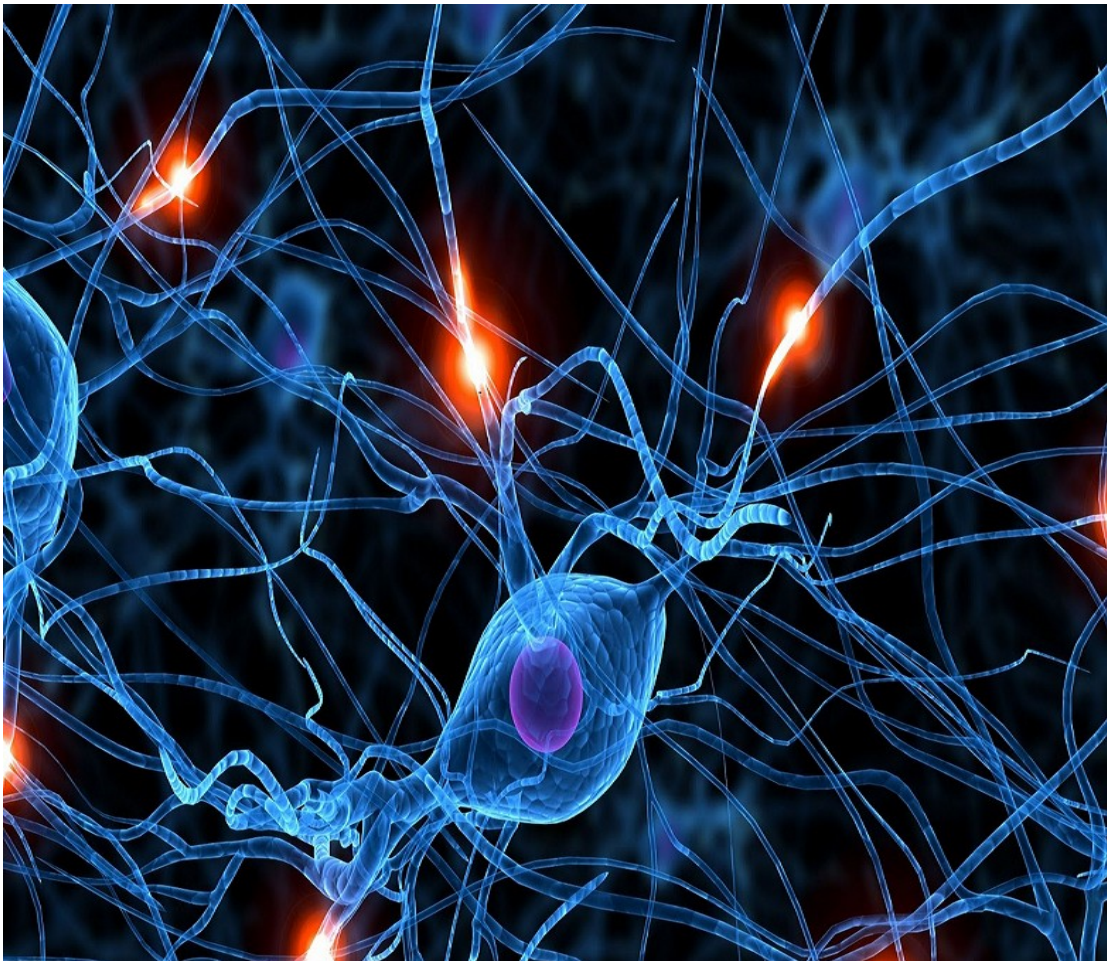
Σχολη Τεχνολογικων Εφαρμογων

Τμημα Αυτοματισμου

Πτυχιακη Εργασια

Θεμα:

Μηχανική Μάθηση με χρήση Νευρωνικού Δικτύου



ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:ΜΙΧΑΛΗΣ ΠΑΠΟΥΤΣΙΔΑΚΗΣ

Φοιτητης: Δημητριος Μπουλουγαρης Α.Μ 34419

Αιγάλεω Μαιος 2013

Στόχος της πτυχιακής αυτής εργασίας είναι ο συνδυασμός της μηχανικής μάθησης και των νευρωνικών δικτύων για τη δημιουργία ενός προγράμματος επιτηρούμενης μάθησης(Supervised learning) που αρχικά θα είναι ικανό να αναγνωρίσει χειρόγραφα ψηφία από το 1 εως το 9,που όμως θα είναι ικανό για αναγνώριση οποιουδήποτε test set δεδομένου ότι δίδεται το αντίστοιχο training set του.

Για την απόδειξη αυτού θα δοθεί ένα σαφώς μικρότερο training set με χειρόγραφα γράμματα της Αγγλικής αλφαβήτου για την αναγνώριση νέων χειρόγραφων γραμμάτων.

Περιεχόμενα

1. Εισαγωγη :

. Τι είναι τεχνητή νοημοσύνη

. Σύντομη ιστορική αναδρομή της εξέλιξης του πεδίου της τεχνητής νουμοσηνής

2. Μηχανική μάθηση

. Μη επιτηρούμενη μάθηση

. Επιτηρούμενη μάθηση

. Ταξινόμηση

. Ανάλυση παλινδρόμησης

. Ενισχυόμενη μάθηση

3.Νευρωνικά δίκτυα

. Η ανάγκη για τα νευρωνικά δίκτυα(ευθύγραμμη και μη ευθύγραμμη υπόθεση)

. Η αναπαράσταση ενός νευρωνικού δικτύου

3α. Η συνάρτηση κόστους στην Λογιστική Ανάλυση Παλινδρόμησης και στα νευρωνικά δύκτια

. Η ελαχιστοποίηση της συνάρτησης κόστους

. Υπολογισμός τον μερικών παραγώγων
.Forward propagation
.Back propagation

. Άριθμητικός έλεγχος κλήσης και τυχαίο ξεκίνημα συναπτικών βαρών

3β. Φτιαχνοντας ένα νευρωνικό δίκτυο

3γ. Συνήθη προβλήματα και η επίλυση τους
.Αξιολογόντας τον αλγόριθμο
. Αξιολογόντας την υποθέση
. Διάγνωση και διακύμανσης

3δ.Προγραμματίζοντας ενα νευρωνικο δικτυο σε Octave gnu-Matlab
4.Οδηγος για την εγκατασταση του Octave Gnu σε Windows
5.Βιβλιογραφεια

1.Εισαγώγη:

Artificial Intelligence:

Ο όρος τεχνητή νοημοσύνη (ΤΝ, εκ του Artificial Intelligence) αναφέρεται στον κλάδο της επιστήμης υπολογιστών ο οποίος ασχολείται με τη σχεδίαση και την υλοποίηση υπολογιστικών συστημάτων που μιμούνται στοιχεία της ανθρώπινης συμπεριφοράς τα οποία υπονοούν έστω και στοιχειώδη ευφυΐα: μάθηση, προσαρμοστικότητα, εξαγωγή συμπερασμάτων, κατανόηση από συμφραζόμενα, επίλυση προβλημάτων κλπ.

Ο Τζον Μακάρθι όρισε τον τομέα αυτόν ως «επιστήμη και μεθοδολογία της δημιουργίας νοούντων

μηχανών».(1965)

Η τεχνητη νοημοσυνη αποτελει αντικειμενο το οποιο παρολο μελετατε για δεκαετιες ακομα δεν εχει λυθει κυριως λογο του οτι καλυπτει ενα πραγματικα τεραστιο τομεα.
Το πεδιο της τεχνητη νοημοσυνη απλονεται απο μηχανες που ειναι πραγματικα ικανες για σκεψη εως αλγοριθμους αναζητησης που χρησιμοποιοιουντε σε επιτραπεζια παιχνιδια και εχει εφαρμογες σχεδον σε οτιδηποτε χρηση κανουμε σημερα με τους υπολογιστες σαν κοινωνια

Η Ιστορία :

Η ιστορια της τεχνιτης νοημοσυνης ξεκίνησε την αρχαιότητα , με τους μύθους, ιστορίες και φήμες τεχνητών όντων προικισμένα με τη νοημοσύνη και συνείδηση .Η Pamela McCorduck γράφει, Η τεχνητη νοημοσηνη ξεκίνησε με την αρχαία επιθυμία να σφυρηλατήσει τους θεούς.

Οι απαρχές της Τεχνητής Νοημοσύνης ανάγονται στους "συλλογισμούς" του Αριστοτέλη (384-322 π.Χ.), οι οποίοι παρείχαν πρότυπα εκφράσεων που έδιναν πάντα σωστά συμπεράσματα από σωστές υποθέσεις (Αριστοτέλεια συλλογιστική).

Οι σπόροι της σύγχρονης AI φυτεύτηκαν από κλασικους φιλοσόφους που προσπάθησαν να περιγράψουν τη διαδικασία της ανθρώπινης σκέψης ως το μηχανικό χειρισμό των συμβόλων . Η εργασία αυτή κατέληξε στην εφεύρεση του προγραμματιζόμενου ψηφιακό υπολογιστήστη τη δεκαετία του 1940, μια μηχανή με βάση την αφηρημένη ουσία της μαθηματικής αιτιολογίας. Αυτή

η συσκευή και οι ιδέες πίσω από αυτό ενέπνευσε μια χούφτα των επιστημόνων να αρχίσουν σοβαρά συζητούν το ενδεχόμενο της δημιουργίας ενός ηλεκτρονικού εγκεφάλου.

Ο όρος τεχνητη νοημοσύνη πρωτοαναφερθηκε το 1956 στο συνδεριο του Dartmouth και απο τοτε η τεχνητη νοημοσυνη εξαπλωθηκε κυριως λογο των αρχων και θεωριων των αφοσιωμενων σε αυτην ερευνητες. Παρόλο την μικρή της σύχρονη ιστορία τα βήματα προόδου που εχουν είναι μικροότερα απο ότι αναμενοταν.
Απο τη γέννηση της 57 χρόνια πριν εχουν υπαρξη ποικιλα προγραμματα τεχνητης νοημοσυνης και εχουν επηρεασει την τεχνολογικη προοδο ποικιλοτροπος

Πινακας Ιστορικης Αναδρομης της Ιστοριας εξελιγης της τεχνιτης νουμοσηνης.
(Εισαγωγη απο wikipedia)

Το 1900

Date	Development
Antiquity	Greek myths of Hephaestus and Pygmalion incorporated the idea of intelligent robots (such as Talos) and artificial beings (such asGalatea and Pandora).[1]
Antiquity	Yan Shi presented King Mu of Zhou with mechanical men.[2]
Antiquity	Sacred mechanical statues built in Egypt and Greece were believed to be capable of wisdom and emotion. Hermes Trismegistuswould write "they have *sensus* and *spiritus* ... by discovering the true nature of the gods, man has been able to reproduce it." Mosaiclaw prohibits the use of automatons in religion.[3]
384 BC–322 BC	Aristotle described the syllogism, a method of formal, mechanical thought.
1st century	Heron of Alexandria created mechanical men and other automatons.[4]
260	Porphyry of Tyros wrote *Isagogê* which categorized knowledge and logic.[5]
~800	Geber develops the Arabic alchemical theory of *Takwin*, the artificial creation of life in the laboratory, up to and including human life.[6]
1206	Al-Jazari created a programmable orchestra of mechanical human beings.[7]
1275	Ramon Llull, Catalan theologian invents the Ars Magna, a tool for combining concepts mechanically, based on an Arabic astrologicaltool, the Zairja. The method would be developed further by Gottfried Leibniz in the 17th century.[8]
~1500	Paracelsus claimed to have created an artificial man out of magnetism, sperm and alchemy.[9]
~1580	Rabbi Judah Loew ben Bezalel of Prague is said to have invented the Golem, a clay man brought to life.[10]
Early 17th century	René Descartes proposed that bodies of animals are nothing more than complex machines (but that mental phenomena are of a different "substance").[11]

1623	Wilhelm Schickard drew a calculating clock on a letter to Kepler. This will be the first of five unsuccessful attempts at designing a*direct entry* calculating clock in the 17th century (including the designs of Tito Burattini, Samuel Morland and René Grillet)).[12]
1641	Thomas Hobbes published *Leviathan* and presented a mechanical, combinatorial theory of cognition. He wrote "...for reason is nothing but reckoning".[13][14]
1642	Blaise Pascal invented the mechanical calculator,[15] the first digital calculating machine[16]
1672	Gottfried Leibniz improved the earlier machines, making the Stepped Reckoner to do multiplication and division. He also invented thebinary numeral system and envisioned a universal calculus of reasoning (alphabet of human thought) by which arguments could be decided mechanically. Leibniz worked on assigning a specific number to each and every object in the world, as a prelude to an algebraic solution to all possible problems.[17]
1727	Jonathan Swift published *Gulliver's Travels*, which includes this description of the Engine, a machine on the island of Laputa: "a Project for improving speculative Knowledge by practical and mechanical Operations " by using this "Contrivance", "the most ignorant Person at a reasonable Charge, and with a little bodily Labour, may write Books in Philosophy, Poetry, Politicks, Law, Mathematicks, and Theology, with the least Assistance from Genius or study."[18] The machine is a parody of Ars Magna, one of the inspirations ofGottfried Leibniz' mechanism.
1750	Julien Offray de La Mettrie published *L'Homme Machine*, which argued that human thought is strictly mechanical.[19]
1769	Wolfgang von Kempelen built and toured with his chess-playing automaton, The Turk.[20] The Turk was later shown to be a hoax, involving a human chess player.
1818	Mary Shelley published the story of *Frankenstein; or the Modern Prometheus*, a fictional consideration of the ethics of creatingsentient beings.[21]
1822–1859	Charles Babbage & Ada Lovelace worked on programmable mechanical calculating machines.[22]
1837	The mathematician Bernard Bolzano made the first modern attempt to formalize semantics.
1854	George Boole set out to "investigate the fundamental laws of those operations of the mind by which reasoning is performed, to give expression to them in the symbolic language of a calculus", inventing Boolean algebra.[23]
1863	Samuel Butler suggested that Darwinian evolution also applies to machines, and speculates that they will one day become conscious and eventually supplant humanity.[24]

1900–1950

Date	Development
1913	Bertrand Russell and Alfred North Whitehead published *Principia Mathematica,* which revolutionized formal logic.
1915	Leonardo Torres y Quevedo built a chess automaton, El Ajedrecista and published

	speculation about thinking and automata.[25]
1923	Karel Čapek's play *R.U.R. (Rossum's Universal Robots)* opened in London. This is the first use of the word "robot" in English.[26]
1920s and 1930s	Ludwig Wittgenstein and Rudolf Carnap lead philosophy into logical analysis of knowledge. Alonzo Church develops Lambda Calculusto investigate computability using recursive functional notation.
1931	Kurt Gödel showed that sufficiently powerful consistent formal systems permit the formulation of true theorems that are unprovable by any theorem-proving machine deriving all possible theorems from the axioms. To do this he had to build a universal, integer-based programming language, which is the reason why he is sometimes called the "father of theoretical computer science".
1941	Konrad Zuse built the first working program-controlled computers.[27]
1943	Warren Sturgis McCulloch and Walter Pitts publish "A Logical Calculus of the Ideas Immanent in Nervous Activity" (1943), laying foundations for artificial neural networks.[28]
1943	Arturo Rosenblueth, Norbert Wiener and Julian Bigelow coin the term "cybernetics". Wiener's popular book by that name published in 1948.
1945	Game theory which would prove invaluable in the progress of AI was introduced with the 1944 paper, Theory of Games and Economic Behavior by mathematician John von Neumann and economist Oskar Morgenstern.
1945	Vannevar Bush published *As We May Think* (The Atlantic Monthly, July 1945) a prescient vision of the future in which computers assist humans in many activities.
1948	John von Neumann (quoted by E.T. Jaynes) in response to a comment at a lecture that it was impossible for a machine to think: "You insist that there is something a machine cannot do. If you will tell me *precisely* what it is that a machine cannot do, then I can always make a machine which will do just that!". Von Neumann was presumably alluding to the Church-Turing thesis which states that any effective procedure can be simulated by a (generalized) computer.

1950s

Date	Development
1950	Alan Turing proposes the Turing Test as a measure of machine intelligence.[29]
1950	Claude Shannon published a detailed analysis of chess playing as search.
1950	Isaac Asimov published his Three Laws of Robotics.
1951	The first working AI programs were written in 1951 to run on the Ferranti Mark 1 machine of the University of Manchester: a checkers-playing program written by Christopher Strachey and a chess-playing program written by Dietrich Prinz.
1952–1962	Arthur Samuel (IBM) wrote the first game-playing program,[30] for checkers (draughts), to achieve sufficient skill to challenge a respectable amateur. His first checkers-playing program was written in 1952, and in 1955 he created a version that learned to play.[31]

1956	The first Dartmouth College summer AI conference is organized by John McCarthy, Marvin Minsky, Nathan Rochester of IBM andClaude Shannon.
1956	The name *artificial intelligence* is used for the first time as the topic of the second Dartmouth Conference, organized by John McCarthy[32]
1956	The first demonstration of the Logic Theorist (LT) written by Allen Newell, J.C. Shaw and Herbert A. Simon (Carnegie Institute of Technology, now Carnegie Mellon University). This is often called the first AI program, though Samuel's checkers program also has a strong claim.
1957	The General Problem Solver (GPS) demonstrated by Newell, Shaw and Simon.
1958	John McCarthy (Massachusetts Institute of Technology or MIT) invented the Lisp programming language.
1958	Herbert Gelernter and Nathan Rochester (IBM) described a theorem prover in geometry that exploits a semantic model of the domain in the form of diagrams of "typical" cases.
1958	Teddington Conference on the Mechanization of Thought Processes was held in the UK and among the papers presented were John McCarthy's *Programs with Common Sense,* Oliver Selfridge's *Pandemonium,* and Marvin Minsky's *Some Methods of HeuristicProgramming and Artificial Intelligence.*
1959	John McCarthy and Marvin Minsky founded the MIT AI Lab.
Late 1950s, early 1960s	Margaret Masterman and colleagues at University of Cambridge design semantic nets for machine translation.

1960s

Date	Development
1960s	Ray Solomonoff lays the foundations of a mathematical theory of AI, introducing universal Bayesian methods for inductive inference and prediction.
1960	*Man-Computer Symbiosis* by J.C.R. Licklider.
1961	James Slagle (PhD dissertation, MIT) wrote (in Lisp) the first symbolic integration program, SAINT, which solved calculus problems at the college freshman level.
1961	In *Minds, Machines and Gödel*, John Lucas[33] denied the possibility of machine intelligence on logical or philosophical grounds. He referred to Kurt Gödel's result of 1931: sufficiently powerful formal systems are either inconsistent or allow for formulating true theorems unprovable by any theorem-proving AI deriving all provable theorems from the axioms. Since humans are able to "see" the truth of such theorems, machines were deemed inferior.
1962	First industrial robot company, Unimation, founded.
1963	Thomas Evans' program, ANALOGY, written as part of his PhD work at MIT, demonstrated that computers can solve the sameanalogy problems as are given on IQ tests.

1963	Edward Feigenbaum and Julian Feldman published *Computers and Thought*, the first collection of articles about artificial intelligence.
1963	Leonard Uhr and Charles Vossler published "A Pattern Recognition Program That Generates, Evaluates, and Adjusts Its Own Operators", which described one of the first machine learning programs that could adaptively acquire and modify features and thereby overcome the limitations of simple perceptrons of Rosenblatt
1964	Danny Bobrow's dissertation at MIT (technical report #1 from MIT's AI group, Project MAC), shows that computers can understand natural language well enough to solve algebra word problems correctly.
1964	Bertram Raphael's MIT dissertation on the SIR program demonstrates the power of a logical representation of knowledge for question-answering systems.
1965	J. Alan Robinson invented a mechanical proof procedure, the Resolution Method, which allowed programs to work efficiently with formal logic as a representation language.
1965	Joseph Weizenbaum (MIT) built ELIZA, an interactive program that carries on a dialogue in English language on any topic. It was a popular toy at AI centers on the ARPANET when a version that "simulated" the dialogue of a psychotherapist was programmed.
1965	Edward Feigenbaum initiated Dendral, a ten-year effort to develop software to deduce the molecular structure of organic compounds using scientific instrument data. It was the first expert system.
1966	Ross Quillian (PhD dissertation, Carnegie Inst. of Technology, now CMU) demonstrated semantic nets.
1966	Machine Intelligence workshop at Edinburgh – the first of an influential annual series organized by Donald Michie and others.
1966	Negative report on machine translation kills much work in Natural language processing (NLP) for many years.
1967	Dendral program (Edward Feigenbaum, Joshua Lederberg, Bruce Buchanan, Georgia Sutherland at Stanford University) demonstrated to interpret mass spectra on organic chemical compounds. First successful knowledge-based program for scientific reasoning.
1968	Joel Moses (PhD work at MIT) demonstrated the power of symbolic reasoning for integration problems in the Macsyma program. First successful knowledge-based program in mathematics.
1968	Richard Greenblatt (programmer) at MIT built a knowledge-based chess-playing program, MacHack, that was good enough to achieve a class-C rating in tournament play.
1968	Wallace and Boulton's program, Snob (Comp.J. 11(2) 1968), for unsupervised classification (clustering) uses the Bayesian Minimum Message Length criterion, a mathematical realisation of Occam's razor.
1969	Stanford Research Institute (SRI): Shakey the Robot, demonstrated combining animal locomotion, perception and problem solving.
1969	Roger Schank (Stanford) defined conceptual dependency model for natural language understanding. Later developed (in PhD dissertations at Yale University) for use in story understanding by Robert Wilensky and Wendy Lehnert, and for use in understanding memory by Janet Kolodner.

1969	Yorick Wilks (Stanford) developed the semantic coherence view of language called Preference Semantics, embodied in the first semantics-driven machine translation program, and the basis of many PhD dissertations since such as Bran Boguraev and David Carter at Cambridge.
1969	First International Joint Conference on Artificial Intelligence (IJCAI) held at Stanford.
1969	Marvin Minsky and Seymour Papert publish *Perceptrons*, demonstrating previously unrecognized limits of this feed-forward two-layered structure. This book is considered by some to mark the beginning of the AI winter of the 1970s, a failure of confidence and funding for AI. Nevertheless significant progress in the field continued (see below).
1969	McCarthy and Hayes started the discussion about the frame problem with their essay, "Some Philosophical Problems from the Standpoint of Artificial Intelligence".

1970s

Date	Development
Early 1970s	Jane Robinson and Don Walker established an influential Natural Language Processing group at SRI.
1970	Jaime Carbonell (Sr.) developed SCHOLAR, an interactive program for computer assisted instruction based on semantic nets as the representation of knowledge.
1970	Bill Woods described Augmented Transition Networks (ATN's) as a representation for natural language understanding.
1970	Patrick Winston's PhD program, ARCH, at MIT learned concepts from examples in the world of children's blocks.
1971	Terry Winograd's PhD thesis (MIT) demonstrated the ability of computers to understand English sentences in a restricted world of children's blocks, in a coupling of his language understanding program, SHRDLU, with a robot arm that carried out instructions typed in English.
1971	Work on the Boyer-Moore theorem prover started in Edinburgh.[34]
1972	Prolog programming language developed by Alain Colmerauer.
1972	Earl Sacerdoti developed one of the first hierarchical planning programs, ABSTRIPS.
1973	The Assembly Robotics Group at University of Edinburgh builds Freddy Robot, capable of using visual perception to locate and assemble models. (See Edinburgh *Freddy* Assembly Robot: a versatile computer-controlled assembly system.)
1973	The Lighthill report gives a largely negative verdict on AI research in Great Britain and forms the basis for the decision by the British government to discontinue support for AI research in all but two universities.
1974	Ted Shortliffe's PhD dissertation on the MYCIN program (Stanford) demonstrated a very practical rule-based approach to medical diagnoses, even in the presence of uncertainty. While it borrowed from DENDRAL, its own contributions strongly influenced the future of expert system development, especially commercial systems.
1975	Earl Sacerdoti developed techniques of partial-order planning in his NOAH system,

	replacing the previous paradigm of search among state space descriptions. NOAH was applied at SRI International to interactively diagnose and repair electromechanical systems.
1975	Austin Tate developed the Nonlin hierarchical planning system able to search a space of partial plans characterised as alternative approaches to the underlying goal structure of the plan.
1975	Marvin Minsky published his widely-read and influential article on Frames as a representation of knowledge, in which many ideas about schemas and semantic links are brought together.
1975	The Meta-Dendral learning program produced new results in chemistry (some rules of mass spectrometry) the first scientific discoveries by a computer to be published in a refereed journal.
Mid 1970s	Barbara Grosz (SRI) established limits to traditional AI approaches to discourse modeling. Subsequent work by Grosz, Bonnie Webber and Candace Sidner developed the notion of "centering", used in establishing focus of discourse and anaphoric references inNatural language processing.
Mid 1970s	David Marr and MIT colleagues describe the "primal sketch" and its role in visual perception.
1976	Douglas Lenat's AM program (Stanford PhD dissertation) demonstrated the discovery model (loosely-guided search for interesting conjectures).
1976	Randall Davis demonstrated the power of meta-level reasoning in his PhD dissertation at Stanford.
1978	Tom Mitchell, at Stanford, invented the concept of Version spaces for describing the search space of a concept formation program.
1978	Herbert A. Simon wins the Nobel Prize in Economics for his theory of bounded rationality, one of the cornerstones of AI known as "satisficing".
1978	The MOLGEN program, written at Stanford by Mark Stefik and Peter Friedland, demonstrated that an object-oriented programmingrepresentation of knowledge can be used to plan gene-cloning experiments.
1979	Bill VanMelle's PhD dissertation at Stanford demonstrated the generality of MYCIN's representation of knowledge and style of reasoning in his EMYCIN program, the model for many commercial expert system "shells".
1979	Jack Myers and Harry Pople at University of Pittsburgh developed INTERNIST, a knowledge-based medical diagnosis program based on Dr. Myers' clinical knowledge.
1979	Cordell Green, David Barstow, Elaine Kant and others at Stanford demonstrated the CHI system for automatic programming.
1979	The Stanford Cart, built by Hans Moravec, becomes the first computer-controlled, autonomous vehicle when it successfully traverses a chair-filled room and circumnavigates the Stanford AI Lab.
1979	BKG, a backgammon program written by Hans Berliner at CMU, defeats the reigning world champion.
1979	Drew McDermott and Jon Doyle at MIT, and John McCarthy at Stanford begin publishing

	work on non-monotonic logics and formal aspects of truth maintenance.
Late 1970s	Stanford's SUMEX-AIM resource, headed by Ed Feigenbaum and Joshua Lederberg, demonstrates the power of the ARPAnet for scientific collaboration.

1980s

Date	Development
Early 1980s	The team of Ernst Dickmanns at Bundeswehr University of Munich builds the first robot cars, driving up to 55 mph on empty streets.
1980s	Lisp machines developed and marketed. First expert system shells and commercial applications.
1980	First National Conference of the American Association for Artificial Intelligence (AAAI) held at Stanford.
1981	Danny Hillis designs the connection machine, which utilizes Parallel computing to bring new power to AI, and to computation in general. (Later founds Thinking Machines Corporation)
1982	The Fifth Generation Computer Systems project (FGCS), an initiative by Japan's Ministry of International Trade and Industry, begun in 1982, to create a "fifth generation computer" (see history of computing hardware) which was supposed to perform much calculation utilizing massive parallelism.
1983	John Laird and Paul Rosenbloom, working with Allen Newell, complete CMU dissertations on Soar (program).
1983	James F. Allen invents the Interval Calculus, the first widely used formalization of temporal events.
Mid 1980s	Neural Networks become widely used with the Backpropagation algorithm (first described by Paul Werbos in 1974).
1985	The autonomous drawing program, AARON, created by Harold Cohen, is demonstrated at the AAAI National Conference (based on more than a decade of work, and with subsequent work showing major developments).
1987	Marvin Minsky published *The Society of Mind*, a theoretical description of the mind as a collection of cooperating agents. He had been lecturing on the idea for years before the book came out (c.f. Doyle 1983).
1987	Around the same time, Rodney Brooks introduced the subsumption architecture and behavior-based robotics as a more minimalist modular model of natural intelligence; Nouvelle AI.
1989	Dean Pomerleau at CMU creates ALVINN (An Autonomous Land Vehicle in a Neural Network).

1990s

Date	Development
Early 1990s	TD-Gammon, a backgammon program written by Gerry Tesauro, demonstrates that reinforcement (learning) is powerful enough to create a championship-level game-

	playing program by competing favorably with world-class players.
1990s	Major advances in all areas of AI, with significant demonstrations in machine learning, intelligent tutoring, case-based reasoning, multi-agent planning, scheduling, uncertain reasoning, data mining, natural language understanding and translation, vision, virtual reality, games, and other topics.
1991	DART scheduling application deployed in the first Gulf War paid back DARPA's investment of 30 years in AI research.[35]
1993	Ian Horswill extended behavior-based robotics by creating Polly, the first robot to navigate using vision and operate at animal-like speeds (1 meter/second).
1993	Rodney Brooks, Lynn Andrea Stein and Cynthia Breazeal started the widely-publicized MIT Cog project with numerous collaborators, in an attempt to build a humanoid robot child in just five years.
1993	ISX corporation wins "DARPA contractor of the year"[36] for the Dynamic Analysis and Replanning Tool (DART) which reportedly repaid the US government's entire investment in AI research since the 1950s.[37]
1994	With passengers on board, the twin robot cars VaMP and VITA-2 of Ernst Dickmanns and Daimler-Benz drive more than one thousand kilometers on a Paris three-lane highway in standard heavy traffic at speeds up to 130 km/h. They demonstrate autonomous driving in free lanes, convoy driving, and lane changes left and right with autonomous passing of other cars.
1995	Semi-autonomous ALVINN steered a car coast-to-coast under computer control for all but about 50 of the 2850 miles. Throttle and brakes, however, were controlled by a human driver.
1995	In the same year, one of Ernst Dickmanns' robot cars (with robot-controlled throttle and brakes) drove more than 1000 miles fromMunich to Copenhagen and back, in traffic, at up to 120 mph, occasionally executing maneuvers to pass other cars (only in a few critical situations a safety driver took over). Active vision was used to deal with rapidly changing street scenes.
1997	The Deep Blue chess machine (IBM) defeats the (then) world chess champion, Garry Kasparov.
1997	First official RoboCup football (soccer) match featuring table-top matches with 40 teams of interacting robots and over 5000 spectators.
1998	Tiger Electronics' Furby is released, and becomes the first successful attempt at producing a type of A.I to reach a domestic environment.
1998	Tim Berners-Lee published his Semantic Web Road map paper.[38]
1999	Sony introduces an improved domestic robot similar to a Furby, the AIBO becomes one of the first artificially intelligent "pets" that is also autonomous.
Late 1990s	Web crawlers and other AI-based information extraction programs become essential in widespread use of the World Wide Web.
Late 1990s	Demonstration of an Intelligent room and Emotional Agents at MIT's AI Lab.
Late	Initiation of work on the Oxygen architecture, which connects mobile and stationary

1990s	computers in an adaptive network.

2000s

Date	Development
2000	Interactive robopets ("smart toys") become commercially available, realizing the vision of the 18th century novelty toy makers.
2000	Cynthia Breazeal at MIT publishes her dissertation on Sociable machines, describing Kismet (robot), with a face that expressesemotions.
2000	The Nomad robot explores remote regions of Antarctica looking for meteorite samples.
2004	OWL Web Ontology Language W3C Recommendation (10 February 2004).
2004	DARPA introduces the DARPA Grand Challenge requiring competitors to produce autonomous vehicles for prize money.
2005	Honda's ASIMO robot, an artificially intelligent humanoid robot, is able to walk as fast as a human, delivering trays to customers in restaurant settings.
2005	Recommendation technology based on tracking web activity or media usage brings AI to marketing. See TiVo Suggestions.
2005	Blue Brain is born, a project to simulate the brain at molecular detail.[1]
2006	The Dartmouth Artificial Intelligence Conference: The Next 50 Years (AI@50) AI@50 (14–16 July 2006)
2007	Philosophical Transactions of the Royal Society, B – Biology, one of the world's oldest scientific journals, puts out a special issue on using AI to understand biological intelligence, titled *Models of Natural Action Selection*[39]
2007	Checkers is solved by a team of researchers at the University of Alberta.

2010s

Date	Development
2011	AI received much public attention in February, 2011, with the Jeopardy! exhibition match during which IBM's Watson soundly defeated the two greatest Jeopardy! champions, Brad Rutter and Ken Jennings.

2.Μηχανική μάθηση

Μη εποπτευόμενη μαθηση

Μηχανικη μαθηση ειναι η μελετη αλγοριθμων που βελτιωνοντε αυτοματα μεσω αποκτησης εμπειριας.

Μη εποπτευόμενη μαθηση ειναι η δυνατοτητα εντοπισμου προτυπων σε μια ροη εισοδου.
Η μη εποπτευόμενη μαθηση ειναι πολυ κοντα με το πρβλημα προβλεψης πυκνοτητας στη στατιστικη αναλυση, παρολοαυτα η μη εποπτευόμενη μαθηση περιλαμβανει και πολλεσ αλλες τεχνικες που προσπαθουν να συνοψισουν και να εξηγισουν χαρακτιριστικα των δεδομενων.
Πολλες μεθοδοι στην μη επιτηρουμενη μαθηση βασιζονται σε τεχνικες εξαγωγης δεδομενων(data mining) που χρησιμοποιουνται για προεπεξεργασια των δεδομενων.

Τεχνηκες μη εποπτευόμενη μαθησης περιλαμβανουν τις εξης τεχνικες:
.clustering (K-means, mixture modelsm hierarchical clustering)
.blind signal seperation using feature extraction techniques for dimensionality reduction(Principal component analysis, independent component analysis,Non-negative matrix factorization,Singular value decomposition)

Εποπτευόμενη μάθηση

Επιβλεπόμενη μάθηση είναι το έργο μηχανικής μάθησης της συναγωγής μιας συναρτησης από τα επονομαζόμενα δεδομενα εκπαίδευσης. Τα δεδομένα εκπαίδευσης αποτελούνται από ένα σύνολο παραδειγμάτων εκπαίδευσης. Σε εποπτευόμενη εκμάθηση, κάθε παράδειγμα είναι ένα ζεύγος που αποτελείται από ένα αντικείμενο εισόδου (τυπικά ένας πίνακας) και μία επιθυμητή τιμή εξόδου (ονομάζεται επίσης το εποπτικό σήμα). Ένας αλγόριθμος εποπτευόμενης εκμάθησης αναλύει τα δεδομένα εκπαίδευσης και παράγει μια συναρτηση, η οποία μπορεί να χρησιμοποιηθεί για τη χαρτογράφηση νέων παραδείγματων. Ένα βέλτιστο σενάριο θα επιτρέψει στον αλγόριθμο να καθορίσει σωστά τις ετικέτες κατηγορίας για αθέατες περιπτώσεις. Αυτό απαιτεί απο τον αλγόριθμο να γενικεύει από τα δεδομένα εκπαίδευσης σε πρωτόγνωρα καταστάσεις με ένα "λογικό" τρόπο

Επιβλεπόμενη μάθηση περιλαμβάνει και την ταξινόμηση και την αριθμητική παλινδρόμησης.

Ταξινόμηση

Ταξινόμηση χρησιμοποιείται για να καθορίσει ποια κατηγορία ανήκει ενα στοιχειο αφού δει μια σειρά από παραδείγματα στοιχείων από διάφορες κατηγορίες.
Στη μηχανική μάθηση και στη στατιστικη, ταξινόμηση είναι το πρόβλημα του εντοπισμού του σε πιο από ένα σύνολο κατηγοριών (υπο-πληθυσμοί) μια νέα παρατήρηση ανήκει, με βάση ένα σύνολο δεδομένων εκπαίδευσης που περιέχει τις παρατηρήσεις (ή περιπτώσεις) των οποίων η κατηγορία μελών είναι γνωστή. Οι επιμέρους παρατηρήσεις αναλύονται σε ένα σύνολο ιδιοτήτων ποσοτικώς, είναι γνωστή ως διάφορες επεξηγηματικές μεταβλητές, χαρακτηριστικά, κ.λπ.

Αυτές οι ιδιότητες μπορούν ποικιλοτρόπως να κατηγοροποιηθούν (π.χ. "Α", "Β", "ΑΒ" ή "Ο", για τον τύπο του αίματος), τακτικό (π.χ. "μεγάλη", "μέτρια" ή "μικρή"), ακέραιος-αξίας (π.χ. τον αριθμό των εμφανίσεων μιας λέξης που ειναι μέρος σε ένα μήνυμα ηλεκτρονικού ταχυδρομείου) ή πραγματικών τιμών (π.χ. μέτρηση της πίεσης του αίματος). Μερικοί αλγόριθμοι λειτουργούν μόνο από την άποψη των διακριτών στοιχείων και απαιτούν ότι οι πραγματικές τιμές ή ακέραιος-πολύτιμων δεδομένων να κατηγοριοποιηθει σε ομάδες (π.χ. λιγότερο από 5, μεταξύ 5 και 10, ή μεγαλύτερη από 10).

Ένα παράδειγμα θα ήταν ανάθεση ενός δεδομένου email σε «spam» ή «μη-spam" τάξεις ή αναθέτοντας μια διάγνωση σε έναν δεδομένο ασθενή, όπως περιγράφεται από παρατηρούμενων χαρακτηριστικών του ασθενούς (φύλο, την αρτηριακή πίεση, την παρουσία ή απουσία ορισμένων συμπτωμάτων, κλπ.).

Ένας αλγόριθμος που υλοποιεί την ταξινόμηση, ειδικά σε μια συγκεκριμένη εφαρμογή, είναι γνωστός ως ένας ταξινομητής. Ο όρος «ταξινομητής» μερικές φορές αναφέρεται επίσης στην μαθηματική συνάρτηση, που υλοποιείται από έναν αλγόριθμο ταξινόμησης, που κατηγοριοποιει τα δεδομένα εισόδου

Αριθμητική παλινδρόμησης

Παλινδρόμηση είναι η προσπάθεια να παραχθεί μια λειτουργία που περιγράφει τη σχέση μεταξύ εισροών και εκροών και προβλέπει πως οι έξοδοι θα πρέπει να αλλάξουν, όπως αλλαζουν οι εισροές.

Ανάλυση παλινδρόμησης είναι μία στατιστική τεχνική για την εκτίμηση των σχέσεων μεταξύ των μεταβλητών. Περιλαμβάνει πολλές τεχνικές για την μοντελοποίηση και ανάλυση πολλών μεταβλητών, όταν η εστίαση είναι στην σχέση μεταξύ μιας εξαρτημένης μεταβλητής και μίας οι περισσότερων ανεξάρτητων μεταβλητων. Πιο συγκεκριμένα, η ανάλυση παλινδρόμησης βοηθά να κατανοήσουμε πώς η τυπική τιμή της εξαρτημένης μεταβλητής αλλάζει όταν οποιαδήποτε από τις ανεξάρτητες μεταβλητές μεταβάλλεται, ενώ οι άλλες ανεξάρτητες μεταβλητές παραμένουν. Πιο συχνά, η ανάλυση παλινδρόμησης υπολογίζει την υπό όρους προσδοκία της εξαρτημένης μεταβλητής που οι ανεξάρτητες μεταβλητές - δηλαδή, η μεση αξία της εξαρτημένης μεταβλητής, όταν οι ανεξάρτητες μεταβλητές ειναι σταθερες. . Σε όλες τις περιπτώσεις, ο στόχος εκτίμησης είναι συνάρτηση των ανεξάρτητων μεταβλητών που ονομάζετε συναρτηση διαβαθμισης. Στην ανάλυση παλινδρόμησης, είναι επίσης ενδιαφέρον ο χαρακτηρισμός της διακύμανσης της εξαρτημένης μεταβλητής γύρω από την συνάρτηση παλινδρόμησης, η οποία μπορεί να περιγραφεί από μια κατανομή πιθανότητας.

Ενισχυτική μάθηση

Σε ενισχυτικής μάθησης , ο παράγοντας ανταμείβεται για καλές απαντήσεις και να τιμωρηθεί για τα λαθος.
Αυτ'ο μπορεί να αναλυθεί από την άποψη της θεωρίας των αποφάσεων, χρησιμοποιώντας έννοιες όπως η χρησιμότητα.
Εμπνευσμένο από την ψυχολογία συμπεριφορικής, ενισχυτική μάθηση είναι μια περιοχή της μηχανικής μάθησης στην επιστήμη των υπολογιστών, ασχολείται με το πώς ένας πράκτορας θα πρέπει να αναλάβει δράση σε ένα περιβάλλον έτσι ώστε να μεγιστοποιηθεί κάποια έννοια σωρευτική ανταμοιβή. Το πρόβλημα, λόγω της γενικότητας της, έχει μελετηθεί σε πολλούς άλλους επιστημονικούς κλάδους, όπως η θεωρία των παιγνίων, θεωρία ελέγχου, επιχειρησιακή έρευνα, θεωρία της πληροφορίας, προσομοίωση με βάση τη βελτιστοποίηση, στατιστικά στοιχεία και οι γενετικοί αλγόριθμοι. Στην έρευνα πράξεων και τη λογοτεχνία ελέγχου το πεδίο όπου μελετώνται μέθοδοι ενισχυτικής μάθησης ονομάζεται προσέγγιστικος δυναμικός προγραμματισμός. Το πρόβλημα έχει μελετηθεί στη θεωρία του βέλτιστου ελέγχου, αν και οι περισσότερες μελέτες υπάρχουν ασχολούνται με την ύπαρξη των βέλτιστων λύσεων και τον χαρακτηρισμό τους, και όχι με τις πτυχές της μάθησης . Στην οικονομία και τη θεωρία παιγνίων, ενισχυτική μάθηση μπορεί να χρησιμοποιηθεί για να εξηγήσει πώς ισορροπία μπορεί να προκύψη στο πλαίσιο οριοθετούμενου ορθολογισμού.

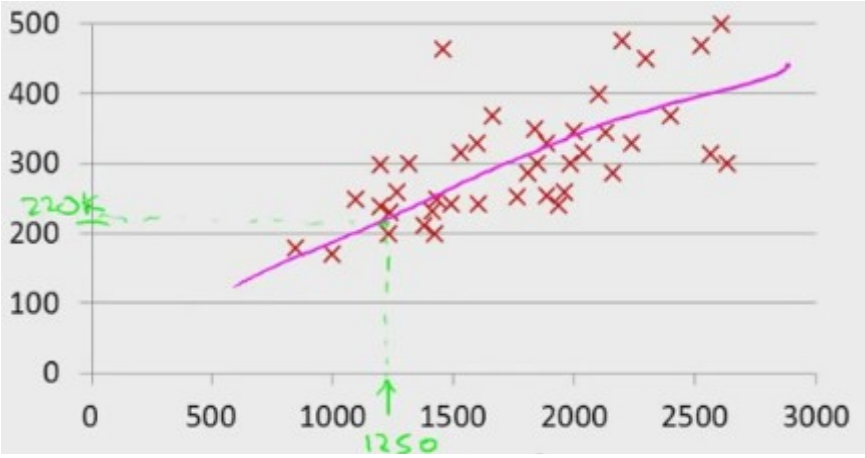
Η μαθηματική ανάλυση των αλγορίθμων μηχανικής μάθησης και η επιδόσεις τους είναι ένας κλάδος της θεωρητικής επιστήμης των υπολογιστών είναι γνωστό ascomputational learning theory

3.Νευρωνικά δίκτυα

. Η ανάγκη για τα νευρωνικά δίκτυα(γραμμικη και μη γραμμικη υπόθεση)

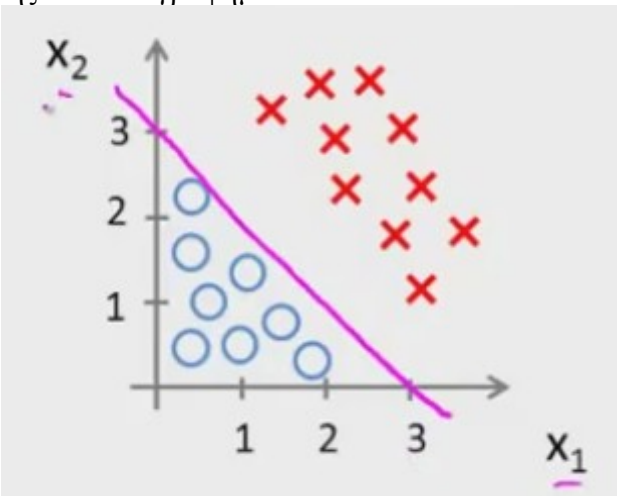
Η γραμμική υπόθεση.

ένα παράδειγμα ενος αλγόριθμου εποπτευόμενης μάθησης
είναι η ακόλουθη

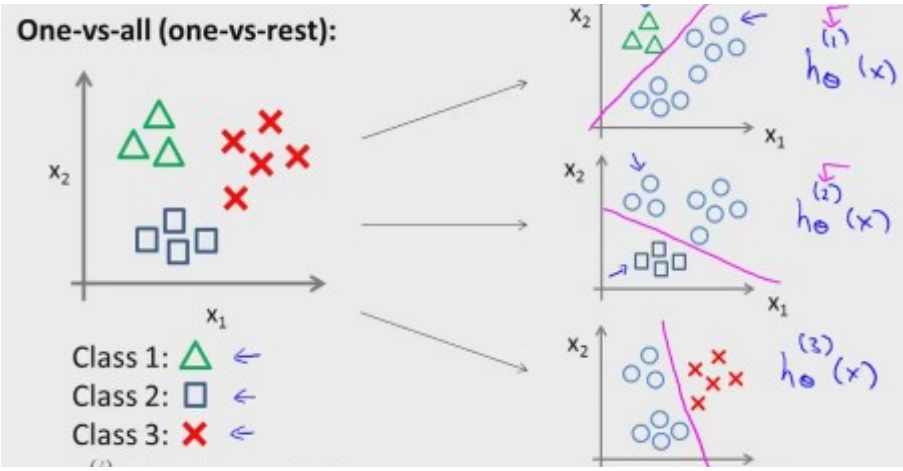


εδώ ο αλγόριθμος προσπαθεί να καθορίσει την τιμή ενός σπιτιού (άξονας y) από το μέγεθός του (x-άξονα)
και η γραμμική υπόθεση είναι η ροζ γραμμή.

σε προβλήματα ταξινόμησης Αυτό το γράφημα είναι σαν αυτό



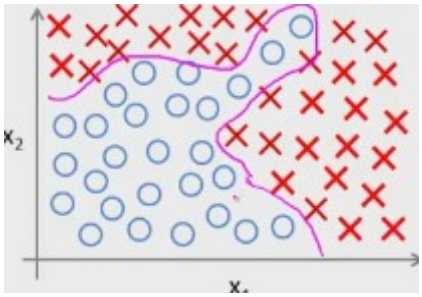
Σε πολλαπλά προβλήματα ταξινόμησης (one-vs-all)
το γράφημα είναι σαν αυτό



Η ανάγκη για έναν αλγόριθμο νευρωνικού δίκτυου είναι η μη γραμμική υπόθεση.
Τόσο η γραμμικη οσο και η λογικη παλινδρομιση μπορεί να λειτουργήσει μόνο με γραμμική υπόθεση.

Ας εξετάσουμε τα ακόλουθα:

Ας πούμε ότι έχουμε αυτό το σύνολο δεδομένων



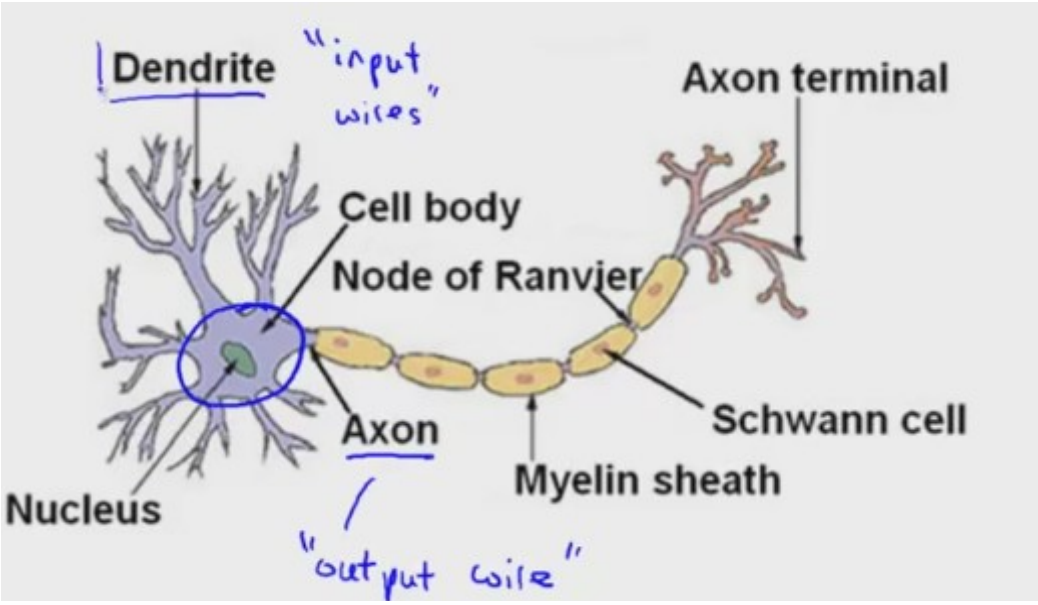
Για να ταιριαξει στο σύνολο δεδομένων, θα πρέπει να προσθέσει πολλά επιπλέον χαρακτηριστικά πολυωνύμου με αποτέλεσμα η σιγμοειδής συνάρτηση να είναι κάπως έτσι

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

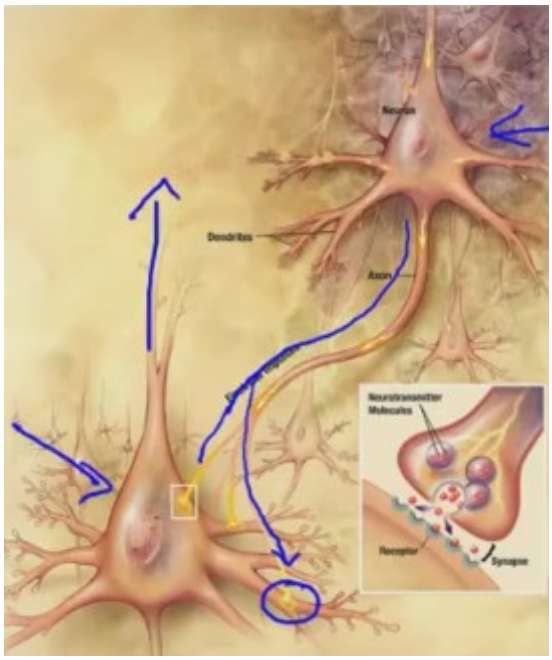
Αυτή η μέθοδος λειτουργεί καλά, όταν έχουμε μόνο δύο παραμέτρους όπως το παραπάνω (x_1, x_2)

Αλλά το πρόβλημα με περισσότερες παραμέτρους σε ένα πρόβλημα που θα πρέπει να προβλέψει εάν ένα σπίτι θα πωληθεί μέσα σε 6 μήνες
θα έχουν χαρακτηριστικά όπως:
το μέγεθος, κρεβατοκάμαρες, δάπεδα, ηλικία ..... και ούτω καθεξής
για να χωρέσει αυτά τα δεδομένα θα κάναμε και με περίπου 5.000.000 χαρακτηριστικά
θα μπορούσαμε να συμπεριλάβουμε λιγότερα χαρακτηριστικά, αλλά στη συνέχεια η υπόθεση δεν θα είναι αρκετά ακριβείς

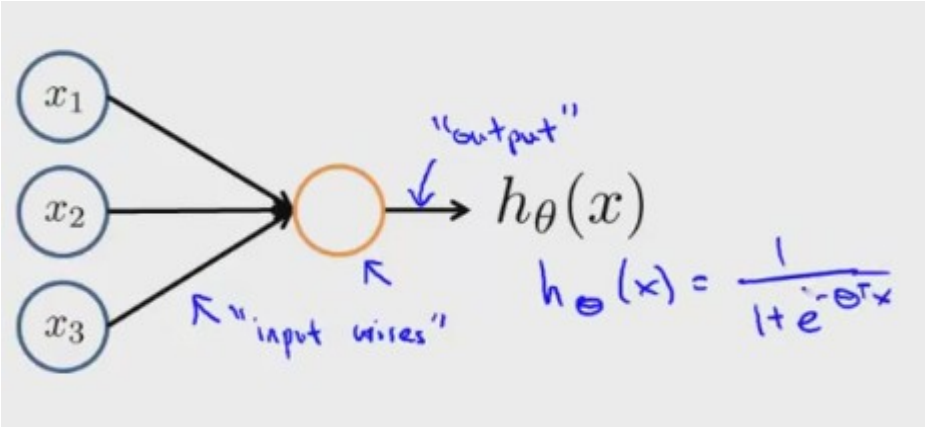
. Η αναπαράσταση ενός νευρωνικού δικτύου

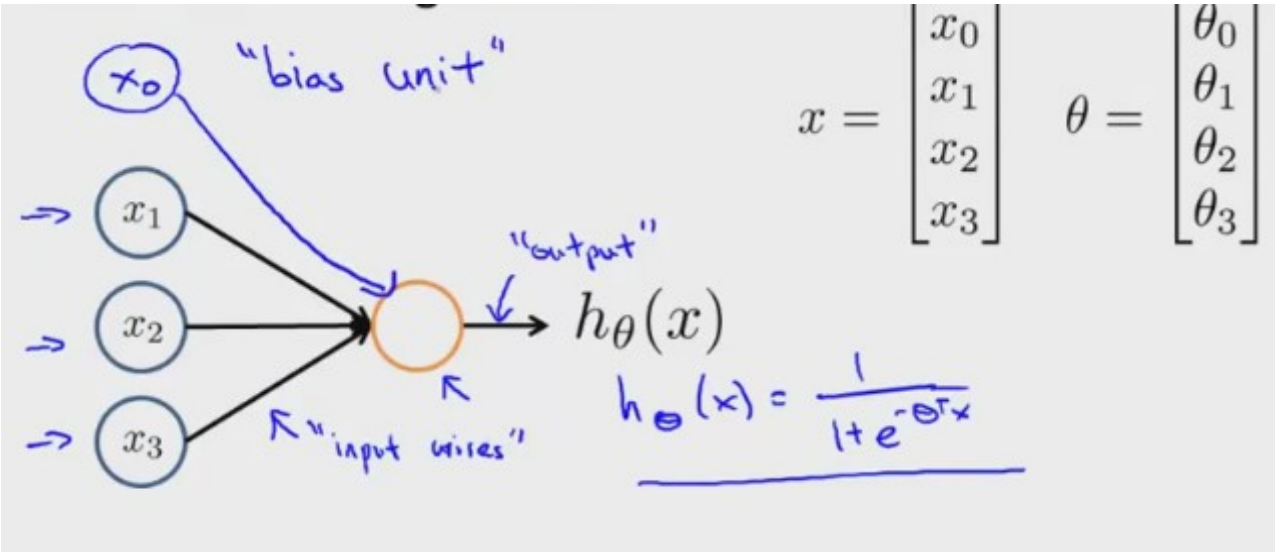


Με ένα απλοϊκό τρόπο ο νευρώνας δρα σαν υπολογιστική μονάδα που παίρνει εισόδους του από τα καλώδια εισόδους του κάνει υπολογισμούς και παράγει εξόδους που πηγαίνουν σε άλλους νευρώνες στον εγκέφαλο.



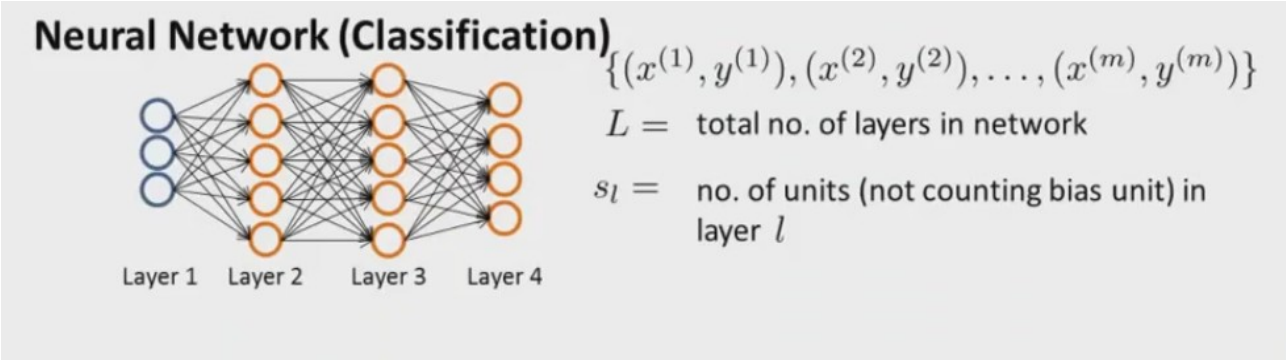
Η αναπαρασταση αυτου του μηχανισμου γινεται ως εξης.





3α. Η συνάρτηση κόστους στην Λογιστική Ανάλυση Παλινδρόμησης και στα νευρωνικά δύκτια

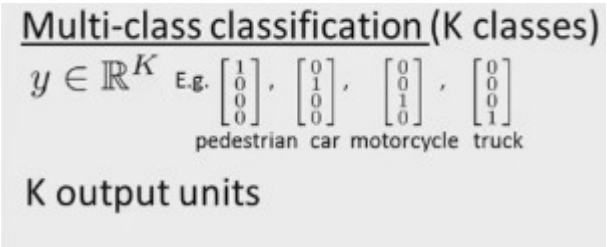
Τα νευρωνικά δίκτυα είναι ένα από τους πιο ισχυρους αλγόριθμους που έχουμε σήμερα.
Θα μιλήσουμε ενός αλγόριθμου μάθησης για την τοποθέτηση των παραμέτρους για ενα νευρωνικό δίκτυο που του δίνεται ένα σύνολο δεδομένων εκπαίδευσης
Θα ξεκινήσουμε μιλώντας για μια συνάρτηση κόστους για το ταίριασμα των παραμέτρων του δικτύου.



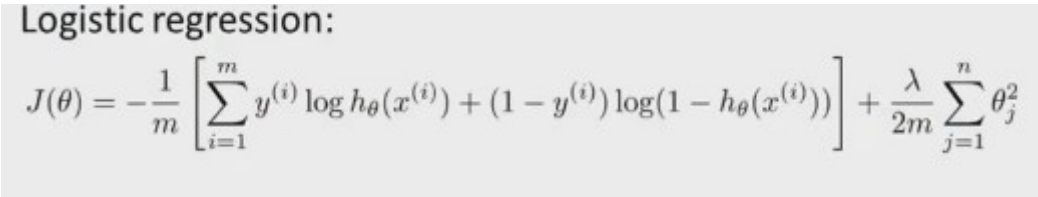
Ας υποθέσουμε ότι έχουμε ένα δίκτυο και ένα σύνολο δεδομένων εκπαίδευσης, όπως το παραπάνω.

Έχουμε L = 4

S1 = 3, S2 = 5, S4 = SL = 4



Συνάρτηση κόστους:



Η συνάρτηση κόστους που χρησιμοποιούμε για το νευρωνικό δίκτυο πρόκειται να είναι μια γενίκευση αυτής που χρησιμοποιούμε για την λογική παλινδρόμηση.

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Με

$$\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

είναι ο όρος γενίκευσης

Το νευρωνικό δίκτυο μας εξάγει πλέον φορείς σε Rk
Και η υπόθεσή μας είναι ενας K διαστάσεων πίνακας

Τώρα θα μιλήσουμε για έναν αλγόριθμο για να προσπαθεια να ελαχιστοποιήθει η συνάρτηση του κόστους, ιδίως του αλγορίθμου ανάστροφης διάδοσης

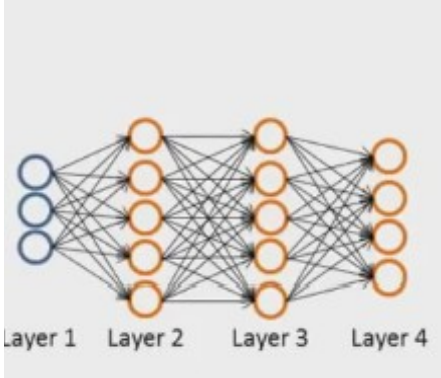
$$\min_\Theta J(\Theta)$$

προκειμένου να χρησιμοποιήθει η κλίση καθόδου ή ενας από τις προηγμένους αλγόριθμους βελτιστοποίσης πρέπει να γράψουμε κώδικα που παίρνει ως είσοδο θήτα και υπολογίζει Jθήτα και τους όρους μερικής παραγώγισης .

- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$

. Υπολογισμός τον μερικών παραγώγων

Ας μιλήσουμε σχετικά με τον τρόπο για τον υπολογισμό των μερικών παραγώγων



Ας υποθέσουμε ότι έχουμε μόνο ένα παράδειγμα εκπαίδευσης (x, y)

Ο υπολογισμός θα πρέπει να κάνουμε είναι να εφαρμόσθει κατ 'αρχάς προς τα εμπρός διάδοση προκειμένου να υπολογιστεί τι εξάγει στην πραγματικότητα η υπόθεση.

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$

Ανωτέρω είναι το στρώμα εισόδου 1
A1 πρώτο κρυμμένο στρώμα μας το στρώμα 2
α3 το τρίτο στρώμα
A4 το τέταρτο στρώμα μας, που είναι και στρώμα εξόδου μας

Έπειτα ώστε να υπολογιστούν οι παράγωγοι πρόκειται να χρησιμοποιήσουμε έναν αλγόριθμο που ονομάζεται αλγόριθμος ανάστροφης διάδοσης

Ο αλγόριθμος ανάστροφης διάδοσης υπολογίζει το

$\delta_j^{(l)}$

Που ειναι το σφάλμα του J κόμβου στο στρώμα L

για κάθε μονάδα εξόδου (στρώμα L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$\delta_j^{(4)}$

Είναι το σφάλμα της j μονάδας στο επόμενο στρώμα που είναι απλά η ενεργοποίηση της μονάδας (που είναι η υπόθεσή μας)

$a_j^{(4)}$

μείον την πραγματική αξία στο συγκεκριμένο παράδειγμα

y_j

Τώρα για το στρώμα 3 και το στρώμα 2, το σφάλμα θα ήταν

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

ο όρος

$$g'(z^{(3)})$$
$$g'(z^{(2)})$$

είναι η παράγωγος της συνάρτησης ενεργόποιησης και εξαρτάτε απο τις τιμές εισόδου που δίνονται απότο Z

$$\underline{a}^{(3)} .* (1 - \underline{a}^{(3)})$$
$$a^{(2)} .* (1 - a^{(2)})$$

Τώρα Ας υποθέσουμε ότι έχουμε ένα σύνολο εκπαίδευσης m παραδείγματων

$$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Πρώτον, θα πρέπει να ρυθμίστει

$$\triangle_{ij}^{(l)} = 0$$

για όλα τα l,i,j

Που θα χρησιμοποιηθούν τελικά ως συσσωρευτές για τον υπολογισμό των παραγώγων του κόστους

Θα πρέπει να εκτελέσει η εμπρός διάδοση για τον υπολογισμό των ενεργοποιήσεων για όλα τα στρώματα

Τότε θα χρησιμοποιήσουμε την έξοδο y να υπολογιστεί το σφάλμα της τελευταίας στρώσης(έξοδος) και να εκτελέσει αλγόριθμος ανάστροφης διάδοσης για να υπολογιστεί το σφάλμα για τα υπόλοιπα στρώματα

Στη συνέχεια πρόκειται να χρησιμοποιθει ο όρος

$$\triangle_{ij}^{(l)}$$

Για συσσωρεύουν οι μερικοι παραγώγοι

$$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

ή αλλιως

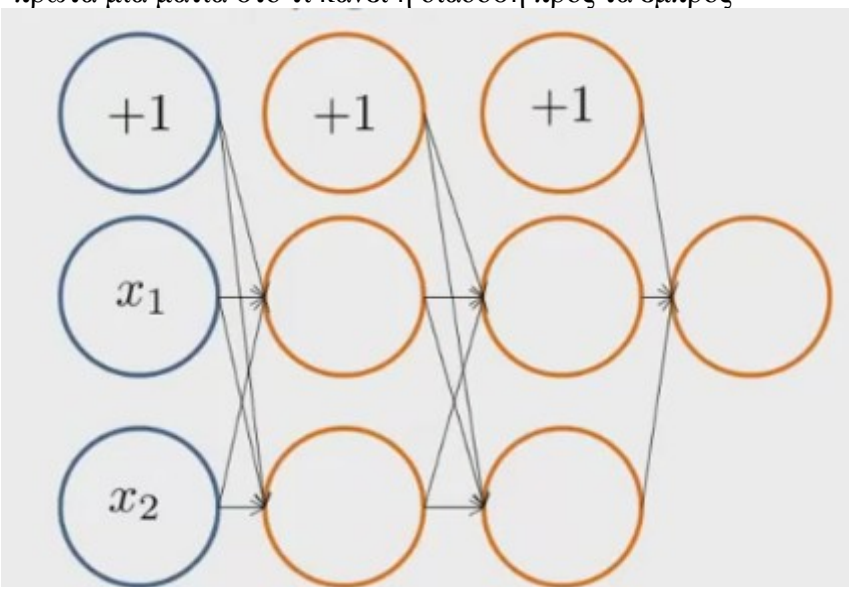
$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$
$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} \qquad \text{if } j = 0$$

και αφού υπολογιστούν αυτούς τους όρους που είναι ακριβώς η μερική παράγωγος της συνάρτησης κόστους σε σχέση με της παραμέτρους

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

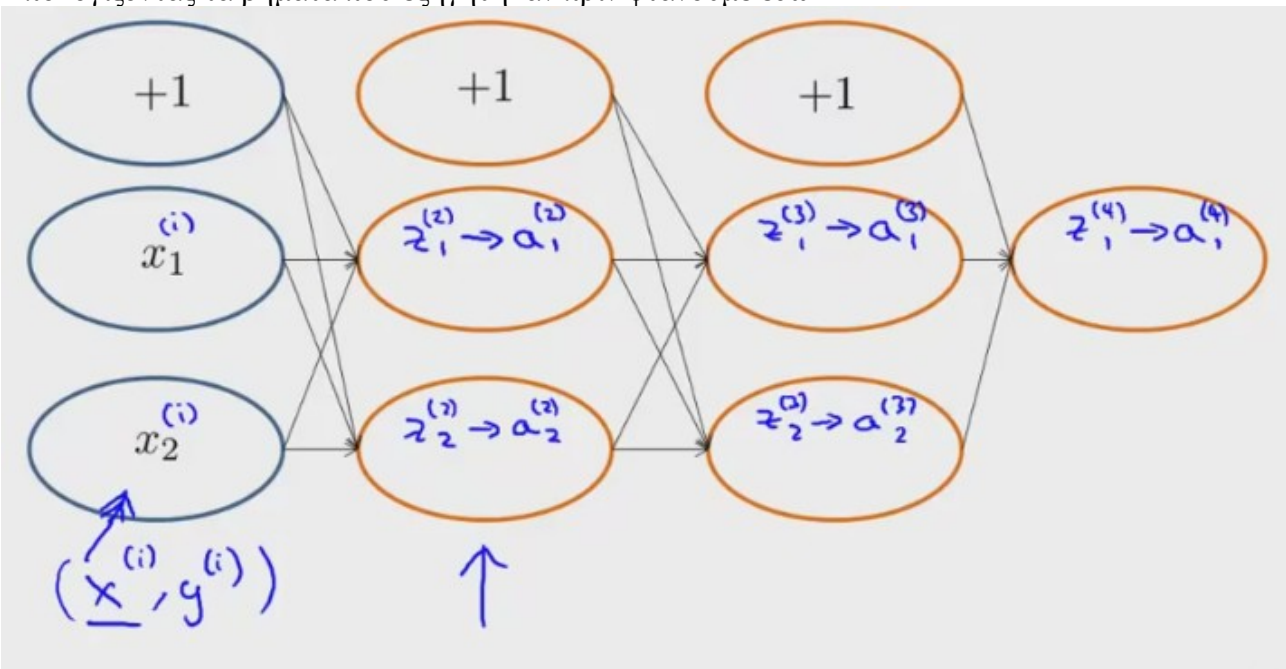
και αυτά μπορούν να χρησιμοποιηθούν είτε σε κάθοδο κλίση ή σε μία ένα απο τους προηγμένους αλγορίθμους βελτιστοποιήσης

Γενικώς ο αλγόριθμος ανάστροφης διάδοσης είναι περισσότερο μαθηματικά πολύπλοκος αλγόριθμος σε σύγκριση με την γραμμική ή την λογιστική παλινδρόμηση και η μαθηματική του απόδειξη δεν αποτελεί το αντικείμενο αυτης της πτυχιακής,αλλά για να κατανοήσουμε καλύτερα τι κάνει ας ρίξουμε πρώτα μια ματιά στο τι κάνει η διάδοση προς τα εμπρός

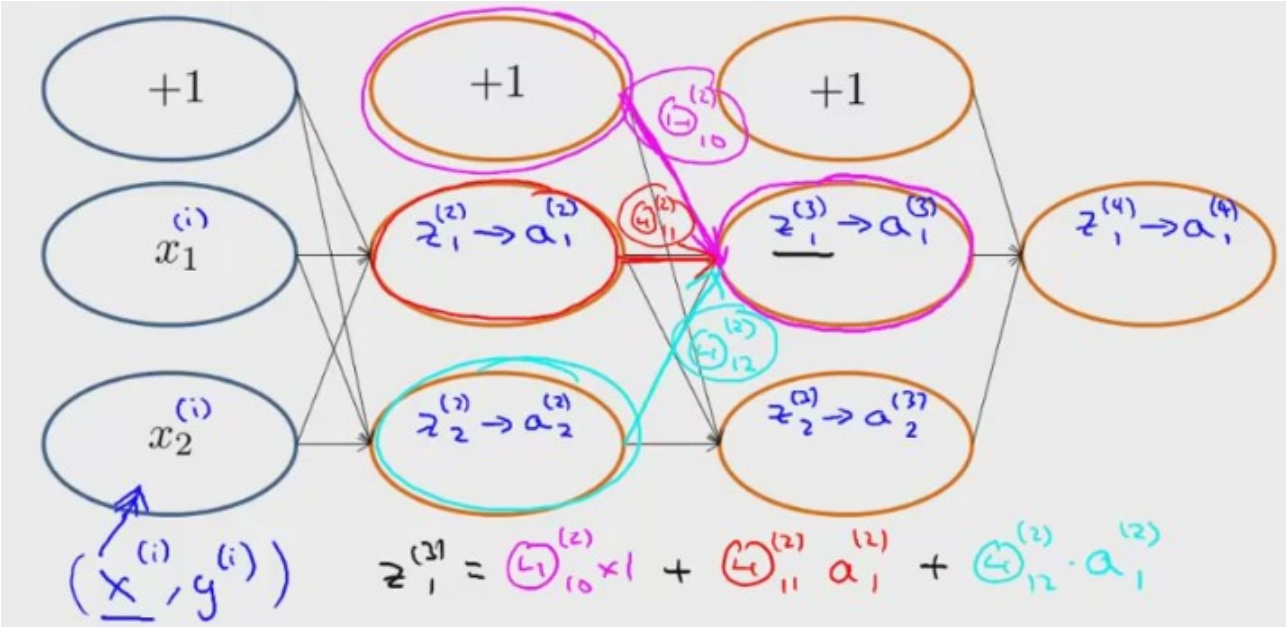


Εδώ παρουσιάζεται ένα νευρωνικό netwrok με δύο μονάδες εισόδου (χωρίς να μετράει η μονάδα πόλωσης)
με 2 κρυμμένα στρώματα με 2 μονάδες σε κάθε στρώση
και ένα στρώμα εξόδου.

Υπολογίζοντας τα βήματα που εξηγήθηκαν πριν φτάνουμε εδώ



Ασ κοιτάξουμε τι πραγματικά κανει αυτος ο υπολογισμός εστιάζοντας στην πρώτη μονάδα του 2ου κρυφού στρώματος
Υπολογίζει την ενεργοποίηση για κάθε μονάδα σε σχέση με την εξόδο της προηγούμενης(τα βάρη μας) και έτσι διαδίδετε εμπρός



Αποδεικνύεται ότι αυτό που κάνει ο αλγόριθμος ανάστροφης διάδοσης είναι μια διαδικασία πολύ παρόμοια με αυτό, αλλά αντι ο υπολογισμός να ρέει από τα αριστερά προς τα δεξιά του δικτύου ,ρέει από τα δεξιά προς τα αριστερά χρησιμοποιώντας ένα πολύ παρόμοιο υπολογισμό ως ανωτέρω.
Για την περαιτέρω κατανόηση του 'αλγόριθμου ανάστροφης διάδοσης ας δούμε την συνάρτηση κόστους

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1-y^{(i)}) \log(1-(h_\Theta(x^{(i)})))\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Αυτό είναι η λειτουργία του κόστους για μία μόνο μονάδα εξόδου
που κανει προς τα εμπρός και προς τα πίσω διάδοση ένα παράδειγμα τη φόρα
X (I), Y (í) και αγνοώντας τη γενίκευση λ = 0

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log h_\Theta(x^{(i)})$$

αυτή η εξίσωση είναι πολύ παρόμοια με το τετραγωνικό σφάλμα

$$\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2)$$

Τώρα ας δούμε τι κάνει ο αλγόριθμος ανάστροφης διάδοσης

Θυμηθείτε οτι ο αλγόριθμος ανάστροφης διάδοσης είναι ο υπολογισμός των σφαλμάτων

$$\delta_j^{(l)}$$

της τιμής ενεργοποίησης της μονάδας

$$a_j^{(l)}$$

επίσημα

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost(i)}$$

για j> = 0
όπου

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

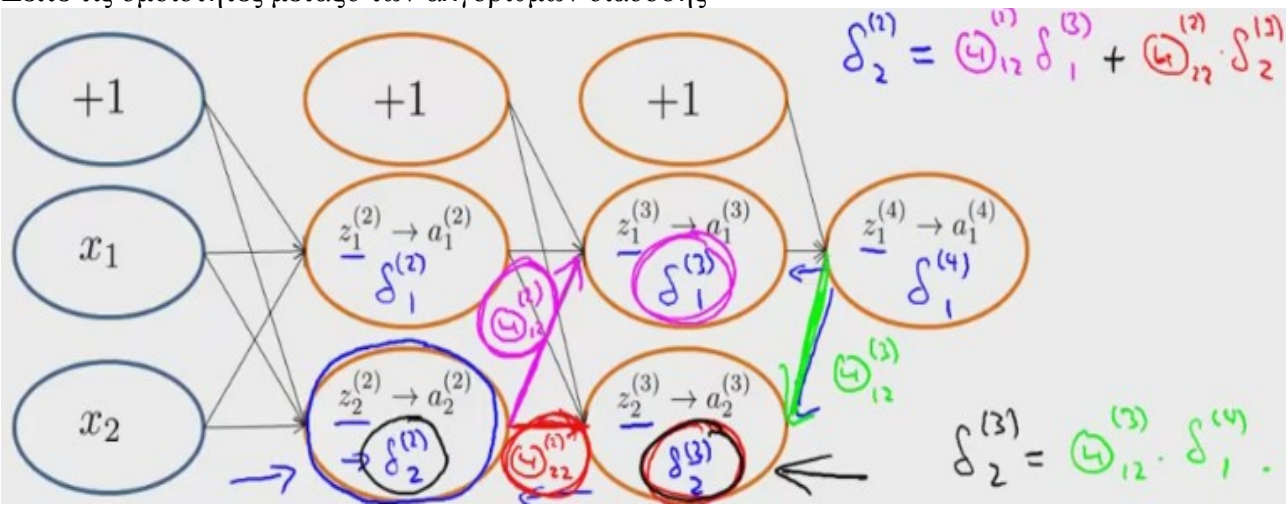
Αν θα μπορούσαμε να πάμε στο εσωτερικό του νευρικού netwrok και να αλλάξετε τις Z τιμές που θα επηρεάζαν τις τιμές πουτο νευρωνικό δίκτυο εξάγει αυτό θα έιχε ως αποτέλεσμα την αλλαγή στη λειτουργία του κόστους.

Έτσι είναι μέτρο του πόσο πολύ θα θέλαμε να μετρήσουμε τα νευρωνικά δίκτυα βάρη προκειμένου να επηρεάσουμε τις ενδιάμεσες αξίες του υπολογισμού, ώστε να επηρεάσουμε το τελικό αποτέλεσμα των νευρωνικών δικτύων, προκειμένου να επηρεάσουμε το ολικό κόστος.

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Μπορείτε να δείτε ότι όταν υπολογίζουμε
Στην πραγματικότητα βλέπουμε πόσο μακριά ήταν η πρόβλεψη από την πραγματική τιμή που θα θέλαμε να έχουμε.

Δείτε τις ομοιότητες μεταξυ των αλγόριθμων διάδοσης

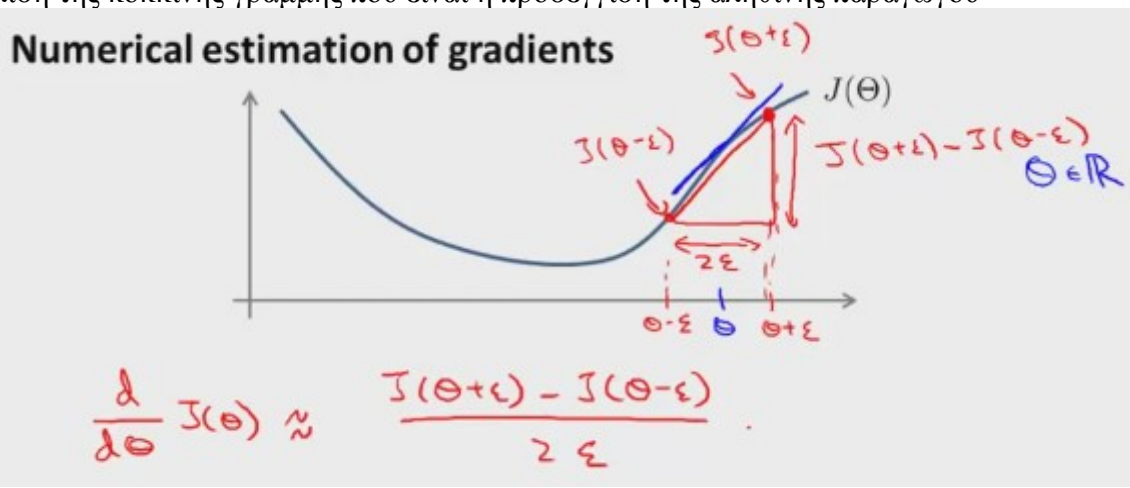


. Άριθμητικός έλεγχος κλήσης και τυχαίο ξεκίνημα συναπτικών βαρών

Κατά την εφαρμογή της ανάστροφης διάδοσης ή παρόμοιο αλγόριθμο κλίσης καθόδου επειδή οι αλγόριθμοι έχουν πολλές λεπτομέρειες και μπορεί να είναι δύσκολο να εφαρμόσουν Είναι καλό να εκτελείτε αριθμητικός ελέγχος της κλίσης για την εξάλειψη τυχόν σφαλμάτων και να βεβαιώνεται ότι ο κώδικας λειτουργεί

Ας Εξετάσουμε το ακόλουθο παράδειγμα

Έχοντας την J συνάρτηση κόστους (θ) ας πούμε ότι θέλουμε να υπολογίσουμε την παράγωγο της θ θα υπολογίσουμε θ + ε και θ-ε συνδέουμε τα δύο σημεία με μια ευθεία γραμμή και χρησιμοποιούμε την κλίση της κόκκινης γραμμής που ειναι η προσέγγιση της αληθινής παραγώγου



Το ε πρέπει να έχει αρκετά μικρή τιμή, όπως ε = 10 ^ (-4)

Έχοντας των πίνακα παραμέτρου θ μπορούμε να υπολογίσουμε όλες τις μερικές παραγώγους

$$\frac{\partial}{\partial\theta_1}J(\theta) \approx \frac{J(\theta_1+\epsilon,\theta_2,\theta_3,\ldots,\theta_n)-J(\theta_1-\epsilon,\theta_2,\theta_3,\ldots,\theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial\theta_2}J(\theta) \approx \frac{J(\theta_1,\theta_2+\epsilon,\theta_3,\ldots,\theta_n)-J(\theta_1,\theta_2-\epsilon,\theta_3,\ldots,\theta_n)}{2\epsilon}$$

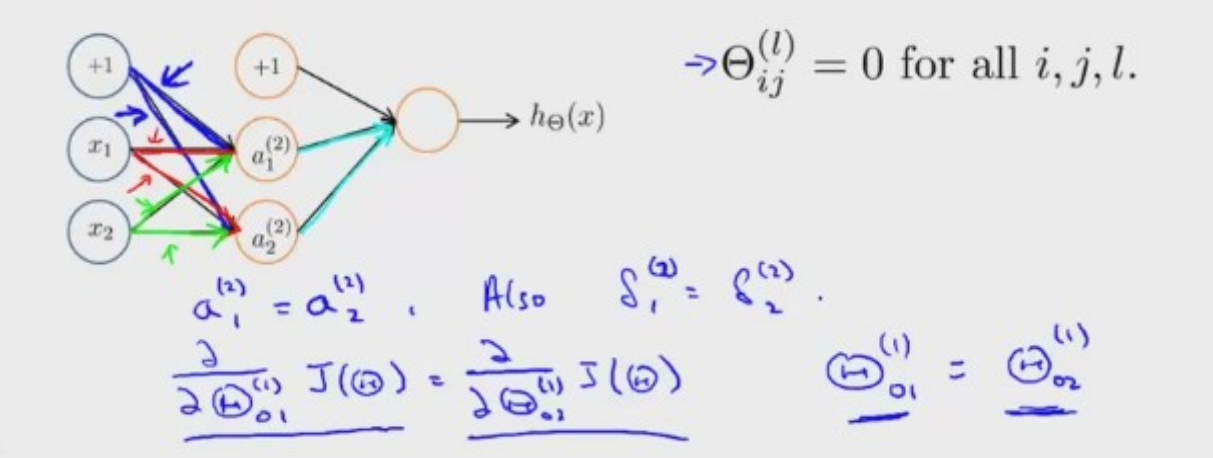
$$\vdots$$

$$\frac{\partial}{\partial\theta_n}J(\theta) \approx \frac{J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n+\epsilon)-J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n-\epsilon)}{2\epsilon}$$

Στα προηγούμενα μέρη υπάρχουν σχεδόν όλα όσα χρειάζονται για να εφαρμόσθει ένα νευρωνικό δίκτυο.

Ένα ακόμα πράγμα που πρέπει να εξεταστεί ειναι η αρχικοποίηση των συνοπτίκων βαρών.

Αν έχουμε αρχικοποίησει όλες τις παραμέτρους στο μηδέν εκτελόντας διαδόχη προς τα εμπρός θα έχουμε τα εξής



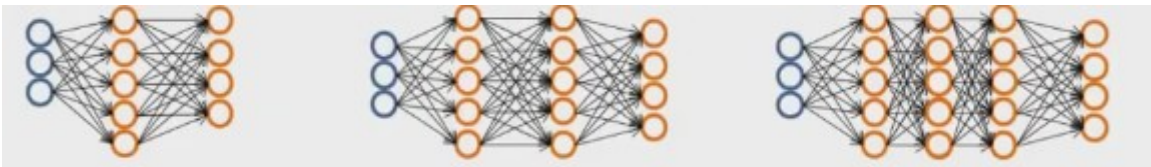
Μετά από κάθε ανανέωση, οι παραμέτροι που αντιστοιχούν στις εισόδους που πηγαίνου σε καθέμια από της δύο κρυμμένες μονάδες είναι πανομοιότυπες!

Με αποτέλεσμα σε όλες της κρυμμένες μονάδες θα υπολογίσει ακριβώς η ίδια συνάρτηση στην εισόδο που σημαίνει ότι η τελική μονάδα παλινδρόμησης θα πάρει την ίδια είσοδο από όλες τις μονάδες

Έτσι, με τα παραπάνω συμπαιρένουμε οτι θα πρέπει να εκτελέστει μια τυχαία αρχικοποίηση και να επιτευχθεί σπάσιμο της συμμετρίας.

3β. Φτιαχνοντας ένα νευρωνικό δίκτυο

Κατά την εκπαίδευση ένος νευρωνικού δίκτυου το πρώτο πράγμα που πρέπει να γιίνεται είναι να βρίσκεται μια αρχιτεκτονική για το νευρωνικο δικτύο



Μπορεί να δει κανείς ότι για τον ίδιο αριθμό εισόδων και εξόδων υπάρχουν διαφορετικές αρχιτεκτονικές που μπορούν να επιλεχθούν

Πλήθος εισόδων : Διάσταση των χαρακτηριστικών
Πλήθος εξόδων : Αριθμός κατηγορίων
αυτά τα δύο είναι αρκετά απλα και εξαρτόνται από το πραγματικό πρόβλημα

Όσον αφορά τον αριθμό των κρυμμένων στρώματων μια λογική επιλόγη είναι 1 κρυμμένο στρώμα, αλλά εάν επιλεχθούν περισσότερα από ένα όλα πρέπει να έχουν τον ίδιο αριθμό μονάδων.

Όσον αφορά τον αριθμό των μονάδων συνήθως οσο περισσότερες τόσο το καλύτερο.

Εκπαίδευση ένος νευρωνικού δικτύου

1. Τυχαία αρχικοποιση συναπτικων βάρων

2. Υλοποιηση αλγοριθμου διαδοχης προς εμπρός για να πάρθει

$$h_{(}x^{(i)}\ ))$$

για καθε

$$x^{(i)}$$

3.Υλοποιηση του κώδικα για τον υπολογισμό του κόστους functiong

$$J(\Theta)$$

4. Υλοποιηση αλγοριθμου διάδοσης προς τα πίσω για τον υπολογισμό των μερικών παραγώγων

$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$$

Συνήθως αυτό γίνεται με ένα βρόχο επαναληψεων(for loop) πάνω από τα παραδείγματα εκπαίδευσης (επίσης υπάρχουν μερικές πολύ προχωρημένες τεχνικές διανυσματοποίησης)

5. Χρηση έλεγχου κλίση για να συγκρίθει η

$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$$

υπολογισμένη με τον αλγόριθμο ανάστροφης διάδοσης εναντια του υπολογισμου χρησιμοποιώντας αριθμητική διαβάθμιση

$$J(\Theta)$$

στη συνέχεια απενεργοποιουμε των κώδικα ελέγχου

6. Χρηση κλίσης καθόδου ή προχωρημένο αλγοριθμου βελτιστοποίσης με τον αλγόριθμο ανάστροφης διάδοσης για να ελαχιστοποιήθει η

$$J(\Theta)$$

ως συνάρτηση των παραμέτρων Θ

Στη θεωρία, σε πολλές περιπτώσεις η

$$J(\Theta)$$

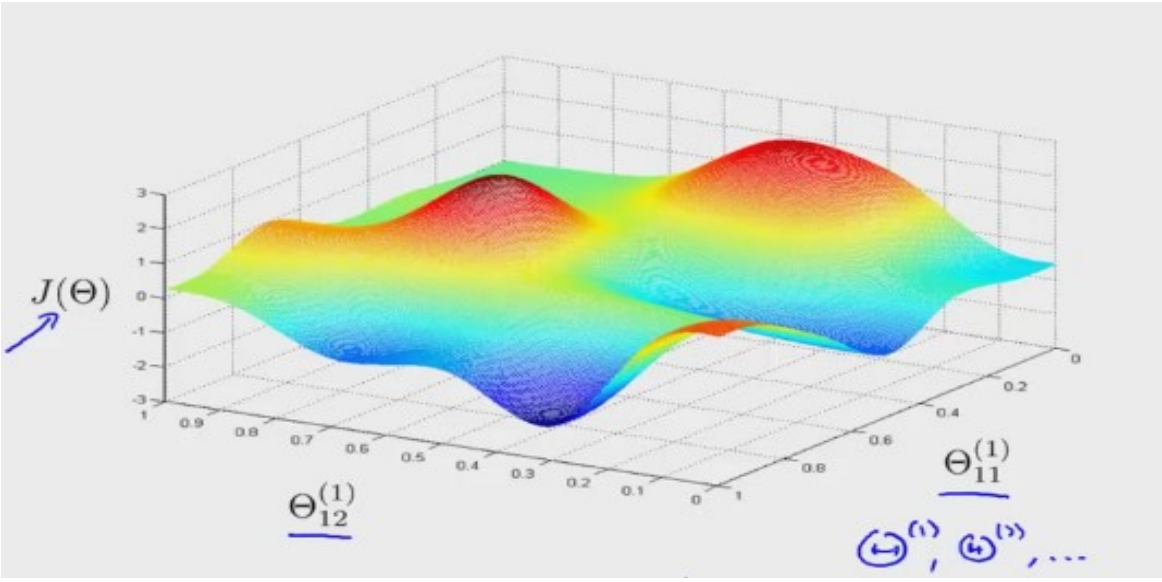
είναι μια μη κυρτή συνάρτηση
αυτό σημαίνει ότι κλίσης καθόδου ή προηγμένων μεθόδων βελτιστοποίησης μπορεί να κολλήσει σε τοπικά βέλτιστα αλλά αποδεικνύεται ότι στην πράξη αυτό δεν είναι συνήθως ένα τεράστιο πρόβλημα,
Ακόμα κι αν δεν μπορούμε να εγγυηθούμε ότι αυτόι οι αλγόριθμοι θα βρούνε μια ολικό ελάχιστο, συνήθως αυτοί οι αλγόριθμοι θα κάνουν πολύ καλή δουλειά ελαχιστοποίησης αυτής της συνάρτηση κόστους και θα πάρθει τουλάχιστον ενα πολύ καλό τοπικό ελάχιστό

Για να πάρετε περισσότερη διαίσθηση για το τι κανει ο αλγοριθμος κλίσης καθόδου σε ένα νευρωνικό δίκτυο

έχουμε μια συνάρτηση κόστους

$$J(\Theta)$$

και ορισμένες παράμετροι Θ

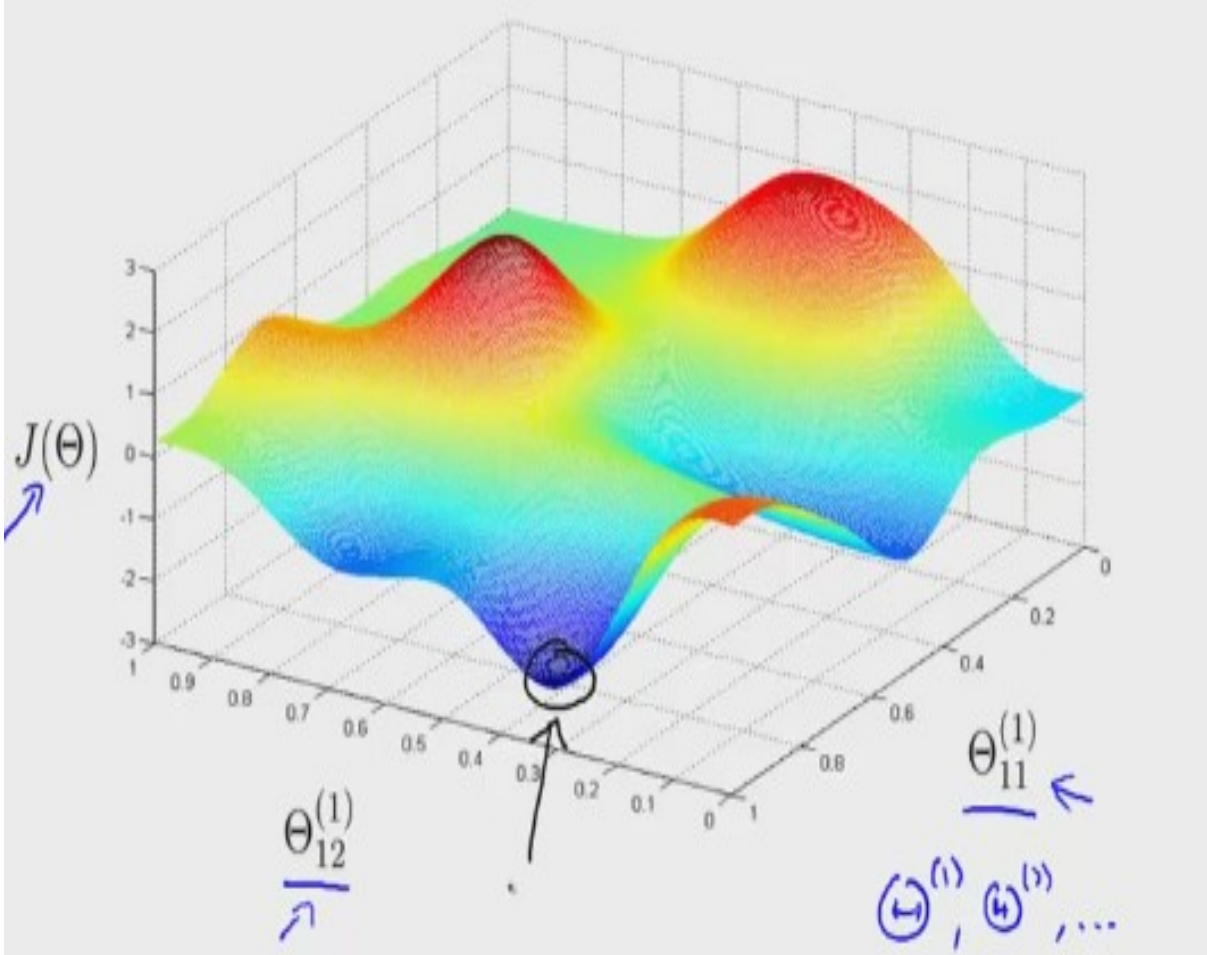


Έστω ότι έχουμε μόνο δύο παράμετροι σε αυτό το νευρωνικό δίκτυο όταν reallity έχουμε πολλά περισσότερα

Έτσι ατή η συνάρτηση κοστους

$$J(\Theta)$$

μέτρα πόσο καλά το νευρωνικό δίκτυο ταιριάζει με τα δεδομένα εκπαίδευσης, ώστε, αν πάρουμε ένα σημείο όπως αυτό

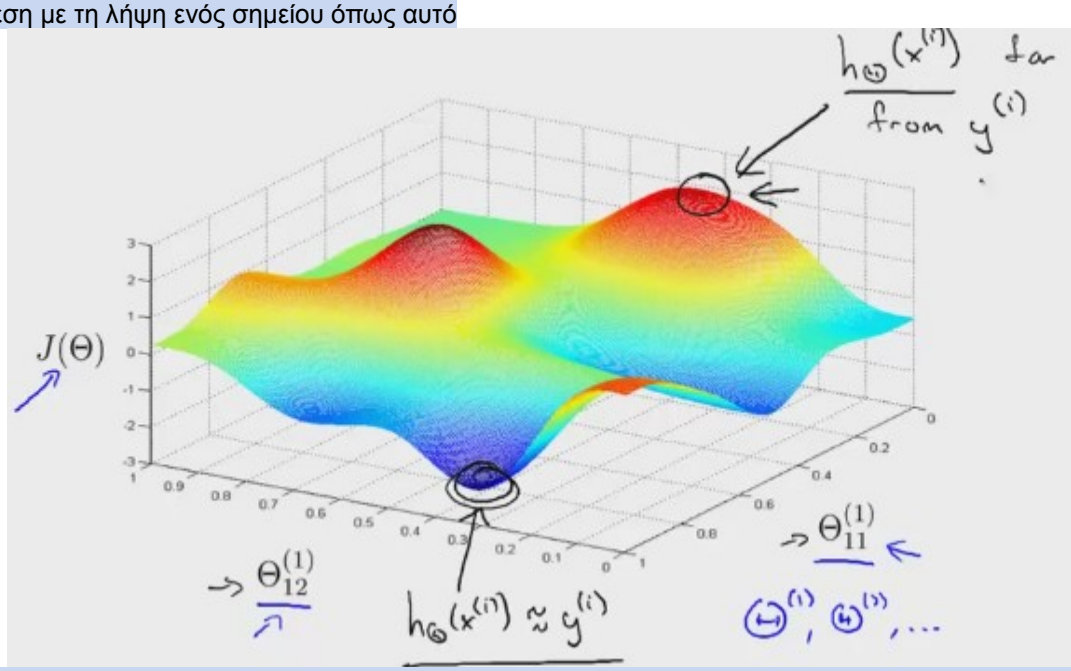


αυτό είναι ένα σημείο όπου

$$J(\Theta)$$

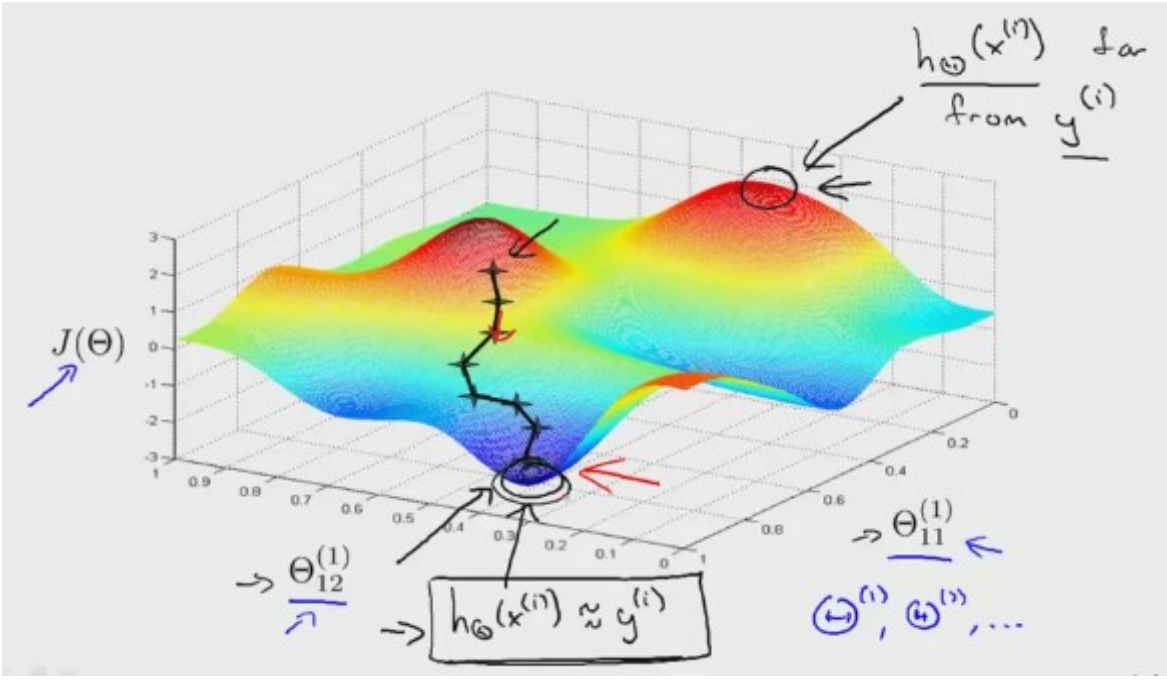
είναι αρκετά χαμηλή και έτσι αυτό αντιστοιχεί σε μια ρύθμιση των παραμέτρων Θ, όπου για τα περισσότερα από τα παραδείγματα εκπαίδευσης η έξοδος της υπόθεσης είναι πολύ κοντά στην y (i) και για αυτό η συνάρτηση κόστους μας είναι πολύ χαμηλή

Σε αντίθεση με τη λήψη ενός σημείου όπως αυτό



Που αντιστοιχεί στο σημείο όπου για πολλά παραδείγματα εκπαίδευσης η έξοδος του νευρικού δικτύου είναι μακριά από την πραγματική τιμή y (i). οπότε σημεία, όπως αυτό αντιστοιχουν στο σημείο όπου η υπόθεση των νευρωνικού δικτύου βγάζει τιμες του συνόλου εκπαίδευσης που απέχουν πολύ από y (I)

Λοιπόν,το τι κανει ο αλγοριθμος κλίσης καθόδου είναι ξεκινώντας από κάποιο τυχαίο σημείο και πηγαίνη με βηματισμό προς τα κατω
Και έτσι αυτό που κάνει ο αλγοριθμος ανάστροφης διάδοσης είναι να υπολογίζει την κατεύθυνση της κλίσης, ενώ η κλίση παίρνει μικρα βήματα προς τα κάτω μέχρι καποια στιγμή να παίρει μια πολύ καλή τοπικού βέλτιστου



3γ. Συνήθη προβλήματα και η επίλυση τους

Ας υποθέσουμε ότι έχουμε εφαρμόσει γενικευμενη γραμμική παλινδρόμηση για την πρόβλεψη των τιμών των κατοικιών
με τη συνάρτηση κόστους να είναι

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{m}\theta_j^2\right]$$

Ωστόσο, όταν ελέγξου την υπόθεσή σας σε ένα νέο σύνολο των σπιτιών θα διαπιστώνουμε ότι κάνει απαράδεκτα μεγάλα λάθη στις προβλέψεις της

κάποιος μπορεί να σκεφτεί να δοκιμάσει είναι να πάρει περισσότερα παραδείγματα εκπαίδευσης, αλλά μερικές φορές το να πάρει περισσότερα δεδομένα εκπαίδευσης δεν βοηθά

Ενα πραγμα που μπορει να δοκιμάστει είναι ένα μικρότερο σύνολο χαρακτηριστικών.

Ή μεγαλιτερο σύνολο χαρακτηριστικών

Ή να δοκιμάστει να προσθεθουν χαρακτηριστικά πολυώνυμου

Ή μείωση του λ

ή αύξηση λ

Ορισμένες από αυτές τις δοκιμές, όπως να πάρθει ένα μεγαλύτερο σύνολο κατάρτισης ή τη περισσότερα χαρακτηριστικά μπορεί να είναι πολύ χρονοβόρες

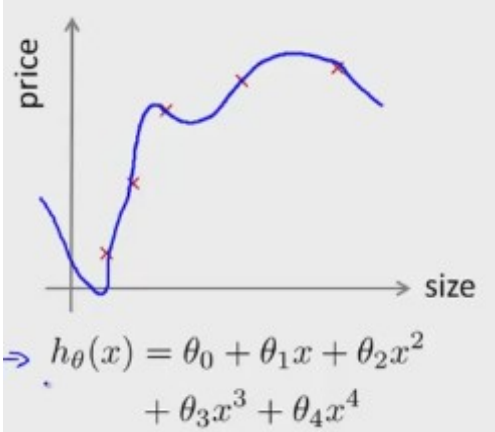
Ο σκοπός αυτής της ενότητας είναι να εισαγάγει μια τεχνική για να αποκλείσει πολλές από αυτές τις επιλογές.

.Αξιολογόντας τον αλγόριθμο

Διάγνωση μηχανικης μαθησης
Ένα τεστ που μπορεί να τρέξει για να αποκτήθει εικόνα για το τι δουλευει η δε δουλευει με εναν αλγόριθμο μηχανικης μάθησης ωστε να γινει πιο κατανοητο το πώς να βελτιώθουν οι επιδόσεις του.

Αξιολογώντας την υπόθεση

Όταν θα ταιριάζουμε με τις παραμετρους της μάθησης του αλγορίθμου μας σκεφτόμαστε για την επιλογή των παραμετρων ωςστε να ελαχιστοποιηθεί το σφάλμα εκπαίδευσης. Μπορει να θεωρηθει ότι το να πάρθει μια πραγματική χαμηλή τιμή του σφάλματος εκπαίδευσης ειναι κατι καλο, αλλά μ επειδή μπορει η υποθεση να έχει ένα πραγματικά χαμηλό σφάλμα αυτό δεν σημαίνει ότι είναι μια καλή υπόθεση. Αυτό είναι ένα παράδειγμα του πώς μια υπόθεση μπορεί να κανει overfit και να αδυνατει να γενικευσεί σε νέα παραδείγματα.



Πώς, λοιπόν, μπορούμε να δούμε αν η υπόθεση αυτή κανει overfiting?
Σε αυτό το απλό παράδειγμα, η υπόθεση μπορεί να σχεδιασθει, αλλά σε γενικές γραμμές για τα προβλήματα με πιο πολλα χαρακτιριστικά από ένα χαρακτηριστικό εως εναν μεγάλο αριθμό από χαρακτηριστικά το να σχεδιάσθει η υπόθεση γίνεται σχεδόν αδυνάτο.
Ο κλασσικός τρόπος με τον αξιολογήται μια υπόθεση είναι αυτός.

Ας υποθέσουμε ότι έχουμε ένα σύνολο δεδομένων, όπως αυτό

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

Εδώ έχουμε δέκα παραδείγματα εκπαίδευσης, αλλά μπορούμε να έχουμε δεκάδες ή εκατοντάδες ή χιλιάδες.

Ενας τρόπος για να βεβαιωθούμεε ότι μπορούμε να ελένξουμε την υπόθεσή μας είναι να χωρίστουν τα δεδομένα που έχουμε σε δύο τμήματα.
Το σύνολο εκπαίδευσης και το σύνολο ελέγχου

	Size	Price	
	2104	400	Training set
	1600	330	
	2400	369	
70%	1416	232	
	3000	540	
	1985	300	
	1534	315	
	1427	199	Test Set
30%	1380	212	
	1494	243	

Πριν από τη διαίρεση του συνόλου δεδομένων πρεπει να γινεται τυχαίο ανακατεμα των δεδομένων

Πρώτα μαθαίνουμε την παράμετρο Θ από τα δεδομένα εκπαίδευσης και υπολογίζουμε το σφάλμα σύνολο ελέγχου

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum^{m_{test}} y_{test}^{(i)} \log h_\theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log h_\theta(x_{test}^{(i)})$$

ή μπορούμε να κάνουμε το σφάλμα ταξινόμησης (0/1 error)

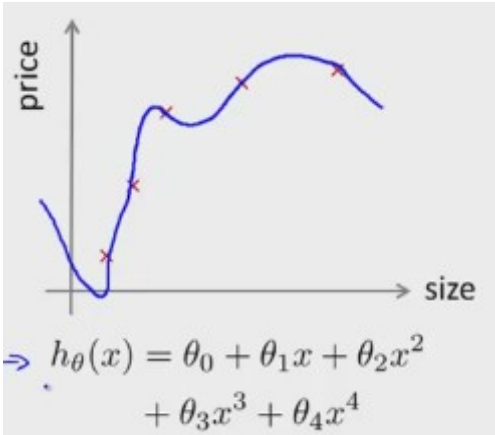
$$err(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, \ y = 0 \\ & \text{or if } h_\theta(x) < 0.5, \ y = 1 \\ 0 & \text{otherwise} \end{cases} \Bigg\} \text{error}$$

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}), y_{test}^{(i)}).$$

παιρνοντας 1 για τη σωστή υπόθεση ή 0 για τη λάθος

Ας υποθέσουμε ότι θέλουμε να αποφασίσουμε ποιο βαθμό πολυωνύμου πρεπει να χρησιμοποιησουμε για να ταιριάξουμε με τα δεδομένα ή ποια χαρακτηριστικά να συμπεριλαβουμε ή να επιλέξουμε την τιμη γενικευσης λ .
Αυτα προβλημάτων ονομάζονται προβλήματα επιλογής μοντέλου.

Στο προηγούμενο παράδειγμα υπερπροσαρμογής



Οι παραμέτροι Θ (0), Θ (1), ... Θ (4) ήταν ταιριασμενες σε ένα σύνολο δεδομένων (σύνολο εκπαίδευσης), το σφάλμα των παραμέτρων, όπως μετράται σε αυτά τα δεδομένα (λάθος εκπαίδευσης) είναι πιθανό να είναι χαμηλότερο από το πραγματικό σφάλμα γενικευσης

Τώρα ας εξετάσουμε την επιλογή μοντέλου
ας πούμε ότι προσπαθουμε να επιλέξεουμε ποιο βαθμό του πολυωνύμου να χρησιμοποιησουμε για να ταιριαξουμε τα δεδομένα

$$h_\theta(x) = \theta_0 + \theta_1 x$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$$
$$\vdots$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$$

έτσι ώστε αν υπάρχει μια επιπλέον παραμέτρος σε αυτόν τον αλγόριθμο ας πούμε d που αντιπροσωπευει το βαθμό πολυωνύμου

$d=1$ 1. $\rightarrow h_\theta(x) = \theta_0 + \theta_1 x$
$d=2$ 2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
$d=3$ 3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$
$\vdots$ $\vdots$
$d=10$ 10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$

ένα πράγμα που θα μπορούσε να γινει είναι να πάρουμε το πρώτο μοντέλο (d = 1) και να ελαχιστοποίησουμε το σφάλμα εκπαίδευσης. παιρνοντας κάποια παραμετρο Θ_1 στη συνέχεια να πάρουμε το δεύτερο μοντέλο παιρνοντας κάποια άλλη παράμετρο Θ_2

και επειτα να υπολογίσουμε τη συνάρτηση κόστους για Θ_n παραμέτρους και να δούμε τι μοντέλο έχει το χαμηλότερο όριο σφάλματος σύνολο ελέγχου

Το πρόβλημα είναι ότι αυτό δεν είναι προκειται να είναι μια καλή εκτίμηση για το πόσο καλή η υπόθεσή γενικεύει επειδή έχουμε διαλέξει την παράμετρο d για την καλύτερη εκτέλεση στο συγκεκριμένο σύνολο δοκιμής.

Λοιπόν, το τι πρέπει να γίνει για να αξιολογήθει μια υπόθεση είναι αντίθετα διάσπαση του σύνολου δεδομένων σε δύο ομάδες να χωριστεί σε τρεις
το πρώτο κομμάτι είναι εκπαίδευσης το δεύτερο κομμάτι είναι το σύνολο επικύρωσης και το τρίτο κομμάτι το σύνολο δοκιμής

	Size	Price	
	2104	400	Training set
	1600	330	
60%	2400	369	
	1416	232	
	3000	540	
	1985	300	
20%	1534	315	Cross validation set (CV)
	1427	199	
20%	1380	212	test set
	1494	243	

Έτσι τώρα, αντί να χρησιμοποιήθει το συνολο δοκιμής για να επιλέχθει το μοντέλο πρόκειται να χρησιμοποιήθει το σύνολο επικύρωσης

$$
\begin{aligned}
h_\theta(x) &= \theta_0 + \theta_1 x \\
h_\theta(x) &= \theta_0 + \theta_1 x + \theta_2 x^2 \\
h_\theta(x) &= \theta_0 + \theta_1 x + \cdots + \theta_3 x^3 \\
&\vdots \\
h_\theta(x) &= \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}
\end{aligned}
$$

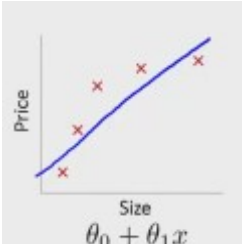
που καταργεί την δέσμευση της δ παραμέτρου με το συνολο δοκιμης και έτσι μπορούμε να χρησιμοποιήσουμε το συνολο δοκιμης για να δουμε αν το επιλεγμένο μοντέλο γενίκευει σωστά

. Διάγνωση και διακύμανσης

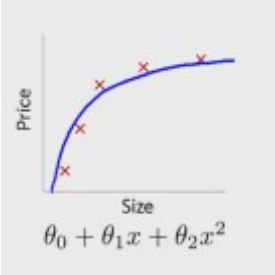
Αν ο αλγόριθμος εκμάθησης δεν αποδίδει έτσι όπως αναμενόταν σχεδόν πάντα είναι επειδή έχει υψηλή προκατάληψη ή πρόβλημα υψηλής διακύμανση, με άλλα λόγια overfittng ή underfiting

Στην περίπτωση αυτή είναι πολύ σημαντικό να καθοριστεί ποια από αυτές τις δύο περιπτώσεις, στην πραγματικότητα συμβαίνει στον αλγόριθμο

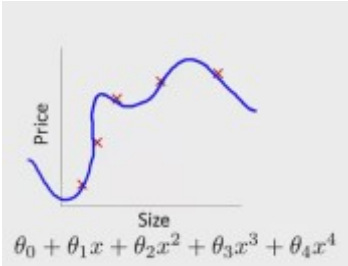
Αυτό είναι ένα παράδειγμα της υπερπροσαρμογής ή underfitting στα δεδομένα



Αυτό είναι ένα παράδειγμα προσαρμογής των δεδομένων όπως θα έπρεπε να είναι



και αυτό ένα παράδειγμα της υψηλής διακύμανσης στα δεδομένα ή overfitting



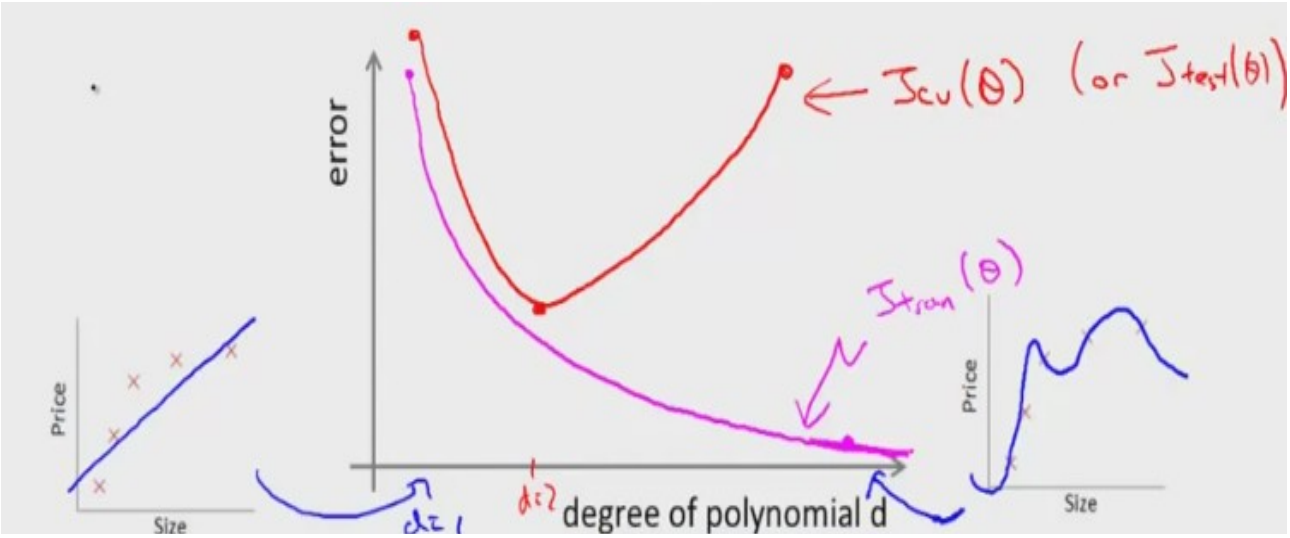
Υπενθιμήζετε ότι η D αναφέρεται στο βαθμό του πολυωνύμου του μοντέλου αντιπροσώπευσης.

Και ότι το λαθος εκπαίδευσης:

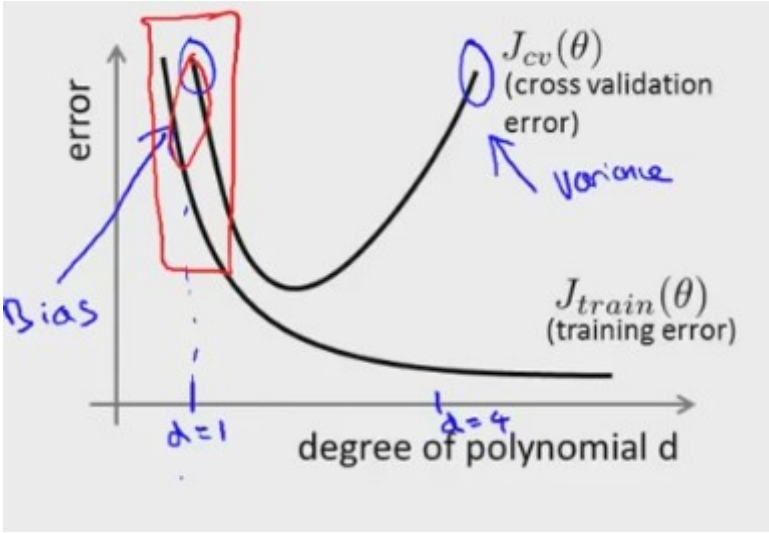
$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Και το λαθος επικύρωσεις :

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

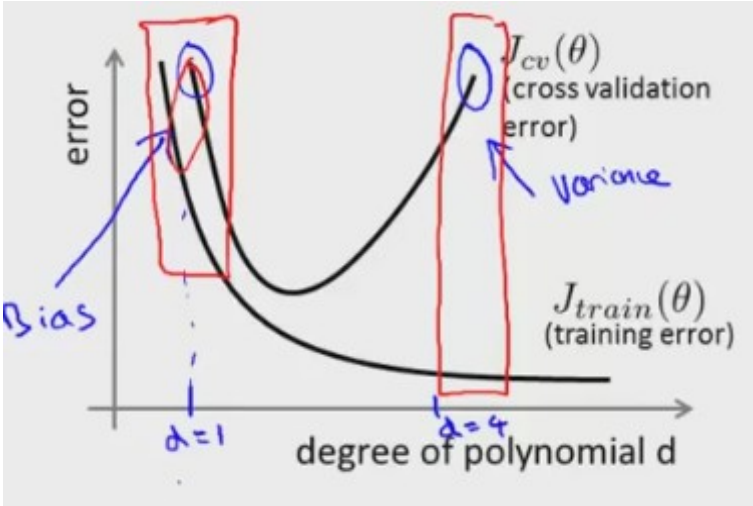


Για σενάριο υπερπροσαρμογής ισχύει:



Δαπιστώνουμε ότι όσο το σφάλμα επικύρωσης οσο και το σφάλμα του συνόλου εκπαίδευσης πρόκειται να είναι υψηλό

Εάν ο αλγόριθμος που πάσχει από υψηλή διακύμανση ισχύει:



το σφάλμα της εκπαίδευσης θα είναι χαμηλό και σε αντίθεση το σφάλμα επικύρωσης θα είναι πολύ μεγαλύτερο

3δ.Προγραμματίζοντας ενα νευρωνικο δικτυο σε Octave gnu-Matlab

Σε αυτη την ενοτητα θα δειχθει κωδικας για Octave gnu η Matlab
για την υλοποιηση ολων οσων επεξηγηθηκαν ανωτερω.

Υπαρχουν δυο εκδοσεις, η μια για αναγνωριση αριθμων και η αλλη κεφαλαιων γραμματων της αγγλικης αλφαβητου.
Αρχικα θα γινει εισαγωγη κωδικα του πρωτου και κυριου προγραμματος (αναγνωριση αριθμων) και επειτα θα δειχθουν οι αλλαγες που εγιναν για την αναγνωριση χαρακτηρων.(οι αλλαγες αυτες ειναι κυριως το μεγεθος της εικονας που εγινε απο 20x20 pixels σε 28x28 pixels και ο καταλληλος κωδικας που επιτρεπει την εισαγωγη τον νεων εικονων(αυτες οι εικονες στο συγκεκριμενο νευρωνικο δικτυο εφοσον ειναι ασπρομαυρες θα μπορουσαν να περιεχουν σχεδον οτιδηποτε(για την εισαγωγη ενχρωμης εικονας θα πρεπει να προηγηθει μια καποια επεξεργασια της εικονας ή να διαμορφωθει αντιστιχα το δικτυο)

Το προγραμμα εκτελει τα βηματα υλοποιησης που εχουν περιγραφει ανωτερω χρησιμοποιωντας τις συναρτησης που ειναι υλοποιημενες στα εξης αρχεια στα εξης αρχεια:

1. NeuralNetMain.m
2. randInitializeWeights.m
3. nn.Costfunction.m
4. fmincg.m
5. DisplayData.m
6. predict.m
7. sigmoid.m
8. sigmoidGradient.m

A. NeuralNetMain.m που ειναι και το κυριως προγραμμα.
Εδω γινονται ολα τα βηματα καλοντας τις αντιστιχες συναρτησης.

A1. φορτωνουμε του dataset μας το ανακατεβουμε και το χωριζουμε στα δυο(δεν κανουμε αξιολογηση της υποθεσης εδω αυτη εχει ηδη γινει) το training kai to test set.Ο διαχωρισμος μπορει να γινει και χωριζοντας το dataset 60%-40% και ανω.

A2. Κανουμε αρχικοποιηση τα συναπτικα βαρη.

A3.Δημιουργουμε ενα μια συντομευση της συναρτησης κοστους για να την περασουμε σαν παραμετρο στον αλγοριθμο βελτιστοποιησης.

A4.Τρεχουμε τον αλγοριθμο βελτιστοποιησης.

A5. Εξαγουμε τα νεα συναπτικα βαρη που εβγαλε η εκπαιδευση του δικυτου.

A6. Καλουμε τη συναρτηση DisplayData για να παρουμε μια εικονα του τι εβλεπε το δικτυο

A7.Τρεχουμε τη συναρτηση predict που εξαγει τα αποτελεσματα που εβγαλε το νευρωνικο δικυτο (με βαση τα βαρυ που εξαγαμε στο A5)

A8.Συγκρινουμε το τι εβγαλε η συναρτηση predict με τα πραγματικα αποτελεσματα του dataset ωστε να δουμε με τι ακριβεια επιτυγχανει την προβλεψη του το νευρωνικο μας δικτυο.

NeuralNetMain.m

```
%% Initialization
clear ; close all;

%% Setup the parameters you will use for this exercise
input_layer_size  = 400;  % 20x20 Input Images of Digits
hidden_layer_size = 25;% 25 hidden units
num_labels = 10;          % 10 labels, from 1 to 10
 % (note   "0" is mapped to label 10)

%% =========== Part 1: Loading and Visualizing Data =============
%  We start by first loading and visualizing the dataset.
%  You will be working with a dataset that contains handwritten digits.
%

% Load Training Data
fprintf('Loading and Visualizing Data ...\n')

load('NeuralDataset.mat');
m = size(X, 1);

% Randomly select 100 data points to display
sel = randperm(size(X, 1));
sel = sel(1:100);
size(X)
K = [ones(size(X,1), 1) X];
K(1:size(y,1),1)=y;
X = K(randperm(size(K,1)),:);
y=X(1:(size(X,1)/2),1);
X1=X(1:(size(X,1)/2),2:end);

X2=X((size(X,1)/2):end,2:end);
Y2=X((size(X,1)/2):end,1);
X=X1;
size(X)
size(y)
size(X2)
size(Y2)

fprintf('Program paused. Press enter to continue.\n');
pause;



%% ================ Part 6: Initializing Pameters ================
```

```
fprintf('\nInitializing Neural Network Parameters ...\n')

initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);

% Unroll parameters
initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];




%% ==================== Part 8: Training NN ====================

%
fprintf('\nTraining Neural Network... \n')


options = optimset('MaxIter', 50);


lambda = 1;

% Create "short hand" for the cost function to be minimized
costFunction = @(p) nnCostFunction(p, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, X, y, lambda);

% Now, costFunction is a function that takes in only one argument (the
% neural network parameters)

fprintf('Program paused. Press enter to continue.\n');
pause;

lambda =1;
[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);

          Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                                                hidden_layer_size, (input_layer_size + 1));
          Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size +
1))):end), ...
                                                num_labels, (hidden_layer_size + 1));


fprintf('\nVisualizing Neural Network... \n')

displayData(Theta1(:, 2:end));

fprintf('\nProgram paused. Press enter to continue.\n');
pause;
```

```
pred = predict(Theta1, Theta2, X);

fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);

pred = predict(Theta1, Theta2, X2);

fprintf('\n Test Set Accuracy: %f\n', mean(double(pred == Y2)) * 100);

end
```

randInitializeWeights.m

```
function W = randInitializeWeights(L_in, L_out)

W = zeros(L_out, 1 + L_in);


epsilon_init = 0.12;
W = rand(L_out, 1 + L_in) * 2 * epsilon_init - epsilon_init;


end
```

nn.Costfunction.m

```
function [J grad] = nnCostFunction(nn_params, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, ...
                                   X, y, lambda)

%

% Reshape nn_params back into the parameters Theta1 and Theta2, the weight matrices
% for our 2 layer neural network
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));

% Setup some useful variables
m = size(X, 1);

% return the following variables correctly
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));




A1 = [ones(m, 1) X];
A2=[ones(m, 1) 1./(1.+exp(-A1*Theta1'))];
A3=1./(1.+exp(-A2*Theta2'));
```

```
for i=1:num_labels; %Transforms y into a m*num_labels matrix of 1s and 0s
    c(:,i)=(y==i);
endfor

J = 1/m*sum(sum(-c.*log(A3)-(1-c).*log(1-A3))); %Without Regularization
REG = sum(sum(Theta1(:,2:end).^2)) + sum(sum(Theta2(:,2:end).^2));

J =  J + (REG*lambda/(2*m) ) ;



for i = 1:m
    a1 = [1, X(i, :)];
    z_2 = Theta1 * a1';
    a2 = [1; sigmoid(z_2)];
    z_3 = Theta2 * a2;
    a3 = sigmoid(z_3);
    yi = 1:size(Theta2,1);
    yi=(yi == y(i));
    delta3 = a3 - yi';

    delta2 = (Theta2' * delta3)(2:end) .* sigmoidGradient(z_2);

    Theta1_grad += delta2 * a1;
    Theta2_grad += delta3 * a2';
endfor
Theta1_grad /= m;
 Theta2_grad /= m;


Theta1_grad(:,2:end)=Theta1_grad(:,2:end)+(lambda*Theta1(:,2:end)/m);

Theta2_grad(:,2:end)=Theta2_grad(:,2:end)+(lambda*Theta2(:,2:end)/m);
% -------------------------------------------------------------

%
=================================================================================
==

% Unroll gradients
grad = [Theta1_grad(:) ; Theta2_grad(:)];


end
```

fmincg.m

```
function [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
```

```
% Read options
if exist('options', 'var') && ~isempty(options) && isfield(options, 'MaxIter')
   length = options.MaxIter;
else
   length = 100;
end


RHO = 0.01;                         % a bunch of constants for line searches
SIG = 0.5;      % RHO and SIG are the constants in the Wolfe-Powell conditions
INT = 0.1;    % don't reevaluate within 0.1 of the limit of the current bracket
EXT = 3.0;                  % extrapolate maximum 3 times the current bracket
MAX = 20;                       % max 20 function evaluations per line search
RATIO = 100;                                % maximum allowed slope ratio

argstr = ['feval(f, X'];                    % compose string used to call function
for i = 1:(nargin - 3)
  argstr = [argstr, ',P', int2str(i)];
end
argstr = [argstr, ')'];

if max(size(length)) == 2, red=length(2); length=length(1); else red=1; end
S=['Iteration '];

i = 0;                                    % zero the run length counter
ls_failed = 0;                          % no previous line search has failed
fX = [];
[f1 df1] = eval(argstr);                  % get function value and gradient
i = i + (length<0);                                  % count epochs?!
s = -df1;                                 % search direction is steepest
d1 = -s'*s;                                        % this is the slope
z1 = red/(1-d1);                            % initial step is red/(|s|+1)

while i < abs(length)                               % while not finished
  i = i + (length>0);                              % count iterations?!

  X0 = X; f0 = f1; df0 = df1;                  % make a copy of current values
  X = X + z1*s;                                      % begin line search
  [f2 df2] = eval(argstr);
  i = i + (length<0);                                  % count epochs?!
  d2 = df2'*s;
  f3 = f1; d3 = d1; z3 = -z1;             % initialize point 3 equal to point 1
  if length>0, M = MAX; else M = min(MAX, -length-i); end
  success = 0; limit = -1;                   % initialize quanteties
  while 1
    while ((f2 > f1+z1*RHO*d1) | (d2 > -SIG*d1)) & (M > 0)
      limit = z1;                                 % tighten the bracket
      if f2 > f1
        z2 = z3 - (0.5*d3*z3*z3)/(d3*z3+f2-f3);                % quadratic fit
      else
        A = 6*(f2-f3)/z3+3*(d2+d3);                            % cubic fit
        B = 3*(f3-f2)-z3*(d3+2*d2);
```

```
      z2 = (sqrt(B*B-A*d2*z3*z3)-B)/A;       % numerical error possible - ok!
    end
    if isnan(z2) | isinf(z2)
      z2 = z3/2;                  % if we had a numerical problem then bisect
    end
    z2 = max(min(z2, INT*z3),(1-INT)*z3);  % don't accept too close to limits
    z1 = z1 + z2;                                   % update the step
    X = X + z2*s;
    [f2 df2] = eval(argstr);
    M = M - 1; i = i + (length<0);                        % count epochs?!
    d2 = df2'*s;
    z3 = z3-z2;                   % z3 is now relative to the location of z2
  end
  if f2 > f1+z1*RHO*d1 | d2 > -SIG*d1
    break;                                        % this is a failure
  elseif d2 > SIG*d1
    success = 1; break;                                   % success
  elseif M == 0
    break;                                            % failure
  end
  A = 6*(f2-f3)/z3+3*(d2+d3);                     % make cubic extrapolation
  B = 3*(f3-f2)-z3*(d3+2*d2);
  z2 = -d2*z3*z3/(B+sqrt(B*B-A*d2*z3*z3));        % num. error possible - ok!
  if ~isreal(z2) | isnan(z2) | isinf(z2) | z2 < 0   % num prob or wrong sign?
    if limit < -0.5                           % if we have no upper limit
      z2 = z1 * (EXT-1);                 % the extrapolate the maximum amount
    else
      z2 = (limit-z1)/2;                              % otherwise bisect
    end
  elseif (limit > -0.5) & (z2+z1 > limit)          % extraplation beyond max?
    z2 = (limit-z1)/2;                                        % bisect
  elseif (limit < -0.5) & (z2+z1 > z1*EXT)       % extrapolation beyond limit
    z2 = z1*(EXT-1.0);                          % set to extrapolation limit
  elseif z2 < -z3*INT
    z2 = -z3*INT;
  elseif (limit > -0.5) & (z2 < (limit-z1)*(1.0-INT))   % too close to limit?
    z2 = (limit-z1)*(1.0-INT);
  end
  f3 = f2; d3 = d2; z3 = -z2;                  % set point 3 equal to point 2
  z1 = z1 + z2; X = X + z2*s;                     % update current estimates
  [f2 df2] = eval(argstr);
  M = M - 1; i = i + (length<0);                        % count epochs?!
  d2 = df2'*s;
end                                           % end of line search

if success                                    % if line search succeeded
  f1 = f2; fX = [fX' f1]';
  fprintf('%s %4i | Cost: %4.6e\r', S, i, f1);
  s = (df2'*df2-df1'*df2)/(df1'*df1)*s - df2;      % Polack-Ribiere direction
  tmp = df1; df1 = df2; df2 = tmp;                      % swap derivatives
  d2 = df1'*s;
  if d2 > 0                                   % new slope must be negative
```

```
      s = -df1;                          % otherwise use steepest direction
      d2 = -s'*s;
    end
    z1 = z1 * min(RATIO, d1/(d2-realmin));         % slope ratio but max RATIO
    d1 = d2;
    ls_failed = 0;                           % this line search did not fail
  else
    X = X0; f1 = f0; df1 = df0;  % restore point from before failed line search
    if ls_failed | i > abs(length)         % line search failed twice in a row
      break;                          % or we ran out of time, so we give up
    end
    tmp = df1; df1 = df2; df2 = tmp;                      % swap derivatives
    s = -df1;                                          % try steepest
    d1 = -s'*s;
    z1 = 1/(1-d1);
    ls_failed = 1;                               % this line search failed
  end
  if exist('OCTAVE_VERSION')
    fflush(stdout);
  end
end
fprintf('\n');
```

DisplayData.m

```
function [h, display_array] = displayData(X, example_width)
%DISPLAYDATA Display 2D data in a nice grid
%   [h, display_array] = DISPLAYDATA(X, example_width) displays 2D data
%   stored in X in a nice grid. It returns the figure handle h and the
%   displayed array if requested.

% Set example_width automatically if not passed in
if ~exist('example_width', 'var') || isempty(example_width)
            example_width = round(sqrt(size(X, 2)));
end

% Gray Image
colormap(gray);

% Compute rows, cols
[m n] = size(X);
example_height = (n / example_width);

% Compute number of items to display
display_rows = floor(sqrt(m));
display_cols = ceil(m / display_rows);

% Between images padding
pad = 1;

% Setup blank display
```

```
display_array = - ones(pad + display_rows * (example_height + pad), ...
                    pad + display_cols * (example_width + pad));

% Copy each example into a patch on the display array
curr_ex = 1;
for j = 1:display_rows
        for i = 1:display_cols
                if curr_ex > m,
                        break;
                end
                % Copy the patch

                % Get the max value of the patch
                max_val = max(abs(X(curr_ex, :)));
                display_array(pad + (j - 1) * (example_height + pad) + (1:example_height), ...
                        pad + (i - 1) * (example_width + pad) + (1:example_width)) = ...
                                                reshape(X(curr_ex, :), example_height,
example_width) / max_val;
                curr_ex = curr_ex + 1;
        end
        if curr_ex > m,
                break;
        end
end

% Display Image
h = imagesc(display_array, [-1 1]);
% Do not show axis
axis image off

drawnow;

end
```

predict.m

```
function p = predict(Theta1, Theta2, X)


m = size(X, 1);
num_labels = size(Theta2, 1);


p = zeros(size(X, 1), 1);

h1 = sigmoid([ones(m, 1) X] * Theta1');
h2 = sigmoid([ones(m, 1) h1] * Theta2');
[dummy, p] = max(h2, [], 2);

end
```

sigmoid.m

```
function g = sigmoid(z)


g = 1.0 ./ (1.0 + exp(-z));
end
```

sigmoidGradient.m

```
function g = sigmoidGradient(z)

g = zeros(size(z));

g = sigmoid(z) .* (1 - sigmoid(z));


end
```

Οι Αλλαγες που γινονται για την αναγνωριση χαρακτιρων ειναι η εξης:
Εισαγετε η συναρτηση InputData για τι δυνατοτητα εισαγωγης πολλων χαρακτηρων για τη δημιουργια ενος Dataset. Η συναρτηση αυτη διαβαζει συνεχεια ενα αρχειο εικονας(πχ Example.png) ελενχει ποσα γραμματα υπαρχουν(αναλογα με το μεγεθος τις εικονας(καθε γραμμα θεωρειτε πως ειναι 28x28 pixels) απλωνει τα pixels σε γραμμες και τα αποθηκευει σε ενα αρχειο ωστε να μπορει να παρει τα δεδομενα το δικτυο.

InputData.m

```
function p=inputtest()
while(1)
fprintf('\n press 0 to stop the process\n');
Q=input('input');
if Q==0
break;
else
end
pit=imread('Example.png');


si=size(reshape(pit,1,[]))

si=size(reshape(pit,1,[]),2)
ri=1;
rj=784;
if si>784
fprintf('\n more than one letters\n');
fprintf('Program paused. Press enter to load.\n');
pause;
letters=si/784

fprintf('\n more than one letters\n');
i=1;
size(pit)
pit=reshape(pit,1,[]);
size(pit)
for i=1:letters


new(i,:)=pit(1,ri:rj);


ri=rj+1;
rj=rj+784;
fprintf('letter saved \n');



fprintf(' Press enter to load.\n');
pause;
```

```
endfor

dlmwrite('Dataset.txt',new,'-append','newline','pc');
p=new;

elseif si<=784
fprintf('\n one letter\n');
fprintf('Program paused. Press enter to load.\n');
pause;

dlmwrite('Dataset.txt',pit,'-append','newline','pc');
p=pit;
endif

end
end
```

Εισαγεται επισης η συναρτηση TrainChar η οποια δινει τη δυνατοτητα αφου το νευρωνικο δικτυο εχει εκπαιδευτη να του δωσουμε μια νεα εικονα και να δουμε την καταλαβαινει το δικτυο πως ειναι.

TrainChar.m

```
function p=TRAINCHAR()
pit=imread('Example.png');


si=size(reshape(pit,1,[]))

si=size(reshape(pit,1,[]),2)
ri=1;
rj=784;
if si>784
fprintf('\n more than one letters\n');
fprintf('Program paused. Press enter to load.\n');
pause;
letters=si/784

fprintf('\n more than one letters\n');
i=1;
size(pit)
pit=reshape(pit,1,[]);
size(pit)
for i=1:letters


new(i,:)=pit(1,ri:rj);
```

```
ri=rj+1;
rj=rj+784;
fprintf('letter  saved \n');



fprintf(' Press enter to load.\n');
pause;

endfor

dlmwrite('Example.txt',new);
p=new;

elseif si<=784
fprintf('\n  one letter\n');
fprintf('Program paused. Press enter to load.\n');
pause;
dlmwrite('Example.txt',pit);
p=pit;
endif

end
```

Η σημαντικοτερες αλλαγες γινονται στο κυριο προγραμμα . Εκει αλλαζει το μεγεθος τον εξοδων του δικτυου (συγκεκριμενα τωρα για τα γραμματα γινονται 26). Επισης δημιουργειτε η δυνατοτητα αμα το δικτυο κανει μια λαθος προβλεψει να ισαχθει απο το χρηστη η σωστη απαντηση και να σωθει στο Training set για μελλοντικη χρηση.
Προσοχη, στο συγκεκριμενο προγραμμα τα αντικειμενα που καλειται το νευρωνικο δικτυο να αναγνωρισει μπορουν να αντιστιχθουν μονο με αριθμους(πχ στο συγκεκριμενο παραδειγμα με τα γραμματα το καθε γραμμα αντιστιχοιζετε με εναν αριθμο. Δλδ 1=A,2=B,3=G....26=Z

NeuralNetMainChar.m

```
%% Initialization
clear ; close all; clc


input_layer_size  = 784;  % 28x28 Input Images of Digits
hidden_layer_size = 25;   % 25 hidden units
num_labels = 26;          % 10 labels, from 1 to 10
                          % (note that we have mapped "0" to label 10)



fprintf('\nInitializing Neural Network Parameters ...\n')

initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);


initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];





fprintf('\nTraining Neural Network... \n')
while(1)
fprintf('\press enter to begin... \n')
pause;

options = optimset('MaxIter', 50);

%  You should also try different values of lambda
lambda = 1;
data=load('ExampleTrainingset.txt');
size(data)

y=data(:,1);
X=data(:,2:end);


costFunction = @(p) nnCostFunction(p, ...
                    input_layer_size, ...
                    hidden_layer_size, ...
                    num_labels, X, y, lambda);


[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);
```

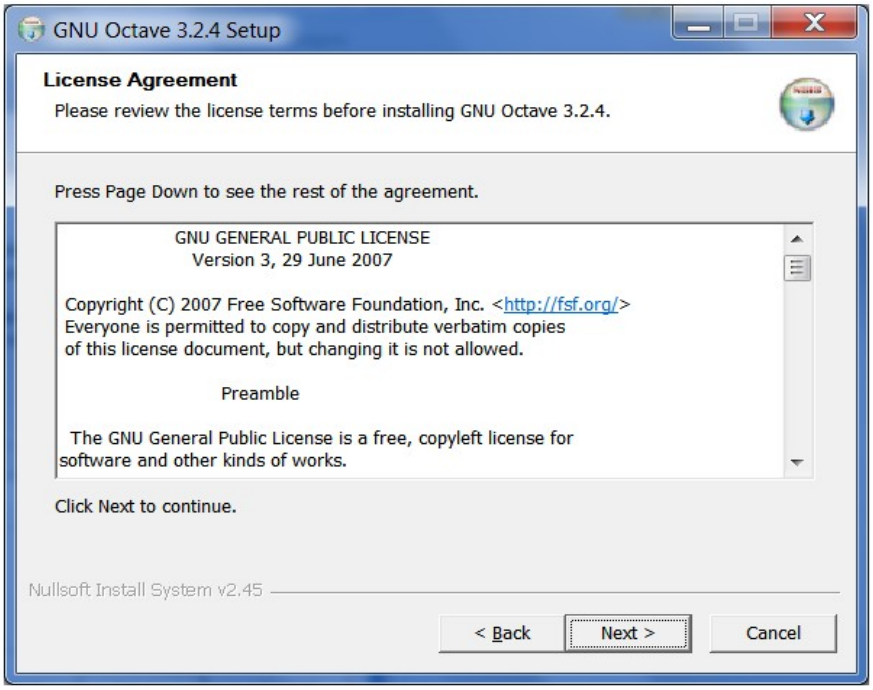
```
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));



pred = predict(Theta1, Theta2, X);

fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);

[p]=TRAINCHAR;
data2=load('Exampletest.txt');

pred = predict(Theta1, Theta2, data2);
pred=pred'

letters=size(p,1);


pit=reshape(p,28,28*letters);

if letters==1
fprintf('\n was the prediction correct ?.\n')


fprintf('\n press 1 if it was or 0 if it wasnt now.\n')
Q=input('input');
if Q==1
CH = [ones(letters, 1) p];
CH(1)=pred;
K=CH(1);
dlmwrite('ExampleTrainingset.txt',CH,'-append','newline','pc');
fprintf('\n the character is now saved for future usage.\n');
fprintf('Program paused. Press enter to load.\n');
pause;


else
fprintf('\n please enter the correct number now .\n');

Q=input('input');



CH = [ones(1, 1) p];
CH(1)=Q;
fprintf('\n the number you entered was.\n')
```

```
Q
fprintf('\n procced and saving this number to database? .\n')

fprintf('\n press 1 to save 0 to abord .\n');

Q=input('input');
if Q==1
dlmwrite('ExampleTrainingset.txt',CH,'-append','newline','pc');
fprintf('\n the character is now saved for future usage.\n');
endif

endif
%=========================word================
elseif letters>=1

fprintf('\n press 1 if it was or 0 if it wasnt now.\n')

Q=input('input');
pause;
if Q==1

ri=1;
rj=784;
new=zeros(letters,784);

new=p;
CH = [ones(size(new,1), 1) new];
size(new);
size(CH);




CH(1:letters,1)=pred;
K=CH(1:letters,1)';


dlmwrite('ExampleTrainingset.txt',CH,'-append','newline','pc');
fprintf('\n the character is now saved for future usage.\n');


fprintf('\n the characters of the word are now saved for future usage.\n');
fprintf('Program paused. Press enter to load.\n');
pause;


else

fprintf('\n please enter the correct numbers now .\n');
ri=1;
rj=784;
new=zeros(letters,784);
```

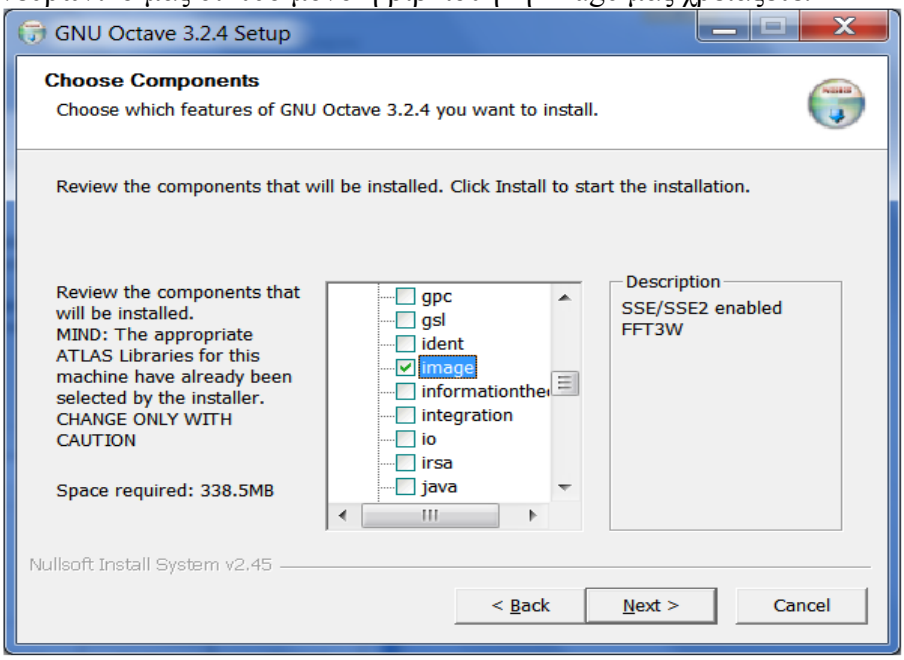
```
new=p;
        CH = [ones(size(new,1), 1) new];
        size(new);
        size(CH);
        for i=1:letters
                        fprintf('\n enter the next number .\n');
                        Q=input('input');
                        CH(i,1)=Q;
                        fprintf('\n the number you entered was.\n')
                        Q

                        pause;

                        endfor
                        CH(:,1);
                        size(CH);
                        fprintf('\n procced and saving this number to database? .\n')

                        fprintf('\n press 1 to save 0 to abord .\n');

                        Q=input('input');
                        pause;
                        if Q==1

                        dlmwrite('ExampleTrainingset.txt',CH,'-append','newline','pc');
                        fprintf('\n the character is now saved for future usage.\n');

                        fprintf('\n the characters of the word are now saved for future usage.\n');
                        endif

                        endif
                        %=========================
                        endif

end
```

4.Οδηγος για την εγκατασταση του Octave gnu σε λειτουργικο Windows.

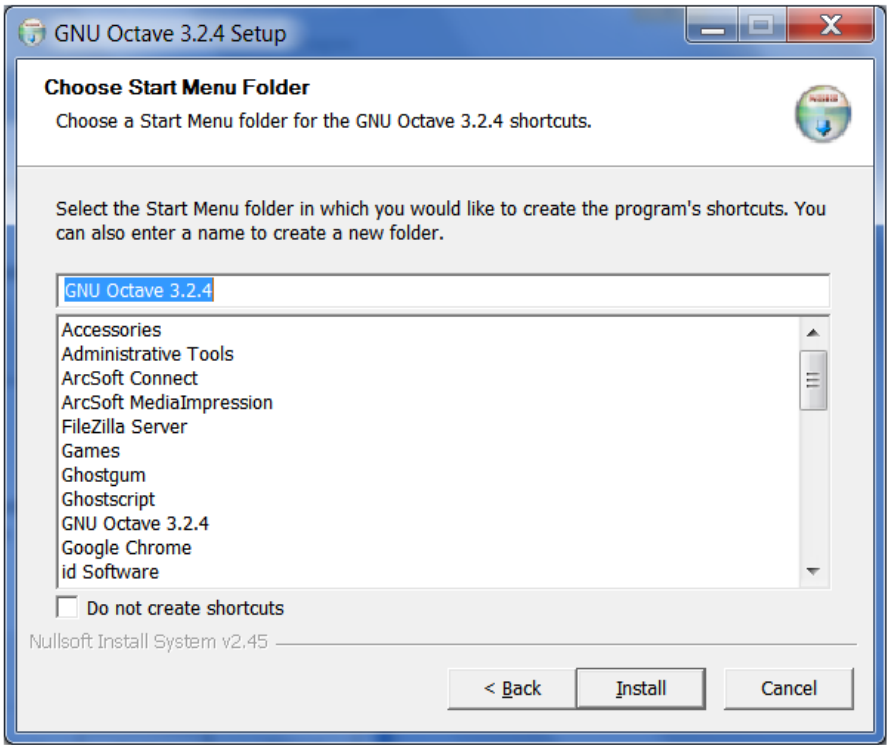
1. Κατεβαστε το Octave απο το http://s3.amazonaws.com/mlclass-resources/software/Octave-3.2.4_i686-pc-mingw32_gcc-4.4.0_setup.exe
2. Τρεξτε το αρχειο και στο παραθυρο που θα εμφανιστει για να του επιτρεψετε να κανει αλλαγες σε αυτον τον υπολογιστει πατηστε YES
3. Στα επομενα δυο παραθυρα πατηστε NEXT



4.Εδω μπορειτε να διαλεξετε πιες βιβλιοθηκες να εγκαταστησετε.
Για το νευρωνικο μας δικτυο μονο η βιβλιοθηκη image μας χρειαζετε.

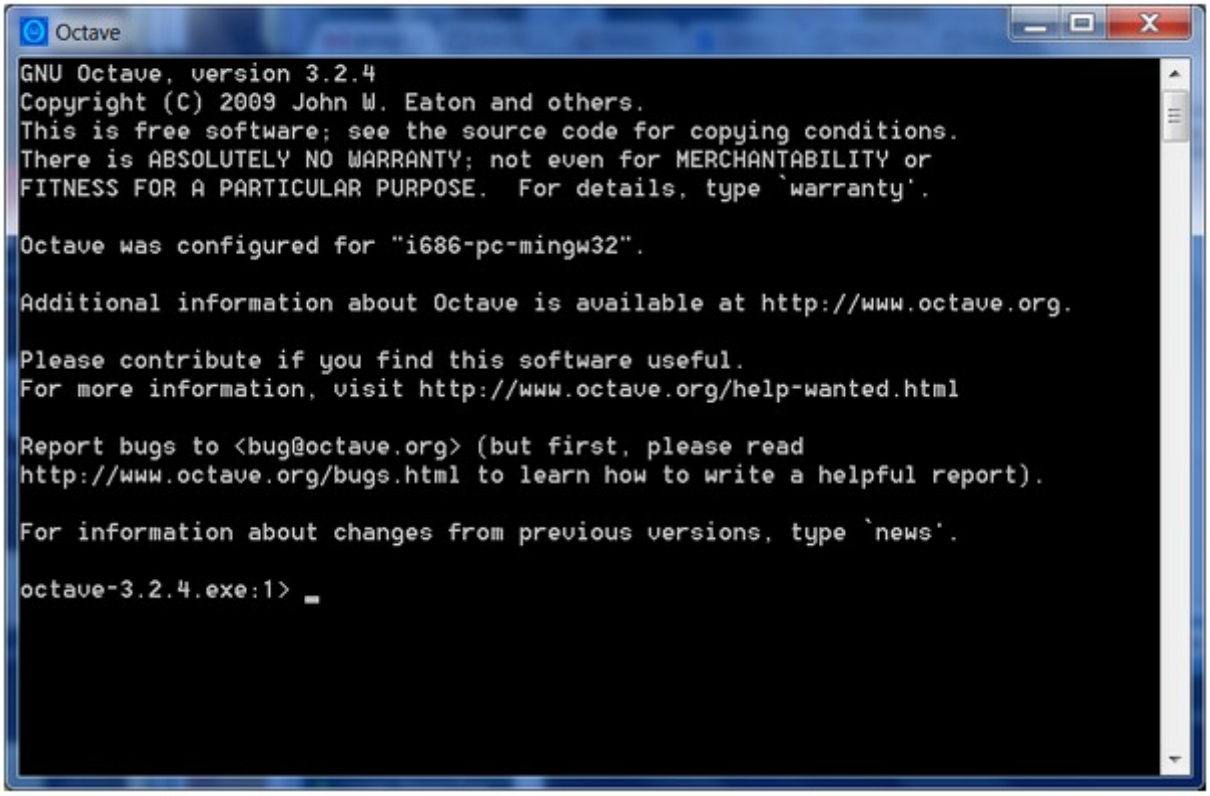


5.Στο επομενο παραθυρο επιλεξτε Install



Τωρα θα πρεπει να αρχισει η εγκατασταση. Οταν τελειωσει πατηστε finish.

Αν τρεξετε το προγραμμα θα δειτε ενα περιβαλλον σαν και αυτο.



Τον κωδικα για το νευρωνικο δικτυο μπορειτε να τον εχετε σε ενα φακελο ας πουμε Nnfolder εκει μπορειτε να εχετε τα αρχεια με τις συναρτησεις και το κυριος σας προγραμμα.
Τον κωδικα μπορειτε να τον εισαγετε σε οποιοδηποτε text editor(notepad,wordpad,notepad++ κλπ..) και απλα το αποθηκευετε με την καταλυξη .m

Επειτα στο commandline του Octave γραφετε το filepath του αρχειου (πχ cd 'C:\Users\teipeir\Documents\Nnfolder')
και γραφετε στην επομενη γραμμη τη συναρτηση που θελετε να καλεσετε (πχ NeuralNetMain ή InputTestSet)