Slide02

Haykin Chapter 2: Learning Processes

CPSC 636-600

Instructor: Yoonsuck Choe

Spring 2008

Introduction

- Property of primary significance in nnet: **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.*

Learning

Sequence of events in nnet learning:

- nnet is **stimulated** by the environment.
- nnet **undergoes changes** in its free parameters as a result of this stimulation.
- nnet **responds in a new way** to the environment because of the changes that have occurred in its internal structure.

A prescribed set of well-defined rules for the solution of the learning problem is called a **learning algorithm**.

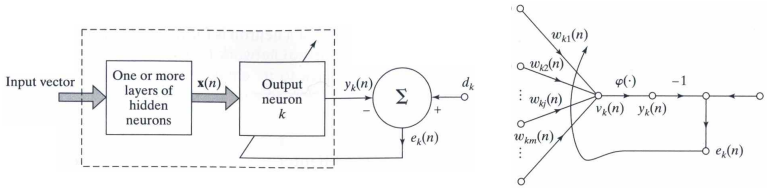
The manner in which a nnet relates to the environment dictates the **learning paradigm** that refers to a **model** of environment operated on by the nnet.

Overview

Organization of this chapter:

1. Five basic learning rules
   error correction, Hebbian, memory-based, copetetive, 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

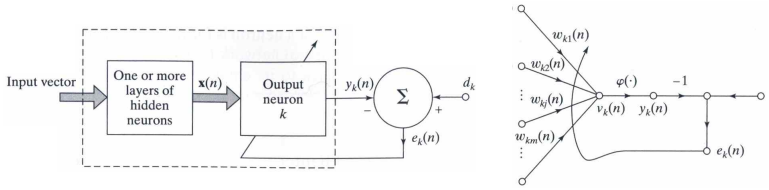


- Input $\mathbf{x}(n)$, output $y_k(n)$, and **desired response** or **target output** $d_k(n)$.
- **Error signal** $e_k(n) = d_k(n) - y_k(n)$
- $e_k(n)$ actuates a *control mechanism* that gradually adjust the *synaptic weights*, to miminize the **cost function (or index of performance)**:

$$\mathcal{E}(n) = \frac{1}{2} e_k^2(n)$$

- When synaptic weights reach a *steady state*, learning is stopped.

Error-Correction Learning: Delta Rule



- *Widrow-Hoff rule*, with **learning rate** η:

$$\Delta w_k j(n) = \eta e_k(n) x_j(n)$$

- With that, we can update the weights:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_k j(n)$$

- There is a sound theoretical reason for doing this, which we will discuss later.

Memory-Based Learning

- All (or most) past experiences are explicitly stored, as input-target pairs $\{\mathbf{x}_i, d_i)\}_{i=1}^N$.
- Two classes $\mathcal{C}_1, \mathcal{C}_2$.
- Given a new input $\mathbf{x}_{\text{test}}$, determine class based on **local neighborhood** of $\mathbf{x}_{\text{test}}$.
  - Criterion used for determining the neighborhood
  - Learning rule applied to the neighborhood of the input, within the set of training examples.

Memory-Based Learning: Nearest Neighbor

- A set of instances observed so far:

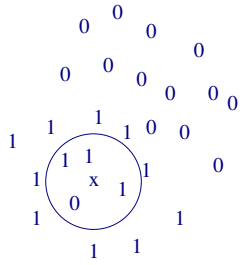
$$X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$$

- Nearest neighbor $\mathbf{x}'_N \in X$ of $\mathbf{x}_{\text{test}}$:

$$\min_i d(\mathbf{x}_i, \mathbf{x}_{\text{test}}) = d(\mathbf{x}_i, \mathbf{x}_{\text{test}})$$

  where $d(\cdot, \cdot)$ is the Euclidean distance.
- $\mathbf{x}_{\text{test}}$ is classified as the same class as $\mathbf{x}'_N$.
- Cover and Hart (1967): The bound on error is at max 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 classlfied patterns that lie nearest to the test vector $\mathbf{x}_{\text{test}}$, for some integer k.
- Assign $\mathbf{x}_{\text{test}}$ 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 $\mathbf{x}$ above will be classified as 1.

Hebbian Learning

- *Donald Hebb*'s postulate of learning appeared in his book *The Organization of Behavior* (1949).

  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.)

Hebbian Synapses

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

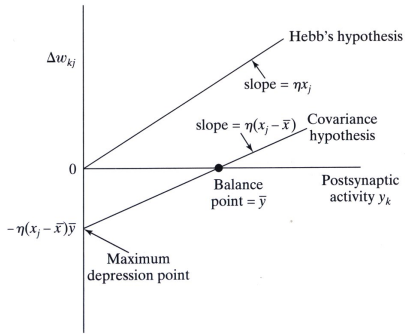
Strong evidence for Hebbian plasticity in the Hippocampus (brain region).

Classification of Synaptic Plasticity

Hebbian: time-dependent, highly local, heavily interactive.

Type	Positively correlated	Negatively correlated
Hebbian	Strengthen	Weaken
Anti-Hebbian	Weaken	Strengthen
Non-Hebbian	$\times$	$\times$

Mathematical Models of Synaptic Plasticity



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

Covariance Rule (Sejnowski 1977)

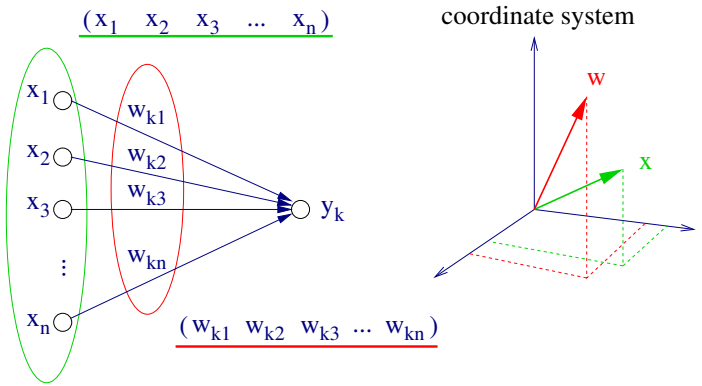
$$\Delta w_{kj} = \eta(x_j - \bar{x})(y_k - \bar{y})$$

- 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**.

Competetive Learning

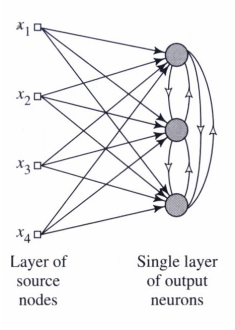
- Output neurons compete with each other for a chance to become active.
- Highly suited to discover statistically salient features (that may aid in classification).
- Three basic elements:
  - Same type of neurons with different weight sets, so that they **respond differently** to a given set of inputs.
  - 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.

Competetive Learning: Example



- Single layer, feedforward excitatory, and lateral inhibitory connections
- Winner selection

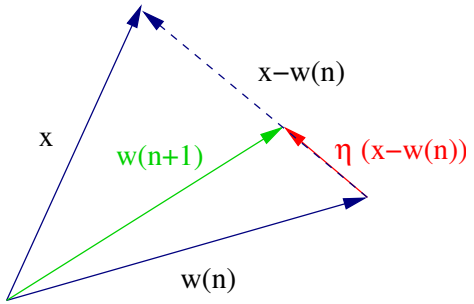
$$y_k = \begin{cases} 1 & \text{if } v_k > v_j \text{ for all } j, j \neq k \\ 0 & \text{otherwise} \end{cases}$$

- Limit: $\sum_j w_{kj} = 1$ for all k.
- Adaptation:

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_k j) & \text{if } k \text{ is the winner} \\ 0 & \text{otherwise} \end{cases}$$

* The synaptic **weight vector** $\mathbf{w}_k = (w_{k1}, w_{k2}, ..., w_{kn})$ is moved toward the input vector.

Competetive Learning



- Adaptation:

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_k j) & \text{if } k \text{ is the winner} \\ 0 & \text{otherwise} \end{cases}$$

Interpreting this as a vector, we get the above plot.

- Weight vectors converge toward local input clusters: clustering.

Boltzmann Learning

- Stochastic learning algorithm rooted in statistical mechanics.
- Recurrent network, binary neurons (on: '+1', off: '-1').
- Energy function E:

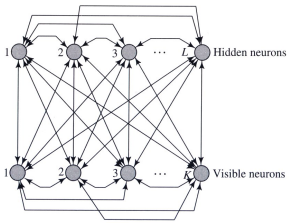
$$E = -\frac{1}{2} \sum_j \sum_{k, k \neq j} w_{kj} x_k x_j$$

- 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 ΔE_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 ρ_{kj}^{+}
  - Correlation of activity during free-running condition ρ_{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**).

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

Learning Paradigms

How neural networks relate to their environment

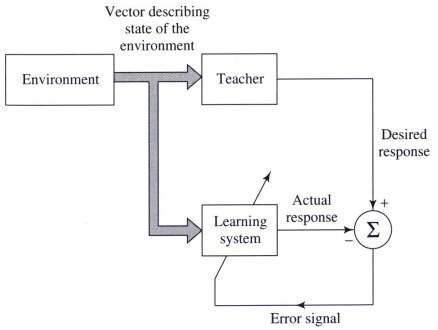
- credit assignment problem
- learning with a teacher
- learning without a teacher

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.

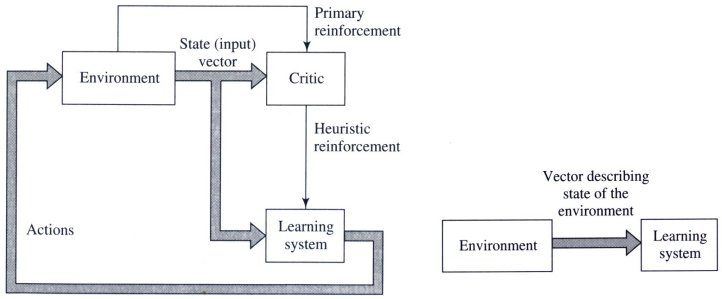
Credit-assignment problem routinely arises in neural network learning. Which neuron, which connection to credit or blame?

Learning 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.

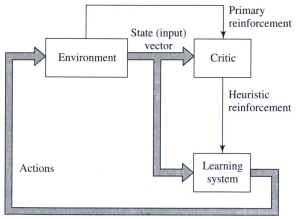
Learning without a Teacher



Two classes

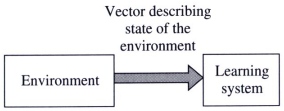
- Reinforcement learning (RL)/Neurodynamic programming
- Unsupervised learning/Self-organization

Learning without a Teacher: Reinforcement Learning



- Learning input-output mapping through continued interaction with the environment.
- **Actor-critic**: cricit 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.
- Competetive 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

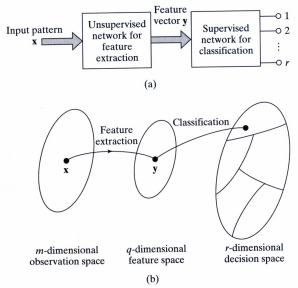
Pattern Association

- **Associtive memory**: brainlike distributed memory that learns association. **Storage** and retrieval (**recall**).
- Pattern association ($\mathbf{x}_k$: **key pattern**, $\mathbf{y}_k$: **memorized pattern**):

$$\mathbf{x}_k \rightarrow \mathbf{y}_k, k = 1, 2, ..., q$$

  - **autoassociation** ($\mathbf{x}_k = \mathbf{y}_k$): given partial or corrupted version of stored pattern and retrieve the original.
  - **heteroassociation** ($\mathbf{x}_k \neq \mathbf{y}_k$): Learn arbitrary pattern pairs and retrieve them.
- Relevant issues: **storage capacity** vs. accuracy.

Pattern Classification



- Mapping between input pattern and a prescrived 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 Approximation

- Nonlinear input-output mapping: $\mathbf{d} = \mathbf{f}(\mathbf{x})$ for an unknown $\mathbf{f}$.
- Given a set of labeled examples $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{d}_i)\}_{i=1}^N$, estimate $\mathbf{F}(\cdot)$ such that

$$\|\mathbf{F}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\| < \epsilon, \text{ for all } \mathbf{x}$$

Function Apprix: System Identification and Inverse System Modeling

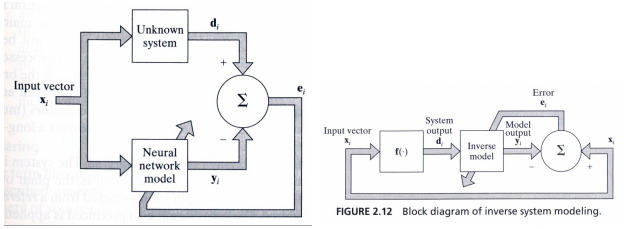


FIGURE 2.12 Block diagram of inverse system modeling.

- System identification: learn function of an unknown system.

$$\mathbf{d} = \mathbf{f}(\mathbf{x})$$

- Inverse system modeling: learn inverse function:

$$\mathbf{x} = \mathbf{f}^{-1}(\mathbf{d})$$

Control

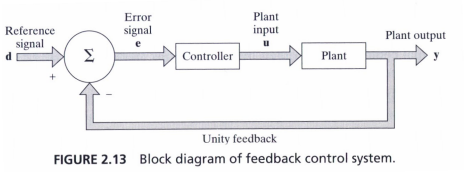


FIGURE 2.13 Block diagram of feedback control system.

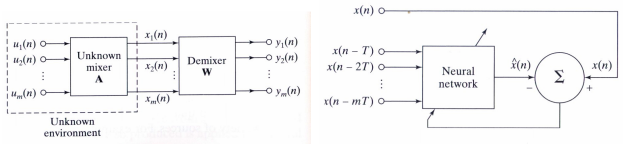
- Control of a *plant*, a process or critical part of a system that is to be maintained in a controlled condition.
- Feedback controller: adjust plant input $\mathbf{u}$ so that the output of the plant $\mathbf{y}$ tracks the reference signal $\mathbf{d}$. Learning is in the form of free-parameter adjustment 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 $\mathbf{u}(n)$ from distorted signal $\mathbf{x}(n)$ when mixing matrix $\mathbf{A}$ is unknown.

$$\mathbf{x}(n) = \mathbf{A}\mathbf{u}(n)$$

- 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

A 4 by 5 matrix can be considered as a 2 by 2 matrix with the matrices

$$\begin{bmatrix} 1&2&3&4&5 \\ 6&7&8&9&10 \\ 11&12&13&14&15 \\ 16&17&18&19&20 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$A = \begin{bmatrix} 1&2&3 \\ 6&7&8 \\ 11&12&13 \end{bmatrix}, B = \begin{bmatrix} 4&5 \\ 9&10 \\ 14&15 \end{bmatrix}, C = \begin{bmatrix} 16&17&18 \end{bmatrix}, D = \begin{bmatrix} 19&20 \end{bmatrix}$$

- When multiplying matrices or matrix and a vector, partitioning them and multiplying the corresponding partitions can be very convenient.
- Consider the 4×5 matrix above (let's call it $\mathbf{X}$). If you have another 5×4 matrix partitioned similarly into $\begin{bmatrix} \mathbf{E}, \mathbf{F} \\ \mathbf{G}, \mathbf{H} \end{bmatrix}$ (let's call it $\mathbf{Y}$), then you can calculate the product as another block matrix:

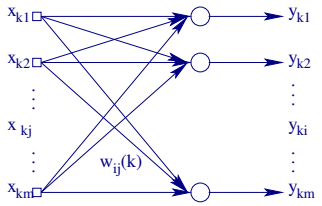
$$\mathbf{XY} = \begin{bmatrix} \mathbf{A}, \mathbf{B} \\ \mathbf{C}, \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{E}, \mathbf{F} \\ \mathbf{G}, \mathbf{H} \end{bmatrix} = \begin{bmatrix} \mathbf{AE} + \mathbf{BG}, \mathbf{AF} + \mathbf{BH} \\ \mathbf{CE} + \mathbf{CG}, \mathbf{CF} + \mathbf{CH}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

Example from `http://algebra.math.ust.hk/matrix_linear_trans/08_partition/lecture.shtml`.

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



q pattern pairs: $(\mathbf{x}_k, \mathbf{y}_k)$, for $k = 1, 2, ..., q$.

- Input (key vector) $\mathbf{x}_k = [x_{k1}, x_{k2}, ..., x_{km}]^T$.
- Output (memorized vector) $\mathbf{y}_k = [y_{k1}, y_{k2}, ..., y_{km}]^T$.
- Weights can be represented as a weight matrix:

$$\mathbf{y}_k = \mathbf{W}(k)\mathbf{x}_k, \text{ for } k = 1, 2, ..., q$$

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

Associative Memory (cont'd)

- Weight matrix:

$$\mathbf{y}_k = \mathbf{W}(k)\mathbf{x}_k, \text{ for } k = 1, 2, ..., q$$

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

$$y_{ki} = [w_{i1}(k), w_{i2}(k), ..., w_{im}(k)] \begin{bmatrix} x_{k1} \\ x_{k2} \\ : \\ x_{km} \end{bmatrix}, i = 1, 2, ..., m$$

$$\begin{bmatrix} y_{k1} \\ y_{k2} \\ : \\ y_{km} \end{bmatrix} = \begin{bmatrix} w_{11}(k), w_{12}(k), ..., w_{1m}(k) \\ w_{21}(k), w_{22}(k), ..., w_{2m}(k) \\ ... \\ w_{m1}(k), w_{m2}(k), ..., w_{mm}(k) \end{bmatrix} \begin{bmatrix} x_{k1} \\ x_{k2} \\ : \\ x_{km} \end{bmatrix}$$

Associative Memory (cont'd)

- With a single $\mathbf{W}(k)$, we can only represent one mapping ($\mathbf{x}_k$ to $\mathbf{y}_k$). For all pairs $(\mathbf{x}_k, \mathbf{y}_k)$ ($k = 1, 2, ..., q$), we need q such weight matrices.

$$\mathbf{y}_k = \mathbf{W}(k)\mathbf{x}_k, \text{ for } k = 1, 2, ..., q$$

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

$$\mathbf{M} = \sum_{k=1}^{q} \mathbf{W}(k)$$

- Will such a simple strategy work? That is, can the following be possible with $\mathbf{M}$?

$$\mathbf{y}_k \approx \mathbf{M}\mathbf{x}_k, \text{ for } k = 1, 2, ..., q$$

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 $\mathbf{x}_k = [0, 0, ...1, ...0]^T$ 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 $\mathbf{y}_k$:

$$\mathbf{M} = \begin{bmatrix} & & \\ \mathbf{y}_1, & \mathbf{y}_2, \ldots, & \mathbf{y}_m \\ & & \end{bmatrix}$$

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

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.,

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

- A more convenient notation is:

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

This can be verified easily using partitioned matrices.

Correlation Matrix Memory: Recall

- Will $\hat{\mathbf{M}}\mathbf{x}_k$ give $\mathbf{y}_k$?
- For convenience, let's say

$$\hat{\mathbf{M}} = \sum_{k=1}^{q} \mathbf{y}_k \mathbf{x}_k^T = \sum_{k=1}^{q} \mathbf{W}(k).$$

- First, consider $\mathbf{W}(k) = \mathbf{y}_k \mathbf{x}_k^T$ only.
  Check if $\mathbf{W}(k)\mathbf{x}_k = \mathbf{y}_k$:

$$\mathbf{W}(k)\mathbf{x}_k = \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_k = \mathbf{y}_k (\mathbf{x}_k^T \mathbf{x}_k) = c\mathbf{y}_k$$

  where $c = \mathbf{x}_k^T \mathbf{x}_k$, a scalar value (the length of vector $\mathbf{x}_k$ squared). If all $\mathbf{x}_k$s were normalized to have length 1, $\mathbf{W}(k)\mathbf{x}_k = \mathbf{y}_k$ will hold!

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 $\mathbf{x}_k^T \mathbf{x}_k \overset{=}{} 1$ (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:

$$\hat{\mathbf{M}}\mathbf{x}_j = \mathbf{y}_j + \underbrace{\sum_{k=1, k\neq j}^{q} \mathbf{y}_k \mathbf{x}_k^T \mathbf{x}_j}_{\text{Noise term}}.$$

- 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 (extreme) case!

Correlation Matrix Memory: Recall (cont'd)

- We can also ask how many items can be stored in $\hat{\mathbf{M}}$, i.e., its **capacity**.
- The capacity is closely related with the **rank** of the matrix $\hat{\mathbf{M}}$. 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:

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + ... + c_n\mathbf{x}_n = 0$$

  only when $c_i = 0$ for all $i = 1, 2, ..., n$.
- 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.

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**.

Statistical Nature of Learning

- Deviation between the target function $f(\mathbf{x})$ and the neural network relization of the function $F(\mathbf{x}, \mathbf{w})$ can be expressed in statistical terms (note $F(\cdot, \cdot)$ is parameterized by the weight $\mathbf{w}$).
- Random input vectors $\mathbf{X} \in \{\mathbf{x}_i\}_{i=1}^N$ and random output scalar values $D \in \{d_i\}_{i=1}^N$
- Suppose we have a training set $\mathcal{T} = \{(\mathbf{x}_i, d_i)\}_{i=1}^N$. The problem is that the target values D in the training set may only be approximate ($D \approx f(\mathbf{X})$, i.e., $D \neq f(\mathbf{X})$).
- So, we end up with a **regressive model**:

$$D = f(\mathbf{X}) + \epsilon,$$

  where $f(\cdot)$ is deterministic and ϵ is a random *expectational error* representing our ignorance.

Statistical Nature of Learning (cont'd)

- The error term ϵ is typically assumed to have a zero mean: $E[\epsilon|\mathbf{x}] = 0$. ($E[\cdot]$ is the expected value of a random variable.)
- In this light, $f(\mathbf{x})$ can be expressed in statistical terms: $f(\mathbf{x}) = E[D|\mathbf{x}]$, since from

$$D = f(\mathbf{X}) + \epsilon, \text{ we can get}$$

$$E[D|\mathbf{x}] = E[f(\mathbf{x}) + \epsilon] = f(\mathbf{x}) + E[\epsilon|\mathbf{x}] = f(\mathbf{x}).$$

- A property that can be derived from the above is that the expectational error term is independent of the regressive function: $E[\epsilon f(\mathbf{X})] = 0$. This will become useful in the following.

Statistical Nature of Learning (cont'd)

- Neural network realization of the regressive model:

$$\mathbf{Y} = F(\mathbf{X}, \mathbf{w}).$$

We want to map the *knowledge* in the training data $\mathcal{T}$ into the weights $\mathbf{w}$.

- We can now define the cost function:

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(d_i - F(\mathbf{x}_i, \mathbf{w}))^2$$

which can be written equivalently as an average over the training set $E_{\mathcal{T}}[\cdot]$:

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}E_{\mathcal{T}}\left[(d_i - F(\mathbf{x}, \mathcal{T}))^2\right]$$

Statistical Nature of Learning (cont'd)

$$d - F(\mathbf{x}, \mathcal{T}) = d - f(\mathbf{x}) + f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}) = \epsilon + (f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T})).$$

With that,

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}E_{\mathcal{T}}\left[(d_i - F(\mathbf{x}, \mathcal{T}))^2\right] \text{ becomes}$$

$$= \frac{1}{2}E_{\mathcal{T}}\left[(\epsilon + (f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}))^2\right]$$

$$= \frac{1}{2}E_{\mathcal{T}}\left[\epsilon^2 + 2\epsilon(f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T})) + (f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}))^2\right]$$

$$= \underbrace{\frac{1}{2}E_{\mathcal{T}}[\epsilon^2]}_{\text{Intrinsic error}} + \underbrace{E_{\mathcal{T}}[\epsilon(f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}))]}_{\text{This reduces to 0}} + \underbrace{\frac{1}{2}E_{\mathcal{T}}[(f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}))^2]}_{\text{We're interested in this!}}$$

Statistical Nature of Learning: Bias/Variance Dillema

The cost function we derived

$$E_{\mathcal{T}}[(f(\mathbf{x}) - F(\mathbf{x}, \mathcal{T}))^2]$$

can be rewritten, knowing $f(\mathbf{x}) = E[D|\mathbf{x}]$:

$$E_{\mathcal{T}}[(E[D|\mathbf{x}] - F(\mathbf{x}, \mathcal{T}))^2]$$
$$= E_{\mathcal{T}}[(E[D|\mathbf{x}] - E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})] + E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})] - F(\mathbf{x}, \mathcal{T}))^2]$$
$$= \underbrace{(E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})] - E[D|\mathbf{x}])^2}_{Bias} + \underbrace{E_{\mathcal{T}}[(F(\mathbf{x}, \mathcal{T}) - E_{\mathcal{T}}[F(\mathbf{x}, \mathcal{T})])^2]}_{Variance}.$$

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

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

Bias/Variance Dillema (cont'd)

- The bias indicates how much $F(\mathbf{x}, \mathcal{T})$ differs from the true function $f(\mathbf{x})$: *approximation error*
- The variance indicates the variance in $F(\mathbf{x}, \mathcal{T})$ 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-Chevonenkis 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 the confidence δ (probably) and tolerable error level ϵ (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 ϵ with that much confidence δ).

Appendix on VC Dimension

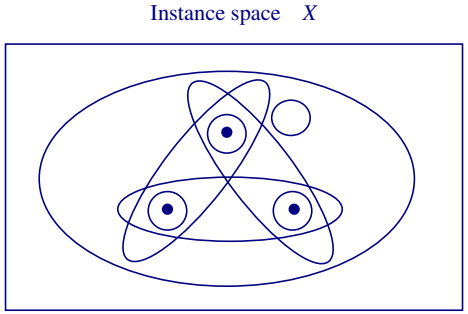
- The concept of Shattering
- VC dimension

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



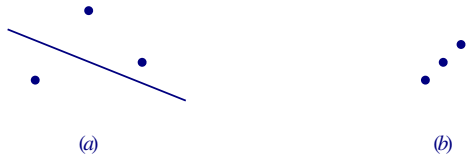
Each closed contour indicates one dichotomy. What kind of classifier function can shatter the instances?

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 $VC(H)$ 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 monotinically as the VC dimension is increased.
- Confidence interval increases monotinically as the VC dimension is increased.
- Sample complexity (in PAC framework) increases as VC dimension increases.