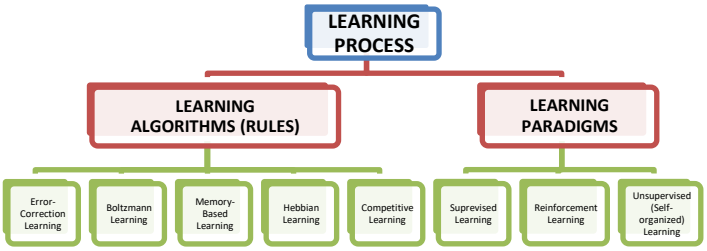
Learning Processes

Source: Haykin, 2009

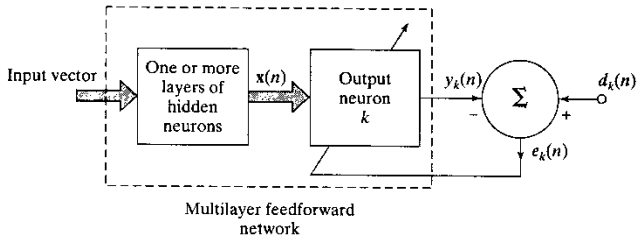
- One can define learning in the context of neural networks as follows [Haykin, 1999]:

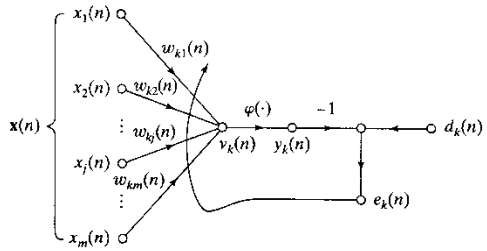
  "Learning is a process by which the free parameters (weights) of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place."
- A prescribed set of well-defined rules for the solution of a learning problem is called a ***learning algorithm***.
- Learning algorithms differ from each other in the way in which the adjustment of a synaptic weight of a neuron is formulated.
- Another factor to be considered is the manner in which a neural network relates to its environment. In this latter context, one can speak of a ***learning paradigm*** that refers to a *model* of the environment in which the neural network operates.



Learning Algorithms

Error-Correction Learning:





- Consider the case of a neuron *k* constituting one of the computational nodes in the output layer of a feedforward neural network
- Let us suppose that $\mathbf{x}(n)$ denotes the signal vector (produced by one or more layers of hidden neurons, which are themselves driven by an input vector) driving neuron *k*
- The argument *n* denotes discrete time, or more precisely, the time step of an iterative learning process
- Let $y_k(n)$ denote the output signal of neuron *k*
- If $d_k(n)$ denotes the desired response (or target output) for the given input, then, the error signal, denoted by $e_k(n)$ is given by:

$$e_k(n) = d_k(n) - y_k(n)$$

- The objective is to minimize a cost function, *J*, defined in terms of $e_k(n)$, such that the actual response of each output neuron in the network approaches the target response for that neuron in some statistical sense
- One of the most popular cost functions is:

$$J = E\left[\frac{1}{2}\sum_k e_k^2(n)\right]$$

- Method of gradient descent is one possible approach to minimizing *J*
- The difficulty in minimizing *J* is that it requires the knowledge of the statistical properties of the underlying process
- There are several alternatives; One alternative is to use the *instantaneous value* of the sum of squared error as the criterion of interest

$$\varepsilon(n) = \frac{1}{2}\sum_k e_k^2(n)$$

- Minimization of the cost function $\varepsilon(n)$ leads to a learning rule commonly referred to as the *delta rule* or *Widrow-Hoff rule*
- Let $w_{kj}(n)$ denote the value of the synaptic weight w_{kj} of neuron *k* excited by element $x_j(n)$ of the signal vector $\mathbf{x}(n)$ at time step *n*
- According to the delta rule, the adjustment $\Delta w_{kj}(n)$ applied to the synaptic weight w_{kj} at time step *n* is defined by:

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n)$$

where η denotes learning rate

- Note that the rule presumes that the error signal is directly measurable (neuron *k* is visible to the outside world)
- The updated weight of w_{kj} is given by:

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

- Learning rate η has to be selected carefully to ensure stability or convergence of the learning process

Notes:

- The cost function surface as a function of the network synaptic weights can be convex (strictly convex or pseudo convex) with a unique optimal solution or concave with numerous local optimal solutions
- Delta rule is essentially a gradient-descent method; Proof?
- All gradient-descent methods lead to local optimal solutions
- Gradient-descent techniques primarily update parameters in the following manner:

$$w_{kj}(n+1) = w_{kj}(n) - \eta\frac{\partial \varepsilon(n)}{\partial w_{kj}(n)}$$

Memory-Based Learning:

- In memory-based learning, all (or most) of the past experiences are explicitly stored in a large memory of correctly classified (or learnt) input-output examples: $\{(\mathbf{x}_i, d_i)\}_{i=1}^N$, where $\mathbf{x}_i$ denotes an input vector and d_i denotes the corresponding desired response
- When classification of a test vector $\mathbf{x}_{\text{test}}$ (not seen before) is required, the algorithm responds by retrieving and analyzing the training data in the "local neighborhood" of $\mathbf{x}_{\text{test}}$

- Memory-based learning can be thought of as a local-approximation learning while error-correction learning can be thought of as a global-approximation learning
- All memory-based learning algorithms involve two essential ingredients:
  1. Criterion used for defining the local neighborhood of the test vector $\mathbf{x}_{test}$
  2. Learning rule applied to the training examples in the local neighborhood of $\mathbf{x}_{test}$

Hebbian Learning:

- Hebb's postulate of learning is the oldest and most famous of all learning rules [Hebb, 1949]:

  "When an axon on a 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."
- One can expand and rephrase Hebb's postulate as a two-part rule [Stent, 1973]:
  1. If two neurons on either side of a synapse are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.
  2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.
- Key properties: Time-depended mechanism, Local mechanism, Interactive mechanism, and Conjunctional or correlational mechanism.
- Mathematical models of Hebbian modifications:
  - Consider a synaptic weight w_{kj} of neuron *k* with presynaptic and postsynaptic signals denoted by x_j and y_k, respectively.
  - The adjustment applied to the synaptic weight at time step *n* can be expressed in the general form:

    $\Delta w_{kj}(n) = \eta F(y_k(n), x_j(n))$

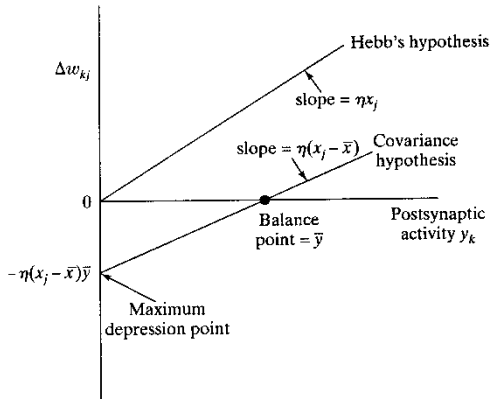
  Activity Product Rule

  $\Delta w_{kj}(n) = \eta y_k(n) x_j(n)$

  Repeated application can lead to exponential growth of w_{kj} (saturation)

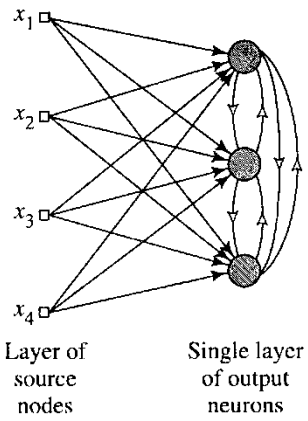
  Covariance Hypothesis

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

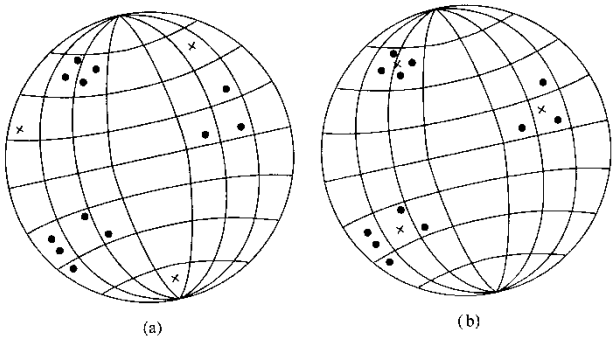


Competitive Learning:

- In competitive learning, the output neurons of a network compete among themselves.
- Unlike Hebbian learning networks where many output neurons can be simultaneously active, in competitive learning, only a single output neuron is active at any one time.
- Three basic elements [Rumelhart and Zipser, 1985]:
  1. Neurons are identical except for some randomly distributed synaptic weights.
  2. A limit imposed on the strength of each neuron.
  3. A mechanism that permits the neurons to compete for the right to win.
- The neurons of the network learn to specialize on ensembles of similar patterns, becoming *feature detectors* for different classes of input patterns.
- For a neuron *k* to be the winning neuron, its induced field v_k for a specified input pattern **x** must be largest among all the neurons in the network.



A simple competitive learning network.



Geometric interpretation of the competitive learning process. (a) Initial state. (b) Final state.

- The output signal y_k of the winning neuron is set equal to one; the output signals of all neurons that loose the competition are set equal to zero:

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

  where the induced local field v_k represents the combined action of all the feedforward (excitatory) and lateral (inhibitory) inputs to the neuron.
- Normally, each neuron is allocated a fixed amount of synaptic weight:

  $\sum_j w_{kj} = 1$ for all *k*
- According to the standard competitive learning rule:

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) & \text{if neuron } k \text{ wins the competition} \\ 0 & \text{if neuron } k \text{ loses the competition} \end{cases}$$

Boltzmann Learning:

- Is a stochastic learning algorithm derived from ideas rooted in statistical mechanics/thermodynamics.
- In a Boltzmann network (or machine) the neurons constitute a recurrent structure and operate in a binary manner ("on" state denoted by +1 and "off" state by -1).
- The network is characterized by an *energy function*, *E*, the value of which is determined by the particular states occupied by the individual neurons:

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

- The network operates by choosing a neuron at random—for example neuron *k*—at some step of the learning process, then flipping the state of the neuron *k* from state x_k to state $-x_k$ at some pseudotemperature *T* with probability:

$$P(x_k \rightarrow -x_k) = \frac{1}{1+\exp\left(-\frac{\Delta E_k}{T}\right)}$$

where ΔE_k is the *energy change* resulting from such a flip.

- If the rule is applied repeatedly, the machine will reach *thermal equilibrium*.
- The neurons of a Boltzmann network partition into two functional groups: *visible* and *hidden*. The visible neurons provide an interface between the network and the environment in which it operates, whereas the hidden neurons always operate freely.
- There are two modes of operation:
  - *Clamped condition*, in which the visible neurons are all clamped onto specific states determined by the environment.
  - *Free-running condition*, in which all the neurons are allowed to operate freely.
- Two distinct implementations of interest include:
  1. Solving a *learning problem* using Boltzmann learning rule: If ρ_{kj}^{+} and ρ_{kj}^{-} denote *correlations* (averaged over all possible equilibrium states of the network) between the states of neurons *j* and *k*, with the network in clamped and free-running conditions, respectively, then the change applied to w_{kj} is defined by [Hinton and Sejnowski, 1986]:

     $$\Delta w_{kj} = \eta(\rho_{kj}^{+} - \rho_{kj}^{-}), \quad j \neq k$$

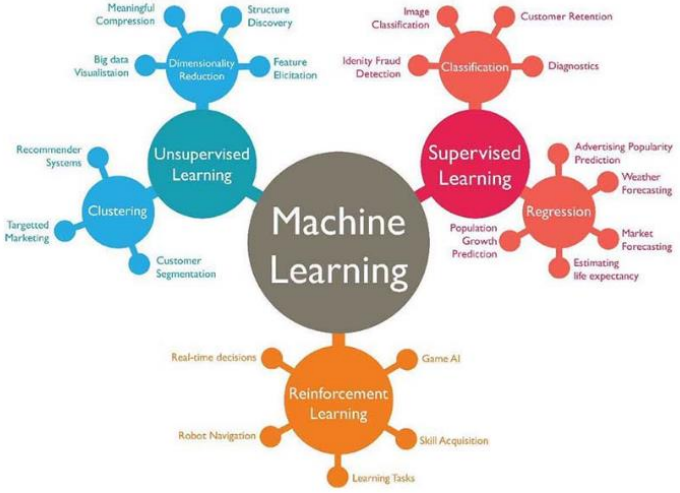
  2. Solving an *optimization problem* by holding the synaptic weights constant. The challenge lies in identifying the synaptic weights such that the overall network, when operated to reach thermal equilibrium, will solve an optimization problem of interest. The states of the neurons in the network at the end represent the solution to the optimization problem.

Credit-Assignment Problem:

- Is the problem of assigning *credit* or *blame* for overall outcomes to each of the internal decisions made by a learning machine.
- In many cases, this problem can be decomposed into two subproblems [Sutton, 1984]:
  1. The assignment of credit for outcomes to actions. This is called the *temporal credit-assignment problem* in that it involves the instants of time *when* the actions that deserve credit were actually taken.
  2. The assignment of credit for actions to internal decisions. This is called the *structural credit-assignment problem* in that it involves assigning credit to the *internal structures* of actions generated by the system.

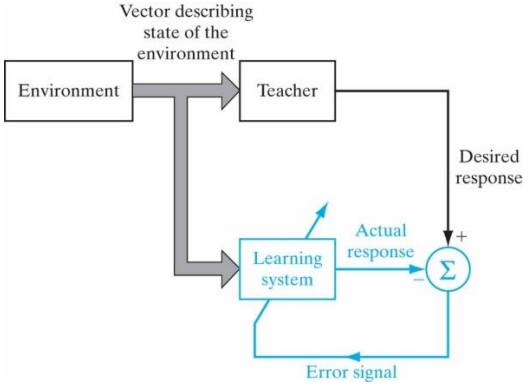
     For example, how do we adapt synaptic weights of neurons in a feedforward network that exist in the hidden layers and not the output layer? Error-correction learning is not directly applicable for it assumes that the neuron is exposed to the outside world (desired response is known).
- The combined temporal and structural credit-assignment problem faces any distributed learning machine that attempts to improve its performance in situations involving temporally extended behavior [Williams, 1988].

Learning Paradigms

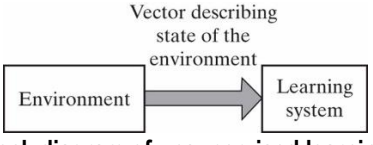


Learning With A Teacher–Supervised Learning:

- Teacher has knowledge of the environment, which may be represented by a set of input-output examples.
- The aim is to make the neural network emulate the teacher.
- When presented with a vector representing the state of the environment, both the neural network as well as the teacher provide responses; any difference in the responses, i.e. the error signal, is used to adjust the synaptic weights of the network.
- When this process is repeated several times, the network adapts to emulate the teacher; the emulation is presumed to be optimum in some statistical sense.



Block diagram of learning with a teacher.



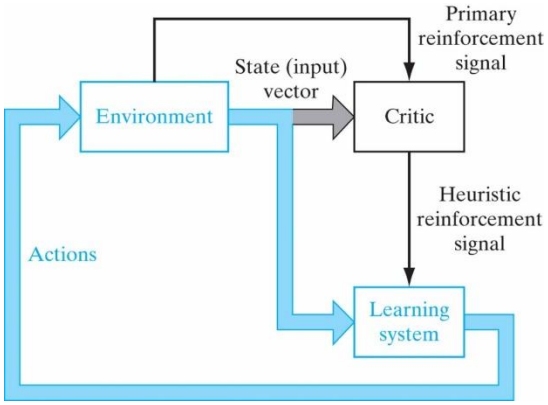
Block diagram of unsupervised learning.

Learning Without A Teacher–Unsupervised Learning:

- In *unsupervised* or *self-organized learning*, there is no external teacher to oversee the learning process.
- Provision is made for a *task-independent measure* of the quality of representation that the network is required to learn, and the synaptic weights are optimized with respect to that measure.
  For example, if the network's task is to group input vectors into distinct classes, the network tunes itself to statistical regularities of the input data, developing internal representations for encoding features of the input, and thereby create new classes automatically.

Learning Without A Teacher–Reinforcement Learning:

- In *reinforcement learning*, learning is performed by the network through continued interaction with the environment.
- One popular form of reinforcement learning involves utilization of a *critic* that converts a *primary reinforcement signal* received from the environment into a higher quality reinforcement signal called the *heuristic reinforcement signal*.
- The reinforcement signal is normally delayed.
- The goal of learning is to minimize a *cost-to-go function* (normally scalar), defined as the expectation of the cumulative cost of *actions* taken over a sequence of steps instead of simply the immediate cost.
  For example, it may turn out that certain actions taken earlier in a sequence of time steps are in fact the best determinants of overall system behavior. The function of the learning machine is to *discover* these actions and to feed them back to the environment.
- Reinforcement learning is difficult to perform for two reasons:
  – There is no teacher present to provide a desired response at each state of the learning process.
  – The delay incurred in the generation of the primary reinforcement signal implies that the learning machine must solve a *temporal credit assignment problem*.
- Reinforcement learning is closely related to *dynamic programming theory*, a theory that provides the mathematical formalism for sequential decision making.



Block diagram of one form of reinforcement learning.

LEARNING TASKS

- The choice of a particular learning algorithm is influenced by the learning task that a neural network is required to perform.
- Examples of learning tasks that are often attempted by neural networks:
  - Pattern Association
  - Pattern Recognition
  - Function Approximation
  - Control
  - Filtering

Pattern Association

- An associative memory is a brainlike distributed memory that learns by *association*.
- Association takes one of two forms:
  1. ***Autoassociation***
     - Here, a network is required to store a set of patterns by repeatedly presenting them to the network. The network is subsequently presented a partial description or distorted version of an original pattern stored in it, and the task is to *retrieve* (recall) that particular pattern.
     - Autoassociation normally involves the use of unsupervised learning.
     - Input and output spaces of the network have the same dimensionality.
  2. ***Heteroassociation***
     - Heteroassociation differs from autoassociation in that an arbitrary set of input patterns is *paired* with another arbitrary set of output patterns.
     - Normally, the type of learning involved in heteroassociation is supervised.
- There are two phases involved in the operation of an associative memory:
  - *Storage phase*, which refers to the training of the network.
  - *Recall phase*, which involves the retrieval of a memorized pattern.
- The challenge is to make the storage capacity as large as possible and yet insist that a large fraction of the memorized patterns be recalled correctly.

Input-output relation of pattern associator.

Pattern Recognition

- Formally defined as the process whereby a received pattern/signal is assigned to one of a prescribed number of classes (categories).
- Pattern recognition performed by a network is statistical in nature, with the patterns being represented by points in a multidimensional decision space.
- The decision space is divided into regions, each one of which is associated with a class.
- The decision boundaries are determined by the learning process.
- The learning process can be supervised (as is the case with feedforward networks) or unsupervised (as is the case with Adaptive Resonance Theory networks).

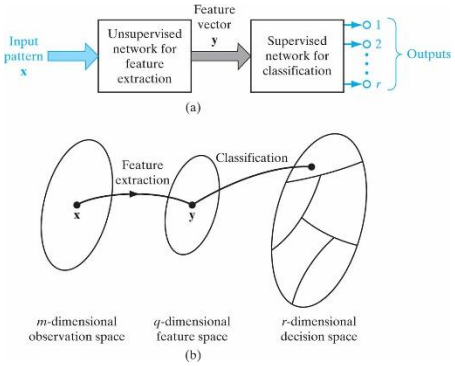
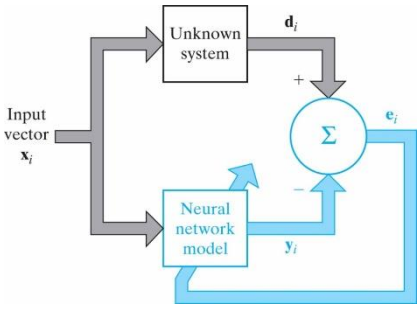


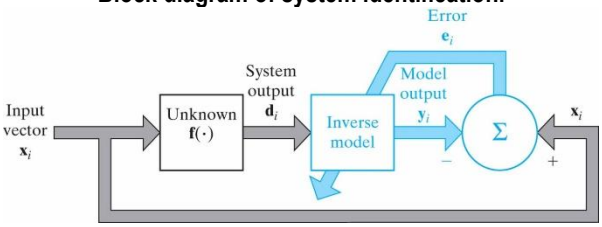
Illustration of a classical approach to pattern classification.

Function Approximation

- The task here is to train the network to learn (approximate) an unknown input output mapping (linear/nonlinear).
- The true mapping is normally unknown, and to make up for this lack of knowledge, one is given a set of labeled examples.
- The problem is a perfect candidate for supervised learning paradigm.



Block diagram of system identification.



Block diagram of inverse system modeling.

STATISTICAL NATURE OF THE LEARNING PROCESS / FINITE SAMPLE-SIZE CONSIDERATIONS (CONTEXT: REGRESSION)

- The focus here is not on the "evolution" of the weight vector **w** but on the "final deviation" between a "target/true" function $f(\mathbf{x})$ and the "learnt" function $F(\mathbf{x},\mathbf{w})$, where deviation is normally expressed in statistical terms.
- Let us suppose that we have *N* realizations of the random vector **X** denoted by $\{\mathbf{x}_i\}_{i=1}^N$ and a corresponding set of realizations of the dependent random scalar *D* denoted by $\{d_i\}_{i=1}^N$, constituting the training sample denoted by:
  $\Im = \{(\mathbf{x}_i, d_i)\}_{i=1}^N$.

- **"True" but Unknown Model:**
  Let us suppose that the true model generating the observation data follows the following relation:
  $D = f(\mathbf{X}) + \varepsilon$
  where ε denotes random error that represents our ignorance about the dependence of *D* on **X**.
  We also assume two properties:
  1. The mean value of the error ε, given any realization **x**, is zero; that is $E(\varepsilon \mid \mathbf{x}) = 0$ or $f(\mathbf{x}) = E(D \mid \mathbf{X} = \mathbf{x})$
     where *E* denotes the statistical expectation operator.
  2. The error ε is uncorrelated with the regression function $f(\mathbf{X})$; that is $E[\varepsilon f(\mathbf{X})] = \mathbf{0}$.

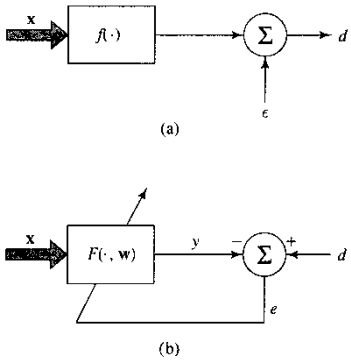
- **Model "Learnt" by Neural Network:**
  Given the training data $\Im$, let us suppose that the vector **w** is obtained by minimizing the following:

$$\xi(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(d_i - F(\mathbf{x}_i,\mathbf{w}))^2 \quad \text{or}$$

$$\xi(\mathbf{w}) = \frac{1}{2}E_{\Im}[(d - F(\mathbf{x},\mathbf{w}))^2]$$

where:
$E_{\Im}$ denotes the *average operator* taken over the entire training sample $\Im$.

Note: The variables that come under $E_{\Im}$ are denoted by **x** and *d*.
In contrast, the statistical expectation operator *E* acts on the whole ensemble of random variables **X** and *D*, which includes $\Im$ as a subset.



Models. (a) Regressive model.
(b) Neural network model

- One can rearrange the terms of the neural network model cost function, and lead to the following:

$$\begin{aligned}
\xi(\mathbf{w}) &= \frac{1}{2}E_{\Im}[(d - F(\mathbf{x},\mathbf{w}))^2] \\
&= \frac{1}{2}E_{\Im}\left[\left\{(d - f(\mathbf{x})) + (f(\mathbf{x}) - F(\mathbf{x},\mathbf{w}))\right\}^2\right] \\
&= \frac{1}{2}E_{\Im}\left[\left\{\varepsilon + (f(\mathbf{x}) - F(\mathbf{x},\mathbf{w}))\right\}^2\right] \\
&= \frac{1}{2}E_{\Im}\left[\varepsilon^2\right] + \frac{1}{2}E_{\Im}\left[(f(\mathbf{x}) - F(\mathbf{x},\mathbf{w}))^2\right] + \\
&\quad E_{\Im}\left[\varepsilon(f(\mathbf{x}) - F(\mathbf{x},\mathbf{w}))\right] \\
&= \underbrace{\frac{1}{2}E_{\Im}\left[\varepsilon^2\right]}_{\text{Intrinsic Error}} + \underbrace{\frac{1}{2}E_{\Im}\left[(f(\mathbf{x}) - F(\mathbf{x},\mathbf{w}))^2\right]}_{\text{Natural Measure of Effectivness}}
\end{aligned}$$

Bias/Variance Dilemma

- We can redefine the squared distance between $f(\mathbf{x})$ and $F(\mathbf{x},\mathbf{w})$ as follows:

$$\begin{aligned}
L_{av}(f(\mathbf{x}) - F(\mathbf{x},\mathbf{w})) &= E_{\Im}\left[(f(\mathbf{x}) - F(\mathbf{x},\mathbf{w}))^2\right] \\
&= E_{\Im}\left[\left(E[D \mid \mathbf{X} = \mathbf{x}] - F(\mathbf{x},\mathbf{w})\right)^2\right] \\
&= E_{\Im}\left[\left\{\left(E[D \mid \mathbf{X} = \mathbf{x}] - E_{\Im}[F(\mathbf{x},\mathbf{w})]\right) + \left(E_{\Im}[F(\mathbf{x},\mathbf{w})] - F(\mathbf{x},\mathbf{w})\right)\right\}^2\right] \\
&= E_{\Im}\left[\left(E[D \mid \mathbf{X} = \mathbf{x}] - E_{\Im}[F(\mathbf{x},\mathbf{w})]\right)^2\right] + E_{\Im}\left[\left(E_{\Im}[F(\mathbf{x},\mathbf{w})] - F(\mathbf{x},\mathbf{w})\right)^2\right] \\
&\quad + 2E_{\Im}\left[\left(E[D \mid \mathbf{X} = \mathbf{x}] - E_{\Im}[F(\mathbf{x},\mathbf{w})]\right)\left(E_{\Im}[F(\mathbf{x},\mathbf{w})] - F(\mathbf{x},\mathbf{w})\right)\right]
\end{aligned}$$

$$= \underbrace{\left(E[D\,|\,\mathbf{X}=\mathbf{x}]-E_{\mathfrak{I}}[F(\mathbf{x},\mathbf{w})]\right)^2}_{Bias^2} + \underbrace{E_{\mathfrak{I}}\left[\left(E_{\mathfrak{I}}[F(\mathbf{x},\mathbf{w})]-F(\mathbf{x},\mathbf{w})\right)^2\right]}_{Variance}$$

$$L_{av}\left(f(\mathbf{x})-F(\mathbf{x},\mathbf{w})\right) = B^2(\mathbf{w})+V(\mathbf{w})$$

where:

$$B(\mathbf{w}) = E[D\,|\,\mathbf{X}=\mathbf{x}]-E_{\mathfrak{I}}[F(\mathbf{x},\mathbf{w})] \leftarrow \textit{Approximation Error}$$

$$V(\mathbf{w}) = E_{\mathfrak{I}}\left[\left(E_{\mathfrak{I}}[F(\mathbf{x},\mathbf{w})]-F(\mathbf{x},\mathbf{w})\right)^2\right] \leftarrow \textit{Estimation Error}$$

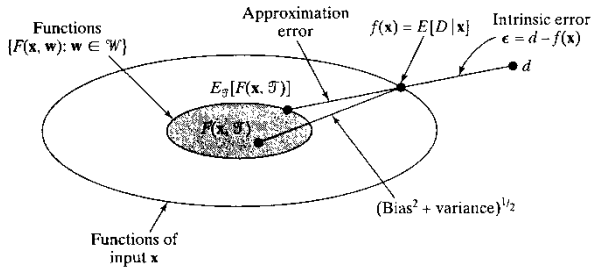
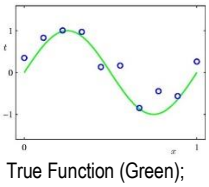


Illustration of various sources of error in solving the learning problem.

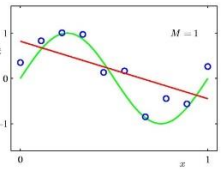
- To achieve good overall performance, both the bias and the variance of the approximating function should be small.
- Unfortunately, we find that in a neural network that learns by example and does so with a training sample of fixed size, the price for achieving a small bias is large variance.
  - *Larger neural networks (models with more parameters) produce smaller bias but larger variance*
  - *Smaller neural networks (models with fewer parameters) produce larger bias but smaller variance*
- Only when the training sample becomes infinitely large can we hope to eliminate both bias and variance at the same time.

Bias/Variance Dilemma – Polynomial Curve Fitting Example (Source: C. Bishop)

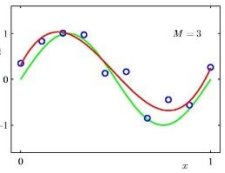
$$y(x,\mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$



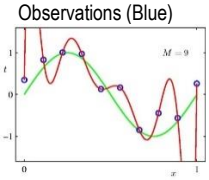
True Function (Green); Observations (Blue)



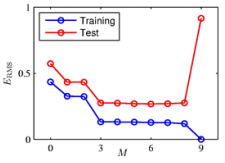
1st Order Polynomial



3rd Order Polynomial



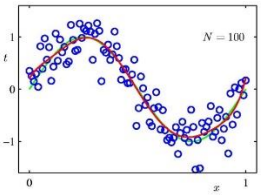
9th Order Polynomial



Over-Fitting

	M = 0	M = 1	M = 3	M = 9
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

Polynomial Coefficients

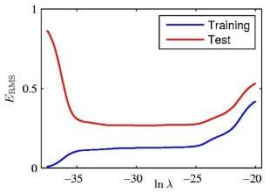


9th Order Polynomial (Data Set Size: *N*=100)

Regularization to Deal with Bias/Variance Dilemma: Penalize Large Coefficient Values

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w})-t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

"Ridge" Regression
L_1 Norm penalty leads to LASSO



9th Order Polynomial (Data Set Size: *N*=100)

	ln λ = −∞	ln λ = −18	ln λ = 0
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

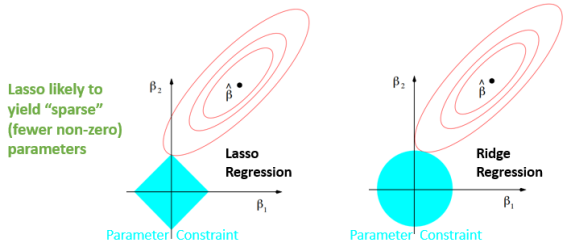
Polynomial Coefficients under Regularization

Price: Loss Function is Distorted!

- **Linear Regression: Regularization Options & Tradeoffs**
- Three Norms: $l_0,\ l_1, l_2$

$$\|\beta\|_0 = \sum_{j=1}^{p} 1\{\beta_j \neq 0\}, \quad \|\beta\|_1 = \sum_{j=1}^{p}|\beta_j|, \quad \|\beta\|_2 = \left(\sum_{j=1}^{p}\beta_j^2\right)^{1/2}.$$

- In constrained form:
  - **Best Subset Selection:** $\min_{\beta\in\mathbb{R}^p} \|y-X\beta\|_2^2 \text{ subject to } \|\beta\|_0 \le k$ (1)
  - **Lasso Regression:** $\min_{\beta\in\mathbb{R}^p} \|y-X\beta\|_2^2 \text{ subject to } \|\beta\|_1 \le t$ (2)
  - **Ridge Regression:** $\min_{\beta\in\mathbb{R}^p} \|y-X\beta\|_2^2 \text{ subject to } \|\beta\|_2^2 \le t$ (3)



- Equivalence, Sparsity, and Convexity:
  - **Best Subset Selection:** $\min_{\beta\in\mathbb{R}^p} \|y-X\beta\|_2^2 \text{ subject to } \|\beta\|_0 \le k$ (1) $\quad \min_{\beta\in\mathbb{R}^p} \frac{1}{2}\|y-X\beta\|_2^2+\lambda\|\beta\|_0$ (4)
  - **Lasso Regression: Convex** $\min_{\beta\in\mathbb{R}^p} \|y-X\beta\|_2^2 \text{ subject to } \|\beta\|_1 \le t$ (2) $\quad \min_{\beta\in\mathbb{R}^p} \frac{1}{2}\|y-X\beta\|_2^2+\lambda\|\beta\|_1$ (5)
  - **Ridge Regression: Convex** $\min_{\beta\in\mathbb{R}^p} \|y-X\beta\|_2^2 \text{ subject to } \|\beta\|_2^2 \le t$ (3) $\quad \min_{\beta\in\mathbb{R}^p} \frac{1}{2}\|y-X\beta\|_2^2+\lambda\|\beta\|_2^2$ (6)

  In Penalized Form

  - **Equivalence**:
    - Problems (2) and (5) are equivalent (for any $t \ge 0$ and solution β in (2), there is a value of $\lambda \ge 0$ such that β also solves (5), and vice versa)
    - Problems (3) and (6) are equivalent as well.
    - Problems (1) and (4) are *not equivalent* and *non-convex*.
  - **Sparsity & Convexity**: Only Lasso leads to both "sparse" and "convex" formulations!