Allocating resource & to find the off-spring humanware as sponsor based on the computational consequence of artificial feed-forward-backpropagation neural network and genetic algorithm

CSE 400 Dissertation

This dissertation submitted in partial fulfillment of the requirement of the degree of Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering
IBAIS University
Dhanmondi, Dhaka 1209
Bangladesh

Supervisor
Sonia Akther Mim
Lecturer
Department of Computer Science and Engineering
IBAIS University
Dhaka 1209, Bangladesh

Student Information
Rashadul Islam
Student identification number: 0301175301
Department of Computer Science and Engineering
IBAIS University
Dhaka 1209, Bangladesh

Date
Tue Dec 3 16:00:10 EST 2019

Contents

List of Figures

List of Tables

Approvals

Acknowledgements

Become grateful to all the fellows:

Category	Entitle	Institution	Geo-location
Supervisor	Sonia Akhter Mim (Faculty)	IBAIS University	Dhaka, Bangladesh
Support & Facilities	Administration	IBAIS University	Dhaka, Bangladesh
Finance	To my family and allies	Home	Bangladesh
Documentation	To my cousins	Home	Bangladesh
Proofreading	To my fiends	Home	Bangladesh

Blank left intentionally for perspectives.

Chapter 1

Abstract

Epistemic virtues and cognitive responsibility is the next generation industrial magnet. To come to such emerging software engineering trend and technology, artificial intelligence explores the new age of civilization, industrial, corporate culture and precise decision making & research industry. Artificial Intelligence is a diverse field of study in which machine learning algorithms acts as the cogwheel. During the literal eventual journey of academic project and this document, the artificial intelligence is the key factor to compile the software engineering projects and the managing such projects. Neural Network is the simplest version of artificial intelligence and machine learning algorithm compare to the cutting-edge applications in human era. To progress the project, the feed-forward neural network with prominent backpropagation algorithm is being used to predict the estimate of the most precise resource i.g. space of a premises, duration of stay in the premises, life style of the client, premises type and how much client can expense. With impeccable loss or cost function and accuracy score, the application can act like a stand-alone application. Though, this application takes higher memory and space while calculation & dataset is small related to the industrial scale, the feed-forward based backpropagation algorithm explains what the management seeking for to come to an evidence based decision. In the second stage of development, the genetic algorithm has been used using python's tpot model, with which the organization has find the manager as sponsor the client during the client's stay in their respected premises. Using this model takes much higher execution time to implement in Java Enterprise based WebApp. With active communication and supervising, supervised learning as part of machine learning task and software engineering theory, practice and project management paradigm(CMMI, PCMM, SCRUM) along with J2EE-MVC architecture and Apache Tomcat Server, the project conclusively outputed the expected scope of our work and predicted results. Limitations and hazards of severity are marginal during the compilation. Future work on this study belongs to large scale mature computing environment and for those who wants to be the part of next generation of innovation in diversity and dynamic decision making system. This software application as well as the dissertation is the path of future or next era of our living civilization.[1]

[1]Keywords: Artificial Intelligence, Machine Learning Algorithms, Feed-forward Neural Network, Backpropagation Algorithm, Genetic Algorithm, Resource Allocation, Prediction, J2EE, MVC, Apache TomCat Server, SCRUM, CMMI, PCMM, Large Scale Computation

Chapter 2

Introduction

The beginning of this academic dissertation starts with the two base statements: (1) to find the inventory resources (like rooms, cabins, and other similarity) using next generation trend entitled as feed-forward based backpropagation algorithm; and (2) to find the sponsor from employee pool according to their overall equipment effectiveness score using genetic algorithm. Parallelly, the heart of the dissertation based on five discrete mathematics propositions: (i) arrangement of the resource (ii) clustering demand & supply (iii) Finding the predicted resource (neural network) (iv) finding sponsor by machine learning algorithm and genetic algorithm (v) allocate the resource to the client. With all these, the core follow up becomes artificial intelligence, genetic algorithm, feed-forward based backpropagation algorithm, software engineering, project management, software risks, engineering ethics, limitations and the project impacts.

The first segment covers the literature. Where, the theoretical perspective about artificial intelligence has been populated. Artificial Intelligence is defined as acting humanly followed by Turing test, thinking humanly followed by cognitive modeling, thinking rationally (based on or in accordance with reason or logic) followed by laws of thoughts and acting rationally followed by rational agent. Historical evolution serves a great deal in the field of artificial intelligence since world war two and makes industrial and individual life at least complete and diverse.

In the second phase, the sub-field of artificial intelligence entitled as Machine Learning Algorithm has been describes with core details. Machine learning is the study that gives computers the ability to learn without being explicitly programmed. The document and the project follows supervised learning as one of the categories of machine learning category. Machine is widely accepted for accuracy, automation, fastness and scalability. Deep learning, a sub-field of machine learning, uses multiple layers to progressively extract higher level features from the primary or secondary input. Neural Network which is one of the heart of this dissertation is one of the most used criteria of deep learning. Neural networks are a class of iterative machine learning algorithms that are using to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons. Feed-forward Neural network is the simplest form of artificial neural network. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or

loops in the network.

In machine learning, specifically deep learning, backpropagation (backprop is an algorithm widely used in the training of feed-forward neural networks for supervised learning; generalizations exist for other artificial neural networks, and for functions generally. Backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network for a single input-output example. This makes it feasible to use gradient methods for training multi-layer networks, updating weights to minimize loss; commonly one uses gradient descent or variants such as stochastic gradient descent. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, iterating backwards one layer at a time from the last layer to avoid redundant calculations of intermediate terms in the chain rule. For backpropagation, the loss function calculates the difference between the network output and its expected output, after a training example has propagated through the network. Inside the project, there has been used logistic or sigmoid activation function, the loss is relatively achievable to get expected estimate results to find the resources with expected F1 (statistical accuracy) score and basic embedded optimization function.

In computer science & engineering and operations research, a genetic algorithm is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as selection, crossover and mutation. Using this phenomenon, the programming phase of the project predict the sponsor from the human resource stacks.

In the last phase of this first segment, software and software engineering has been defined. In this phase, software engineering process, software engineering principles, software requirement model, design model and its quality attributes, and software architecture literally described. The Model-View-Controller software architecture is broadly used in the project. Software quality can be defined as: An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce and those who use it. One of the core objective of this work is to breach good quality.

The second segment covers the methodology that has been used in the development project, project management and dissertation. The scope of the project or dissertation documentation has been mathematically described. The research method has been detailed and while maintaining quality and efficiency of the project, there replaces great impact of choosing "Mix Research Method" which follows both quantitative and qualitative methods. The entire process, process of work, and managing the entire life cycle of the dissertation gets simple while following this methods with very low limitations.

In this segment, Agile software development is the rising node while developing the project. This comprises various approaches to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s). This term advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change. SCRUM is one of prominent agile method which has been used in the project and being detailed on this segment. Same time, the platform and system design, data collection & processing, software framework, J2EE MVC and multi-tier or N-tier architecture, client-server model can

be found. Snapshot of feed-forward backpropagation algorithm is visible in this segment. As part of genetic algorithm to find the expected sponsor, TPOT(Tree-based Pipeline Optimization Tool) has been used which is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. During the software development, there has been used Apache Tomcat Server and People Capability Maturity Model to ease the work flow and better understanding. To the end, the procedures of the work has been mentioned.

In the third segment, the implementation results has been listed and briefly discussed. In the fourth segment, the engineering ethics are enlisted. In the fifth segment, the engineering limitations has been concerned. In the sixth segment, the impacts and future of such project has been proportioned and not limited not in resource allocation but also large scale industrial use with effective hardware resources and using high computing resources.

Final words not limited only with those segments, the limitations exists while developing such dynamic system and software using artificial intelligence and genetic algorithm but supervised software project management using the right algorithm alike feed-forward based algorithm, TPOT: genetic algorithm, SCRUM: agile methodology, PCMM and high powered hardware & computing machines makes this possible to compile the complete work to come to a complete software application. The future of such work is the industrial and live software application which may be implemented in health-care, hospitality, manufacturing, sales-engineering, dynamic decision making and the list goes on as the industrial revolution and people gets matured. Such application is for those who belongs to the competitive edge of technology to sustain in the highly trendy, large scale competition and production with market research and searching & asking for real time decision making environment.

Chapter 3

Literature

3.1 Artificial Intelligence

3.1.1 Artificial intelligence

3.1.1.1 What is Artificial Intelligence

Artificial Intelligence(AI) is one of the newest fields in science and engineering. Work started in earnest soon after World War II, and the name itself was coined in 1956. AI currently encompasses a huge variety of subfields, ranging from the general (learning and perception) to the specific, such as playing chess, proving mathematical theorems, writing poetry, driving a car on a crowded street, and diagnosing diseases. AI is relevant to any intellectual task; it is truly a universal field.

In the following figure, there are eight definitions of AI, laid out along two dimensions. The definitions on top are concerned with thought processes and reasoning, whereas the ones on the bottom address behavior. The definitions on the left measure success in terms of fidelity to human performance, whereas the ones on the right measure against an ideal performance measure, called rationality. A system is rational if it does the "right thing," given what it knows. Historically, all four approaches to AI have been followed, each by different people with different methods. A human-centered approach must be in part an empirical science, involving observations and hypotheses about human behavior. A rationalist approach involves a combination of mathematics and engineering.

3.1.1.1.1 Approach 1: Acting humanly: The Turing Test The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. Details of the test ensure whether a computer would really be intelligent if it passed. The computer would need to possess the following capabilities:

1. natural language processing: to enable it to communicate successfully in English;

2. knowledge representation: to store what it knows or hears;

Artificial Intelligence		
Concerned With	Measure success in terms of fidelity to human performance	Measure against an ideal performance measure
Thought process and reasoning	Thinking Humanly 1. "The exciting new effort to make comput- ers think.....machines with minds, in the full and literal sense."[Haugeland 89] 2. "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning....."[Bellman 78]	Thinking Rationally 1. "The study of mental faculties through the use of computational models."[Charniak 85] 2. "The study of the computations that make it possible to perceive, reason, and act."[Winston 92]
Behavior	Acting Humanly 1. "The art of creating machines that perform functions that require intelligence when performed by people." [Kurzweil 92] 2. "The study of how to make computers do things at which, at the moment, people are better." [Rich 91]	Acting Rationally 1. "Computational Intelligence is the study of the design of intelligent agents." [Poole 98] 2. "AI.....is concerned with intelligent behavior in artifacts."[Nilsson 98]

Table 3.1: Definition of Artificial Intelligence

3. automated reasoning: to use the stored information to answer questions and to draw new conclusions;

4. machine learning: to adapt to new circumstances and to detect and extrapolate patterns.

Turings test deliberately avoided direct physical interaction between the interrogator and the computer, because physical simulation of a person is unnecessary for intelligence. However, the so-called total Turing Test includes a video signal so that the interrogator can test the subjects perceptual abilities, as well as the opportunity for the interrogator to pass physical objects "through the hatch." To pass the total Turing Test, the computer will need:

1. computer vision: to perceive objects, and

2. robotics: to manipulate objects and move about.

3.1.1.1.2 Approach 2: Thinking humanly: The cognitive modeling Taken a motion of such that a given program thinks like a human, then there

needs to determine how humans think. There are three ways to do this:

(a) through introspection: trying to catch our own thoughts as they go by;

(b) through psychological experiments: observing a person in action; and

(c) through brain imaging: observing the brain in action. Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the programs inputoutput behavior matches corresponding human behavior, that is evidence that some of the programs mechanisms could also be operating in humans.

For example, Allen Newell and Herbert Simon, who developed GPS, the "General Problem Solver" (Newell and Simon,1961), were not content merely to have their program solve problems correctly. They were more concerned with comparing the trace of its reasoning steps to traces of human subjects solving the same problems. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind.

Real cognitive science, however, is necessarily based on experimental investigation of actual humans or animals.

3.1.1.1.3 Approach 3: Thinking rationally: The "laws of thought" The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking," that is, irrefutable reasoning processes. His syllogisms provided patterns for argument structures that always yielded correct conclusions when given correct premises; for example, "Socrates is a man; all men are mortal; therefore, Socrates is mortal." These laws of thought were supposed to govern the operation of the mind; their study initiated the field called logic.

Logicians in the 19th century developed a precise notation for statements about all kinds of objects in the world and the relations among them. (Contrast this with ordinary arithmetic notation, which provides only for statements about numbers.) By 1965, programs existed that could, in principle, solve any solvable problem described in logical notation. (Although if no solution exists, the program might loop forever.) The so called logicist tradition within artificial intelligence hopes to build on such programs to create intelligent systems. There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between solving a problem "in principle" and solving it in practice. Even problems with just a few hundred facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first. Although both of these obstacles apply to any attempt to build computational reasoning systems, they appeared first in the logicist tradition.

3.1.1.1.4 Approach 4: Acting rationally: The rational agent An agent is just something that acts (agent comes from the Latin agere, to do). Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a

prolonged time period, adapt to change, and create and pursue goals. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

In the "laws of thought" approach to AI, the emphasis was on correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve ones goals and then to act on that conclusion. On the other hand, correct inference is not all of rationality; in some situations, there is no provably correct thing to do, but something must still be done. There are also ways of acting rationally that cannot be said to involve inference.

For example, recoiling from a hot stove is a reflex action that is usually more successful than a slower action taken after careful deliberation.

All the skills needed for the Turing Test also allow an agent to act rationally. Knowledge representation and reasoning enable agents to reach good decisions. There need to generate comprehensible sentences in natural language to get by in a complex society with effective behavior. The rational-agent approach has two advantages over the other approaches. First, it is more general than the "laws of thought" approach because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is more amenable to scientific development than are approaches based on human behavior or human thought. The standard of rationality is mathematically well defined and completely general, and can be "unpacked" to generate agent designs that provably achieve it. Human behavior, on the other hand, is well adapted for one specific environment and is defined by, well, the sum total of all the things that humans do.

Rationality simplifies the problem and provides the appropriate setting for most of the foundational material in the field. [Russell 03]

3.1.1.1.5 Types of Artificial Intelligence AI can be classified in any number of ways there are two types of main classification:[GN]

(a) Type1 (based on tasks)

   i. Weak AI or Narrow AI: It is focused on one narrow task, the phenomenon that machines which are not too intelligent to do their own work can be built in such a way that they seem smart. An example would be a deck game where a machine beats human where in which all rules and moves are fed into the machine. Here each and every possible scenario need to be entered beforehand manually. Each and every weak AI will contribute to the building of strong AI.

   ii. Strong AI: The machines that can actually think and perform tasks on its own just like a human being. There are no proper existing examples for this but some industry leaders are very keen on getting close to build a strong AI which has resulted in rapid progress.

(b) Type2 (based on functionalities)

   i. Reactive Machines: This is one of the basic forms of AI. It doesnt have past memory and cannot use past information to information for the future actions.

ii. Limited Memory: AI systems can use past experiences to inform future decisions. Some of the decision-making functions in self-driving cars have been designed this way. Observations used to inform actions happening in the not so distant future, such as a car that has changed lanes. These observations are not stored permanently.

iii. Theory of Mind: This type of AI should be able to understand peoples emotion, belief, thoughts, expectations and be able to interact socially Even though a lot of improvements are there in this field this kind of AI is not complete yet.

iv. Self-awareness: An AI that has its own conscious, super intelligent, self-awareness and sentient (In simple words a complete human being).

3.1.1.1.6 History of Artificial Intelligence Formally, in the 1940s and 50s, a handful of scientists from a variety of fields (mathematics, psychology, engineering, economics and political science) began to discuss the possibility of creating an artificial brain. The field of artificial intelligence research was founded as an academic discipline in 1956.

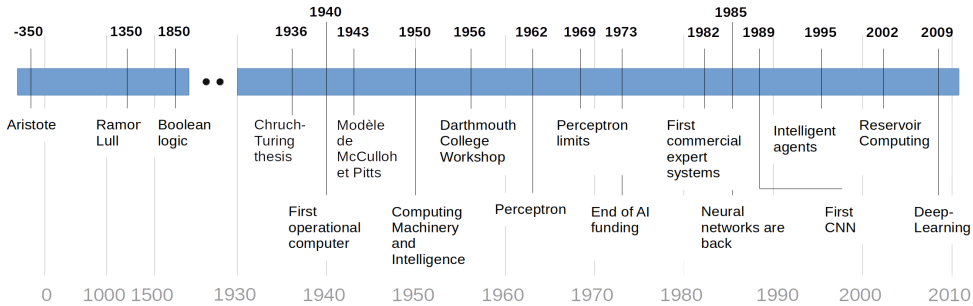


Figure 3.1: Historical journey of artificial intelligence

3.1.2 Machine learning algorithm

"the study that gives computers the ability to learn without being explicitly programmed." - described by Arthur Samuel, a computer scientist who pioneered the study of artificial intelligence in 1959.

Alan Turings seminal paper (Turing, 1950) introduced a benchmark standard for demonstrating machine intelligence, such that a machine has to be intelligent and responsive in a manner that cannot be differentiated from that of a human being:

"Machine Learning is an application of artificial intelligence where a computer/-machine learns from the past experiences (input data) and makes future predictions. The performance of such a system should be at least human level."

Technically, by Tom M. Mitchells (1997): A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

3.1.2.1 Elements of Machine Learning

Every machine learning algorithm has three components:

(a) Representation:
How to represent knowledge. Examples include decision trees, sets of rules, instances, graphical models, neural networks, support vector machines, model ensembles and others.

(b) Evaluation:
The way to evaluate candidate programs (hypotheses). Examples include accuracy, prediction and recall, squared error, likelihood, posterior probability, cost, margin, entropy k-L divergence and others.

(c) Optimization:
The way candidate programs are generated known as the search process. For example combinatorial optimization, convex optimization, constrained optimization.

All machine learning algorithms are combinations of these three components. A framework for understanding all algorithms.

3.1.2.2 Machine learning workflow

3.1.2.3 Machine learning Categories

Machine Learning is generally categorized into three types:

(a) Supervised Learning

(b) Unsupervised Learning

(c) Semi-supervised Learning

(d) Reinforcement learning

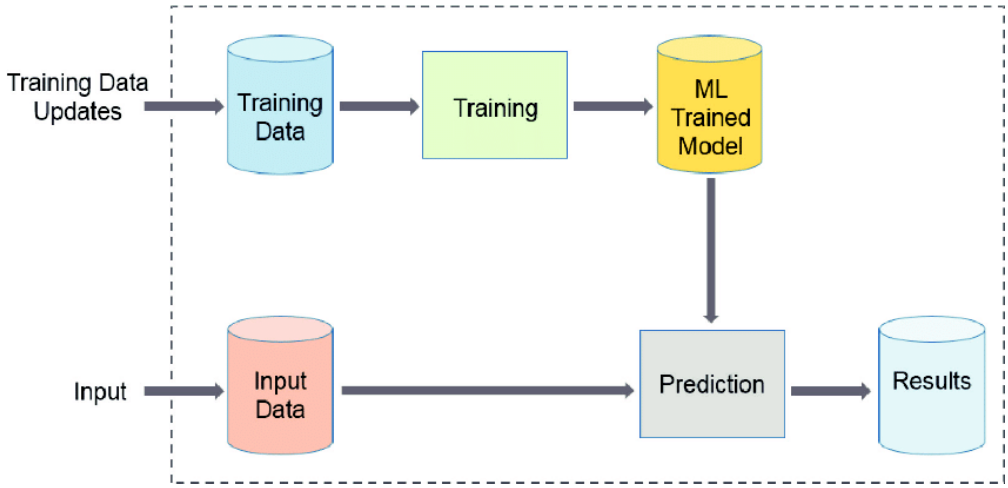


Figure 3.2: Machine learning workflow

3.1.2.3.1 Hebbian theory Hebbian theory is a neuroscientific theory claiming that an increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell. It is an attempt to explain synaptic plasticity, the adaptation of brain neurons during the learning process. It was introduced by Donald Hebb in his 1949 book The Organization of Behavior. The theory is also called Hebb's rule, Hebb's postulate, and cell assembly theory. Hebb states it as follows:

"Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability... When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

The theory is often summarized as "Cells that fire together wire together."

3.1.2.3.2 Supervised Learning or inductive learning Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly. Simply, supervised machine learning is using past data to make decisions.

Literally, Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.[Russell 03]

3.1.2.3.3 Unsupervised Learning Unsupervised machine learning are used when the information used to train is neither classified nor labeled. Unsupervised

Supervised machine learning techniques		
Problem	Description	Examples or cases
Classification	Determine the discrete class to which each individual belongs, based on input data	Spam filtering, sentiment analysis, fraud detection, customer ad targeting, churn prediction, support case flagging, content personalization, detection of manufacturing defects, customer segmentation, event discovery, genomics, drug efficacy
Regression	Predict the real-valued output for each individual, based on input data	Stock-market prediction, demand forecasting, price estimation, ad bid optimization, risk management, asset management, weather forecasting, sports prediction
Recommendation	Predict which alternatives a user would prefer	Product recommendation, job recruiting, Netflix Prize, online dating, content recommendation
Imputation	Infer the values of missing input data	Incomplete patient medical records, missing customer data, census data

Table 3.2: Supervised machine learning techniques

learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesnt figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Unsupervised learning is a type of self-organized Hebbian learning that helps find previously unknown patterns in data set without pre-existing labels. It is also known as self-organization and allows modeling probability densities of given inputs.[Hinton 99]

3.1.2.3.4 Semi-supervised Learning Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiringunlabeled data generally doesnt require additional resources.[Wha]

3.1.2.3.5 Reinforcement learning Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method

allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.[Weber 08]

Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Actions may affect not only the immdediate reward but also the next situation and though that, all subsequent rewards. These two characteristics:

(a) Trial and error search

(b) Delayed reward

are the two most prominent distinguishing features of reinforcement learning. [Sutton 18]

3.1.2.3.6 Advantages to machine learning

3.1.2.3.6.1 Accurate Machine learning uses data to discover the optimal decision making engine for your problem. As you collect more data, the accuracy can increase automatically.

3.1.2.3.6.2 Automated As answers are validated or discarded, the Machine learning model can learn new patterns automatically. This allows users to embed Machine learning directly into an automated workflow.

3.1.2.3.6.3 Fast Machine learning can generate answers in a matter of milliseconds as new data streams in, allowing systems to react in real time.

3.1.2.3.6.4 Customizable Many data driven problems can be addressed with machine learning. Machine learning models are custom built from your own data, and can be configured to optimize whatever metric drives your business.

3.1.2.3.6.5 Scalable As your business grows, Machine learning easily scales to handle increased data rates. Some Machine learning algorithms can scale to deals with large amounts of data on many machines in the cloud.

3.1.2.3.7 Machine learning algorithms Most common machine learning algorithms are as follows[Mac]:

Figure 3.3: Reinforcement learning

3.1.2.3.7.1 Classification algorithms

(a) Decision tree
(b) Decision stump
(c) Naive Bayes
(d) Gaussian Naive Bayes
(e) Bernoulli Naive Bayes
(f) Multinomial Naive Bayes
(g) K Nearest Neighbours
(h) Support Vector Machine
(i) Linear Support Vector Classifier
(j) NuSVC

(k) Stochastic Gradient Descent Classifier

(l) Bayesian network

(m) Logistic Regression

(n) Zero Rule

(o) One Rule

(p) Linear Discriminant Analysis

(q) Quadratic Discriminant Analysis

(r) Fisher's Linear Discriminant

3.1.2.3.7.2 Regression Analysis

(a) Linear Regression

(b) Polynomial Regression

(c) Poisson Regression

(d) Ordinary Least Squares Regression

(e) Ordinal Regression

(f) Support Vector Regression

(g) Gradient Descent Regression

(h) Stepwise Regression

(i) Lease Absolute Selection and Shrinkage

(j)

(k) Operator Regression

(l) Ridge Regression

(m) Elastic Net Regression

(n) Bayesian Linear Regression

(o) Least Angled Regression

(p) Neural Network Regression

(q) Locally Estimated Scatterplot Smoothing

(r) Multivariate Adaptive Regression Splines

(s) Locally Weighted Regression

(t) Quantile Regression

(u) Principal Component Regression

(v) Partial Least Squares Regression

3.1.2.3.7.3 Neural Network

(a) Perceptron
(b) Multilayer Perceptron
(c) Recurrent Neural Network
(d) Convolutional Neural Network
(e) Deep Belief Network
(f) Hopfield Networks
(g) Learning Vector Quantization
(h) Stacked Autoencoder
(i) Boltzmann Machine
(j) Restricted Boltzmann Machine
(k) Generative Adversarial Networks

3.1.2.3.7.4 Anomaly Detection

(a) Isolation Forest
(b) Once Class SVM
(c) PCA Based Anomaly Detection
(d) Fast Minimum Covariance Determinant
(e) Local Outlier Factor

3.1.2.3.7.5 Dimensionality Reduction

(a) Singular Value Decomposition
(b) Forward Feature Selection
(c) Backward Feature Elemination
(d) Subset Selection
(e) Principal Component Analysis
(f) Partial Least Squares Regression
(g) Latent Dirichlet Analysis
(h) Regularized Discriminant Analysis
(i) t Distributed Stochastic Neighbour Embedding
(j) Factor Analysis
(k) Multidimensional Scaling
(l) Auto Encoder
(m) Independent Component Analysis
(n) Isomap
(o) Local Linear Embedding
(p) Locality Sensitive Hashing
(q) Sammon Mapping

3.1.2.3.7.6 Ensemble

(a) Random Forest
(b) Bagging (Bootstrap Aggregation)
(c) Ada Boost
(d) Gradient Boosting
(e) Gradient Boosted Regression Trees
(f) Extreme Gradient Boosting
(g) Voting Classifier
(h) Extremely Randomized Trees
(i) Boosted Decision Tree
(j) Category Boosting
(k) Stacked Generalization

3.1.2.3.7.7 Clustering

(a) K Means Clustering
(b) K Medians Clustering
(c) Mean Shift Clustering
(d) K Modes Clustering
(e) Fuzzy K Modes
(f) Fuzzy C Means
(g) Mini Batch K Means Clustering
(h) Hierarchical Clustering
(i) Expectation Maximization
(j) Density Based Spatial Clustering
(k) Minimum Spanning Trees
(l) Quality Threshold
(m) Gaussian Mixture Model
(n) Spectral Clustering

3.1.2.3.7.8 Clustering

(a) Apriori
(b) Equivalence Class Clustering and bottom-up Lattice Traversal

3.1.2.3.7.9 Regularization

(a) Least Absolute Shrinkage and Selection Operator Regularization
(b) Ridge Regularization
(c) Elastic Net Regularization

3.1.2.3.8 Deep learning or deep structured learning or hierarchical learning Deep learning is a class of machine learning algorithms that[Deng 14, Page 199,200] uses multiple layers to progressively extract higher level features from the raw input.

Learning can be supervised, semi-supervised or unsupervised.

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 and analog.

3.1.2.4 Neural network

3.1.2.4.1 Neural Network

Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons. [Aggarwal 18]

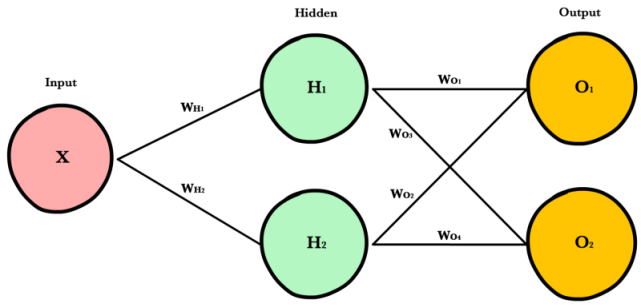


Figure 3.4: Neural Network

Neural networks are trained iteratively using optimization techniques like gradient descent. After each cycle of training, an error metric is calculated based on the difference between prediction and target. The derivatives of this error metric are calculated and propagated back through the network using a technique called backpropagation. Each neurons coefficients (weights) are then adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold.

3.1.2.4.2 Architecture of neural network

3.1.2.4.3 Neuron

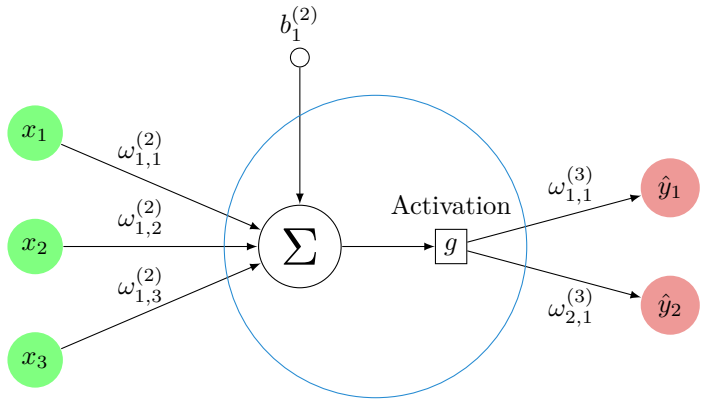


Figure 3.5: Architecture of Neural Network

A neuron takes a group of weighted inputs, applies an activation function, and returns an output.

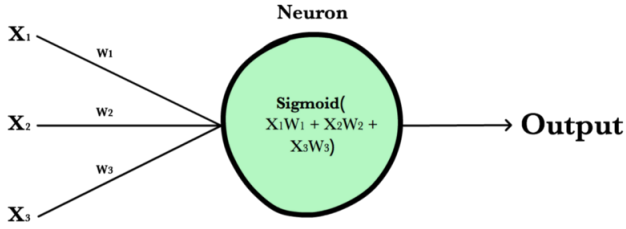


Figure 3.6: Neuron of Neural Network

Inputs to a neuron can either be features from a training set or outputs from a previous layers neurons. Weights are applied to the inputs as they travel along synapses to reach the neuron. The neuron then applies an activation function to the sum of weighted inputs from each incoming synapse and passes the result on to all the neurons in the next layer.

3.1.2.4.4 Synapse

Synapses are like roads in a neural network. They connect inputs to neurons, neurons to neurons, and neurons to outputs. In order to get from one neuron to another, you have to travel along the synapse paying the toll (weight) along the way. Each connection between two neurons has a unique synapse with a unique weight attached to it. When we talk about updating weights in a network, were really talking about adjusting the weights on these synapses.

3.1.2.4.5 Weights

Explanation of weights (parameters)

3.1.2.4.6 Bias

Bias terms are additional constants attached to neurons and added to the weighted input before the activation function is applied. Bias terms help models represent patterns that do not necessarily pass through the origin. For example, if all your features were 0, would your output also be zero? Is it possible there is some base value upon which your features have an effect? Bias terms typically accompany weights and must also be learned by your model.

3.1.2.4.7 Layers

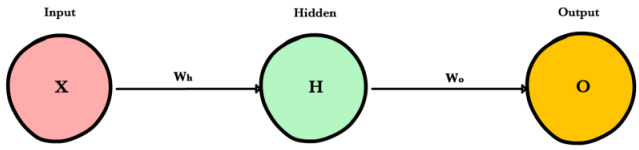


Figure 3.7: Layers of Neural Network

3.1.2.4.7.1 Input Layer

Holds the data your model will train on. Each neuron in the input layer represents a unique attribute in your dataset (e.g. height, hair color, etc.).

3.1.2.4.7.2 Hidden Layer

Sits between the input and output layers and applies an activation function before passing on the results. There are often multiple hidden layers in a network. In traditional networks, hidden layers are typically fully-connected layerseach neuron receives input from all the previous layers neurons and sends its output to every neuron in the next layer. This contrasts with how convolutional layers work where the neurons send their output to only some of the neurons in the next layer.

3.1.2.4.7.3 Output Layer

The final layer in a network. It receives input from the previous hidden layer, optionally applies an activation function, and returns an output representing your models prediction.

3.1.2.4.7.4 Weighted Input

A neurons input equals the sum of weighted outputs from all neurons in the previous layer. Each input is multiplied by the weight associated with the synapse

connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weightsone for each each synapse.

(a) **Single Input**

$$\begin{aligned} Z &= Input \times Weight \\ &= xw \end{aligned} \tag{3.1}$$

(b) **Multiple Inputs**

$$\begin{aligned} Z &= \sum_{i=1}^{n} x_i w_i \\ &= x_1 w_1 + x_2 w_2 + x_3 w_3 \end{aligned} \tag{3.2}$$

Notice, its exactly the same equation we use with linear regression! In fact, a neural network with a single neuron is the same as linear regression! The only difference is the neural network post-processes the weighted input with an activation function.

3.1.2.4.8 Activation Functions

Activation functions live inside neural network layers and modify the data they receive before passing it to the next layer. Activation functions give neural networks their powerallowing them to model complex non-linear relationships. By modifying inputs with non-linear functions neural networks can model highly complex relationships between features. Popular activation functions include relu and sigmoid.

Activation functions typically have the following properties:

(a) **Non-linear**

In linear regression were limited to a prediction equation that looks like a straight line. This is nice for simple datasets with a one-to-one relationship between inputs and outputs, but what if the patterns in our dataset were non-linear? (e.g. x2, sin, log). To model these relationships we need a non-linear prediction equation.ź Activation functions provide this non-linearity.

(b) **Continuously differentiable**

To improve our model with gradient descent, we need our output to have a nice slope so we can compute error derivatives with respect to weights. If our neuron instead outputted 0 or 1 (perceptron), we wouldnt know in which direction to update our weights to reduce our error.

(c) **Fixed Range Activation functions**

Typically squash the input data into a narrow range that makes training the model more stable and efficient.

3.1.2.4.9 Loss Functions

A loss function, or cost function, is a wrapper around our models predict function that tells us how good the model is at making predictions for a given set of parameters. The loss function has its own curve and its own derivatives. The slope of this curve tells us how to change our parameters to make the model more accurate! We use the model to make predictions. We use the cost function to update our parameters. Our cost function can take a variety of forms as there are many different cost functions available. Popular loss functions include: MSE (L2) and Cross-entropy Loss.

3.1.2.4.10 Model fitting

- Under-fitting: When the model has fewer features and hence not able to learn from the data very well. This model has high bias.
- Over-fitting: When the model has complex functions and hence able to fit the data very well but is not able to generalize to predict new data. This model has high variance.

3.1.2.4.11 Optimization Algorithms

Optimisation Algoritms are used to update weights and biases i.e. the internal parameters of a model to reduce the error. They can be divided into two categories:

(a) Constant Learning Rate Algorithms: Most widely used Optimisation Algorithm, the Stochastic Gradient Descent falls under this category.

$$\mathrm{W}^{k+1} = W^k - \eta \times (\delta J(W)) \tag{3.3}$$

Here η is called as learning rate which is a hyperparameter that has to be tuned. Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully slow convergence i.e will result in small baby steps towards finding optimal parameter values which minimize loss and finding that valley which directly affects the overall training time which gets too large. While a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

A similar hyperparameter is momentum, which determines the velocity with which learning rate has to be increased as we approach the minima.

(b) Adaptive Learning Algorithms: The challenge of using gradient descent is that their hyper parameters have to be defined in advance and they depend heavily on the type of model and problem. Another problem is that the same learning rate is applied to all parameter updates. If we have sparse data, we may want to update the parameters in different extent instead.

Adaptive gradient descent algorithms such as Adagrad, Adadelta, RMSprop, Adam, provide an alternative to classical SGD. They have per-paramter

learning rate methods, which provide heuristic approach without requiring expensive work in tuning hyperparameters for the learning rate schedule manually.

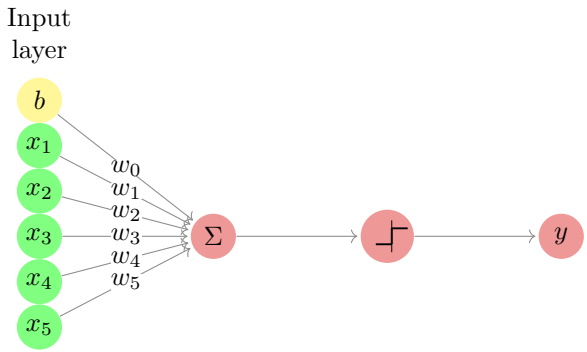


Figure 3.8: A representation of the Perceptron.

3.1.3 Feed forward backpropagation neural network

Machine learning is a complex subfield of artifical intelligence that witnessed a very quick expansion in the recent years. It aims to enable computers to *learn* how to tackle problems by detecting patterns and regularities in the training data and trying to generalize this extracted knowledge to new, unseen data. Among its many powerful tools, artificial neural networks are models that take inspiration from what is known about the human brain, by mimicking its connectivity patterns, learning rules and signal propagation, under the constraints imposed by our limited knowledge of the brain and the intrinsically different "hardware" at our disposal.

3.1.3.1 Artificial neural networks

Biological brains are composed by a large number of simple elements, called neurons, that are highly interconnected. The number of neurons in the human brain is estimated to be around 10^{11}, each one connected to a little less than 10^4 other neurons, resulting in a total between 10^{14} and 10^{15} synapses. The activity of each of these either excites or inhibits the surrounding neurons it is connected to, generating a complex network of interactions. Artificial Neural Networks (ANNs) take inspiration from this understanding of the human brain, building networks composed of many artificial neurons, small elements that perform very simple operations on their inputs.

3.1.3.2 Brief history of neural networks

In 1943 McCulloch and Pitts defined a mathematical model of how a biological neuron works[McCulloch 43]. The artificial neuron they proposed was able to solve simple binary problems, but did not learn. In 1949[Hebb 49] suggested that humans learn by enhancing the neural pathways between neurons that collaborate, and weakening the others. Only a decade later this learning rule inspired the Perceptron (see Figure 3.8), the first ANN that was able to vary its own weights, i.e., to find the setting that allowed it to exhibit the desired behavior[Rosenblatt 57].

The activation rule of the perceptron is very simple and is at the base of many modern neural networks. Given an n-dimensional input $\mathbf{x} = (x_1, \ldots, x_n)$, the weighted sum of each dimension of the input x_i and its associated weight w_i is computed as

$$z = \sum_{i=1}^{n} (w_i \cdot x_i) + b,$$

where the weight w_i is often referred to as *preactivation* and to simplify the notation the bias term b will often be replaced by an equivalent input term $x_0 = 1$ weighted by $w_0 = b$. Note that this is simply the dot product of the weight vector $\mathbf{w} = (w_1, \ldots, w_n)$ and the input vector $\mathbf{x}$. The result of this first affine transformation is then passed through a step function of the form

$$y = \begin{cases} 1, & \text{if } z \geq 0, \\ 0, & \text{otherwise,} \end{cases}$$

that determines the binary output of the Perceptron.

This can be used, for instance, to classify whether the input belongs to a specific class or not. Note that the model can be easily extended to handle multiple classes, by simply adding more dimensions to y and assigning each of them to a different class.

The behavior of the Perceptron is indeed remarkable, but the biggest innovation of[Rosenblatt 57] is most probably the update algorithm that allowed one to modify the weights of the model. Given a training pair $(\mathbf{x}, y)$ consisting of an input and its corresponding desired output, the weights and the bias – also called the *parameters* of the network – are *learned* according to the following rule:

$$w_i^{\text{new}} = w_i^{\text{old}} - \eta \cdot (\hat{y} - y) \cdot x_i, \tag{3.4}$$

where $\hat{y}$ is the output of the Perceptron, y is the target (i.e., desired) value, x_i and w_i^{old} are respectively the i-th input and weight at the previous iteration and η is a scaling factor that allows to adjust the magnitude by which the weights are modified.

The introduction of a model that could learn from data was welcomed with excitement as the beginning of a new era and research in ANNs became very active for approximately a decade, until in 1969 Minsky and Papert published a detailed mathematical analysis of the Perceptron, demonstrating that a single layered Perceptron could not model basic operations like the XOR logic operation[Minsky 69]. The limit of Perceptrons is that they can only solve linearly separable problems, and fail at tackling nonlinearly separable problems like the XOR (see Figure 3.9). This is not the case for Multi-Layer Perceptrons (MLP, depicted in Figure 3.10), an evolution of Perceptrons that introdu ces one or more intermediate layers (called *hidden layers*) between the input and the output. Since Since in each of these layers the preactivation is followed by a nonlinearity, the result is a nonlinear transformation that can project the input into a linearly separable space.

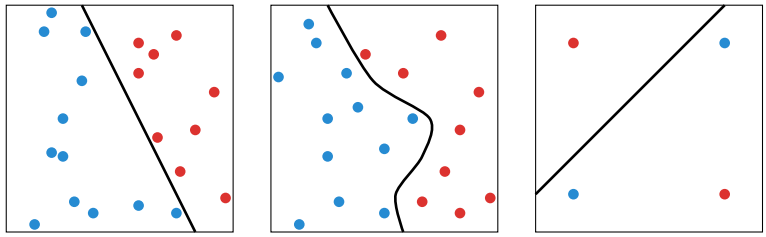


Figure 3.9: a) A linearly separable problem. b) A nonlinearly separable problem. c) The XOR problem with a tentative solution that fails at separating the points of the space.

The excessive enthusiasm for the early successes of ANNs turned into strong disappointment: even if[Minsky 69] showed that an MLP could model the XOR bitwise operation, they also pointed out that Rosenblatt's learning algorithm was limited to single layered Perceptrons and could not autonomously learn how to solve the problem. The expectation of an artificial intelligence that could learn by itself to solve problems and interact with humans appeared suddenly unrealistic and most of the research community lost interest in ANNs. The field experienced a severe slow down and most of the fundings were cut.

After a decade known as the first AI Winter, in 1982 John Hopfield presented a model of human memory that did not only give insights on how the brain works, but was also useful in practical applications and had a sound and detailed mathematical grounding. At the same time, at the US-Japan Joint Conference on Cooperative/Competitive Neural Networks, Japan announced a renewed effort in building neural networks and the fear that the US might be left behind renewed their effort on this topic.

The breakthrough that completely restored the interest in the field came in 1986, when[Rumelhart 86a] rediscovered the backpropagation algorithm[Linnainmaa 70, Werbos 74] that allowed to train ANNs composed by multiple layers by performing gradient descent (see Section 3.1.3.6). However once again the expectation was set too high and AI was expected to allow to do things like translating languages, carring on conversations and interpreting pictures. In late 80s and early 90s AI fundings were cut again when those very high demands had not been attended.

Despite that, many researcher were convinced by then of the potential of AI and kept working on it. By 2006 ANNs established the state of the art in many official international competitions in several domains, attracting again attention and fundings. Since then, they have been constantly the focus of study and innovation.

3.1.3.3 Perceptrons

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some

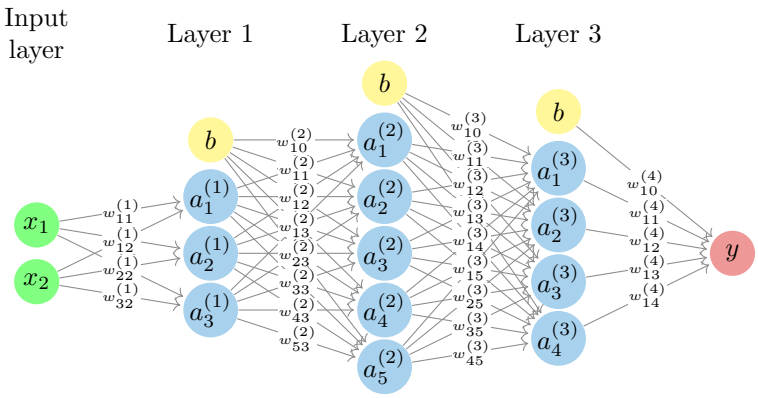


Figure 3.10: A MultiLayer Perceptron. The sum and the nonlinearity nodes have been omitted for the sake of clarity.

specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. Perceptron is an algorithm for learning a binary classifier called a threshold function: a function that maps its input x (a real-valued vector) to an output value f (x) (a single binary value):

In the context of neural networks, a perceptron is an artificial neuron using the Heaviside step function as the activation function.[Minsky 17]

3.1.3.4 Feedforward neural networks, or Multilayer perceptrons(MLPs)

Consider the network in Figure 3.10. As opposed to the Perceptron in Figure 3.8, the MLP has multiple hidden layers, where each neuron of one hidden layer is connected to all the neurons of the previous and the next layer. Each connection from the i-th neuron of layer $l-1$ to the j-th neuron of layer l is associated to a weight $w_{ij}^{(l)}$, and all the weights are stored in a matrix $\mathbf{W^{(l)}}$.

Similarly to the Perceptron case, each layer computes an affine transformation

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)}, \tag{3.5}$$

followed by a nonlinearity σ – usually more smooth than the one used in the Perceptron – called *activation function*

$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}). \tag{3.6}$$

One hidden layer can thus be seen as a function $h^{(l)}$ that depends on some *parameters* specific to that layer – namely a matrix of weights $\mathbf{w}^{(l)}$ and the bias vector $b^{(l)}$ – as well as some *hyperparameters*, such as the number of neurons and the choice of activation function. It is common to denote the set of trainable parameters that fully characterize a layer as $\boldsymbol{\theta^{(l)}}$.

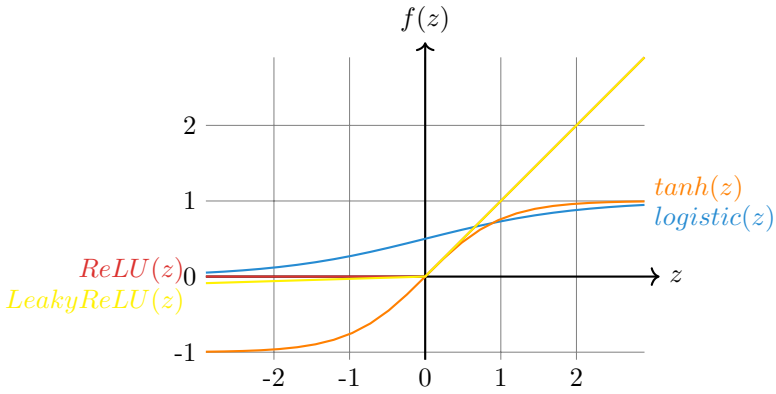


Figure 3.11: Some of the most common activation functions: sigmoid, tanh, ReLU and Leaky ReLU. ReLU and Leaky ReLU are overlapping for $z \geq 0$. Best viewed in colors.

The processing performed by the hierarchy of layers of the MLP results in the output vector $\mathbf{y}$ and is equivalent to a composition of multiple functions

$$\mathbf{y} = h_{\boldsymbol{\theta}}^{(L)} h_{\boldsymbol{\theta}}^{(L-1)} \dots h_{\boldsymbol{\theta}}^{(1)}. \tag{3.7}$$

Generally, each layer of an ANN computes some activation based on its input and a nonlinear activation function. The choice of which activation function to use in each layer can have a big impact on the performance of the model and is sometimes constrained by the semantic assigned to the output of some units.

3.1.3.5 Activation functions

The activation function is one of the most important component of an ANN. As explained in the previous sections, to tackle nonlinearly separable problems it is imperative to map the input into a space that is linearly separable. The activation function does this by performing an *element-wise nonlinear transformation* of the pre-activation that comes from the affine transformation.

The affine transformation and the nonlinearity work together in a very tight interaction: the latter is fixed and does not evolve during training, but maps its input to a different space in a (usually highly) non-linear way; the former, is determined by the weights that are learned during training and exploits the latter to map the incoming activation into a new space where they are easier to separate. In addition to this it is interesting to point out that if there was no activation function, since the composition of multiple affine transformation is an affine transformation, the layers of the MLP could be replaced by a single equivalent layer, and the MLP would become a Perceptron.

3.1.3.5.1 Logistic

The logistic, often called *sigmoid*, is a differentiable monotonically increasing function that takes any real-valued number and maps it to $[0, 1]$. As evident from its representation in Figure 3.11, for large negative numbers it asymptotes to 0 while for large positive numbers it asymptotes to 1. It is defined as

$$logistic(\mathbf{z}) = \frac{1}{1 + \exp(-\mathbf{z})}. \tag{3.8}$$

The logistic function has probably been the most used nonlinearity historically due to its possible interpretation as the producing rate of a neuron, or its probability to produce, given its potential (i.e., the level of excitement provoked by its incoming spikes): when the potential is low the neuron produces less often whereas when the potential is high the frequency of the spikes increases.

Another very important property of the logistic function is that it is very fast to compute its derivative, once solved analytically:

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{z}} logistic(\mathbf{z}) &= \frac{\exp(-\mathbf{z})}{(1 + \exp(-\mathbf{z}))^2}, \\
&= \frac{1}{1 + \exp(-\mathbf{z})} \cdot \frac{\exp(-\mathbf{z})}{1 + \exp(-\mathbf{z})}, \\
&= logistic(\mathbf{z}) \cdot \frac{\exp(-\mathbf{z})}{1 + \exp(-\mathbf{z})}, \\
&= logistic(\mathbf{z}) \cdot \frac{1 + \exp(-\mathbf{z}) - 1}{1 + \exp(-\mathbf{z})}, \\
&= logistic(\mathbf{z}) \cdot \left(1 - \frac{1}{1 + \exp(-\mathbf{z})}\right), \\
&= logistic(\mathbf{z}) \cdot (1 - logistic(\mathbf{z})).
\end{aligned} \tag{3.9}$$

The use of the logistic function is not as widespread as it used to be though, due to two major drawbacks:

- *Saturation kills the gradient:*
  backpropagation (see Section 3.1.3.6) relies on the gradient of the error to determine the parameters update. The logistic function saturates at both ends, resulting in a very small or zero gradient. This problem – often referred to as *vanishing gradient* – makes training very slow or prevents it in some cases. This also makes the logistic units very sensitive to the initialization of the weights of the network, that ideally should be such that the initial input of the logistic function is close to zero.
- *The output is not zero-centered:*
  the dynamics of ANNs are usually complex and difficult to inspect, but it is widely believed that normalizing the intermediate activations of the network (i.e., to zero-center their mean and impose unit variance) helps training[Ioffe 15, Laurent 15, Arpit 16, Cooijmans 16]. The output of the logistic function is always positive, which causes the mean activation

to be non-zero. This could introduce undesirable dynamics that could slow down or prevent training.

3.1.3.5.2 Hyperbolic tangent (tanh)

The hyperbolic tangent, typically shortened as *tanh*, is a differentiable monotonically increasing function that maps any real-valued number to $[-1, 1]$. This nonlinear function suffers from the same vanishing problem as the logistic, but its mean is centered in zero. Furthermore, the tanh is often chosen where it is desirable to be able to increase or decrease some quantity by small amounts, thanks to its codomain. It is defined as

$$tanh(\mathbf{z}) = \frac{1 - exp(-2\mathbf{z})}{1 + exp(-2\mathbf{z})}. \tag{3.10}$$

3.1.3.5.3 Rectified Linear Unit (ReLU)

Since its introduction, the Rectified Linear Unit (ReLu)[Jarrett 09, Nair 10] has become the nonlinearity of choice in many applications[Krizhevsky 12, LeCun 15, Glorot 11]. It is defined as

$$relu(\mathbf{z}) = max(0, \mathbf{z}). \tag{3.11}$$

Although very simple, it has some very interesting properties and only a few drawbacks:

- *No positive saturation:* the ReLU is zero for non-positive inputs, but does not saturate otherwise. This ensures a flow of gradient whenever the input is positive, that was found to significantly speed up the convergence of training.
- *Cheap to compute:* as opposed to many other activation functions that require expensive operations, such as for instance exponentials, ReLU's implementation simply amounts to thresholding at zero. Another important characteristic is that the gradient is trivial to compute:

$$\nabla(relu(\mathbf{z}^{(l)})) = \begin{cases} \mathbf{a}^{(l-1)}, & \text{if } \mathbf{z}^{(l)} > 0, \\ 0, & \text{if } \mathbf{z}^{(l)} < 0, \\ undefined, & \text{if } \mathbf{z}^{(l)} = 0. \end{cases} \tag{3.12}$$

- *Induce sparsity:* ReLU units induce sparsity, indeed whenever the input preactivation is negative their activation is zero. Sparsity is usually a desired property: as opposed to dense encoding, sparsity will produce representations where only a few entries change upon small variations of the input, i.e., it will produce a representation that is more consistent and robust to perturbations. Furthermore, sparsity allows for a more compact encoding, which is desirable in many contexts such as, e.g., data compression and efficient data transfer. Finally, it is also usually easier to linearly separate sparse representations[Glorot 11].

- *ReLU units can die:* when a large gradient flows through a ReLU unit it can change its weights in such a way that will prevent it from ever being active again. In this case every input will put the ReLU unit on the flat zero side. This will prevent any gradient flow and the unit will never leave this state becoming *de facto* "dead". This can be alleviated by using a lower learning rate or choosing some modification of ReLU less sensitive to this problem.

3.1.3.5.4 Leaky Rectified Linear Unit (Leaky ReLU)

Leaky ReLUs are one of the most adopted alternatives to ReLUs. They have been proposed as a way to alleviate the dying units problem of ReLUs, by preventing the unit from saturating thus allowing a small gradient to always flow through the unit, potentially recovering extreme values of the weights. Leaky ReLUs are defined as

$$leaky\_relu(\mathbf{z}) = max(\beta * \mathbf{z}, \mathbf{z}), \tag{3.13}$$

where β is a small constant.

3.1.3.5.5 Softmax

One peculiar nonlinearity that deserves a particular mention is the softmax. This function differs from the previously described activations in that it does not only depend on the value of the preactivation in one dimension (i.e., the value of one unit), but rather on the preactivation in all the dimensions altogether. The softmax is a *squashing function* that maps its input to a categorical distribution. It is defined as

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_{k=0}^{K} \exp(z_k)}, \tag{3.14}$$

where K is the number of classes, i.e., of dimensions (or neurons).
Note that it is possible to add a temperature parameter T to the softmax that allows it to control its steepness (see Figure 3.12), i.e., to control the randomness of predictions. When T is high the distribution over the classes will be more uniform - with the extreme case of $T = \inf$, where all the classes have equal probability. When T is low, the probability curve is more peacky on the class with higher probability and has a light tail on the other classes.

$$softmax(z_i) = \frac{\exp(z_i/T)}{\sum_{k=0}^{K} \exp(z_k/T)}. \tag{3.15}$$

3.1.3.6 Backpropagation

The learning rule originally introduced with Perceptrons did not allow one to train models with multiple layers (i.e., with *hidden* layers), such as the one depicted in Figure 3.10.

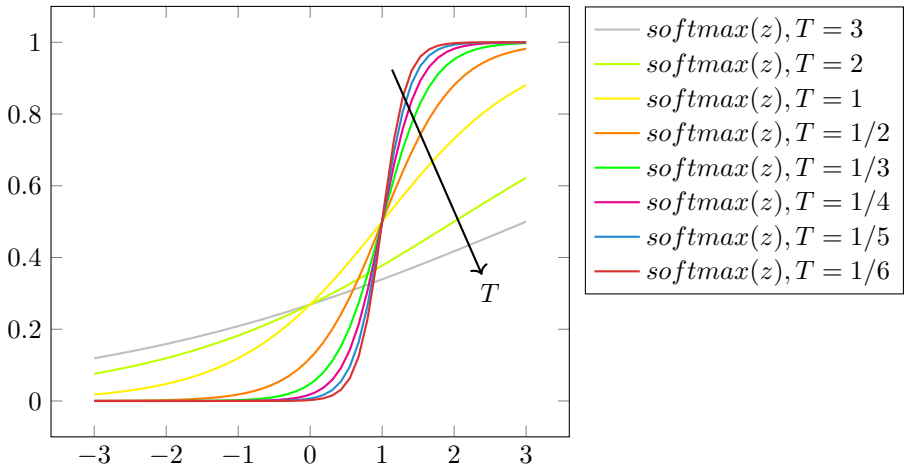


Figure 3.12: The behaviour of softmax as temperature T grows. The plots have been obtained considering a bidimensional input setting where the preactivation associated to the second class z_1 is always 1. As T decreases, the function becomes steeper.

This was not possible since to compute the variation of the weights of a layer with Equation 3.4 it is necessary to know the correct value of its output, which is given only for the last layer.

To address this obstacle it suffices to notice that the computation performed by each activation function is a nonlinear, but differentiable function of the inputs. This allows for the computation of the partial derivatives of the error (e.g., the expected value of the quadratic loss) with respect to the weights of the network. In other words, it is possible to use calculus to determine the amount by which each neuron of the last layer contributed to causing the error, and then further split the responsibility of each of them among the ones of the preceding layer. In this way, the error can be *propagated backwards* through the layers of the network, assigning to each weight its amount of blame. This information can be used by an *optimization algorithm* to iteratively change the weights to minimize the error.

The backpropagation algorithm[Rumelhart 86b] has some resemblance with the learning rule of the Perceptron (Equation 3.4). The main idea in that case was to modify each weight of the network by a factor proportional to the error ($E = \hat{y} - y$, in the Perceptron) and to the input. Even if in MLPs it is usually common to consider different kinds of cost functions, the same concept applies: the learning procedure tries to modify the weights in order to minimize some error

$$\mathbf{W}|_t = \mathbf{W}|_{t-1} - \eta \left. \frac{\partial E}{\partial \mathbf{W}} \right|_{t-1}, \tag{3.16}$$

where η is a scaling factor typically referred to as *learning rate* that determines the size of the gradient descent steps.

It is easier to understand backpropagation with a practical example. Consider once again Figure 3.10: the network processes a bidimensional input

$\mathbf{x}$ and, after three layers of affine transformations followed by nonlinearities, returns a scalar value $\hat{y}$. To keep the narrative as general as possible a 1×1 vector $\hat{\mathbf{y}}$ will be used to denote the output to encompass the case of multiple output units. The correct output $\mathbf{y}$ is given and is used to compute the error, or *cost*, with some *differentiable* metric. For this example consider the mean squared error (MSE)

$$E_{mse} = \frac{1}{M} \sum_{\mathcal{D}} \frac{1}{2} \left\| \mathbf{y} - \hat{\mathbf{y}} \right\|_2 , \tag{3.17}$$

the summation is done over a dataset $\mathcal{D}$ of M samples, each composed by an input $\mathbf{x}$ and its associated desired output $\mathbf{y}$. $\hat{\mathbf{y}}$ is used as a compact notation for $\hat{\mathbf{y}}(\mathbf{x})$ and represents the output of the network for one input sample $\mathbf{x}$. It is possible to compute the fraction of the error that can be attributed to each weight of the network by taking the derivative of the error with respect to it

$$\begin{aligned} \frac{\partial E_{mse}}{\partial w_{ij}^{(l)}} &= \frac{\partial}{\partial w_{ij}^{(l)}} \left(\frac{1}{m} \sum_{\mathcal{D}} \left[\frac{1}{2} \left\| \mathbf{y} - \hat{\mathbf{y}} \right\|_2 \right] \right), \\ &= \frac{1}{m} \sum_{\mathcal{D}} \left[\frac{1}{2} \frac{\partial}{\partial w_{ij}^{(l)}} \left[\left(\mathbf{y} - \hat{\mathbf{y}} \right)^T \left(\mathbf{y} - \hat{\mathbf{y}} \right) \right] \right], \\ &= \frac{1}{m} \sum_{\mathcal{D}} \left[\left(\mathbf{y} - \hat{\mathbf{y}} \right)^T \frac{\partial (-\hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} \right]. \end{aligned} \tag{3.18}$$

Backpropagation allows one to compute the partial derivative $-\frac{\partial \hat{\mathbf{y}}}{\partial w_{ij}^{(l)}}$ exploiting the chain rule of derivation. Consider, e.g., one weight of the second layer $w_{ij}^{(2)}$

$$\begin{aligned} -\frac{\partial \hat{\mathbf{y}}}{\partial w_{ij}^{(2)}} &= -\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(4)}} \cdot \frac{\partial \mathbf{z}^{(4)}}{\partial w_{ij}^{(2)}}, \\ &= -\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(4)}} \cdot \frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}^{(3)}} \cdot \frac{\partial \mathbf{a}^{(3)}}{\partial w_{ij}^{(2)}}, \\ &= -\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(4)}} \cdot \frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}^{(3)}} \cdot \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{z}^{(3)}} \cdot \frac{\partial \mathbf{z}^{(3)}}{\partial w_{ij}^{(2)}}, \\ &= -\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(4)}} \cdot \frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}^{(3)}} \cdot \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{z}^{(3)}} \cdot \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{a}^{(2)}} \cdot \frac{\partial \mathbf{a}^{(2)}}{\partial w_{ij}^{(2)}}, \\ &= -\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(4)}} \cdot \frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}^{(3)}} \cdot \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{z}^{(3)}} \cdot \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{a}^{(2)}} \cdot \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} \cdot \frac{\partial \mathbf{z}^{(2)}}{\partial w_{ij}^{(2)}}. \end{aligned} \tag{3.19}$$

Replacing $\hat{\mathbf{y}}$ with the corresponding activation (i.e., $\mathbf{a}^{(4)}$ in this example), from Equation 3.5 and Equation 3.6 follows

$$-\frac{\partial \hat{\mathbf{y}}}{\partial w_{ij}^{(2)}} = -\frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{z}^{(4)}} \cdot \mathbf{W}^{(4)} \cdot \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{z}^{(3)}} \cdot \mathbf{W}^{(3)} \cdot \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} \cdot [a_j^{(1)}]_i, \tag{3.20}$$

with $[a_j^{(1)}]_i$ a vector of the same size as $\mathbf{z}^{(2)}$ whose i-th element is equal to $a_j^{(1)}$ and all the other elements are 0. Each derivative $\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}}$ corresponds to an $n \times n$ matrix

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \begin{bmatrix} \frac{\partial \sigma_0(\mathbf{z}^{(l)})}{\partial z_0^{(l)}} & \dots & \frac{\partial \sigma_0(\mathbf{z}^{(l)})}{\partial z_n^{(l)}}, \\ \vdots & \ddots & \vdots, \\ \frac{\partial \sigma_n(\mathbf{z}^{(l)})}{\partial z_0^{(l)}} & \dots & \frac{\partial \sigma_n(\mathbf{z}^{(l)})}{\partial z_n^{(l)}} \end{bmatrix}, \tag{3.21}$$

with n the cardinality of $\mathbf{z}^{(l)}$ and $\mathbf{a}^{(l)}$. Note that the derivative of the nonlinear function of choice can be computed analytically as done in Equation 3.9 and Equation 3.12 and applied directly in the formula. Interestingly, when σ is a function applied elementwise (e.g. in the case of logistic, tanh and ReLU activations), $\sigma_i(\mathbf{z}^{(l)}) = \sigma(z_i^{(l)})$, i.e. $\sigma_i(\mathbf{z}^{(l)})$ only depends on the i-th element of z. When this is the case Equation 3.21 is a diagonal matrix

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \begin{bmatrix} \sigma'(z_0^{(l)}) & & 0, \\ & \ddots & , \\ 0 & & \sigma'(z_n^{(l)}) \end{bmatrix}, \tag{3.22}$$

which can be exploited to speed up computation.

3.1.3.7 Error or Loss function

A neural network learns to map a set of inputs to a set of outputs from training data.

Calculate the perfect weights for a neural network is comprehensive task; there are too many unknowns. Instead, the problem of learning is cast as a search or optimization problem and an algorithm is used to navigate the space of possible sets of weights the model may use in order to make good or good enough predictions.

Neural network model is trained using the stochastic gradient descent optimization algorithm and weights are updated using the backpropagation of error algorithm.[Tan 12]

The gradient in gradient descent refers to an error gradient. The model with a given set of weights is used to make predictions and the error for those predictions is calculated.

The gradient descent algorithm seeks to change the weights so that the next evaluation reduces the error, meaning the optimization algorithm is navigating down the gradient (or slope) of error.

In the context of an optimization algorithm, the function used to evaluate a candidate solution (i.e. a set of weights) is referred to as the objective

function. To maximize or minimize the objective function, meaning that we are searching for a candidate solution that has the highest or lowest score respectively.

With neural networks, mostly to deal with to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply loss (The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function) [LeCun 15, Page 82]. The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared. In calculating the error of the model during the optimization process, a loss function must be chosen [Reed 99, Page 155].

3.1.3.7.1 Types of Loss Function

Begins with several types:[Typ][Murphy 12]

i. Regression Loss Functions
   A. Mean Squared Error Loss
   B. Mean Squared Logarithmic Error Loss
   C. Mean Absolute Error Loss
   D. Huber Loss, Smooth Mean Absolute Error
ii. Binary Classification Loss Functions
   A. Binary Cross-Entropy
   B. Hinge Loss
   C. Squared Hinge Loss
iii. Multi-Class Classification Loss Functions
   A. Multi-Class Cross-Entropy Loss
   B. Sparse Multiclass Cross-Entropy Loss
   C. Kullback Leibler Divergence Loss
iv. Embedding loss function
   A. Poisson
   B. Cosine Error

3.1.3.7.2 Mean Squared Error Loss

In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errorsthat is, the average squared difference between the estimated values and what is estimated. Mathematically, the loss is the mean overseen data of the squared differences between true and predicted values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y_i})^2 \tag{3.23}$$

here:
Y : Actual value
$\bar{Y_i}$: Predicted value
n : Number of data point

3.1.3.7.3 Root Mean Squared Logarithmic Error Loss

Root Mean squared logarithmic error (RMSLE) can be interpreted as a measure of the ratio between the true and predicted values. Root Mean Squared Logarithmic Error Loss only care about the percentual difference.

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2} \tag{3.24}$$

here:
ϵ = RMSLE value
n = total number of observations in the data set
p_i = the prediction
a_i = actural response for i
$i\log(x)$ = natural logarithm of x

3.1.3.7.4 Mean Absolute Error Loss

In statistics, mean absolute error (MAE) is a measure of difference between two continuous variables. The loss is the mean overseen data of the absolute differences between true and predicted values.

$$\begin{aligned} \text{MAE} &= \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} \\ &= \frac{\sum_{i=1}^{n}|e_i|}{n} \end{aligned} \tag{3.25}$$

here:
y_i : Actual value
x_i : Predicted Value

3.1.3.7.5 Huber Loss, Smooth Mean Absolute Error

Huber loss is less sensitive to outliers in data than the squared error loss. Its also differentiable at 0. Its basically absolute error, which becomes quadratic when error is small. How small that error has to be to make it quadratic depends on a hyperparameter, (delta), which can be tuned.

$$\mathrm{L}_\delta(a) = \begin{cases} \frac{1}{2}a^2, & \text{for } |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta^2), & \text{otherwise} \end{cases}$$

3.1.3.7.6 Binary Cross-Entropy

Cross-entropy is the default loss function to use for binary classification problems.

It is intended for use with binary classification where the target values are in the set 0, 1.
Mathematically, it is the preferred loss function under the inference framework of maximum likelihood. It is the loss function to be evaluated first and only changed if you have a good reason.
Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1. The score is minimized and a perfect cross-entropy value is 0.

$$\iota(\bar{y}, y) = -\frac{1}{N}\sum_{i}^{N}[y_i \log \bar{y}_i + (1 - y_i)\log(1 - \bar{y}_i))] \tag{3.26}$$

here:
y : Actual Value
y_i : Predicted Value
N : Total number of datapoints.

3.1.3.7.7 Hinge Loss

Primarily developed for use with Support Vector Machine (SVM) models.
It is intended for use with binary classification where the target values are in the set -1, 1.
The hinge loss function encourages examples to have the correct sign, assigning more error when there is a difference in the sign between the actual and predicted class values.
Reports of performance with the hinge loss are mixed, sometimes resulting in better performance than cross-entropy on binary classification problems.

$$\iota(y) = max(0, 1 - t \times y) \tag{3.27}$$

3.1.3.7.8 Squared Hinge Loss

The squared hinge loss is a loss function used for maximum margin binary classification problems.

$$\iota(y, \bar{y}) = \sum_{i=0}^{N}(max(0, 1 - y_i \times \bar{y}_i)^2) \tag{3.28}$$

3.1.3.7.9 Multi-Class Categorical Cross-Entropy Loss

Cross-entropy is the default loss function to use for multi-class classification problems.
In this case, it is intended for use with multi-class classification where the target values are in the set 0, 1, 3, , n, where each class is assigned a unique integer value.
Mathematically, it is the preferred loss function under the inference framework of maximum likelihood. It is the loss function to be evaluated first and only changed if you have a good reason.

Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for all classes in the problem. The score is minimized and a perfect cross-entropy value is 0.

The cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "artificial" probability distribution q, rather than the "true" distribution p.

$$\text{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} 1_{y_i \in C_c} \log p_{model}\left[y_i \in C_c\right] \tag{3.29}$$

here:
N : Numer of datasets
C : Total number of Classes
p_{model} : Probabilit predicted by the model for the ith observation to belong to the cth category

The double sum is over the observations i, whose number is N, and the categories c, whose number is C. The term $1_{y_i \in C_c}$ is the indicator function of the i^{th} observation belonging to the c^{th} category. The $p_{model}[y_i \in C_c]$ is the probability predicted by the model for the i^{th} observation to belong to the c^{th} category.

3.1.3.7.10 Sparse Multiclass Cross-Entropy Loss

The only difference between sparse categorical cross entropy and categorical cross entropy is the format of true labels. When we have a single-label, multi-class classification problem, the labels are mutually exclusive for each data, meaning each data entry can only belong to one class. Then we can represent $\text{y}_t rueusingone - hotembeddings$.

3.1.3.7.11 Kullback Leibler Divergence Loss

Kullback Leibler Divergence, or KL Divergence for short, is a measure of how one probability distribution differs from a baseline distribution.

A KL divergence loss of 0 suggests the distributions are identical. In practice, the behavior of KL Divergence is very similar to cross-entropy. It calculates how much information is lost (in terms of bits) if the predicted probability distribution is used to approximate the desired target probability distribution.

As such, the KL divergence loss function is more commonly used when using models that learn to approximate a more complex function than simply multi-class classification, such as in the case of an autoencoder used for learning a dense feature representation under a model that must reconstruct the original input.

$$\text{D}_{KL}(P \,||\, Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{3.30}$$

here:
D_{KL} : Notation of Kullback leibler divergence

In other words, it is the expectation of the logarithmic difference between the probabilities P and Q, where the expectation is taken using the probabilities P. The Kullback-Leibler divergence is defined only if for all x, Q(x)=0 Q(x)=0 implies P(x)=0 P(x)=0 (absolute continuity). Whenever P(x) is zero the contribution of the corresponding term is interpreted as zero.

3.1.3.7.12 Poisson

Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant rate and independently of the time since the last event.

$$\log(E(Y\,|x)) = \alpha + \beta' x \tag{3.31}$$

3.1.3.7.13 Cosine Error

Computes the cosine similarity between labels and predictions.

$$\begin{aligned} \cos(\theta) &= \frac{A \cdot B}{||\,A\,||\,||\,B\,||} \\ &= \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \end{aligned} \tag{3.32}$$

3.1.3.8 Optimisation Functions

First order optimisation functions are:[Agrawal]

i. Stochastic Gradient Decent
ii. Adagrad
iii. Adam

Stochastic Gradient Decent was much faster than the other algorithms but the results produced were far from optimum. Both, Adagrad and Adam produced better results that SGD, but they were computationally extensive. Adam was slightly faster than Adagrad. Thus, while using a particular optimization function, one has to make a trade off between more computation power and more optimum results.

3.1.3.8.1 Stochastic Gradient Decent

Gradient Descent calcultes gradient for the whole dataset and updates values in direction opposite to the gradients until we find a local minima. Stochastic Gradient Descent performs a parameter update for each training example unlike normal Gradient Descent which performs only one update. Thus it is much faster. Gradient Decent algorithms can further be improved by tuning important parametes like momentum, learning rate etc.

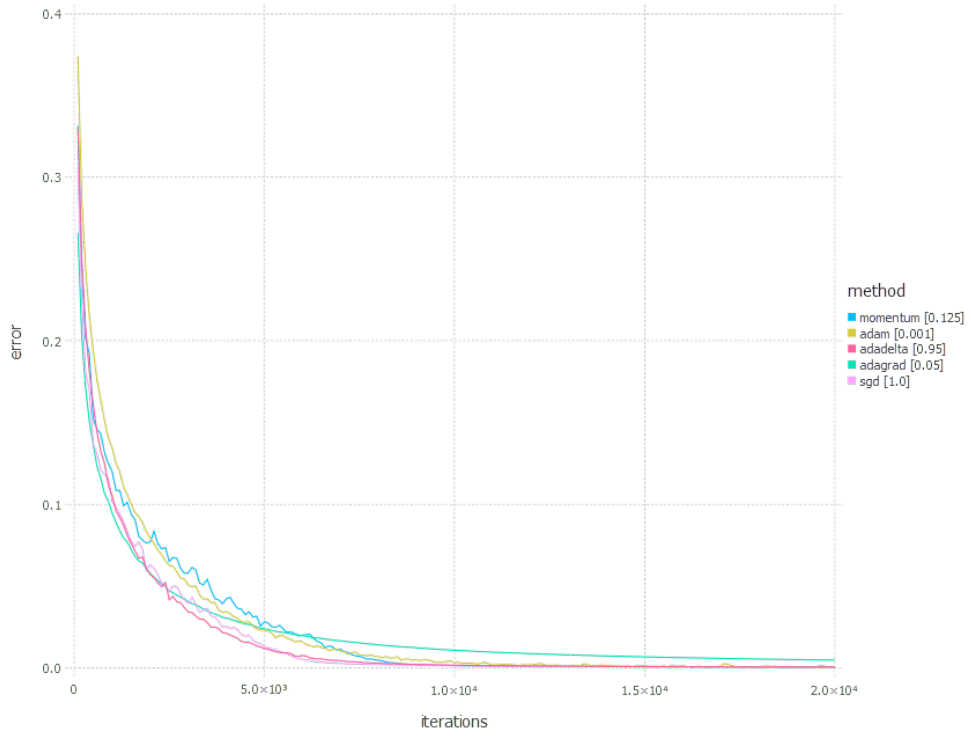


Figure 3.13: Choosing optimiser and comparison

3.1.3.8.2 Adagrad

Adagrad is more preferrable for a sparse data set as it makes big updates for infrequent parameters and small updates for frequent parameters. It uses a different learning Rate for every parameter at a time step based on the past gradients which were computed for that parameter. Thus we do not need to manually tune the learning rate.

3.1.3.8.3 Adam

Adam stands for Adaptive Moment Estimation. It also calculates different learning rate. Adam works well in practice, is faster, and outperforms other techniques.

3.2 Genetic Algorithm: Search and Operational Research

In computer science and operations research, a genetic algorithm (GA) is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.[Mitchell 98] John Holland introduced genetic algorithms in 1960 based on the concept of Darwins theory of evolution; afterwards, his student David E. Goldberg extended GA in 1989.

3.2.0.1 Evolutionary Algorithms & Biological Inspiration

Evolutionary algorithms are a set of generic meta-heuristic algorithms which solve optimisation problems by imitating aspects of biological evolution.Genetic algorithms are a subset of Evolutionary algorithms inspired by Charles Darwin's work on evolution by natural selection.

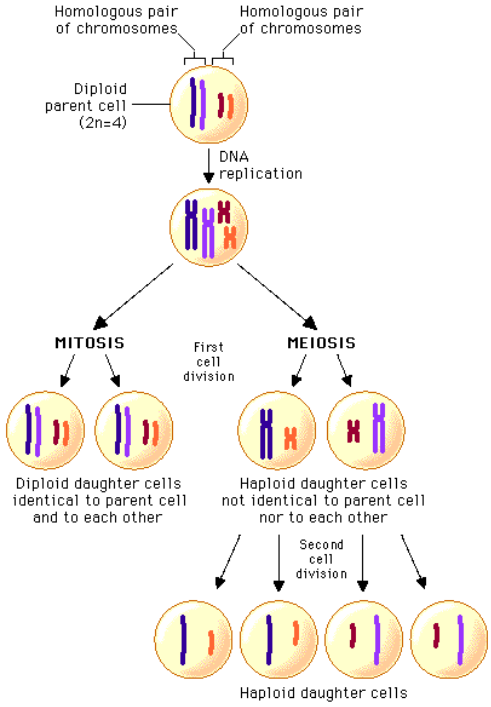


Figure 3.14: Meiosis and Mitosis Process

The above picture depicts the process of Meiosis and Mitosis where in the bottom you have crossovers of chromosomes forming new genotypes and the latter involves copying of genetic information to new offspring.The top figure is biological phenomenon of mutation can be thought of as errors during the copying process.Down the lane we can observe that the mutation,crossover and selection of encoded bit strings of features of the data are analogous to these biological operations to a certain extent.

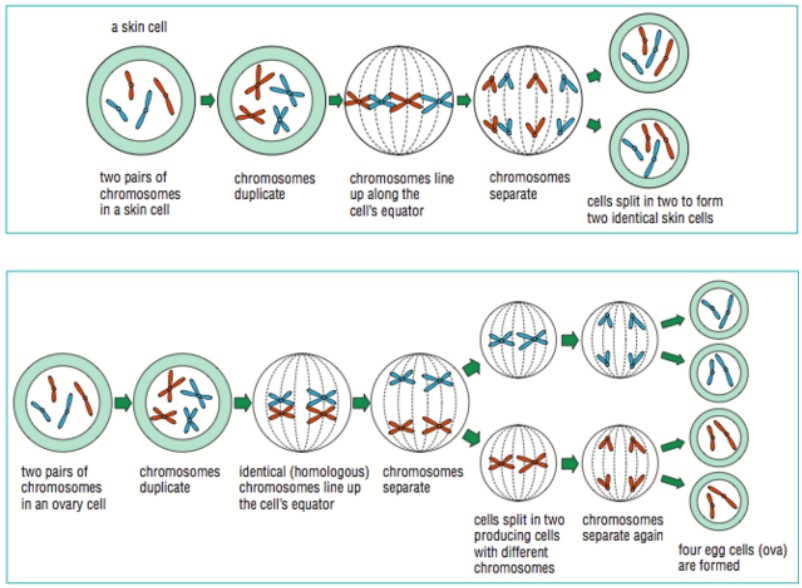
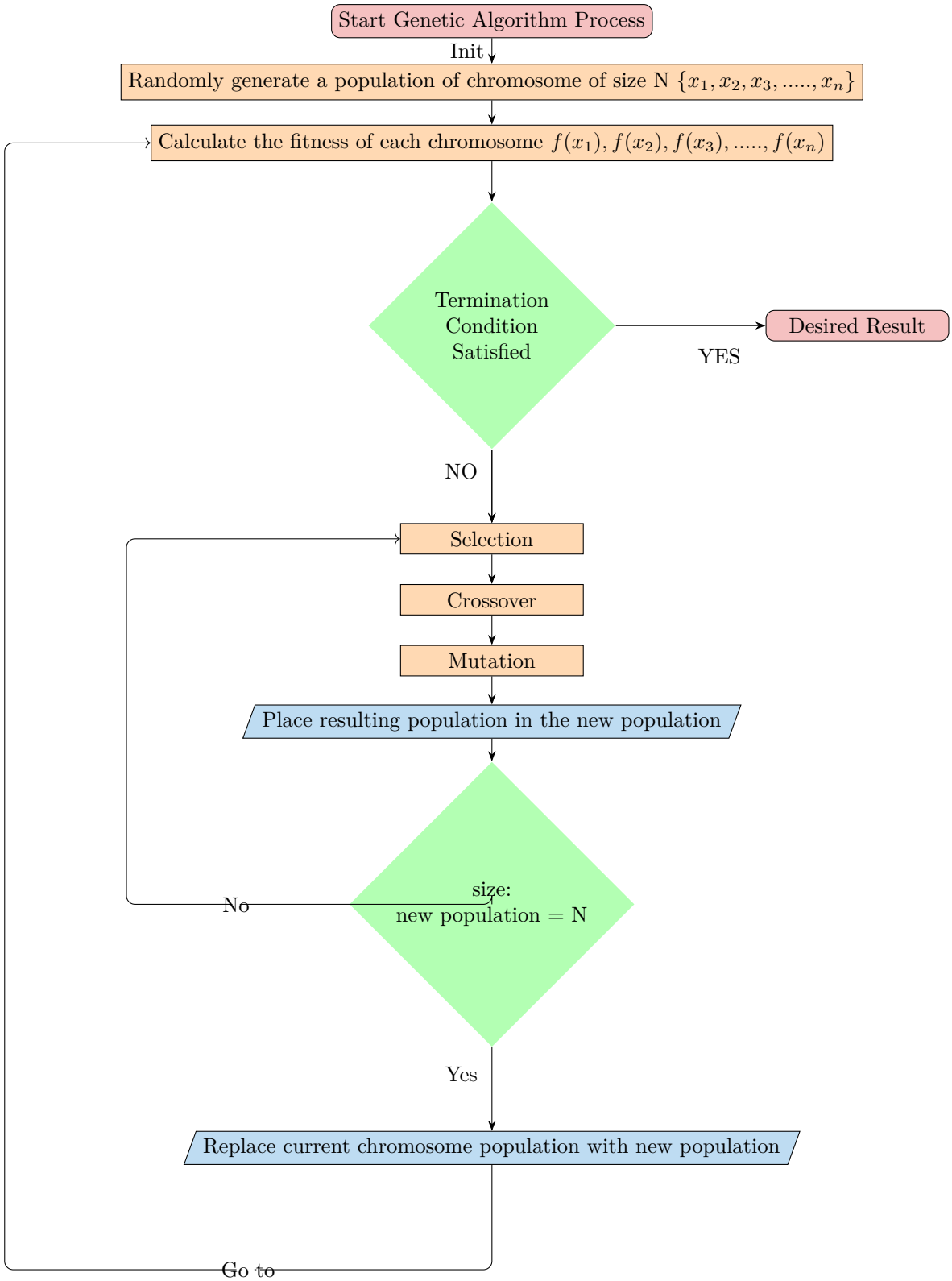


Figure 3.15: Brief outlook of Meiosis and Mitosis

3.2.0.2 Detailing Genetic Algorithm



3.2.0.2.1 Key terms of Genetic Algorithm

3.2.0.2.1.1 chromosome Genetic Algorithm belongs to a class of stochastic search method based on biological evolution. They represent highly parallel adaptive search process. Central theme of the genetic algorithm is a population where individuals in the population represent possible solutions. An individual is called chromosome, an analogy with the genetic chromosome which is usually represented by a bit string consisting 0s and 1s.

3.2.0.2.1.2 Population Population(P)- It is a set of initial Hypotheses which undergo operations every generation to give rise to a probably better(evolved) population.The size of population is constant every generation.

3.2.0.2.1.3 Generation Refers to the number of iterations of the algorithm (Stages of population modification).That is, first iteration results in the formation of 1st generation.

New population is generated from old population with three basic genetic operators: selection or reproduction, crossover, and mutation.

3.2.0.2.1.4 Selection Process of probabilistically selecting n(1-r) members out of a population(p) whose size is n to be added to new population(P) whose size is also n .Probability of selecting a hypotheses is given by:

$$\text{select}(\text{chromosome}_i) = \frac{fitness(chromosome_i)}{fitness(\sum_{j=1}^{n} chromosome_i)} \tag{3.33}$$

Most widely selection procedure is roulette wheel selection.

3.2.0.2.1.5 Fitness function Fitness function is a criterion used to decide selection of hypothesis for further generations(or operations).

3.2.0.2.1.6 Crossover(Analogous to meiosis operation) In genetic algorithms and evolutionary computation, crossover, also called recombination, is a genetic operator used to combine the genetic information of two parents to generate new offspring. It is one way to stochastically generate new solutions from an existing population, and analogous to the crossover. The Crossover operator randomly chooses a crossover point where two parent chromosomes break and then exchange the chromosome parts after that point. If the pair of chromosome does not crossover, then the choromosome cloning takes place and the offspring are created as exact copies of each parent. A value of 0.7 for the crossover probability generally produces great results.

3.2.0.2.1.7 Mutation(Analogous to biological Mutation) Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. Mutation, which is rare in nature, represents a change in the gene. This may lead to significant improvement in fitness, but often has rather destructive results. The mutation operator flips a randomly selected

gene in a chromosome. Mutation can occur at any gene in a chromosome with some probability. The mutation probability is quite small in nature and in the range of 0.0001 and 0.01.

The use mutation is technically to provide a guarantee that the search algorithm is not trapped on a local optimum. The sequence of selection and crossover operations may create any homogenous set of solutions. Under such conditions, all chromosomes are identical. Mutation provides loss of genetic diversity.

3.2.0.2.1.8 Advantages of Genetic Algorithm

- Inherently parallel
- Good in noisy environments
- Gives a set of possible or close by Solution set.
- Easy to understand.

3.2.0.2.1.9 Disadvantages of Genetic Algorithm

- They don't scale well with complexity.
- They may not converge to a perfect solution.

3.3 Software Engineering and Development

3.3.1 Definition

3.3.1.1 Software and Software Engineering

3.3.1.1.1 Software

Software is:

i. instructions (computer programs) that when executed provide desired features, function, and performance;
ii. data structures that enable the programs to adequately manipulate information, and
iii. descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

3.3.1.1.2 Charactetistics of Software

i. Software is developed or engineered; it is not manufactured in the classical sense.
ii. Software doesnt wear out, but it does deteriorate.
iii. Although the industry is moving toward component-based construction, most software continues to be custom built.

3.3.1.1.3 Software Application Domain

Today, seven broad categories of computer software present continuing challenges for software engineers:

3.3.1.1.3.1 System software

A collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) processes complex, but determinate,4 information structures. Other systems applications (e.g., operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data.

In either case, the systems software area is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces.

3.3.1.1.3.2 Application software

Stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making. In addition to conventional data processing applications, application software is used to control business functions in real time (e.g., point-of-sale transaction processing, real-time manufacturing process control).

3.3.1.1.3.3 Engineering/scientific software

Has been characterized by number crunching algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. However, modern applications within the engineering/scientific area are moving away from conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

3.3.1.1.3.4 Embedded software

Resides within a product or system and is used to implement and control features and functions for the end user and for the system itself. Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

3.3.1.1.3.5 Product-line software

Designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

3.3.1.1.3.6 Web applications

Called "WebApps," this network-centric software category spans a wide array of applications. In their simplest form, WebApps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as Web 2.0 emerges, WebApps are evolving into sophisticated computing environments that not only provide stand-alone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

3.3.1.1.3.7 Artificial intelligence software

Makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

3.3.1.1.3.8 Open-world computing

The rapid growth of wireless networking may soon lead to true pervasive, distributed computing. The challenge for software engineers will be to develop systems and application software that will allow mobile devices, personal computers, and enterprise systems to communicate across vast networks.

- Netsourcing: the World Wide Web is rapidly becoming a computing engine as well as a content provider. The challenge for software engineers is to architect simple (e.g., personal financial planning) and sophisticated applications that provide a benefit to targeted end-user markets worldwide.

- Open source: a growing trend that results in distribution of source code for systems applications (e.g., operating systems, database, and development environments) so that many people can contribute to its development. The challenge for software engineers is to build source code that is self-descriptive, but more importantly, to develop techniques that will enable both customers and developers to know what changes have been made and how those changes manifest themselves within the software.

3.3.1.1.3.9 Legacy software

In computing, a legacy system is an old method, technology, computer system, or application program, "of, relating to, or being a previous or outdated computer system," yet still in use. Often referencing a system as "legacy" means that it paved the way for the standards that would follow it. This can also imply that the system is out of date or in need of replacement.

3.3.1.1.4 Software Engineering

A definition proposed by Fritz Bauer[NAT] at the seminal conference on the subject still serves as a basis for discussion:

[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

The IEEE has developed a more comprehensive definition when it states:

Software Engineering:

i. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

ii. The study of approaches as in above number 1.

Software engineering is a layered technology:

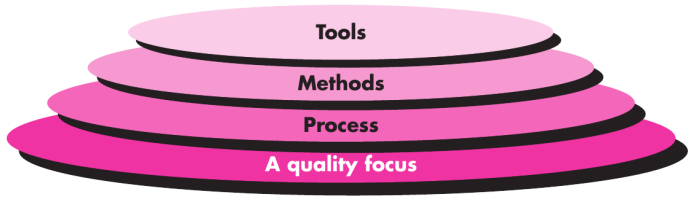


Figure 3.16: Software Engineering Layers

i. Any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management, Six Sigma, and similar philosophies foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering. The bedrock that supports software engineering is a quality focus.

ii. The foundation for software engineering is the process layer. The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework that must be established for effective delivery of software engineering technology. The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed. A generic process framework for software engineering encompasses five activities:

A. Communication
B. Planning
C. Modeling
D. Construction
E. Deployment

Software engineering process framework activities are complemented by a number of umbrella activities. In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

- Software project tracking and control: allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- Risk management: assesses risks that may affect the outcome of the project or the quality of the product.
- Software quality assurance: defines and conducts the activities required to ensure software quality.
- Technical reviews: assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- Measurement: defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders needs; can be used in conjunction with all other framework and umbrella activities.
- Software configuration management: manages the effects of change throughout the software process.
- Reusability management: defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- Work product preparation and production: encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

iii. Software engineering methods provide the technical how-to's for building software. Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support. Software engineering methods rely on

a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

iv. Software engineering tools provide automated or semiautomated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

3.3.1.1.5 Software Engineering practice and principles

3.3.1.1.5.1 Software Engineering practice

In a classic book, How to Solve It, written before modern computers existed, George Polya outlined the essence of problem solving, and consequently, the essence of software engineering practice:

i. Understand the problem (communication and analysis).
ii. Plan a solution (modeling and software design).
iii. Carry out the plan (code generation).
iv. Examine the result for accuracy (testing and quality assurance).

3.3.1.1.5.2 Software Engineering principles

David Hooker has proposed seven principles that focus on software engineering practice as a whole. They are reproduced in the following paragraphs:

i. The First Principle: The Reason It All Exists
A software system exists for one reason: to provide value to its users. All decisions should be made with this in mind. Before specifying a system requirement, before noting a piece of system functionality, before determining the hardware platforms or development processes, ask yourself questions such as: Does this add real value to the system? If the answer is no, don't do it. All other principles support this one.

ii. The Second Principle: KISS (Keep It Simple, Stupid!)
Software design is not a haphazard process. There are many factors to consider in any design effort. All design should be as simple as possible, but no simpler. This facilitates having a more easily understood and easily maintained system. This is not to say that features, even internal features, should be discarded in the name of simplicity. Indeed, the more elegant designs are usually the more simple ones. Simple also does not mean quick and dirty. In fact, it often takes a lot of thought and work over multiple iterations to simplify. The payoff is software that is more maintainable and less error-prone.

iii. The Third Principle: Maintain the Vision
A clear vision is essential to the success of a software project. Without one, a project almost unfailingly ends up being of two [or more] minds about itself. Without conceptual integrity, a system threatens to become a patchwork of incompatible designs, held together by the wrong

kind of screws. Compromising the architectural vision of a software system weakens and will eventually break even the well-designed systems. Having an empowered architect who can hold the vision and enforce compliance helps ensure a very successful software project.

iv. The Fourth Principle: What You Produce, Others Will Consume
Seldom is an industrial-strength software system constructed and used in a vacuum. In some way or other, someone else will use, maintain, document, or otherwise depend on being able to understand your system. So, always specify, design, and implement knowing someone else will have to understand what you are doing. The audience for any product of software development is potentially large. Specify with an eye to the users. Design, keeping the implementers in mind. Code with concern for those that must maintain and extend the system. Someone may have to debug the code you write, and that makes them a user of your code. Making their job easier adds value to the system.

v. The Fifth Principle: Be Open to the Future
A system with a long lifetime has more value. In todays computing environments, where specifications change on a moments notice and hardware platforms are obsolete just a few months old, software lifetimes are typically measured in months instead of years. However, true industrial-strength software systems must endure far longer. To do this successfully, these systems must be ready to adapt to these and other changes. Systems that do this successfully are those that have been designed this way from the start. Never design yourself into a corner. Always ask what if, and prepare for all possible answers by creating systems that solve the general problem, not just the specific one.14 This could very possibly lead to the reuse of an entire system.

vi. The Sixth Principle: Plan Ahead for Reuse
Reuse saves time and effort.15Achieving a high level of reuse is arguably the hardest goal to accomplish in developing a software system. The reuse of code and designs has been proclaimed as a major benefit of using object-oriented technologies. However, the return on this investment is not automatic. To leverage the reuse possibilities that object-oriented [or conventional] programming provides requires forethought and planning. There are many techniques to realize reuse at every level of the system development process. . . . Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

vii. The Seventh principle: Think!
This last principle is probably the most overlooked. Placing clear, complete thought before action almost always produces better results. When you think about something, you are more likely to do it right. You also gain knowledge about how to do it right again. If you do think about something and still do it wrong, it becomes a valuable experience. A side effect of thinking is learning to recognize when you dont know something, at which point you can research the answer. When clear thought has gone into a system, value comes out. Applying the first six principles requires intense thought, for which the potential rewards are

enormous.

3.3.2 Engineering Process and Development

3.3.2.1 Software Process Models

The software process is represented schematically in the following figure. Referring to the figure, each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress.

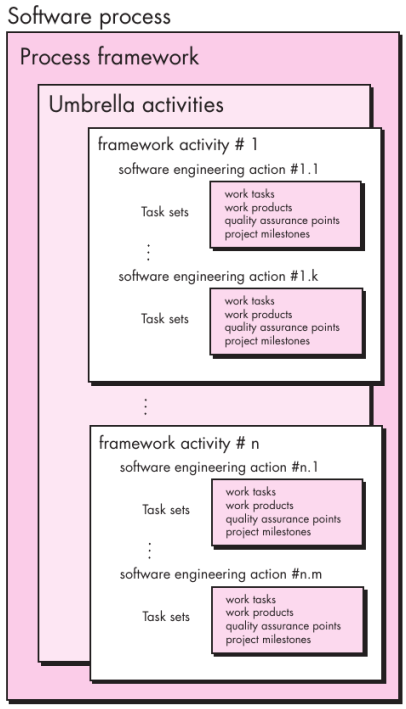


Figure 3.17: Software Process Framework

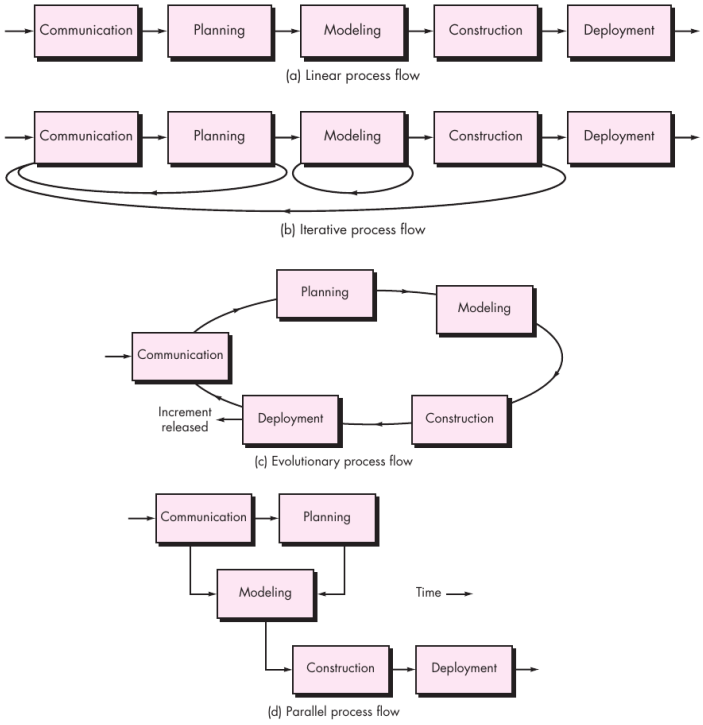


Figure 3.18: Software Process Flow

If the project was considerably more complex with many stakeholders, each with a different set of (sometime conflicting) requirements, the communication activity might have six distinct actions: inception, elicitation, elaboration, negotiation, specification, and validation. Each of these software engineering actions would have many work tasks and a number of distinct work products.

3.3.2.1.1 Process pattern

Ambler has proposed a template for describing a process pattern:

i. Pattern Name:
The pattern is given a meaningful name describing it within the context of the software process (e.g., TechnicalReviews).

ii. Forces:
The environment in which the pattern is encountered and the issues that make the problem visible and may affect its solution.

iii. Type
The pattern type is specified. Ambler suggests three types:

A. Stage pattern:
Defines a problem associated with a framework activity for the process. Since a framework activity encompasses multiple actions and work tasks, a stage pattern incorporates multiple task patterns that are relevant to the stage (framework activity). An example of a stage pattern might be EstablishingCommunication. This pattern would incorporate the task pattern RequirementsGathering and others.

B. Task pattern
Defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., RequirementsGathering is a task pattern).

C. Phase pattern
Define the sequence of framework activities that occurs within the process, even when the overall flow of activities is iterative in nature. An example of a phase pattern might be SpiralModel or Prototyping.

iv. Initial context
Describes the conditions under which the pattern applies. Prior to the initiation of the pattern: (1) What organizational or team-related activities have already occurred? (2) What is the entry state for the process? (3) What software engineering information or project information already exists?

v. Problem
The specific problem to be solved by the pattern.

vi. Solution
Describes how to implement the pattern successfully. This section describes how the initial state of the process (that exists before the pattern is implemented) is modified as a consequence of the initiation of the pattern. It also describes how software engineering information or project information that is available before the initiation of the pattern is transformed as a consequence of the successful execution of the pattern.

vii. Resulting Context
Describes the conditions that will result once the pattern has been successfully implemented. Upon completion of the pattern: (1) What organizational or team-related activities must have occurred? (2) What is the exit state for the process? (3) What software engineering information or project information has been developed?

viii. Related Patterns
Provide a list of all process patterns that are directly related to this one. This may be represented as a hierarchy or in some other diagrammatic form. For example, the stage pattern Communication encompasses the task patterns: ProjectTeam, CollaborativeGuidelines, ScopeIsolation, RequirementsGathering, ConstraintDescription, and ScenarioCreation.

ix. Known Uses and Examples
Indicate the specific instances in which the pattern is applicable. For example, Communication is mandatory at the beginning of every software project, is recommended throughout the software project, and is mandatory once the deployment activity is under way.

3.3.2.1.2 Prescriptive Process Models

Prescriptive process models define a prescribed set of process elements and a predictable process work flow.

i. Waterfall Model

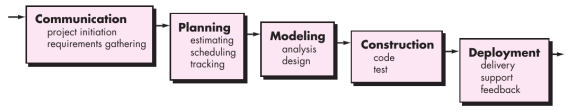


Figure 3.19: Waterfall model

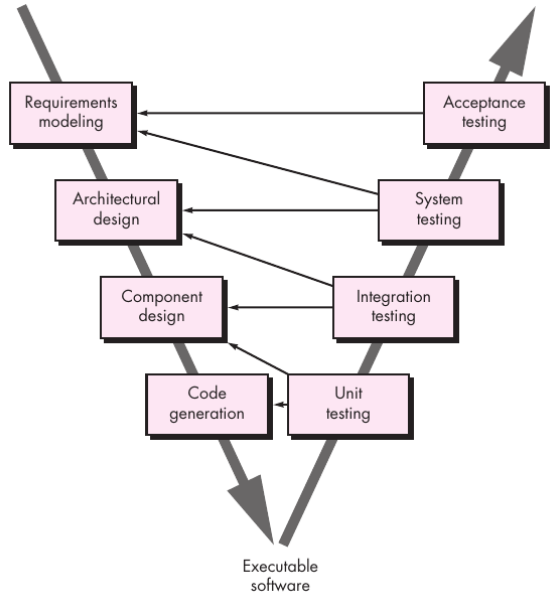


Figure 3.20: Waterfall V model

ii. Incremental Process Models
The incremental model combines elements of linear and parallel process flows. increment, until the complete product is produced. The incremental process model focuses on the delivery of an operational product with each increment.

iii. Evolutionary Process Models
Evolutionary models are iterative.

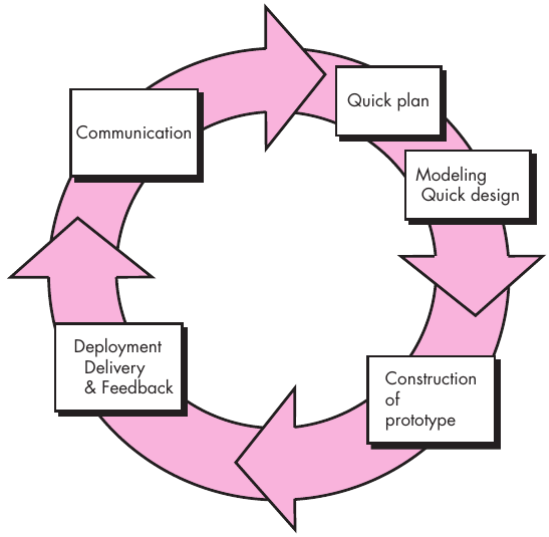


Figure 3.21: Prototyping Model

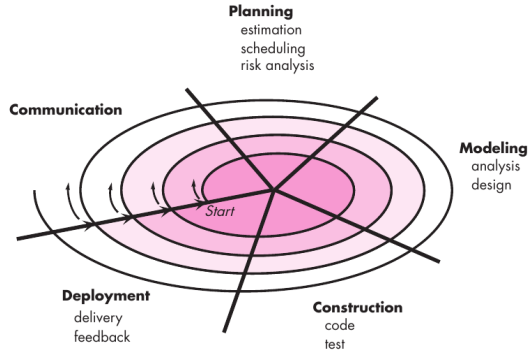


Figure 3.22: Spiral Model

iv. Concurrent Models
The concurrent development model, sometimes called concurrent engineering, allows a software team to represent iterative and concurrent elements of any of the process models. The concurrent model is often more appropriate for product engineering projects where different engineering teams are involved.

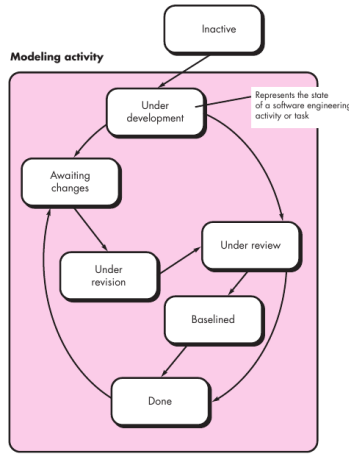


Figure 3.23: Concurrent Model

v. Component-Based Models
Evolutionary in nature & demanding an iterative approach to the creation of software.

vi. The Formal Methods Model Formal methods are a particular kind of mathematically based techniques for the specification, development and verification of software and hardware systems.

vii. Aspect-Oriented Software Development Provides a process and methodological approach for defining, specifying, designing, and constructing aspects - "mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern".

viii. Unified Process

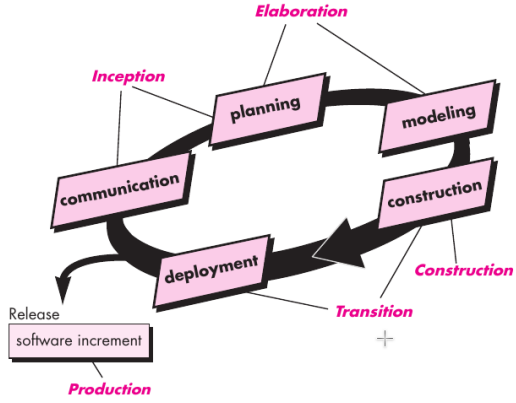


Figure 3.24: Unified Process

3.3.2.2 Agile Development

Ivar Jacobson provides a useful discussion:

"Agility has become today's buzzword when describing a modern software process. Everyone is agile. An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about. Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product. Support for changes should be built-in everything we do in software, something we embrace because it is the heart and soul of software. An agile team recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project."

An agile process reduces the cost of change because software is released in increments and change can be better controlled within an increment.

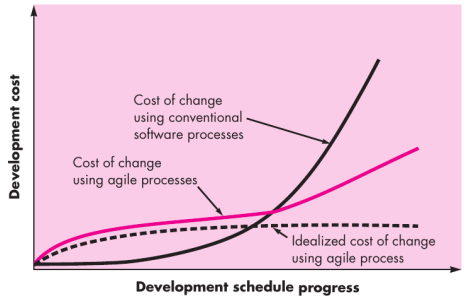


Figure 3.25: Agile: Change costs as a function of time in development

Human factors of agile development are: competence, common focus, collaboration, decision making ability, fuzzy problem solving ability, mutual trust and respect, and self organization.

3.3.2.2.1 Agility Principle

The Agile Alliance defines 12 agility principles for those who want to achieve agility:

i. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
ii. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
iii. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
iv. Business people and developers must work together daily throughout the project.
v. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
vi. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
vii. Working software is the primary measure of progress.
viii. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

ix. Continuous attention to technical excellence and good design enhances agility.

x. Simplicitythe art of maximizing the amount of work not done-is essential.

xi. The best architectures, requirements, and designs emerge from self organizing teams.

xii. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

3.3.2.2.2 Agility Process Models

i. Extreme Programming (XP)

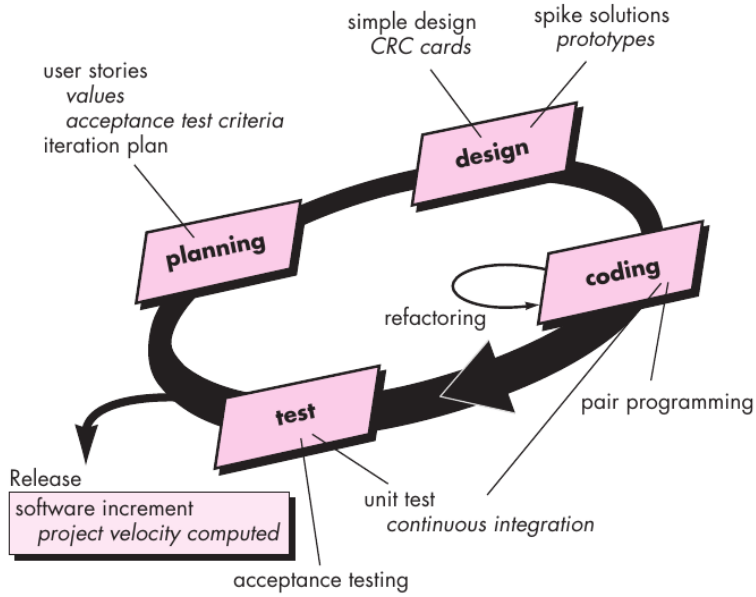


Figure 3.26: Extreme Programming Process

ii. Adaptive Software Development (ASD)

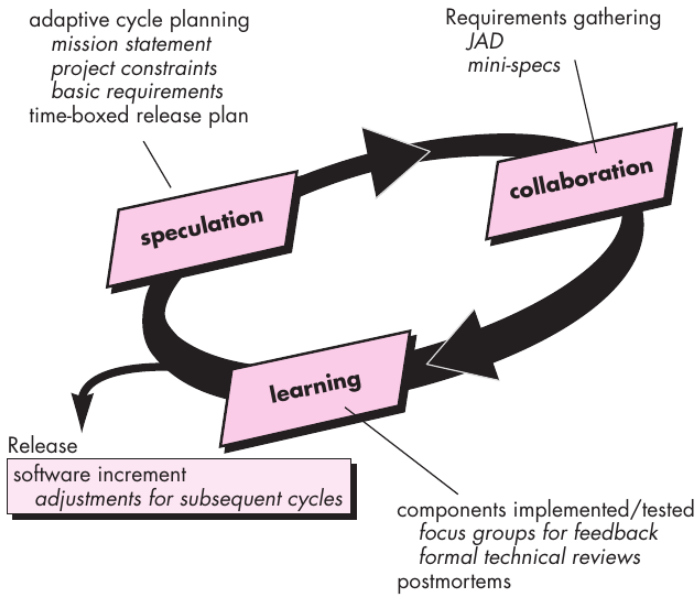


Figure 3.27: Adaptive Software Development

iii. Scrum

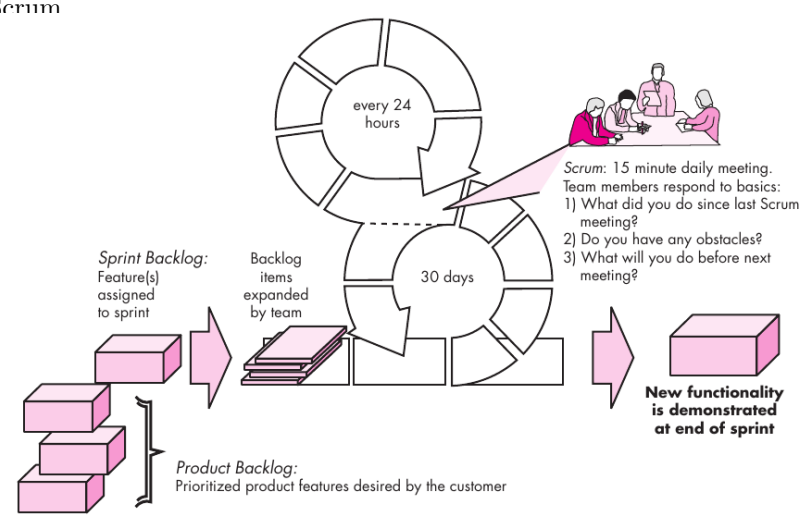


Figure 3.28: Scrum process flow

iv. Dynamic Systems Development Method (DSDM)

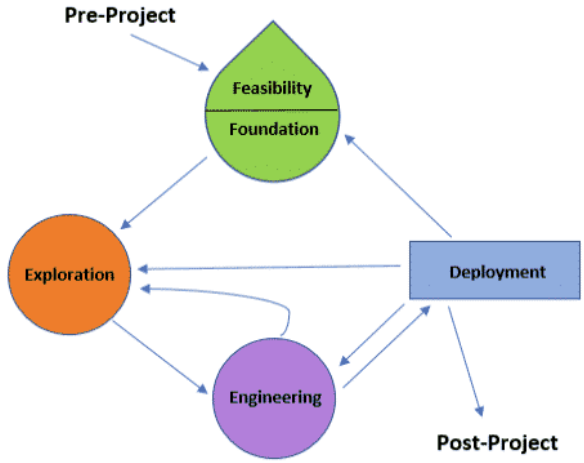


Figure 3.29: Dynamic Systems Development Method

v. Crystal Crystal is a family of process models with the same "genetic code" but different methods for adapting to project characteristics.

vi. Feature Drive Development (FDD) Feature Driven Development (FDD) was originally conceived by Peter Coad and his colleagues as a practical process model for object-oriented software engineering. Stephen Palmer and John Felsing have extended and improved Coads work, describing an adaptive, agile process that can be applied to moderately sized and larger software projects. Like other agile approaches, FDD adopts a philosophy that (1) emphasizes collaboration among people on an FDD team; (2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments, and (3) communication of technical detail using verbal, graphical, and text-based means. FDD emphasizes software quality assurance activities by encouraging an incremental development strategy, the use of design and code inspections, the application of software quality assurance audits, the collection of metrics, and the use of patterns (for analysis, design, and construction).

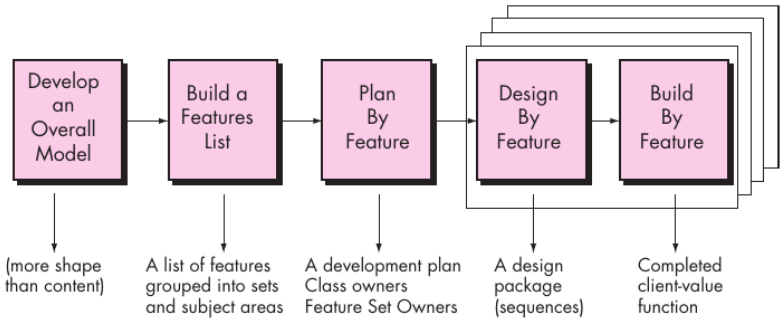


Figure 3.30: Feature Drive Development

vii. Lean Software Development (LSD) Lean Software Development (LSD)

has adapted the principles of lean manufacturing to the world of software engineering. The lean principles that inspire the LSD process can be summarized as eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people, and optimize the whole.

Figure 3.31: Lean Software Development

viii. Agile Modeling (AM)
Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of software-based systems. Simply put, Agile Modeling (AM) is a collection of values, principles, and practices for modeling software that can be applied on a software development project in an effective and light-weight manner. Agile models are more effective than traditional models because they are just barely good, they dont have to be perfect.

ix. Agile Unified Process (AUP)

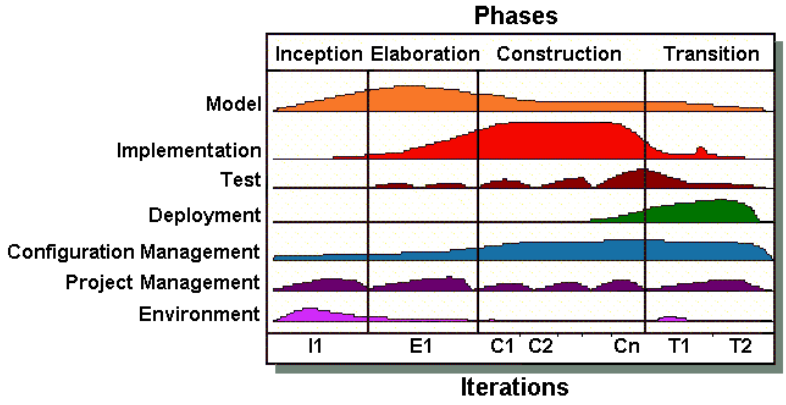


Figure 3.32: Agile Unified Process

3.3.3 Modeling

3.3.3.1 Core principles

3.3.3.1.1 Principles That Guide Process

i. Be agile
ii. Focus on quality at every step.
iii. Be ready to adapt.
iv. Build an effective team.
v. Establish mechanisms for communication and coordination.
vi. Manage change.
vii. Assess risk.
viii. Create work products that provide value for others.

3.3.3.1.2 Principles That Guide Practice

i. Divide and conquer.
ii. Understand the use of abstraction.
iii. Strive for consistency.
iv. Focus on the transfer of information.
v. Build software that exhibits effective modularity.
vi. Look for patterns.
vii. When possible, represent the problem and its solution from a number of different perspectives.
viii. Remember that someone will maintain the software.

3.3.3.1.3 Principles That Guide Each Framework Activity

3.3.3.1.3.1 Communication Principles

i. Listen
ii. Prepare before you communicate.
iii. Someone should facilitate the activity.
iv. Face-to-face communication is best.
v. Take notes and document decisions.
vi. Strive for collaboration.
vii. Stay focused; modularize your discussion.
viii. If something is unclear, draw a picture.
ix. (a) Once you agree to something, move on. (b) If you cant agree to something, move on. (c) If a feature or function is unclear and cannot be clarified at the moment, move on.
x. Negotiation is not a contest or a game. It works best when both parties win.

3.3.3.1.3.2 Planning Principles

i. Understand the scope of the project.
ii. Involve stakeholders in the planning activity.
iii. Recognize that planning is iterative.
iv. Estimate based on what you know.
v. Consider risk as you define the plan.
vi. Be realistic.
vii. Adjust granularity as you define the plan.
viii. Define how you intend to ensure quality.
ix. Describe how you intend to accommodate change.
x. Track the plan frequently and make adjustments as required.

3.3.3.1.3.3 Modeling Principles

i. The primary goal of the software team is to build software, not create models.
ii. Travel lightdont create more models than you need.
iii. Strive to produce the simplest model that will describe the problem or the software.
iv. Build models in a way that makes them amenable to change.
v. Be able to state an explicit purpose for each model that is created.
vi. Adapt the models you develop to the system at hand.
vii. Try to build useful models, but forget about building perfect models.
viii. Dont become dogmatic about the syntax of the model. If it communicates content successfully, representation is secondary.
ix. If your instincts tell you a model isnt right even though it seems okay on paper, you probably have reason to be concerned.
x. Get feedback as soon as you can.

3.3.3.1.3.4 Requirement Modeling Principles

i. The information domain of a problem must be represented and understood.
ii. The functions that the software performs must be defined.
iii. The behavior of the software (as a consequence of external events) must be represented.
iv. The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.
v. The analysis task should move from essential information toward implementation detail.

3.3.3.1.3.5 Requirement Modeling Principles

i. Design should be traceable to the requirements model.
ii. Always consider the architecture of the system to be built.
iii. Design of data is as important as design of processing functions.
iv. Interfaces (both internal and external) must be designed with care.
v. User interface design should be tuned to the needs of the end user.
vi. Component-level design should be functionally independent.
vii. Components should be loosely coupled to one another and to the external environment.
viii. Design representations (models) should be easily understandable.
ix. The design should be developed iteratively. With each iteration, the designer should strive for greater simplicity.

3.3.3.1.3.6 Construction Principles

i. Coding Principles
   A. Preparation principles: Before you write one line of code, be sure you
      - Understand of the problem youre trying to solve.
      - Understand basic design principles and concepts.
      - Pick a programming language that meets the needs of the software to be built and the environment in which it will operate.
      - Select a programming environment that provides tools that will make your work easier.
      - Create a set of unit tests that will be applied once the component you code is completed.
   B. Programming principles: As you begin writing code, be sure you
      - Constrain your algorithms by following structured programming practice.
      - Consider the use of pair programming.
      - Select data structures that will meet the needs of the design.
      - Understand the software architecture and create interfaces that are consistent with it.
      - Keep conditional logic as simple as possible.
      - Create nested loops in a way that makes them easily testable.
      - Select meaningful variable names and follow other local coding standards.
      - Write code that is self-documenting.
      - Create a visual layout (e.g., indentation and blank lines) that aids understanding.
   C. Validation Principles: After youve completed your first coding pass, be sure you
      - Conduct a code walkthrough when appropriate.

- Perform unit tests and correct errors youve uncovered.
- Refactor the code.

D. Testing Principles. In a classic book on software testing, Glen Myers states a number of rules that can serve well as testing objectives:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as-yet-undiscovered error.
- A successful test is one that uncovers an as-yet-undiscovered error.

3.3.3.1.3.7 Deployment Principles

i. Customer expectations for the software must be managed.
ii. A complete delivery package should be assembled and tested.
iii. A support regime must be established before the software is delivered.
iv. Appropriate instructional materials must be provided to end users.

3.3.3.2 Understanding requirement engineering

Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system. It encompasses seven distinct tasks:

i. Inception
At project inception,there require to establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

ii. Elicitation
Elicitation deal with: ask the customer, the users, and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis.
Christel and Kang identify a number of problems that are encountered as elicitation occurs: problem of scope, problem of understanding, and problems of volatility.

iii. Elaboration
Elaboration is driven by the creation and refinement of user scenarios that describe how the end user (and other actors) will interact with the system. Each userscenario is parsed to extract analysis classesbusiness domain entities that are visible to the end user. The attributes of each analysis class are defined, and the services that are required by each class are identified. The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced.

iv. Negotiation
Negotiation isn't unusual for customers and users to ask for more than canachieved, given limited business resources. Its also relatively common for different customers or users to propose conflicting requirements, arguing that their version "essential for our special needs." You have to reconcile these conflicts through a process of negotiation. There should be no winner and no loser in an effective negotiation. Both sides win, because a "deal" that both can live with is solidified.

v. Specification
In the context of computer-based systems (and software), the term specification means different things to different people. A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.

vi. Validation
Validation. The work products produced as a consequence of requirements engineering are assessed for quality during a validation step. Requirements validation examines the specification5 to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product. The primary requirements validation mechanism is the technical review.

vii. Requirement Management
Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.

It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.

3.3.3.2.1 Elements of Requirements Model

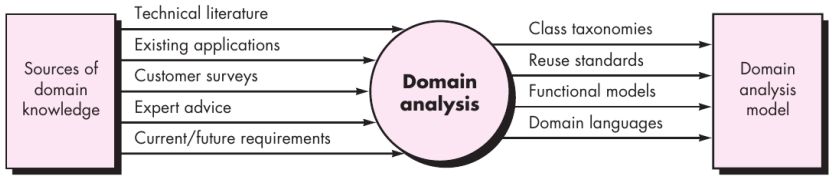


Figure 3.33: Domain Analysis: Input and Output

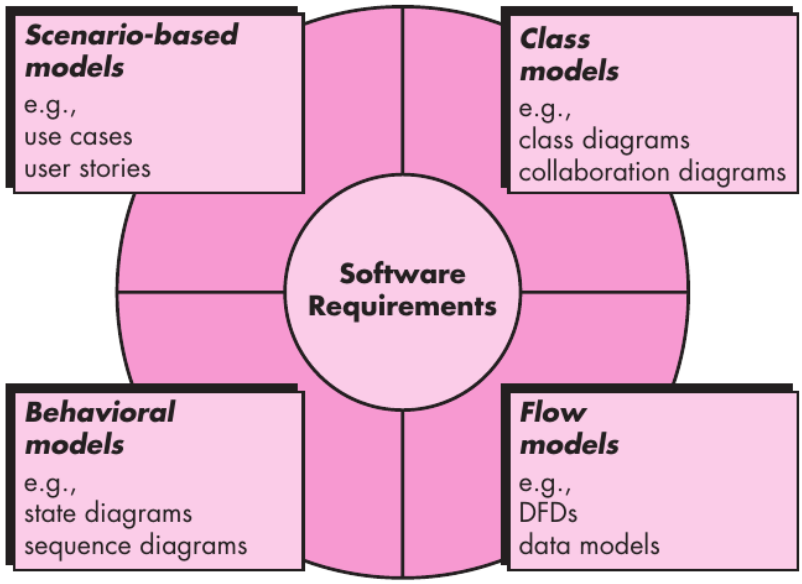


Figure 3.34: Elements of Software Requirements

3.3.3.3 Design concepts

The Roman architecture critic Vitruvius advanced the notion that well-designed buildings were those which exhibited firmness, commodity, and delight. The same might be said of good software. Firmness: A program should not have any bugs that inhibit its function. Commodity: A program should be suitable for the purposes for which it was intended. Delight: The experience of using the program should be a pleasurable one. Here we have the beginnings of a theory of design for software.

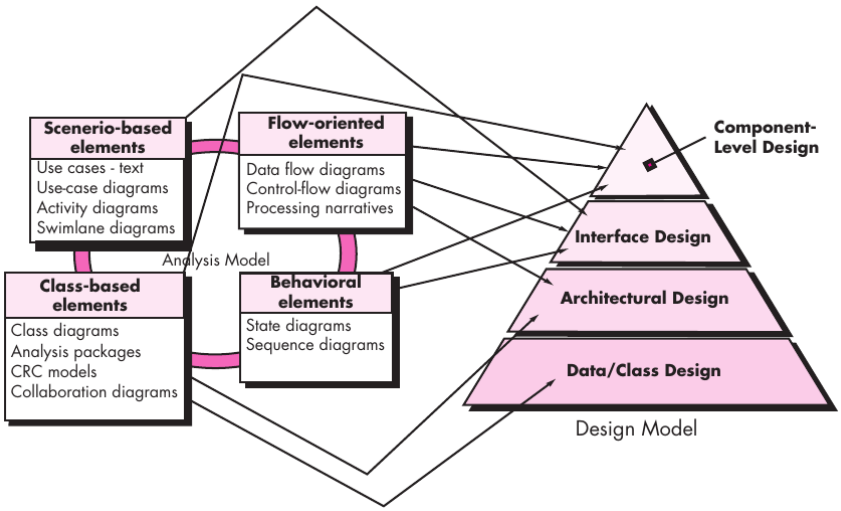


Figure 3.35: Design Model

3.3.3.3.1 Great Design: Quality Attributes

Hewlett-Packard developed a set of software quality attributes that has been given the acronym FURPSfunctionality, usability, reliability, performance, and supportability. The FURPS quality attributes represent a target for all software design:

i. Functionality: is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system.

ii. Usability: is assessed by considering human factors, overall aesthetics, consistency, and documentation.

iii. Reliability: is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.

iv. Performance: is measured by considering processing speed, response time, resource consumption, throughput, and efficiency.

v. Supportability: combines the ability to extend the program (extensibility), adaptability, serviceabilitythese three attributes represent a more common term, maintainabilityand in addition, testability, compatibility, configurability (the ability to organize and control elements of the software configuration), the ease with which a system can be installed, and the ease with which problems can be localized.

3.3.3.3.2 Design Concepts

i. Abstraction

ii. Architecture

iii. Patterns

iv. Seperation of Concerns

v. Modularity

vi. Information Hiding
vii. Functional Independence
viii. Refinement
ix. Aspects
x. Refactoring
xi. Object Oriented Design Concepts
xii. Design Classes

3.3.3.3.3 Interface design

Theo Mandel coins three golden rules:

i. Place the user in control.
ii. Reduce the user's memory load.
iii. Make the interface consistent.

3.3.4 Software Architecture and Taxonomy

The architecture of a system is a comprehensive framework that describes its form and structure - its components and how they fit together.

Figure 3.36: Taxonomy of Software Architectural Styles

3.3.4.1 WebApp Architecture

WebApp architecture describes an infrastructure that enables a Web-based systemapplication to achieve its business objectives. Jacyntho and his colleagues describe the basic characteristics of this infrastructure in the following manner:

Applications should be built using layers in which different concerns are taken into account; in particular, application data should be separated from the page's contents (navigation nodes) and these contents, in turn, should be clearly separated from the interface look-and-feel (pages). The authors suggest a three-layer design architecture that decouples interface from navigation and from application behavior. They argue that keeping interface, application, and navigation separate simplifies implementation and enhances reuse.

The Model-View-Controller (MVC) architecture [Kra88]9 is one of a numbersuggested WebApp infrastructure models that decouple the user interface from the WebApp functionality and informational content.

The model (sometimes referred to as the "model object") contains all application-specific content and processing logic, including all content objects, access to external data/information sources, and all processing functionality that is application specific.

The view contains all interface specific functions and enables the presentation of content and processing logic, including all content objects, access to external data/information sources, and all processing functionality required by the end user.

The controller manages access to the model and the view and coordinates the flow of data between them. In a WebApp, "the view is updated by the controller with data from the model based on user input".

A schematic representation of the MVC architecture is shown in the figure. Referring to the figure, user requests or data are handled by the controller. The controller also selects the view object that is applicable based on the user request. Once the type of request is determined, a behavior request is transmitted to the model, which implements the functionality or retrieves the content required to accommodate the request. The model object can access data stored in a corporate database, as part of a local data store, or as a collection of independent files. The data developed by the model must be formatted and organized by the appropriate view object and then transmitted from the application server back to the client-based browser for display on the customers machine.

3.3.5 Quality Management

Software quality can be defined1 as: An effective software process applied in a manner that creates a useful product that provides measurable value for those who produceand those who use it.

3.3.5.1 Garvin's Quality Dimensions

David Garvin suggests that quality should be considered by taking a multidimensional viewpoint that begins with an assessment of conformance and terminates with a transcendental (aesthetic) view.

Dimension	Refers to
Performance	Primary product characteristics
Feature	Secondary characteristics, added features
Reliability	Consistency of performance overtime
Conformance	Meet specifications or industry standards, workmanship
Durability	Useful life, includes repair
Serviceability	Resolution of problems and compalints, ease of repair, speed and competence of repair
Aesthetics	Sensory characteristics (which means it appeals you)
Perception	Past performance and other intangibles such as being ranking first

Table 3.3: Software Quality Dimensions: Garvin

3.3.5.2 McCall's Quality Factors

McCall, Richards, and Walters propose a useful categorization of factors that affect software quality. These software quality factors,focus on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments.

Figure 3.37: McCall's Quality Factors

Factor	Refers to
Correctness specification and fulfills the customers mission objectives.	The extent to which a program satisfies its
Reliability	The extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform its function.
Integrity	Extent to which access to software or data by unauthorized persons can be controlled.
Usability	Effort required to learn, operate, prepare input for, and interpret output of a program.
Maintainability	Maintainability. Effort required to locate and fix an error in a program.
Flexibility	Effort required to modify an operational program.
Testability	Effort required to test a program to ensure that it performs its intended function.
Portability	Effort required to transfer the program from one hardware and/or software system environment to another.
Reusability	Extent to which a program [or parts of a program] can be reused in other applicationsrelated to the packaging and scope of the functions that the program performs.
Interoperability	Effort required to couple one system to another.

Table 3.4: Software Quality Factors: McCall

3.3.5.3 ISO 9126 Quality Factors

The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

3.3.5.4 Review techniques

A review - any review - is a way of using the diversity of a group of people to:

i. Point out needed improvements in the product of a single person or team;

ii. Confirm those parts of a product in which improvement is either not desired or not needed;

iii. Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

Factor or Attributes	Refers to
Functionality	The degree to which the software satisfies stated needs as indicated by the following subattributes: suitability, accuracy, interoperability, compliance, and security.
Reliability	The amount of time that the software is available for use as indicated by the following subattributes: maturity, fault tolerance, recoverability.
Usability	The degree to which the software is easy to use as indicated by the following subattributes: understandability, learnability, operability.
Efficiency	The degree to which the software makes optimal use of system resources as indicated by the following subattributes: time behavior, resource behavior.
Maintainability	The ease with which repair may be made to the softwareindicated by the following subattributes: analyzability, changeability, stability, testability.
Portability	The ease with which the software can be transposed from one environment to another as indicated by the following subattributes: adaptability, installability, conformance, replaceability.

Table 3.5: ISO 9126 Quality Factors

3.3.5.4.1 Review Metrics

Metrics	Exploration
Preparation effort, E_p	the effort (in person-hours) required to review a work product prior to the actual review meeting
Assessment effort, E_a	the effort (in person-hours) that is expended during the actual review
Rework effort, E_r	the effort (in person-hours) that is dedicated to the correction of those errors uncovered during the review
Work product size, WPS	a measure of the size of the work product that has been reviewed (e.g., the number of UML models, or the number of document pages, or the number of lines of code)
Minor errors found, Err_{minor}	the number of errors found that can be categorized as minor (requiring less than some prespecified effort to correct)
Major errors found, Err_{major}	the number of errors found that can be categorized as major (requiring more than some prespecified effort to correct)

Table 3.6: Review Metrics

Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
		Number of UML errors
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
		Layout appropriateness
	Patterns	Number of patterns used
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error

Figure 3.38: Software quality goals, attributes, and metrics

The total review effort and the total number of errors discovered are defined as:
$E_{review} = \text{Ep} + \text{Ea} + \text{Er}$
$Err_{tot} = Err_{minor} + Err_{major}$

Error density represents the errors found per unit of work product reviewed:
Error density = $Err_{tot}\,\overline{WPS}$

3.3.5.5 Software Quality Assurance

Software quality assurance (SQA) is a means of monitoring the software engineering processes and methods used to ensure proper quality. This is accomplished by many and varied approaches. It may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.

It includes standards and procedures that administrators may use to review and audit software products and activities to verify that the software meets standards. According

to ISO/IEC 15504 v.2.5 (SPICE), it is a supporting process that provides the independent assurance that all work products, activities and processes comply with the predefined plans and ISO 15504.

SQA encompasses the entire software development process, including requirements definition, software design, coding, code reviews, source code control, software configuration management, testing, release management and product integration. It is organized into goals, commitments, abilities, activities, measurements and verification.

SQA involves a three-prong approach:

i. Organization-wide policies, procedures and standards
ii. Project-specific policies, procedures and standards
iii. Compliance to appropriate procedures

Elements of software quality assurance:

i. Standards
ii. Reviews and audits
iii. Testing
iv. Error/defect collection and analysis
v. Change management
vi. Education
vii. Vendor management
viii. Security management
ix. Safety
x. Risk management

3.3.5.6 Managing software projects

3.3.5.6.1 Project management concepts

Effective software project management focuses on the four Ps: people, product, process, and project.

Metrics	Deals with
People	Recruitment, training, organization, team development
Product	Defining project scope and design of product
Process	Establish framework for software development
Project	Understand complexities of project development

Table 3.7: Project management concepts: 4 P's

3.3.5.6.1.1 W^5HH Principle Barry Boehm states: "you need an organizing principle that scales down to provide simple [project] plans for simple projects." Boehm suggests an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources. He calls it the W^5HH Principle, after a series of questions that lead to a definition of key project characteristics and the resultant project plan:

i. Why is the system being developed? All stakeholders should assess the validity of business reasons for the software work. Does the business purpose justify the expenditure of people, time, and money?

ii. What will be done? The task set required for the project is defined. When will it be done? The team establishes a project schedule by identifying when project tasks are to be conducted and when milestones are to be reached.

iii. Who is responsible for a function? The role and responsibility of each member of the software team is defined.

iv. Where are they located organizationally? Not all roles and responsibilities reside within software practitioners. The customer, users, and other stakeholders also have responsibilities.

v. How will the job be done technically and managerially? Once product scope is established, a management and technical strategy for the project must be defined.

vi. How much of each resource is needed? The answer to this question is derived by developing estimates (Chapter 26) based on answers to earlier questions.

3.3.5.6.2 Estimation for software projects

3.3.5.6.2.1 Impact and risk assessment

Components / Category		Performance	Support	Cost	Schedule
Catastrophic	1	Failure to meet the requirement would result in mission failure		Failure results in increased costs and schedule delays with expected values in excess of $500K	
	2	Significant degradation to nonachievement of technical performance	Nonresponsive or unsupportable software	Significant financial shortages, budget overrun likely	Unachievable IOC
Critical	1	Failure to meet the requirement would degrade system performance to a point where mission success is questionable		Failure results in operational delays and/or increased costs with expected value of $100K to $500K	
	2	Some reduction in technical performance	Minor delays in software modifications	Some shortage of financial resources, possible overruns	Possible slippage in IOC
Marginal	1	Failure to meet the requirement would result in degradation of secondary mission		Costs, impacts, and/or recoverable schedule slips with expected value of $1K to $100K	
	2	Minimal to small reduction in technical performance	Responsive software support	Sufficient financial resources	Realistic, achievable schedule
Negligible	1	Failure to meet the requirement would create inconvenience or nonoperational impact		Error results in minor cost and/or schedule impact with expected value of less than $1K	
	2	No reduction in technical performance	Easily supportable software	Possible budget underrun	Early achievable IOC

Figure 3.39: Impact and Risk Assessment

3.3.5.6.2.2 Empirical Estimation Model

An estimation model for computer software uses empirically derived formulaspredict effort as a function of LOC (Lines of Code) or FP (Functional Point). An estimation model reflects the population of projects from which it has been derived. Therefore, the model is domain sensitive.

The Constructive Cost Model (COCOMO) is a procedural software cost estimation model developed by Barry W. Boehm is a good example of empirical estimation model.

3.3.5.6.2.3 Intermediate COCOMO Model

Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes. This extension considers a set of four "cost drivers", each with a number of subsidiary attributes. Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value). An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4.

Software project	a_b	b_b	c_b	d_b	Effort	Schedule
Organic	3.2	1.05	2.5	0.38	$E = a_b \times KLoC^{b_b}$	$DEV = c_b \times E^{d_b}$
Semidetached	3.0	1.12	2.5	0.35	$E = a_b \times KLoC^{b_b}$	$DEV = c_b \times E^{d_b}$
Embedded	2.8	1.2	2.5	0.32	$E = a_b \times KLoC^{b_b}$	$DEV = c_b \times E^{d_b}$

Table 3.8: Cost drivers and ratings of Intermediate COCOMO Model

The Intermediate Cocomo formula now takes the form:

For	Equation	Measuring unit
Effort	$E = a_b \times (KLoC^{b_b}) \times (EAF)$	person per month
Development time	$DEV = c_b \times E^{d_b}$	months
Productivity	$\frac{KLoC}{E}$	Lines of code for per person per month
Average Employee	$\frac{E}{DEV}$	full time software personal

Table 3.9: Equation of Intermediate COCOMO Model Estimation

where,
KLoC is the estimated number of thousands of delivered lines of code for the project,
EAF is the factor calculated above.
The coefficient a and the exponent b are given in the table.

[McConnell 04][Larman 04][Fowler 12][Pfleeger 06][Sommerville 07][Pressman 05][Brooks 95]

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database	0.94	1.00	1.08	1.16		
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Table 3.10: Cost drivers and ratings of Intermediate COCOMO Model

Chapter 4

Methodology

4.1 Scopes of the dissertation

Preparing and compiling this project takes vast amount of reading, preparation, data collection-cleaning-preparing, selecting trend & tools, installing required & relative industrial software platform and environment and finally coding for the software application based on the following core scopes:

i. Proposition 1: Arrangement of the resource
There exists some resources(g(R)) such that the resources are not belong to any other existing services(g(X))

where:
g(R) is the number of rooms and employee;
and g(X) is the allocated employee and rooms

ii. Proposition 2: Clustering demand supply
There exists some resources(g(R) and g(X)) such that resources belong to the demand(g(D)); and resources are available to supply(g(S)) from the g(X); and g(S) are directly proportional to the demand(g(D))
where:
g(D) is the client's requirement;
g(S) is the available resources of the organization;
g(X) is the set of available services

Clustering the supply of existing resources (humanware, services, and vacant rooms) according to the client's demand

iii. Proposition 3: Finding the predicted resource
There exists some resources(g(E)), g(R)) such that the resources belong to the resources(g(A)) and g(E) g(R) belongs to the demand(g(A))
where:
the predicted(g(E)) is set of employee and g(A) is the predicted outcome;

outcome of the resources can be shown from Feedforward Backpropagation Neural Network using supervised learning and trained dataset

iv. Proposition 4: Finding sponsor by machine learning algorithm and genetic algorithm
There exists set of employees g(E) who belongs to the subset of g(GA) and belongs to the predicted outcome (g(A))
where:
g(GA) is the employee who is responsible to deal the g(D) of the client during his stay at the premised;
and g(GA) is equavalent of demand(g(A));
g(A) is equavalent of g(D);
g(D) is equavalent of g(S);
g(R) <= g(X) <= g(D) <= g(S) <= g(A) <= g(GA)

Using genetic algorithm and programming to find the optimized outcome

v. Proposition 5: Allocate the resource
Iff, Proposition 1 to 4 ϵ true

4.2 Research Methods

4.2.1 Selecting Research Method

Quantitative research is the systematic empirical investigation of observable phenomena via statistical, mathematical, or computational techniques.[Given 08] The objective of quantitative research is to develop and employ mathematical models, theories, and hypotheses pertaining to phenomena. The process of measurement is central to quantitative research because it provides the fundamental connection between empirical observation and mathematical expression of quantitative relationships. Quantitative data is any data that is in numerical form such as statistics, percentages, etc. The researcher analyses the data with the help of statistics and hopes the numbers will yield an unbiased result that can be generalized to some larger population. Qualitative research, on the other hand, inquires deeply into specific experiences, with the intention of describing and exploring meaning through text, narrative, or visual-based data, by developing themes exclusive to that set of participants.

Qualitative research is a scientific method of observation to gather non-numerical data.[Babbie 16] This type of research "refers to the meanings, concepts, definitions, characteristics, metaphors, symbols, and description of things" and not to their "counts or measures". This research answers why and how a certain phenomenon may occur rather than how often.

While maintaining quality and efficiency of the project, there replaces great impact of choosing "Mix Research Method" which follows both quantitative and qualitative methods. The entire process, process of work, and managing the entire life cycle of the dissertation gets simple while following this methods with very low limitations.

4.2.2 Project Design and Data Design

In this fragment, the discussion on project design and data collection, selecting platform and designing system has been populated.

4.2.2.1 Finding the Stakeholders

Stakeholder is a member of "groups without whose support the organization would cease to exist". During the entire project and dissertation, there are two supervisor, one lead supervisor, executive from hotel chain management, one developer, one proofreader, one plagiarism checker, and one dissertation author. Almost past 300 days, all these fellows continuously and actively share the knowledge to compile the utmost service and product.

4.2.2.2 Software Process, Project management, and Development Life Cyle: SCRUM

Agile methodology is impeccable industrial project management process which is by nature evolutionary and iterative. To work on the project, there has been choosen Scrum software process which belongs to the agile methodology.

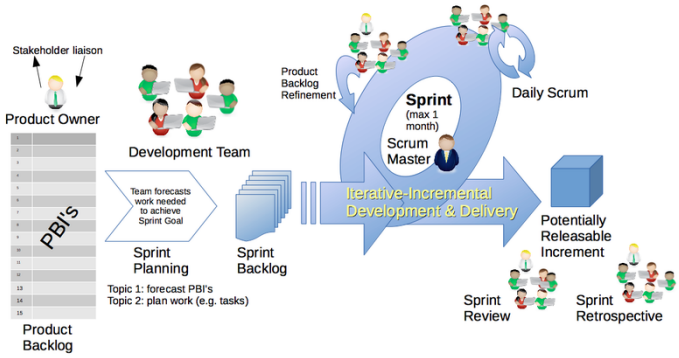


Figure 4.1: Methodology: Scrum

Scrum is an agile process framework for managing complex knowledge work, with an initial emphasis on software development. Hirotaka Takeuchi and Ikujiro Nonaka introduced the term scrum in the context of product development. It is designed for teams of ten or fewer members, who break their work into goals that can be completed within timeboxed iterations, called sprints, no longer than one month and most commonly two weeks, then track progress and re-plan in quarter-minute time-boxed stand-up meetings, called daily scrums.

Scrum is a lightweight, iterative and incremental framework for managing complex work. The framework challenges assumptions of the traditional, sequential approach to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved.

A key principle of Scrum is the dual recognition that customers will change their minds about what they want or need (often called requirements volatility) and that there will be unpredictable challengesfor which a predictive or planned approach is not suited.
As such, Scrum adopts an evidence-based empirical approach accepting that the problem cannot be fully understood or defined up front, and instead focusing on how to maximize the team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in market conditions.

4.2.2.2.1 Roles: Scrum Framework

4.2.2.2.1.1 Product owner The product owner, representing the product's stakeholders and the voice of the customer (or may represent the desires of a committee), is responsible for delivering good business results. Hence, the product owner is accountable for the product backlog and for maximizing the value that the team delivers. The product owner defines the product in customer-centric terms (typically user stories), adds them to the product backlog, and prioritizes them based on importance and dependencies. This role requires a deep understanding of both sides: the business and the engineers (developers) in the scrum team. Communication is a core responsibility of the product owner. As the face of the team to the stakeholders, the following are some of the communication tasks of the product owner to the stakeholders:

i. Define and announce releases.
ii. Communicate delivery and team status.
iii. Share progress during governance meetings.
iv. Share significant RIDAs (risks, impediments, dependencies, and assumptions) with stakeholders.
v. Negotiate priorities, scope, funding, and schedule.
vi. Ensure that the product backlog is visible, transparent and clear.

A product owner's ability to communicate effectively is also enhanced by being skilled in techniques that identify stakeholder needs, negotiate priorities between stakeholder interests, and collaborate with developers to ensure effective implementation of requirements.

4.2.2.2.1.2 Development team The development team has from three to nine members who carry out all tasks required to build increments of valuable output every sprint.
The term "developer" refers to anyone who plays a role in the development and support of the system or product, and can include researchers, architects, designers, data specialists, statisticians, analysts, engineers, programmers, and testers, among others. The team is self-organizing. The team should still be encouraged to interact directly with customers and/or stakeholders to gain maximum understanding and immediacy of feedback.

4.2.2.2.1.3 Scrum master Scrum is facilitated by a scrum master, who is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The scrum master is not a traditional team lead or project manager but acts as a buffer between the team and any distracting influences. The

scrum master ensures that the scrum framework is followed. The scrum master helps to ensure the team follows the agreed processes in the Scrum framework, often facilitates key sessions, and encourages the team to improve. The role has also been referred to as a team facilitator or servant-leader to reinforce these dual perspectives.

The core responsibilities of a scrum master include (but are not limited to):

i. Helping the product owner maintain the product backlog in a way that ensures the needed work is well understood so the team can continually make forward progress
ii. Helping the team to determine the definition of done for the product, with input from key stakeholders
iii. Coaching the team, within the Scrum principles, in order to deliver high-quality features for its product[33]
iv. Promoting self-organization within the team
v. Helping the scrum team to avoid or remove impediments to its progress, whether internal or external to the team
vi. Facilitating team events to ensure regular progress
vii. Educating key stakeholders on Agile and Scrum principles
viii. Coaching the development team in self-organization and cross-functionality

The scrum master helps people and organizations adopt empirical and lean thinking, leaving behind hopes for certainty and predictability.

4.2.2.2.2 Workflow: Scrum Framework

4.2.2.2.2.1 Sprint A sprint (also known as iteration or timebox) is the basic unit of development in Scrum. The sprint is a timeboxed effort; that is, the length is agreed and fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common. Each sprint starts with a sprint planning event that establishes a sprint goal and the required product backlog items. The team accepts what they agree is ready and translate this into a sprint backlog, with a breakdown of the work required and an estimated forecast for the sprint goal. Each sprint ends with a sprint review and sprint retrospective, that reviews progress to show to stakeholders and identify lessons and improvements for the next sprints.

4.2.2.2.2.2 Sprint planning At the beginning of a sprint, the scrum team holds a sprint planning event to:

i. Mutually discuss and agree on the scope of work that is intended to be done during that sprint
ii. Select product backlog items that can be completed in one sprint
iii. Prepare a sprint backlog that includes the work needed to complete the selected product backlog items
iv. Agree the sprint goal, a short description of what they are forecasting to deliver at the end of the sprint.
v. The recommended duration is four hours for a two-week sprint (pro-rata for other sprint durations)

A. During the first half, the whole scrum team (development team, scrum master, and product owner) selects the product backlog items they believe could be completed in that sprint
B. During the second half, the development team identifies the detailed work (tasks) required to complete those product backlog items; resulting in a confirmed sprint backlog
C. As the detailed work is elaborated, some product backlog items may be split or put back into the product backlog if the team no longer believes they can complete the required work in a single sprint

vi. Once the development team has prepared their sprint backlog, they forecast (usually by voting) which tasks will be delivered within the sprint.

4.2.2.2.2.3 Daily scrum A daily scrum in the computing room. This centralized location helps the team start on time. Each day during a sprint, the team holds a daily scrum (or stand-up) with specific guidelines. No detailed discussions should happen during the daily scrum. Once the meeting ends, individual members can get together to discuss issues in detail; such a meeting is sometimes known as a 'breakout session' or an 'after party'.[38]

4.2.2.2.2.4 Sprint review At the end of a sprint, the team holds two events: the sprint review and the sprint retrospective.

i. At the sprint review, the team:
   A. reviews the work that was completed and the planned work that was not completed
   B. presents the completed work to the stakeholders (a.k.a. the demo)
   C. collaborates with the stakeholders on what to work on next
ii. Guidelines for sprint reviews:
   A. Incomplete work cannot be demonstrated.
   B. The recommended duration is two hours for a two-week sprint (proportional for other sprint-durations).[17]

4.2.2.2.2.5 Sprint retrospective At the sprint retrospective, the team Reflects on the past sprint and identifies and agrees on continuous process improvement actions.
Guidelines for sprint retrospectives:

i. Three main questions are asked in the sprint retrospective: What went well during the sprint? What did not go well? What could be improved for better productivity in the next sprint?
ii. The recommended duration is one-and-a-half hours for a two-week sprint (proportional for other sprint duration(s))
iii. This event is facilitated by the scrum master

4.2.2.2.2.6 Backlog refinement Backlog refinement (formerly called grooming) is the ongoing process of reviewing product backlog items and checking that they are appropriately prepared and ordered in a way that makes them clear and executable

for teams once they enter sprints via the sprint planning activity. Product backlog items may be broken into multiple smaller ones. Acceptance criteria may be clarified. Dependencies may be identified and investigated.

The backlog can also include technical debt (also known as design debt or code debt). This is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer.

4.2.2.2.2.7 Product backlog The product backlog is a breakdown of work to be done and contains an ordered list of product requirements that a scrum team maintains for a product. Common formats include user stories and use cases. The requirements define features, bug fixes, non-functional requirements, etc. -whatever must be done to deliver a viable product. The product owner prioritizes product backlog items (PBIs) based on considerations such as risk, business value, dependencies, size, and date needed. The product backlog is what will be delivered, ordered into the sequence in which it should be delivered. It is visible to everyone but may only be changed with the consent of the product owner, who is ultimately responsible for ordering product backlog items for the development team to choose. The product backlog: captures requests to modify a productincluding new features, replacing old features, removing features, and fixing issues; and ensures the development team has work that maximizes business benefit to the product owner. The product owner and the scrum team work together to develop the breakdown of work; this becomes the product backlog, which evolves as new information surfaces about the product and about its customers, and so later sprints may address new work.

4.2.2.2.2.8 Sprint backlog The sprint backlog is the list of work the development team must address during the next sprint. The list is derived by the scrum team progressively selecting product backlog items in priority order from the top of the product backlog until they feel they have enough work to fill the sprint. The product backlog items may be broken down into tasks by the development team. Tasks on the sprint backlog are never assigned (or pushed) to team members by someone else; rather team members sign up for (or pull) tasks as needed according to the backlog priority and their own skills and capacity. This promotes self-organization of the development team and developer buy-in.

The sprint backlog is the property of the development team, and all included estimates are provided by the development team. Often an accompanying task board is used to see and change the state of the tasks of the current sprint, like to do, in progress and done.

Once a sprint backlog is committed, no additional work can be added to the sprint backlog except by the team. Once a sprint has been delivered, the product backlog is analyzed and reprioritized if necessary, and the next set of functionality is selected for the next sprint.

4.2.2.2.2.9 Increment The increment is the potentially releasable output of the sprint that meets the sprint goal.

4.2.2.3 Platform and System Design

A computing platform or digital platform is the environment in which a piece of software is executed. It may be the hardware or the operating system (OS), even a web browser and associated application programming interfaces, or other underlying software, as long as the program code is executed with it. Computing platforms have different abstraction levels, including a computer architecture, an OS, or runtime libraries. A computing platform is the stage on which computer programs can run.

Components	Entitle
Processor	GPU Processor
Architecture	$x86_64$
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	4
Model name	Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
CPU MHz	2394.466
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	3072K
Browser	Firefox, localhost:80
Operating System	Linux, Unix

Table 4.1: Platform for the project

4.2.2.4 Data collection and data processing

Primary data is the new information collected through original or first-hand research while secondary data is information which has been collected in the past by others. To participate in this project, the secondary data about the client and employee has been randomly generated with the scenario of a related hospitality industry real-time data. Soon after collecting data, the raw data has been organized to maintain a structured data description, seperate data into train-test-ranking-input sub fields, scaling the data, split the data for training and prediction, labelling the data, remove the null data fields and making data ready to pass as argument to the python models for future prediction and other related computing.

4.3 Software Development Engineering & Scientific Tools

4.3.1 Software Development tools

4.3.1.1 Software frameworks

4.3.1.2 Software architecture

4.3.1.2.1 Model-View-Controller (MVC) Using netbeans, there has been used the web application in simple forms to interact with the client and the organization.

Category	Entitle
Development	Python, Jupyter Notebook, Java Development Kit, Netbeans, PlantUML, TaskJuggler
Application runtime	Linux Kernel, Java Runtime Environment
Database	MySql
Web Application	Apache Tomcat Server
Security	Spring Security, SSL, HTTPS Authentication, Encrypt Password, Form-based Authentication
Platform	

Table 4.2: Methodology: Software Framework

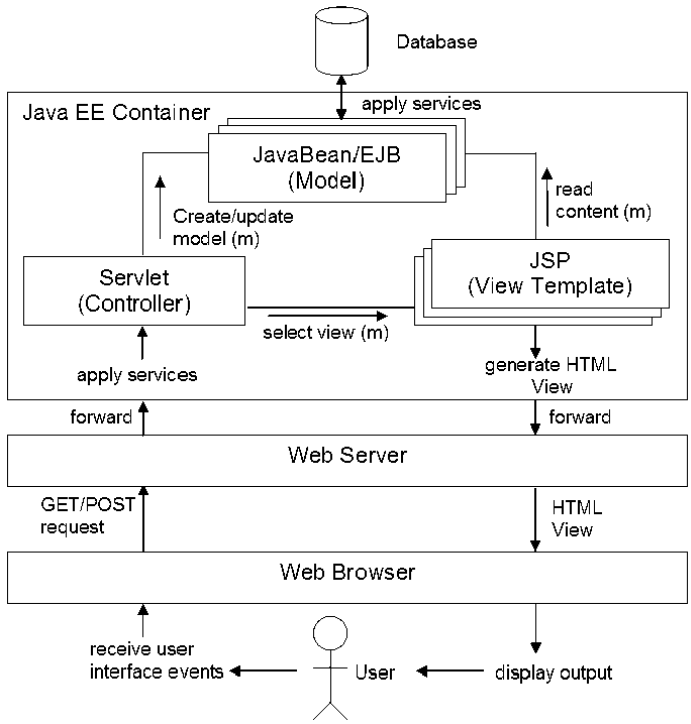


Figure 4.2: Software Architecture: J2EE MVC

4.3.1.2.2 Client-Server (Multi-tier or N-tier) Clientserver model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. The heart of this project is depend on this feature.

4.3.1.2.2.1 Advantages of Client Server Model

i. Centralized: Centralized back-up is possible in client-server networks, i.e., all the data is stored in a server.

ii. Security: These networks are more secure as all the shared resources

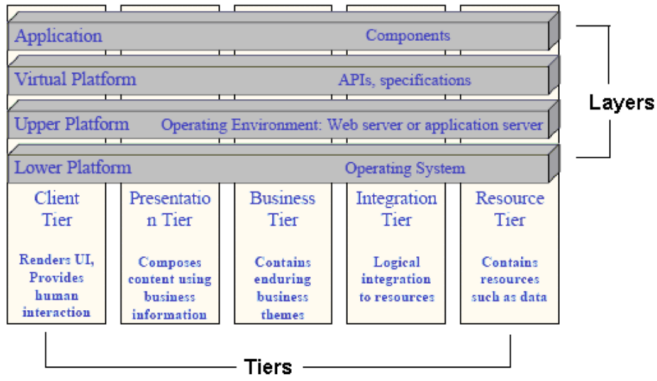


Figure 4.3: Software Architecture: Multi-tier

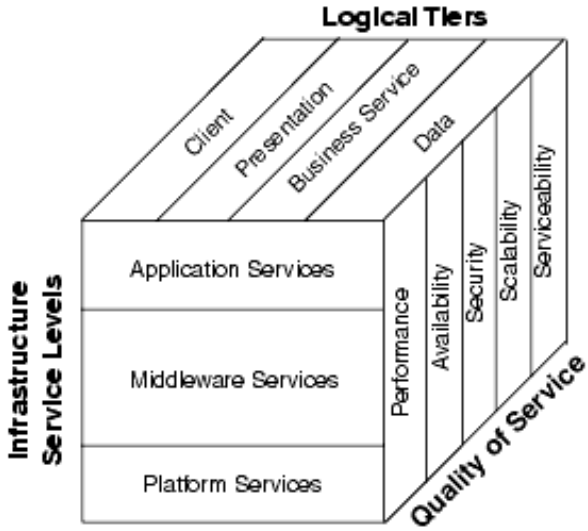


Figure 4.4: 3D View: Multi-tier

are centrally administered.

iii. Performance: The use of the dedicated server increases the speed of sharing resources. This increases the performance of the overall system.

iv. Scalability: Able to increase the number of clients and servers separately, i.e., the new element can be added, or we can add a new node in a network at any time.

4.3.1.3 Feedforward Based Backpropagation Neural Network

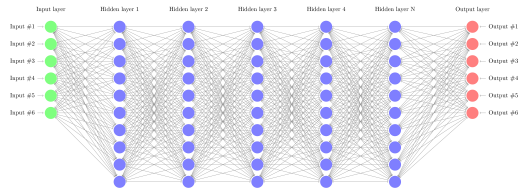


Figure 4.5: Neural Network: Feedforward Backpropagation

In the project, there used siz input layers and outpur layers, and ten hidden layer to compute the predictions.

4.3.1.4 TPOT: Genetic Algorithm

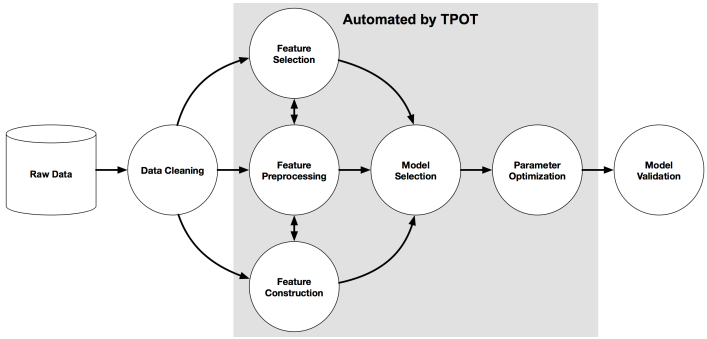


Figure 4.6: Genetic Algoritm: TPOT

A Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. TPOT stands for Tree-based Pipeline Optimization Tool. TPOT will automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for your data.[Le 19]

Automated Machine Learning algorithms aren't as simple as fitting one model on the dataset; they are considering multiple machine learning algorithms (random forests, linear models, SVMs, etc.) in a pipeline with multiple preprocessing steps (missing value imputation, scaling, PCA, feature selection, etc.), the hyperparameters for all of the models and preprocessing steps, as well as multiple ways to ensemble or stack the algorithms within the pipeline. Typical TPOT runs will take hours to days to finish (unless it's a small dataset), but you can always interrupt the run partway through and see the best results so far. TPOT also provides a warm start parameter that lets you restart a TPOT run from where it left off.AutoML algorithms can recommend different solutions for the same dataset. AutoML algorithms can take a long time to finish their search. TPOT using scoring function for evaluating pipelines.

4.3.1.5 Distributed J2EE And Apache Tomcat Server

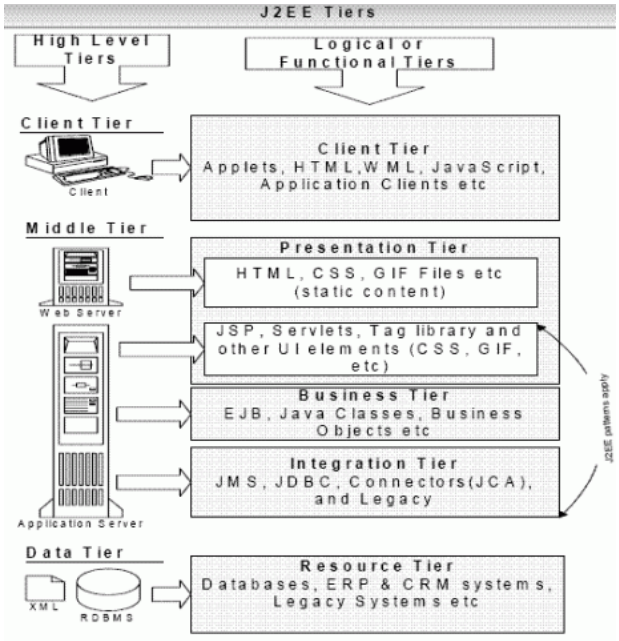


Figure 4.7: Distributed J2EE Multitier

4.3.1.6 Aerial View of the Proposed Layered System

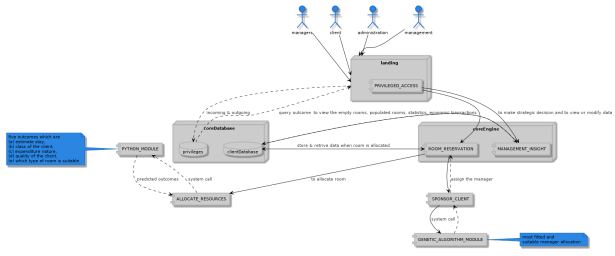
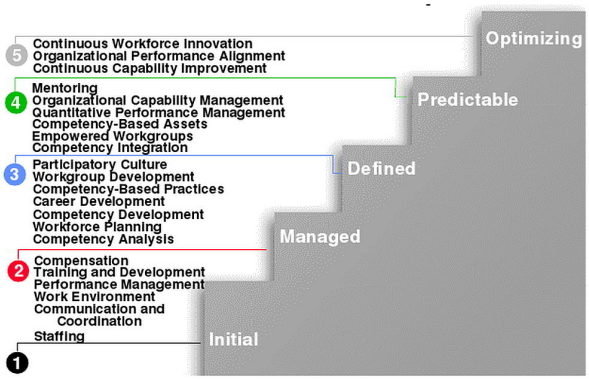


Figure 4.8: Aerial View of the Proposed Layered System

- the above diagram is the simplist view of the proposed system for the project.

4.3.2 People Capability Maturity Model



Thoughout the project, the PCMM has been followed to maintain the standards.

4.3.3 Software Engineering Ethics

The ACM and IEEE Computer Society jointly released the Software Engineering Code of Ethics and Professional Practice, which outlines 8 principles of software engineering ethics: the obligation of the software engineer to the general public, the client and employer, the product, the profession, colleagues, the engineer himself or herself, and the ethical management of software engineering projects which this project takes as strategic motion to maintain ethics.

4.4 Procedure: What and How about the project

The core project and purpose of this document is based on the following five propositions which is complete data driven perspectives:

i. Proposition 1: Arrangement of the resource
In this phase, all the resources like rooms, client information, inventory will be collected for next calculation.

ii. Proposition 2: Clustering demand supply
Finding the client's demand, the host organization will cluster or find suitable similar resource pool before assign the resource and employee to the client as the client's personal guide or host manager.

iii. Proposition 3: Finding the predicted resource Using neural network with backpropagation algorithm, the most higher estimated resources will be allocated to the client.

iv. Proposition 4: Finding sponsor by automated machine learning algorithm
Using genetic algorithm and tpot, the predicted employee who is most matched for that specific client will be assigned to host as manager during the life cycle of the client under the hood of the organization.

v. Proposition 5: Allocating predicted resources from above Finally, the predicted available resources will comes out and predicted employee details will be released for the client.

The complete process is automated and machine generated, so the hazard is quite less and the process is time sensitive calculations.

Chapter 5

Implementation Results & discussion

5.1 Finding resources: Feed-forward based Backpropagation Algorithm

5.1.1 Labelled Input

```
# INPUT 1:
# Historical Data of Client

stay,  expense,    social status,  reservation status, client's ranking score,     emmployee_oee_score
1,     1000,       1,              1,                  100,
                   10
1,     800,        2,              1,                  80,
                   10
1,     600,        3,              1,                  60,
                   7
1,     700,        1,              1,                  50,
                   7
1,     10000,      1,              1,                  100,
                   100
.....................................................
.....................................................
23,    6000,       5,              5,                  17,
                   93
24,    5000,       6,              6,                  7,
                   92

# INPUT 2:
# Ranking of each row/client's historical data

ranking
100
80
60
100
180
....
```

```
....
99
100


# INPUT 3:
# Primary Input

stay, expense, social status, reservation status, client's ranking score, emmployee_oee_score
25, 1, 1, 0, 98, 0
```

5.1.2 Primary Python Code

5.1.2.1 Core Neural Network

```
# Title: Finding the sponsor from employee list
#
# CSE 400 - Dissertation: Implementation and Results
# Student     : Rashadul Islam
# Id          : 0301175301
# Department  : Department of CSE
# Semester    : Spring 2019
# Written on  : Wed Sep 10 10:00:00 BST 2019
#

# Library
import numpy as np
import pandas as pd
from numpy import genfromtxt

from NeuralNetworkFeedForwardBackpropagation import Neural_Network

from matplotlib import pyplot as plt

# sample input append at the end
#
# data description
#                           stay, expense, social status, reservation status, client's ranking score, emmployee_oee_score), y = score on test
train = genfromtxt("labeledData.csv", skip_header=6, delimiter=",")
ranking = genfromtxt("ranking.csv", skip_header=1, delimiter=",")
inputs = genfromtxt("input.csv", delimiter=",")


# y1 = np.array(train[:,6])
ranking = np.vstack(ranking)
resourceStatus = np.vstack((train, inputs))

# scale units
resourceStatus = resourceStatus / np.amax(resourceStatus, axis=0)  # scaling input data
ranking = (
    ranking / 100
)  # scaling output data (max test score is 100 as ranking each row)
```

```
38
39 # split data
40 X = np.split(resourceStatus, [23])[0]  # total row =
       resourceStatus row -1
41 pred = np.split(resourceStatus, [23])[1]  # total row =
       resourceStatus row -1
42
43 NN = Neural_Network()
44
45 average_cost = 0
46 cost_func = []
47 y_pred = []
48
49 for i in range(1500):  # trains the NN 1000 times
50     print("# " + str(i) + "\n")
51     print("Input (scaled): \n" + str(X))
52     print("Actual Output: \n" + str(ranking))
53     print("Predicted Output: \n" + str(NN.forward(X)))
54     pred1 = NN.forward(X)
55     y_pred.append(pred1)
56     print("=" * 52)
57     error = np.mean(np.square(ranking - NN.forward(X)))
58     print(
59         "Loss: \n" + str(np.mean(np.square(ranking - NN.forward(
       X))))
60     )  # mean sum squared loss
61     print("=" * 52)
62     NN.train(X, ranking)
63     cost_func.append(error)
64
65 for i in range(1500):
66     plt.plot(cost_func)
67     plt.xlabel("epoch")
68     plt.ylabel("cost_func")
69
70 # NN.saveWeights()
71
72 # Prediction
73 print("Predicted data based on trained weights: ")
74 print("Input (scaled): \n" + str(pred))
75 NN.predict(pred)
76
77
78 print("Error or loss in 1500 epoch")
79 print("=" * 52)
80 print(error)
81 print("=" * 52)
82
83 # Accuracy or F1 Score
84 from numpy import array
85
86 y_pred = array(y_pred)
87
88 pred2 = y_pred[:23, -1]
89 pred = np.vstack(pred2[:, -1])
90
91 from sklearn.model_selection import train_test_split
92
93 X_train, X_test, y_train, y_test = train_test_split(train,
       ranking, test_size=0.99)
94
```

```
print(y_test.shape)

# from sklearn.metrics import accuracy_score
# accuracy_score(y_test,pred)*100

from sklearn.metrics import r2_score

f1 = r2_score(y_test, pred) * 100

print ("F1 Score:")
print(f1)

######################################################################
# Protected by
# Creative Commons CC BY-NC-ND 4.0 International Public License
######################################################################
```

5.1.2.2 Core Neural Network

```
# Title: Finding the sponsor from employee list
#
# CSE 400 - Dissertation: Implementation and Results
# Student     : Rashadul Islam
# Id          : 0301175301
# Department  : Department of CSE
# Semester    : Spring 2019
# Written on  : Wed Sep 10 10:00:00 BST 2019
#

import numpy as np
import pandas as pd


class Neural_Network(object):
    def __init__(self):
        # parameters
        self.inputSize = 6
        self.outputSize = 6
        self.hiddenSize = 10

        # weights
        self.W1 = np.random.randn(
            self.inputSize, self.hiddenSize
        ) # (3x2) weight matrix from input to hidden layer
        self.W2 = np.random.randn(
            self.hiddenSize, self.outputSize
        ) # (3x1) weight matrix from hidden to output layer

    def forward(self, X):
        # forward propagation through our network
        self.z = np.dot(
            X, self.W1
        ) # dot product of X (input) and first set of 3x2
    weights
        self.z2 = self.sigmoid(self.z) # activation function
        self.z3 = np.dot(
            self.z2, self.W2
        ) # dot product of hidden layer (z2) and second set of
    3x1 weights
        o = self.sigmoid(self.z3) # final activation function
        return o
```

```
41
42      def sigmoid(self, s):
43          # activation function
44          return 1 / (1 + np.exp(-s))
45
46      def sigmoidPrime(self, s):
47          # derivative of sigmoid
48          return s * (1 - s)
49
50      def backward(self, X, y, o):
51          # backward propagate through the network
52          self.o_error = y - o  # error in output
53          self.o_delta = self.o_error * self.sigmoidPrime(
54              o
55          )  # applying derivative of sigmoid to error
56
57          self.z2_error = self.o_delta.dot(
58              self.W2.T
59          )  # z2 error: how much our hidden layer weights
    contributed to output error
60          self.z2_delta = self.z2_error * self.sigmoidPrime(
61              self.z2
62          )  # applying derivative of sigmoid to z2 error
63
64          self.W1 += X.T.dot(
65              self.z2_delta
66          )  # adjusting first set (input --> hidden) weights
67          self.W2 += self.z2.T.dot(
68              self.o_delta
69          )  # adjusting second set (hidden --> output) weights
70
71      def train(self, X, y):
72          o = self.forward(X)
73          self.backward(X, y, o)
74
75      # def saveWeights(self):
76      # np.savetxt("w1.txt", self.W1, fmt="%s")
77      # np.savetxt("w2.txt", self.W2, fmt="%s")
78
79      def predict(self, predict_resources):
80          data1 = ""
81          pre = self.forward(predict_resources).astype(np.float)
82          # data1 = pd.DataFrame([pre.split(' ') for pre in pre.
    split(' ')])
83          data1 = pd.DataFrame.from_records(pre)
84
85          print("=" * 52)
86          print("Results:")
87          # print(data1)
88          print("=" * 52)
89          print("Will stay in the hotel:")
90          print(data1[0][0])
91          print("Will expense: ")
92          print(data1[1][0] * 1000)
93          print("Client's life stle status: ")
94          print(data1[2][0] * 100)
95          print("Room type: ")
96          print(data1[3][0] * 10)
97          print("Client reputation score: ")
98          print(data1[4][0] * 100)
99          print("Employee OEE score: ")
```

```
            print(data1[5][0] * 100)
            print("=" * 52)


#####################################################################
# Protected by
# Creative Commons CC BY-NC-ND 4.0 International Public License
#####################################################################
```

5.1.3 Generated Output

```
====================================================
Results:
====================================================
Will stay in the hotel:
0.9975548366026425
Will expense:
992.0554075024014
Client's life stle status:
99.0608786241832
Room type:
9.984109264400264
Client reputation score:
99.25137288175236
Employee OEE score:
99.60083548721165
====================================================
Error or loss in 1000 epoch
====================================================
0.22045368085527617
====================================================

F1 Score:
0.19601353361615725
```

5.1.4 Loss graph

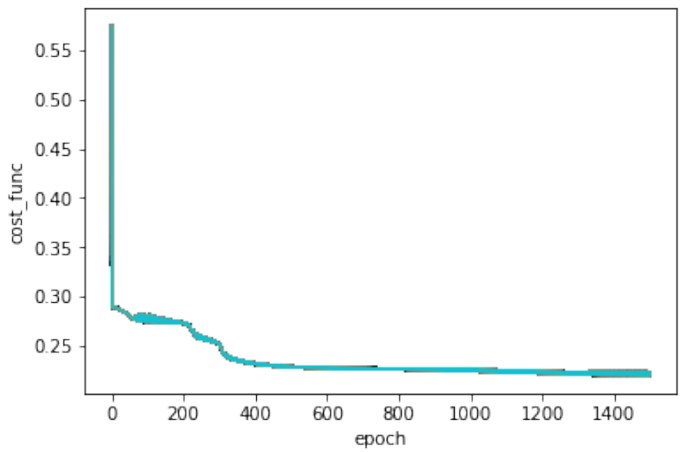


Figure 5.1: Loss or Cost Function

5.2 Finding Sponsor: Genetic Algorithm and Programming

5.2.1 Labelled Input

```
# INPUT 1:
# Historical Data: Employee Details with OEE Score

Employee id,Employee name,Client_Stays,Room_Status,
    Client_quality_score,Working_hours,Availability_score,
    Quality,Performance,Reliability_score,OEE_score
1,A,10,VIP,80,7,58.33,119.58,58.33,82.7,79.74
1,A,9,Business,75,5,41.67,107.42,41.67,85.81,69.14
1,A,8,Business,75,4,33.33,95.33,33.33,87.99,62.5
1,A,7,CIP,85,8,66.67,83.67,66.67,86.94,75.98
1,A,6,High,70,,0,71,0,93.05,41.01
.....
.....
5,D,8,Business,79,6,50,95.5,50,86.59,70.52
5,D,27,Average,61,8,66.67,323.67,66.67,58.27,128.82
5,D,15,Business,72,5,41.67,179.42,41.67,77.49,85.06
5,D,5,Below,55,7,58.33,59.58,58.33,90.94,66.8

# INPUT 2:
# Historical Data: Sales/Profit Per Employee Made with each
    client

Employee id,Employee name,Client_Stays,Room_Status,
    Client_quality_score,Working_hours,Availability_score,
    Quality,Performance,Reliability_score,OEE_score,Profit
1,A,10.00,VIP,80.00,7.00,58.33,119.58,58.33,82.70,79.74,90000
1,A,9.00,Business
    ,75.00,5.00,41.67,107.42,41.67,85.81,69.14,72000
1,A,8.00,Business,75.00,4.00,33.33,95.33,33.33,87.99,62.50,64000
1,A,7.00,CIP,85.00,8.00,66.67,83.67,66.67,86.94,75.98,70000
1,A,6.00,High,70.00,,0.00,71.00,0.00,93.05,41.01,42000
.....
.....
5,D,3.00,High,73.00,4.00,33.33,35.33,33.33,95.31,49.33,21000
5,D,8.00,Business,79.00,6.00,50.00,95.50,50.00,86.59,70.52,64000
5,D,27.00,Average
    ,61.00,8.00,66.67,323.67,66.67,58.27,128.82,162000
5,D,15.00,Business
    ,72.00,5.00,41.67,179.42,41.67,77.49,85.06,120000
5,D,5.00,Below,55.00,7.00,58.33,59.58,58.33,90.94,66.80,25000
```

5.2.2 Primary Python Code

```
# Finding the sponsor from employee list
#
# CSE 400 - Dissertation: Implementation and Results
# Student     : Rashadul Islam
# Id          : 0301175301
# Department  : Department of CSE
# Semester    : Spring 2019
# Written on  : Wed Sep 10 10:00:00 BST 2019
#

# Library
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Import Dataset as train and test as input
train = pd.read_csv("sales-per-employee.csv")
test = pd.read_csv("employee.csv")

# Preparing Dataset
number = preprocessing.LabelEncoder()

# Based on OEE_score finding the most possible sponsor from the
    employees
train["OEE_score"] = np.sqrt(train["OEE_score"])
test["OEE_score"] = np.sqrt(test["OEE_score"])

test["Profit"] = 0

combi = train.append(test)

col = ["Room_Status"]
for i in col:
    combi[i] = number.fit_transform(combi[i].astype("str"))
    combi[i] = combi[i].astype("object")

train = combi[: train.shape[0]]
test = combi[train.shape[0] :]
test.drop("Profit", axis=1, inplace=True)

# dataset preparation for tpot
tpot_train = train.drop(["Employee id"], axis=1)
tpot_test = test.drop(["Employee id"], axis=1)
tpot_train = train.drop(["Employee name", "Room_Status"], axis
    =1)
tpot_test = test.drop(["Employee name", "Room_Status"], axis=1)
target = tpot_train["Profit"]
tpot_train.drop("Profit", axis=1, inplace=True)

# To automate the machine learning
# to optimizes machine learning pipelines using genetic
    programming
# through optimization

from tpot import TPOTRegressor

X_train, X_test, y_train, y_test = train_test_split(
    tpot_train, target, train_size=0.8, test_size=0.2
)

# Genetic Algorithm
# to find the model
# please insert as more generations and population size to get
    the
# appropriate Cross Validation (C V) Score for model validation

# Pipeline optimizer

# to get the sample estimate model faster
```

```
tpot = TPOTRegressor(generations=2, population_size=10,
    verbosity=3)

# most suitable optimizer
#tpot = TPOTRegressor(verbosity=3, cv=5)

# Learning and predicting
tpot.fit(X_train, y_train)

# Estimate error or loss
print("Cross validation score")
print(tpot.score(X_test, y_test))

# Prediction
tpot_pred = tpot.predict(tpot_test)

# Results
final = pd.DataFrame(data=tpot_pred)
final = final.rename(columns={"0": "Profit"})
final["Employee id"] = test["Employee id"]
final["Employee name"] = test["Employee name"]
final["OEE_score"] = test["OEE_score"]
final["Client_Stays"] = test["Client_Stays"]
final["Room_Status"] = test["Room_Status"]
final.columns = [
    "Profit",
    "Employee id",
    "Employee name",
    "OEE_score",
    "Client_Stays",
    "Room_Status",
]
final.sort_values(by=["OEE_score"], ascending=False)

# Sponsor details based on possible sales over the client as
    profit
final["Sponsor"] = final.loc[final[["Profit", "OEE_score"]].
    idxmax()[1]]
print (final)

# Sponsor may be
print (final.loc[final[["Profit", "OEE_score"]].idxmax()[1]])
```

5.2.3 Generated Output

```
==================================
Details of the Sponsor
==================================
Profit              193506
Employee id              1
Employee name            A
OEE_score          11.6589
Client_Stays            31
Room_Status              1


Cross validation score
-27617860.591307577
```

5.2.4 Cross Validation Score

5.3 Estimation

5.3.1 Project Cost, Development time

According to Constructive Cost Model (COCOMO) Intermediate Model:

Project complexity is "Very High" which is equivallent to EAF 1.3 and Boehms definition of systems: the system is embedded system:

Cost Factor	Very Low	Low	Normal	High	Very High	Extra High
Product Complexity (EAF)	0.70	0.85	1.00	1.15	1.30	1.65

Table 5.1: Results: Cost Factor

Category	a_b	b_b	c_b	d_b
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.2	2.5	0.32

Table 5.2: Results: System types

Factors	Results	Measurement unit
Effort	3.53066272382387	Person per month
Development time	3.88767265623533	Months
Labor rate	15000	USD/m
Productivity	259.158144397608	LoC per person per month
Average staffing per month	0.908168726130052	Full time software personnel
Cost of the project	205890.913946991	USD

Table 5.3: Results: Estimate Effort, Development time, Productivity and Cost of the application

5.3.2 Project Risk

The project risk catagorize as marginal which effects the components verily: the technical performance very low, has responsive software support, has sufficient financial support and goes through a realistic, achievable schedule.

5.4 Project Management: Schedule and Effort

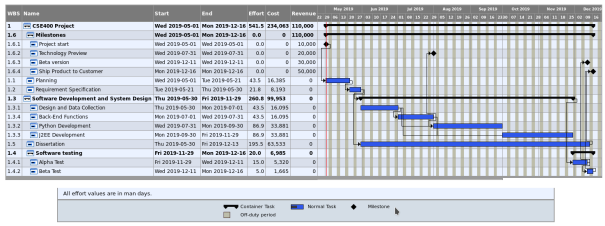


Figure 5.2: Project Schedule

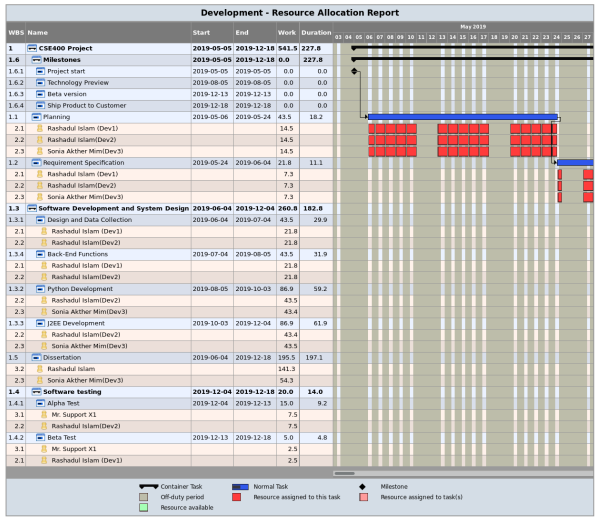


Figure 5.3: Development: Resource Allocation Report

Figure 5.4: Resource Management: Contacts

Chapter 6

Engineering ethics

Software engineering has evolved into a respected, worldwide profession. As professionals, software engineers should abide by a code of ethics that guides the work that they do and the products that they produce. An ACM/IEEE-CS Joint Task Force has produced a Software Engineering Code of Ethics and Professional Practices (Version 5.1). The code states:

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following eight principles:

i. PUBLIC: Software engineers shall act consistently with the public interest.
ii. CLIENT AND EMPLOYER: Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
iii. PRODUCTSoftware engineers shall ensure that their products and related modifications meet the highest professional standards possible.
iv. JUDGMENTSoftware engineers shall maintain integrity and independence in their professional judgment.
v. MANAGEMENTSoftware engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
vi. PROFESSIONSoftware engineers shall advance the integrity and reputation of the profession consistent with the public interest.
vii. COLLEAGUESSoftware engineers shall be fair to and supportive of their colleagues.
viii. SELFSoftware engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession

. Although each of these eight principles is equally important, an overriding theme appears: a software engineer should work in the public interest. On a personal level, a software engineer should abide by the following rules:

i. Never steal data for personal gain.

ii. Never distribute or sell proprietary information obtained as part of your work on a software project.
iii. Never maliciously destroy or modify another persons programs, files, or data.
iv. Never violate the privacy of an individual, a group, or an organization.
v. Never enter into a system for sport or profit.
vi. Never create or promulgate a computer virus or worm.
vii. Never use computing technology to facilitate discrimination or harassment.

Software industry based protective legislation are:

i. Allows companies to release software without disclosing known defects
ii. Exempts developers from liability for any damages resulting from these known defects
iii. Constrains others from disclosing defects without permission from the original developer
iv. Allows the incorporation of "self-help" software within a product that can disable (via remote command) the operation of the product
v. Exempts developers of software with "self-help" from damages should the software be disabled by a third party.

Chapter 7

Engineering limitations

i. Software hazards occurs while maintaining higher generations in genetic algorithm which considered as major severity and likelihood of the occurance is probable.
ii. Costs is estimated higher cause of technical and research skills.
iii. Optimized model takes enough time:
    - Genetic algorithm takes 4+ hours in CPU for real world implementation
    - Feed forward backpropagation algorithm takes 10-15+ hours in GPU
iv. Need team work instead individual effort.
v. Literary possible but industrially incomplete because of the inefficient loss function.
vi. Not enough library access for content and bibliography.
vii. Ethical standards and legislation are not industrially exercised.
viii. Halting problem.
ix. Takes enough memory and space.
x. Complexity is very high.
xi. Big O notation is complex to calculate.
xii. Sound knowledge does not exist to the owners to purchase such application.
xiii. Integration of Python and J2EE is a highly complex procedure.
xiv. Industrial dataset is not publicly available.
xv. Carbon footprints are large in size.

Chapter 8

Impacts

8.1 Impacts

i. Impacting nearly every technological aspect of society.
ii. Environmental impact in climate change and change management.
iii. Fewer errors.
iv. Able to perform repetitive tasks.
v. Difficulty exploration with continuity with greater responsibility.
vi. Artificial thinker and decision surfer.
vii. Can available $24 \times 7 \times 365$
viii. Domain and industrial expert and advisor.

8.2 Future plan

The future of this project is:

i. Automated industrial live software application which may be implemented in health-care, hospitality, manufacturing, sales-engineering, dynamic decision making

Such application is for those who belongs to the competitive edge of technology to sustain in the highly trendy, large scale competition and production with market research and searching & asking for real time decision making environment.

Chapter 9

Conclusion

Artificial Intelligence is the pretty innovation till now. With Artificial Intelligence, one can travel like light years or imagination of precisions. While working on the project and this dissertation, the industrial scale implementation brought the new thought of historical data driven predictions using Feed-forward Backpropagation algorithm and genetic algorithm. Any sector wants to implement automation their hospitality or host the new client can implement such solution or architecture to maintain a high quality service. The developed application pass the labelled data to the system and generates a wise and prestigious premium prediction from the inventory to hosting-the-client culture in any corporate culture. The next generation emerging technology, with this application, enhances the quality of life, service and product while maintaining a healthy relationship with all stakeholders. The purpose of the dissertation is not only limited with this scope of work, but also finding sponsor who can host the client according to clients previous history or similar pattern of the client. Genetic algorithm, herself, is an evolutionary development of the software engineering community. With genetic algorithm, the system can find the most fitted employee from the worker pool who has ranking for their arts and works while earning profit for the organization. Preparing this dissertation involves two phase: in the first phase - the application has been developed & and implemented with minor hazards, and with CMMI level experience and software project management industrial skills; in the second phase - the documentation leads through the complete academic contexts and diversity. Technical limitations exists while developing such application while literal limitation are not escapable or avoidable. Finally, the application is worthy to spend with in terms of technical, corporate, accountability and quality - which estimated as a state-of-art project & system in the view of software engineering.

[square]natbib

Bibliography

[Aggarwal 18] C.C. Aggarwal. Neural networks and deep learning: A textbook. Springer International Publishing, 2018.

[Agrawal] Apoorva Agrawal. *Loss Functions and Optimization Algorithms. Demystified.*

[Arpit 16] Devansh Arpit, Yingbo Zhou, Bhargava U Kota & Venu Govindaraju. *Normalization Propagation: A Parametric Technique for Removing Internal Covariate Shift in Deep Networks.* arXiv preprint arXiv:1603.01431, 2016.

[Babbie 16] R. Babbie. The basics of social research. Cengage Learning, 2016.

[Bellman 78] R.E. Bellman. Artificial intelligence: can computers think? Boyd & Fraser Pub. Co., 1978.

[Brooks 95] J.F.P. Brooks. The mythical man-month: Essays on software engineering, anniversary edition, 2/e. Pearson Education, 1995.

[Charniak 85] E. Charniak. Introduction to artificial intelligence. ADDISON-WESLEY SERIES IN COMPUTER SCIENCE. Pearson Education, 1985.

[Cooijmans 16] Tim Cooijmans, Nicolas Ballas, César Laurent & Aaron Courville. *Recurrent Batch Normalization.* arXiv preprint arXiv:1603.09025, 2016.

[Deng 14] L. Deng & D. Yu. Deep learning: Methods and applications. Foundations and Trends(r) in Signal Processing Series. Now Publishers, 2014.

[Fowler 12] M. Fowler. Patterns of enterprise application architecture: Pattern enterpr applica arch. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012.

[Given 08] L.M. Given. The sage encyclopedia of qualitative research methods: A-l ; vol. 2, m-z index. A Sage Reference Publication. SAGE Publications, 2008.

[Glorot 11] X. Glorot, A. Bordes & Y. Bengio. *Deep Sparse Rectifier Neural Networks.* In AISTATS'2011, 2011.

[GN] Chethan Kumar GN. *Artificial Intelligence: Definition, Types, Examples, Technologies.*

[Haugeland 89] J. Haugeland. Artificial intelligence: The very idea. A Bradford book. MIT Press, 1989.

[Hebb 49] D. O. Hebb. The organization of behavior. Wiley, New York, 1949.

[Hinton 99] G.E. Hinton, T.J. Sejnowski, H.H.M.I.C.N.L.T.J. Sejnowski & T.A. Poggio. Unsupervised learning: Foundations of neural computation. A Bradford Book. MCGRAW HILL BOOK Company, 1999.

[Ioffe 15] Sergey Ioffe & Christian Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* 2015.

[Jarrett 09] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato & Yann LeCun. *What is the Best Multi-Stage Architecture for Object Recognition?* In ICCV'09, 2009.

[Krizhevsky 12] Alex Krizhevsky, Ilya Sutskever & Geoffrey E Hinton. *Imagenet classification with deep convolutional neural networks.* In Advances in neural information processing systems, pages 1097–1105, 2012.

[Kurzweil 92] R. Kurzweil. The age of intelligent machines. Viking, 1992.

[Larman 04] C. Larman. Agile and iterative development: A manager's guide. Agile software development series. Addison-Wesley, 2004.

[Laurent 15] César Laurent, Gabriel Pereyra, Philemon Brakel, Ying Zhang & Yoshua Bengio. *Batch Normalized Recurrent Neural Networks.* CoRR, vol. abs/1510.01378, 2015.

[Le 19] TT Le, W Fu & JH Moore. *Scaling tree-based automated machine learning to biomedical big data with a feature set selector.* Bioinformatics (Oxford, England), 2019.

[LeCun 15] Yann LeCun, Yoshua Bengio & Geoffrey Hinton. *Deep Learning.* Nature, vol. 521, pages 436–444, 2015.

[Linnainmaa 70] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki, 1970.

[Mac] *101 Machine Learning Algorithms for Data Science with Cheat Sheets | R-bloggers.*

[McConnell 04] S. McConnell. Code complete. Developer Best Practices. Pearson Education, 2004.

[McCulloch 43] W. S. McCulloch & W. Pitts. *A Logical Calculus of Ideas Immanent in Nervous Activity.* The bulletin of mathematical biophysics, vol. 5, pages 115–133, 1943.

[Minsky 69] M. L. Minsky & S. A. Papert. Perceptrons. MIT Press, Cambridge, 1969.

[Minsky 17] M. Minsky, S.A. Papert & L. Bottou. Perceptrons: An introduction to computational geometry. The MIT Press. MIT Press, 2017.

[Mitchell 98] M. Mitchell. An introduction to genetic algorithms. A Bradford book. Bradford Books, 1998.

[Murphy 12] K.P. Murphy. Machine learning: A probabilistic perspective. Adaptive Computation and Machine Learning series. MIT Press, 2012.

[Nair 10] V. Nair & G. E Hinton. *Rectified linear units improve restricted Boltzmann machines*. In Proc. 27th International Conference on Machine Learning, 2010.

[NAT] *NATO Software Engineering Conference. Garmisch, Germany, 7th to 11th October 1968 - nato1968e.pdf*.

[Nilsson 98] N.J. Nilsson. Artificial intelligence: A new synthesis. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science, 1998.

[Pfleeger 06] S.L. Pfleeger & J.M. Atlee. Software engineering: Theory and practice. Pearson Prentice Hall, 2006.

[Poole 98] D.I. Poole, D.L. Poole, R.A. GOEBEL, D. Poole, A.K. Mackworth, A. Mackworth & R. Goebel. Computational intelligence: A logical approach. Oxford University Press, 1998.

[Pressman 05] R.S. Pressman. Software engineering: A practitioner's approach. McGraw-Hill higher education. McGraw-Hill, 2005.

[Reed 99] R. Reed & R.J. MarksII. Neural smithing: Supervised learning in feedforward artificial neural networks. A Bradford Book. MIT Press, 1999.

[Rich 91] E. Rich & K. Knight. Artificial intelligence. Artificial Intelligence Series. McGraw-Hill, 1991.

[Rosenblatt 57] Frank Rosenblatt. *The Perceptron — a perceiving and recognizing automaton*. Rapport technique 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y., 1957.

[Rumelhart 86a] David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. *Learning Representations by Back-Propagating Errors*. Nature, vol. 323, pages 533–536, 1986.

[Rumelhart 86b] David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. *Learning representations by back-propagating errors*. Nature, vol. 323, no. 6088, pages 533–536, October 1986.

[Russell 03] S.J. Russell, P. Norvig & J. Canny. Artificial intelligence: A modern approach. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.

[Sommerville 07] I. Sommerville. Software engineering. International computer science series Software engineering. Addison-Wesley, 2007.

[Sutton 18] R.S. Sutton & A.G. Barto. Reinforcement learning: An introduction. Adaptive Computation and Machine Learning series. MIT Press, 2018.

[Tan 12] N.H. Tan, L.S.S. Ho & L.H. Keung. Computational intelligence and its applications: Evolutionary computation,

fuzzy logic, neural network and support vector machine techniques. World Scientific Publishing Company, 2012.

[Typ] *Types of Loss Function.*

[Weber 08] C. Weber, M. Elshaw & N.M. Mayer. Reinforcement learning. IntechOpen, 2008.

[Werbos 74] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[Wha] *What is Machine Learning? A definition - Expert System.*

[Winston 92] P.H. Winston. Artificial intelligence. A-W Series in Computerscience. Addison-Wesley Publishing Company, 1992.