Introduction

- Property of primary significance in net: learn from its environment, and improve its performance through learning.
- Iterative adjustment of synaptic weights.
- Learning: hard to define.
  - One definition by Mendel and McClaren: Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

Overview

Organization of this chapter:
1. Five basic learning rules
   - error correction, Hebbian, memory-based, competitive, and Boltzmann
2. Learning paradigms
   - credit assignment problem, supervised learning, unsupervised learning
3. Learning tasks, memory, and adaptation
4. Probabilistic and statistical aspects of learning
Error-Correction Learning

- Input $x(n)$, output $y_k(n)$, and desired response or target output $d_k(n)$.
- Error signal $e_k(n) = d_k(n) - y_k(n)$
- $e_k(n)$ actuates a control mechanism that gradually adjust the synaptic weights, to minimize the cost function (or index of performance):
  \[ E(n) = \frac{1}{2} e_k^2(n) \]
- When synaptic weights reach a steady state, learning is stopped.

Memory-Based Learning

- All (or most) past experiences are explicitly stored, as input-target pairs $\{(x_i, d_i)\}_{i=1}^N$.
- Two classes $C_1, C_2$.
- Given a new input $x_{\text{test}}$, determine class based on local neighborhood of $x_{\text{test}}$.
  - Criterion used for determining the neighborhood
  - Learning rule applied to the neighborhood of the input, within the set of training examples.
- Widrow-Hoff rule, with learning rate $\eta$:
  \[ \Delta w_{kj}(n) = \eta e_k(n)x_j(n) \]
- With that, we can update the weights:
  \[ w_{kj}(n + 1) = w_{kj}(n) + \Delta w_{kj}(n) \]
- There is a sound theoretical reason for doing this, which we will discuss later.

Memory-Based Learning: Nearest Neighbor

- A set of instances observed so far:
  \[ X = \{x_1, x_2, \ldots, x_N\} \]
- Nearest neighbor $x'_{N} \in X$ of $x_{\text{test}}$:
  \[ \min_i d(x_i, x_{\text{test}}) = d(x'_{N}, x_{\text{test}}) \]
  where $d(\cdot, \cdot)$ is the Euclidean distance.
- $x_{\text{test}}$ is classified as the same class as $x'_{N}$.
- Cover and Hart (1967): error (for Nearest Neighbor) is at most twice that of the optimal (Bayes probability of error), given
  - The classified examples are independently and identically distributed.
  - The sample size $N$ is infinitely large.
Memory-Based Learning: \( k \)-Nearest Neighbor

- Identify \( k \) classified patterns that lie nearest to the test vector \( x_{\text{test}} \), for some integer \( k \).
- Assign \( x_{\text{test}} \) to the class that is most frequently represented by the \( k \) neighbors (use majority vote).
- In effect, it is like averaging. It can deal with outliers. The input \( x \) above will be classified as 1.

Hebbian Synapses

- Time-dependent mechanism
- Local mechanism
- Interactive mechanism
- Correlative/conjunctive mechanism

Strong evidence for Hebbian plasticity in the Hippocampus (brain region).

Classification of Synaptic Plasticity

<table>
<thead>
<tr>
<th>Type</th>
<th>Positively correlated</th>
<th>Negatively correlated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hebbian</td>
<td>Strengthen</td>
<td>Weaken</td>
</tr>
<tr>
<td>Anti-Hebbian</td>
<td>Weaken</td>
<td>Strengthen</td>
</tr>
<tr>
<td>Non-Hebbian</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
</tbody>
</table>

Hebbian Learning


  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 changes take place in one or both cells such that A’s efficiency as one of the cells firing B, is increased.

- **Hebbian synapse**
  - If two neurons on either side of a synapse are activated simultaneously, the synapse is strengthened.
  - If they are activated asynchronously, the synapse is weakened or eliminated. (This part was not mentioned in Hebb.)
Mathematical Models of Synaptic Plasticity

- General form: \( \Delta w_{kj}(n) = F(y_k(n), x_j(n)) \)
- Hebbian learning (with learning rate \( \eta \)): \( \Delta w_{kj}(n) = \eta y_k(n) x_j(n) \)
- Covariance rule: \( \Delta w_{kj} = \eta (x_j - \bar{x})(y_k - \bar{y}) \)

Covariance Rule (Sejnowski 1977)

\( \Delta w_{kj} = \eta (x_j - \bar{x})(y_k - \bar{y}) \)
- Convergence to a nontrivial state
- Prediction of both potentiation and depression.
- Observations:
  - Weight enhanced when both pre- and post-synaptic activities are above average.
  - Weight depressed when
    * Presynaptic activity more than average, and postsynaptic activity less than average.
    * Presynaptic activity less than average, and postsynaptic activity more than average.

Competitive Learning

- Output neurons compete with each other for a chance to become active.
- Highly suited to discover statistically salient features (that may aid in classification).
- Three basic elements:
  - Same type of neurons with different weight sets, so that they respond differently to a given set of inputs.
  - A limit imposed on the strength of each neuron.
  - Competition mechanism, to choose one winner: winner-takes-all neuron.

Inputs and Weights Seen as Vectors in High-dimensional Space

- Inputs and weights can be seen as vectors: \( \mathbf{x} \) and \( \mathbf{w}_k \). Note that the weight vector belongs to a certain output neuron \( k \), and thus the index.
**Competitive Learning: Example**

- Single layer, feedforward excitatory, and lateral inhibitory connections
- Winner selection
  \[ y_k = \begin{cases} 
  1 & \text{if } v_k > v_j \text{ for all } j, j \neq k \\
  0 & \text{otherwise} 
\end{cases} \]
- Limit: \( \sum_j w_{kj} = 1 \) for all \( k \).
- Adaptation:
  \[ \Delta w_{kj} = \begin{cases} 
  \eta(x_j - w_{kj}) & \text{if } k \text{ is the winner} \\
  0 & \text{otherwise} 
\end{cases} \]

* The synaptic weight vector \( w_k = (w_{k1}, w_{k2}, \ldots, w_{kn}) \) is moved toward the input vector.

**Boltzmann Learning**

- Stochastic learning algorithm rooted in statistical mechanics.
- Recurrent network, binary neurons (on: ‘+1’, off: ‘-1’).
- Energy function \( E \):
  \[ E = -\frac{1}{2} \sum_j \sum_{k,k \neq j} w_{kj} x_k x_j \]
- Activation:
  - Choose a random neuron \( k \).
  - Flip state with a probability (given temperature \( T \))
  \[ P(x_k \rightarrow -x_k) = \frac{1}{1 + \exp(-\Delta E_k/T)} \]
  where \( \Delta E_k \) is the change in \( E \) due to the flip.

**Boltzmann Machine**

- Two types of neurons
  - **Visible** neurons: can be affected by the environment
  - **Hidden** neurons: isolated
- Two modes of operation
  - **Clamped**: visible neuron states are fixed by environmental input and held constant.
  - **Free-running**: all neurons are allowed to update their activity freely.
Boltzmann Machine: Learning and Operation

- Learning:
  - Correlation of activity during clamped condition $\rho_{kj}^{+}$
  - Correlation of activity during free-running condition $\rho_{kj}^{-}$
  - Weight update: $\Delta w_{kj} = \eta(\rho_{kj}^{+} - \rho_{kj}^{-})$, $j \neq k$.

- Train weights $w_{kj}$ with various clamping input patterns.
- After training is completed, present new clamping input pattern that is a partial input of one of the known vectors.
- Let it run clamped on the new input (subset of visible neurons), and eventually it will complete the pattern (pattern completion).

$$\text{Correl}(x, y) = \frac{\text{Cov}(x, y)}{\sigma_x \sigma_y}$$

Credit-Assignment Problem

- How to assign credit or blame for overall outcome to individual decisions made by the learning machine.
- In many cases, the outcomes depend on a sequence of actions.
  - Assignment of credit for outcomes of actions (temporal credit-assignment problem): When does a particular action deserve credit.
  - Assignment of credit for actions to internal decisions (structural credit-assignment problem): assign credit to internal structures of actions generated by the system.

Credit-assignment problem routinely arises in neural network learning. Which neuron, which connection to credit or blame?

Learning Paradigms

How neural networks relate to their environment

- credit assignment problem
- learning with a teacher
- learning without a teacher

Learning with a Teacher

- Also known as supervised learning
- Teacher has knowledge, represented as input–output examples. The environment is unknown to the nnet.
- Nnet tries to emulate the teacher gradually.
- Error-correction learning is one way to achieve this.
- Error surface, gradient, steepest descent, etc.
Learning without a Teacher

Two classes

- Reinforcement learning (RL)/Neurodynamic programming
- Unsupervised learning/Self-organization

Learning without a Teacher: Reinforcement Learning

- Learning input-output mapping through continued interaction with the environment.
- **Actor-critic**: circuit converts primary reinforcement signal into higher-quality, heuristic reinforcement signal (Barto, Sutton, ...).
- Goal is to optimize the cumulative cost of actions.
- In many cases, learning is under delayed reinforcement. Delayed RL is difficult since (1) teacher does not provide desired action at each step, and (2) must solve temporal credit-assignment problem.
- Relation to dynamic programming, in the context of optimal control theory (Bellman).

Learning without a Teacher: Unsupervised Learning

- Learn based on task-independent measure of the quality of representation.
- Internal representations for encoding features of the input space.
- Competitive learning rule needed, such as *winner-takes-all*.

Learning Tasks, Memory, and Adaptation

Learning tasks

- Pattern association
- Pattern recognition
- Function approximation
- Control
- Filtering/Beamforming

Memory and adaptation
Pattern Association

- **Associative memory**: brain-like distributed memory that learns association. Storage and retrieval (recall).
- Pattern association ($x_k$: **key pattern**, $y_k$: **memorized pattern**):
  $$x_k \rightarrow y_k, \, k = 1, 2, \ldots, q$$
  - **autoassociation** ($x_k = y_k$): given partial or corrupted version of stored pattern and retrieve the original.
  - **heteroassociation** ($x_k \neq y_k$): Learn arbitrary pattern pairs and retrieve them.
- Relevant issues: **storage capacity** vs. accuracy.

Function Approximation

- Nonlinear input-output mapping: $d = f(x)$ for an unknown $f$.
- Given a set of labeled examples $T = \{(x_i, d_i)\}_{i=1}^N$, estimate $F(\cdot)$ such that
  $$\|F(x) - f(x)\| < \epsilon, \text{ for all } x$$

Pattern Classification

- Mapping between input pattern and a prescribed number of classes (categories).
- Two general types:
  - Feature extraction (observation space to feature space: cf. dimensionality reduction), then classification (feature space to decision space).
  - Single step (observation space to decision space).

Function Approx: System Identification and Inverse System Modeling

- System identification: learn function of an unknown system.
  $$d = f(x)$$
- Inverse system modeling: learn inverse function:
  $$x = f^{-1}(d)$$
Control

- Control of a plant, a process or critical part of a system that is to be maintained in a controlled condition.

- Feedback controller: adjust plant input $u$ so that the output of the plant $y$ tracks the reference signal $d$. Learning is in the form of free-parameter adjustment in the controller.

Filtering, Smoothing, and Prediction

Extract information about a quantity of interest from a set of noisy data.

- Filtering: estimate quantity at time $n$, based on measurements up to time $n$.

- Smoothing: estimate quantity at time $n$, based on measurements up to time $n + \alpha$ ($\alpha > 0$).

- Prediction: estimate quantity at time $n + \alpha$, based on measurements up to time $n$ ($\alpha > 0$).

Blind Source Separation and Nonlinear Prediction

- Blind source separation: recover $u(n)$ from distorted signal $x(n)$ when mixing matrix $A$ is unknown.

$$x(n) = Au(n)$$

- Nonlinear prediction: given $x(n - T), x(n - 2T), ...$, estimate $x(n)$ ($\hat{x}(n)$ is the estimated value).

Linear Algebra Tip: Partitioned (or Block) Matrices

- When multiplying matrices or matrix and a vector, partitioning them and multiplying the corresponding partitions can be very convenient.

- Consider the $4 \times 5$ matrix above (let’s call it $X$). If you have another $5 \times 4$ matrix partitioned similarly into $E, F$, $G, H$ (let’s call it $Y$), then you can calculate the product as another block matrix:

$$XY = \begin{bmatrix} A, B \\ C, D \end{bmatrix} \begin{bmatrix} E, F \\ G, H \end{bmatrix} = \begin{bmatrix} AE + BG, AF + BH \\ CE + DG, CF + DH \end{bmatrix}$$

Memory

- **Memory**: relatively enduring neural alterations induced by an organism’s interaction with the environment.
- Memory needs to be accessible by the nervous system to influence behavior.
- Activity patterns need to be stored through a learning process.
- Types of memory: short-term and long-term memory.

Associative Memory (cont’d)

- Weight matrix:

\[ y_k = W(k)x_k, \text{ for } k = 1, 2, \ldots, q \]

\[ y_{ki} = \sum_{j=1}^{m} w_{ij}(k)x_{kj}, \text{ for } i = 1, 2, \ldots, m \]

\[ y_{ki} = [w_{i1}(k), w_{i2}(k), \ldots, w_{im}(k)] \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{km} \end{bmatrix}, \text{ for } i = 1, 2, \ldots, m \]

- Input (key vector) \( x_k = [x_{k1}, x_{k2}, \ldots, x_{km}]^T \).
- Output (memorized vector) \( y_k = [y_{k1}, y_{k2}, \ldots, y_{km}]^T \).
- Weights can be represented as a weight matrix (see next slide):

\[ y_k \approx Mx_k, \text{ for } k = 1, 2, \ldots, q \]

\[ y_{ki} = \sum_{j=1}^{m} w_{ij}(k)x_{kj}, \text{ for } m = 1, 2, \ldots, m \]

- With a single \( W(k) \), we can only represent one mapping \((x_k, y_k)\). For all pairs \((x_k, y_k)\), we need \( q \) such weight matrices.

\[ y_k = W(k)x_k, \text{ for } k = 1, 2, \ldots, q \]

- One strategy is to combine all \( W(k) \) into a single memory matrix \( M \) by simple summation:

\[ M = \sum_{k=1}^{q} W(k) \]

- Will such a simple strategy work? That is, can the following be possible with \( M \)?

\[ y_k \approx Mx_k, \text{ for } k = 1, 2, \ldots, q \]
**Associative Memory: Example – Storing Multiple Mappings**

With fixed set of key vectors \( \mathbf{x}_k \), an \( m \times m \) matrix can store \( m \) arbitrary output vectors \( \mathbf{y}_k \).

- Let \( \mathbf{x}_k = [0, 0, \ldots, 1, \ldots, 0]^T \) where only the \( k \)-th element is 1 and all the rest is 0.
- Construct a memory matrix \( \mathbf{M} \) with each column representing the arbitrary output vectors \( \mathbf{y}_k \) (use the block matrix technique):

\[
\mathbf{M} = \begin{bmatrix} \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m \end{bmatrix}
\]

- Then, \( \mathbf{y}_k = \mathbf{Mx}_k \), for all \( k = 1, 2, \ldots, m \).
- But, we want \( \mathbf{x}_k \) to be arbitrary too!

41

**Correlation Matrix Memory**

- With \( q \) pairs \( (\mathbf{x}_k, \mathbf{y}_k) \), we can construct a candidate memory matrix that stores all \( q \) mappings as:

\[
\hat{\mathbf{M}} = \sum_{k=1}^{q} \mathbf{y}_k \mathbf{x}_k^T = \mathbf{y}_1 \mathbf{x}_1^T + \mathbf{y}_2 \mathbf{x}_2^T + \ldots + \mathbf{y}_q \mathbf{x}_q^T,
\]

where \( \mathbf{y}_k \mathbf{x}_k^T \) represents the outer product of vectors that results in a matrix, i.e.,

\[
(\mathbf{y}_k \mathbf{x}_k^T)_{ij} = y_{kj} x_{ki}.
\]

- A more convenient notation is:

\[
\hat{\mathbf{M}} = \begin{bmatrix} \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_q \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_q \end{bmatrix} = \mathbf{Y} \mathbf{X}^T.
\]

This can be verified easily using partitioned matrices.

42

**Correlation Matrix Memory: Recall (cont’d)**

- Now, back to \( \hat{\mathbf{M}} \): under what condition will \( \hat{\mathbf{M}} \mathbf{x}_j \) give \( \mathbf{y}_j \) for all \( j \)? Let’s begin by assuming \( \mathbf{x}_k^T \mathbf{x}_k = 1 \) (key vectors are normalized).

- We can decompose \( \hat{\mathbf{M}} \mathbf{x}_j \) as follows:

\[
\hat{\mathbf{M}} \mathbf{x}_j = \sum_{k=1}^{q} \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_j = \mathbf{y}_j \mathbf{x}_j^T \mathbf{x}_j + \sum_{k=1, k \neq j}^{q} \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_j.
\]

- We know \( \mathbf{y}_j \mathbf{x}_j^T \mathbf{x}_j = \mathbf{y}_j \), so it now becomes:

\[
\hat{\mathbf{M}} \mathbf{x}_j = \mathbf{y}_j + \sum_{k=1, k \neq j}^{q} \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_j.
\]

- If all keys are orthogonal (perpendicular to each other), then for an arbitrary \( k \neq j \), \( \mathbf{x}_k^T \mathbf{x}_j = \|\mathbf{x}_k\|\|\mathbf{x}_j\| \cos(\theta_{kj}) = 1 \times 1 \times 0 = 0 \), so the noise term becomes 0, and hence \( \hat{\mathbf{M}} \mathbf{x}_j = \mathbf{y}_j + 0 = \mathbf{y}_j \). The example in page 41 is one such case, an extreme one!
Correlation Matrix Memory: Recall (cont’d)

- We can also ask how many items can be stored in \( \hat{M} \), i.e., its capacity.
- The capacity is closely related with the rank of the matrix \( \hat{M} \).
  The rank means the number of linearly independent column vectors (or row vectors) in the matrix.
- Linear independence means a linear combination of the vectors can be zero only when the coefficients are all zero:
  \[ c_1 x_1 + c_2 x_2 + \ldots + c_n x_n = 0 \]
  only when \( c_i = 0 \) for all \( i = 1, 2, \ldots, n \).
- The above and the examples in the previous pages are best understood by running simple calculations in Octave or Matlab. See the src/ directory for example scripts.

45

Statistical Nature of Learning

- Deviation between the target function \( f(x) \) and the neural network realization of the function \( F(x, w) \) can be expressed in statistical terms (note \( F(\cdot, \cdot) \) is parametrized by the weight \( w \)).
- Random input vectors \( X \in \{x_i\}_{i=1}^N \) and random output scalar values \( D \in \{d_i\}_{i=1}^N \)
- Suppose we have a training set \( T = \{(x_i, d_i)\}_{i=1}^N \). The problem is that the target values \( D \) in the training set may only be approximate \( (D \approx f(X)), i.e., D \neq f(X)) \).
- So, we end up with a regressive model:
  \[ D = f(X) + \epsilon, \]
  where \( f(\cdot) \) is deterministic and \( \epsilon \) is a random expectational error representing our ignorance.

47

Adaptation

- When the environment is stationary (the statistic characteristics do not change over time), supervised learning can be used to obtain a relatively stable set of parameters.
- If the environment is nonstationary, the parameters need to be adapted over time, on an on-going basis (continuous learning or learning-on-the-fly).
- If the signal is locally stationary (pseudostationary), then the parameters can repeatedly be retrained based on a small window of samples, assuming these are stationary: continual training with time-ordered samples.

46

Statistical Nature of Learning (cont’d)

- The error term \( \epsilon \) is typically assumed to have a zero mean:
  \[ E[\epsilon|x] = 0, (E[\cdot] \text{ is the expected value of a random variable.}) \]
- In this light, \( f(x) \) can be expressed in statistical terms:
  \[ f(x) = E[D|x], \text{ since from} \]
  \[ D = f(X) + \epsilon, \text{ we can get} \]
  \[ E[D|x] = E[f(X) + \epsilon] = f(x) + E[\epsilon|x] = f(x). \]
- A property that can be derived from the above is that the expectational error term is independent of the regressive function:
  \[ E[\epsilon f(X)] = 0. \text{ This will become useful in the following.} \]
Statistical Nature of Learning (cont’d)

- Neural network realization of the regressive model:
  \[ Y = F(X, w). \]
  We want to map the knowledge in the training data \( \mathcal{T} \) into the weights \( w \).

- We can now define the cost function:
  \[ \mathcal{E}(w) = \frac{1}{2} \sum_{i=1}^{N} (d_i - F(x_i, w))^2 \]
  which can be written equivalently as an average over the training set \( \mathcal{E}_T[\cdot] \):
  \[ \mathcal{E}(w) = \frac{1}{2} \mathcal{E}_T[(d_i - F(x, \mathcal{T}))^2] \]

Statistical Nature of Learning: Bias/Variance Dilemma

The cost function we derived
\[ \mathcal{E}_T[(f(x) - F(x, \mathcal{T}))^2] \]
can be rewritten, knowing \( f(x) = E[D|x] \):
\[
\begin{align*}
\mathcal{E}_T[(E[D|x] - F(x, \mathcal{T}))^2] &= \mathcal{E}_T[(E[D|x] - E[T[F(x, \mathcal{T})]] + E[T[F(x, \mathcal{T})] - F(x, \mathcal{T})]^2] \\
&= (\mathcal{E}_T[F(x, \mathcal{T})]^2 - E[D|x]^2) + \mathcal{E}_T[(F(x, \mathcal{T}) - E[T[F(x, \mathcal{T})]^2]].
\end{align*}
\]

The last step above is obtained using \( \mathcal{E}_T[E[D|x]^2] = E[D|x]^2 \),
\( \mathcal{E}_T[E[T[F(x, \mathcal{T})]^2] = E[T[F(x, \mathcal{T})]^2] \), and
\( \mathcal{E}_T[E[D|x]F(x, \mathcal{T})] = E[D|x]E[T[F(x, \mathcal{T})]]. \)

* Note: \( E[c] = c \) and \( E[cX] = cE[X] \) for constant \( c \) and random variable \( X \).

Bias/Variance Dilemma (cont’d)

- The bias indicates how much \( F(x, \mathcal{T}) \) differs from the true function \( f(x) \): approximation error

- The variance indicates the variance in \( F(x, \mathcal{T}) \) over the entire training set \( \mathcal{T} \): estimation error

- Typically, achieving smaller bias leads to higher variance, and smaller variance leads to higher bias.
Statistical Learning Theory

- Statistical learning theory addresses the fundamental issue of how to control the **generalization ability** of a neural network in mathematical terms.

- Certain quantities such as **sample size** and the **Vapnik-Chervonenkis dimension** (VC dimension) is closely related to the **bounds** on generalization error.

- The **probably approximately correct** (PAC) learning model is another framework to study such bounds. In this case, the confidence $\delta$ (probably) and tolerable error level $\epsilon$ (approximately correct) are important quantities. Given these, and other measures such as the VC dimension, we can calculate the **sample complexity** (how many samples are needed to achieve that level of correctness $\epsilon$ with that much confidence $\delta$).

Appendix on VC Dimension

- The concept of Shattering

- VC dimension

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 a function class $\mathcal{F}$ if and only if for every dichotomy of $S$ there exists some function in $\mathcal{F}$ consistent with this dichotomy.

Three Instances Shattered

Instance space $X$

Each closed contour indicates one dichotomy. What kind of classifier function can shatter the instances?
The Vapnik-Chervonenkis Dimension

Definition: The Vapnik-Chervonenkis dimension, \( VC(\mathcal{F}) \), of function class \( \mathcal{F} \) defined over sample space \( X \) is the size of the largest finite subset of \( X \) shattered by \( \mathcal{F} \). If arbitrarily large finite sets of \( X \) can be shattered by \( \mathcal{F} \), then \( VC(\mathcal{F}) \equiv \infty \).

Note that \( |\mathcal{F}| \) can be infinite, while \( VC(\mathcal{H}) \) finite!

VC Dim. of Linear Decision Surfaces

- When \( \mathcal{F} \) is a set of lines, and \( S \) a set of points, \( VC(\mathcal{F}) = 3 \).
- \((a)\) can be shattered, but \((b)\) cannot be. However, if at least one subset of size 3 can be shattered, that’s fine.
- Set of size 4 cannot be shattered, for any combination of points (think about an XOR-like situation).

Uses of VC Dimension

- Training error decreases monotonically as the VC dimension is increased.
- Confidence interval increases monotonically as the VC dimension is increased.
- Sample complexity (in PAC framework) increases as VC dimension increases.