**NEURAL NETWORKS**

Nowadays, there is a recent surge of interest in neural networks, which are based on continuous-space representation of the input and non-linear functions. Neural networks are capable of modeling complex patterns in data.

*A simple neuron*

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

*A more complicated neuron*

The previous neuron doesn't do anything that conventional computers don't do already. A more sophisticated neuron (figure 2.1) is the McCulloch and Pitts model (MCP). The difference from the previous model is that the inputs are «weighted», the effect that each input has at decision making is dependent on the weight of the particular input. The weight of an input is a number which when multiplied with the input gives the weighted input. These weighted inputs are then added together and if they exceed a pre-set threshold value, the neuron fires. In any other case the neuron does not fire.

The addition of input weights and of the threshold makes this neuron a very flexible and powerful one. The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold. Various algorithms exist that cause the neuron to «adapt»; the most used ones are the «Delta rule» and the back error propagation. The former is used in feed-forward networks and the latter in feedback networks.

*Pattern recognition*

An important application of neural networks is pattern recognition. Pattern recognition can be implemented by using a feed-forward neural network that has been trained accordingly. During training, the network is trained to associate outputs with input patterns. When the network is used, it identifies the input pattern and tries to output the associated output pattern. The power of neural networks comes to life when a pattern that has no output associated with it, is given as an input. In this case, the network gives the output that corresponds to a taught input pattern that is least different from the given pattern.

**POLAR COORDINATE SYSTEM**

In mathematics, the polar coordinate system is a two-dimensional coordinate system in which each point on a plane is determined by an angle and a distance. The polar coordinate system is especially useful in situations where the relationship between two points is most easily expressed with angles and distance; in the more familiar Cartesian or rectangular coordinate system, such a relationship can only be found through trigonometric formulae.

As the coordinate system is two-dimensional, each point is determined by two polar coordinates: the radial coordinate and the angular coordinate. The radial coordinate denotes the point's distance from a central point known as the pole (equivalent to the origin in the Cartesian system). The angular coordinate (also known as the polar angle or the azimuth angle, and usually denoted by 0 or denotes the positive or anticlockwise (counterclockwise) angle required to reach the point from the 0° ray or polar axis (which is equivalent to the positive x-axis in the Cartesian coordinate plane).

The concepts of angle and radius were already used by ancient peoples of the 1st millennium BCE. Hipparchus (190-120 BCE) created a table of chord functions giving the length of the chord for each angle, and there are references to his using polar coordinates in establishing stellar positions. In On Spirals, Archimedes describes the Archimedean spiral, a function whose radius depends on the angle. The Greek work, however, did not extend to a full coordinate system.

There are various accounts of the introduction of polar coordinates as part of a formal coordinate system. Gregoire de Saint-Vincent and Bonaventura Cavalieri independently introduced the concepts in the mid-seventeenth century. Saint-Vincent wrote about them privately in 1625 and published his work in 1647, while Cavalieri published his in 1635 with a corrected version appearing in 1653. Cavalieri first used polar coordinates to solve a problem relating to the area within an Archimedean spiral. Blaise Pascal subsequently used polar coordinates to calculate the length of parabolic arcs.

The actual term polar coordinates has been attributed to Gregorio Fontana and was used by 18th-century Italian writers. The term appeared in English in George Peacock's 1816 translation of Lacroix's Differential and Integral Calculus. Alexis Clairaut was the first to think of polar coordinates in three dimensions, and Leonhard Euler was the first to actually develop them.

**Regression Analysis**

Closely related to but conceptually very much different from regression analysis is correlation analysis, where the primary objective is to measure the strength or degree of linear association between two variables. The correlation coefficient measures this strength of linear association. For example, we may be interested in finding the correlation coefficient between marks of mathematics and statistics examination, between smoking and lung cancer and so on. In regression analysis, we are not primarily interested in such a measure. Instead, we try to estimate or predict the average value of one variable on the basis of the fixed values of other variables. Thus, we want to know whether predict the average mark on a mathematics examination by knowing a student’s marks on a statistics examination.

In regression analysis, there is an asymmetry in the way the dependent and explanatory variables are treated. The dependent variable is assumed to be statistical, random or stochastic, that is, to have a probability distribution. On the other hand, the explanatory variables are assumed to have fixed values (in repeated sampling). But in correlation analysis we treat any two variables symmetrically; there is no distinction between the dependent and explanatory variables. The correlation between marks of mathematics and statistics examinations is the same as that between marks of statistics and mathematics examinations. Moreover, both variables are assumed to be random. While most of the correlation theory is based on the assumption of randomness of variables, whereas most of the regression theory is based on the assumption that the dependent variable is stochastic but explanatory variables are fixed or non-stochastic.

Thus, regression analysis is largely concerned with estimating and/or predicting the population mean or average value of the dependant variable on the basis of the known value of the explanatory variable. We start by a simple linear regression model, that is, by the relationship between two variables, one dependent and one explanatory, related with a linear function. If we are studying the dependence of a variable on only a single explanatory variable, such as consumption expenditure on income, such study is known as the simple or two-variable regression analysis.

**Fermat’s Last Theorem**

Few mathematicians today believe that Fermat had a valid proof of his “Theorem,” which is called his Last Theorem because it was the last of his assertions that remained unproved. The history of Fermat’s Last Theorem is fascinating, with literally hundreds of mathematicians making important contributions. Even a brief summary could easily fill a book. This is not our intent in this volume, so we will be content with a few brief remarks.

One of the first general results on Fermat’s Last Theorem, as opposed to verification for specific exponents n, was given by Sophie Germain in 1823. She proved that if both p and 2p + 1 are primes then the equation ap + bp = cp has no solutions in integers a; b; c with p not dividing the product abc. A later result of a similar nature, due to A. Wieferich in 1909, is that the same conclusion is true if the quantity 2p − 2 is not divisible by p2. Meanwhile, during the latter part of the nineteenth century a number of mathematicians, including Richard Dedekind, Leopold Kronecker, and especially Ernst Kummer, developed a new field of mathematics called algebraic number theory and used their theory to prove Fermat’s Last Theorem for many exponents, although still only a finite list. Then, in 1985, L.M. Adleman, D.R. Heath-Brown, and E. Fouvry used a refinement of Germain’s criterion together with difficult analytic estimates to prove that there are infinitely many primes p such that ap + bp = cp has no solutions with p not dividing abc.

In 1986 Gerhard Frey suggested a new line of attack on Fermat’s problem using a notion called modularity. Frey’s idea was refined by Jean-Pierre Serre, and Ken Ribet subsequently proved that if the Modularity Conjecture is true, then Fermat’s Last Theorem is true. Precisely, Ribet proved that if every semistable elliptic curve is modular then Fermat’s Last Theorem is true. The Modularity Conjecture, which asserts that every rational elliptic curve is modular, was at that time a conjecture originally formulated by Goro Shimura and Yutaka Taniyama. Finally, in 1994,

Andrew Wiles announced a proof that every semistable rational elliptic curve is modular, thereby completing the proof of Fermat’s 350-year-old claim.

Few mathematical or scientific discoveries arise in a vacuum. Even Sir Isaac

Newton, the transcendent genius not noted for his modesty, wrote that “If I have seen further, it is by standing on the shoulders of giants.” Here is a list of some of the giants, all contemporary mathematicians, whose work either directly or indirectly contributed to Wiles’s brilliant proof.

**Number theory and algebra**

Number theory, or in older usage arithmetic, is a branch of pure mathematics devoted primarily to the study of the integers. It is sometimes called "The Queen of Mathematics" because of its foundational place in the discipline. Number theorists study prime numbers as well as the properties of objects made out of integers (e.g., rational numbers) or defined as generalizations of the integers (e.g., algebraic integers).

Number theory and algebra play an increasingly significant role in computing and communications, as evidenced by the striking applications of these subjects to such fields as cryptography and coding theory. My goal in writing this book was to provide an introduction to number theory and algebra, with an emphasis on algorithms and applications, that would be accessible to a broad audience. In particular, I wanted to write a book that would be appropriate for typical students in computer science or mathematics who have some amount of general mathematical experience, but without presuming too much specific mathematical knowledge.

The mathematical prerequisites are minimal: no particular mathematical concepts beyond what is taught in a typical undergraduate calculus sequence are assumed.

The computer science prerequisites are also quite minimal: it is assumed that the reader is proficient in programming, and has had some exposure to the analysis of algorithms, essentially at the level of an undergraduate course on algorithms and data structures.

Even though it is mathematically quite self-contained, the text does presuppose that the reader is comfortable with mathematical formalism and also has some experience in reading and writing mathematical proofs. Readers may have gained such experience in computer science courses such as algorithms, automata or complexity theory, or some type of “discrete mathematics for computer science students” course. They also may have gained such experience in undergraduate mathematics courses, such as abstract or linear algebra. The material in these mathematics courses may overlap with some of the material presented here; however, even if the reader already has had some exposure to this material, it nevertheless may be convenient to have all of the relevant topics easily accessible in one place; moreover, the emphasis and perspective here will no doubt be different from that in a traditional mathematical presentation of these subjects.

**PURE MATHEMATICS**

Mathematics can be subdivided into the study of quantity, structure, space, and change (i.e. arithmetic, algebra, geometry, and analysis).

*Quantity*

The study of quantity starts with numbers, first the familiar natural numbers and integers ("whole numbers") and arithmetical operations on them, which are characterized in arithmetic. The deeper properties of integers are studied in number theory, from which come such popular results as Fermat's Last Theorem. The twin prime conjecture and Goldbach's conjecture are two unsolved problems in number theory.

*Structure*

Many mathematical objects, such as sets of numbers and functions, exhibit internal structure as a consequence of operations or relations that are defined on the set. Mathematics then studies properties of those sets that can be expressed in terms of that structure; for instance number theory studies properties of the set of integers that can be expressed in terms of arithmetic operations. Moreover, it frequently happens that different such structured sets (or structures) exhibit similar properties, which makes it possible, by a further step of abstraction, to state axioms for a class of structures, and then study at once the whole class of structures satisfying these axioms. Thus one can study groups, rings, fields and other abstract systems; together such studies (for structures defined by algebraic operations) constitute the domain of abstract algebra.

*Space*

The study of space originates with geometry.

Change

Understanding and describing change is a common theme in the natural sciences, and calculus was developed as a powerful tool to investigate it. Functions arise here, as a central concept describing a changing quantity. The rigorous study of real numbers and functions of a real variable is known as real analysis, with complex analysis the equivalent field for the complex numbers. Functional analysis focuses attention on (typically infinite-dimensional) spaces of functions. One of many applications of functional analysis is quantum mechanics. Many problems lead naturally to relationships between a quantity and its rate of change, and these are studied as differential equations. Many phenomena in nature can be described by dynamical systems; chaos theory makes precise the ways in which many of these systems exhibit unpredictable yet still deterministic behavior.

**LINUX SYSTEM**

Linux is an operating system: a series of programs that let you interact with your computer and run other programs.

An operating system consists of various fundamental programs which are needed by your computer so that it can communicate and receive instructions from users; read and write data to hard disks, tapes, and printers; control the use of memory; and run other software. The most important part of an operating system is the kernel. In a GNU/Linux system, Linux is the kernel component. The rest of the system consists of other programs, many of which were written by or for the GNU Project.

Linux is modelled on the Unix operating system. From the start, Linux was designed to be a multi-tasking, multi-user system. These facts are enough to make Linux different from other well-known operating systems. However, Linux is even more different than you might imagine. In contrast to other operating systems, nobody owns Linux. Much of its development is done by unpaid volunteers.

Development of what later became GNU/Linux began in 1984, when the Free Software Foundation began development of a free Unix-like operating system called GNU.

The GNU Project has developed a comprehensive set of free software tools for use with Unix™ and Unix-like operating systems such as Linux. These tools enable users to perform tasks ranging from the mundane (such as copying or removing files from the system) to the arcane (such as writing and compiling programs or doing sophisticated editing in a variety of document formats).

While many groups and individuals have contributed to Linux, the largest single contributor is still the Free Software Foundation, which created not only most of the tools used in Linux, but also the philosophy and the community that made Linux possible.

The Linux kernel first appeared in 1991, when a Finnish computing science student named Linus Torvalds announced an early version of a replacement kernel for Minix to the Usenet newsgroup comp.os.minix. See Linux International's Linux History Page.

Linus Torvalds continues to coordinate the work of several hundred developers with the help of a number of subsystem maintainers. There is an official website for the Linux kernel. Information about the linux-kernel mailing list can be found on the linux-kernel mailing list FAQ.

Linux users have immense freedom of choice in their software.

**DATABASE MODELLING LAYERS**

Today’s dominant information modeling methodology for producing database designs factors an information model into four main levels: Physical, Logical (Relational), Conceptual, and Programming/Presentation. The physical model describes how data are represented in physical resources such as memory, wire or disk. The vocabulary of concepts discussed at this layer includes record formats, file partitions and groups, heaps, and indexes. The physical model is typically invisible to the application – applications usually target the logical or relational data model described in the next section. Changes to the physical model should not impact application logic, but may impact application performance.

A logical data model is a complete and precise information model of the target domain. The relational model is the representation of choice for most logical data models. The concepts discussed at the logical level include tables, rows, and primary key-foreign key constraints, and normalization.

While normalization helps to satisfy important application requirements such as data consistency and increased concurrency with respect to updates and OLTP performance, it also introduces significant challenges for applications. Data at the logical level is too fragmented and application logic needs to aggregate rows from multiple tables into higher level entities that more closely resemble the artifacts of the application domain.

The conceptual model captures the core information entities from the problem domain and their relationships. A well-known conceptual model is the Entity-Relationship Model introduced by Peter Chen in 1976. UML is a more recent example of a conceptual model. Unfortunately, however, the conceptual data model is captured inside a database design tool that has little or no connection with the code and the relational schema used to implement the application. The database design diagrams created in the early phases of the application life cycle usually stay “pinned to a wall” growing increasingly disjoint from the reality of the application implementation with time. However, a conceptual data model can be as real, precise, and focused on the concrete “concepts” of the application domain as a logical relational model. A goal of the Microsoft Data Access vision is to make the conceptual data model a concrete feature of the data platform.

**APPLICATION EVOLUTION**

Data-based applications 10–20 years ago were typically structured as data monoliths; closed systems with logic factored by verb-object functions that interacted with a database system at the logical schema (e.g. relational) level.

A typical order entry system built around a relational database management system (RDBMS) 20 years ago would have logic partitioned around verb-object functions associated with how users interacted with the system. In fact, the user interaction model via “screens” or “forms” became the primary factoring for logic – there would be a new-order screen, and update-customer screen. The system may have also supported batch updates of SKU’s, inventory, etc. The application logic was tightly bound to the logical relational schema.

Much of the data-centric logic (e.g. validation logic) is embedded within the application logic. People typically wrote batch programs to interact directly with the logical schema to perform updates. Programming languages did not support representation of high-level abstractions directly – objects did not exist. These applications can be characterized as being closed systems whose logical data consistency was maintained by application logic implemented at the logical schema level. An order was an order because the new-order logic ensured that it was.

A key reason for custom data-centric logic by applications is the well-known application impedance mismatch problem. The logical schema does not match the level of abstraction of the application. Applications address this problem by developing at the data abstraction (e.g. relational) and by writing custom mapping code to bridge the gap between the application and the data abstractions. This not only leads to duplication of effort but also reduces application development productivity.

Several significant trends have shaped the way that modern data-based applications are factored and deployed today. Chief among these are object oriented factoring, service level application composition, and higher level data services. When we think about the factoring, composition, and services from above, we can see that the conceptual entities are an important part of today’s applications. It is also easy to see how these entities must be mapped to a variety of representations and bound to a variety of services.

**Design patterns**

A design pattern in architecture and computer science is a formal way of documenting a solution to a design problem in a particular field of expertise. An organized collection of design patterns that relate to a particular field is called a pattern language.

In the object oriented world, design patterns tell us how, in the context of a certain problem, we should structure the classes and objects. They do not translate directly into the solution; rather have to be adapted to suit the problem context.

With knowledge of design patterns, you can talk in – pattern language, because a lot of known design patterns have been documented. In discussions, one could ask, if one shouldn't use <design-pattern> for this implementation without talking about how classes would be implemented and how objects would be created. It is similar to asking a tester to do Boundary Value Analysis, rather than triggering a long talk on the subject.

Design patterns should not be confused with frameworks and libraries.

Python is a ground-up object oriented language which enables one to do object oriented programming in a very easy manner. While designing solutions in Python, especially the ones that are more than use-and-throw scripts which are popular in the scripting world, they can be very handy. Python is a rapid application development and prototyping language. So, design patterns can be a very powerful tool in the hands of a Python programmer.

*Design Pattern Classifications.*

• Creational Patterns – They describe how best an object can be created. A simple example of such design pattern is a singleton class where only a single instance of a class can be created. This design pattern can be used in situations when we cannot have more than one instances of logger logging application messages in a file.

• Structural Patterns – They describe how objects and classes can work together to achieve larger results. A classic example of this would be the facade pattern where a class acts as a facade in front of complex classes and objects to present a simple interface to the client.

• Behavioral Patterns – They talk about interaction between objects. Mediator design pattern, where an object of mediator class, mediates the interaction of objects of different classes to get the desired process working, is a classical example of behavioral pattern.