Lecture 8: VC Dimension. Alternative NN architectures

- ◊ VC Dimension for linear decision surfaces
- ◊ VC Dimension for neural networks
- ◊ Sparse Distributed Memories

Recall from last time: Shattering a Set of Instances

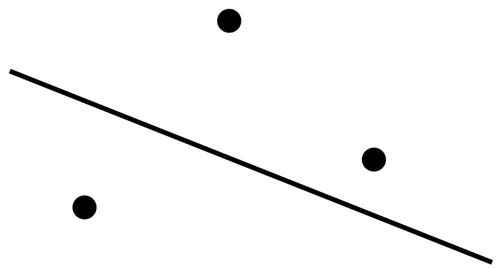
Definition: A **dichotomy** of a set S is a partition of S into two disjoint subsets.

Definition: A set of instances S is **shattered** by hypothesis space H if and only if for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.

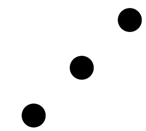
The Vapnik-Chervonenkis (VC) Dimension

Definition: The **Vapnik-Chervonenkis dimension**, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) \equiv \infty$.

VC Dimension of Linear Decision Surfaces



(a)



(b)

For an n-dimensional space, VC dimension of linear estimators is $n+1$.

Sample Complexity from VC Dimension

A lower and upper bound on the number of examples m have been established. Both depend on $\log(\frac{1}{\delta}$, and $\frac{1}{\epsilon}$.

The upper bound depends on $VC(H)$ (the VC dimension of the hypothesis space) and has an additional factor of $\log\frac{1}{\epsilon}$.

This is a lot tighter than the previous bound that we had, which depended on $\log|H|$*!* Why?

If $VC(H) = d$, then H can shatter d instances, which requires 2^d distinct hypotheses. Hence $d \leq \log_2|H|$.

The lower bound depends on $VC(C)$ (the VC dimension of the concept space).

VC Dimension of Neural Networks

Let G be a directed layered graph with n input nodes, s internal nodes and 1 output node, with each internal node having at most r inputs. Let C be a concept class of VC dimension d, corresponding to what can be represented by the internal nodes. Let C_G be the set of unfctions that can be represented by D.

Then $VC(C_G) \leq 2ds \log(es)$.

Immediate consequence: for networks of perceptrons, the VC dimension is:

$$VC(C_G) \leq 2(r+1)s \log(es).$$

And the bad news...

Sigmoid-like functions *can* have infinite VC dimension! E.g.

$$\frac{1}{1+e^{-x}} + cx^3 e^{-x^2} \sin x$$

(see Macintyre and Sontag, 1993).

However: the usual sigmoid function, as well as the hyperbolic tangent, have finite VC dimension! :-)

But: it is doubly exponential... :-(

However, in practice, neural networks seem to approximate well even with a lot fewer examples (sometimes fewer than the number of weights).

Alternative analyses (see, e.g. Bartlett, 1996) suggest that the error may be related to the *magnitude* of the weights, rather than the number of weights, if the nodes are kept in their linear regions.

Associative Memories

- Studied since the beginning of neural nets (e.g. Hopfield nets) as mechanisms for storing and retrieving data (rather than approximating a function)
- Trying to mimic the way in which the human memory has very large capacity, and fast access, but "inexact" memory
- Main idea: the inputs are used as an "address" in a memory, to retrieve more data (could be the class label, other data items that are related, etc)

Sparse Distributed Memories (Kanerva coding)

- The address space is far larger than the number of locations that we can afford
- Therefore, we will sample the address space and only have locations for a few samples
- When we need to retrieve something, we go to all locations within a"small" Hamming distance, retrieve all their contents, and cumulate them; then we ake just the sign of each location
- When we need to store something, we go to all "close" locations, and add a -1 for every 0 in our pattern and a +1 for every 1.
- Why does it work?
  - high-dimensional space, so data is very spread out
  - but at the same time, it is "close"to intermediate points
  - when data is retrieved, the desired item will be at every location, plus some other items (a lot fewer)
- It also seems to eliminate noise...

SDMs as feed-forward neural networks

- The address matrix corresponds to the input weights (which go into the hidden units)
- The content matix corresponds to the output weights (which link the hidden units to the output units)
- The hidden units are thresholded by the Hamming distance
- The output units use the signum of what is computed
- *But the input weights are decided upon once, and then they are fixed!*
- *And the threshold is chosen such that just a few locations get activated at once*
- This means that training is fast, but less flexible
- Which one is mor believable to you as a model of human memory?

CMACs

- CMACs were proposed by Albus as a model for the cerebellum (1971)
- Used widely today especially in robotics, where problems are in continuous state spaces of few dimensions

Can achieve much finer discretization than with a fixed grid of the same size!

SDMs as CMACs

The addresses in a CMAC are NOT placed randomly, but systematically according to the discretization

This makes activation computation efficient

Also, CMACs are trained by error correction for the weights (which is a gradient-based mechanism)