Learning

Learning of a Neural Network

1. Learning with supervised learning
2. Learning with Unsupervised learning
3. Learning with Reinforcement Learning

What is supervised learning?

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.

How supervised learning works

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

- **Classification** uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.
- **Regression** is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

What is unsupervised learning?

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster **unlabeled datasets**. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

Challenges of unsupervised learning

While unsupervised learning has many benefits, some challenges can occur when it allows machine learning models to execute without any human intervention. Some of these challenges can include:

- Computational complexity due to a high volume of training data
- Longer training times
- Higher risk of inaccurate results
- Human intervention to validate output variables
- Lack of transparency into the basis on which data was clustered

Unsupervised learning models are used for **three main tasks**: clustering, association and dimensionality reduction:

- **Clustering** is a data mining technique for grouping unlabeled data based on their similarities or differences. For example, K-means clustering algorithms assign similar data points into groups, where the K value represents the size of the grouping and granularity. This technique is helpful for market segmentation, image compression, etc.
- **Association** is another type of unsupervised learning method that uses different rules to find relationships between variables in a given dataset. These methods are frequently used for market basket analysis and recommendation engines, along the lines of "Customers Who Bought This Item Also Bought" recommendations.
- **Dimensionality reduction** is a learning technique used when the number of features (or dimensions) in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the data integrity. Often, this technique is used in the preprocessing data stage, such as when autoencoders remove noise from visual data to improve picture quality.

Common unsupervised learning approaches

Unsupervised learning models are utilized for three main tasks—clustering, association, and dimensionality reduction. Below we'll define each learning method and highlight common algorithms and approaches to conduct them effectively.

Clustering

Clustering is a data mining technique which groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects into groups represented by structures or patterns in the information. Clustering algorithms can be categorized into a few types, specifically exclusive, overlapping, hierarchical, and probabilistic.

- **K-means clustering** is a common example of an exclusive clustering method where data points are assigned into K groups, where K represents the number of clusters based on the distance from each group's centroid. The data points closest to a given centroid will be clustered under the same category. A larger K value will be indicative of smaller groupings with more granularity whereas a smaller K value will have larger groupings and less granularity. K-means clustering is commonly used in market segmentation, document clustering, image segmentation, and image compression.

Overlapping clusters differs from exclusive clustering in that it allows data points to belong to multiple clusters with separate degrees of membership. "Soft" or fuzzy k-means clustering is an example of overlapping clustering.

Hierarchical clustering

Hierarchical clustering, also known as hierarchical cluster analysis (HCA), is an unsupervised clustering algorithm that can be categorized in two ways; they can be agglomerative or divisive. Agglomerative clustering is considered a "bottoms-up approach." Its data points are isolated as separate groupings initially, and then they are merged together iteratively on the basis of similarity until one cluster has been achieved. Four different methods are commonly used to measure similarity:

1. **Ward's linkage:** This method states that the distance between two clusters is defined by the increase in the sum of squared after the clusters are merged.
2. **Average linkage:** This method is defined by the mean distance between two points in each cluster
3. **Complete (or maximum) linkage:** This method is defined by the maximum distance between two points in each cluster
4. **Single (or minimum) linkage:** This method is defined by the minimum distance between two points in each cluster

Euclidean distance is the most common metric used to calculate these distances; however, other metrics, such as Manhattan distance, are also cited in clustering literature.

Divisive clustering can be defined as the opposite of agglomerative clustering; instead it takes a "top-down" approach. In this case, a single data cluster is divided based on the differences between data points. Divisive clustering is not commonly used, but it is still worth noting in the context of hierarchical clustering. These clustering processes are usually visualized using a dendrogram, a tree-like diagram that documents the merging or splitting of data points at each iteration.

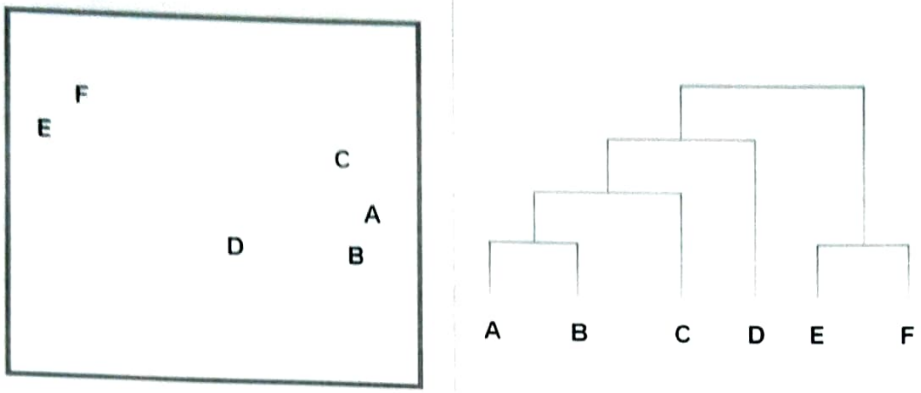
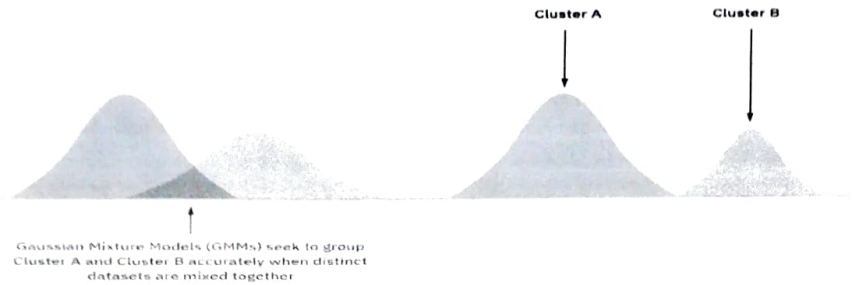


Diagram of a Dendrogram; reading the chart "bottom-up" demonstrates agglomerative clustering while "top-down" is indicative of divisive clustering

Probabilistic clustering

A probabilistic model is an unsupervised technique that helps us solve density estimation or "soft" clustering problems. In probabilistic clustering, data points are clustered based on the likelihood that they belong to a particular distribution. The Gaussian Mixture Model (GMM) is the one of the most commonly used probabilistic clustering methods.

- **Gaussian Mixture Models** are classified as mixture models, which means that they are made up of an unspecified number of probability distribution functions. GMMs are primarily leveraged to determine which Gaussian, or normal, probability distribution a given data point belongs to. If the mean or variance are known, then we can determine which distribution a given data point belongs to. However, in GMMs, these variables are not known, so we assume that a latent, or hidden, variable exists to cluster data points appropriately. While it is not required to use the Expectation-Maximization (EM) algorithm, it is a commonly used to estimate the assignment probabilities for a given data point to a particular data cluster.



Association Rules

An association rule is a rule-based method for finding relationships between variables in a given dataset. These methods are frequently used for market basket analysis, allowing companies to better understand relationships between different products. Understanding consumption habits of customers enables businesses to develop better cross-selling strategies and recommendation engines. Examples of this can be seen in Amazon's "Customers Who Bought This Item Also Bought" or Spotify's "Discover Weekly" playlist. While there are a few different algorithms used to generate association rules, such as Apriori, Eclat, and FP-Growth, the Apriori algorithm is most widely used.

Dimensionality reduction

While more data generally yields more accurate results, it can also impact the performance of machine learning algorithms (e.g. overfitting) and it can also make it difficult to visualize datasets. Dimensionality reduction is a technique used when the number of features, or dimensions, in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the integrity of the dataset as much as possible. It is commonly used in the preprocessing data stage, and there are a few different dimensionality reduction methods that can be used, such as:

Principal component analysis

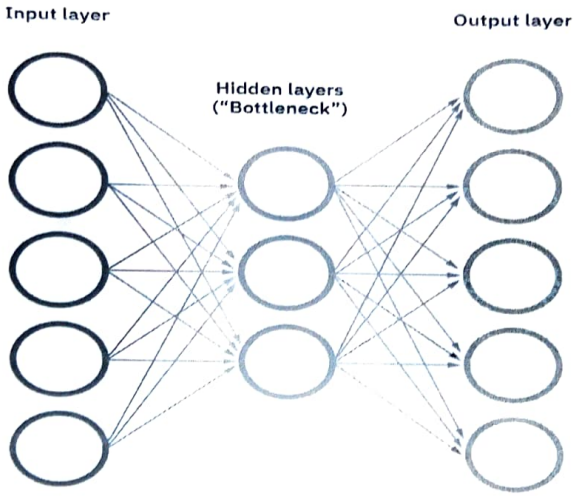
Principal component analysis (PCA) is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction. This method uses a linear transformation to create a new data representation, yielding a set of "principal components." The first principal component is the direction which maximizes the variance of the dataset. While the second principal component also finds the maximum variance in the data, it is completely uncorrelated to the first principal component, yielding a direction that is perpendicular, or orthogonal, to the first component. This process repeats based on the number of dimensions, where a next principal component is the direction orthogonal to the prior components with the most variance.

Singular value decomposition

Singular value decomposition (SVD) is another dimensionality reduction approach which factorizes a matrix, A, into three, low-rank matrices. SVD is denoted by the formula, $A = USV^T$, where U and V are orthogonal matrices. S is a diagonal matrix, and S values are considered singular values of matrix A. Similar to PCA, it is commonly used to reduce noise and compress data, such as image files.

Autoencoders

Autoencoders leverage neural networks to compress data and then recreate a new representation of the original data's input. Looking at the image below, you can see that the hidden layer specifically acts as a bottleneck to compress the input layer prior to reconstructing within the output layer. The stage from the input layer to the hidden layer is referred to as "encoding" while the stage from the hidden layer to the output layer is known as "decoding."



Applications of unsupervised learning

Machine learning techniques have become a common method to improve a product user experience and to test systems for quality assurance. Unsupervised learning provides an exploratory path to view data, allowing businesses to identify patterns in large volumes of data more quickly when compared to manual observation. Some of the most common real-world applications of unsupervised learning are:

- **News Sections:** Google News uses unsupervised learning to categorize articles on the same story from various online news outlets. For example, the results of a presidential election could be categorized under their label for "US" news.
- **Computer vision:** Unsupervised learning algorithms are used for visual perception tasks, such as object recognition.
- **Medical imaging:** Unsupervised machine learning provides essential features to medical imaging devices, such as image detection, classification and segmentation, used in radiology and pathology to diagnose patients quickly and accurately.
- **Anomaly detection:** Unsupervised learning models can comb through large amounts of data and discover atypical data points within a dataset. These anomalies can raise awareness around faulty equipment, human error, or breaches in security.
- **Customer personas:** Defining customer personas makes it easier to understand common traits and business clients' purchasing habits. Unsupervised learning allows businesses to build better buyer persona profiles, enabling organizations to align their product messaging more appropriately.
- **Recommendation Engines:** Using past purchase behavior data, unsupervised learning can help to discover data trends that can be used to develop more effective cross-selling strategies. This is used to make relevant add-on recommendations to customers during the checkout process for online retailers.

The **main difference between supervised and unsupervised learning**: Labeled data

The main distinction between the two approaches is the use of labeled datasets. To put it simply, supervised learning uses labeled input and output data, while an unsupervised learning algorithm does not.

In supervised learning, the algorithm "learns" from the training dataset by iteratively making predictions on the data and adjusting for the correct answer. While supervised learning models tend to be more accurate than unsupervised learning models, they require upfront human intervention to label the data appropriately. For example, a supervised learning model can predict how long your commute will be based on the time of day, weather conditions and so on. But first, you'll have to train it to know that rainy weather extends the driving time.

Unsupervised learning models, in contrast, work on their own to discover the inherent structure of unlabeled data. Note that they still require some human intervention for validating output variables. For example, an unsupervised learning model can identify that online shoppers often purchase groups of products at the same time. However, a data analyst would need to validate that it makes sense for a recommendation engine to group baby clothes with an order of diapers, applesauce and sippy cups.

Other key differences between supervised and unsupervised learning

- **Goals:** In supervised learning, the goal is to predict outcomes for new data. You know up front the type of results to expect. With an unsupervised learning algorithm, the goal is to

get insights from large volumes of new data. The machine learning itself determines what is different or interesting from the dataset.

- **Applications**: Supervised learning models are ideal for spam detection, sentiment analysis, weather forecasting and pricing predictions, among other things. In contrast, unsupervised learning is a great fit for anomaly detection, recommendation engines, customer personas and medical imaging.
- **Complexity:** Supervised learning is a simple method for machine learning, typically calculated through the use of programs like R or Python. In unsupervised learning, you need powerful tools for working with large amounts of unclassified data. Unsupervised learning models are computationally complex because they need a large training set to produce intended outcomes.
- **Drawbacks**: Supervised learning models can be time-consuming to train, and the labels for input and output variables require expertise. Meanwhile, unsupervised learning methods can have wildly inaccurate results unless you have human intervention to validate the output variables.

Supervised vs. unsupervised learning: Which is best for you?

Choosing the right approach for your situation depends on how your data scientists assess the structure and volume of your data, as well as the use case. To make your decision, be sure to do the following:

- **Evaluate your input data:** Is it labeled or unlabeled data? Do you have experts that can support additional labeling?
- **Define your goals:** Do you have a recurring, well-defined problem to solve? Or will the algorithm need to predict new problems?
- **Review your options for algorithms:** Are there algorithms with the same dimensionality you need (number of features, attributes or characteristics)? Can they support your data volume and structure?

Classifying big data can be a real challenge in supervised learning, but the results are highly accurate and trustworthy. In contrast, unsupervised learning can handle large volumes of data in real time. But, there's a lack of transparency into how data is clustered and a higher risk of inaccurate results. This is where semi-supervised learning comes in.

Semi-supervised learning: The best of both worlds

Can't decide on whether to use supervised or unsupervised learning? **Semi-supervised learning** is a happy medium, where you use a training dataset with both labeled and unlabeled data. It's particularly useful when it's difficult to extract relevant features from data — and when you have a high volume of data.

Semi-supervised learning is ideal for medical images, where a small amount of training data can lead to a significant improvement in accuracy. For example, a radiologist can label a small subset of CT scans for tumors or diseases so the machine can more accurately predict which patients might require more medical attention.

3. Learning with Reinforcement Learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation.

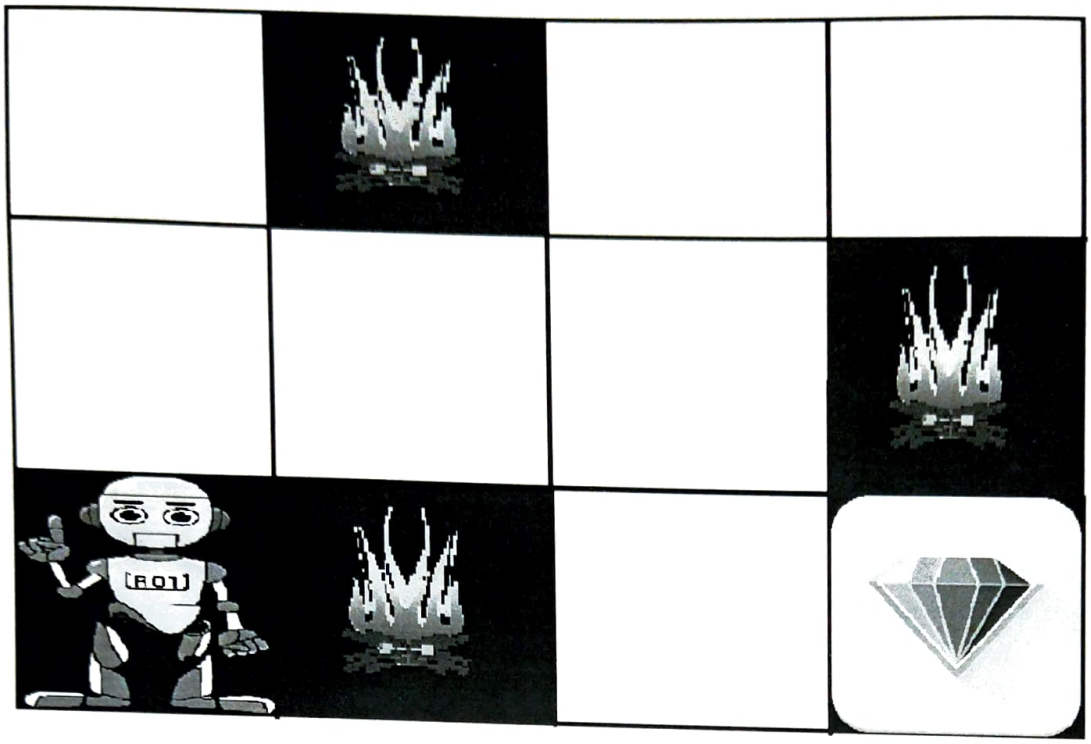
Through interaction with the environment and feedback in the form of rewards or penalties, the network gains knowledge. Finding a policy or strategy that optimizes cumulative rewards over time is the goal for the network. This kind is frequently utilized in gaming and decision-making applications.
Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.

Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice it made was correct, neutral or incorrect. It is a good technique to use for automated systems that have to make a lot of small decisions without human guidance.

Reinforcement learning is an autonomous, self-teaching system that essentially learns by trial and error. It performs actions with the aim of maximizing rewards, or in other words, it is learning by doing in order to achieve the best outcomes.

Example:

The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.



The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fired. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Difference between Reinforcement learning and Supervised learning:

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning, the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.

Reinforcement learning

Example: Chess game,text summarization

Supervised learning

Example: Object recognition,spam detetction

Hebbian Learning Rule

Hebbian Learning Rule, also known as Hebb's Learning Rule, was proposed by Donald O Hebb. It is one of the first and also easiest learning rules in the neural network. It is used for pattern classification. It is a single layer neural network, i.e. it has one input layer and one output layer. The input layer can have many units, say n. The output layer only has one unit. Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

Algorithm

- ✓Set all weights to zero, $w_i = 0$ for i=1 to n, and bias to zero.
- ✓For each input vector, S(input vector) : t(target output pair), repeat steps 3-5.
- ✓Set activations for input units with the input vector $X_i = S_i$ for i = 1 to n.
- ✓Set the corresponding output value to the output neuron, i.e. y = t.
- ✓Update weight and bias by applying Hebb rule for all i = 1 to n:

$$w_i(new) = w_i(old) + x_i y$$

$$b(new) = b(old) + y$$

Example

	INPUT			TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

There are 4 training samples, so there will be 4 iterations. Also, the activation function used here is Bipolar Step Function so the range is [-1,1].

Step 1 :

Set weight and bias to zero, $w = [\,0\,0\,0\,]^T$ and $b = 0$.

Step 2 :

Set input vector $X_i = S_i$ for i = 1 to 4.

$X_1 = [\,-1\,-1\,1\,]^T$

$X_2 = [\,-1\,1\,1\,]^T$

$X_3 = [\,1\,-1\,1\,]^T$

$X_4 = [\,1\,1\,1\,]^T$

Step 3 :

Output value is set to y = t.

Step 4 :

Modifying weights using Hebbian Rule:

First iteration –

$w(new) = w(old) + x_1y_1 = [\,0\,0\,0\,]^T + [\,-1\,-1\,1\,]^T . [\,-1\,] = [\,1\,1\,-1\,]^T$

For the second iteration, the final weight of the first one will be used and so on.

Second iteration –

$w(new) = [\,1\,1\,-1\,]^T + [\,-1\,1\,1\,]^T . [\,-1\,] = [\,2\,0\,-2\,]^T$

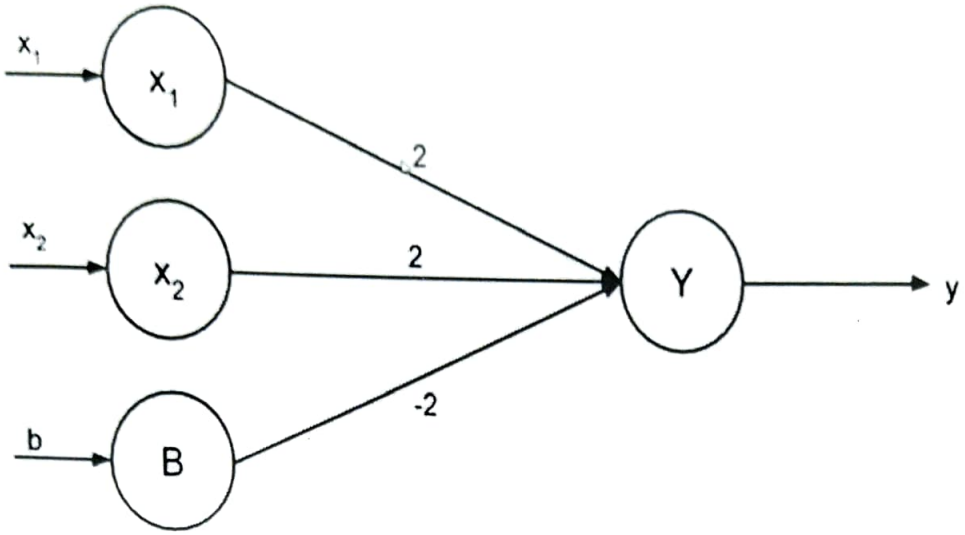
Third iteration –

$w(new) = [\,2\,0\,-2]^T + [\,1\,-1\,1\,]^T . [\,-1\,] = [\,1\,1\,-3\,]^T$

Fourth iteration –

$w(new) = [\,1\,1\,-3]^T + [\,1\,1\,1\,]^T . [\,1\,] = [\,2\,2\,-2\,]^T$

So, the final weight matrix is $[\,2\,2\,-2\,]^T$



Testing the Neural Network

For $x_1 = -1$, $x_2 = -1$, b = 1, Y = (-1)(2) + (-1)(2) + (1)(-2) = -6 = f(-6) = -1

For $x_1 = -1$, $x_2 = 1$, b = 1, Y = (-1)(2) + (1)(2) + (1)(-2) = -2 = f(-2) = -1

For $x_1 = 1$, $x_2 = -1$, b = 1, Y = (1)(2) + (-1)(2) + (1)(-2) = -2 = f(-2) = -1

For $x_1 = 1$, $x_2 = 1$, b = 1, Y = (1)(2) + (1)(2) + (1)(-2) = 2 = f(2) = +1

The results are all compatible with the original table.

Bipolar step function:
If x<0 then f(x) is -1
If x>=0 then f(x) is +1

Decision Boundary :

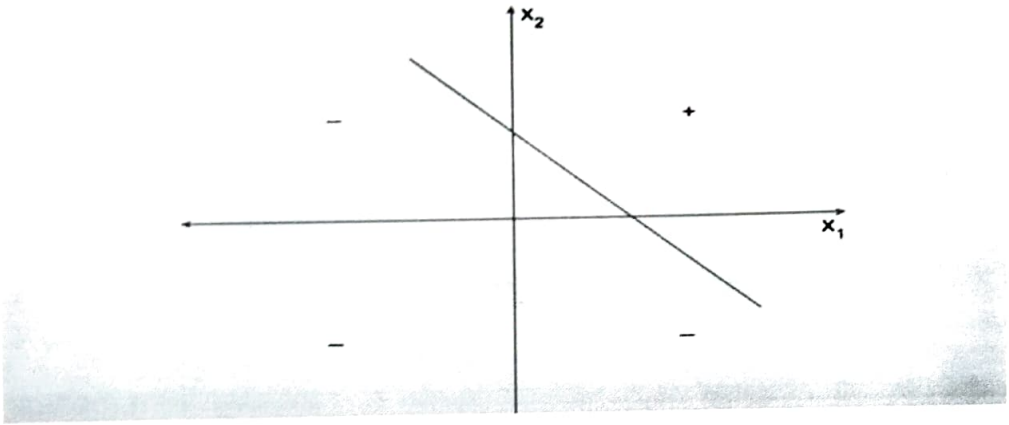
$2x_1 + 2x_2 - 2b = y$

Replacing y with 0, $2x_1 + 2x_2 - 2b = 0$

Since bias, b = 1, so $2x_1 + 2x_2 - 2(1) = 0$

$2(x_1 + x_2) = 2$

The final equation, $x_2 = -x_1 + 1$

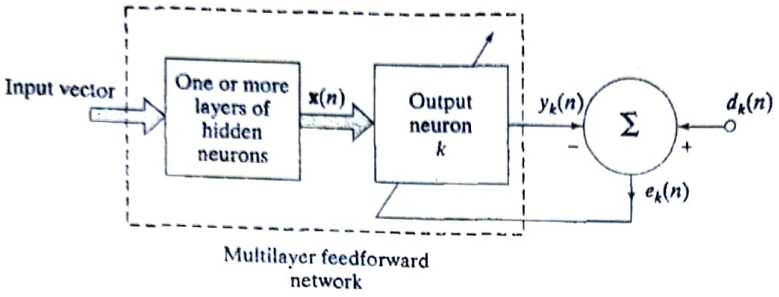


ERROR-CORRECTION LEARNING

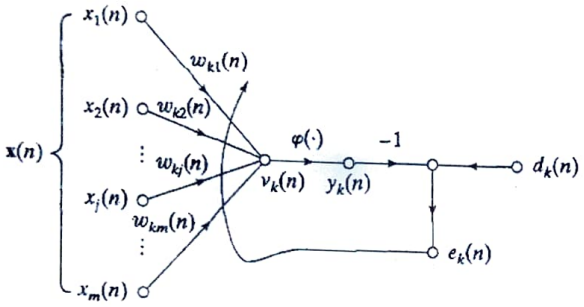
To illustrate our first learning rule, consider the simple case of a neuron k constituting the only computational node in the output layer of a feedforward neural network, as depicted in Fig. 2.1a. Neuron k is driven by a *signal vector* $\mathbf{x}(n)$ produced by one or more layers of hidden neurons, which are themselves driven by an input vector (stimulus) applied to the source nodes (i.e., input layer) of the neural network. The argument n denotes discrete time, or more precisely, the time step of an iterative process involved in adjusting the synaptic weights of neuron k. The *output signal* of neuron k is denoted by $y_k(n)$. This output signal, representing the only output of the neural network, is compared to a *desired response* or *target output,* denoted by $d_k(n)$. Consequently, an *error signal,* denoted by $e_k(n)$, is produced. By definition, we thus have

$$e_k(n) = d_k(n) - y_k(n) \tag{2.1}$$

The error signal $e_k(n)$ actuates a *control mechanism,* the purpose of which is to apply a sequence of corrective adjustments to the synaptic weights of neuron k. The corrective adjustments are designed to make the output signal $y_k(n)$ come closer to the desired



(a) Block diagram of a neural network, highlighting the only neuron in the output layer



(b) Signal-flow graph of output neuron

FIGURE 2.1 Illustrating error-correction learning.

response $d_k(n)$ in a step-by-step manner. This objective is achieved by minimizing a *cost function* or *index of performance*, $\mathscr{E}(n)$, defined in terms of the error signal $e_k(n)$ as:

$$\mathscr{E}(n) = \frac{1}{2} e_k^2(n) \tag{2.2}$$

That is, $\mathscr{E}(n)$ is the *instantaneous value of the error energy*. The step-by-step adjustments to the synaptic weights of neuron k are continued until the system reaches a *steady state* (i.e., the synaptic weights are essentially stabilized). At that point the learning process is terminated.

The learning process described herein is obviously referred to as *error-correction learning*. In particular, minimization of the cost function $\mathscr{E}(n)$ leads to a learning rule commonly referred to as the *delta rule* or *Widrow–Hoff rule*, named in honor of its originators (Widrow and Hoff, 1960). Let $w_{kj}(n)$ denote the value of 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

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \tag{2.3}$$

where η is a positive constant that determines the *rate of learning* as we proceed from one step in the learning process to another. It is therefore natural that we refer to η as the *learning-rate parameter*. In other words, the delta rule may be stated as:

Keep in mind that the delta rule, as stated herein, presumes that the error signal is *directly measurable*. For this measurement to be feasible we clearly need a supply of desired response from some external source, which is directly accessible to neuron k. In other words, neuron k is *visible* to the outside world, as depicted in Fig. 2.1a. From this figure we also observe that error-correction learning is in fact *local* in nature. This is merely saying that the synaptic adjustments made by the delta rule are localized around neuron k.

Having computed the synaptic adjustment $\Delta w_{kj}(n)$, the updated value of synaptic weight w_{kj} is determined by

$$w_{kj}(n + 1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{2.4}$$

Figure 2.1b shows a signal-flow graph representation of the error-correction learning process, focusing on the activity surrounding neuron k. The input signal x_j and induced local field v_k of neuron k are referred to as the *presynaptic* and *postsynaptic signals* of the jth synapse of neuron k, respectively. From Fig. 2.1b we see that error-correction learning is an example of a *closed-loop feedback system*. From control theory we know that the stability of such a system is determined by those parameters that constitute the feedback loops of the system. In our case we only have a single feedback loop, and one of those parameters of particular interest is the learning-rate parameter η. It is therefore important that η is carefully selected to ensure that the stability or convergence of the iterative learning process is achieved. The choice of η also has a profound influence on the accuracy and other aspects of the learning process. In short, the learning-rate parameter η plays a key role in determining the performance of error-correction learning in practice.

INTRODUCTION

- Let $d_k(n)$ denote some desired response or target response for neuron k at time n. Let the corresponding value of the actual response (output) of this neuron be denoted by $y_k(n)$.
- Typically, the actual response $y_k(n)$ of neuron k is different from the desired response $d_k(n)$. Hence, we may define an **error signal**

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

- The ultimate purpose of **error-correction learning** is to minimize a cost function based on the error signal $e_k(n)$.
- A criterion commonly used for the cost function is the **instantaneous value of the mean square-error** criterion

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

- The network is then optimized by **minimizing** J(n) **with respect to the synaptic weights of the network**. Thus, according to the error-correction learning rule (or delta rule), the synaptic weight adjustment is given by

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

- Let $w_{kj}(n)$ denote the value of the synaptic weight w_{kj} at time n. At time n an *adjustment* $\Delta w_{kj}(n)$ is applied to the synaptic weight $w_{kj}(n)$, yielding the updated value

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

ALGORITHM

Step 1: Initialize the weights of input vector and bias weight to any random value between 0 and 1.

Step 2:

Calculate the error of output neuron 'k' during an epoch 'n' using following formulae.

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

Step3:

Calculate the weight difference Δw_{kj} by using the delta rule

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

where η is Learning rate, which will be assumed as any value between 0 or 1.

Step 4:

Update the weights by using the following formulae.

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

↑ new weight ↑ old weight

Step 5:

Repeat Steps 2 to 4 until the error energy becomes as minimum as possible

Eg:

Let x_1, x_2 are the input Vectors and y is the target output.

x1	x2	yd
0	0	0
0	1	0
1	0	0
1	1	1

Assume that, the activation function ψ is binary step function.

$$\psi(v_k) = \begin{cases} 0, & \text{if } v_k \le 0 \\ 1, & \text{if } v_k > 0 \end{cases}$$

Assume the Learning rate $\eta = 0.1$.

Initialize the weights $\omega_1 = 0.3$ $\omega_2 = -0.1$ $b = -0.1$

epoch 1: $\psi(x_1\omega_1 + x_2\omega_2 + b)$ $(\eta)(e)(x_1)$ $(\eta)(e)(x_2)$ $\omega_1 + \Delta\omega_1$ $\omega_2 + \Delta\omega_2$

x_1	x_2	y_d	y_{cal}	$e = y_d - y_{cal}$	$\Delta\omega_1$	$\Delta\omega_2$	ω_1	ω_2
							0.3	-0.1
0	0	0	0	0	0	0		
							0.3	-0.1
0	1	0	0	0	0	0		
							0.2	-0.1
1	0	0	1	-1	-0.1	0		
							0.3	0
1	1	1	0	1	0.1	0.1		

epoch 2:

x_1	x_2	y_d	y_{act}	e	Δw_1	Δw_2	w_1	w_2
					0	0	0.3	0
0	0	0	0	0				
0	1	0	0	0	0	0	0.3	0
1	0	0	1	-1	$(0.1)(-1)(1)$ $=-0.1$	0	0.2	0
1	1	1	1	0	0	0	0.2	0

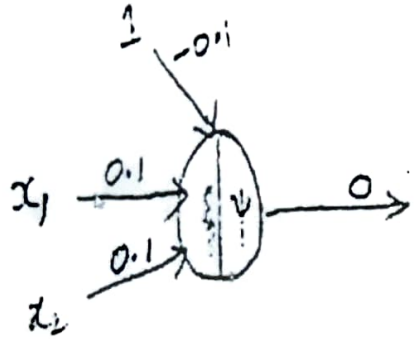
Epoch 3:

x_1	x_2	y_d	y_{act}	e	$\Delta w1$	$\Delta w2$	$w1$	$w2$
0	0	0	0	0	0	0	0.2	0
0	1	0	0	0	0	0	0.2	0
1	0	0	1	-1	-0.1	0	0.1	0
1	1	1	0	1	0.1	0.1	0.2	0.1

Epoch 4:

x_1	x_2	y_d	y_{act}	e	$\Delta w1$	$\Delta w2$	$w1$	$w2$
0	0	0	0	0	0	0	0.2	0.1
0	1	0	0	0	0	0	0.2	0.1
1	0	0	1	-1	-0.1	0	0.1	0.1
1	1	1	1	0	0	0	0.1	0.1

∴ The error correction learning NN for AND gate



Epoch 5:

x_1	x_2	y_d	y_{ack}	e	Δw_1	Δw_2	w_1	w_2
0	0	0	0	0	0	0	0.1	0.1
0	1	0	0	0	0	0	0.1	0.1
1	0	0	0	0	0	0	0.1	0.1
1	1	1	1	0	0	0	0.1	0.1

Competitive Learning:

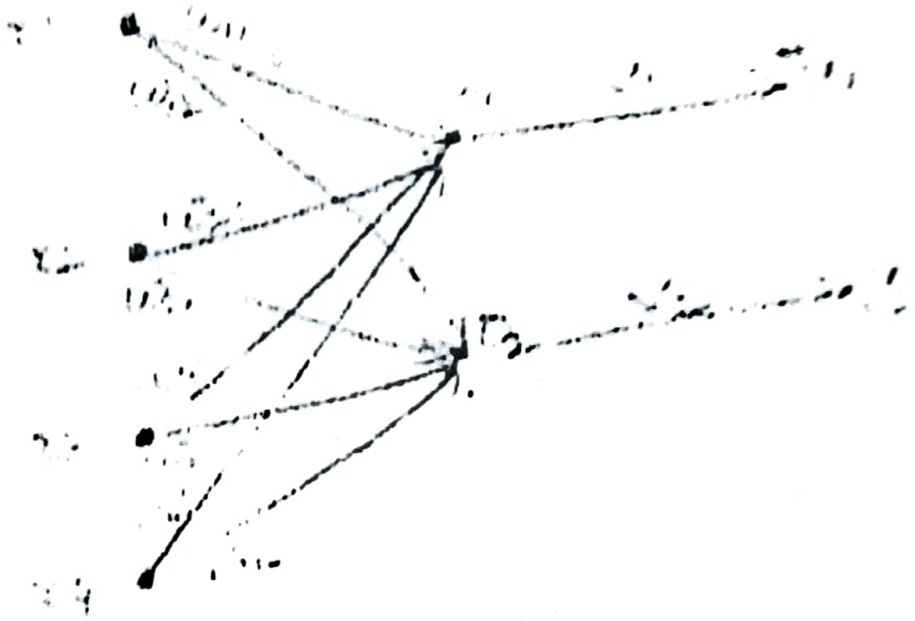
Example

Construct a Competitive Learning neural network for the input vectors

[0 0 1 1] [1 0 0 0] [0 1 1 0] [0 0 0 1].

Assume the learning weight $\eta = 0.5$ and the no. of clusters that input vectors falls into is 2

no. of clusters = 2



Step 1 :

Initialize w_{ij} matrix to random values

$$w_{ij} = \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{matrix} \begin{matrix} 1 & \quad 2 \end{matrix} \\ \begin{pmatrix} 0.2 & 0.9 \\ 0.4 & 0.7 \\ 0.6 & 0.5 \\ 0.8 & 0.3 \end{pmatrix} \end{matrix}$$

Step 2:

Calculate the induced outputs v_1 and v_2 of o_1 and o_2

for input vector $[0 \quad 0 \quad 1 \quad 1]$
$x_1 \quad x_2 \quad x_3 \quad x_4$

$$V_1 = x_1 * w_{11} + x_2 * w_{21} + x_3 * w_{31} + x_4 * w_{41}.$$

$$V_1 = 0+0+1*0.6+1*0.8$$

$$\boxed{V_1 = 1.4}$$

$$V_2 = x_1 * w_{12} + x_2 * w_{22} + x_3 * w_{32} + x_4 * w_{42}$$

$$V_1 = 0+0+1*0.5+1*0.3$$

$$\boxed{V_2 = 0.8}$$

Since $V_1 > V_2$, output of O_1 is Y_1

$$\Rightarrow \boxed{Y_1 = 1 \quad Y_2 = 0}$$

Calculate the change in weights

$$\Delta w_{ij} = \begin{cases} \eta (x_i - w_{ij}) & \text{if } y_j \text{ is winner} \\ 0 & \end{cases}$$

$$\Delta w_{11} = \eta (x_1 - w_{11}) = 0.5 (0 - 0.2) = -0.1$$

$$\Delta w_{21} = \eta (x_2 - w_{21}) = 0.5 (0 - 0.4) = -0.2$$

$$\Delta w_{31} = \eta (x_3 - w_{31}) = 0.5 (1 - 0.6) = 0.2$$

$$\Delta w_{41} = \eta (x_4 - w_{41}) = 0.5 (1 - 0.8) = 0.1$$

Step 3: calculate the induced outputs V_1 & V_2 of o/p

for input vector $[x_1 \; x_2 \; x_3 \; x_4] = [1 \; 0 \; 0 \; 0]$

$$V_1 = x_1 * w_{11} + x_2 * w_{21} + x_3 * w_{31} + x_4 * w_{41}$$

$$= 1 * 0.1 + 0 + 0 + 0 = 0.1$$

$$V_2 = x_1 * w_{12} + x_2 * w_{22} + x_3 * w_{32} + x_4 * w_{42}$$

$$= 1 * 0.9 + 0 + 0 + 0 = 0.9$$

$\Delta\omega_{12} = \eta(x_1 - \omega_{12}) = 0.5(1-0.9) = 0.05$

$\Delta\omega_{22} = \eta(x_2 - \omega_{22}) = 0.5(0-0.7) = -0.35$

$\Delta\omega_{32} = \eta(x_3 - \omega_{32}) = 0.5(0-0.5) = -0.25$

$\Delta\omega_{42} = \eta(x_4 - \omega_{42}) = 0.5(0-0.3) = -0.15$

Update weight matrix

$$\omega_{ij} = \omega_{ij} + \Delta\omega_{ij}$$

$$\omega_{ij} = \begin{pmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{pmatrix}$$

$\Delta w_0 = 0$

$\Delta w_{12} = 0$

$\Delta w_{32} = 0$

$\Delta w_{42} = 0$

→ update weight matrix $w_{ij} = w_{ij} + \Delta w_{ij}$
$= 0.2 - 0.1 = 0.1$

$$w_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$$

$\Delta w_{12} = \eta(x_1 - w_{12}) = 0.5(1-0.9) = 0.05$

$\Delta w_{22} = \eta(x_2 - w_{22}) = 0.5(0-0.7) = -0.35$

$\Delta w_{32} = \eta(x_3 - w_{32}) = 0.5(0-0.5) = -0.25$

$\Delta w_{42} = \eta(x_4 - w_{42}) = 0.5(0-0.3) = -0.15$

update weight matrix

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

$$w_{ij} = \begin{pmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{pmatrix}$$

Step 4:

Calculate the induced matrix outputs V_1 & V_2 of O_1 & O_2 for input vector $[0 \; 1 \; 1 \; 0]$

$$V_1 = x_1 * w_{11} + x_2 * w_{21} + x_3 * w_{31} + x_4 * w_{41}$$

$$= 0 + 1 * 0.2 + 1 * 0.8 + 0$$

$$= 1$$

$$v_2 = x_1 * \omega_{12} + x_2 * \omega_{32} + x_3 * \omega_{32} + x_4 * \omega_{42}$$

$$= 0 + 1 * 0.35 + 1 * 0.35 + 0$$

$$= 0.60$$

Since $v_1 > v_2$ ⇒ $\begin{vmatrix} y_1 & 1 \\ Y_1 & 0 \end{vmatrix}$

$$\Delta\omega_{11} = \eta(x_1 - \omega_{11}) = 0.5(0 - 0.1) = -0.05$$

$$\Delta\omega_{21} = \eta(x_2 - \omega_{21}) = 0.5(1 - 0.2) = 0.5(0.8) = 0.4$$

$$\Delta\omega_{31} = \eta(x_3 - \omega_{24}) = 0.5(1 - 0.8) = 0.5(0.2) = 0.1$$

$$\Delta\omega_{41} = \eta(x_4 - \omega_{41}) = 0.5(0 - 0.9) = (-0.9)(0.5) = -0.45$$