CAP 4630
Artificial Intelligence

Instructor: Sam Ganzfried

sganzfri@cis.fiu.edu

Schedule

- 12/5, 12/7: Machine learning (classification, regression, clustering, deep learning(neural networks))
- 12/7: Project presentations and class project due
  - Project code due 12/5 at 12am on Moodle.
- Final exam on 12/14
  - 12pm in GL-139
- Evaluation
  - Log on to MyFIU portal at https://my.fiu.edu.
  - Click on SPOTs.
  - Select the course from the list of SPOTs.
  - Click on the instructor's name.
  - You will now be on the form and can share your perceptions and type comments.

Announcements

- HW4 back next week
- Project final paper due today 12/7: 2-4 page paper in pdf
- Project presentations today: ~2-3 minutes each
  - Ok for just one student to present from a group of 2-3
- Project competition results available next week

Supervised vs. unsupervised learning

- Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way (see inductive bias).
  - Includes classification (e.g., for decision trees) and regression.

Bias-variance tradeoff for supervised learning

- A first issue is the tradeoff between bias and variance. Imagine that we have available several different, but equally good, training data sets. A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x. A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets. The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm. Generally, there is a tradeoff between bias and variance. A learning algorithm with low bias must be "flexible" so that it can fit the data well. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance. A key aspect of many supervised learning methods is that they are able to adjust this tradeoff between bias and variance (either automatically or by providing a bias/variance parameter that the user can adjust).

Bias-variance tradeoff

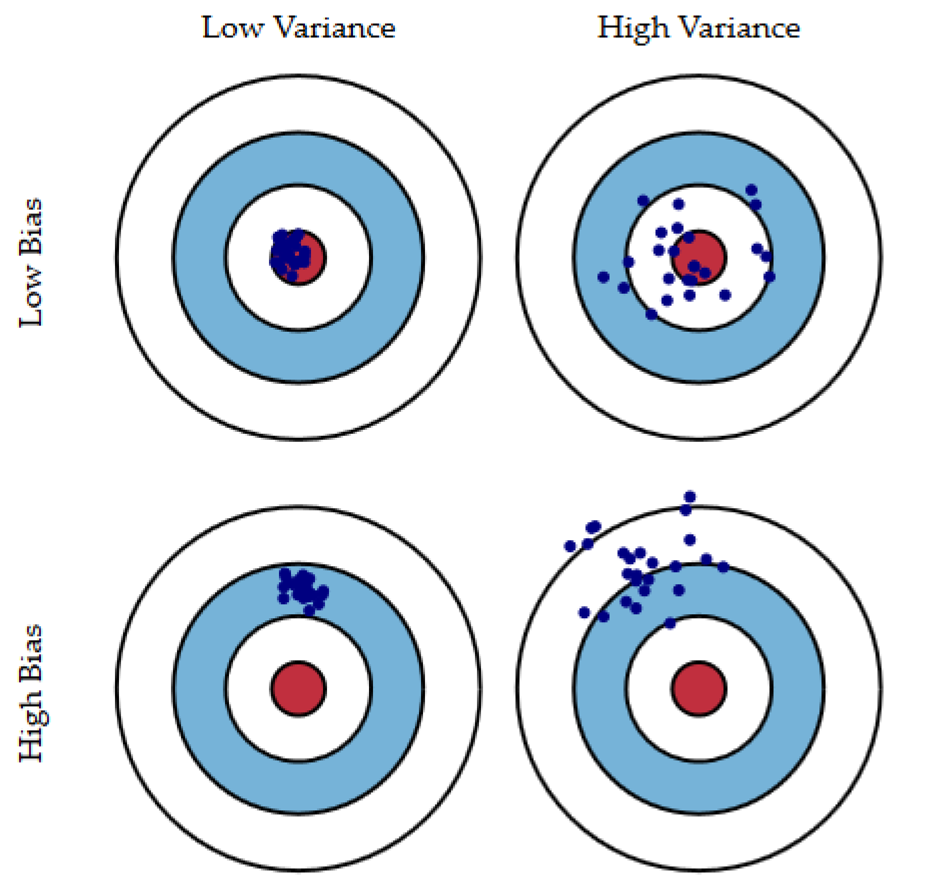


Fig. 1 Graphical illustration of bias and variance.

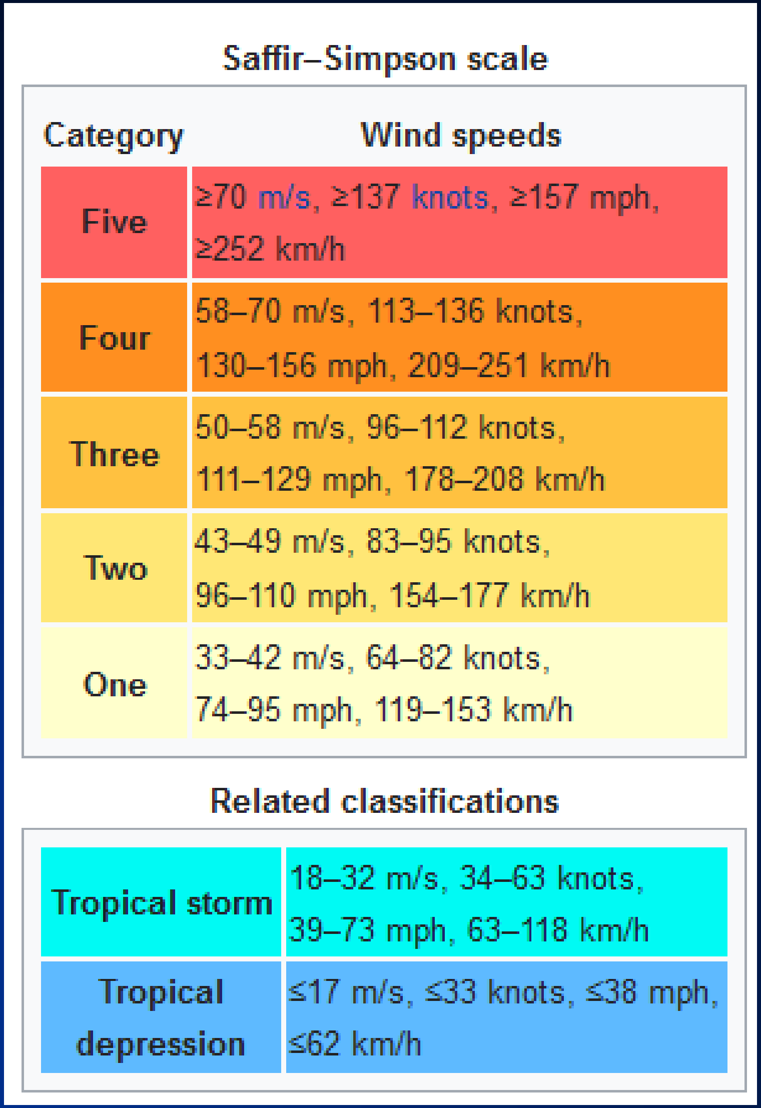
Supervised learning approaches

- We covered decision trees and regression
- Boosting
- Naïve Bayes classifier (e.g., for natural language processing)
- Nearest neighbor algorithm ("k-nn")
- Maximum entropy classifier
- Support vector machines
- Random forests
- Many others. Python has built-in libraries and packages for the main ones.
- Logistic regression
- Neural networks

Unsupervised learning

- Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from "unlabeled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabeled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.
- Approaches to unsupervised learning include:
  - Clustering
  - Anomaly detection
  - Neural networks
  - Approaches for latent variable modeling (e.g., expectation-maximization (EM) algorithm)

Clustering for hurricane classification



Saffir–Simpson scale

Category	Wind speeds
Five	≥70 m/s, ≥137 knots, ≥157 mph, ≥252 km/h
Four	58–70 m/s, 113–136 knots, 130–156 mph, 209–251 km/h
Three	50–58 m/s, 96–112 knots, 111–129 mph, 178–208 km/h
Two	43–49 m/s, 83–95 knots, 96–110 mph, 154–177 km/h
One	33–42 m/s, 64–82 knots, 74–95 mph, 119–153 km/h

Related classifications

Tropical storm	18–32 m/s, 34–63 knots, 39–73 mph, 63–118 km/h
Tropical depression	≤17 m/s, ≤33 knots, ≤38 mph, ≤62 km/h

Hurricane AI

- The Saffir–Simpson hurricane wind scale (SSHWS), formerly the Saffir–Simpson hurricane scale (SSHS), classifies hurricanes – Western Hemisphere tropical cyclones that exceed the intensities of tropical depressions and tropical storms – into five categories distinguished by the intensities of their sustained winds. To be classified as a hurricane, a tropical cyclone must have maximum sustained winds of at least 74 mph (33 m/s; 64 kn; 119 km/h) (Category 1). The highest classification in the scale, Category 5, consists of storms with sustained winds exceeding 156 mph (70 m/s; 136 kn; 251 km/h).
- The classifications can provide some indication of the potential damage and flooding a hurricane will cause upon landfall.
- Officially, the Saffir–Simpson hurricane wind scale is used only to describe hurricanes forming in the Atlantic Ocean and northern Pacific Ocean east of the International Date Line. Other areas use different scales to label these storms, which are called "cyclones" or "typhoons", depending on the area.
- There is some criticism of the SSHS for not taking rain, storm surge, and other important factors into consideration, but SSHS defenders say that part of the goal of SSHS is to be straightforward and simple to understand.

Hurricane AI

- The scale was developed in 1971 by civil engineer Herbert Saffir and meteorologist Robert Simpson, who at the time was director of the U.S. National Hurricane Center (NHC).[1] The scale was introduced to the general public in 1973,[2] and saw widespread use after Neil Frank replaced Simpson at the helm of the NHC in 1974.[3]
- The initial scale was developed by Saffir, a structural engineer, who in 1969 went on commission for the United Nations to study low-cost housing in hurricane-prone areas.[4] While performing the study, Saffir realized there was no simple scale for describing the likely effects of a hurricane. Mirroring the utility of the Richter magnitude scale in describing earthquakes, he devised a 1–5 scale based on wind speed that showed expected damage to structures. Saffir gave the scale to the NHC, and Simpson added the effects of storm surge and flooding.

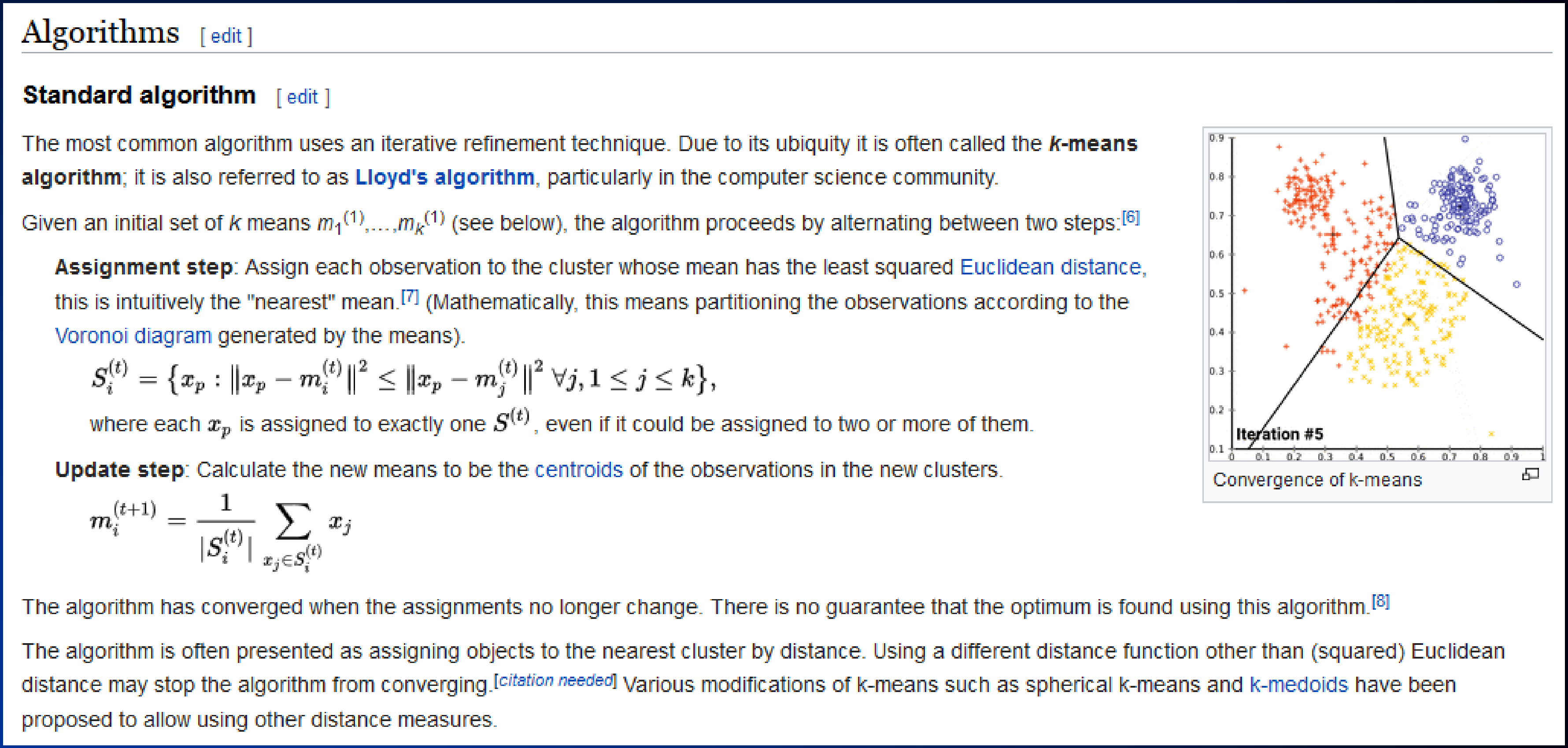
AI clustering

- **Cluster analysis** or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

Clustering

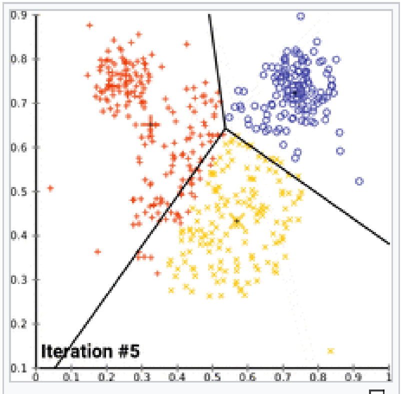
- Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It is often necessary to modify data preprocessing and model parameters until the result achieves the desired properties.

K-means clustering algorithm



Algorithms [edit]

Standard algorithm [edit]



Convergence of k-means

The most common algorithm uses an iterative refinement technique. Due to its ubiquity it is often called the ***k*-means algorithm**; it is also referred to as **Lloyd's algorithm**, particularly in the computer science community.

Given an initial set of k means $m_1^{(1)},\ldots,m_k^{(1)}$ (see below), the algorithm proceeds by alternating between two steps:[6]

Assignment step: Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean.[7] (Mathematically, this means partitioning the observations according to the Voronoi diagram generated by the means).

$$S_i^{(t)} = \big\{ x_p : \big\| x_p - m_i^{(t)} \big\|^2 \le \big\| x_p - m_j^{(t)} \big\|^2 \ \forall j, 1 \le j \le k \big\},$$

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

Update step: Calculate the new means to be the centroids of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

The algorithm has converged when the assignments no longer change. There is no guarantee that the optimum is found using this algorithm.[8]

The algorithm is often presented as assigning objects to the nearest cluster by distance. Using a different distance function other than (squared) Euclidean distance may stop the algorithm from converging.[citation needed] Various modifications of k-means such as spherical k-means and k-medoids have been proposed to allow using other distance measures.

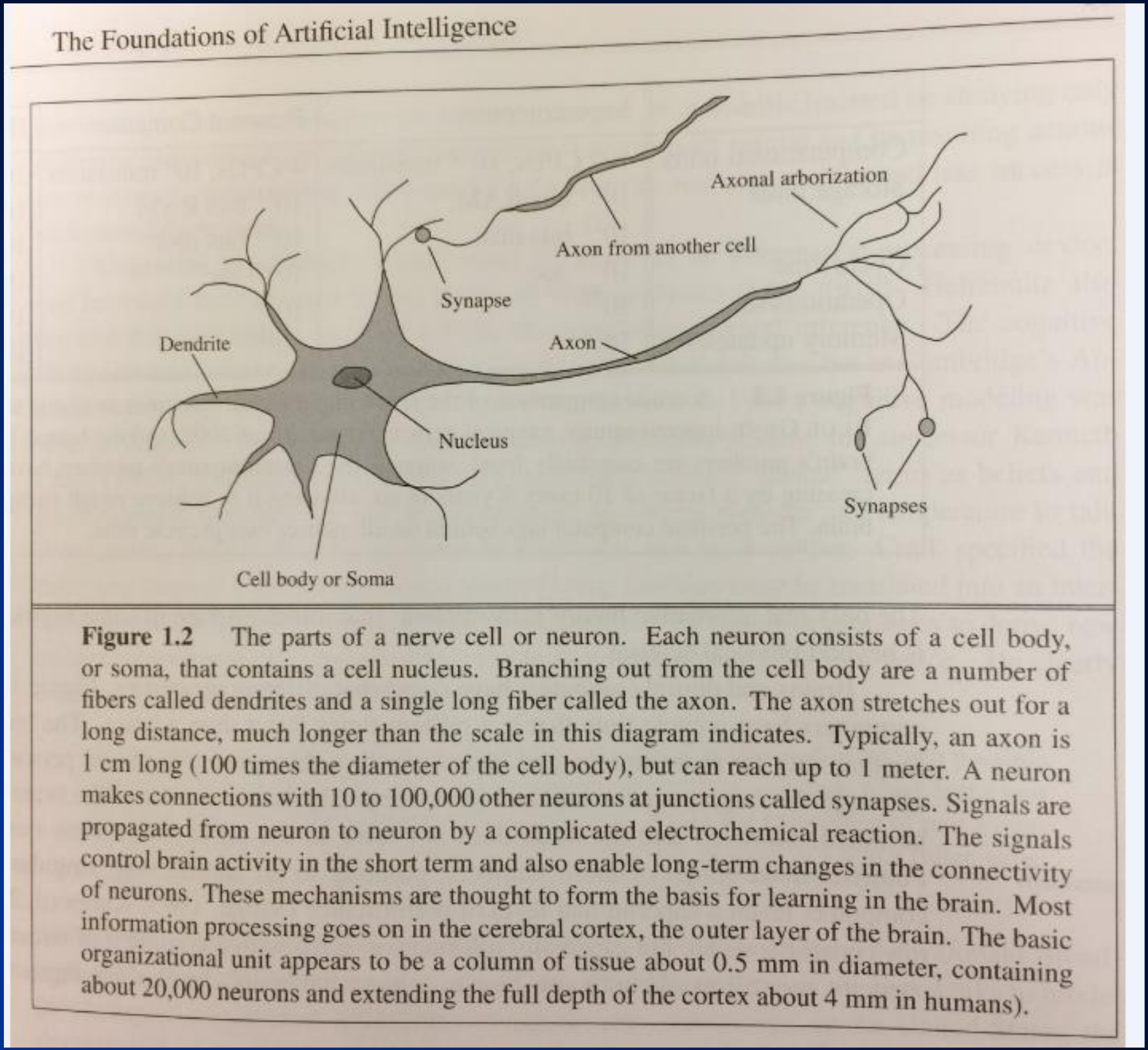
Other clustering approaches

- K-medoids
- K-medians
- Different initialization methods, e.g., kmeans++
  - Typically initial cluster means are chosen at random

Artificial neural networks

- We now turn to what seems to be a somewhat unrelated topic: the brain. In fact, as we will see, the technical ideas we have discussed so far in this chapter turn out to be useful in building mathematical models of the brain’s activity; conversely, thinking about the brain has helped in extending the scope of the technical ideas.

Neuron



The Foundations of Artificial Intelligence

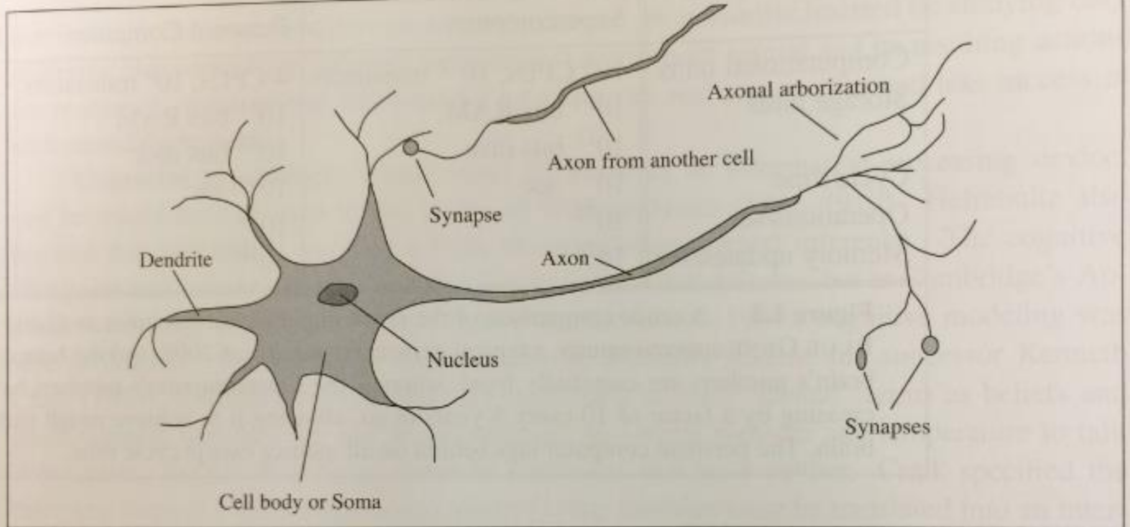


Figure 1.2 The parts of a nerve cell or neuron. Each neuron consists of a cell body, or soma, that contains a cell nucleus. Branching out from the cell body are a number of fibers called dendrites and a single long fiber called the axon. The axon stretches out for a long distance, much longer than the scale in this diagram indicates. Typically, an axon is 1 cm long (100 times the diameter of the cell body), but can reach up to 1 meter. A neuron makes connections with 10 to 100,000 other neurons at junctions called synapses. Signals are propagated from neuron to neuron by a complicated electrochemical reaction. The signals control brain activity in the short term and also enable long-term changes in the connectivity of neurons. These mechanisms are thought to form the basis for learning in the brain. Most information processing goes on in the cerebral cortex, the outer layer of the brain. The basic organizational unit appears to be a column of tissue about 0.5 mm in diameter, containing about 20,000 neurons and extending the full depth of the cortex about 4 mm in humans).

Artificial neural networks

- One of the basic findings of neuroscience is the hypothesis that mental activity consists primarily of electrochemical activity in networks of brain cells called **neurons**. Inspired by this hypothesis, some of the earliest AI work aimed to create artificial **neural networks**. Figure 18.19 shows a simple mathematical model of the neuron devised by McCulloch and Pitts (1943). Roughly speaking, it “fires” when a linear combination of its inputs exceeds some (hard or soft) threshold—that is, implements a linear classifier of the kind described previously. A neural network is just a collection of units connected together; the properties of the network are determined by its topology and the properties of the “neurons.”

Linear classification

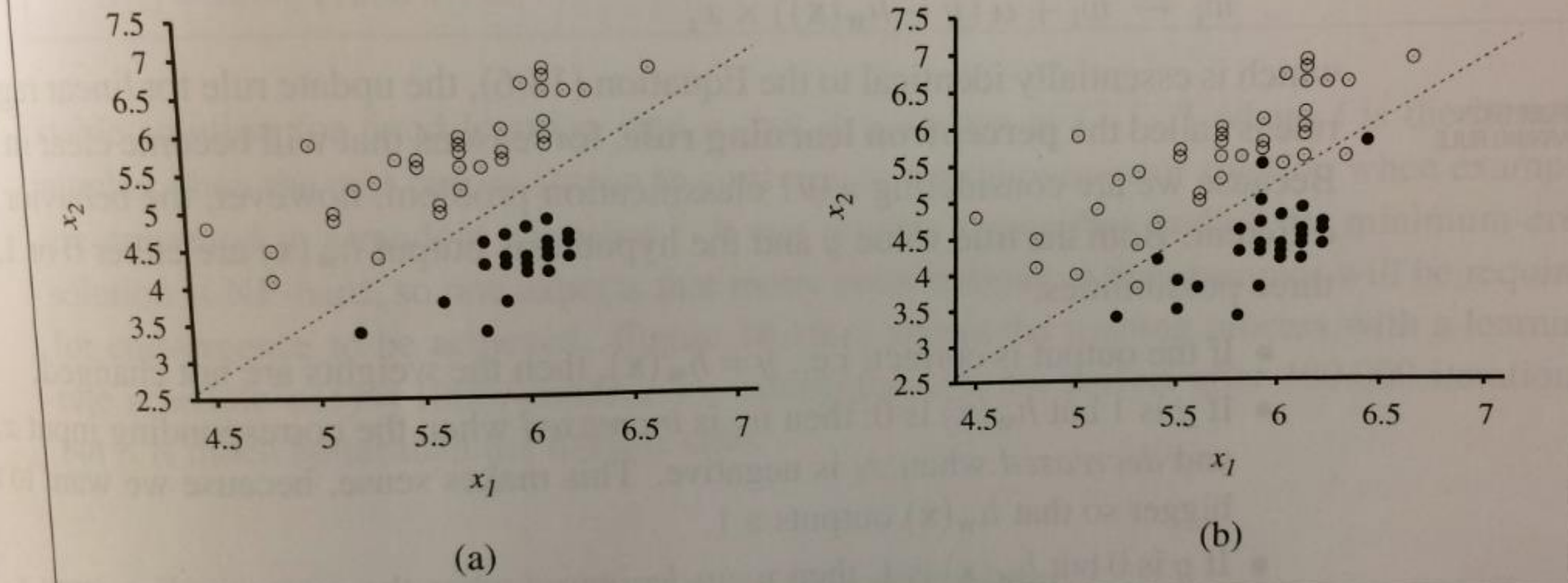


Figure 18.15 (a) Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998). Also shown is a decision boundary between the classes. (b) The same domain with more data points. The earthquakes and explosions are no longer linearly separable.

Artificial neural networks

- Since 1943, much more detailed and realistic models have been developed, both for neurons and for larger systems in the brain, leading to the modern field of **computational neuroscience**. On the other hand, researchers in AI and statistics became interested in the more abstract properties of neural networks, such as their ability to perform distributed computation, to tolerate noisy inputs, and to learn. Although we understand now that other kinds of systems—including Bayesian networks–have these properties, neural networks remain one of the most popular and effective forms of learning system and are worthy of study in their own right.

Artificial neural networks

- Neural networks are composed of **units** connected by directed **links**. A link from unit i to unit j serves to propagate the **activation** a_i from i to j. Each link also has a numeric **weight** $w_{i,j}$ associated with it, which determines the strength and sign of the connection. Just as in linear regression models, each unit has a dummy input $a_0=1$ with an associated weight $w_{0,j}$. Each unit j first computes a weighted sum of its inputs:

  $in_j = \sum^n_{i=0} w_{i,j}\ a_i$

- Then it applies an **activation function** g to this sum to derive the output:

  $A_j = g(in_j) = g(\sum^n_{i=0} w_{i,j}\ a_i)$

Mathematical model of neuron

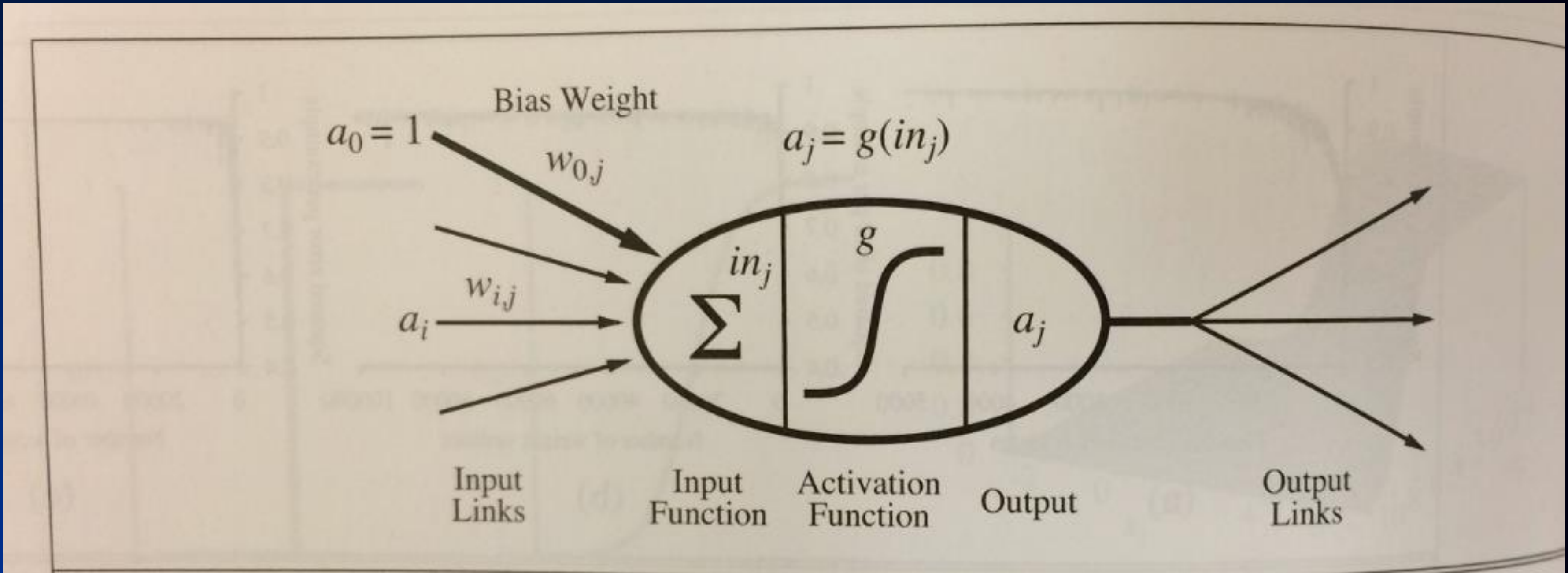


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

Artificial neural networks

- The activation function g is typically either a hard threshold (Figure 18.17a), in which case the unit is called a **perceptron**, or a logistic function (Figure 18.17b), in which case the term **sigmoid perceptron** is sometimes used. Both of these nonlinear activation functions ensure the important property that the entire network of units can represent a nonlinear function. The logistic activation function has the added advantage of being differentiable.

Threshold vs. logistic function

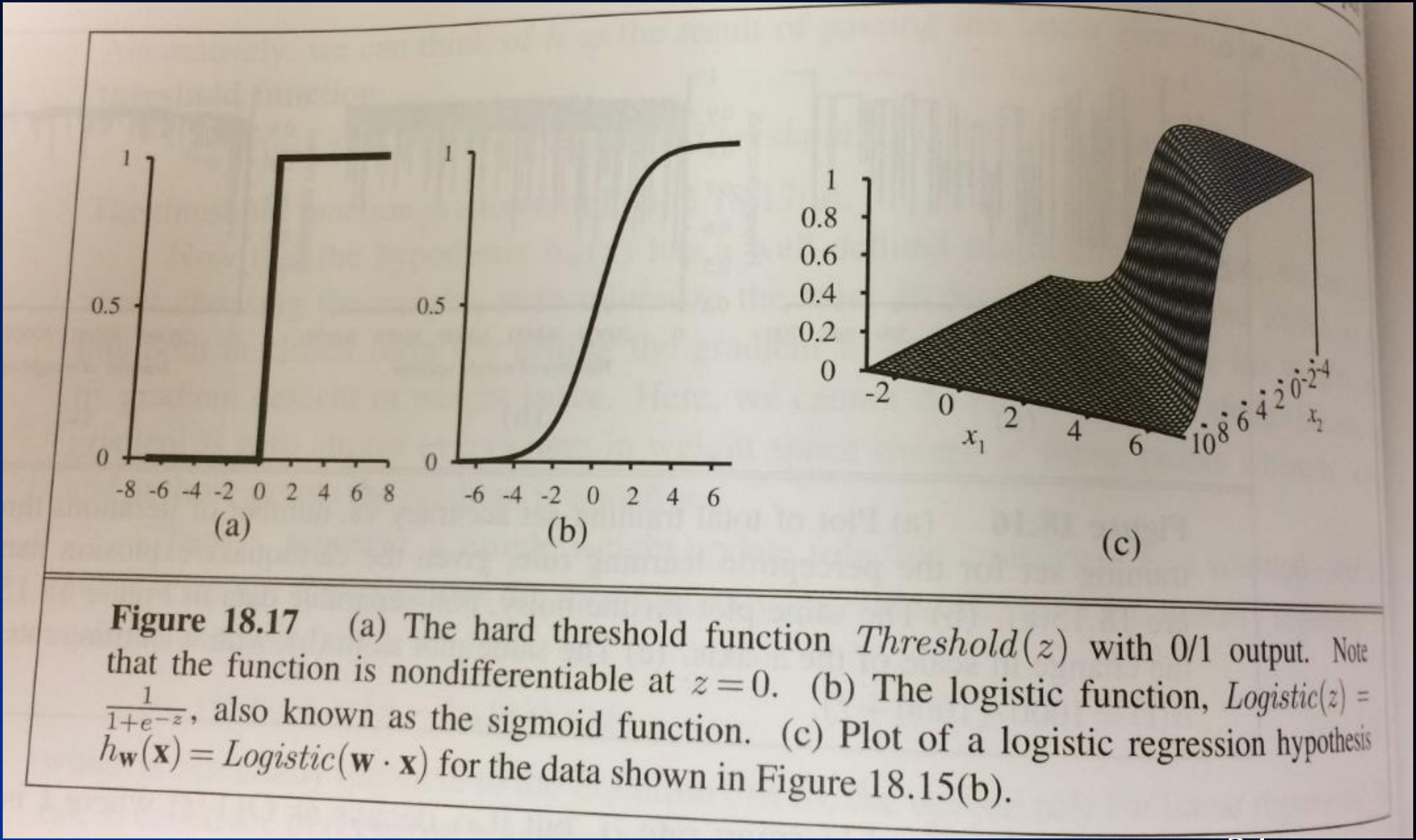


Figure 18.17 (a) The hard threshold function *Threshold*(z) with 0/1 output. Note that the function is nondifferentiable at $z = 0$. (b) The logistic function, *Logistic*(z) = $\frac{1}{1+e^{-z}}$, also known as the sigmoid function. (c) Plot of a logistic regression hypothesis $h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})$ for the data shown in Figure 18.15(b).

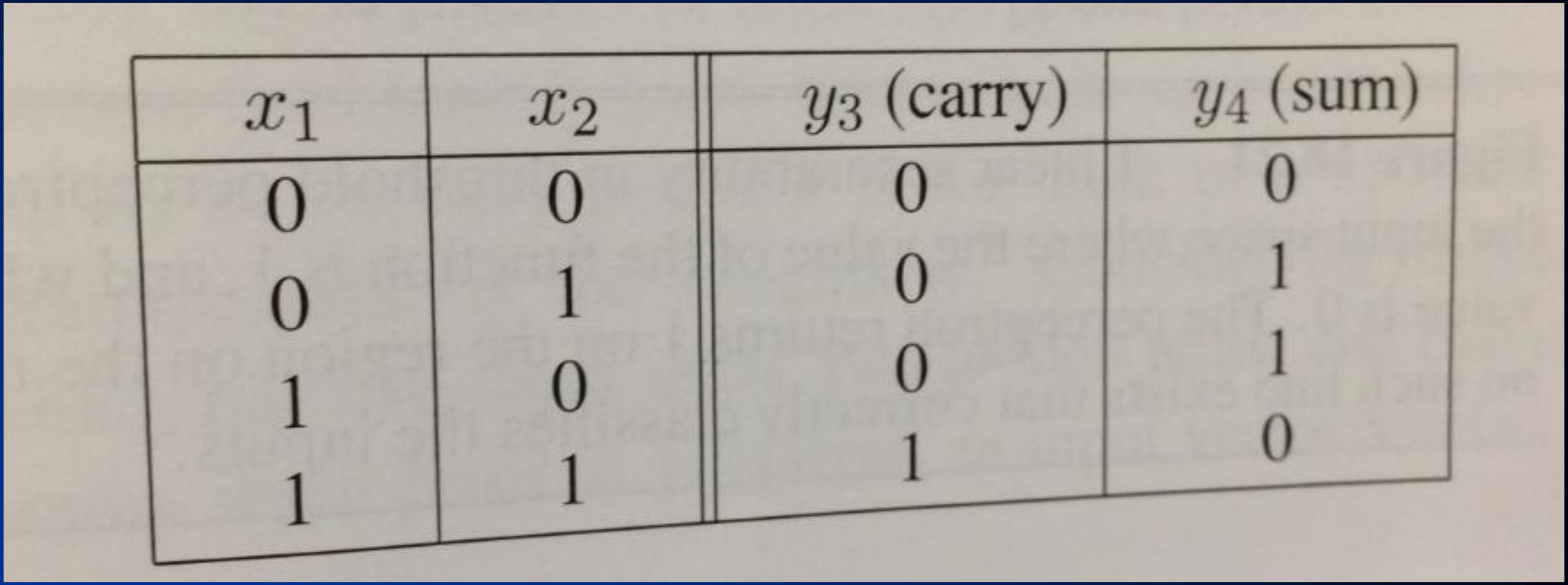
Artificial neural networks

- Having decided on the mathematical model for individual "neurons," the next task is to connect them together to form a network. There are two fundamentally distinct ways to do this. A **feed-forward network** has connections only in one direction—that is, it forms a directed acyclic graph. Every node receives input from "upstream" nodes and delivers output to "downstream" nodes; there are no loops. A feed-forward network represents a function of its current input; thus, it has no internal state other than the weights themselves. A **recurrent network**, on the other hand, feeds its outputs back into its own inputs. This means that the activation levels of the network form a dynamical system that may reach a stable state or exhibit oscillations or even chaotic behavior. Moreover, the response of the network to a given input depends on its initial state, which may depend on previous inputs. Hence, recurrent networks (unlike feed-forward networks) can support short-term memory. This makes them more interesting as models of the brain, but also more difficult to understand.

Artificial neural networks

- Feed-forward networks are usually arranged in **layers**, such that each unit receives input only from units in the immediately preceding layer. In the next two subsections, we will look at single-layer networks, in which every unit connects directly from the network’s inputs to its outputs, and multilayer networks, which have one or more layers of **hidden units** that are not connected to the outputs of the network. So far, we have considered only learning problems with a single output variable y, but neural networks are often used in cases where multiple outputs are appropriate. For example, if we want to train a network to add two input bits, each a 0 or a 1, we will need one output for the sum bit and one for the carry bit. Also, when the learning problem involves classification into more than two classes—for example, when learning to categorize images of handwritten digits—it is common to use one output unit for each class.

Artificial neural networks



x_1	x_2	y_3 (carry)	y_4 (sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Artificial neural networks

- A network with all the inputs connected directly to the outputs is called a **single-layer neural network**, or a **perceptron network**. Figure 18.20 shows a simple two-input, two-output perceptron network. With such a network, we might hope to learn the two-bit adder function, for example. Here are all the training data we will need:

Perceptron network

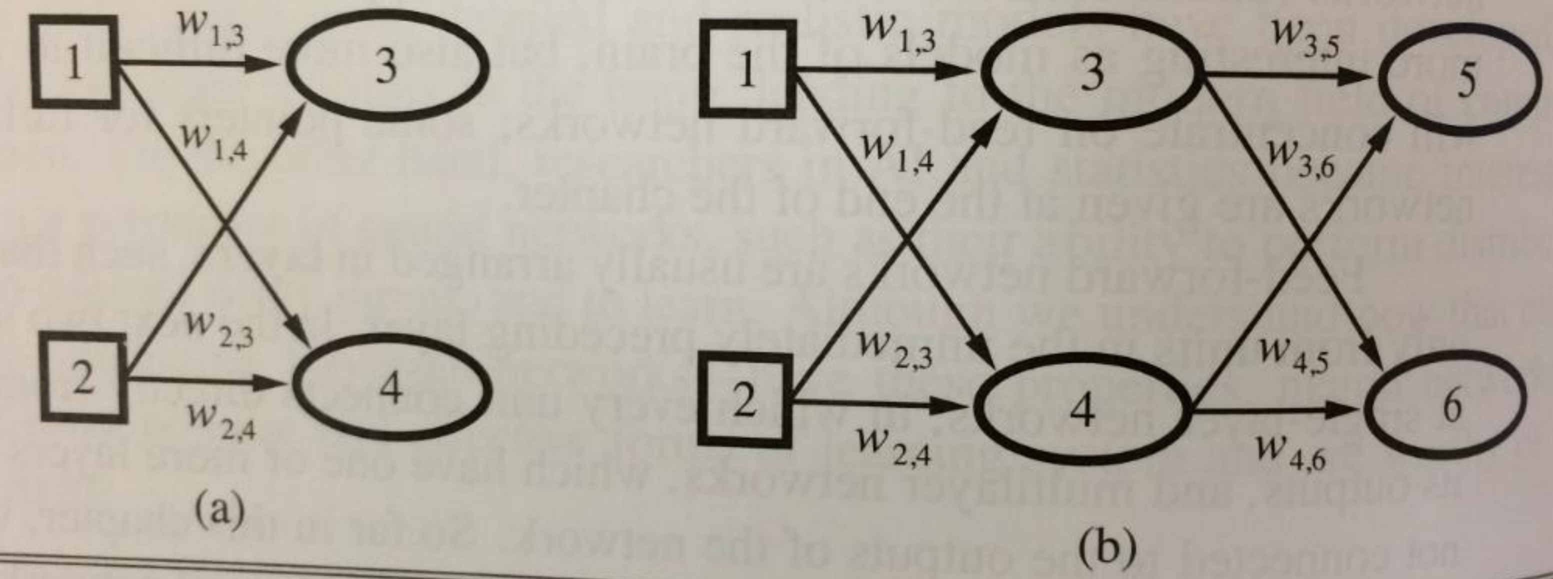


Figure 18.20 (a) A perceptron network with two inputs and two output units. (b) A neural network with two inputs, one hidden layer of two units, and two output units. Not shown are the dummy inputs and their associated weights.

Artificial neural networks

- The first thing to notice is that a perceptron network with m outputs is really m separate networks, because each weight affects only one of the outputs. Thus, there will be m separate training processes. Furthermore, depending on the type of activation function used, the training processes will be either the **perceptron learning rule** or gradient descent rule for the **logistic regression**.

Artificial neural networks

- If you try either method on the two-bit-adder data, something interesting happens. Unit 3 learns the carry function easily, but unit 4 completely fails to learn the sum function. No unit 4 is not defective! The problem is with the sum function itself. We saw that linear classifiers (whether hard or soft) can represent linear decision boundaries in the input space. This works fine for the carry function, which is a logical AND. The sum function, however, is an XOR (exclusive OR) of the two inputs. As Figure 18.21© illustrates, this function is not linearly separable so the perceptron cannot learn it.

Linear separability

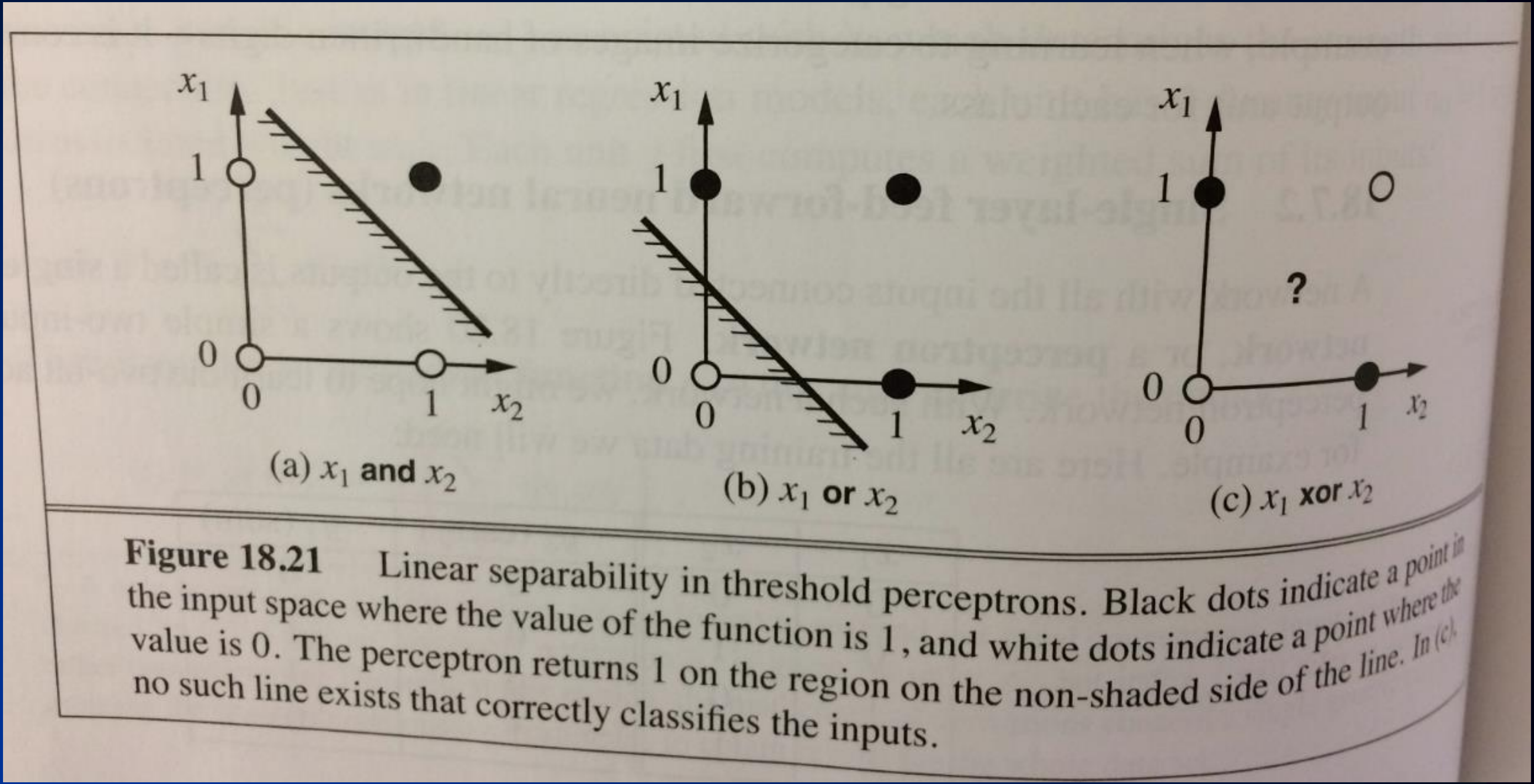


Figure 18.21 Linear separability in threshold perceptrons. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The perceptron returns 1 on the region on the non-shaded side of the line. In (c), no such line exists that correctly classifies the inputs.

Artificial neural networks

- The linearly separable functions constitute just a small fraction of all Boolean functions. The inability of perceptrons to learn even such simple functions as XOR was a significant setback to the nascent neural network community in the 1960s. Perceptrons are far from useless, however. Section 18.6.4 noted that logistic regression (i.e., training a sigmoid perceptron) is even today a very popular and effective tool. Moreover, a perceptron can represent some quite "complex" Boolean functions very compactly. For example, the **majority function**, which outputs a 1 only if more than half of its n inputs are 1, can be represented by a perceptron with each $w_i = 1$ and with $w_0 = -n/2$. A decision tree would need exponentially many nodes to represent this function.

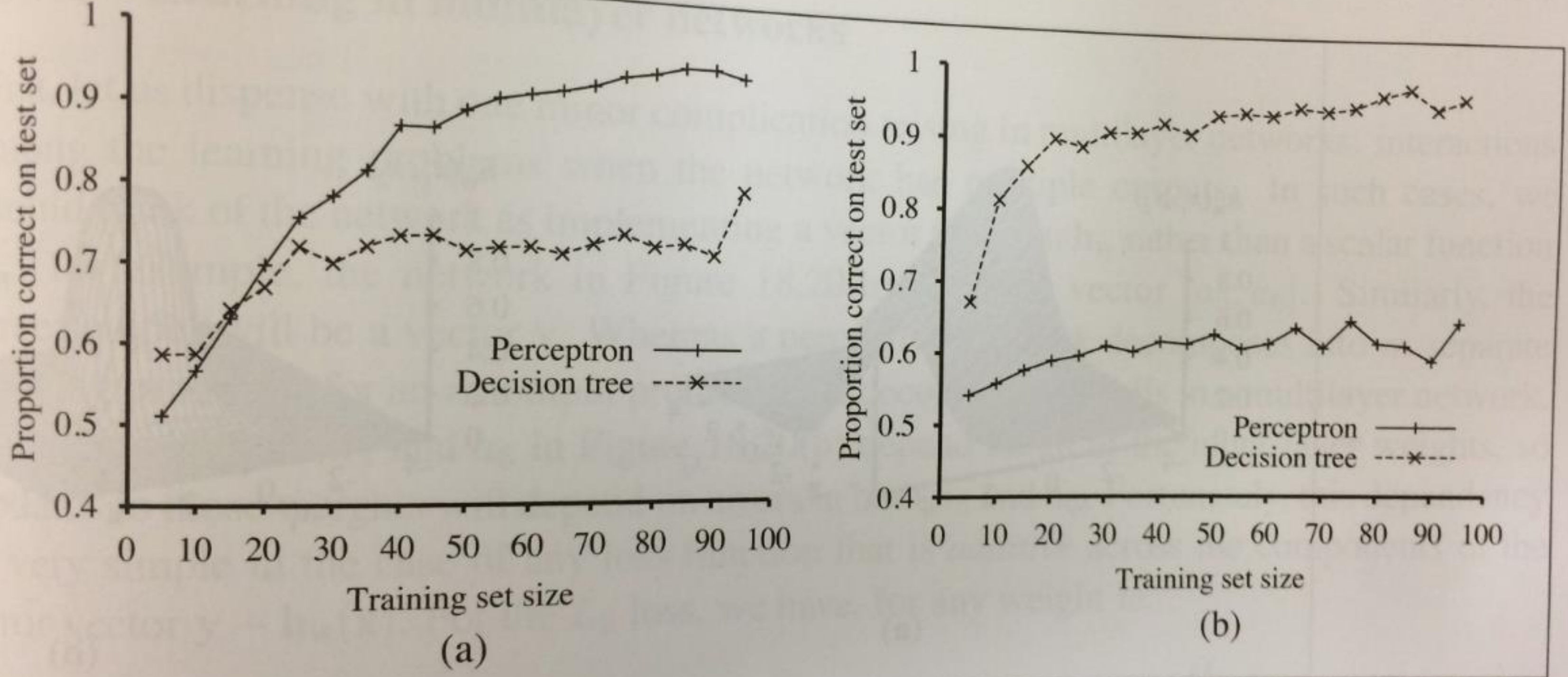


Figure 18.22 Comparing the performance of perceptrons and decision trees. (a) Perceptrons are better at learning the majority function of 11 inputs. (b) Decision trees are better at learning the *WillWait* predicate in the restaurant example.

Artificial neural networks

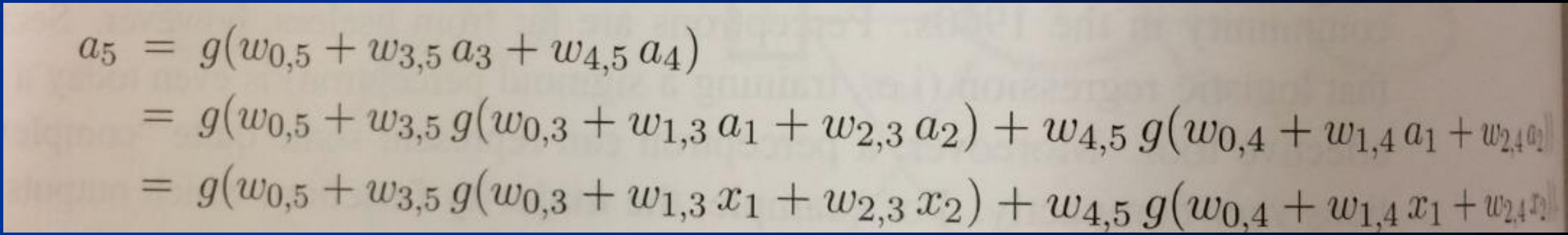
- Figure 18.22 shows the learning curve for a perceptron on two different problems. On the left, we show the curve for learning the majority function with 11 Boolean inputs (i.e., the function outputs a 1 if 6 or more inputs are 1). As we would expect, the perceptron learns the function quite quickly, because the majority function is linearly separable. On the other hand, the decision-tree learner makes no progress, because the majority function is very hard (although not impossible) to represent as a decision tree. On the right, we have the restaurant example. The solution problem is easily represented as a decision tree, but is not linearly separable. The best plane through the data correctly classifies only 65%.

Artificial neural networks

- (McCulloch and Pitts, 1943) were well aware that a single threshold unit would not solve all their problems. In fact, their paper proves that such a unit can represent the basic Boolean functions AND, OR, and NOT and then goes on to argue that any desired functionality can be obtained by connecting large numbers of units into (possibly recurrent) networks of arbitrary depth. The problem was that nobody knew how to train such networks.

Artificial neural networks

- This turns out to be an easy problem if we think of a network the right away: as a function $h_w(x)$ parameterized by the weights w. Consider the simple network shown in 18.20(b), which has two input units, two hidden units, and two outputs. (In addition, each unit has a dummy input fixed at 1). Given an input vector x = (x1,x2), the activations of input units are set to (a1, a2) = (x1, x2). The output at unit 5 is given by



$$\begin{aligned} a_5 &= g(w_{0,5} + w_{3,5}\, a_3 + w_{4,5}\, a_4) \\ &= g(w_{0,5} + w_{3,5}\, g(w_{0,3} + w_{1,3}\, a_1 + w_{2,3}\, a_2) + w_{4,5}\, g(w_{0,4} + w_{1,4}\, a_1 + w_{2,4}\, a_2 \\ &= g(w_{0,5} + w_{3,5}\, g(w_{0,3} + w_{1,3}\, x_1 + w_{2,3}\, x_2) + w_{4,5}\, g(w_{0,4} + w_{1,4}\, x_1 + w_{2,4}\, x_2 \end{aligned}$$

Artificial neural networks

- Thus, we have the output expressed as a function of the inputs and the weights. A similar expression holds for unit 6. As long as we can calculate the derivatives of such expressions with respect to the weights, we can use the gradient-descent loss-minimization method to train the network.

Homework for next class

- Exam on 12/14 at 12pm in GL-139