# Lecture 1

## Recommended system

Recommendation engines (REs) are most famously used for explaining what machine learning is to any unknown person or a newbie who wants to understand the field of machine learning. A classic example could be how Amazon recommends books similar to the ones you have bought, which you may also like very much! Also, empirically, the recommender engine seems to be an example of large-scale machine learning that everyone understands, or perhaps already understood. But, nonetheless, recommendation engines are being used everywhere. For example, the people you may know feature in Facebook or LinkedIn, which recommends by showing the most probable people you might like to befriend or professionals in your network who you might be interested in connecting with. Of course, these features drive their businesses big time and it is at the heart of the company's driving force.

The idea behind an RE is to predict what people might like and to uncover relationships between items/products to aid in the discovery process; in this way, it is similar to a search engine. But a major difference is, search engine works in a reactive manner; they show results only when the user requests something–but a recommendation engine is proactive– it tries to present people with relevant content that they did not necessarily search for or that they might not have heard of in the past.

Recommender systems are utilized in a variety of areas and are most commonly recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for services such as Amazon, or content recommenders for social media platforms such as Facebook and Twitter. These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. There are also popular recommender systems for specific topics like restaurants and online dating. Recommender systems have also been developed to explore research articles and experts, collaborators, and financial services

## Evaluation of recommendation engine model

Evaluation of any model needs to be calculated in order to determine how good the model is with respect to the actual data so that its performance can be improved by tuning hyperparameters and so on. In fact, the entire machine learning algorithm's accuracy is measured based on its type of problem. In the case of classification problems, confusion matrix, whereas in regression problems, mean squared error or adjusted R-squared values need to be computed.

Mean squared error is a direct measure of the reconstruction error of the original sparse user-item matrix (also called A) with two low-dimensional dense matrices (X and Y). It is also the objective function which is being minimized across the iterations:

$$\text{Mean Squared Error} = \frac{\text{Sum of Squared Errors}}{\text{Number of Observations}}$$

$$\text{RMSE (Root Mean Squared Error)} = \sqrt{\text{Mean Squared Error}}$$

Root mean squared errors provide the dimension equivalent to the original dimension of the variable measure, hence we can analyze how much magnitude the error component has with the original value. In our example, we have computed the root mean square error (RMSE) for the movie lens data.

Root mean squared errors provide the dimension equivalent to the original dimension of the variable measure, hence we can analyze how much magnitude the error component has with the original value. In our example, we have computed the root mean square error (RMSE) for the movie lens data.

Recommender systems usually make use of either or both collaborative filtering and content-based filtering (also known as the personality-based approach), as well as other systems such as knowledge-based systems. Collaborative filtering approaches build a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in. Content-based filtering approaches utilize a series of discrete, pre-tagged characteristics of an item in order to recommend

additional items with similar properties. Current recommender systems typically combine one or more approaches into a hybrid system.

### Hyperparameter selection in recommendation engines using grid search

In any machine learning algorithm, the selection of hyperparameter plays a critical role in how well the model generalizes the underlying data. In a similar way, in recommendation engines, we have the following hyperparameter to play with:

- **Number of iterations:** The higher the number of iterations, the better the algorithm converges. In practice, it has been proven that ALS converges within 10 iterations, but readers are recommended to try various values and see how the algorithm works.
- **Number of latent factors:** These are the explanatory variables which try to provide reasons behind crowd behavior patterns. The higher the latent factors, the better the model, but too high a value may not provide significant lift.
- **Learning rate:** The learning rate is a tunable knob to change the rate of convergence of the algorithm. Too high a value may shoot out rather than converge due to high oscillations, and too low a value may let the algorithm take too many steps to converge.

### Multi-criteria recommender systems

Multi-criteria recommender systems (MCRS) can be defined as recommender systems that incorporate preference information upon multiple criteria. Instead of developing recommendation techniques based on a single criterion value, the overall preference of user u for the item i, these systems try to predict a rating for unexplored items of u by exploiting preference information on multiple criteria that affect this overall preference value. Several researchers approach MCRS as a multi-criteria decision making (MCDM) problem, and apply MCDM methods and techniques to implement MCRS systems.

## Risk-aware recommender systems

The majority of existing approaches to recommender systems focus on recommending the most relevant content to users using contextual information, yet do not take into account the risk of disturbing the user with unwanted notifications. It is important to consider the risk of upsetting the user by pushing recommendations in certain circumstances, for instance, during a professional meeting, early morning, or late at night. Therefore, the performance of the recommender system depends in part on the degree to which it has incorporated the risk into the recommendation process. One option to manage this issue is DRARS, a system which models the context-aware recommendation as a bandit problem.

## Mobile recommender systems

Mobile recommender systems make use of internet-accessing smart phones to offer personalized, context-sensitive recommendations. This is a particularly difficult area of research as mobile data is more complex than data that recommender systems often have to deal with. It is heterogeneous, noisy, requires spatial and temporal auto-correlation, and has validation and generality problems

# Lecture 2

## Content-based filtering

Content-based methods try to use the content or attributes of the item, together with some notion of similarity between two pieces of content, to generate similar items with respect to the given item. In this case, cosine similarity is used to determine the nearest user or item to provide recommendations.

In this system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past, or is examining in the present. It does not rely on a user sign-in mechanism to generate this often temporary profile. In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

To create a user profile, the system mostly focuses on two types of information:

1. A model of the user's preference.
2. A history of the user's interaction with the recommender system.

Basically, these methods use an item profile (i.e., a set of discrete attributes and features) characterizing the item within the system. To abstract the features of the items in the system, an item presentation algorithm is applied. A widely used algorithm is the tf?idf representation (also called vector space representation). The system creates a content-based profile of users based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually rated content vectors using a variety of techniques. Simple approaches use the average values of the rated item vector while other sophisticated methods use machine learning techniques such as Bayesian Classifiers, cluster analysis, decision trees, and artificial neural networks in order to estimate the probability that the user is going to like the item.

A key issue with content-based filtering is whether the system is able to learn user preferences from users' actions regarding one content source and use them

across other content types. When the system is limited to recommending content of the same type as the user is already using, the value from the recommendation system is significantly less than when other content types from other services can be recommended. For example, recommending news articles based on browsing of news is useful, but would be much more useful when music, videos, products, discussions etc. from different services can be recommended based on news browsing. To overcome this, most content-based recommender systems now use some form of hybrid system.

Content based recommender systems can also include opinion-based recommender systems. In some cases, users are allowed to leave text review or feedback on the items. These user-generated texts are implicit data for the recommender system because they are potentially rich resource of both feature/aspects of the item, and users' evaluation/sentiment to the item. Features extracted from the user-generated reviews are improved meta-data of items, because as they also reflect aspects of the item like meta-data, extracted features are widely concerned by the users. Sentiments extracted from the reviews can be seen as users' rating scores on the corresponding features. Popular approaches of opinion-based recommender system utilize various techniques including text mining, information retrieval, sentiment analysis (see also Multimodal sentiment analysis) and deep learning

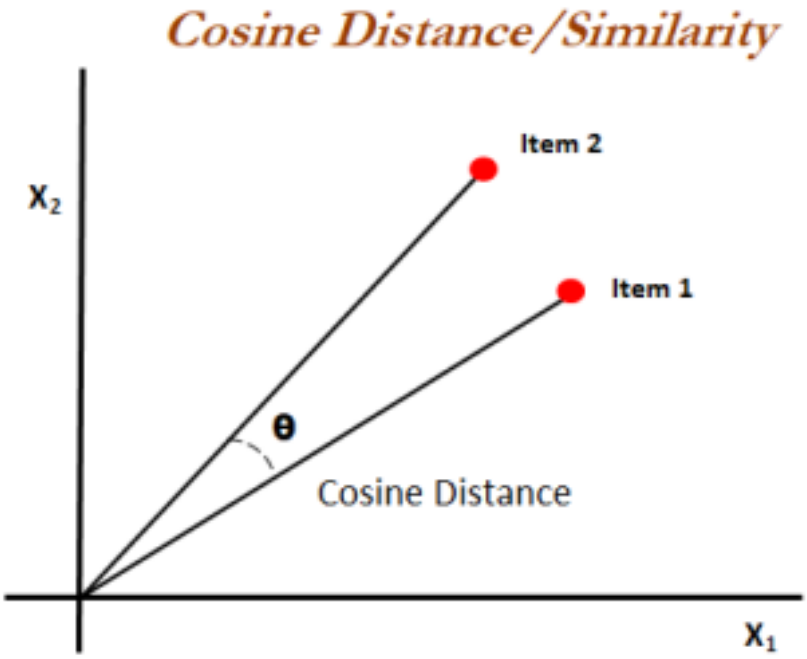
Example: If you buy a book, then there is a high chance you'll buy related books which have frequently gone together with all the other customers, and so on.

## Cosine similarity

As we will be working on this concept, it would be nice to reiterate the basics. Cosine similarity is a measure of similarity between two nonzero vectors of an inner product space that measures the cosine of the angle between them. Cosine of 0 deg is 1 and it is less than 1 for any other angle:

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Here, $A_i$ and $B_i$ are components of vector $A$ and $B$ respectively:




A key issue with content-based filtering is whether the system is able to learn user preferences from users' actions regarding one content source and use them across other content types. When the system is limited to recommending content of the same type as the user is already using, the value from the recommendation system is significantly less than when other content types from other services can be recommended. For example, recommending news articles based on browsing of news is useful, but would be much more useful when music, videos, products, discussions etc. from different services can be recommended based on news browsing. To overcome this, most content-based recommender systems now use some form of hybrid system.

Content based recommender systems can also include opinion-based recommender systems. In some cases, users are allowed to leave text review or feedback on the items. These user-generated texts are implicit data for the recommender system because they are potentially rich resource of both feature/aspects of the item, and users' evaluation/sentiment to the item. Features extracted from the user-generated reviews are improved meta-data of items, because as they also reflect aspects of the

item like meta-data, extracted features are widely concerned by the users. Sentiments extracted from the reviews can be seen as users' rating scores on the corresponding features. Popular approaches of opinion-based recommender system utilize various techniques including text mining, information retrieval, sentiment analysis (see also Multimodal sentiment analysis) and deep learnin

# Lecture 3

## Collaborative filtering

Collaborative filtering is a form of wisdom-of-the-crowd approach, where the set of preferences of many users with respect to items is used to generate estimated preferences of users for items with which they have not yet rated/reviewed. It works on the notion of similarity. Collaborative filtering is a methodology in which similar users and their ratings are determined not by similar age and so on, but by similar preferences exhibited by users, such as similar movies watched, rated, and so on.

### Advantages of collaborative filtering over content-based filtering

Collaborative filtering provides many advantages over content-based filtering. A few of them are as follows:

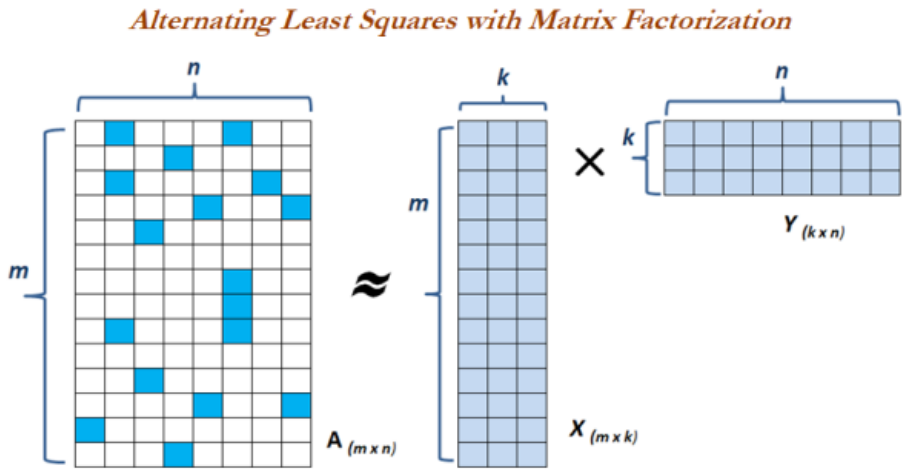
- **Not required to understand item content:** The content of the items does not necessarily tell the whole story, such as movie type/genre, and so on.
- **No item cold-start problem:** Even when no information on an item is available, we still can predict the item rating without waiting for a user to purchase it.
- **Captures the change in user interests over time:** Focusing solely on content does not provide any flexibility on the user's perspective and their preferences.
- **Captures inherent subtle characteristics:** This is very true for latent factor models. If most users buy two unrelated items, then it is likely that another user who shares similar interests with other users is highly likely to buy that unrelated item.

**Matrix factorization using the alternating least squares algorithm for collaborative filtering**

**Alternating least squares (ALS)** is an optimization technique to solve the matrix factorization problem. This technique achieves good performance and has proven relatively easy to implement. These algorithms are members of a broad class of latent-factor models and they try to explain observed interactions between a large number of users and items/movies through a relatively small number of unobserved, underlying reasons/factors. The matrix factorization algorithm treats the user-item data (matrix dimensions $m \times n$) as a sparse matrix and tries to reconstruct with two lower-dimensional dense matrices (X and Y, where X has dimensions $m \times k$ and Y has dimensions $k \times n$, in which k is the number of latent factors).

Latent factors can be interpreted as some explanatory variables which try to explain the reasons behind the behavior of users, as collaborative filtering is all about trying to predict based on the behavior of users, rather than attributes of movies or users, and so on:

$$A_{(m \times n)} \approx X_{(m \times k)} \times Y_{(k \times n)}$$




By multiplying X and Y matrices, we try to reconstruct the original matrix A, by reducing the root mean squared errors between original available ratings from sparse matrix A ($m \times n$ dimensions) and constructed dense matrix by multiplying

X ($m \times k$ dimensions) and Y ($k \times n$ dimensions) matrices. However, the matrix obtained from multiplying X and Y fills all the slots in $m \times n$ dimensions, but we will reduce the errors between only available ratings in A. By doing so, all the other values in blank slots will produce reasonably sensible ratings.

However, in this scenario, there are too many unknown values. Unknown values are nothing but the values that need to be filled in X and Y matrices so that it will try to approximate the original ratings in matrix A as closely as possible. To solve this problem, initially, random values are generated between 0 to 1 from uniform distribution and multiplied by 5 to generate the values between 0 to 5 for both X and Y matrices. In the next step, the ALS methodology is actually applied; the following two steps are applied iteratively until the threshold number of iterations is reached:

1. X values are updated by utilizing Y values, learning rate ($\lambda$), and original sparse matrix (A)

2. Y values are updated by utilizing X values, learning rate ($\lambda$), and original sparse matrix (A)

The learning rate ($\lambda$) is used to control the rate of convergence; a high value leads to rapid change in values (similar to optimization problems) and sometimes will lead to overshoot from the optimum value. In a similar way, a low value requires many iterations to converge the solution.

The mathematical representation of ALS is provided here:

$$A = X \times Y$$

$$X = A \times Y^{-1}$$

$$X = A \times (Y^T \times Y^{T^{-1}}) \times Y^{-1}$$

$$X = A \times Y^T \times (YY^T)^{-1}$$

$$minimize\ error = minimize|X - A \times Y^T \times (YY^T)^{-1}|$$

The goal is to minimize the error or squared differences between the two. Hence, it has been called the least squares technique. In simple machine learning terminology, we can call this a regression problem, as the error between actual and predicted is being minimized. Practically, this equation was never solved by computing inverses. However, equivalence has been achieved by computing Y from X and again computing X from Y and, this way, it will continue until all iterations are reached and this is where the alternating part came actually. At the start, Y was artificially generated and X will be optimized based on Y, and later by optimizing Y based on X; by doing so, eventually the solution starts to converge towards the optimum over the number of iterations.

# Lecture 4

## Artificial neural network

The brain is a highly complex, nonlinear, and parallel computer (information-processing system). It has the capability to organize its structural constituents, known as neurons, so as to perform certain computations (e.g., pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. Example - human vision and sonar of bat. In its most general form, a neural network is a machine that is designed to model the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on a digital computer.

*A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:*

1. *Knowledge is acquired by the network from its environment through a learning process.*

2. *Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.*

The procedure used to perform the learning process is called a *learning algorithm*, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective. However, it is also possible for a neural network to modify its own topology, which is motivated by the fact that neurons in the human brain can die and new synaptic connections can grow.

### Benefits of ANN

It is apparent that a neural network derives its computing power through, first, its massively parallel distributed structure and, second, its ability to learn and therefore

generalize. Generalization refers to the neural network?s production of reasonable outputs for inputs not encountered during training (learning).

Neural networks offer the following useful properties and capabilities:

1. *Nonlinearity.* An artificial neuron can be linear or nonlinear.A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear. Moreover, the nonlinearity is of a special kind in the sense that it is distributed throughout the network.

2. *Input-Output Mapping.* A popular paradigm of learning, called learning with a teacher, or supervised learning, involves modification of the synaptic weights of a neural network by applying a set of labeled training examples, or task examples. The network is presented with an example picked at random from the set, and the synaptic weights (free parameters) of the network are modified to minimize the difference between the desired response and the actual response of the network produced by the input signal in accordance with an appropriate statistical criterion. The training of the network is repeated for many examples in the set, until the network reaches a steady state where there are no further significant changes in the synaptic weights.

3. *Adaptivity.* Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment. In particular, a neural network trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental conditions. Moreover, when it is operating in a nonstationary environment (i.e., one where statistics change with time), a neural network may be designed to change its synaptic weights in real time. makes it a useful tool in adaptive pattern classification, adaptive signal processing, and adaptive control.

4. *Evidential Response.* In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to select, but also about the confidence in the decision made.This latter information may be used to reject ambiguous patterns, should they arise, and thereby improve the classification performance of the network.

5. *Contextual Information.* Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, contextual information is dealt with naturally by a neural network.

6. *Fault Tolerance.* A neural network, implemented in hardware form, has the potential to be inherently fault tolerant, or capable of robust computation, in the sense that its performance degrades gracefully under adverse operating conditions.

## Biological Neuron

## Mathematical model of neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network. The block diagram given shows the model of a neuron, which forms the basis for designing a large family of neural networks. There is three basic element of the neural model.

1. A set of synapses, or connecting links, each of which is characterized by a weight or strength of its own. Specifically, a signal $x_j$ at the input of synapse $j$ connected to neuron $k$ is multiplied by the synaptic weight $w_{kj}$.

2. An *adder* for summing the input signals,weighted by the respective synaptic strengths of the neuron; the operations described here constitute a *linear combiner*

3. An activation function for limiting the amplitude of the output of a neuron.The activation function is also referred to as a squashing function, in that it squashes (limits) the permissible amplitude range of the output signal to some finite value.

The neural model of Fig also includes an externally applied bias, denoted by $b_k$. The bias $b_k$ has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively.

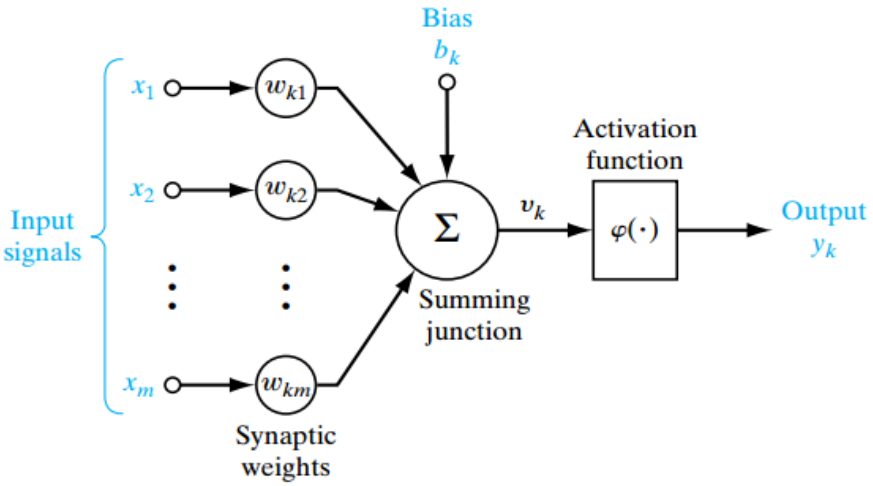



Figure 1: Nonlinear model of a neuron, labeled $k$

In mathematical terms, we may describe the neuron k depicted in Figure by writing the pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

and

$$y_k = \phi(u_k + b_k)$$

where $x_1, x_2, ..., x_m$ are the input signal; $w_{k1}, w_{k2}, ..., w_{km}$ are the respective synaptic weight of neuron $k$; $u_k$ is the linear combiner output due to the input signals;$b_k$ is the bias; $\phi(.)$ is the activation function; and $y_k$ is the output signal of the neuron.

### Type of activation function

The activation function, denoted by (v), defines the output of a neuron in terms of the induced local field v.

1. *Threshold Function* For this type of activation function, we have

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

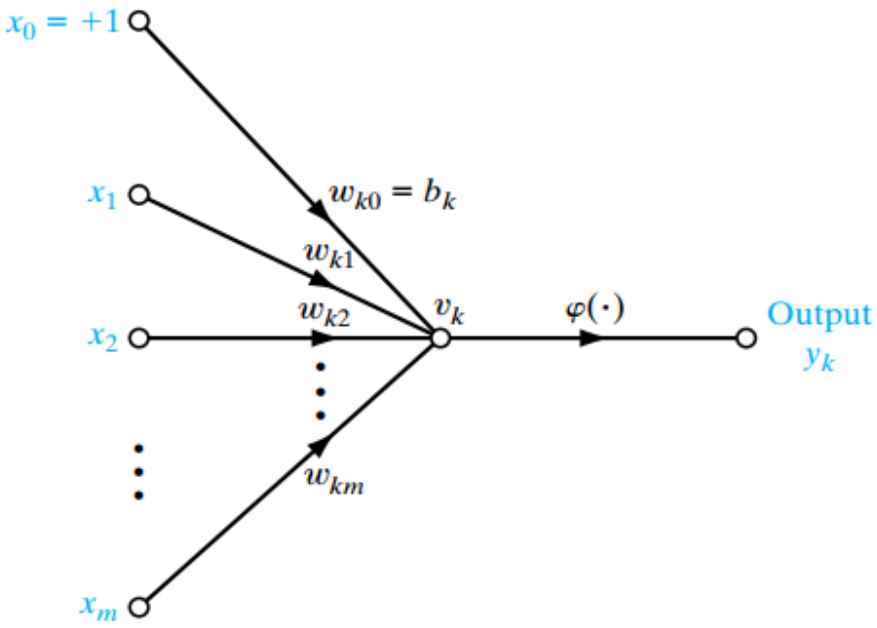



Figure 2: Nonlinear model of a neuron, labeled $k$

In neural computation, such a neuron is referred to as the McCulloch?Pitts model,In this model, the output of a neuron takes on the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise.

2. *Sigmoid Function* The sigmoid function, whose graph is "S"-shaped, s defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. An example of the sigmoid function is the logistic function, defined by

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

where a is the slope parameter of the sigmoid function.

## Network Architectures

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network.We may therefore speak of learning algorithms (rules) used in the design of neural networks as being

*structured*.

In general, we may identify three fundamentally different classes of network architectures:

### Single-Layer Feedforward Network

In a layered neural network, the neurons are organized in the form of layers. In the simplest form of a layered network, we have an *input layer* of source nodes that projects directly onto an *output layer* of neurons (computation nodes), but not vice versa. In other words, this network is strictly of a *feedforward type*.Such a network is called a *single-layer network*, with the designation "single-layer" referring to the output layer of computation nodes (neurons).

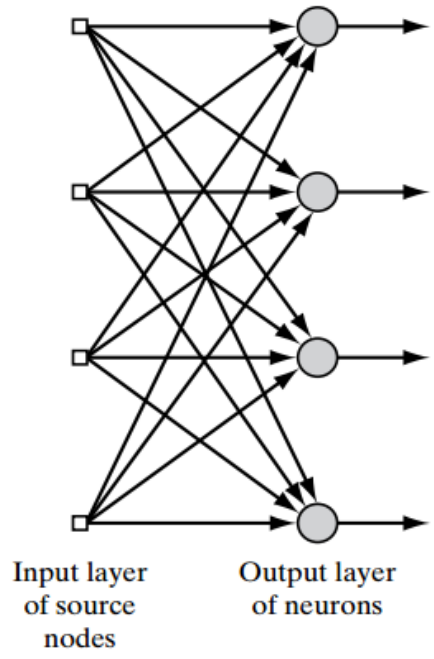



Figure 3: Feedforward network with a single layer neurons.

### Multilayer Feedforward Networks

The second class of a feedforward neural network distinguishes itself by the presence of one or more *hidden layers*, whose computation nodes are correspondingly called *hidden neurons* or *hidden units*; the term "hidden" refers to the fact that this part of the neural network is not seen directly from either the input or output of the network. he function of hidden neurons is to intervene between the external input and the network output in some useful manner. By adding one or more hidden layers, the network is enabled to extract higher-order statistics from its input. The neural network is said to be *fully connected* in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. If, however, some of the communication links (synaptic connections) are missing from the network, we say that the network is *partially connected*.

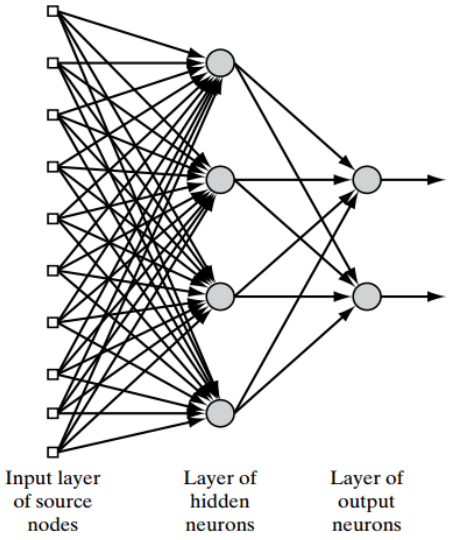



Figure 4: Fully connecte feedforward network with one hidden layer and one output layer..

### Recurrent Networks

A recurrent neural network distinguishes itself from a feedforward neural network in that it has at least one feedback loop. For example, a recurrent network may

consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons

The feedback loops involve the use of particular branches composed of unit-time delay elements (denoted by $z_{-1}$), which result in a nonlinear dynamic behavior, assuming that the neural network contains nonlinear units.

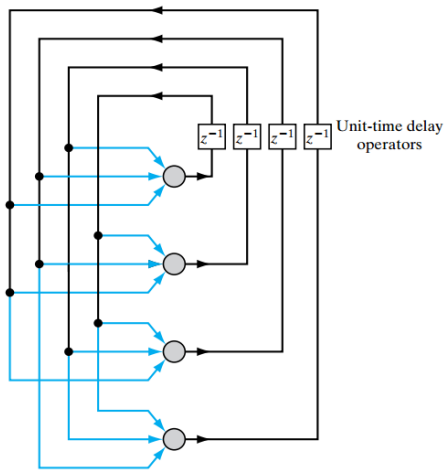



Figure 5: Recurrent network with no self-feedback loops and no hidden neurons.

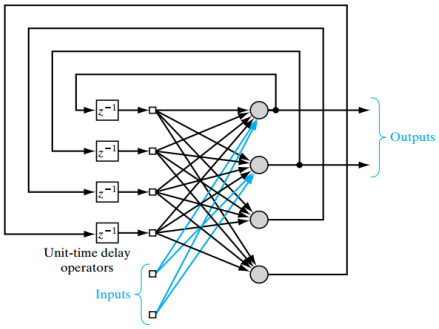



Figure 6: Recurrent network with hidden neurons.

# Lecture 5

## Perceptron

The perceptron is the simplest form of a neural network used for the classification of patterns said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane). Basically, it consists of a single neuron with adjustable synaptic weights and bias.

Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes. The proof of convergence of the algorithm is known as the perceptron convergence theorem.

By expanding the output (computation) layer of the perceptron to include more than one neuron, we may correspondingly perform classification with more than two classes. However, the classes have to be linearly separable for the perceptron to work properly.

Rosenblatt?s perceptron is built around a nonlinear neuron, namely, the McCulloch?Pitts model of a neuron - a neural model consists of a linear combiner followed by a hard limiter (performing the signum function). The summing node of the neural model computes a linear combination of the inputs applied to its synapses, as well as incorporates an externally applied bias. The resulting sum, that is, the induced local field, is applied to a hard limiter. Accordingly, the neuron produces an output equal to +1 if the hard limiter input is positive, and -1 if it is negative.

In the signal-flow graph model of Fig. 7, the synaptic weights of the perceptron are denoted by $w_1, w_2, ..., w_m$. Correspondingly, the inputs applied to the perceptron are denoted by $x_1, x_2, ..., x_m$. The externally applied bias is denoted by b. From the model, we find that the hard limiter input, or induced local field, of the neuron is

$$v = \sum_{i=1}^{m} w_i x_i + b$$

The goal of the perceptron is to correctly classify the set of externally applied stimuli $x_1, x_2, ..., x_m$ into one of two classes, $\mathcal{C}_1$ or $\mathcal{C}_2$. The decision rule for the classification is

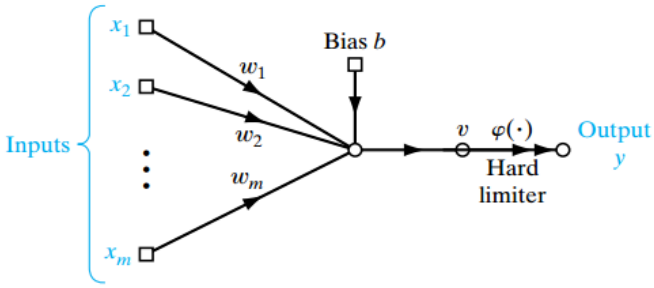



Figure 7: Recurrent network with hidden neurons.

to assign the point represented by the inputs $x_1, x_2, ..., x_m$ to class $\mathcal{C}_1$ if the perceptron output $y$ is +1 and to class $\mathcal{C}_2$ if it is -1.

In the simplest form of the perceptron, there are two decision regions separated by a hyperplane, which is defined by

$$\sum_{i=1}^{m} w_i x_i + b$$

for the case of two input variables $x1$ and $x2$, for which the decision boundary takes the form of a straight line. A point $(x_1, x_2)$ that lies above the boundary line is assigned to class $\mathcal{C}_1$, and a point $(x_1, x_2)$ that lies below the boundary line is assigned to class $\mathcal{C}_2$. The effect of the bias b is merely to shift the decision boundary away from the origin.

The synaptic weights $w_1, w_2, ..., w_m$ of the perceptron can be adapted on an iterationby-iteration basis. For the adaptation, we may use an error-correction rule known as the perceptron convergence algorithm, given in Fig-10

## The Batch Perception Algorithm

We define the *perceptron cost function* as

$$J(\mathbf{w}) = \sum_{\mathbf{x}(n)\in\mathcal{X}} \left(-\mathbf{w}^T\mathbf{x}(n)d(n)\right)$$

where $\mathcal{X}$ is the set of samples $\mathbf{x}$ misclassified by a perceptron using w as its weight vector . If all the samples are classified correctly, then the set is empty, in which

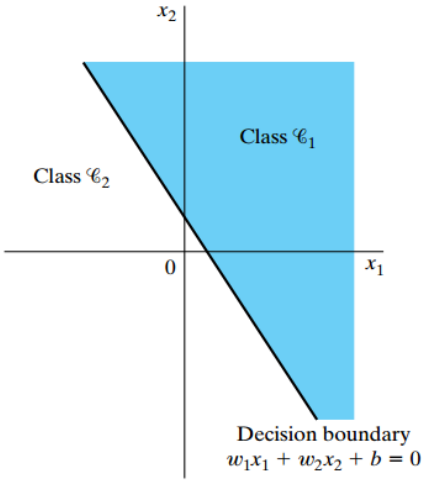



Figure 8: hyperplane, in this example, as decision boundary for a two-dimensional, two-class pattern-classification problem.

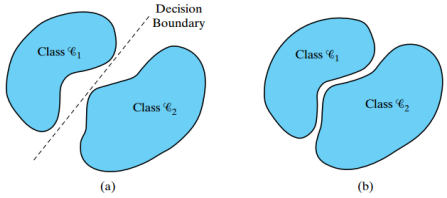



Figure 9: (a) A pair of linearly separable patterns. (b) A pair of non-linearly separable patterns.

case the cost function $J(\mathbf{w})$ is zero. In any event, the nice feature of the cost function $J(\mathbf{w})$ is that it is *differentiable* with respect to the weight vector $\mathbf{w}$. Thus, differentiating $J(\mathbf{w})$ with respect to $\mathbf{w}$ yields the *gradient vector*

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x}(n)\in\mathcal{X}} (-\mathbf{x}(n)d(n))$$

where the *gradient operator*

$$\boldsymbol{\nabla} = \left[\frac{\partial}{\partial \boldsymbol{w}_1}, \frac{\partial}{\partial \boldsymbol{w}_2}, \ldots, \frac{\partial}{\partial \boldsymbol{w}_m}\right]^T$$

In the method of steepest descent, the adjustment to the weight vector w at each timestep of the algorithm is applied in a direction *opposite* to the gradient vector.

*Variables and Parameters:*

$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), ..., x_m(n)]^T$
$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), ..., w_m(n)]^T$
$b$ = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
$\eta$ = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, ....$
2. *Activation.* At time-step $n$, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where sgn$(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$
5. *Continuation.* Increment time step $n$ by one and go back to step 2.

Figure 10: Perceptron Convergence Algorithm

Accordingly, the algorithm takes the form

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\nabla J(\mathbf{w}) \\ &= \mathbf{w}(n) + \eta(n) \sum_{\mathbf{x}(n)\in\mathcal{X}} \mathbf{x}(n)d(n) \end{aligned}$$

which includes the single-sample correction version of the perceptron convergence algorithm as a special case. Moreover, Eq. (1.42) embodies the batch perceptron algorithm for computing the weight vector,given the sample set $\mathbf{x}(1), \mathbf{x}(2), ....$In particular,the adjustment applied to the weight vector at time-step $n+1$ is defined by the sum of all the samples misclassified by the weight vector $\mathbf{w}(n)$, with the sum being scaled by the learning-rate parameter $\eta(n)$.The algorithm is said to be of the "batch" kind because at each time-step of the algorithm, a batch of misclassified samples is used to compute the adjustment.

# Lecture 6

## Multilayer network

Rosenblatt?s perceptron, which is basically a single-layer neural network is limited to the classification of linearly separable patterns. To overcome the practical limitations of the perceptron, we look to a neural network structure known as the *multilayer perceptron.*

The following three points highlight the basic features of multilayer perceptrons:

- The model of each neuron in the network includes a nonlinear activation function that is *differentiable.*
- The network contains one or more layers that are *hidden* from both the input and output nodes.
- The network exhibits a high degree of *connectivity*, the extent of which is determined by synaptic weights of the network.

The learning process of multilayer perceptron is more difficult because the search has to be conducted in a much larger space of possible functions, and a choice has to be made between alternative representations of the input pattern.

A popular method for the training of multilayer perceptrons is the back-propagation algorithm, which includes the LMS algorithm as a special case.The training proceeds in two phases:

1. In the *forward phase*, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output. Thus, in this phase, changes are confined to the activation potentials and outputs of the neurons in the network.
2. In the backward phase, an error signal is produced by comparing the output of the network with a desired response. The resulting error signal is propagated through the network, again layer by layer, but this time the propagation is performed in the backward direction. In this second phase, successive adjustments are made to the synaptic weights of the network.

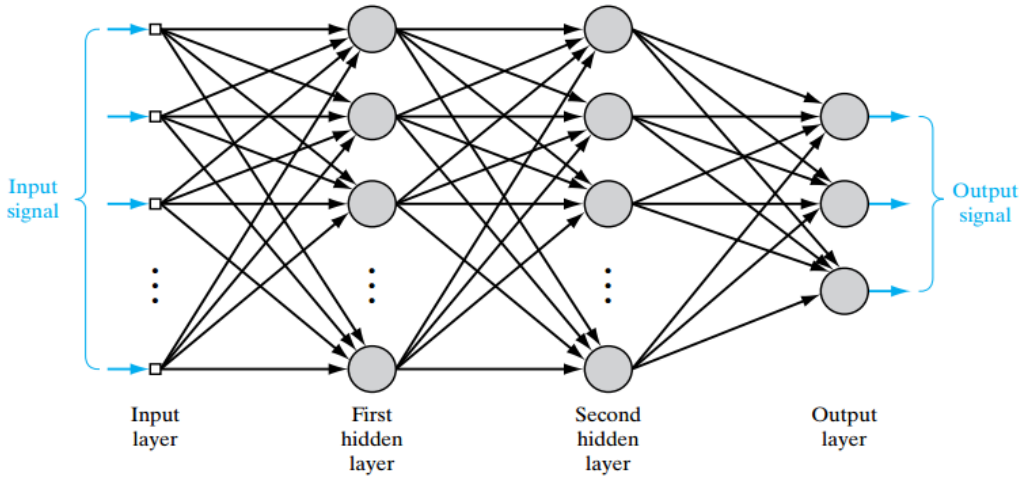



Figure 11: Architectural graph of a multilayer perceptron with two hidden layers

Two kinds of signals are identified in this network:

1. *Function Signals.* A function signal is an input signal (stimulus) that comes in at the input end of the network, propagates forward (neuron by neuron) through the network, and emerges at the output end of the network as an output signal. We refer to such a signal as a "function signal" for two reasons. First, it is presumed to perform a useful function at the output of the network. Second, at each neuron of the network through which a function signal passes, the signal is calculated as a function of the inputs and associated weights applied to that neuron.

2. *Error Signals.* An error signal originates at an output neuron of the network and propagates backward (layer by layer) through the network. We refer to it as an "error signal" because its computation by every neuron of the network involves an error-dependent function in one form or another.

Each hidden or output neuron of a multilayer perceptron is designed to perform two computations:

1. the computation of the function signal appearing at the output of each neuron, which is expressed as a continuous nonlinear function of the input signal and synaptic weights associated with that neuron;

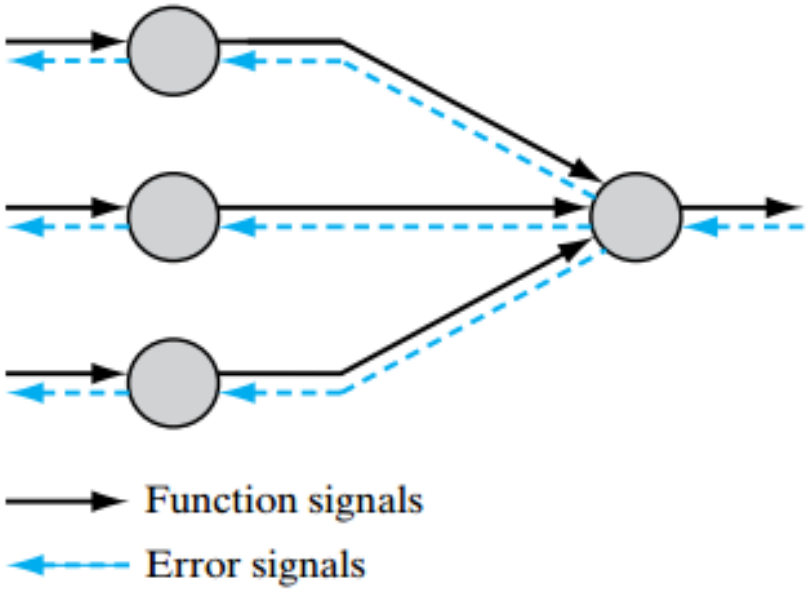



Figure 12: forward propagation of function signals and back propagation of error signals.

2. the computation of an estimate of the gradient vector (i.e., the gradients of the error surface with respect to the weights connected to the inputs of a neuron), which is needed for the backward pass through the network.

## Function of the Hidden Neurons

The hidden neurons act as feature detectors; as such, they play a critical role in the operation of a multilayer perceptron.As the learning process progresses across the multilayer perceptron, the hidden neurons begin to gradually 'discover" the salient features that characterize the training data.They do so by performing a nonlinear transformation on the input data into a new space called the feature space. In this new space, the classes of interest in a pattern-classification task, for example, may be more easily separated from each other than could be the case in the original input data space. Indeed, it is the formation of this feature space through supervised learning that distinguishes the multilayer perceptron from Rosenblatt?s perceptron.

## Batch Learning And On-Line Learning

Consider a multilayer perceptron with an input layer of source nodes, one or more hidden layers, and an output layer consisting of one or more neurons;

denote the training sample used to train the network in a supervised manner. Let $y_j(n)$ denote the function signal produced at the output of neuron $j$ in the output layer by the stimulus $x(n)$ applied to the input layer. Correspondingly, the error signal produced at the output of neuron $j$ is defined by

$$e_j(n) = d_j(n) - y_j(n)$$

where $d_j(n)$ is the ith element of the desired-response vector $\mathbf{d}(n)$. The instantaneous error energy of neuron $j$ is defined by

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

Summing the error-energy contributions of all the neurons in the output layer, we express the *total instantaneous error* energy of the whole network as

$$\begin{aligned}\mathcal{E}(N) &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n) \\ &= \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C}e_j^2(n)\end{aligned}$$

Naturally, the instantaneous error energy, and therefore the average error energy, are both functions of all the adjustable synaptic weights (i.e., free parameters) of the multilayer perceptron.

### Batch Learning

In the batch method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are performed after the presentation of all the N examples in the training sample t that constitute one epoch of training. In other words, the cost function for batch learning is defined by the average error energy eav. Adjustments to the synaptic weights of the multilayer perceptron are made on an *epoch-by-epoch basis.*

# Lecture 7

## Backpropagation

The popularity of on-line learning for the supervised training of multilayer perceptrons has been further enhanced by the development of the back-propagation algorithm. To describe this algorithm, consider Fig[**?**] , which depicts neuron j being fed by a set of

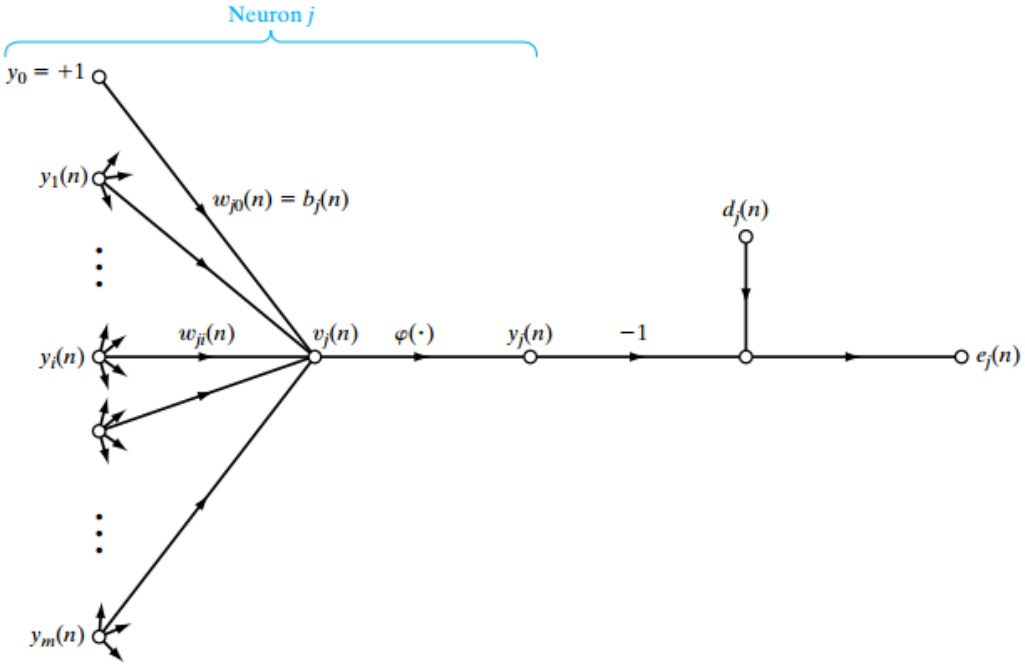



Figure 13: Signal-flow graph highlighting the details of output neuron j.

function signals produced by a layer of neurons to its left. The induced local field $v_j(n)$ produced at the input of the activation function associated with neuron $j$ is therefore

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

where m is the total number of inputs (excluding the bias) applied to neuron j. The synaptic weight $w_{j0}$ (corresponding to the fixed input $eq$ equals the bias bj

applied to neuron j. Hence, the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \phi_j(v_j(n))$$

In a manner similar to the LMS algorithm studied in Chapter 3, the backpropagation algorithm applies a correction wji(n) to the synaptic weight wji(n), which is proportional to the partial derivative According to the chain rule of calculus, we may express this gradient as

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

The partial derivative represents a sensitivity factor, determining the direction of search in weight space for the synaptic weight wji

Differentiating both sides of with respect to ej(n), we get

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n)$$

Differentiating both sides of Eq. with respect to yj(n), we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

Next, differentiating Eq. with respect to vj(n), we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'\left(v_j(n)\right)$$

where the use of prime (on the right-hand side) signifies differentiation with respect to the argument. Finally, differentiating Eq. with respect to wji(n) yields

$$\frac{\partial v_j(n)}{\partial w_{\bar{\mu}}(n)} = y_i(n)$$

The use of Eqs. to in Eq. yields

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'\left(v_j(n)\right) y_i(n)$$

The correction w ji(n) applied to wji(n) is defined by the delta rule, or

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$$

where $\eta$is the learning-rate parameter of the back-propagation algorithm. The use of the minus sign in Eq. accounts for gradient descent in weight space (i.e., seeking a direction for weight change that reduces the value of ). Accordingly, the use of Eq. in Eq. yields

where the local gradient is defined by j(n)

$$\begin{aligned}\delta_j(n) &= \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} \\ &= \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= e_j(n)\varphi_j'\left(v_j(n)\right)\end{aligned}$$

The local gradient points to required changes in synaptic weights.According to Eq. (4.16), the local gradient j(n) for output neuron j is equal to the product of the corresponding error signal ej(n) for that neuron and the derivative (vj(n)) of the associated activation function.

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

we note that a key factor involved in the calculation of the weight adjustment !wji(n) is the error signal ej(n) at the output of neuron j. Even though hidden neurons are not directly accessible,they share responsibility for any error made at the output of the network. The question, however, is to know how to penalize or reward hidden neurons for their share of the responsibility.

### Neuron j Is an Output Node

When neuron j is located in the output layer of the network, it is supplied with a desired response of its own. We may use Eq. to compute the error signal ej(n) associated with this neuron; see Fig.. Having determined ej(n), we find it a straightforward matter to compute the local gradient j(n) by using Eq. .

### Neuron j Is an Hidden Node

When neuron j is located in a hidden layer of the network, there is no specified desired response for that neuron.Accordingly, the error signal for a hidden neuron

would have to be determined recursively and working backwards in terms of the error signals of all the neurons to which that hidden neuron is directly connected; development of the back-propagation algorithm gets complicated. Consider the situation in Fig. which depicts neuron j as a hidden node of the network. According to

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$
$$= -\frac{\partial \delta(n)}{\partial y_j(n)} \varphi_j'(v_j(n)), \qquad \text{neuron } j \text{ is hidden}$$

where in the second line we have used Eq. (4.11). To calculate the partial derivative , we may proceed as follows: $8(n) = \frac{1}{2}\sum_{k=C} e_k^2(n)$, neuron $k$ is an output node

which is Eq. (4.4) with index k used in place of index j. We have made this substitution in order to avoid confusion with the use of index j that refers to a hidden neuron under case 2. Differentiating Eq. with respect to the function signal yj(n), we get

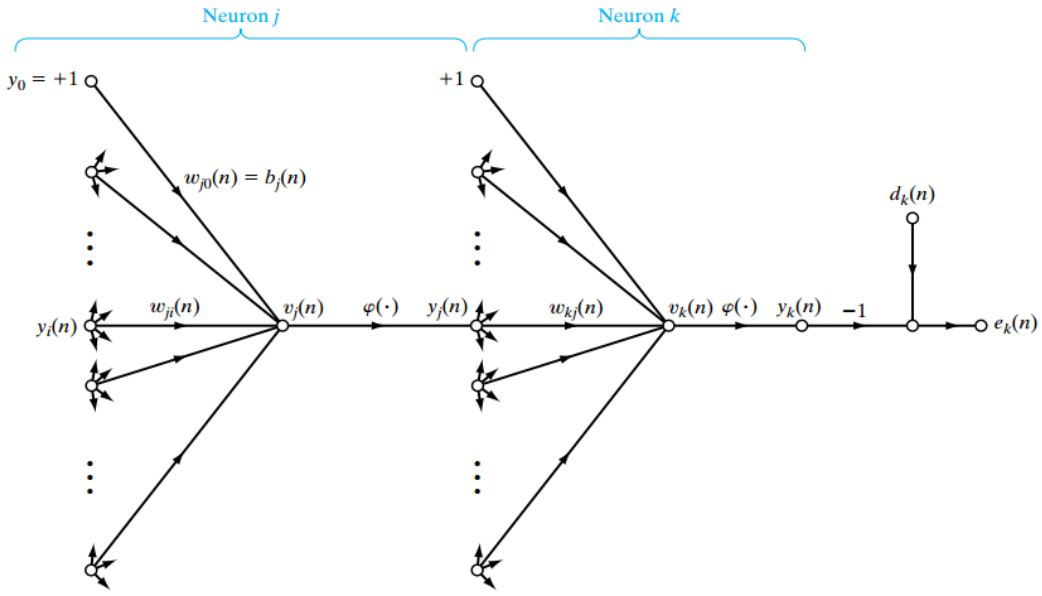



Figure 14: Signal-flow graph highlighting the details of output neuron j.

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}$$

Next we use the chain rule for the partial derivative and rewrite Eq. (4.19) in the equivalent form

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

However, from Fig., we note that $e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k\left(v_k(n)\right)$, neuron $k$ is an output node

where m is the total number of inputs (excluding the bias) applied to neuron k. Here again, the synaptic weight wk0(n) is equal to the bias bk(n) applied to neuron k, and the corresponding input is fixed at the value . Differentiating Eq. (4.23) with respect to yj(n) yields

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= -\sum_k e_k(n) \varphi_k'\left(v_k(n)\right) w_{kj}(n) \\ &= -\sum_k \delta_k(n) w_{kj}(n)\end{aligned}$$

We now summarize the relations that we have derived for the back-propagation algorithm. First, the correction wji(n) applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule:

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{jt}(n) \end{pmatrix} = \begin{pmatrix} \text{learning-} \\ \text{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{local} \\ \text{gradient} \\ 8_j(n) \end{pmatrix} \times \begin{pmatrix} \text{input signal} \\ \text{of neuronj} \\ y_i(n) \end{pmatrix}$$

# Lecture 8

## Introduction Deep Learning

Deep learning (also known as deep structured learning or differential programming) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised

Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analog.

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

Most modern deep learning models are based on artificial neural networks, specifically, Convolutional Neural Networks (CNN)s, although they can also include propositional formulas or latent variables organized layer-wise in deep generative models such as the nodes in deep belief networks and deep Boltzmann machines

In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements

of edges; the third layer may encode a nose and eyes; and the fourth layer may recognize that the image contains a face. Importantly, a deep learning process can learn which features to optimally place in which level on its own. (Of course, this does not completely eliminate the need for hand-tuning; for example, varying numbers of layers and layer sizes can provide different degrees of abstraction.)

The word "deep" in "deep learning" refers to the number of layers through which the data is transformed. More precisely, deep learning systems have a substantial credit assignment path (CAP) depth. The CAP is the chain of transformations from input to output. CAPs describe potentially causal connections between input and output. For a feedforward neural network, the depth of the CAPs is that of the network and is the number of hidden layers plus one (as the output layer is also parameterized). For recurrent neural networks, in which a signal may propagate through a layer more than once, the CAP depth is potentially unlimited. No universally agreed upon threshold of depth divides shallow learning from deep learning, but most researchers agree that deep learning involves CAP depth higher than 2. CAP of depth 2 has been shown to be a universal approximator in the sense that it can emulate any function.[14] Beyond that, more layers do not add to the function approximator ability of the network. Deep models (CAP ¿ 2) are able to extract better features than shallow models and hence, extra layers help in learning the features effectively.

Deep learning architectures can be constructed with a greedy layer-by-layer method. Deep learning helps to disentangle these abstractions and pick out which features improve performance

## Deep neural networks

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. For example, a DNN that is trained to recognize dog breeds will go over the given image and calculate the probability that the dog

in the image is a certain breed. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks.

DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives.The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network.

Deep architectures include many variants of a few basic approaches. Each architecture has found success in specific domains. It is not always possible to compare the performance of multiple architectures, unless they have been evaluated on the same data sets.

DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network did not accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.

## Challenges

As with ANNs, many issues can arise with naively trained DNNs. Two common issues are overfitting and computation time.

DNNs are prone to overfitting because of the added layers of abstraction, which allow them to model rare dependencies in the training data. Regularization methods such as Ivakhnenko's unit pruning can be applied during training to combat overfitting. Alternatively dropout regularization randomly omits units from the hidden layers during training. This helps to exclude rare dependencies. Finally, data can be augmented via methods such as cropping and rotating such that smaller training

sets can be increased in size to reduce the chances of overfitting

DNNs must consider many training parameters, such as the size (number of layers and number of units per layer), the learning rate, and initial weights. Sweeping through the parameter space for optimal parameters may not be feasible due to the cost in time and computational resources. Various tricks, such as batching (computing the gradient on several training examples at once rather than individual examples) speed up computation. Large processing capabilities of many-core architectures (such as GPUs or the Intel Xeon Phi) have produced significant speedups in training, because of the suitability of such processing architectures for the matrix and vector computations

## Application

### Automatic speech recognition

Large-scale automatic speech recognition is the first and most convincing successful case of deep learning. LSTM RNNs can learn "Very Deep Learning" tasks that involve multi-second intervals containing speech events separated by thousands of discrete time steps, where one time step corresponds to about 10 ms. LSTM with forget gates is competitive with traditional speech recognizers on certain tasks

### Image recognition

A common evaluation set for image classification is the MNIST database data set. MNIST is composed of handwritten digits and includes 60,000 training examples and 10,000 test examples. As with TIMIT, its small size lets users test multiple configurations. A comprehensive list of results on this set is available.

Deep learning-based image recognition has become "superhuman", producing more accurate results than human contestants. This first occurred in 2011.

Deep learning-trained vehicles now interpret 360 camera views. Another example is Facial Dysmorphology Novel Analysis (FDNA) used to analyze cases of human malformation connected to a large database of genetic syndromes.

### Natural language processing

Neural networks have been used for implementing language models since the early 2000s.LSTM helped to improve machine translation and language modeling.

Other key techniques in this field are negative sampling and word embedding. Word embedding, such as word2vec, can be thought of as a representational layer in a deep learning architecture that transforms an atomic word into a positional representation of the word relative to other words in the dataset; the position is represented as a point in a vector space. Using word embedding as an RNN input layer allows the network to parse sentences and phrases using an effective compositional vector grammar. A compositional vector grammar can be thought of as probabilistic context free grammar (PCFG) implemented by an RNN. Recursive auto-encoders built atop word embeddings can assess sentence similarity and detect paraphrasing. Deep neural architectures provide the best results for constituency parsing, sentiment analysis, information retrieval, spoken language understanding, machine translation,contextual entity linking, writing style recognition, Text classification and others

### Recommendation systems

Recommendation systems have used deep learning to extract meaningful features for a latent factor model for content-based music and journal recommendations. Multiview deep learning has been applied for learning user preferences from multiple domains. The model uses a hybrid collaborative and content-based approach and enhances recommendations in multiple tasks.

### Mobile advertising

Finding the appropriate mobile audience for mobile advertising is always challenging, since many data points must be considered and analyzed before a target segment can be created and used in ad serving by any ad server. Deep learning has been used to interpret large, many-dimensioned advertising datasets. Many data points are collected during the request/serve/click internet advertising cycle. This information

can form the basis of machine learning to improve ad selection.