}^{N-1}\binom{P-1}{k-1} = 2\sum_{k=0}^{N-1}\binom{P}{k} \end{aligned} \tag{8}$$

where we have used $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ [and used the convension that $\binom{n}{k} = 0$ for $k < 0$] .