SPARSE CODING AND COMPRESSED SENSING: LOCALLY COMPETITIVE ALGORITHMS AND RANDOM PROJECTIONS

by

William Edward Hahn

A Dissertation Submitted to the Faculty of

The Charles E. Schmidt College of Science

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Florida Atlantic University

Boca Raton, FL

August 2016

SPARSE CODING AND COMPRESSED SENSING:

LOCALLY COMPETITIVE ALGORITHMS AND RANDOM PROJECTIONS

by

William Edward Hahn

This dissertation was prepared under the direction of the candidate's dissertation advisor, Dr. Elan Barenholtz, Center for Complex Systems, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the Charles E. Schmidt College of Science and was accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

SUPERVISORY COMMITTEE:

Elan Barenholtz, Ph.D.
Dissertation Advisor

George Kalantzis, Ph.D.

Steven Bressler, Ph.D.
Interim Chair, Center for Complex Systems

Howard Hock, Ph.D.

Janet Blanks, Ph.D.
Interim Dean, The Charles E. Schmidt College of Science

Deborah L. Floyd, Ee. D.
Dean, Graduate College

07-07-2016
Date

ACKNOWLEDGEMENTS

I would like to thank my advisor, Elan Barenholtz, for his wisdom, guidance, patience, and friendship. I also wish to thank my committee members, George Kalantzis and Howard Hock for their valued advice and support. I would like to thank Janet Blanks for her continuous encouragement and for her recommendation that I study sparse modeling. Special thanks to Keyla Thamsten and Rhona Frankel for all of their help and hard work.

To all the of faculty, students, and friends of the Center for Complex Systems and Brain Sciences, thank you for allowing me to join this community and for sharing your creative spirits and passionate minds.

ABSTRACT

Author:	William Edward Hahn
Title:	Sparse Coding and Compressed Sensing: Locally Competitive Algorithms and Random Projections
Institution:	Florida Atlantic University
Dissertation Advisor:	Dr. Elan Barenholtz
Degree:	Doctor of Philosophy
Year:	2016

For an 8-bit grayscale image patch of size n x n, the number of distinguishable signals is $256^{(n^2)}$. Natural images (e.g.,photographs of a natural scene) comprise a very small subset of these possible signals. Traditional image and video processing relies on band-limited or low-pass signal models. In contrast, we will explore the observation that most signals of interest are sparse, i.e. in a particular basis most of the expansion coefficients will be zero. Recent developments in sparse modeling and L1 optimization have allowed for extraordinary applications such as the single pixel camera, as well as computer vision systems that can exceed human performance. Here we present a novel neural network architecture combining a sparse filter model and locally competitive algorithms (LCAs), and demonstrate the networks ability to classify human actions from video. Sparse filtering is an unsupervised feature learning algorithm designed to optimize the sparsity of the feature distribution directly without having the need to model the data distribution. LCAs are defined by a system of differential equations where the initial conditions define an optimization problem and

the dynamics converge to a sparse decomposition of the input vector. We applied this architecture to train a classifier on categories of motion in human action videos. Inputs to the network were small 3D patches taken from frame differences in the videos. Dictionaries were derived for each action class and then activation levels for each dictionary were assessed during reconstruction of a novel test patch. We discuss how this sparse modeling approach provides a natural framework for multi-sensory and multimodal data processing including RGB video, RGBD video, hyper-spectral video, and stereo audio/video streams.

DEDICATION

Thelbert Garris Worthington Martha Finch Abernathy Edward Nicholas Hahn, and Alexander Tatge Finamore

SPARSE CODING AND COMPRESSED SENSING: LOCALLY COMPETITIVE ALGORITHMS AND RANDOM PROJECTIONS

Chapter 1

Introduction

1.1 COMPUTERS AND THE BRAIN

When we talk mathematics, we may be discussing a secondary language built on the primary language of the nervous system. - John Von Neumann

This work brings together research from computer science and neuroscience into a framework that elucidates some of the functions of biological brains. It is difficult to describe the brain given that I am a brain and cannot be objective in the matter. It is made even more difficult by the effects of evolution, both genetic and cultural. Many of the capabilities we attribute to humans brains are due to an elaborate mix of genetics, development, information processing, and cultural education. In this work we will focus on the information processing aspects of the brain, i.e how the brain learns to adapt itself to create action policies.

There likely at least many ideas about brains as there are brains in the world. Therefore, it is important to emphasize the historical context of much of the modern artificial intelligence and machine learning achievements that are now commonplace. As LeVar Burton says "you don't have to take my word for it." It is a goal of the current document to leave many of the original voices intact, to convey how the story of ar-

tificial intelligence is the story of humanity and our journey to understand our place in the universe. Sir Karl Popper Popper writes "Before we as individuals are even conscious of our existence we have been profoundly influenced for a considerable time (since before birth) by our relationship to other individuals who have complicated histories, and are members of a society which has an infinitely more complicated and longer history than they do (and are members of it at a particular time and place in that history); and by the time we are able to make conscious choices we are already making use of categories in a language which has reached a particular degree of development through the lives of countless generations of human beings before us...We are social creatures to the inmost centre of our being. The notion that one can begin anything at all from scratch, free from the past, or unindebted to others, could not conceivably be more wrong."[99]

It is hard to find the origins computation and harder still to justify linear narratives that explain the developments of technology, computing technology in particular. A common place to start is in 1844 with Ada Lovelace who pronounced "It does not appear to me that cerebral matter need be more unmanageable to mathematicians than sidereal & planetary matter & movements; if they would but inspect it from the right point of view...I have my hopes, and very distinct ones too, of one day getting cerebral phenomena such that I can put them into mathematical equations–in short, a law or laws for the mutual actions of the molecules of brain. I hope to bequeath to the generations a calculus of the nervous system."[57]

Nineteen years later in 1863 Samuel echoed "There are few things of which the present generation is more justly proud than of the wonderful improvements which are daily taking place in all sorts of mechanical appliances. It is unnecessary to mention these

here, for they are sufficiently obvious; our present business lies with considerations which may somewhat tend to humble our pride and to make us think seriously of the future prospects of the human race. If we revert to the earliest primordial types of mechanical life, to the lever, the wedge, the inclined plane, the screw and the pulley, or (for analogy would lead us one step further) to that one primordial type from which all the mechanical kingdom has been developed, we mean to the lever itself, and if we then examine the machinery of the Great Eastern, we find ourselves almost awestruck at the vast development of the mechanical world, at the gigantic strides with which it has advanced in comparison with the slow progress of the animal and vegetable kingdom. We shall find it impossible to refrain from asking ourselves what the end of this mighty movement is to be." [22]

John McCarthy the scientist who coined the term artificial intelligence claimed "Well, there are two ways of looking at things. You can either look at it from the point of view of biology, or from the point of view of computer science. From the point of view of biology, you could try to imitate the nervous system insofar as you understood the nervous system, or you could try to imitate human psychology insofar as you understand human psychology. The computer-science way of looking at it says that we look at the world and we try to see what problems it presents in order to achieve goals and think about the world rather than about the biology per se. And I would say that the computer-science approach is the one that so far has had the most success."[104]

Further, McCarthy defined Artificial Intelligence as a science, namely the study of problem solving and goal achieving processes in complex situations. A basic science like mathematics or physics, requiring experimentation, and with problems distinct

from applications, and distinct from the study of how human and animal brains work.[104]

The Internet encyclopedia entry on the so called "AI winter" lists 1970 "as the abandonment of connectionism. The first spring would come in the 1980's with John Hopfield's interpretation of computation as physical process, trying to consider digital computers, analog computers, and the brain as having things in common.

This idea was a revival of much older ideas in particular John von Neumann in 1951 describes how "it is perfectly possible that the simplest and only practical way to actually say what constitutes a visual analogy consists in giving a description of the connections of the visual brain. A new, essentially logical, theory is called for in order to understand high-complication automata and, in particular, the central nervous system. It may be, however, that in this process logic will have to undergo a pseudomorphosis to neurology to a much greater extent than the reverse."[157]

Indeed Konrad Zuse, who independently pioneered many of the early ideas in hardware and software design, in a 1980's lecture at the computer history museum lecture tells the audience "I concentrated my ideas more on the relations between man and machine.....I didn't see any border between calculating and thinking...surely at that time the computers we could make at that time they were far away from being electronic brains...today I hope that your and my brains are ahead of the computers."[172]

Butler in 1863 had already asked "In what direction is it tending? What will be its upshot?" [22] As Turing suggests it is the general goal of AI is to "investigate the question as to whether it is possible for machinery to show intelligent behaviour. It is

usually assumed without argument that it is not possible." [151] In line with Turing, it is the authors contention "that machines can be constructed which will simulate the behaviour of the human mind very closely."[151]

Butler pointed out the connection between Darwin's 'new' theory of evolution and the proliferation of mechanical aids that we becoming common in the 19th century. He writes "We regret deeply that our knowledge both of natural history and of machinery is too small to enable us to undertake the gigantic task of classifying machines into the genera and sub-genera, species, varieties and sub-varieties, and so forth, of tracing the connecting links between machines of widely different characters, of pointing out how subservience to the use of man has played that part among machines which natural selection has performed in the animal and vegetable kingdoms, of pointing out rudimentary organs which exist in some few machines, feebly developed and perfectly useless, yet serving to mark descent from some ancestral type which has either perished or been modified into some new phase of mechanical existence." [22] Butler drawing on first principles and seeking to generalize the tree of life reveals "as the vegetable kingdom was slowly developed from the mineral, and as in like manner the animal supervened upon the vegetable, so now in these last few ages an entirely new kingdom has sprung up, of which we as yet have only seen what will one day be considered the antediluvian prototypes of the race."[22] He uses the words "mechanical life," "the mechanical kingdom," and "the mechanical world". Analogous to how a naturalist would describe a turtle shell Butler describes a pocket watch "the beautiful structure of the little animal, watch the intelligent play of the minute members which compose it; yet this little creature is but a development of the cumbrous clocks of the thirteenth century it is no deterioration from them. The day may come when clocks, which certainly at the present day are not diminishing

in bulk, may be entirely superseded by the universal use of watches, in which case clocks will become extinct like the earlier saurians, while the watch (whose tendency has for some years been rather to decrease in size than the contrary) will remain the only existing type of an extinct race."[22]

The idea of a digital computer was already "an old one" in 1950.[152] Turing wrote of Charles Babbage, Lucasian Professor of Mathematics at Cambridge from 1828 to 1839, and how he "planned such a machine, called the Analytical Engine, but it was never completed."[152] Even though it was never fully operational Babbage's Engine nonetheless was able to drive the worlds imagination. Harry Wilmot Buxton proclaimed "The marvelous pulp and fibre of a brain had been substituted by brass and iron: he had taught wheelwork to think." Doron Swade, biographer and historian of computing describes how Babbages remarked "It will jam, it will break, but it will never deceive." Babbage, tasked with eliminating errors in mathematical tables has been said to exclaim "I wish to God these calculations had been executed by steam!"

Gregory Chaitin relays how Gottfried Wilhelm Leibniz, long before Baggage had "talked about avoiding disputesand he was probably thinking of political disputes and religious disputesby calculating who was right instead of arguing about it! Instead of fighting, you should be able to sit down at a table and say, "Gentleman, let us compute¡' What a beautiful fantasy!"[30]

Turing knew that "although Babbage had all the essential ideas, his machine was not at that time such a very attractive prospect. The storage was to be purely mechanical, using wheels and cards."[152] Turing's most detailed information of "Babbages Analytical Engine comes from a memoir by Lady Lovelace (1842). In it she states,

'The Analytical Engine has no pretensions to originate anything. It can do whatever we know how to order it to perform"'.[152]

Hartree in 1949 added "This does not imply that it may not be possible to construct electronic equipment which will "think for itself," or in which, in biological terms, one could set up a conditioned reflex, which would serve as a basis for learning."[152] Turing felt that "whether this is possible in principle or not is a stimulating and exciting question, suggested by some of these recent developments but it did not seem that the machines constructed or projected at the time had this property."[152]"

Turing asked who could be "certain that original work that he has done was not simply the growth of the seed planted in him by teaching, or the effect of following well-known general principles."[152] "The objection says that a machine can never take us by surprise."[152]" He found that "machines take me by surprise with great frequency. This is largely because I do not do sufficient calculation to decide what to expect them to do, or rather because, although I do a calculation, I do it in a hurried, slipshod fashion, taking risks."[152]

Reminstant of Godel's theorem Turing again cautioned "the view that machines cannot give rise to surprises is due, I believe, to a fallacy to which philosophers and mathematicians are particularly subject. This is the assumption that as soon as a fact is presented to a mind all consequences of that fact spring into the mind simultaneously with it. It is a very useful assumption under many circumstances, but one too easily forgets that it is false. A natural consequence of doing so is that one then assumes that there is no virtue in the mere working out of consequences from data and general principles."[152]

Shimon Edelman put it best saying "Turing's legacy for the cognitive and brain sciences can therefore be summarized by observing that, just as nothing in biology makes sense except in the light of evolution, as Dobzhansky famously remarked, nothing about the mind/brain makes sense except in the light of computation".[84] Peter Denning suggests "computing is no longer a science of the artificial. It is a science of natural information processes. The remarkable shift to this realization occurred only in the last decade. Computing is mature enough to be described in terms of its fundamental principles. The principles reveal computing's deep structure and how it applies in many fields. They reveal common aspects of technology and create opportunities for innovation. They open entirely new ways to stimulate the excitement and curiosity of young people about the world of computing. In the 1940s, computation was seen as a tool for solving equations, cracking codes, analyzing data, and managing business processes. By the 1980s, computation had advanced to become a new method in science, joining the traditional theory and experiment. During the 1990s, computation advanced even further as people in many fields discovered they were dealing with information processes buried in their deep structures – for example, quantum waves in physics, DNA in biology, brain patterns in cognitive science, information flows in economic systems. Computation has entered everyday life with new ways to solve problems, new forms of art, music, motion pictures, and commerce, new approaches to learning, and even new slang expressions."[35]

In 1911 Stephane Leduc laid the foundation for synthetic and artificial biology when he cautioned "the synthesis of life, should it ever occur, will not be the sensational discovery which we usually associate with the idea. If we accept the theory of evolution, then the first dawn of the synthesis of life must consist in the production of forms inter-

mediate between the inorganic and the organic world, forms which possess only some of the rudimentary attributes of life, to which other attributes will be slowly added in the course of development by the evolutionary action of the environment.“[88] This would seem to be the primary goal of complex systems and brain sciences, to discover the intermediate forms that mark the emergence of brain like objects.

Andrew Coward, a systems engineer who's background includes the development and maintenance of large scale real-time electronic telecommunications networks, has suggested that the “practical needs of very complex learning systems include resource limitations and learning without interference with prior learning“ and that what is required is to build a “map between the performance of cognitive tasks and the processes at anatomical, physiological and chemical levels that implement the tasks.” Coward claims this can be a way to “understand the performance of cognitive tasks in terms of brain anatomy, physiology and chemistry using techniques analogous with those used in computer science to relate system features to transistor operations“.[31]Further Andrew Coward has laid out a very through outline for understanding the function of brain anatomy in term of a Recommendation System, he describes “the tasks required of a complex, dynamic system include performing many different behaviours, behavior recommendation and selection, behaviour priority and sequence management, detecting and defining many different conditions, heuristically defining most of the conditions, limiting the resources required, condition resource management, and information flow management” [31].

Often called the Common Cortical Algorithm hypothesis there is the idea that a lot of what we consider human intelligence can be explained by a single learning algorithm. In particular most perception (input processing) in the brain may be due to one

learning algorithm. Experiments have shown that animals that have visual inputs wired to either their auditory cortex or somatosensory cotex can learn to see.[106]

Demis Hassabis co-founder of DeepMind, the AI start up Google acquired for a half a billion dollars is quoted in 2010 as saying "If you last looked seriously at neuroscience circa 2005 - you are out of date." Many people have never heard of DARPA, the Defense Advanced Research Projects Agency, the organization responsible for among other things the Internet. (not to be confused with the world wide web invented at CERN) Even fewer people have heard of IARPA, Intelligence Advanced Research Projects Activity who has projects that include "a new generation of machine learning algorithms with human-like performance characteristics by using cortical computing primitives as their basis of operation"

Manager of cognitive computing for IBM Research, Dharmendra S. Modha, commented: "neuroanatomists have not found a hopelessly tangled, arbitrarily connected network, completely idiosyncratic to the brain of each individual, but instead a great deal of repeating structure within an individual brain and a great deal of homology across species The astonishing natural reconfigurability gives hope that the core algorithms of neurocomputation are independent of the specific sensory or motor modalities and that much of the observed variation in cortical structure across areas represents a refinement of a canonical circuit; it is indeed this canonical circuit we wish to reverse engineer."

Bruno Olshausen, who together with David Fields started the modern field of sparse modeling in neuroscience agrees "thats where were going to start to learn about the tricks that biology uses. I think the key is that biology is hiding secrets well," "We

just dont have the right tools to grasp the complexity of whats going on."[129]

"Invariably the explanatory metaphors of a given era incorporate the devices and spectacles of the day and in perhaps subtler ways they may reflect the propellant social forms and daily texture of life. Theorizing about brain and mind has been especially susceptible to sporadic reformulation in terms of the technological experience of the day. For example, the water technology of antiquity (fountains, pumps, water clocks) underlies the Greek pneumatic concept of the soul (pneuma) and the Roman physician Galen's theory of the four humours; the clockwork mechanisms proliferating during the Enlightenment are ticking with seminal influence inside Le Mettrie's L'Homme machine; Victorian pressurized steam engines and hydraulic machines are churning beneath Freud's hydraulic construction of the unconscious and it's libidinal economy; the arrival of the telegraph network provided Helmholtz with his basic moral metaphor, as did reverberating relay circuits and solenoid for hemispheric memory and so on."[34]

1.2 NEURAL NETWORKS

"The mind can be understood in terms of the brain 'The Astonishing Hypothesis' is that 'You,' your joys and your sorrows, your memories and your ambitions, your sense of personal identity and free will, are in fact no more than the behavior of a vast assembly of nerve cells and their associated molecules." - Francis Crick [33]

Neural networks as models of the human brain can be traced back to Ramon y Cajal's "neuron doctrine", Golgi staining, and subsequent 1906 Nobel prize for this work [165], and even further back to Alfred Smee's work in 1849 [146].

In 1943 Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, developed the first generation of neural networks as simplified models of nervous function [105]. Biological neurons are incredibly complex. To fully describe all of the detailed chemistry and molecular operations of even just a single cell is not feasible by even the world's fastest computers.

Neural networks are an attempt to model the information[1] processing properties of a neuron using mathematics. The first generation of neural models were networks of switches with an all-or-nothing on/off characteristic. In other words, the neuron was a two-state device. The mathematics of switching and network theory, popular in the early 20th century, seemed natural tools to model the nervous system. These early networks of switches were able to reproduce simple logical functions, and it was thought that the brain might be a system for implementing logic in switches, similar to the early telephone networks. With the advent of the theory of computation due to Alan Turing, a new science emerged and possibility that the brain could be modeled as a universal digital computer.

The second generation of neural networks describes models with an analog or continuous output response. This allows for more accurate modeling, as well as providing greater utility in computer applications that rely on neural networks for there operation. Most of the neural networks popular in research today are of this second generation. The most important development in second generation neural networks

[1]Given a learning system L trained on a dataset D with error or energy level E the information content of a signal d is proportional to the change in error or energy when L is trained on $D+d$. Or alternatively: Given a learning system L, trained on a dataset D that results in parameters P, the information content of a signal d is proportional to the change in parameters P when L is trained on $D+d$. $H(d) = L(D) - L(D+d)$

was the discovery of the backpropagation algorithm. Backpropagation is a technique used to adjust the parameters, otherwise known as weights, in a neural network model. Essentially a form of the chain rule, back propagation modifies the weights in a neural network until a particular loss function is minimized. For example, we will consider the case of supervised learning in a three-layer neural network. The three layers consist of an input layer, a hidden layer, and an output layer. The input layer represents the pixel intensities of a two-dimensional gray-scale image. The hidden layer learns the features that separate the input vectors and the output layer provides the labels.

The third generation of neural networks describes mathematical models in which time plays an active element in the operation and function in the network. This idea was popularized by John Hopfield in the 1980s, who applied statistical models from the physics of spin glasses and Ising models to the problem of mathematical neurons connected in a network. This paved the way for the interpretation of neural networks as a form of natural computing, and that they might ultimately be implemented in non-biological substrates. Neural models that can be implemented in hardware allow for direct emulation rather than digital simulation. There is continuing need for high speed, low power, electronic sensors and computers, and third generation analog neural networks in hardware present a possible solution.

Machine learning is a branch of computer science that deals with algorithms that can adjust their own parameters based on the data given to the algorithm. The classic example is Arthur L. Samuel's 1959 "Studies in Machine Learning Using the Game of Checkers" where "Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program."[137] We find that these ideas are

not uncommon, a patent by Putzrath in 1961 reads “the present invention relates to information processing apparatus, and more particularly to electrical apparatus for recognizing patterns, such as speech patterns, by simulated neural processes.” [122] A few years later Ed Feigenbaum would pronounce “The thing we call ’A.I.’ - computers doing intelligent things - is the manifest destiny of Computer Science.“ [50]

Deep learning is based on two big ideas, learning features from unlabeled data and learning multiple layers of representation. [110] Despite many decades of research, a broad theoretical approach to the brain remains elusive. Behavioral and physiological approaches have often been stymied by the complexity of the living brain. Theoretical and computational approaches, while offering greater experimental flexibility, may be too artificial in that they do not generally take into consideration the constraints and demands of a real agent acting in a physical environment.

“One way of setting about our task of building a thinking machine would be to take a man as a whole and to try to replace all the parts of him by machinery. He would include television cameras, microphones, loudspeakers, wheels and handling servo-mechanisms as well as some sort of electronic brain. This would of course be a tremendous undertaking. The object if produced by present techniques would be of immense size, even if the brain part were stationary and controlled the body from a distance. In order that the machine should have a chance of finding things out for itself it should be allowed to roam the countryside...” [151]
Embedding the models from computational neuroscience into robotic agents that will sense, act and learn in a real-world environmeng using simple, low-cost, wireless robotic devices (rovers), will allow the construction of a unified model of perception and action based on neurologically inspired machine-learning networks.

Remotely Operated Vehicle for Education and Research (R.O.V.E.R.) consisting of a color video camera and a microphone, with the ability to move via tread motion. Each rover is independently controlled wirelessly by a devoted brain, consisting of an artificial neural network housed on a separate computer (local cloud). These computers both control the rover and receive and record the resulting perceptual feedback. The rovers behave within a physical environment and are subjected to reinforcement protocols (i.e. reward and punishment positive and negative feedback in response to a behavioral outcome) similar to those employed in behavioral neuroscience. Learning of both perceptual features as well as action selection is based on neurally inspired machine learning architectures applied over this perception/action/feedback data stream. The rovers are tested for their ability to learn relatively simple behaviors, such as obstacle avoidance, future work will explore more complex goals such as social interaction between rovers with the overarching goal of developing a unified approach to embedded biological intelligence.

Recent advances in a class of machine learning derived from biological neural systems (deep learning networks) have led to remarkable progress in previously unsolved problems in neuroscience, computer vision and robotics. More recent research has successfully implemented these techniques in reinforcement learning (RL) contexts as well, for example artificial-intelligence (AI) based on machine learning video game play. These development results suggest that machine learning may provide a unified theoretical framework for understanding biological neural systems. However, previous research efforts in these areas has taken place independently in different disciplines, with little attempt to integrate them. The proposed research therefore aims to combine the latest developments in mathematical learning theory and ex-

perimental neuroscience to create a testbed for further understanding neurally based complex behaviors.developing and testing theories of embedded biological intelligence. This project aims to establish FAU as a leader in emerging artificial intelligence (AI) technologies and computational neuroscience and build research connections between departments, colleges, and campuses. Current and future areas of multidisciplinary research include autonomous vehicles, environmental monitoring, visual and auditory prosthesis, automated image analysis, and computational medicine.

"The organisation of a machine into a universal machine would be most impressive if the arrangements of interference involve very few inputs. The training of the human child depends largely on a system of refiards and punishments, and this suggests that it ought to be possible to carry through the organising with only two interfering inputs, one for pleasure or reward (R) and the other for pain or punishment (P)." [151]

To demonstrate that a machine-learning architecture can develop purposeful behavior sequences within a real-world environment. The primary goal of this proposal is to develop and test a novel research paradigm: a robotic 'model organism' with a flexible, biologically motivated neural architecture whose behavior is determined by a reinforcement learning protocol. The goal is to demonstrate that basic cognitive functions, such as navigation and reward goal seeking, may be realized in these robots using a unified machine-learning architecture, validating both the overall research approach and as well as the hypothesis that machine learning provides a broad unified theoretical framework for understanding biological intelligence.

To develop complex behavioral patterns including delayed reward such as hierarchical

goal-seeking, and social interactions and communication.. A second stage of this research will assess the robots ability to engage in social more complex behaviors such as generating sub-goals in service of larger goals as well as social behavior such as modeling the behaviors actions of other robots and competing and/or cooperating with them. The rovers are equipped with microphones and light, allowing for active communications among them.

To assess differences across neural architectures, including pathology.pathology. Once the basic proof-of-concept has been established, this paradigm may be used to compare theoretical behavioral models (e.g. architectures with varying levels of computational power), and to model potential dysfunctions underlying disorders, such as memory and learning deficits by disrupting the relevant simulated neural mechanisms and observing the resulting neural and behavioral consequences.
"A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine." Alan Turing 1948

Turing wrote that the "idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer."[152] Turing claims that the "analogy with the human brain is used as a guiding principle." [151] Thomas Insel of NIMH has cautioned that "there is a sense from many places that whoever figures out how the brain computes will come up with the next generation of computers." Bruno Olshausen has observed that "if you could solve these problems, its going to open up a vast, vast potential of commercial value."

Turing wished to investigate other types of unorganised machine, and he envisaged

the procedurenowadays used extensively by connectionistsof simulating a neural network and its training regimen using an ordinary digital computer (just as an engineer may use a computer to simulate an aircraft wing or a weather analyst to simulate a storm system). [153] The crowd sourced Internet encyclopedia defines 'Connectionism' is "a set of approaches in the fields of artificial intelligence, cognitive psychology, cognitive science, neuroscience, and philosophy of mind, that models mental or behavioral phenomena as the emergent processes of interconnected networks of simple units." However Jack Copeland has pointed out that "it is not widely realized that Turing wrote a blueprint for much of the connectionist project as early as 1948."[153] Turing knew that the "potentialities of human intelligence can only be realised if suitable education is provided."[151] Turing himself suggested that "the training process renders certain neural pathways effective and others ineffective."[153] I.J. Good later explained how "the machine will be able to learn from experience, by means of positive and negative reinforcement, and the instruction of the machine will resemble that of a child." [62] Copeland suggests that "from a historical point of view, Turing's idea that an initially unorganized neural network can be organized by means of interfering training is of considerable significance, since it did not appear in the earlier work of McCulloch and Pitts."[153] Copeland reports that "so far as is known, [Turing] was the first person to consider building computing machines out of trainable networks of randomly arranged neuron-like elements."[153]

1.3 RANDOM PROJECTIONS

"Compressed sensing and sparse methods have played an important role in the medical imaging field, including image reconstruction, image enhancement, image segmentation, anomaly detection, disease classification, and image database retrieval."[49]

The CS theory builds upon the fundamental fact that many signals can be represented using only few (sparse), linearly combined, elements of a suitable basis or dictionary.[119] Sparsity is the crucial property in the CS framework, as without sparse representation in the higher dimensional space, the lower dimension random projections are not sufficient for effective reconstruction.[119] The theory of compressed sensing originates from results in the field of high-dimensional statistics.[119] The oldest of these algorithms derives from combinatorial group testing during WWII. In these problems we suppose that there are n total items and k anomalous elements that we are seeking.[47] By a sparse representation, we mean that for a signal of length n, we can represent it with k less than n nonzero coefficients; by a compressible representation, we mean that the signal is well-approximated by a signal with only k nonzero coefficients.[47] Sparse approximation "forms the foundation of transform coding schemes that exploit signal sparsity and compressibility, including the JPEG, MPEG, and MP3 standards."[47] According to Candes the "crucial observation is that one can design efficient sensing or sampling protocols that capture the useful information content embedded in a sparse signal and condense it into a small amount of data." [24] Nonlinear optimization algorithms "lead to recovery of signals from very few measurements, significantly fewer than required by the Shannon-Nyquist sampling theorem."[119] These results "can be of material value for multiple facets of neuroscience research, particularly for neuronal data analysis, fluorescence microscopy, gene-expression analysis, and connectomics."[119] The "groundbreaking contribution" of compressed sensing is that a "simple, linear measurement process" can allow for a compressed encoding and efficient decoding.[119] A neural network can be considered "as the projection from one brain area to another via a convergent axonal pathway."[119] While it might seem that a recovery task would be "impossible because there is no way to reconstruct a signal during the times/places that the

signal is not measured,"[25] CS has however shown that this "seemingly impossible optimization program (subset selection) can be solved using a tractable amount of computation."[41]

Signals that are compressible and sparse can be "represented with high fidelity by preserving only the values and locations of the largest coefficients of the signal." [47] These protocols are nonadaptive and simply require correlating the signal with a small number of fixed waveforms that are incoherent with the sparsifying basis. [24] Incoherency means that any column of the sensing matrix has dense (opposite of sparse) representation in the matrix (i.e write one column as a linear combination of other columns and the coefficients will be dense) [119] Candes describes incoherence as a phenomena that "extends the duality between time and frequency; just as a Dirac (spike) in the time domain is spread out in the frequency domain, incoherence expresses the idea that objects having a sparse representation must be spread out in the domain in which they are acquired."[24] When considering a neural network model "the basis set can be represented by the activity of cells that exhibit certain properties, regarding, e.g., their receptive fields, such as mammalian visual cortex cells , or their spatial firing patterns, such as grid cells." [119] Thus we see have that the "representation of a signal is actually 'summarized' by the encoded, compressed version through a measurement/sensing procedure."[119] We can now answer "whether neural circuits are capable of implementing L1 minimization"[119], we can see that LCA is a simple model for how this behavior might occur in biological neural networks.

Aaronson describes an attempt as to how one might simulate a brain, i.e. pass a Turing Test, "maybe we do know all the inputs well ever need, but we just cant write them

in a big enough table, so we write them down in this compressed form." [1] The major idea of sparse modeling is that the information rate of a signal may be smaller than suggested by traditional signal processing assumptions.[24] For many discrete-time signals the number of degrees of freedom is much smaller than its signal length. [24] Around 2004, Emmanuel Cands, Terence Tao, and David Donoho showed that sparse signals may be "reconstructed with even fewer samples than the sampling theorem requires."[25] Even though compressed sensing is relatively new topic "thousands of papers have appeared in this area, and hundreds of conferences, workshops, and special sessions have been dedicated to this growing research field." [47] A typical task in signal processing is to reconstruct a signal from a series of sample measurements.[25] Random sampling protocols allow a sensor to "very efficiently capture the information in a sparse signal without trying to comprehend that signal." [24] Baraniuk reports "that random projections have recently emerged as a surprisingly useful tool in signal processing."[12] Work on Random Projections (RP) has led to a "powerful, yet extremely simple methodology for dealing with the curse of dimensionality."[119] Random projections are a universal in that this "encoding process can proceed without knowledge of the structure that makes the signal compressible."[12] According to Donoho, we are enabled to "sample smarter not faster; we can replace front-end acquisition complexity with back-end computing".[41]

Elder reports that although there are extraordinary advances in computer hardware, "the acquisition and processing of signals in application areas such as imaging, video, medical imaging, remote surveillance, spectroscopy, and genomic data analysis continue to pose a tremendous challenge."[47] As a typical case " we might wish to identify defective products in an industrial setting, or identify a subset of diseased tissue samples in a medical context."[47] However we find that "in many important

and emerging applications, the Nyquist rate is so high that we end up with far too many samples."[42] The key point being that the sampling theorem results provides a sufficient condition not a necessary one. [25] Baraniuk claims the "key revelation is that the relevant structure in a signal can be preserved when that signal is projected onto a small number of random basis functions."[12] Now the problem is the "design a collection of tests that allow us to identify the support (and possibly the values of the nonzeros) of x while also minimizing the number of tests performed."[47]

"Why Random Projections?" asks Durrant, who reflects that random projections are, "linear, cheap, and universal. Target dimension does not depend on data dimensionality for Johnson Lindenstrauss Lemma (JLL), and JLL works with high probability for any fixed finite point set. The technique is oblivious to data distribution and tractable to analysis."[43] We do find that while "information is lost through such a projection, that information tends to be incoherent with the relevant structure in the signal."[12] These new results "dramatically reduces the number of measurements needed for efficient reconstruction, compared with the ones indicated by the Shannon-Nyquist sampling theorem."[119] We find many diverse motivations for RP in the literature including "avoid the collection of lots of data" and of primary interest here in the "theory of cognitive learning (RP Perceptron)".[43] In the case of biological neural network connections "multiplication can be thought of as the influence of one region to another...this could represent projection from the cortex...in this case y would be the activity (firing rates) of a subset of M neurons.[119] Of primary concern to psychologists is the question of "how does the brain learn concepts from a handful of examples when each example contains many features?" [119] Random projections preserve "the concept" and in the new "low-dimensional space, the number of examples and time required to learn concepts are comparatively small."[43] Neurons

can form a basis set "if their (appropriate) combination can generate any signal f (activity pattern of neurons) in the cortex."[119]

There exists an intimate linkage between the CS theory and the JLL.[11] "With high probability", it can be shown that "a random projection of a sparse, high-dimensional signal vector onto a lower-dimensional space contains enough information to enable signal reconstruction with small or zero error."[11] Thus the JLL "shows that with high probability the geometry of a point cloud is not disturbed by certain mappings onto a space of dimension logarithmic in the number of points." [11] The statement and proofs of the JLL have been simplified considerably by using random linear projections and concentration inequalities. From JLL we have "high-probability guarantees that for a suitably large k, independently of the data dimension, random projection approximately preserves data geometry of a finite point set. In particular norms and dot products approximately preserved with high probability."[43]

The pixels of a CT scan or of an 'Ansel Adams' photograph can be represented as vector f, with N-dimensions and gray scale intensity.[119] In both image examples we find that in practice with very high probability "very few coefficients are needed to represent the image via a wavelet basis set, thus the x vector is sparse."[119] Typically digital cameras would need take a measurement for each of the N dimensions or pixels, in modern consumer digital cameras that are sensitive in the visible spectrum this is accomplished by using an array of semiconductor detectors. This technology has ridden the wave of Moore's law and thus we have low cost camera pixel arrays (cell phone camera hardware about $ 1), for light outside the visible spectrum new methods are needed as high resolutions detectors are not possible. Different detector that might be used "include a photomultiplier tube or an avalanche photodiode for low-light

(photon-limited) imaging (more on this below), a sandwich of several photodiodes sensitive to different light wavelengths for multimodal sensing, and a spectrometer for hyperspectral imaging."[42] Many have now suggested we "work with random projections of the data."[43]

New types of camera architectures have been proposed in which "rather than measuring pixel samples of the scene under view, we measure inner products between the scene and a set of test functions."[42] This new imaging architecture is based on "a digital micro-mirror device (DMD) with the new mathematical theory and algorithms of compressive sampling (CS)."[42] The DMD single-pixel camera is an Optical Computer (OC) that "measures the inner products between an N-pixel sampled version x of the incident light-field from the scene." [119] A DMD, consisting of an array of N tiny mirrors, catches the light of a scene; "the reflected light is then collected and focused onto a single photon detector (the single pixel) that integrates the product to compute the measurement as its output voltage, the voltage across the photo detector is then sent to analog to digital converter.[42] To compute CS randomized measurements, we set the mirror orientations randomly using a pseudo-random number generator, measure, and then repeat the process M times to obtain the measurement vector y." [42] Random test functions play a key role as each measurement is a random sum of pixel values taken across the entire image, and in this manner sub-Nyquist image acquisition is achieved. [42]

Eldar reminds us that a "popular techniques for signal compression is known as transform coding", and this technique involves "finding a basis or frame that provides sparse or compressible representations for signals in a class of interest."[47] In Magnetic Resonance Image (MRI) reconstruction, "sparsity in transformed space

such as wavelet has been successfully used to speed up scanning time and improve reconstruction quality."[49] Consider sparsity in the case of a Fourier basis set, "sparsity implies that the majority of the energy of signal f is contained in a few frequency components."[119] Duarte discusses how we can combine "sampling and compression into a single non-adaptive linear measurement process."[42] Donoho, Romberg, Candes, Tao, Baraniuk and others work has shown that for an "efficient and reversible encoding process" matrices "must fulfill three very important conditions termed sparsity, incoherency, and isometry."[119]

Candes describes Compressed Sensing as, "a way to use numerical optimization to reconstruct the full-length signal from a small amount of collected data." [24] The goal of single-pixel camera design is that it "reduces the required size, complexity, and cost of the photon detector array down to a single unit, which enables the use of exotic detectors."[42] The Johnson-Lindenstrauss Lemma demonstrates that random projections preserve image structure by embedding points with minimal disruption of their pair-wise distances.[12] When the original image is projected from its N-dimensional space to an M-dimensional space, it has a new compressed version y.[119] Candes describes Compressed Sensing as "a very simple and efficient signal acquisition protocol, which samples in a signal independent fashion at a low rate and later uses computational power for reconstruction from what appears to be an incomplete set of measurements."[24] When the image is "compressible by an algorithm like JPEG, the CS theory enables us to stably reconstruct an image of the scene from fewer measurements than the number of reconstructed pixels."[42]

With "prior knowledge or assumptions about the signal", namely that the signal of interest is sparse in some basis, it turns out to be "possible to perfectly reconstruct

a signal from a series of measurements."[25] Random sampling has been shown to be a good approach for most known basis for example sines, cosines, wavelets, curvelets, etc.[95]Low dimensional RPs form "an efficient encoding that can be used as a compressed representation of the original data; high dimensional patterns can then be recovered by appropriate decoding processes."[119] Compressed sensing theory then tells us that "data with high dimensionality is represented by sparse components of a suitable basis set, it is possible to reconstruct them by their RPs." [119] Richard Baraniuk and others have suggested that random projections "provide dimensionality reduction, which can significantly simplify certain computations."[12]

Derived from " the way the brain processes information, neuroscience, neural network, and dynamical systems communities have been proposing novel computational concepts. These concepts are fundamentally different from the standard Turing or von Neumann Machine methods, which are widely implemented in most computational systems."[85] Several new algorithms, though discovered independently, share common features and carry "many ideas towards a new computational paradigm of neural networks."[118] Turing himself had begun to "investigate other types of unorganized machines, and also to try out organizing methods that would be more nearly analogous to 'methods of education'." [151] Reservoir computing (RC) "recently appeared as a generic name for designing a new research stream including mainly echo state networks (ESNs), liquid state machines (LSMs), and a few other models like back-propagation decorrelation (BPDC)." [118] The key component of a RC is "a large, distributed, nonlinear recurrent network, the so-called reservoir, with trainable output connections, devoted to reading out the internal states induced in the reservoir by input patterns."[118]

The main advantage of RC is to use a "fixed randomly connected network as reservoir", without the need of training. [118] One of these concepts is known as Echo State Network, Liquid State Machine or more generally as Reservoir Computing (RC). Larger has described RC as being based on "the computational power of complex recurrent networks operating in a dynamical and transient-like fashion." Further Larger comments on how "RC benefits from the advantages of recurrent neural networks, while at the same time avoiding the problems in the training procedure."[85] Paugam reports that Reservoir Computing is a "family of versatile basic computational metaphors with a clear biological footing.[118] Reservoir Computing is the idea "do not adapt the internal connection weights, initialize them randomly" and only "train the output directly, using a specific classifier or regression method."[107] In most RC "models linear regression or recursive least mean squares, are applied to readout neurons only."[118] When the activation function f_a out is a linear classifier the Reservoir Computer is called an Echo State Network.[107] Liquid State Machines LSM are "rather similar to the ESN, except for the fact that the function f_{in} is usually a spiking neural network or a network of threshold logic gates."[107]

The Single-Layer Feedforward Neural Networks architecture known as Extreme Learning Machine (ELM) was proposed by Huang et al.[107] Given that it is feed forward, unlike the Reservoir Computing algorithms, there is "no recurrence in the neural network of ELM-based techniques."[107] The central notion of the ELM is the random nature of the network weights.[107] The only parameter is the number of neurons in the hidden layer and even then "one does not have to know beforehand the exact best number of neurons required for ELM to perform well."[107] The output weights "can be computed from the hidden layer output matrix H and target values by using a Moore-Penrose generalized inverse of H"[107] It has be shown that "computational

time is minimal while adding new random neurons to the ELM."[107]

Bruno Olshausen has suggested that if we could figure out how biology naturally deals with noisy computing elements, it would lead to a completely different model of computation. [115] While Geoffrey Hinton reminds us that "the brain is confronted by a buzzing, blooming confusion. It needs to fit many different models and use the wisdom of crowds." [87] Andrew Coward has strong evidence that "unambiguous behavioural meanings for condition detections in a complex learning system" are not practical, because they can cause "major interference between new learning and prior learning." [31] Further Andrew Coward has described how "partially ambiguous behavioural meanings require more information handling resources, but makes it feasible to limit interference between new and prior learning." [31] Andrew claims this is would also be useful to an evolutionary process in that "if information processes are recommendations, there is a lower probability that a mutation will have fatal consequences, because any behaviour must be supported by recommendations from multiple sources." [31]

Biological systems and modern computing hardware systems are both constrained by size, power, and reliability, thus "unconventional computing methods that directly address these issues are of increasing interest."[3] There is a clear "need to better understand, and perhaps exploit, probability in computation."[3] Stochastic computing was proposed "as a low-cost alternative to conventional binary computing."[3] Stochastic computers store and manipulate "information in the form of digitized probabilities." [3] Stochastic computers represents "continuous values by streams of random bits."[3] Complex computations are performed with "simple bit-wise operations on the streams."[3] "Stochastic computing is distinct from the study of randomized

algorithms."[3]

A basic feature of Stochastic Computing numbers themselves are interpreted as probabilities in that numbers are represented by streams of bits. The bits can be streams in time on a single wire or multiple wires grouped in space, as with the Bundle Computer or both in the case of the Ergodic Computer) that can be "processed by very simple circuits"[3] Stochastic computers are fail-soft in that the probabilistic nature of the elements makes them insensitive to the particular value of any unit, thus providing utility "under both normal and faulty conditions." [3] "Bit-streams of this type and the probabilities they represent are known as stochastic numbers. For example, the probability of observing a 1 at an arbitrary bit position in a bit-stream S containing 25% 1s and 75% 0s is $p = 0.25$. Neither the length nor the structure of S need be fixed, and the positions can, in principle, be chosen randomly."[3] Stochastic computing enables very low-cost implementations of arithmetic operations using standard logic elements. Multiplication can be performed in a stochastic circuit by a single AND gate, "consider two binary bit-streams that are combined with logical AND. If the probabilities of seeing a 1 on the input bit-streams are p1 and p2, then the probability of 1 at the output of the AND gate is p1 p2, assuming that the two bit-streams are suitably uncorrelated or independent."[3]

In the following example, the probability of seeing a 1 in each of the signals is $\langle p \rangle = 0.25$.

- $\langle (0, 0, 1, 0) \rangle = 0.25$
- $\langle (1, 0, 0, 0) \rangle = 0.25$
- $\langle (0, 1, 0, 0, 0, 0, 1, 0) \rangle = 0.25$

- $\langle(1,1,0,0,0,0,0,0)\rangle = 0.25$

```
function Stochastic_Computation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Stochastic Computing Multiplier
clear all
clc
close all


p=0.5


q=0.125


N=10^6;


sp = p > rand(1,N);
sq = q > rand(1,N);


sum(sp)/length(sp)


mean(sp)


sum(sq)/length(sq)


mean(sq)
```

```
disp('Multiplier')

spq=and(sp,sq);

spq1=sp.*sq;

sum(spq)/length(spq)

sum(spq1)/length(spq1)

p*q



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Stochastic Computing Adder (a+b)/2
% This SC adder is just a 2-way multiplexer
% with a random bit-stream r of probability value 1/2 applied to its
% control (select) input.

r=rand(size(sq))>0.5;

spq=sp;

spq(r)=sq(r);

sum(spq)/length(spq);
```

```
(p+q)/2;

2*(sum(spq)/length(spq))

p+q

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Stochastic Computing Subtraction

r=rand(size(sq))>0.5;

spq=sp;

spq(r)=~sq(r);

% sum(spq)/length(spq);

% (p+q)/2;

2*(sum(spq)/length(spq))-1

p-q

end
```

1.4 CONTRIBUTIONS

- Blackout / Feedback Amplifier minimizes Total Variation / Gradient Sparsity of receptive fields
- Simulated Autonomous Land Vehicle in a Neural Network Dataset
- ALVINN Remotely Operated Vehicle for Education and Research
- Sparse Filtering Video
- Sparse Filtering with Locally Competitive Algorithms
- LCA Implementation
- Dictionary Learning Locally Competitive Algorithms (DLCA)
- Iterative Block Thresholding Algorithms
- Iterative Cubic Thresholding Algorithms
- LCA Compressed Sensing

We propose a new model framework for the study of brain science: A theoretical foundation for the information processing and computational primitives inherent to neurons. We utilize a Hopfield model combined with Hebbian learning. We describe how sparse modeling is accomplished by neurons to model and organize data. We then show how this same model LCA can be used to recover under-sampled data vectors in a process know as compressed sensing.

Locally competitive algorithms (LCA) represent a class of nonlinear dynamic neural networks that achieve sparse modeling via leaky integrators interacting through local, nonlinear competition between units. LCA can be implemented in analog hardware and offer an ultra-low power physical solution to achieving L1-constrained least squares optimization. According to Los Alamos National Lab's Neuroscience Researcher Garrett Kenyon LCA "captures a good chunk of the computer vision and theoretical neuroscience being done in the last decade." In this work we show that in addition to being able to solve (1) sparse approximation problems, LCA is also able to (2) learn a dictionary and (3) recover undersampled signals known as compressed sensing.

In this novel classification work, simulated LCA networks are employed for two tasks: first, deriving class-specific sparse dictionaries and second, classification based on the ℓ_1 norm of class-specific activation coefficients. This approach is applied to the recognition of butterflies in natural imagery [86] and to detecting human actions in video [65]. The class-specific dictionary learning approach described here is conducive to learning contextual properties of images (i.e., background properties that probabilistically co-occur with objects) that can improve classification performance. The work is extended further into an embedded, mobile environment.

Sparse coding is a method of representing data based on a linear combination of a few dictionary elements [100]. In recent years, much multi-disciplinary research has been conducted on sparse models and their applications [100].

$_{z \in \mathbb{R}^K} ||Wz - x||^2 + \lambda ||z||_1$

z is the code (want)

x is the data (have)

W is adapted

Encoding requires optimization

Decoding is trivial

Compressed sensing

$$_{z\in\mathbb{R}^K}||Wz - x||^2 + \lambda||z||_1 \tag{1.1}$$

z is the data (want)

x is the code (have)

W is universal

Encoding is trivial

Decoding requires optimization

Chapter 2

Models of Neural Computation

2.1 BACK PROPAGATION IN NEURAL NETWORKS

A neural network is a feed-forward, directed acyclic graph. The back propagation procedure trains the weights of an acyclic graph with known data, and is an example of supervised learning.

Back Propagation is a special case of Automatic differentiation (AD) which "uses exact formulas along with floating-point values, instead of expression strings as in symbolic differentiation, and it involves no approximation error as in numerical differentiation using difference quotients.AD is a third alternative, also called computational differentiation or algorithmic differentiation."[109][124]

The key idea is to compute the error

$$\mathrm{Err}_i = (y_i - a_i) \tag{2.1}$$

between the known solution and the network output,

$$E \equiv \frac{1}{2} \sum_i^n \mathrm{Err}_i^2 \tag{2.2}$$

then to adjust the weights from a random initial state towards a configuration that

matches the input.

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \tag{2.3}$$

$$\Delta_i = \text{Err} \times f'(in_i) \tag{2.4}$$

$$\Delta_j = f'(in_i) \sum_i W_{j,i} \Delta_i \tag{2.5}$$

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \tag{2.6}$$

There are n output nodes. For a node i in the output layer

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial f(in_i)}{\partial W_{j,i}} \tag{2.7}$$

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i) f'(in_i)\frac{\partial in_i}{\partial W_{j,i}} \tag{2.8}$$

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i) f'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_k W_{k,i} a_j\right) \tag{2.9}$$

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i) f'(in_i) a_j = -a_j \Delta_i \tag{2.10}$$

$$\Delta_i = (y_i - a_i) f'(in_i) \tag{2.11}$$

for $f(x, y)$ with f differentiable

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial u} \tag{2.12}$$

$$\frac{\partial f}{\partial v} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial v} \tag{2.13}$$

For a node j in the hidden layer:

$$\frac{\partial E}{\partial W_{k,j}} = \frac{\partial}{\partial W_{k,j}} E(a_{j1}, a_{j2}, \ldots, a_{jm}) \tag{2.14}$$

where j_i are the indices of the nodes in the same layer as node j.

$$\frac{\partial E}{\partial W_{k,j}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial W_{k,j}} + \sum_i \frac{\partial E}{\partial a_i}\frac{\partial a_i}{W_{k,j}} \tag{2.15}$$

where $\sum_i$ runs over all other nodes i in the same layer as node j

$$\frac{\partial E}{\partial W_{k,j}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial W_{k,j}} \tag{2.16}$$

$$\frac{\partial a_i}{W_{k,j}} = 0 \quad \text{for } i \neq j \tag{2.17}$$

$$\frac{\partial E}{\partial W_{k,j}} = \frac{\partial E}{\partial a_j} f'(in_i) a_k \tag{2.18}$$

$$\frac{\partial E}{\partial a_j} = \frac{\partial}{\partial a_j} E(a_{k1}, a_{k2}, \ldots, a_{km}) \tag{2.19}$$

where k_i are the indices of the nodes in the layer after node j

$$\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{2.20}$$

where $\sum_k$ runs over all nodes k that node j connects to.

$$\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} f'(in_k) W_{j,k} \tag{2.21}$$

$$\Delta_j \equiv f'(in_j) \sum_k W_{j,k} \Delta_k \tag{2.22}$$

$$\frac{\partial E}{\partial W_{k,j}} = -\Delta_j a_k \tag{2.23}$$

Now the network is designed to classify new, unknown inputs based on its training. It is possible to tune the weights from the error due to the dependence of the outputs on the weights. The final output vector are the o_j

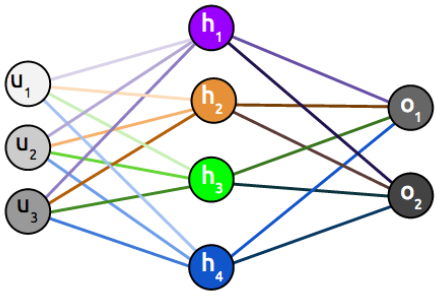
$$o_j = f(S_j) = \frac{1}{1 + e^{-S_j}} \quad (j = 1, 2, ..., n) \tag{2.24}$$

which functionally depend on an outgoing signal S_j

$$S_j = \sum_{i=1}^{m} w_{ij} h_i \qquad (j = 1, 2, ..., n), \ (i = 1, 2, ..m) \tag{2.25}$$

Each S_j is the sum of the product of the weights w_{hj} by the hidden layer output components h_i.

For example, in the following graph, $n = 2$ as there are only two components in the final output vector. This vector consists of the o_j: o_1 and o_2. The hidden layer vector consists of the h_i: h_1, h_2, h_3 and h_4. Hence $m = 4$. The shaded lines on the right represent the weights. The dark purple line at the top right connecting h_1 to o_1 is the weight w_{11}. The lines beneath it are the remaining w_{ij}. In this example, there are eight weights ($8 = 4 \times 2 = m \times n$)



To find the final output from the second node o_2, first add the inner product of $\overrightarrow{h_i}$ and the w_{i2} (lines leading to o_2). This sum is the signal S_2:

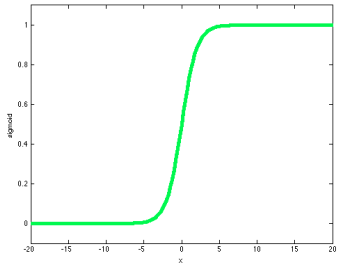
$$S_2 = \sum_{i=1}^{4} w_{i2}h_i \ = w_{12}h_1 + w_{22}h_2 + w_{32}h_3 + w_{42}h_4 \quad (j = 2),\ (i = 1, 2, 3, 4)$$

The numerical value of this signal becomes the argument of a sigmoid function. The final output vector for the second node is:

$$o_2 = \frac{1}{1 + e^{-S_2}} = \frac{1}{1 + e^{-(w_{12}h_1 + w_{22}h_2 + w_{32}h_3 + w_{42}h_4)}}$$

Two intrinsic properties of the sigmoid function are useful in neural networks and back propagation. The first important feature is that as the argument S approaches $-\infty$, the sigmoid function approaches zero, and as the argument S approaches $+\infty$, the sigmoid function approaches one.

Have a look at an example sigmoid curve:



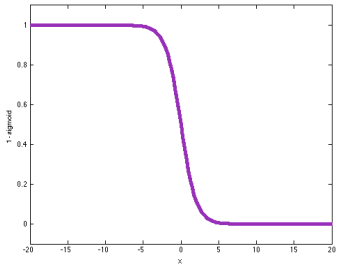
On the normalized horizontal axis is the input argument S. On the vertical axis is the sigmoid function o $= 1/(1+e^{-S})$. Almost all inputs return the value zero or one. The exceptions are the data points near $S = 0$, where the slope rises steeply through the point (0, 0.5). The signal S determines the final output answer.

Here is the final binary output for this example:

$$o = \frac{1}{1+e^{-S}} = \begin{cases} 0 & \text{if } S_2 < 0 \text{ (e.g.. “no”)} \\ 1 & \text{if } S_2 > 0 \text{ (e.g.. “yes”)} \end{cases}$$

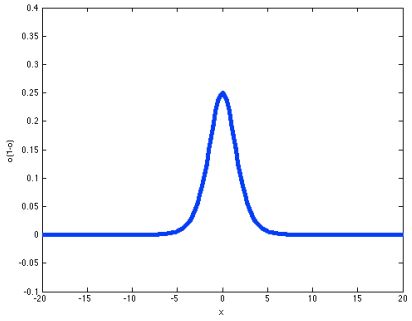
The curve is similar to a discrete step function (a function that is zero before a threshold and one after); however, the sigmoid has an advantage in the relationship between the function and its derivative. The following graph shows $1 - o$:

$$1 - o = 1 - \frac{1}{1+e^{-S}} = \frac{1+e^{-S}}{1+e^{-S}} - \frac{1}{1+e^{-S}} = \frac{1+e^{-S}-1}{1+e^{-S}} = \frac{e^{-S}}{1+e^{-S}}$$

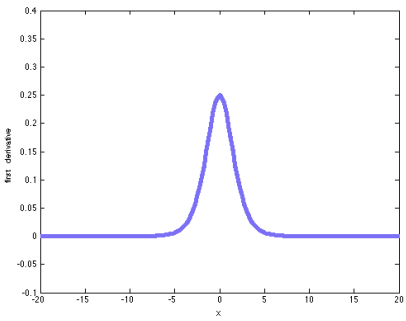


The two figures show that the derivative of the sigmoid function o is identical to the product of $o \times (1 - o)$.

(a) product: $o \times (1 - o)$



(b) derivative: $o'(S)$



This can be verified mathematically:

$$\begin{aligned} o'(S) &= \frac{\mathrm{d}}{\mathrm{d}S}(1 + e^{-S})^{-1} \\ &= -(1 + e^{-S})^{-2} \times -(e^{-S}) \\ &= \frac{e^{-S}}{(1 + e^{-S})^2} \end{aligned}$$

This expression is identical to:

$$\begin{aligned} o \times (1 - o) &= \frac{1}{1 + e^{-S}} \times \left(1 - \frac{1}{1 + e^{-S}}\right) \\ &= \frac{1}{1 + e^{-S}} \times \frac{e^{-S}}{1 + e^{-S}} \\ &= \frac{e^{-S}}{(1 + e^{-S})^2} \end{aligned}$$

The gradient of the network output is immediately known along with the information of the network output. Multiply $o \times (1 - o)$, and the gradient is exactly the same.

The goal of the network is to establish weights that reproduce known data; the difference between the network output and the true solution is the Error.[107]The expression for the error is:

$$E = \frac{1}{2}\sum_{j=1}^{n}(o_j - y_j)^2 \tag{2.26}$$

An algebraic expression can be found relating the network output to the necessary change in the weights.

The gradient of the weights are found from the derivative of the error (and the chain rule):

$$\begin{aligned} \Delta w_{ij} &= -\frac{\partial E}{\partial w_{ij}} \\ &= -\frac{\partial E}{\partial o_j} \times \frac{\partial o_j}{\partial w_{ij}} \end{aligned}$$

From the expression for the error above, find $-\frac{\partial E}{\partial o_j}$:

$$\begin{aligned} \frac{\partial E}{\partial o_j} &= \frac{\partial}{\partial o_j}\left(\frac{1}{2}\sum_{j=1}^{n}(o_j - y_j)^2\right) \\ &= (o_j - y_j) \end{aligned}$$

Apply the calculus chain rule of derivatives again to find $\frac{\partial o_j}{\partial w_{ij}}$

$$\begin{aligned}\frac{\partial o_j}{\partial w_{ij}} &= \frac{\partial o_j}{\partial S_j} \times \frac{\partial S_j}{\partial w_{ij}} \\ &= o'(S_j) \times \frac{\partial S_j}{\partial w_{ij}} \\ &= o_j \times (1 - o_j) \times h_i\end{aligned}$$

Hence, the numerical change in the weights can be calculated from the known data and products of the neural network outputs:

$$\Delta w_{ij} = -(o_j - y_j) \times o_j \times (1 - o_j) \times h_i$$

The weights are updated iteratively in back propagation, calculated from the output and error gradients, as in the formula above. Also consider the first set of weights, the lines on the left side of the visual graph, which multiply the inputs and are summed in the same way to form the argument of the sigmoid for each node of the hidden layer. Each hidden node produces

$$h_i = \frac{1}{1 + e^{-\sum w_{oi} u_o}}$$

Where the u_o are the initial inputs and the w_{oi} are the first set of weights.
A step-by-step iteration scheme for back propagation:

1. Initialize weights in both groups with random numbers

2. send known data through the network

3. compute the outputs from the hidden layer, h_i, and the final outputs, o_j:

$$h_i = \frac{1}{1 + e^{-\sum w_{oi} u_o}}$$

$$o_j = \frac{1}{1 + e^{-\sum w_{ij} h_i}}$$

4. calculate the 1st and 2nd layer weight gradients

$$\begin{aligned} \Delta w_{ij} &= o_j(1 - o_j)(o_j - y_j)h_i = \delta_j h_i \qquad \text{(second layer)} \\ \Delta w_{oi} &= h_i(1 - h_i)u_o \sum_{j=1}^{n} w_{ij}\delta_j \quad = \delta_i u_o \quad \text{(first layer)} \end{aligned}$$

5. adjust the weight vector at time "t+1" as:

$$w_{oi}^{t+1} = w_{oi}^{t} + \Delta w_{oi}$$

$$w_{ij}^{t+1} = w_{ij}^{t} + \Delta w_{ij}$$

6. Continue iteratively until obtaining the lowest possible error.

Additionally, a learning rate parameter $0 < \eta \leq 1$ may multiply the gradients:

$$w_{oi}^{t+1} = w_{oi}^{t} + \eta \Delta w_{oi}$$

$$w_{ij}^{t+1} = w_{ij}^{t} + \eta \Delta w_{ij}$$

As well as a weight momentum parameter α for learning efficiency:

$$\Delta w_{oi} = \eta \delta_i u_o + \alpha w_{oi}(\tau - 1)$$

where τ is another parameter.

$$a_i \leftarrow f(in_i) = f\left(\sum_j W_{j,j} a_j\right) \tag{2.27}$$

$$E = \frac{1}{2}\mathrm{Err}^2 \equiv \frac{1}{2}\left(y - f\left(\sum_{j=0}^{n} W_j x_j\right)\right)^2 \tag{2.28}$$

$$\frac{\partial E}{\partial W_j} = \mathrm{Err} \times \frac{\partial \mathrm{Err}}{\partial W_j} = \mathrm{Err} \times \frac{\partial}{\partial W_j}\left(y - f\left(\sum_{j=0}^{n} W_j x_j\right)\right) \tag{2.29}$$

$$\frac{\partial E}{\partial W_j} = -\mathrm{Err} \times f'\left(\sum_{j=0}^{n} W_j x_j\right) \times x_j \tag{2.30}$$

$$W_j \leftarrow W_j + \alpha \times \mathrm{Err} \times f'\left(\sum_{j=0}^{n} W_j x_j\right) \times x_j \tag{2.31}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.32}$$

$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = e^{-x} f(x)^2 \tag{2.33}$$

$$f(x) = \frac{1}{1 + e^{-x}} \Rightarrow f(x) + e^{-x} f(x) = 1 \Rightarrow e^{-x} = \frac{1 - f(x)}{f(x)} \tag{2.34}$$

$$f'(x) = \frac{1 - f(x)}{f(x)} f(x)^2 = (1 - f(x)) f(x) \tag{2.35}$$

```
function Hahn_NN_2D_Landscape2

clear all

close all
```

```
clc
tic
[X,Y,Z] = Hahn_landscapes(3);

X = X - min(X(:));
X = X / max(X(:));

Y = Y - min(Y(:));
Y = Y / max(Y(:));

Z = Z - min(Z(:));
Z = Z / max(Z(:));



figure(1)
surf(X,Y,Z,'EdgeColor','none')
view(-144,30)
set(gcf,'color','w')
title('Neural Network Function Approximation')
snapnow;



pattern=randi(size(Z,1),500,2);

testpattern=randi(size(Z,1),500,2);
```

```
category=[];

for i = 1:size(pattern,1)

    category=[category; Z(pattern(i,1),pattern(i,2))];

end



testcategory=[];

for i = 1:size(testpattern,1)

    testcategory=[testcategory; Z(testpattern(i,1),testpattern(i,2))];

end



plot(pattern(:,1),pattern(:,2),'x')
scatter3(pattern(:,1),pattern(:,2),category)
set(gcf,'color','w')
title('Neural Network Function Approximation Input')
snapnow;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Visualize Raw Data
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

r=randperm(size(pattern,1)); %Shuffle Patterns

c=500; %Use the First 400 Patterns
pattern=pattern(r(1:c),:);
category=category(r(1:c),:);

bias=ones(size(pattern,1),1); %Add Bias (Default Resting State Potential)
pattern = [pattern bias];
testpattern = [testpattern bias];

n1 = size(pattern,2);   %Set the Number of Input Nodes Equal to Number of
%Pixels in the Input image
n2 = 100;   %n2-1        %Number of Hidden Nodes (Free Parameter)
n3 = size(category,2);  %Set the Number of Output Nodes Equal to the
%Number of Distinct Categories {left,forward,right}

w1 = 0.005*(1-2*rand(n1,n2-1)); %Randomly Initialize Hidden Weights
w2 = 0.005*(1-2*rand(n2,n3));   %Randomly Initialize Output Weights

dw1 = zeros(size(w1));          %Set Initial Hidden Weight Changes to Zero
dw2 = zeros(size(w2));          %Set Initial Output Changes to Zero

L = 0.0001;          % Learning    %Avoid Overshooting Minima
M = 0.9;            % Momentum    %Smooths out the learning landscape
```

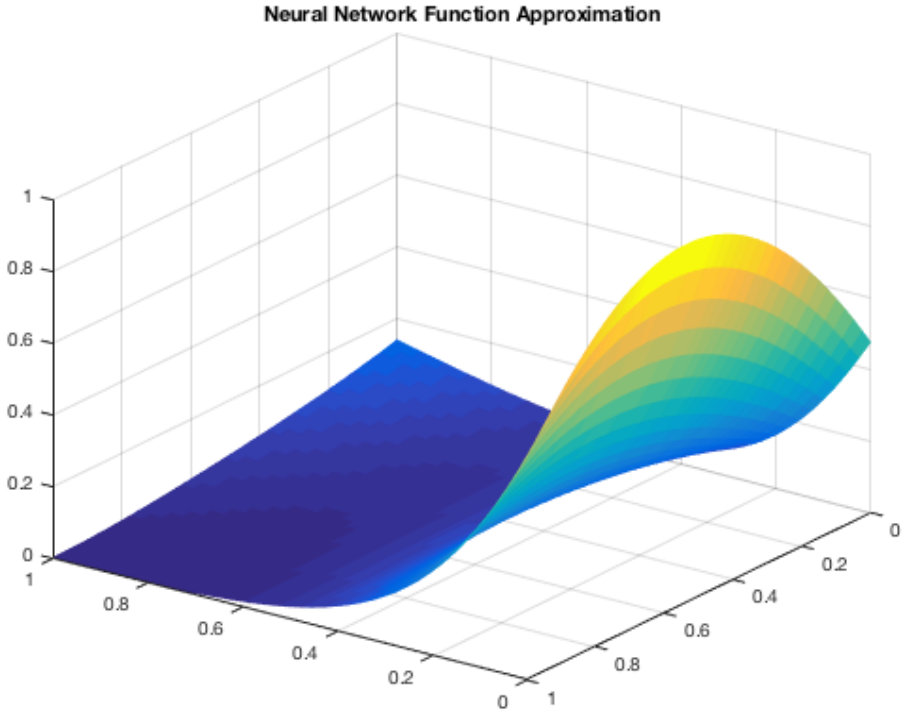
```
sse=size(pattern,1);  % Set Error Large so that Loop Starts
sseplot=[size(pattern,1) size(pattern,1) size(pattern,1)];
%Convergence Plot

for loop=1:1000

    act1 = [af(pattern * w1) bias];
    act2 = af(act1 * w2);

    error = category - act2;  %Calculate Error

    delta_w2 = error .* act2 .* (1-act2); %Backpropagate Errors
    delta_w1 = delta_w2*w2' .* act1 .* (1-act1);
    delta_w1(:,size(delta_w1,2)) = []; %Remove Bias

    dw1 = L * pattern' * delta_w1 + M * dw1;
    %Calculate Hidden Weight Changes
    dw2 = L * act1' * delta_w2 + M * dw2;
    %Calculate Output Weight Changes

    w1 = w1 + dw1; %Adjust Hidden Weights
    w2 = w2 + dw2; %Adjust Output Weights

    act12 = [af(testpattern * w1) bias];
    act22 = af(act12 * w2);
```

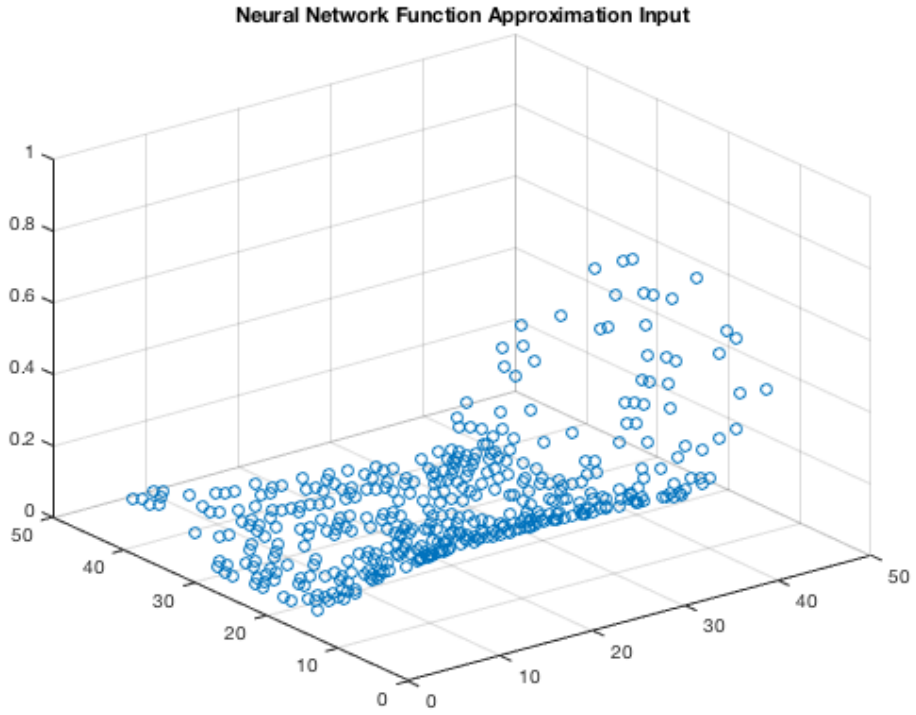
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------------------%
%Plots
%--------------------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(121)
scatter3(pattern(:,1),pattern(:,2),act2, 'bx')
view(loop/10,30)
hold on
scatter3(pattern(:,1),pattern(:,2),category, 'go')
view(loop/10,30)
hold off
title('Training')
subplot(122)
scatter3(testpattern(:,1),testpattern(:,2),act22, 'rx')
view(loop/10,30)
hold on
scatter3(testpattern(:,1),testpattern(:,2),testcategory, 'ko')
view(loop/10,30)
hold off
title('Testing')
set(gcf,'color','w')
if mod(loop,100)==0
snapnow;
end
```

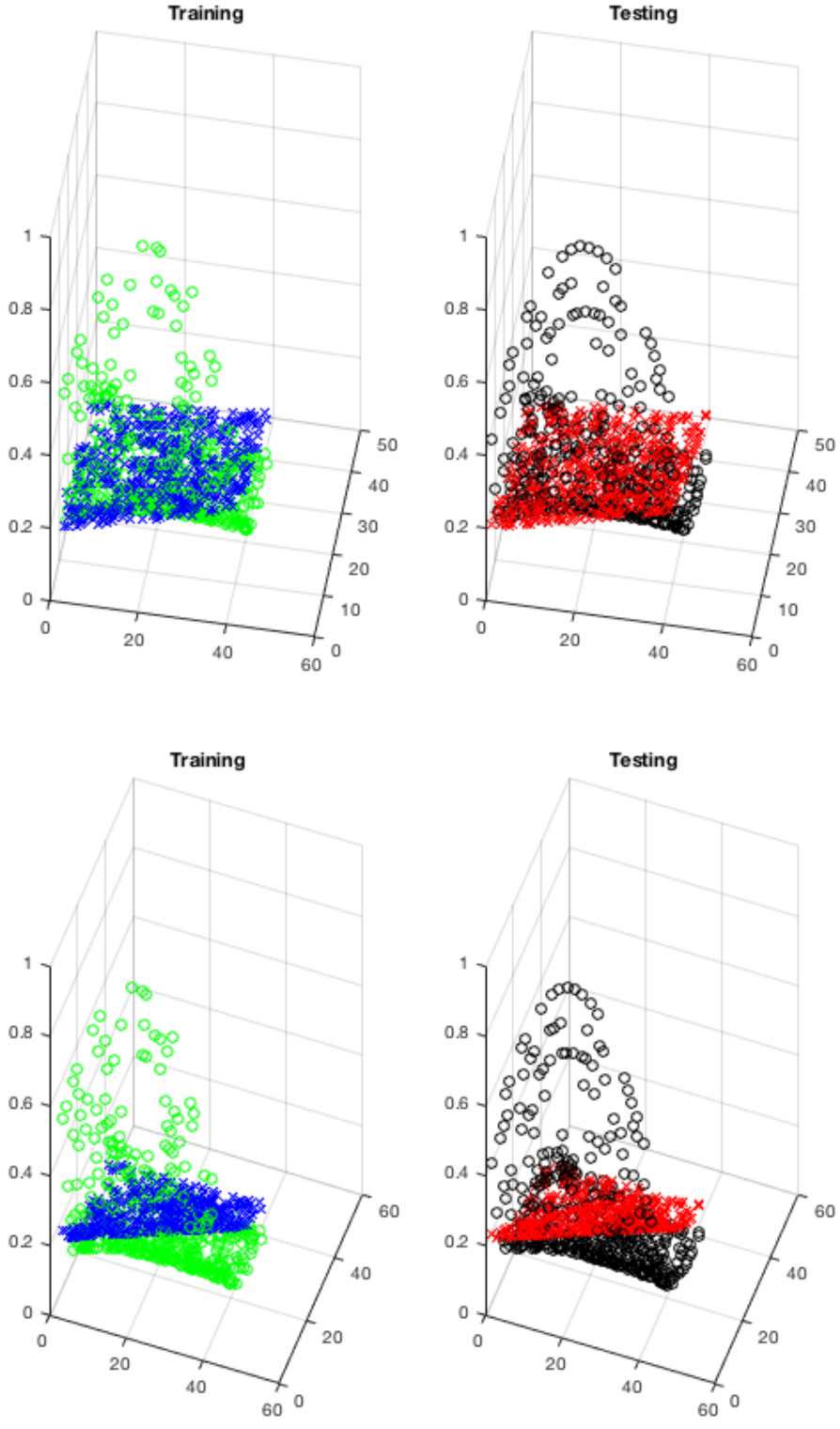
```
    % sse = sum(sum(error.^2)) % Error Reports - Not used by Algorithm
    % sseplot=[sseplot sse];
    % pause(0.05)

    drawnow()

end %end for loom

% plot(sseplot)



toc
end %end function



%%%%%%%%%%
function action = af (weighted_sum)
% plot(1./(1+exp(-(-10:0.1:10))))

action = 1./(1+exp(-weighted_sum));   % Logistic / Sigmoid Function
end

%%%%%%%%%%
function [X,Y,Z] = Hahn_landscapes(i)

size_search_space=1;
```

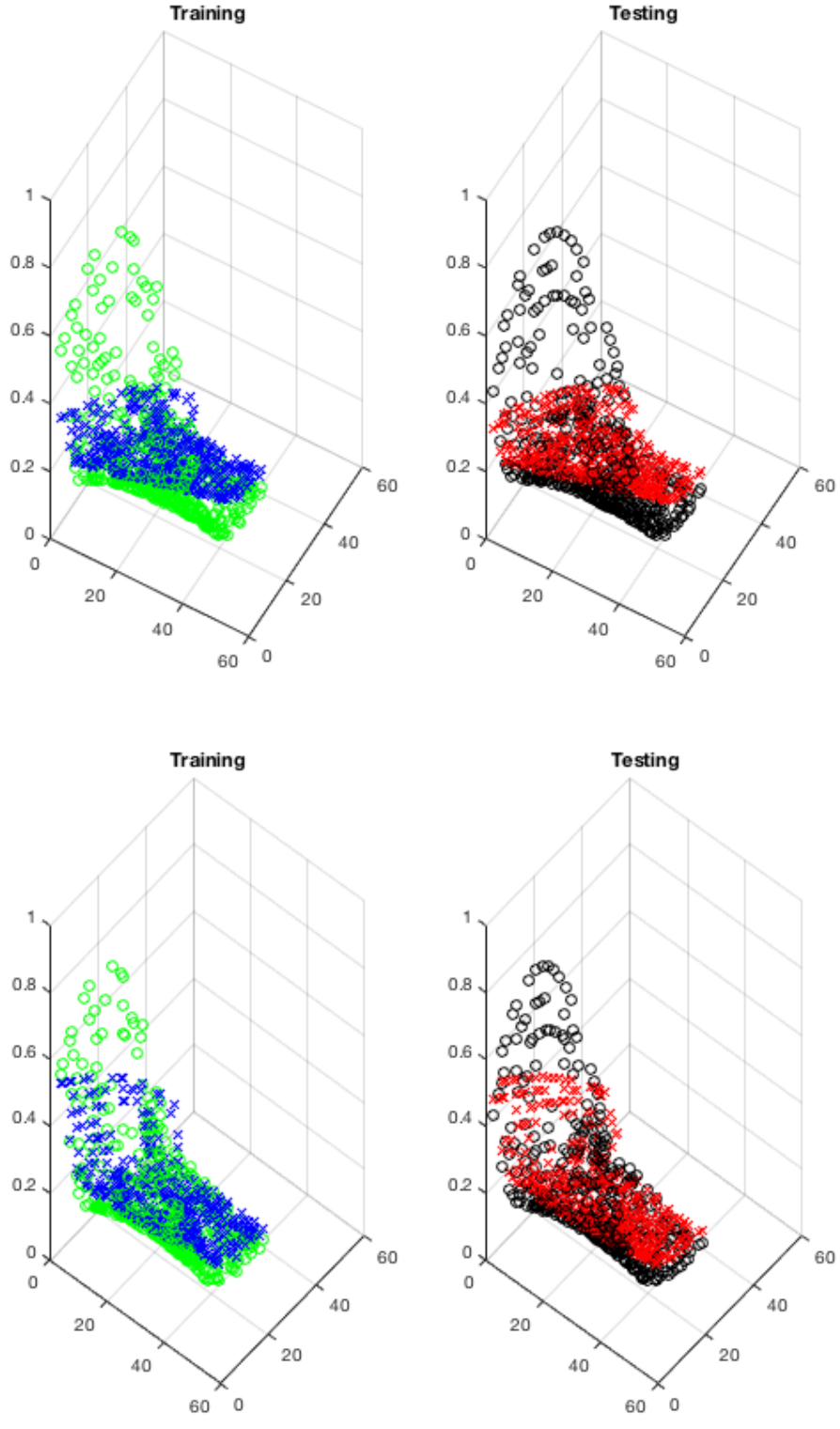
```
resolution=20;

[X,Y] = meshgrid(-size_search_space:size_search_space/
  resolution:size_search_space,-size_search_space:
   size_search_space/resolution:size_search_space);

Z=fitness(i,X,Y);

end



function z = fitness(c,x1,x2)

switch c
    case 0
        z=-2*exp(-.01*(x1-5).^2 - .01*(x2-5).^2);
    case 1
        z=20*(1 - exp(-0.2*sqrt(0.5*(x1.^2 + x2.^2))))-
         exp(0.5*(cos(2*pi*x1) + cos(2*pi*x2))) + exp(1);
    case 2
        z=(1.5 - x1 + x1.*x2).^2 + (2.25 - x1 + x1.*x2.^2).^2 +
         (2.625 - x1 + x1.*x2.^3).^2;
    case 3
        z=sin(x1).*exp((1-cos(x2)).^2) +
```

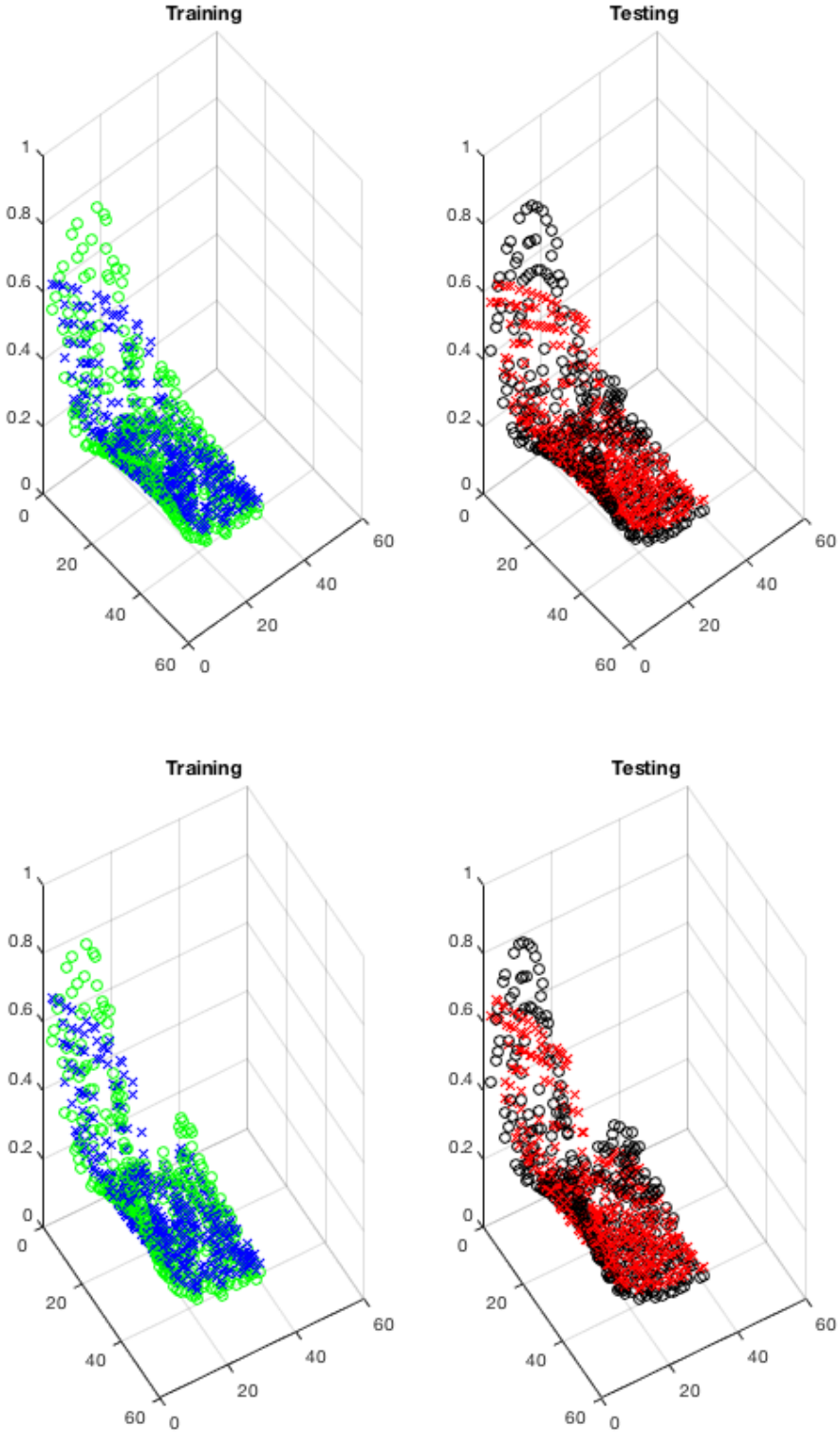
```
          cos(x2).*exp((1-sin(x1)).^2) + (x1-x2).^2;
    case 4
        z=(x1 + 2*x2 - 7).^2 + (2*x1 + x2 - 5).^2;
    otherwise
        disp('derp');
  end
end

Elapsed time is 72.558348 seconds.
```

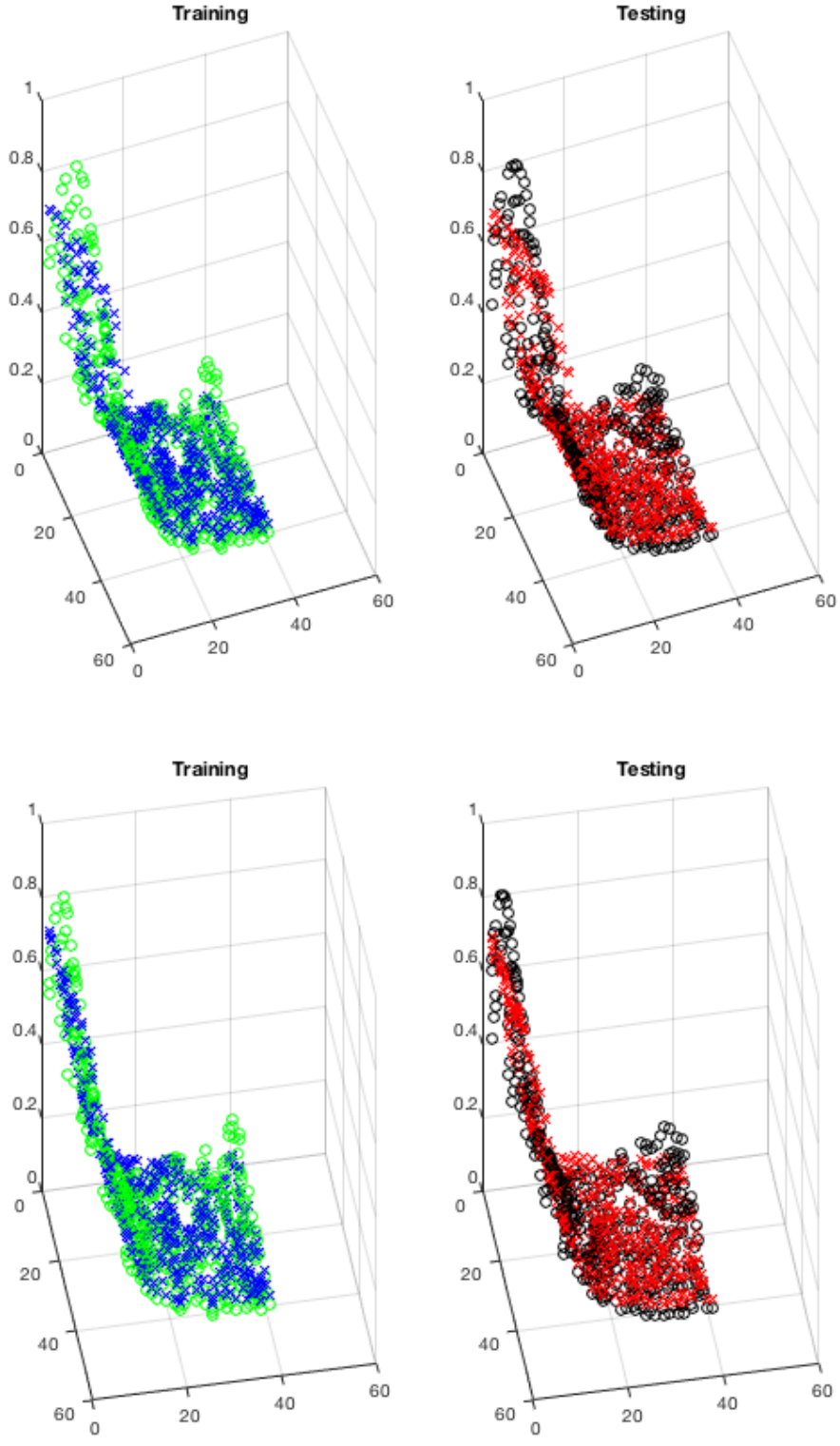

Neural Network Function Approximation
1
0.8
0.6
0.4
0.2
0
1
0.8
0.6
0.4
0.2
0
1
0.8
0.6
0.4
0.2
0

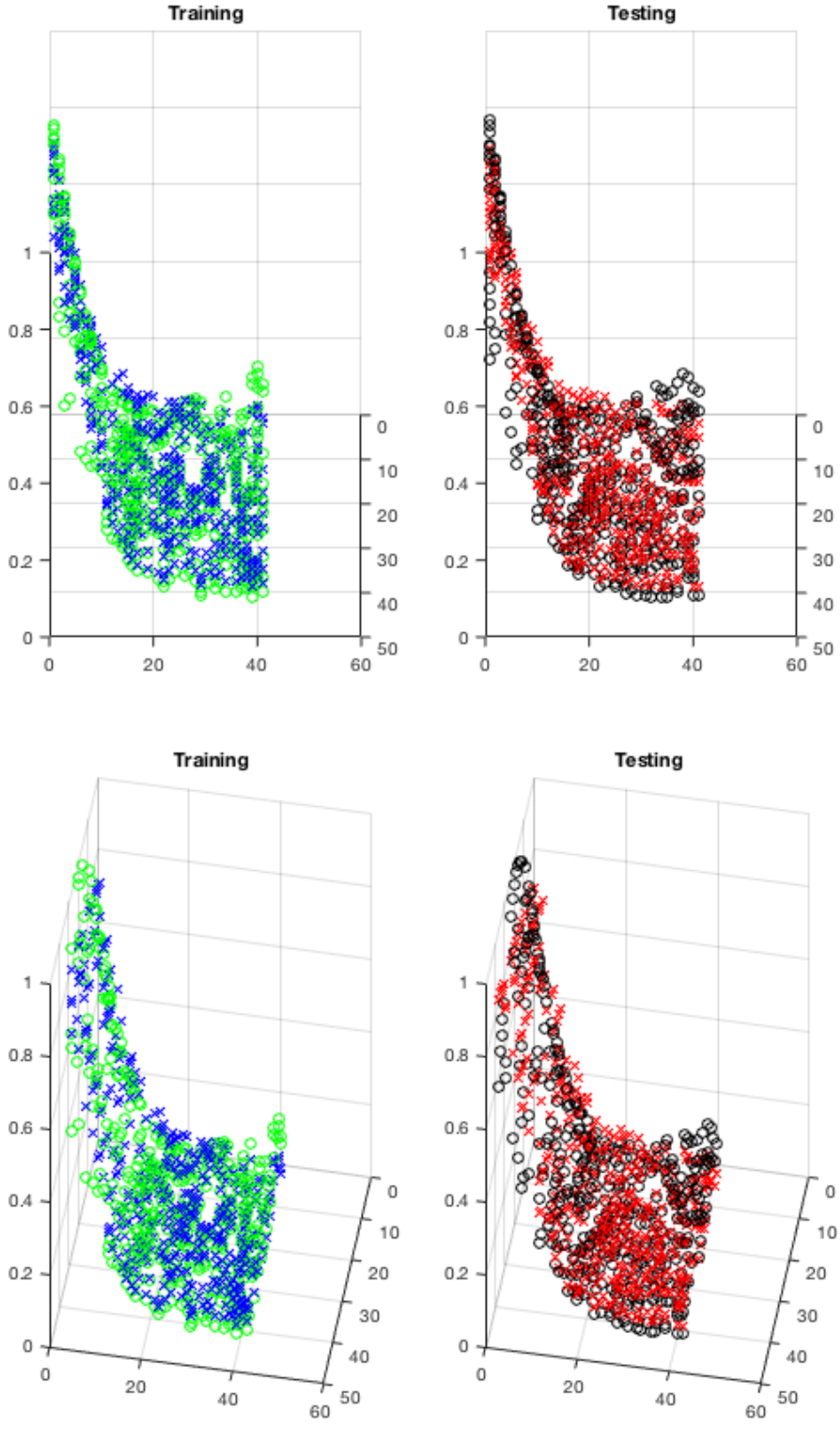

Neural Network Function Approximation Input
1
0.8
0.6
0.4
0.2
0
50
40
30
20
10
0
0
10
20
30
40
50


Training
Testing
Training
Testing


Training
Testing
Training
Testing
1
0.8
0.6
0.4
0.2
0
0
20
40
60
0
20
40
60


Training
Testing
Training
Testing
1
0.8
0.6
0.4
0.2
0
0
20
40
60
0
20
40
60


Training
Testing
Training
Testing
1
0.8
0.6
0.4
0.2
0
0
20
40
60


Training
Testing
Training
Testing
0
0.2
0.4
0.6
0.8
1
0
20
40
60
0
10
20
30
40
50

Car and Driver:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------%
%
% Machine Perception and Cognitive Robotics Laboratory
%
%     Center for Complex Systems and Brain Sciences
%
%               Florida Atlantic University
%
%-------------------------------------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------%
% William Hahn & Elan Barenholtz
% Feedback Amplifier
% Backpropagation Neural Network
% Supervised Learning - Car and Driver Dataset
% August 26th, 2014
% Revised June 2, 2015
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function MPCR_NN_CarDriver


load Hahn_CarDriver1.mat


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Visualize Raw Data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=[5 8 55 100 200]%1:size(pattern,1)
    pattern(i,:);
    subplot(121)
    imagesc(reshape(pattern(i,:),101,1175))
    colormap(gray)
    subplot(122)
    imagesc(category(i,:))
    snapnow

end

r=randperm(size(pattern,1)); %Shuffle Patterns

c=400; %Use the First 400 Patterns
pattern=pattern(r(1:c),:);
category=category(r(1:c),:);

bias=ones(size(pattern,1),1); %Add Bias (Default Resting State Potential)
pattern = [pattern bias];

n1 = size(pattern,2);   %Set the Number of Input Nodes Equal to Number of
%Pixels in the Input image
n2 = 10;   %n2-1        %Number of Hidden Nodes (Free Parameter)
n3 = size(category,2);  %Set the Number of Output Nodes Equal to the Number of
```

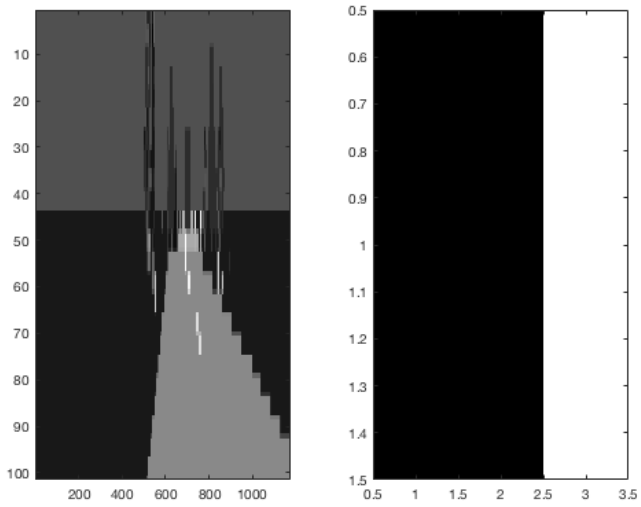
```
%Distinct Categories {left,forward,right}

w1 = 0.005*(1-2*rand(n1,n2-1)); %Randomly Initialize Hidden Weights
w2 = 0.005*(1-2*rand(n2,n3));   %Randomly Initialize Output Weights

dw1 = zeros(size(w1));          %Set Initial Hidden Weight Changes to Zero
dw2 = zeros(size(w2));          %Set Initial Output Changes to Zero

L = 0.01;              % Learning    %Avoid Overshooting Minima
M = 0.9;            % Momentum    %Smooths out the learning landscape

sse=size(pattern,1);  % Set Error Large so that Loop Starts
sseplot=[size(pattern,1) size(pattern,1) size(pattern,1)]; %Convergence Plot

for loop=1:100

    b=1;
%   b=0.1;

    act1 = [af(b*pattern * w1) bias];
    act2 = af(act1 * w2);

    error = category - act2;  %Calculate Error

    sse = sum(sum(error.^2)); % Error Reports - Not used by Algorithm
    sseplot=[sseplot sse];
```

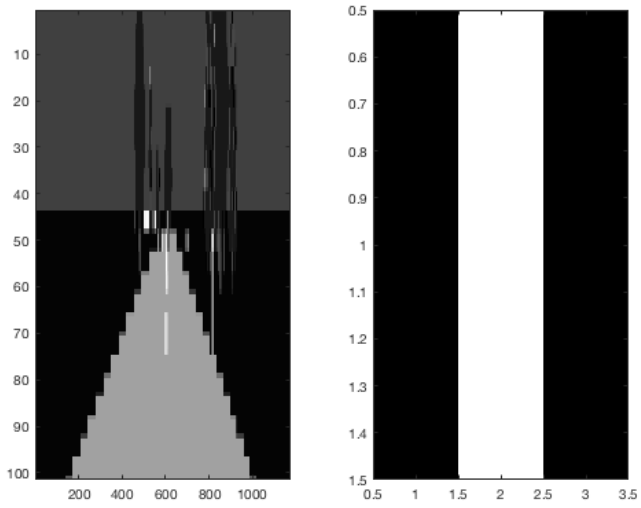
```
    delta_w2 = error .* act2 .* (1-act2); %Backpropagate Errors
    delta_w1 = delta_w2*w2' .* act1 .* (1-act1);
    delta_w1(:,size(delta_w1,2)) = []; %Remove Bias

    dw1 = L * pattern' * delta_w1 + M * dw1; %Calc. Hidden Weight Changes
    dw2 = L * act1' * delta_w2 + M * dw2;    %Calc. Output Weight Changes

    w1 = w1 + dw1; %Adjust Hidden Weights
    w2 = w2 + dw2; %Adjust Output Weights

    w1 = w1 + 0.0005*(1-2*randn(n1,n2-1));
    w2 = w2 + 0.0005*(1-2*randn(n2,n3));

    w1=w1/norm(w1);
    w2=w2/norm(w2);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Visualize Input Weights as Receptive Fields
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     if mod(loop,50)==0
%     figure(1)
%     set(gcf,'color','w');
    for i =1:n2-1
        subplot(sqrt(n2-1),sqrt(n2-1),i)
```

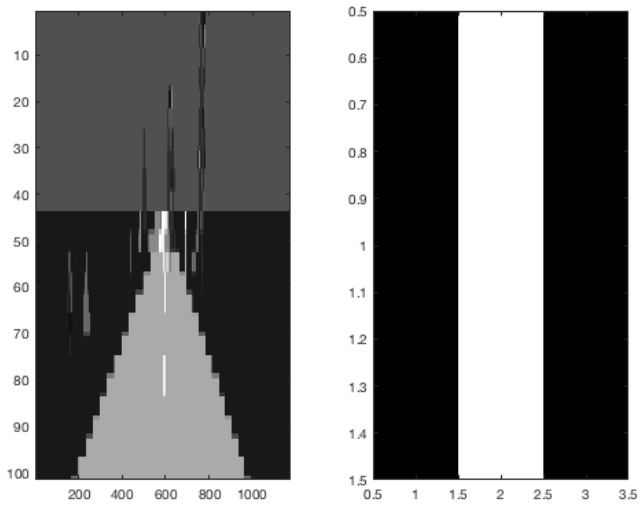
```
        imagesc(reshape(w1(1:n1-1,i),101,1175))
%         colormap(gray)
        axis off
    end

%     end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Visualize Network Performance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% if mod(loop,5)==0

%     figure(2)
%     set(gcf,'color','w');
%     subplot(141)
%     imagesc(category)
%     subplot(142)
%     imagesc(WTA(act2))
%     subplot(143)
%     imagesc(abs(category-WTA(act2)))
%     subplot(144)
%     plot(sseplot)
      drawnow()
```

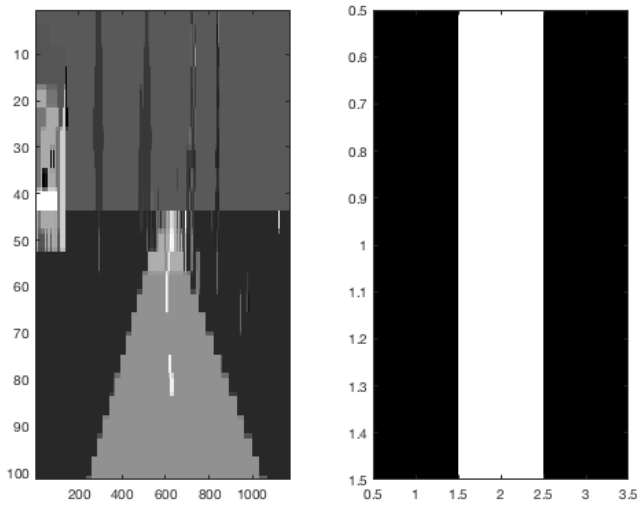
```
% end

end %end for loop

snapnow;

end %end function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function action = af (weighted_sum)

action = 1./(1+exp(-weighted_sum));   % Logistic / Sigmoid Function

end




function x = WTA(x)

for i=1:size(x,1)

[a,b]=max(x(i,:));

x(i,:)=1:size(x,2)==b;
```

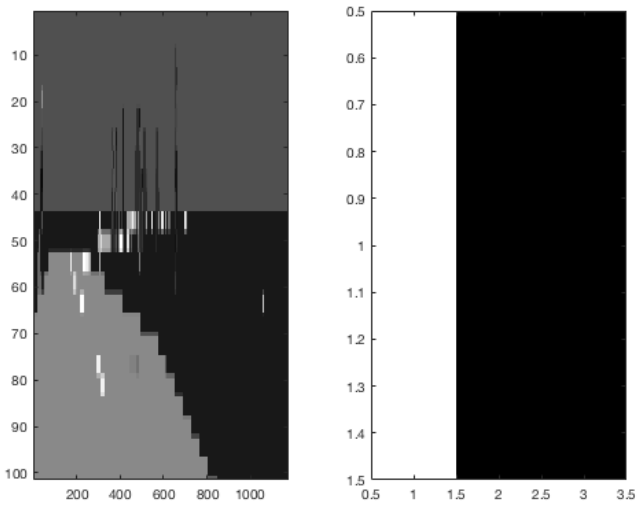
```
end


end
```

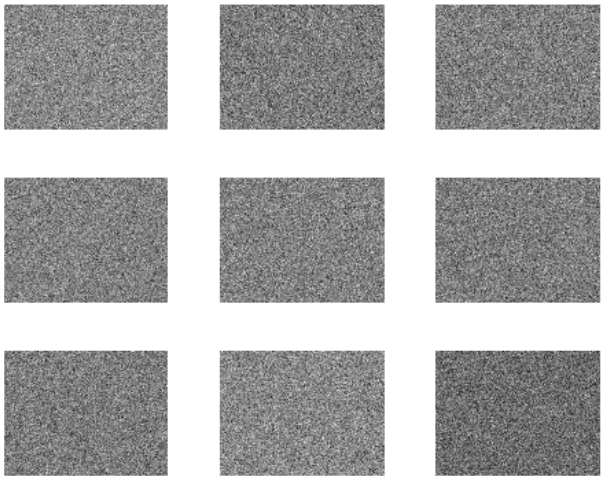

10
20
30
40
50
60
70
80
90
100
200 400 600 800 1000
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
0.5 1 1.5 2 2.5 3 3.5

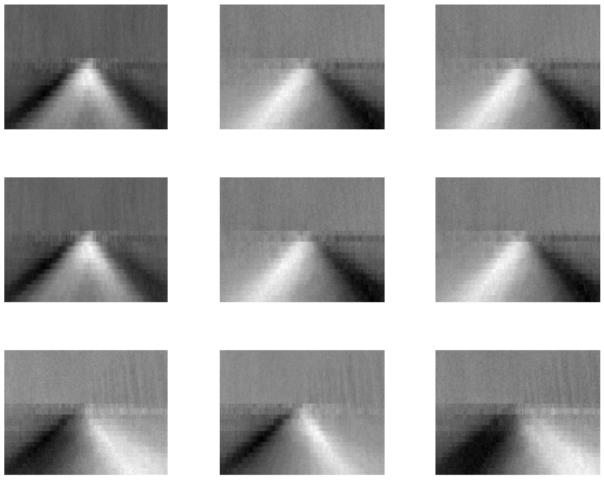

10
20
30
40
50
60
70
80
90
100
200 400 600 800 1000
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
0.5 1 1.5 2 2.5 3 3.5


10
20
30
40
50
60
70
80
90
100
200
400
600
800
1000
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
0.5
1
1.5
2
2.5
3
3.5


10
20
30
40
50
60
70
80
90
100
200
400
600
800
1000
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
0.5
1
1.5
2
2.5
3
3.5


10
20
30
40
50
60
70
80
90
100
200
400
600
800
1000
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
0.5
1
1.5
2
2.5
3
3.5





RALVINN

RALVINN: Recommendation Architecture Land Vehicle in a Neural Network, a sparse modeling and probabilistic reinforcement learning driven recommendation architecture for mobile robotics and autonomous vehicles.

```
########################################################
# -----------------------------------------------------#
#
# Machine Perception and Cognitive Robotics Laboratory
# Center for Complex Systems and Brain Sciences
#           Florida Atlantic University
#
#------------------------------------------------------#
########################################################
#------------------------------------------------------#
#
# Distributed ALVINN, See:
# Pomerleau, Dean A. Alvinn:
# An autonomous land vehicle in a neural network.
# No. AIP-77. Carnegie-Mellon Univ Pittsburgh Pa
# Artificial Intelligence And Psychology Project, 1989.
#
#------------------------------------------------------#
```

```
##########################################################

from roverBrain import *

if __name__ == '__main__':
    brain = roverBrain()

##########################################################
# -------------------------------------------------------#
#
# Machine Perception and Cognitive Robotics Laboratory
#   Center for Complex Systems and Brain Sciences
#           Florida Atlantic University
#
#-------------------------------------------------------#
##########################################################
#-------------------------------------------------------#
#
# Distributed ALVINN, See:
# Pomerleau, Dean A. Alvinn:
# An autonomous land vehicle in a neural network.
# No. AIP-77. Carnegie-Mellon Univ Pittsburgh Pa
# Artificial Intelligence And Psychology Project, 1989.
#
#-------------------------------------------------------#
##########################################################

from roverShell import *


class roverBrain():
    def __init__(self):
        self.quit = False
        self.rover = roverShell()

        self.fps = 10
        self.windowSize = [840, 380]
        self.imageRect = (0, 0, 320, 240)
        self.displayCaption = "DALVINN"
```

```
    pygame.init()
    pygame.display.init()
    pygame.display.set_caption(self.displayCaption)
    self.screen = pygame.display.set_mode(self.windowSize)
    self.clock = pygame.time.Clock()
    self.run()


def run(self):
    sleep(1.5)
    while not self.quit:
        self.parseControls()
        self.refreshVideo()
    self.rover.quit = True
    pygame.quit()

def blitscale(self, x):
    x -= np.min(x)
    x = x / np.linalg.norm(x)
    x *= 255.0 / x.max()

    return x


def refreshVideo(self):

    self.rover.lock.acquire()
    image = self.rover.currentImage
    self.rover.lock.release()

    image = pygame.image.load(cStringIO.StringIO(image), 'tmp.jpg').convert()

    imagearray = pygame.surfarray.array3d(image)
    imagearray = imresize(imagearray, (32, 24))
    image10 = pygame.surfarray.make_surface(imagearray)

    self.screen.blit(image10, (400, 0))
    pygame.display.update((400, 0, 32, 24))
```

```
        for k in range(min(1, self.rover.n2)):
            imagew11 = pygame.surfarray.make_surface(np.reshape(self.blitscale(self.rover.w1[:-1, k]), (32, 24, 3)))
            self.screen.blit(imagew11, (500 + 40 * k, 0))
            pygame.display.update((500 + 40 * k, 0, 32, 24))

        for k in range(min(1, self.rover.n2)):
            imagedw11 = pygame.surfarray.make_surface(np.reshape(self.blitscale(self.rover.dw1[:-1, k]), (32, 24, 3)))
            self.screen.blit(imagedw11, (500 + 40 * k, 50))
            pygame.display.update((500 + 40 * k, 50, 32, 24))

        self.screen.blit(image, (0, 0))
        pygame.display.update(self.imageRect)

        self.clock.tick(self.fps)


    def parseControls(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                self.quit = True
            elif event.type == KEYDOWN:
                if event.key in (K_j, K_k, K_SPACE, K_u, K_i, K_o):
                    self.updatePeripherals(event.key)
                elif event.key in (K_w, K_a, K_s, K_d, K_q, K_e, K_z, K_c, K_r):
                    self.updateTreads(event.key)
                else:
                    pass
            elif event.type == KEYUP:
                if event.key in (K_w, K_a, K_s, K_d, K_q, K_e, K_z, K_c, K_r):
                    self.updateTreads()
                elif event.key in (K_j, K_k):
                    self.updatePeripherals()
                else:
                    pass
            else:
                pass
```

```
def updateTreads(self, key=None):

    if key is None:
        self.rover.treads = [0, 0]
    elif key is K_w:
        self.rover.treads = [1, 1]
    elif key is K_s:
        self.rover.treads = [-1, -1]
    elif key is K_a:
        self.rover.treads = [-1, 1]
    elif key is K_d:
        self.rover.treads = [1, -1]
    elif key is K_q:
        #print self.rover.nn_treads
        #self.rover.treads = self.rover.nn_treads
        self.rover.treads = [.1, 1]
    elif key is K_e:
        self.rover.treads = [1, .1]
    elif key is K_z:
        self.rover.treads = [-.1, -1]
    elif key is K_c:
        self.rover.treads = [-1, -.1]
    else:
        pass


def updatePeripherals(self, key=None):
    if key is None:
        self.rover.peripherals['camera'] = 0
    elif key is K_j:
        self.rover.peripherals['camera'] = 1
    elif key is K_k:
        self.rover.peripherals['camera'] = -1
    elif key is K_u:
        self.rover.peripherals['stealth'] = not \
            self.rover.peripherals['stealth']
    elif key is K_i:
        self.rover.peripherals['lights'] = not \
            self.rover.peripherals['lights']
```

```
        elif key is K_o:
            self.rover.peripherals['detect'] = not \
                self.rover.peripherals['detect']
        elif key is K_SPACE:
            pass
        else:
            pass

##########################################################
# -------------------------------------------------------#
#
# Machine Perception and Cognitive Robotics Laboratory
# Center for Complex Systems and Brain Sciences
#           Florida Atlantic University
#
#--------------------------------------------------------#
##########################################################
#--------------------------------------------------------#
#
# Distributed ALVINN, See:
# Pomerleau, Dean A. Alvinn:
# An autonomous land vehicle in a neural network.
# No. AIP-77. Carnegie-Mellon Univ Pittsburgh Pa
# Artificial Intelligence And Psychology Project, 1989.
#
#--------------------------------------------------------#
##########################################################

import pygame
from pygame.locals import *
from time import sleep
from datetime import date
from random import choice
from string import ascii_lowercase, ascii_uppercase
import threading
import cStringIO
import numpy as np
from scipy.misc import imresize
from scipy import ndimage as ndi
```

```
from af import *

from rover import Rover20



class roverShell(Rover20):
    def __init__(self):
        Rover20.__init__(self)
        self.quit = False
        self.lock = threading.Lock()

        self.treads = [0, 0]
        self.nn_treads = [0, 0]
        self.currentImage = None
        self.peripherals = {'lights': False, 'stealth': False, \
                            'detect': True, 'camera': 0}

        self.action_choice = 1
        self.action_labels = ['forward', 'backward', 'left', 'right']
        self.action_vectors_motor = [[1, 1], [-1, -1], [-1, 1], [1, -1]]
        self.action_vectors_neuro = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]

        self.n1 = 32 * 24 * 3
        self.n2 = 2
        self.n3 = 4

        self.w1 = 0.00001 * np.random.random((self.n1 + 1, self.n2))
        self.w2 = 0.01 * np.random.random((self.n2 + 1, self.n3))

        self.dw1 = np.zeros(self.w1.shape)
        self.dw2 = np.zeros(self.w2.shape)

        self.L1 = 0.000001
        self.L2 = 0.001
        self.M = 0.9



    # main loop
    def processVideo(self, jpegbytes, timestamp_10msec):
```

```
        self.lock.acquire()
        if self.peripherals['detect']:
            self.processImage(jpegbytes)
            self.currentImage = jpegbytes
        else:
            self.currentImage = jpegbytes
        self.lock.release()
        self.setTreads(self.treads[0], self.treads[1])
        self.setPeripherals()
        if self.quit:
            self.close()


    # openCV operations
    def processImage(self, jpegbytes):


        self.currentImage = imresize(
            pygame.surfarray.array3d(pygame.image.load(cStringIO.StringIO(jpegbytes), 'tmp.jpg').convert()),
            (32, 24))

        self.currentImage = ndi.gaussian_filter(self.currentImage, .5) -
        ndi.gaussian_filter(self.currentImage, 1)

        self.currentImage = self.currentImage / 255.0

        self.pattern = np.tile(np.reshape(self.currentImage, (32 * 24 * 3)), (64, 1))

        self.pattern = self.pattern + 0.001 * (1 - np.random.random((self.pattern.shape[0],
        self.pattern.shape[1])))

        self.bias = np.ones((self.pattern.shape[0], 1))

        self.pattern = np.concatenate((self.pattern, self.bias), axis=1)

        self.act1 = np.concatenate((np.squeeze(np.array(af(np.dot(self.pattern, self.w1)))), self.bias), axis=1)

        self.act2 = np.squeeze(np.array(af(np.dot(self.act1, self.w2))))
```

```
        self.act22 = 0 * self.act2

        for i in range(self.act2.shape[0]):
            self.act22[i, np.argmax(self.act2[i, :])] = 1

        self.nn_treads = self.action_vectors_motor[np.argmax(np.sum(self.act22, axis=0))]

        print self.nn_treads

        if np.sum(np.abs(self.treads)):

            for i in range(np.asarray(self.action_vectors_motor).shape[0]):
                if self.action_vectors_motor[i] == self.treads:
                    break

            self.category = np.tile(self.action_vectors_neuro[i], (self.pattern.shape[0], 1))

            self.error = self.category - self.act2

            self.sse = np.power(self.error, 2).sum

            self.delta_w2 = self.error * self.act2 * (1 - self.act2)

            self.delta_w1 = np.dot(self.delta_w2, self.w2.transpose()) * self.act1 * (1 - self.act1)

            self.delta_w1 = np.delete(self.delta_w1, -1, 1)

            self.dw1 = np.dot(self.L1, np.dot(self.pattern.transpose(), self.delta_w1)) + self.M * self.dw1
            self.dw2 = np.dot(self.L2, np.dot(self.act1.transpose(), self.delta_w2)) + self.M * self.dw2
            self.w1 = self.w1 + self.dw1
            self.w2 = self.w2 + self.dw2

            self.w1 = self.w1 + 0.00001 * (-0.5 + np.random.random((self.w1.shape[0], self.w1.shape[1])))
            self.w2 = self.w2 + 0.00001 * (-0.5 + np.random.random((self.w2.shape[0], self.w2.shape[1])))


    # camera features
    def setPeripherals(self):
        if self.peripherals['lights']:
```

```
            self.turnLightsOn()
        else:
            self.turnLightsOff()

        if self.peripherals['stealth']:
            self.turnStealthOn()
        else:
            self.turnStealthOff()

        if self.peripherals['camera'] in (-1, 0, 1):
            self.moveCameraVertical(self.peripherals['camera'])
        else:
            self.peripherals['camera'] = 0

##########################################################
import numpy as np

def af(x):
    return [1 / (1 + np.exp(-x))]

##########################################################
```

A major recent development in neural networks is the Dropout idea due to Hinton's Lab. Consider a neural network with L hidden layers. Let $l \in \{1, \ldots, L\}$ index the hidden layers of the network. Let $z^{(l)}$ denote the vector of inputs into layer l, $y^{(l)}$ denote the vector of outputs from layer l, where $y^{(0)} = x$ is the input. $W^{(l)}$ and $b^{(l)}$ are the weight and basis at layer l. For $l \in \{0, \ldots, L-1\}$

$$z^{(l+1)} = W^{(l+1)} y^l + b^{(l+1)} \tag{2.36}$$

$$y^{(l+1)} = f(z^{(l+1)}) \tag{2.37}$$

where f is any activation. With dropout,

$$r_i^{(l)} \sim \text{Bernoulli}(p) \tag{2.38}$$

$$\widetilde{y}^{(l)} = r^{(l)} * y^{(l)} \tag{2.39}$$

$$z^{(l+1)} = W^{(l+1)}\widetilde{y}^{l} + b^{(l+1)} \tag{2.40}$$

$$y^{(l+1)} = f(z^{(l+1)}) \tag{2.41}$$

here $r^{(l)}$ is a vector of Bernoulli random variables each of which has probability p of being 1.

This vector is sampled for each layer and multiplied element-wise with the output of that layer, $y^{(l)}$, to create the thinned outputs $\widetilde{y}^{(l)}$ The thinned outputs are then used as input to the next layer. At the time of the test the weights are scaled as $W_{test}^{(l)} = pW^{(l)}$

"For past decades gradient descent based methods have mainly been used in various learning algorithms of feedforward neural networks; however, it is clear that gradient descent based learning methods are generally very slow due to improper learning steps and may easily converge to local minimums. Many iterative learning steps are required by such learning algorithms in order to obtain better learning performance." [76] "The popular learning algorithm used in feedforward neural networks is the back-propagation learning algorithm where gradients can be computed efficiently by propagation from the output to the input."[76]

There are several issues with the back-propagation learning algorithms:

1. When the learning rate is too small, the learning algorithm converges very slowly

   However, when learning rate is too large, the algorithm becomes unstable and diverges

2. Another peculiarity of the error surface that impacts the performance of the back-propagation learning algorithm is the presence of local minima

   It is undesirable that the learning algorithm stops at a local minimum if it is located far above a global minimum[76]

3. Neural network may be over-trained by using back- propagation algorithms and obtain worse generalization performance

   Thus, validation and suitable stopping methods are required in the cost function minimization procedure

4. Gradient-based learning is very time-consuming in most applications[76]

Many researchers have explored the universal approximation capabilities of standard multi-layer feedforward neural networks; in real applications, the neural networks are trained in finite training set.[76]

"In the case of function approximation with a finite training set it has been shown that "a single-hidden layer feedforward neural network (SLFN) with at most N hidden neurons and with almost any nonlinear activation function can learn N distinct observations with zero error."[76]

2.2 RESERVOIR COMPUTING

Frank Rosenblatt's dream was that perceptron would be "the embryo of an electronic computer that will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.[75] It is very hard to imagine how such a machine could be build with build using traditional human engineering practices. Biological networks are sprawling, often apparently random looking systems of interconnected things. "Traditionally, all the parameters of the feedforward networks need to be tuned, and thus there exists the dependency between different layers of parameters (weights and biases)."[76] "Machines which are largely random in their construction in this way will be called Unorganized Machines. The machine is made up from a rather large number N of similar units [and] can behave in a very complicated manner when the number of units is large...this suggests that the cortex of the infant is an unorganized machine." [151] John von Neumann wondered how an imperfect neural network, containing many random connections, can be made to perform reliably those functions which might be represented by idealized wiring diagrams[75] Hinton has suggested that "maybe the kind of signal processing problem that the brain solves is very different from the kind of signal processing problem engineers solve". [87]

Dithering is a technique of using "intentionally applied forms of noise used to randomize quantization error, preventing large-scale patterns such as color banding in images. Dither is routinely used in processing of both digital audio and video data, and is often one of the last stages of mastering audio to a CD."[120] "One of the earliest applications of dither came in World War II. Airplane bombers used mechanical computers to perform navigation and bomb trajectory calculations. Curiously, these computers (boxes filled with hundreds of gears and cogs) performed more accurately

when flying on board the aircraft, and less well on ground. Engineers realized that the vibration from the aircraft reduced the error from sticky moving parts. Instead of moving in short jerks, they moved more continuously. Small vibrating motors were built into the computers, and their vibration was called dither from the Middle English verb "didderen," meaning "to tremble." Today, when you tap a mechanical meter to increase its accuracy, you are applying dither, and modern dictionaries define dither as a highly nervous, confused, or agitated state. In minute quantities, dither successfully makes a digitization system more analog." [120]

Machine Learning techniques based on Random Projections have been developing at an increasing rate recently, leading to two main frameworks: Reservoir Computing (RC) methods and Extreme Learning Machine (ELM) methods.[107] According to Miche "most of the current and past machine learning techniques using random projections can be sorted into these categories"[107] Reservoir Computing comprises various methods such as Echo-State Networks (ESN) and Liquid-State Machines (LSM). [107] The difference between RC and ELM techniques "lies in the recurrence of the underlying neural network."[107] RC makes use of a "pool of neurons, randomly interconnected with each other, which can be described as a Recurrent Neural Network(RNN)." [107] RNN have been shown theoretically to be "powerful tools for solving complex spatio-temporal machine learning tasks."[107]

Random projection techniques for performance and speed in machine learning:

1. Reservoir Computing

    - spatio-temporal; recurrence; continuous
    - Echo State network

- Liquid State machine

2. Extreme Learning Machine

   - spatial; not temporal (no recurrence); discrete
   - Single-Layer Feedforward Neural Network

Machine Learning Techniques based on Random Projections

Johnson-Lindenstrauss

Random matrices can preserve pairwise distances.[107]

For any $0 < \varepsilon < \frac{1}{2}$ *and sample* $x_1, x_2, ..., x_n \in \mathbb{R}^d$, *there exists an* $f : \mathbb{R}^d \to \mathbb{R}^k$ *for* $k = O(\varepsilon^{-2} \log n)$ *such that*

$$\forall i, j \ (1-\varepsilon)||x_i - x_j||_2 \leq ||f(x_i) - f(x_j)||_2 \leq (1+\varepsilon)||x_i - x_j||_2 \tag{2.42}$$

.

$$k = \Omega \frac{\varepsilon^{-2} \log(n)}{\log\left(\frac{1}{\varepsilon}\right)} \tag{2.43}$$

.

The data points in the high dimensional space "are mapped towards a lower dimension space with a linear transformation that maintains the data's topology by preserving the pairwise distances...simple projection matrix that preserves the distances to the same factor than the J-L theorem mentions, at the expense of a probability on the distance conservation."[107]

Input data

$$\mathbf{x} = \big(\mathbf{x}_1, ..., \mathbf{x}_N\big)^T,\ \mathbf{x}_i \in \mathbb{R}^d \tag{2.44}$$

to model

$$\mathbf{y} = \big(y, ..., y_N\big)^T, \tag{2.45}$$

Output of the reservoir $\hat{\mathbf{y}}$ at step $k+1$

$$\hat{\mathbf{y}}(k+1) = \mathbf{f}^{out}\Big(\mathbf{W}^{out} \cdot \big[\mathbf{x}(k+1), \mathbf{f}^{in}\big(\mathbf{W}^{in} \cdot \mathbf{x}(k+1) + \mathbf{W}\mathbf{s}(k) + \mathbf{W}^{back}\mathbf{y}(k)\big), \mathbf{y}(k)\big]\Big) \tag{2.46}$$

with $\mathbf{x}(k)$ denoting the input data $\mathbf{x}$ fed to the network at step k; $\mathbf{W}^{out}$ is the output weight matrix,$\mathbf{W}^{in}$ is the input weight matrix, $\mathbf{W}$ is the internal weight matrix, and $\mathbf{W}^{back}$ is the back-projection of output to internal network weight matrix; $\mathbf{s}$ is the internal network state; $\mathbf{f}^{in}$ is the internal netwrok activation function; and $\mathbf{f}^{out}$ is the readout function;
when $\mathbf{f}^{out}$ is a linear classifier, the Reservoir Computer is called Echo State Network

Usually $\mathbf{f}^{in}$ is sigmoid; when $\mathbf{f}^{in}$ is a spiking neural network, the Reservoir Computer is called Liquid State Machine

Extreme Learning Machine prpoposed by Huang et al. uses Single-Layer Feedforward Neural Network; no recurrence; random intiation of weights. A Single Layer Feed Forward Neural Network (Directed Acyclic Graph) with M hidden neurons, where f

is the activation function, $\mathbf{w}_i$ the input weights and β_i the output weights:

$$\sum_{i=1}^{M} \beta_i \mathbf{f}\big(\mathbf{w}_i \mathbf{x}_j + b_i\big) = y_j, \quad 1 \leq j \leq N \tag{2.47}$$

Compact matrix notation of SLFN:

$$\mathbf{H}\beta = \mathbf{y} \tag{2.48}$$

Given a training set

$$\aleph = \{(x_i, t_i) \mid x_i \in \mathbb{R}^n, t_i \in \mathbb{R}^m, i = 1, \ldots, N\} \tag{2.49}$$

with activation function $f(x)$, and hidden node number $\widetilde{N}$,

1. Randomly assign input weight w_i and bias b_i, $i = 1, \ldots, \widetilde{N}$.
2. Calculate the hidden layer output matrix $\mathbb{H}$
3. Calculate the output weight matrix $\beta = \mathbb{H}^{\dagger}\mathbb{T}$, where $\mathbb{T} = [t_1, \ldots, t_N]^T$, where the matrix $H^{\dagger}$ is the Moore-Pennrose generalized pseudoinverse.

"The Extreme Learning Machine (ELM) is one of the hottest connectionist model at the present" and there is much dispute over the origins of these ideas (See `http://elmorigin.wix.com/originofelm`)

1. "The kernel (or constrained-optimization-based) version of ELM (ELM-Kernel) is identical to kernel ridge regression (for regression and single-output classification, as well as the LS-SVM with zero bias; for multiclass multi-output classification)."

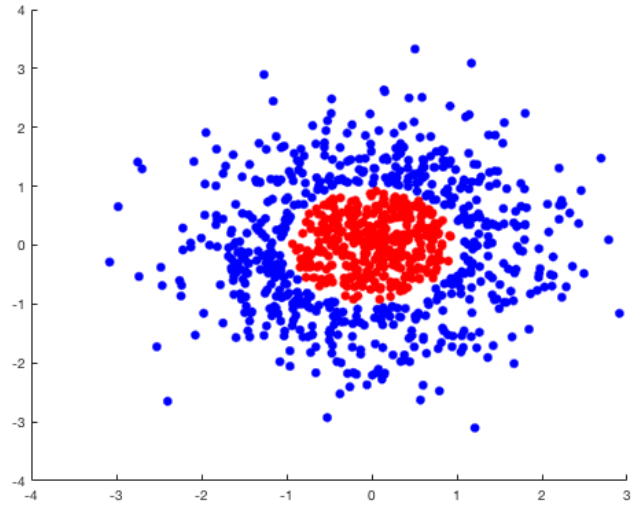
2. "ELM-SLFN (the single-layer feedforward network version of the ELM) is identical to the randomized neural network (RNN, with omission of bias) and another simultaneous work, i.e., the random vector functional link (RVFL, with omission of direct input-output links)."

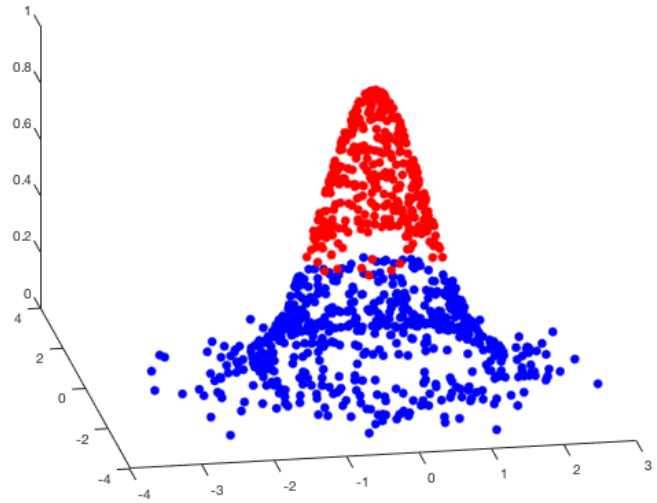
3. "ELM-RBF is identical to the randomized RBF neural network (with a performance-degrading randomization of RBF radii or impact factors).'

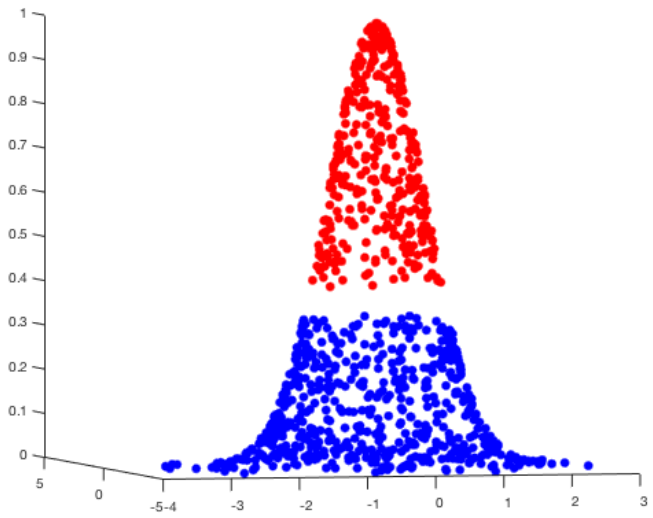
Gaussian Kernel Method demonstrates the utility of mapping to a higher dimension

```
%Hahn
%Gaussian Kernel Method Visualization
n=1000


x=randn([1,n]);
y=randn([1,n]);


a=x.^2+y.^2<0.9


b=x.^2+y.^2>1.1


z=exp(-(x.^2+y.^2));



plot3(x(a),y(a),z(a),'.','MarkerSize',20,'color', [1 0 0])
hold on
plot3(x(b),y(b),z(b),'.','MarkerSize',20,'color', [0 0 1])
view(0,90)
```

```
snapnow

view(-10,30)

snapnow

view(-14,3)

snapnow
```







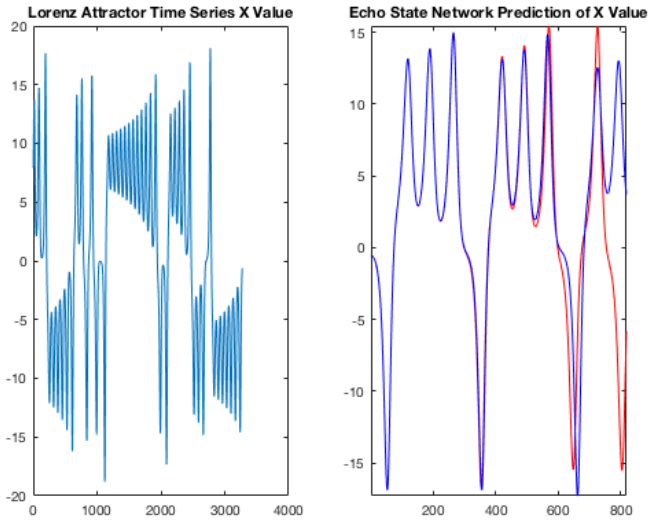

```
function Hahn_Thesis_Echo_Network

%-------------------------------------------------------%

%Echo State Network Time Series Prediction

%See: http://minds.jacobs-university.de/mantas

%http://organic.elis.ugent.be/sites/organic.elis.ugent.be/files/PracticalESN.pdf

%http://www.physics.emory.edu/faculty/weeks//research/tseries8.html

%Data set created by Runge-Kutta integration of the Lorenz equations

%The Lorenz equations are given by:

% dx/dt = sigma * (y - x)

% dy/dt = r * x - y - x * z

% dz/dt = x * y - b * z

% sigma=10.0, r = 28.0, b = 8/3

% Step size = 0.01

%-------------------------------------------------------%


clear all
```

```
close all
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

data = load('LORENZ.DAT');

m = [floor(0.8*size(data,1)) floor(0.15*size(data,1)) floor(0.05*size(data,1))]; %
n = [1 500 1];  %nodes: network input - reservoir - output



a = 0.3;
r = 1e-8*eye(1+n(1)+n(2));

Wi = (rand(n(2),1+n(1))-0.5);
Wr = 0.1.*(rand(n(2),n(2))-0.5);

x = zeros(n(2),1);
X = zeros(1+n(1)+n(2),m(1)-m(2));
Y = zeros(n(3),m(3));



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Train
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Yt = data(m(2)+2:m(1)+1)';

for t = 1:m(1)

    u = data(t);

    x = (1-a)*x + a*tanh( Wi*[1;u] + Wr*x );

    if t > m(2)
        X(:,t-m(2)) = [1;u;x];
    end

end

Wo=(Yt*X')/(X*X'+r);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Test
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u = data(m(1)+1);

for t = 1:m(3)

    x = (1-a)*x + a*tanh( Wi*[1;u] + Wr*x );
```

```
    y = Wo*[1;u;x];

    Y(:,t) = y;

    u = y;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1);
subplot(121)
plot(data(m(1)-4*m(3):m(1)-1));
title('Lorenz Attractor Time Series X Value')

subplot(122)
plot( data(m(1)+2:m(1)+m(3)+1), 'r');
hold on;
plot( Y', 'b' );
hold off;
axis tight;
title('Echo State Network Prediction of X Value')
```

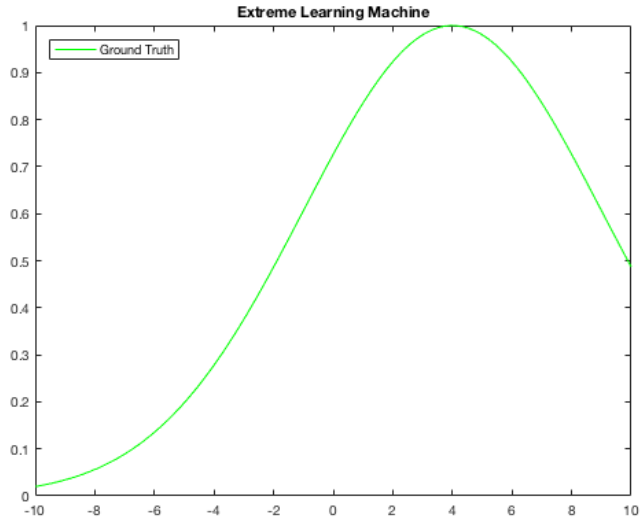
```
end
```

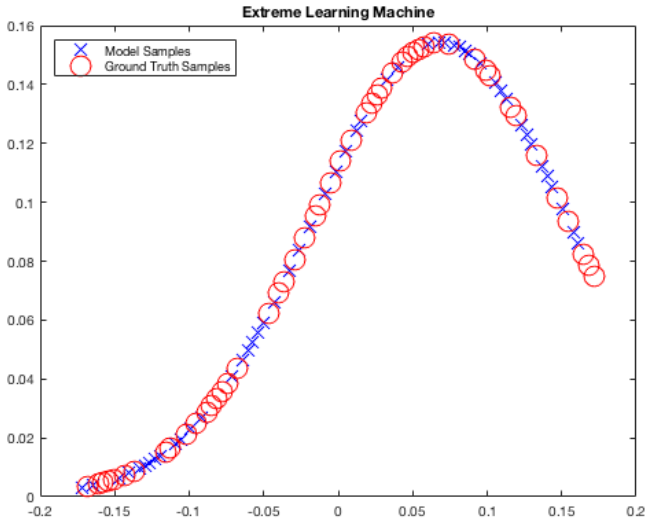



```
function Hahn_Thesis_ELM()


close all

clear all

clc


x=linspace(-10,10,100)';

% y=10.*x+0.5;

% y=x.^2;

% y=abs(sin(x/8));

% y=0.5*sinc(x/8);

y=exp(-0.02.*(x-4).^2);


plot(x,y,'g-')
```

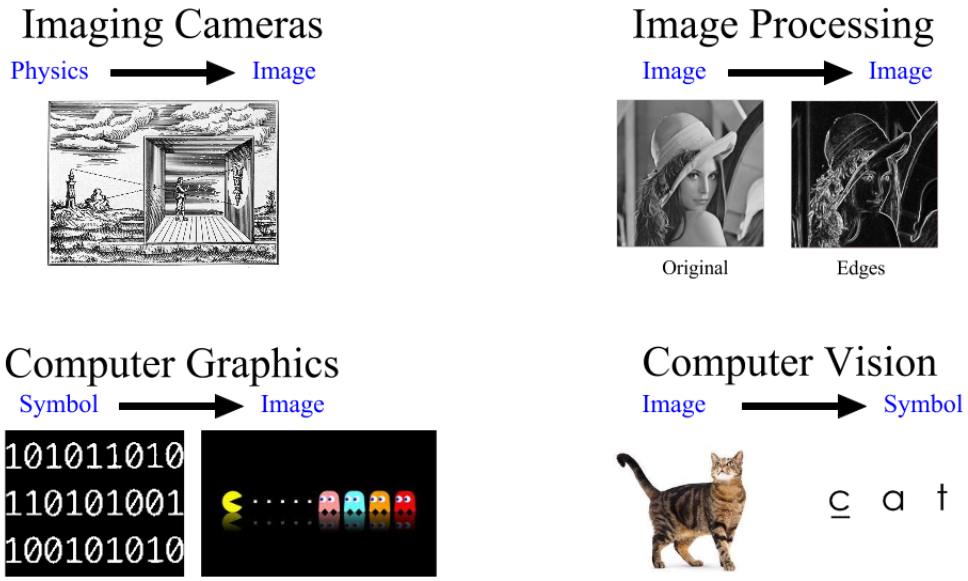
```
legend('Ground Truth','Location','NorthWest')
title('Extreme Learning Machine')
snapnow


x=x/norm(x);
y=y/norm(y);

r=randperm(size(x,1));

x=x(r);
y=y(r);

x1=x(1:end/2); %train data
x2=x(end/2+1:end); %test data

y1=y(1:end/2); %train data
y2=y(end/2+1:end); %test data


h=1000;

W1=randn(size(x1,2)+1,h)+eps; %random input weights

H1=tanh(([x1 ones(size(x1,1),1)]*W1)); %hidden layer activation
```

```
B=H1\y1; %linear model to learn output weights



H2=tanh(([x2 ones(size(x2,1),1)]*W1)); %hidden layer activation for testing

y02=(H2*B); %hidden layer times linear model weights yields output


plot(x2,y02,'bx','Markersize', 10)
hold on
plot(x1,y1,'ro','Markersize', 15)
legend('Model Samples', 'Ground Truth Samples','Location','NorthWest')
title('Extreme Learning Machine')
snapnow


end
```


Extreme Learning Machine
Ground Truth
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-10
-8
-6
-4
-2
0
2
4
6
8
10


Extreme Learning Machine
Model Samples
Ground Truth Samples
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2

2.3 INVERSE GRAPHICS



Images can be interpreted as functions. When continuous, we can consider the function $f(x) = f(h, v)$, where h and v are continuous, as mapping from

$$f : \mathbb{R}^2 \to \mathbb{R}^d \tag{2.50}$$

When discrete, we can consider the function $f(n) = (n_1, n_2)$ as mapping from

$$f : \mathbb{Z}^2 \to \mathbb{R}^d \tag{2.51}$$

2.4 DICTIONARY LEARNING

Sparse least squares energy function captures a good chunk of the neuroscience and computer vision research in the last decade.- Garrett Kenyon

Given an N-dimensional stimulus $s \in \mathbb{R}^N$ we seek a representation in terms of a dictionary $\mathcal{D}$ composed of M vectors $\{\phi_m\}$ that span the space $\mathbb{R}^N$. When the dictionary is overcomplete ($M > N$), there are an infinite number of ways to choose coefficients

$\{a_m\}$ such that

$$s = \sum_{m=1}^{M} a_m \phi_m$$

In optimal sparse approximation, we seek the coefficients having the fewest number of nonzero entries by solving the minimization problem,

$$\min_a \quad ||a||_0 \quad \text{subject to} \quad s = \sum_{m=1}^{M} a_m \phi_m$$

where the ℓ^0 "norm" denotes the number of nonzero elements of $a = [a_1, a_2, \ldots, a_M]$ This combinatorial optimization problem is NP-hard.

$$\min_{\alpha} ||\alpha||_0^0 \quad s.t. \quad ||D\alpha - y||_2^2 \quad \leq \quad \varepsilon^2$$

Set $L = 1$

Gather all support $\{Si\}_i$ of cardinality

Solve the LS problem

$$\min_{\alpha} ||D\alpha - y||_2^2 \quad s.t. \quad \text{supp}(\alpha) = S_i$$

LS error $\leq \varepsilon^2$

If: $K = 1000$ (dictionary elements), $L = 10$ (sparsity), 1 nano-sec for LS, then this would take 2.52288×10^{23} nano-seconds or 8×10^6 years. (1000 choose 10)

2.5 BASIS REPRESENTATION

A basis consists of N linearly independent vectors $\{v_i\}, i = 1, \ldots, N$

A basis representation of a signal $x(t)$ breaks the signal into discrete elements, "atoms," which form a linear combination of *basis* signals $\psi_{\gamma\in\Gamma}$, characterized by expansion coefficients α:

$$x(t) = \sum_{\gamma\in\Gamma} \alpha(\gamma)\psi_\gamma(t) \tag{2.52}$$

Where Γ is a discrete index set $(\mathbb{Z}, \mathbb{N}, \mathbb{Z}\times\mathbb{Z}, \mathbb{N}\times\mathbb{Z})$...

With a discrete series of inner products between the signal and each of the basis functions, we linearly translate the signal into a list of coefficients, α, breaking the signal up into manageable "chunks" that are easier to compute and have semantic information about which frequencies are in the signal.[25]

These discrete representations uniquely describe any signal and have a number of advantages including allowing inputs to and outputs from digital computers.

The signal, translated linearly into into a discrete list of numbers, is mapped in such a way that it can be reconstructed (i.e. the translation is lossless). The linear transform is a series of inner products. If $x(t)$ is smooth, the magnitudes of the coefficients fall off quickly as k increases, and this energy compaction provides a kind of implicit compression.[25]

The transformation can be shown as:

$$(x) \rightarrow \left\{ \begin{array}{c} \langle x(t), \psi_1(t) \rangle \\ \langle x(t), \psi_2(t) \rangle \\ \vdots \\ \langle x(t), \psi_\gamma(t) \rangle \end{array} \right\} \tag{2.53}$$

for some fixed set of signals

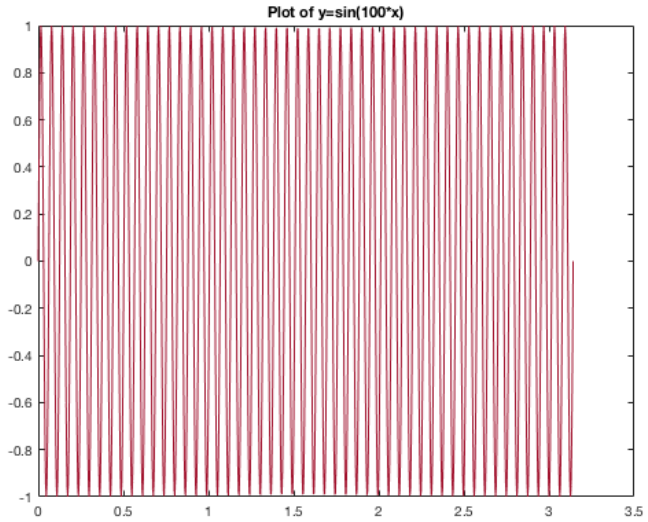
$$\{\psi_\gamma(t)\}_{\gamma \in \Gamma} \tag{2.54}$$

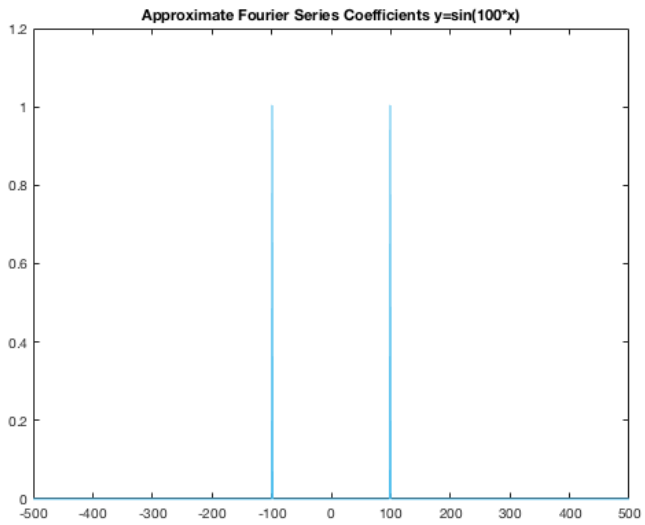
Fourier series Let us choose a familiar basis of harmonic, complex sinusoids to represent the signal $x(t) \in L_2([0,1])$:

$$x(t) = \sum_{k \in \mathbb{Z}} \alpha(k) e^{j2\pi kt} \tag{2.55}$$

```
fmax=500;     % Max frequency
N=1000;       % Number of steps

h=pi/(N-1);   % Step Size

s=ones(1,N);  % Simpson's rule coefficients
s(2:2:N-1)=4; % Quadrutple count the evens
s(3:2:N-2)=2; % Double count the odds
s=s*h/3;      % Multiply by the step size and scale by a third

[n,x]=meshgrid(-fmax:fmax,0:h:pi); % Set up matricies
f = exp(-i*x.*n);
```

```
y = sin(100*x);
fsc = s * (y .* f); % Fourier series coefficients
fsc=fsc.*(fsc>1);
plot(x,y)
title('Plot of y=sin(100*x)')
set(gcf,'color','w')
snapnow;
plot(n,abs(fsc)); % Plot spectrum
title('Approximate Fourier Series Coefficients y=sin(100*x)')
set(gcf,'color','w')
snapnow;
```






```
end
```

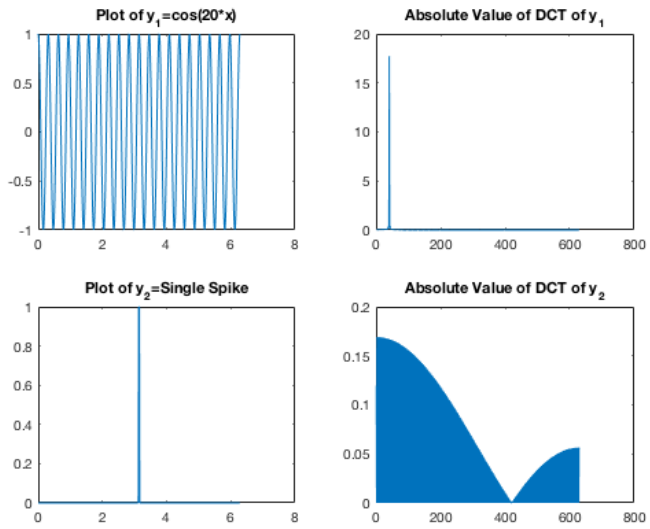
$$\sin(z) = \frac{\exp(iz) - \exp(iz)}{2\imath} \tag{2.56}$$

The Fourier transform of a continuous time signal $x(t)$ is denoted by $\hat{x}(\omega)$, and is given by

$$\hat{x}(\omega) = \int x(t) e^{-j\omega t} dt \tag{2.57}$$

```
x=0:0.01:2*pi;

y=cos(20*x);

z=dct(y); %discrete cosine transform

y2=0*x;
```

```
y2(end/2-1:end/2+1)=1;


z=dct(y2);



subplot(221)
plot(x,y)
title('Plot of y_1=cos(20*x)')

subplot(222)
plot(abs(z))
title('Absolute Value of DCT of y_1')

subplot(223)
plot(x,y2)
title('Plot of y_2=Single Spike')

subplot(224)
plot(abs(z))
title('Absolute Value of DCT of y_2')
```

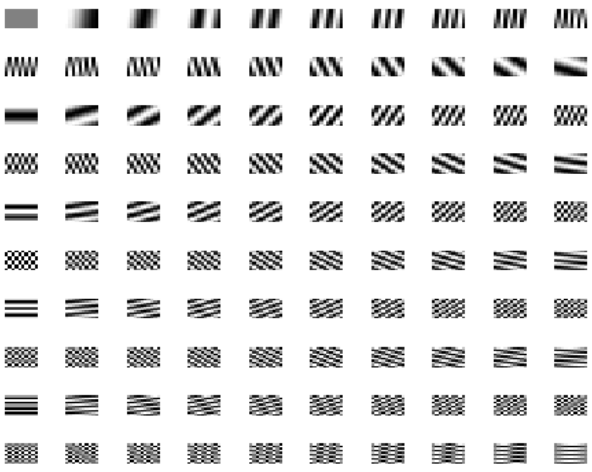



```
function Hahn_Thesis_DCT()
A1 = imread('cameraman.tiff'); %Load Image
A = A1([125:134],[145:154]); % Select patch

x = im2double(A(:)); %Vectorize Patch
n = length(x);

D = idct(eye(n)); %Create DCT Basis

a=D*x;  %Analysis Operator

y=D'*a; %Synthesis Operator

sum(x-y)  %Reconstruction Error
```

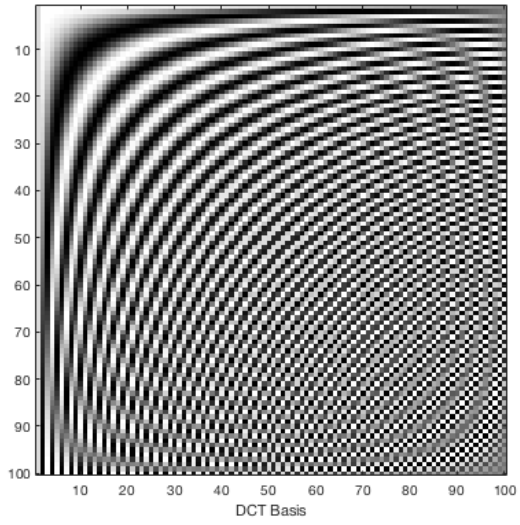
```
%Plots
figure('name','Hahn DCT 2D Demo  Basis Functions')
for i=1:100
    subplot(10,10,i)
    imagesc(reshape(D(:,i),10,10))
    axis off
    colormap(gray)

end

set(gcf,'color','w')
snapnow;

figure('name','Hahn DCT 2D Demo Patch Reconstruction')

imagesc(A1)
xlabel('Full Image')
axis image
colormap(gray)
set(gcf,'color','w')
snapnow;

imagesc(D)
xlabel('DCT Basis')
axis image
colormap(gray)
```

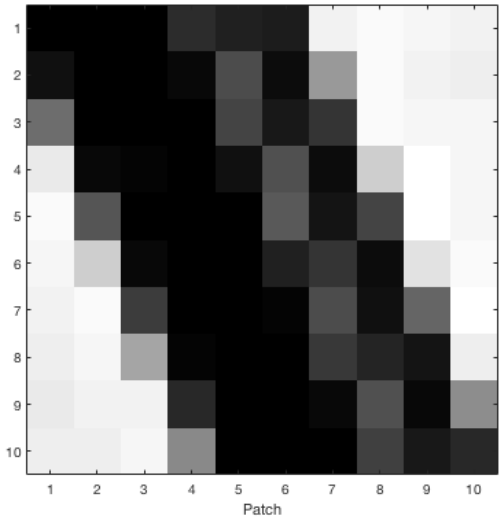
```
set(gcf,'color','w')
snapnow;

imagesc(A)
xlabel('Patch')
axis image
colormap(gray)
set(gcf,'color','w')
snapnow;

imagesc(reshape(x,10,10))
xlabel('Vectorized Patch Reshaped')
axis image
colormap(gray)
set(gcf,'color','w')
snapnow;

imagesc(reshape(y,10,10))
xlabel('DCT Reconstructed Patch')
axis image
colormap(gray)
set(gcf,'color','w')
snapnow;

figure('name','Hahn DCT 2D Demo Coefficients')
bar(a)
```

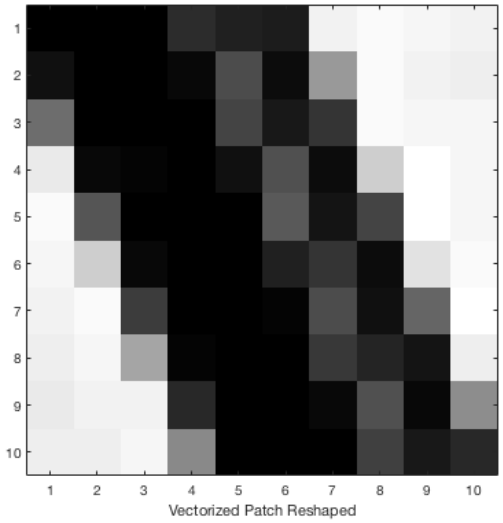
```
set(gcf,'color','w')
snapnow;

end

ans =

  -4.4825e-15
```

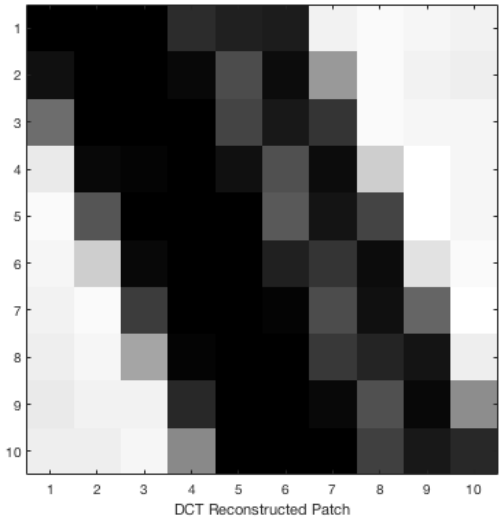


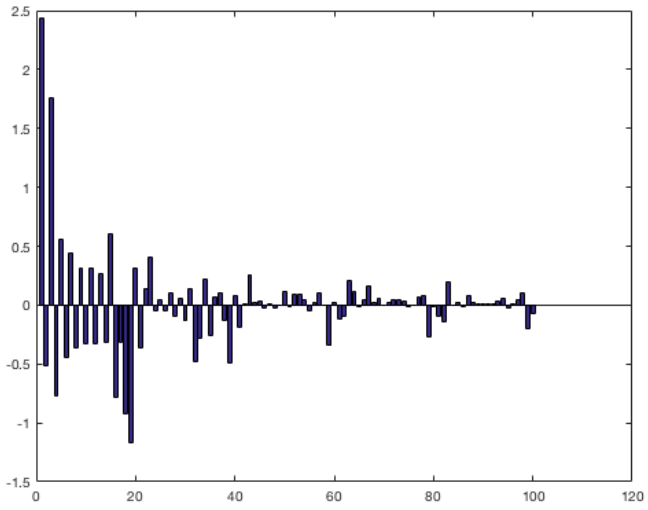

50
100
150
200
250
50
100
150
200
250
Full Image


10
20
30
40
50
60
70
80
90
100
10
20
30
40
50
60
70
80
90
100
DCT Basis


1
2
3
4
5
6
7
8
9
10
1 2 3 4 5 6 7 8 9 10
Patch


1
2
3
4
5
6
7
8
9
10
1 2 3 4 5 6 7 8 9 10
Vectorized Patch Reshaped

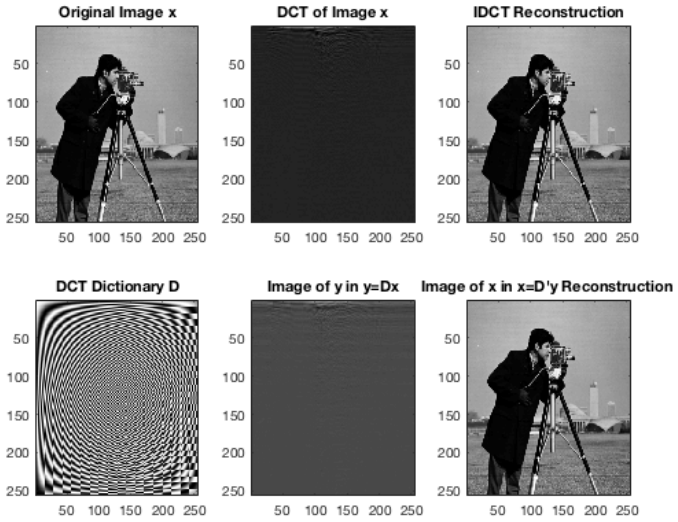





```
function HahnDCT_Image

clc;

close all;

clear;


cd('/Users/williamedwardhahn/Documents/MATLAB')
```

```
I = double(imread('cameraman.tiff'));

I2=dct(I);

I3=idct(I2);

D=idct(eye(size(I)));

y=D*I;

x=D'*y;



subplot(231)
imagesc(I);
colormap(gray)
title('Original Image x')

subplot(232)
imagesc(I2);
colormap(gray)
title('DCT of Image x')

subplot(233)
imagesc(I3);
```

```
colormap(gray)
title('IDCT Reconstruction')

subplot(234)
imagesc(D);
colormap(gray)
title('DCT Dictionary D')

subplot(235)
imagesc(y);
colormap(gray)
title('Image of y in y=Dx')

subplot(236)
imagesc(x);
colormap(gray)
title('Image of x in x=D''y Reconstruction ')

end
```



Remember that ωt is unit-less. Here $\omega = 2\pi k$. Therefore k is the frequency, since $f = \omega/2\pi$. The α coefficients are found as the inner product of the signal with the basis functions:

$$\begin{aligned} \alpha(k) &= \langle\, x(t),\ e^{-j'2\pi kt} \,\rangle = \\ \int_0^1 x(t)e^{-j'2\pi kt}dtx(t) &= \sum_{k\in\mathbb{Z}} \left(\int_0^1 x(t)e^{-j'2\pi kt}dt \right) e^{j2\pi kt} = \\ \sum_{k\in\mathbb{Z}} \left(x(t)e^{-j'2\pi kt}dt \right) e^{j2\pi kt} & \end{aligned}$$

(2.57)

Where

$$\int_0^1 e^{-j'2\pi kt} e^{j2\pi kt} dt = A2\pi\delta(j - j')$$

Notice the coefficients are different for each basis function k, but are constant in time (during the interval integrated over.) Neither the signal nor the basis functions (harmonic oscillators) are constant in time. The inner product of the signal with each basis function returns a constant number.

In this basis representation, representations of smooth functions have a naturally sparse property; as k increases, the magnitudes of $|\alpha(k)|$ drop off. Justin Romberg calls this an energy compaction with implicit compression.[25]

The Fourier series (harmonic oscillators) is an example of an orthonormal basis.

Two conditions must be met:

First, the inner product of each basis function with every other basis function is zero (orthogonal condition), except for its inner product with itself, which is one (normal condition):

$$\langle \psi_\gamma,\ \psi_{\gamma'} \rangle = \begin{cases} 1 & \text{if } \gamma = \gamma' \\ 0 & \text{if } \gamma \neq \gamma' \end{cases} \tag{2.59}$$

The basis also must span the space H.

$$\text{span}\{\psi_\gamma\}_{\gamma\in\Gamma} = H \tag{2.60}$$

for finite dimensions there is no $x \in H$ such that $\langle \psi_\gamma, x \rangle = 0$ for all $\gamma \in \Gamma$

Non-orthogonal bases in $\mathbb{R}^N$

When $x \in \mathbb{R}^N$, basis representations fall squarely into the realm of linear algebra.

Let $\psi_0, \psi_1, \ldots \psi_{N-1}$ be a set of N linearly independent vectors in R^N. Since the ψ_k are linearly independent, then every $x \in \mathbb{R}^N$ produces a unique sequence of inner products against the $\{\psi_k\}$. That is we can recover x from a sequence of inner products

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{N-1} \end{bmatrix} = \begin{bmatrix} \langle x, \psi_0 \rangle \\ \langle x, \psi_1 \rangle \\ \vdots \\ \langle x, \psi_{N-1} \rangle \end{bmatrix} \tag{2.61}$$

we know that Ψ^* is invertible since it is square and its rows are linearly independent Orthobases are nice since they not only allow every signal to be decomposed as a linear combination of elements, but we have a simple and explicit way of computing the coefficients (the $\alpha(\gamma) = \langle x, \psi_\gamma \rangle$) in this expansion. Associated with an orthobasis $\{\psi_\gamma\}\gamma \in \Gamma$ for a space H are two linear operators. The first operator

$$\Psi^* : H \to \ell_2(\Gamma) \tag{2.62}$$

maps the signal x(t) in H to the sequence of expansion coefficients in $\ell_2(\Gamma)$. if H is finite dimensional, it may be more appropriate to write the range of this mapping as R^N rather than $\ell_2(\Gamma)$. The mapping Ψ^* is called the analysis operator, and its action is given by

$$\Psi^*[x(t)] = \{\langle x(t), \psi_\gamma(t) \rangle\}_{\gamma \in \Gamma} = \{\alpha(\gamma)\}_{\gamma \in \Gamma} \tag{2.63}$$

The second operator $\Psi : \ell_2(\Gamma) \to H$ takes a sequence of coefficients in $\ell_2(\Gamma)$ and uses them to build up a signal.

The mapping Ψ is called the synthesis operator, and its action is given by

$$\Psi\left[\{\alpha(\gamma)\}_{\gamma\in\Gamma}\right] = \sum_{\gamma\in\Gamma} \alpha(\gamma)\psi_\gamma(t) \tag{2.64}$$

Ψ and Ψ^* can be represented as matrices where the basis functions $\psi_\gamma(t)$ are the columns of Ψ and rows of Ψ^*.

Sampling a bandlimited signal:

Suppose that the frequency of the signal, is *bandlimited* to $[-\pi/T, \pi/T]$. Outside of this frequency domain the signal is set to zero. A Fourier transform (the inner product) translates the signal into the frequency domain and:

$$\hat{x}(\omega) = \int x(t)e^{-j\omega t}dt = 0 \tag{2.65}$$

for $|\omega| > \pi/T$

$$x[n] = x(nT) \tag{2.66}$$

$$x(t) = \sum_{n=-\infty}^{\infty} x[n]\frac{\sin(\pi(t-nT))}{\pi(t-nT)/T}$$

(2.66)

"Shannon-Nyquist sampling theorem tells us that we can reconstruct $x(t)$ from point samples are equally spaced by T" [25].

Suppose that $x(t)$ is bandlimited to $[-\pi/T, \pi/T]$:

$$\hat{x}(\omega) = \int x(t)e^{-jwt}dt = 0 \quad \text{for } |\omega| > \pi/T \tag{2.67}$$

Then the Shannon-Nyquist sampling theorem tells us that we can reconstruct x(t) from point samples that are equally spaced by T :

$$x[n] = x(nT) \tag{2.68}$$

$$x(t) = \sum_{n=-\infty}^{\infty} x[n]\frac{\sin(\pi(t-nT))}{\pi(t-nT)/T} \tag{2.69}$$

We can re-interpret this as a basis decomposition

$$x(t) = \sum_{n=\infty}^{\infty} \alpha(n)\psi_n(t) \tag{2.70}$$

with

$$\psi_n(t) = \sqrt{T}\frac{\sin(\pi(t-nT))}{\pi(t-nT)} \tag{2.71}$$

$$\alpha(n) = \sqrt{T}x(nT) \tag{2.72}$$

If x(t) is band-limited, then

$$\alpha(n) = \sqrt{T}x(nT) = \frac{\sqrt{T}}{2\pi}\int_{-\pi/T}^{\pi/T} \hat{x}(w)e^{j\omega nT}d\omega = \frac{1}{2\pi}\langle \hat{x}(\omega), \hat{\psi}(\omega)\rangle \tag{2.73}$$

Expansion on basis

$$u = \sum_i = c_i v_i \tag{2.74}$$

Orthonormal basis

$$\langle v_i, v_j \rangle = \begin{cases} 1, & i = j \\ 0, & otherwise \end{cases} \tag{2.75}$$

Expansion coefficients

$$\langle v_i, \mathbf{u} \rangle = c_i \tag{2.76}$$

Expansion

$$\mathbf{u} = \sum_i \langle v_i, \mathbf{u} \rangle v_i \tag{2.77}$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = u_1 \,[100$$

$+\mathrm{u}_2\,[01 + u_3\,[1 \quad (2.78)$

$$\mathbf{u} = \sum_i u_i \mathbf{e_i} = \sum_i \langle e_i, \mathbf{u} \rangle \mathbf{e_i} \tag{2.79}$$

We will use $L_2(\mathbb{R})$ to denote the space of finite-energy signals:

$$x(t) \in L_2(\mathbb{R}) \Leftrightarrow \int_{-\infty}^{\infty} |x(t)|^2 dt < \infty \tag{2.80}$$

Similarly $L_2([a,b])$ is the space of finite-energy signals that are either supported on (i.e. zero outside of) [a, b] or are periodic with period b ? a:

$$x(t) \in L_2([a,b]) \Leftrightarrow \int_{-\infty}^{\infty} |x(t)|^2 dt < \infty \tag{2.81}$$

We will use $\ell_2(\mathbb{Z})$ to denote the space of discrete signals which are square-summable:

$$x[n] \in \ell_2 \Leftrightarrow \sum_{n=-\infty}^{\infty} |x[n]|^2 < \infty \tag{2.82}$$

The space of discrete signals which have finite length N are simply vectors in $\mathbb{R}^N$

We will occasionally use the term Hilbert space to mean a collection of signals that is closed under addition, and for which we have an inner product defined, namely the "unified language of functional analysis and Hilbert spaces. In the finite case, this reduces to basic linear algebra."[25]

We will use the generic notation $||\cdot||_2$ for the (standard Euclidean) norm induced by the standard inner product. If x(t) is a continuous-time signal, then

$$||x||_2^2 = \int |x(t)|^2 dt \tag{2.83}$$

and if $x[n]$ is a sequence/vector, then

$$||x||_2^2 = \sum_{\gamma \in \Gamma} |x(\gamma)|^2 \tag{2.84}$$

We will also make extensive use of the ℓ_1 and ℓ_∞ norm of a finite vector over Γ. These norms are defined as

$$||x||_1 = \sum_{\gamma \in \Gamma} |x[\gamma]| \tag{2.85}$$

$$||x||_\infty = \max_{\gamma \in \Gamma} |x[\gamma]| \tag{2.86}$$

Orthobasis $\{\psi_i\}$

$$\langle y, \psi_i \rangle = \langle f, \psi_i \rangle + \langle z, \psi_i \rangle \tag{2.87}$$

$$\tilde{y}_i = \alpha_i + z_i \tag{2.88}$$

z_i Gaussian white noise sequence σ noise level

$$\alpha_i = \langle f, \psi_i \rangle \tag{2.89}$$

coordinates of f

Classical model: signal of interest f is lowpass Observable frequencies: $0 \leq \omega \leq \Omega$ $\hat{f}(w)$ is nonzero only for $\omega \leq B$

Normalization is a "canonical neural computation."[27]

Define the ℓ^p norm of the vector to be

$$||x||_p = \left(\sum_m |x_m|^p \right)^{(1/p)}$$

Define the inner product between x and y to be

$$\langle x, y \rangle = \sum_m x_m y_m$$

Sparse Coding of Images

$$I(x, y) = \sum_i a_i \phi_i(x, y)$$

Vector Norms:

The ℓ_p^N norm of the vector $x \in \mathbb{R}^N$ is given by

$$||x||\ell_p := ||x||\ell_p^N := \begin{cases} \left(\sum_{j=1}^{N} |x_j|^p\right)^{1/p} & \text{for} \quad 0 < p < \infty \\ \\ \max_{j=1\ldots N} |x_j| & \text{for} \quad p = \infty \end{cases}$$

2.6 SPARSE REPRESENTATION

Having a sparse representation plays a fundamental role in how well we can, compress, denoise, restore signals and images.

Signal/image f(t) in the time/spatial domain Decompose f as a superposition of atoms

$$f(t) = \sum_i \alpha_i \psi_i(t)$$

ψ_i = Basis Functions

α_i = Expansion Coefficients

Classical example: Fourier series

ψ_i = complex sinusoids α_i = Fourier Coefficients

Modern example: wavelets

ψ_i = "little waves" α_i = wavelet Coefficients

Exotic example: curvelets

Frame operators $\Psi, \tilde{\Psi}$

map images to sequences and back

Two sequences of functions: $\{\psi_i(t)\}, \{\tilde{\psi}(t)\}$

Analysis (inner products):

$$\alpha = \tilde{\Psi}^*[f]$$

$$\alpha_i = \langle \tilde{\psi}_i, f \rangle$$

Synthesis (superposition):

$$f = \Psi[\alpha]$$

$$f = \sum_i \alpha_i \psi_i(t)$$

Length and Angle Preservation:

if $\{\psi_i(t)\}$ is an orthobasis, then

$$||\alpha||_{\ell_2}^2 = ||f||_{\ell_2}^2$$

$$\sum_i \alpha_i \beta_i = \int f(t) g(t) dt$$

$$\beta = \tilde{\Psi}[g]$$

$$\psi_i(t) = \tilde{\psi}_i(t)$$

i.e. all sizes and angles are preserved. Overcomplete tight frames have similar properties. This property of (approximately) preserving angles and distances is key to the results in compressed sensing and sparse recovery, in particular the notions of the Restricted Isometry Property and the Johnson Lindenstrauss lemma as will be seen the the subsequent chapters.

Linear S-term approximation: keep S coefficients in fixed locations

$$f_S(t) = \sum_{m=1}^{S} \alpha_m \psi_m(t) \tag{2.90}$$

$$|\alpha_m| \lesssim m^{-r} \Rightarrow ||f - f_S||_2^2 \lesssim S^{-2r+1}$$

(2.91)

Take $f(t)$ periodic, d-times continuously differentiable, Ψ= Fourier series:

$$||f - f_S||_2^2 \lesssim S^{-2d} \tag{2.92}$$

The smoother the function, the better the approximation. "A smooth function is a function that has continuous derivatives up to some desired order over some domain. The number of continuous derivatives necessary for a function to be considered smooth depends on the problem at hand, and may vary from two to infinity. "[163]

Nonlinear S-term approximation: keep S largest coefficients

$$f_S(t) = \sum_{\gamma \in \Gamma_S} \alpha_\gamma \psi_\gamma(t) \tag{2.93}$$

Fast decay of sorted coefficients $\Rightarrow$ good approximation

Γ_S = location of S largest $|\alpha_m|$

$$|\alpha|_{(m)} \lesssim m^{-r} \Rightarrow ||f - f_S||_2^2 \lesssim S^{-2r+1} \tag{2.94}$$

$|\alpha|_{(m)} = m^{th}$ largest coefficient

Total Variation: An image x is said to be K-sparse in gradient if

$$\begin{aligned} ||TV(x)||_0 &= K \\ TV(x)_{n,m} &= \sqrt{p(n,m)^2 + q(n,m)^2} \\ p(n,m)x(n,m+1) - x(n,m) & \\ q(n,m) &= x(n+1,m) - x(n,m) \end{aligned} \tag{2.95}$$

Total variation formulation of sparse least squares is given by

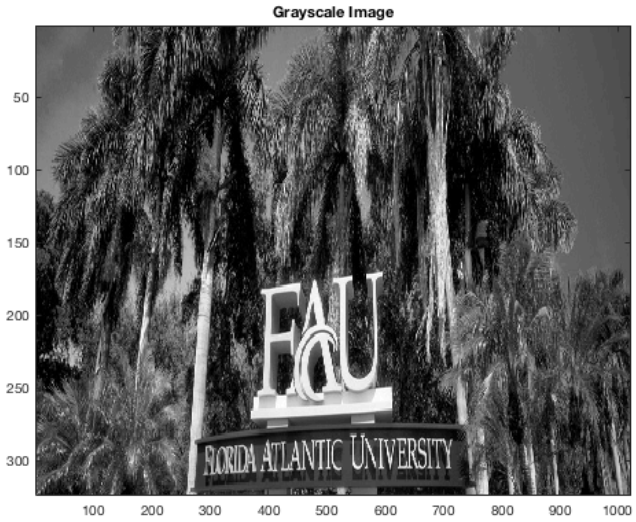
$$E = ||\mathbf{I} - \mathbf{\Phi} \cdot T(\mathbf{v})||_2^2 \; + \; \lambda ||T(\mathbf{v})||_0 \qquad \text{reconstruction} \quad + \quad \text{Sparsity}$$

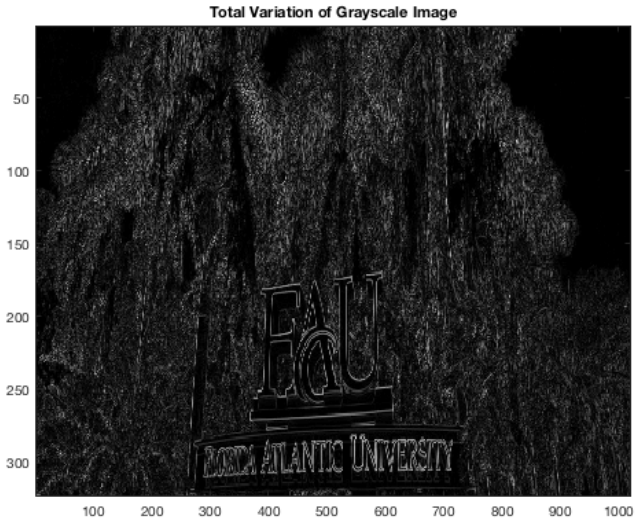
```
function Hahn_Thesis_Total_Variation

[X, map]=imread('https://www.fau.edu/explore/images/fau.jpg','jpg');

figure(7)
imagesc(X)
set(gcf,'color','w')
title('Color Image')
snapnow;

Iw1=rgb2gray(im2double(X));

p=Iw1(:,2:end)-Iw1(:,1:end-1);

q=Iw1(2:end,:)-Iw1(1:end-1,:);
```

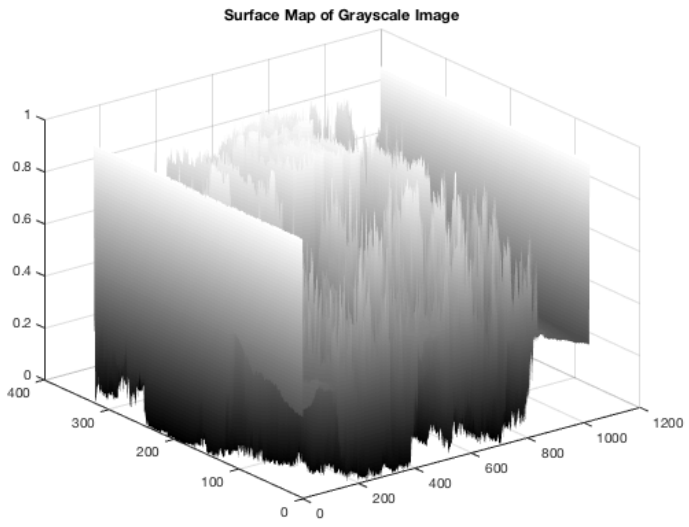
```
[pn, pm]=size(p);

[qn, qm]=size(q);

n=min(pn,qn);
m=min(pm,qm);

p=p(1:n,1:m);
q=q(1:n,1:m);

TV=sqrt(p.^2+q.^2);

sum(sum(TV>0.01))



figure(1)
imagesc(Iw1)
colormap(gray)
shading interp
set(gcf,'color','w')
title('Grayscale Image')
snapnow;

figure(2)
imagesc(TV)
colormap(gray)
```

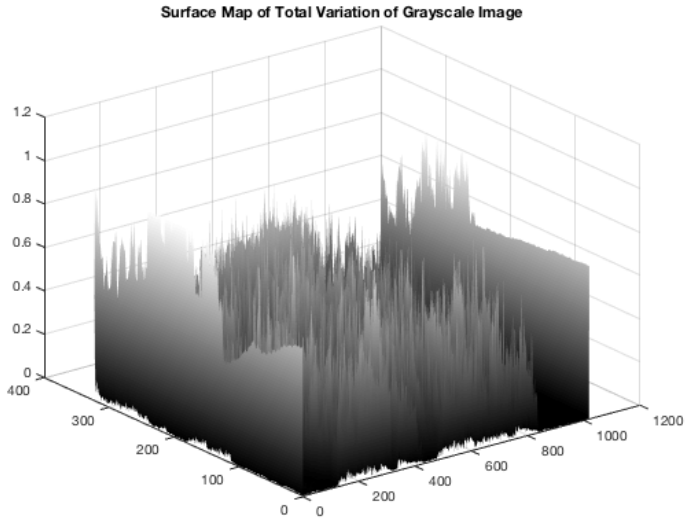
```
shading interp
set(gcf,'color','w')
title('Total Variation of Grayscale Image')
snapnow;

figure(3)
surf(Iw1)
colormap(gray)
shading interp
set(gcf,'color','w')
title('Surface Map of Grayscale Image')
snapnow;

figure(4)
surf(TV)
colormap(gray)
shading interp
set(gcf,'color','w')
title('Surface Map of Total Variation of Grayscale Image')
snapnow;

figure(5)
hist(Iw1(:))
set(gcf,'color','w')
title('Histogram of Grayscale Image')
snapnow;
```

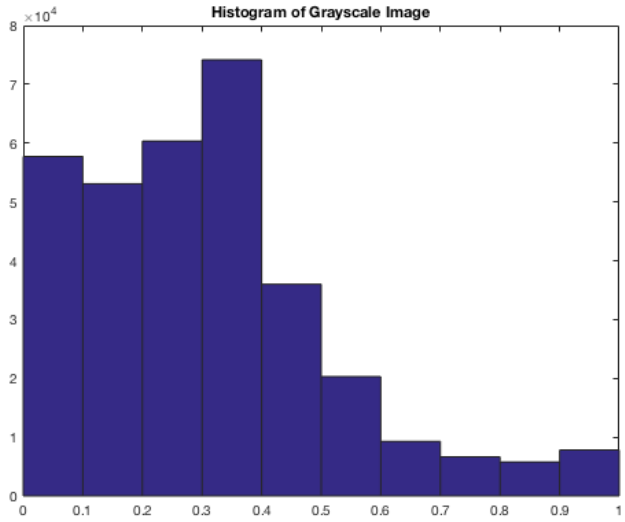
```
figure(6)
hist(TV(:))
set(gcf,'color','w')
title('Histogram of Total Variation Grayscale Image')
snapnow;

end

ans =

      287553
```

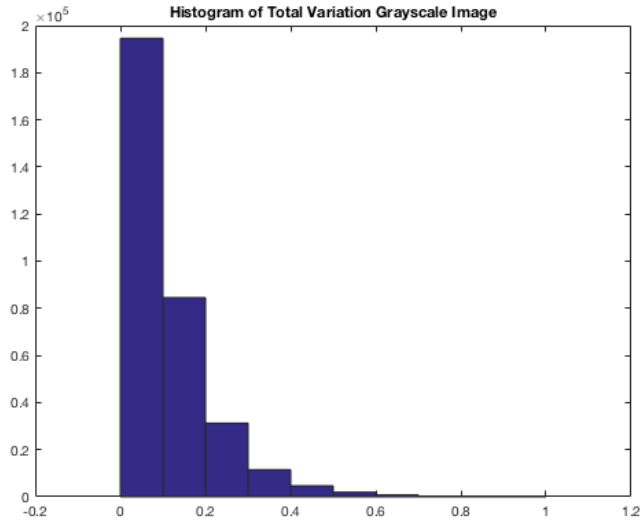

Grayscale Image
50
100
150
200
250
300
100
200
300
400
500
600
700
800
900
1000
FAU
FLORIDA ATLANTIC UNIVERSITY


Total Variation of Grayscale Image
50
100
150
200
250
300
100
200
300
400
500
600
700
800
900
1000
FAU
FLORIDA ATLANTIC UNIVERSITY


Surface Map of Grayscale Image
1
0.8
0.6
0.4
0.2
0
400
300
200
100
0
0
200
400
600
800
1000
1200


Surface Map of Total Variation of Grayscale Image
1.2
1
0.8
0.6
0.4
0.2
0
400
300
200
100
0
0
200
400
600
800
1000
1200





2.7 SPARSE CODING

$$_{z\in\mathbb{R}^K}||Wz - x||^2 + \lambda||z||_1 \tag{2.96}$$

z is the code (want)

x is the data (have)

W is adapted

Encoding requires optimization

Decoding is trivial

For simplicity assume that the basis is orthonormal. Using the $N \times N$ basis matrix $\psi = [\psi_1|\psi_2|,\ldots_N]$ with the vectors $\{\psi_i\}$ as columns, a signal can be expressed as

$$\mathbf{x} = \sum_{i=1}^{N} s_i \psi_i \tag{2.97}$$

$$\mathbf{x} = \psi s \tag{2.98}$$

where s is the $N \times 1$ column vector of weighting coefficients $s_i = \langle \mathbf{x}, \psi_i \rangle = \psi_i^T \mathbf{x}$ where ^{T} represents transposition.

$\mathbf{x}$ and $\mathbf{s}$ are equivalent representations of the signal, where $\mathbf{x}$ is in the time or space domain and $\mathbf{s}$ is in the ψ domain

The signal is K-sparse if it is a linear combination of only K basis vectors

only K of the s_i coefficients are nonzero and $(N - K)$ are zero

Consider a real-valued, finite-length, one-dimensional, discrete-time signal x, which can be viewed as an N column vector in $\mathbb{R}^N$ with elements $x[n]$, $n = 1, 2, \ldots, N$.

Images and higher-dimensional data are vectorized it into a one-dimensional vector.

Any signal in $\mathbb{R}^N$ can be represented in terms of a basis of $N \times 1$ vectors $\{\psi_i\}_{i=1}^N$.

Information scalability: detection¡ classification ¡ estimation ¡ reconstruction. Compressive measurements sufficient statistics. Enables new sensing architectures and modalities most useful when measurements are expensive[39]

2.8 COMPRESSED SENSING

"You can look at an algorithm as just a big compression scheme." [1]

Quickly building models and perceptions of the environment is an extraordinary capacity of a biological network of neurons that must depend on compression. Only compression could enable the otherwise computationally impossible performances of the brain. A sensing of the environment is to measure, and a compressed sensing of the environment is to sparsely acquire and under-sample, so as to minimize time and energy spent, while maximizing the information gained.

Sparse coding and other compression techniques demonstrate that it is possible to represent and transmit structured data in a much more efficient manner than non-structured data, which is subject to the Shannon/Nyquist constraint. Nevertheless, most imaging systems do not treat these cases differently at the front/sensing end but only after first performing non-efficient sampling and then discarding much of the data via compression algorithms. Compressed (or compressive) sensing is an approach that uses the structure inherent in data to reduce the sampling rate at the front end of the system.

Compressed sensing depends on subjecting the to-be encoded data to a random projection and then minimizing...

$$_{z\in\mathbb{R}^K}||Wz - x||^2 + \lambda||z||_1 \tag{2.99}$$

z is the data (want)

x is the code (have)

W is universal

Encoding is trivial

Decoding requires optimization

Compressive sensing "has the potential to impact every field of engineering and applied science that has to do with data acquisition and processing."[**?**]

2.9 SPARSE FILTERING

Intellectual, Genetical, and Cultural Searches "A very typical sort of problem requiring some sort of initiative consists of those of the form Find a number n such that..."[151]

- Sparse filtering is a new feature learning algorithm which is easy to implement and essentially hyperparameter-free.[111]
- Sparse filtering is efficient and scales to handle large input dimensions.hahn
- Sparse filtering works by optimizing exclusively for sparsity in the feature distribution.[111]

Let

$$f_j^{(i)} \tag{2.100}$$

represent the j^{th} feature value (rows) for the i^{th} example (columns), where

$$f_j^{(i)} = \mathbf{w}_j^T \mathbf{x}^{(i)} \tag{2.101}$$

First, normalize the feature distribution matrix by the rows.

Thus, by dividing each feature row by its ℓ_2-norm across all examples, the rows are forced to be equally active (with unit norm.)

$$\widetilde{\mathbf{f}}_j = \frac{\mathbf{f}_j}{||\mathbf{f}_j||_2} \tag{2.102}$$

Next, normalize the matrix by columns. By normalize these features they lie on the ℓ_2-ball, by computing

$$\hat{\mathbf{f}}^{(i)} = \frac{\widetilde{\mathbf{f}}^{(i)}}{||\widetilde{\mathbf{f}}^{(i)}||_2} \tag{2.103}$$

Finally, sum the absolute values of all the entries. The normalized features are optimized for sparseness by using the ℓ_1 penalty. For datasets of M examples we have the sparse filtering objective:

$$\text{minimize} \quad \sum_{i=1}^{M} \left|\left|\hat{\mathbf{f}}^{(i)}\right|\right|_1 = \sum_{i=1}^{M} \left|\left| \frac{\widetilde{\mathbf{f}}^{(i)}}{||\widetilde{\mathbf{f}}^{(i)}||_2} \right|\right|_1 \tag{2.104}$$

The Sparse Filtering Objective Function can be summed up as:

1. Normalize Across Rows
2. Normalize Across Columns
3. Cost Function = Sum of the Normalized Entries

$$\alpha = \Psi^* x \tag{2.105}$$

$$x = \Psi^{*-1} \alpha \tag{2.106}$$

```
function HahnSparseFilteringMNIST
```

	x_1	x_2	x_3	x_4	x_j	x_m
f_1						
f_2						
f_3						
f_4						
f_i						
f_n						

```
load HahnMNISTRawData.mat

who


data1=data1_train_x;


label1=[];


for k=1:size(data1_train_y,1)


    label1=[label1; find(data1_train_y(k,:))-1];


end


filterplot(data1(:,1:100)',[100 256]);

title('Collection of Example MNIST Digits')

set(gcf,'color','w')

snapnow
```

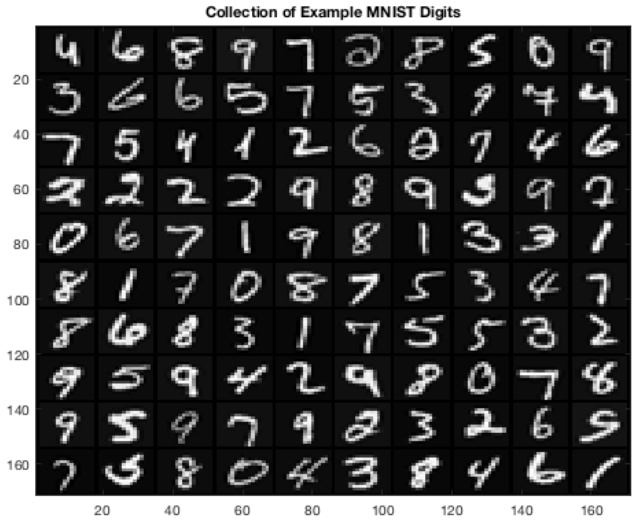
```
for i=200:210

    imagesc(reshape(data1(:,i),[16 16]))
    title('Example MNIST Digits')
    set(gcf,'color','w')
    label1(i);
    snapnow;

end

for j=1:11

    data=[];

    for i=1:6000

        if label1(i)==j-1 | j == 11

            data=[data data1(:,i)];

        end

    end

    m=mean(data);
    X = data-m(ones(1,size(data,1)),:);
```

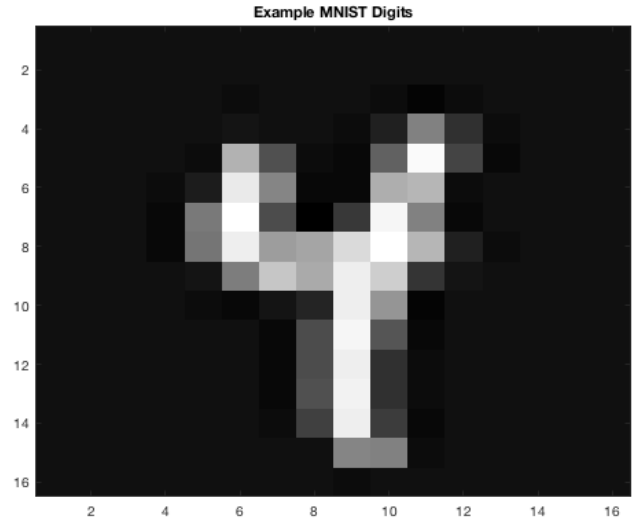
```
nf = 100;  % # Features

N=500;

nm=[nf, size(X, 1)];

W = randn(nm);

W = W(:);

t=0.5;



for l = 2:N

    [f,g] = SF(W,X,nf);

    W = W - t*g;

end

filterplot(W,nm);
title(['Learned Dictionary: ' num2str(j-1)])
set(gcf,'color','w')
drawnow
```

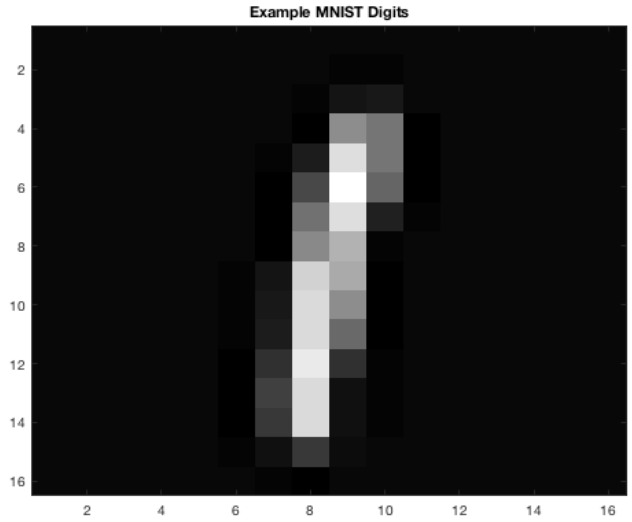
```
    snapnow;

end

end



function [f, g] = SF(W, X, nf)

W = reshape(W, [nf, size(X,1)]);

f = W*X;

fs   = sqrt(f.^2 + 1e-8);

ell2fs = sqrt(sum(fs.^2,2) + 1e-8);

normfs  = fs./ell2fs(:,ones(1,size(fs,2)));

ell2normfs = sqrt(sum(normfs'.^2,2) + 1e-8);

fhat = normfs'./ell2normfs(:,ones(1,size(normfs',2)));

f  = sum(sum(fhat, 2), 1);

a=sum(ones(size(fhat)).*normfs', 2)./(ell2normfs.^2);
```

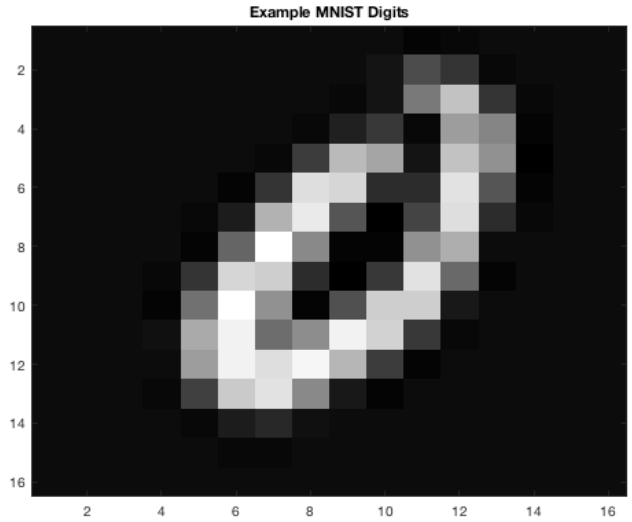
```
g = ones(size(fhat))./ell2normfs(:,ones(1,size(fhat,2))) - fhat.*a(:,ones(1,size(f

a=sum(g'.*fs, 2)./(ell2fs.^2);

g = g'./ell2fs(:,ones(1,size(g',2))) - normfs.*a(:,ones(1,size(normfs,2)));

g = (g .* ((W*X)./fs)) * X';

g = g(:);

end

Your variables are:

data1_test_x   data1_test_y   data1_train_x  data1_train_y


function [D] = filterplot(X,nm)

X = reshape(X, nm);

X = -1 + 2 * ((X - min(min(X)))/(max(max(X)) - min(min(X))));

[m n] = size(X);

w = round(sqrt(n));
```

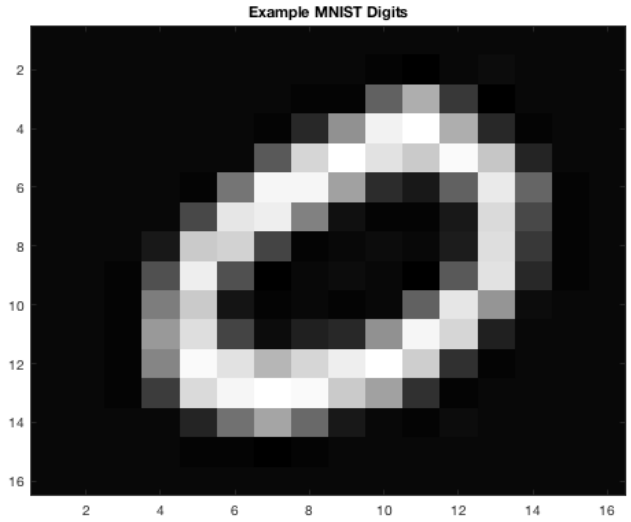
```
h = (n / w);


c = floor(sqrt(m));
r = ceil(m / c);


p = 1;


D = - ones(p + r * (h + p),p + c * (w + p));


k = 1;
for j = 1:r
    for i = 1:c
        D(p + (j - 1) * (h + p) + (1:h), p + (i - 1) * (w + p) + (1:w)) = reshape(
        k = k + 1;
    end


end


imagesc(D)
% surf(D) shading interp
colormap(gray)


end
```

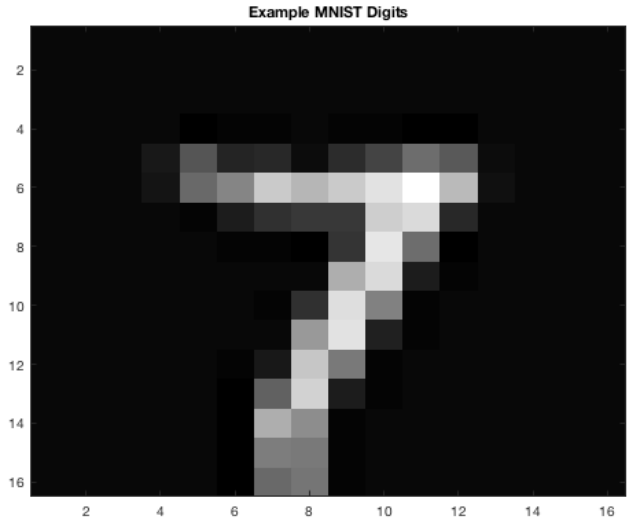

Collection of Example MNIST Digits
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160

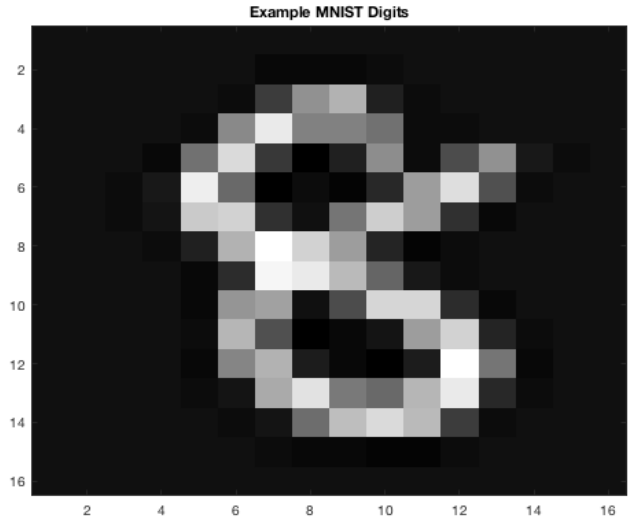

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

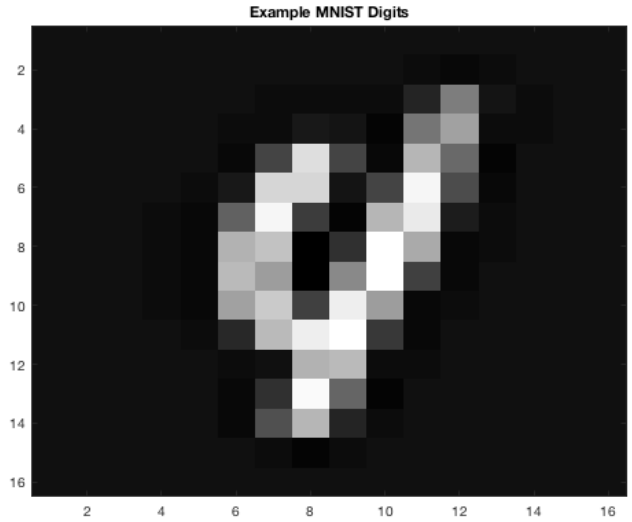

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

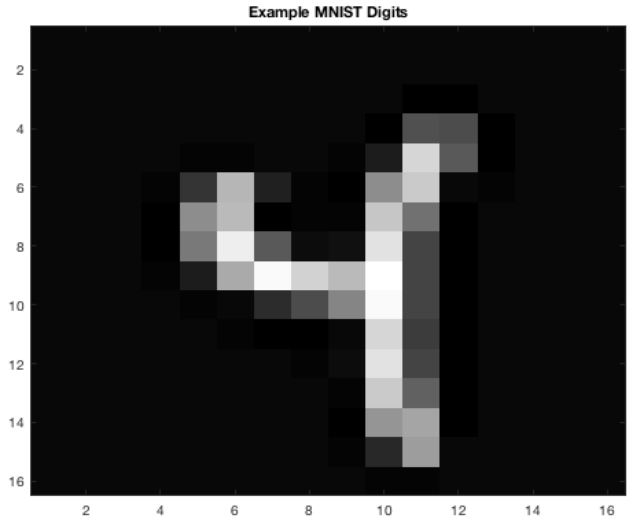

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

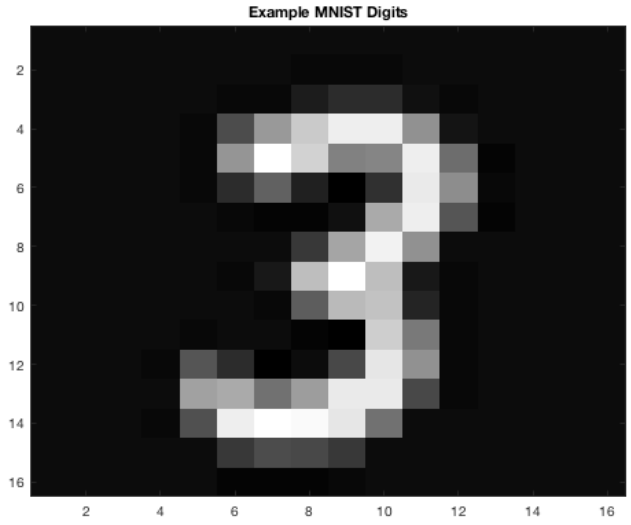

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

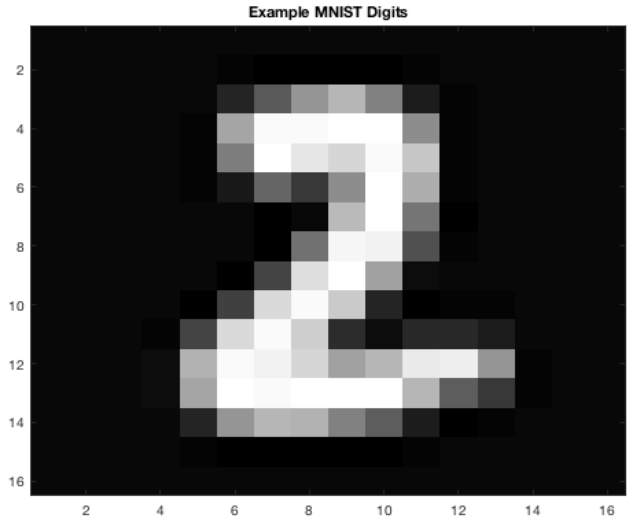

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

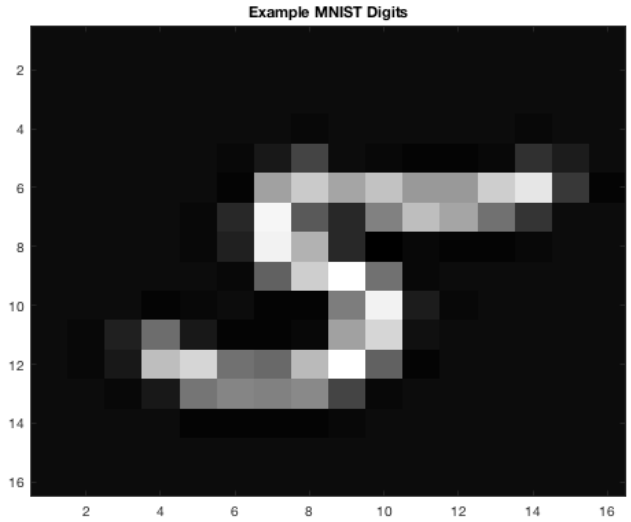

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

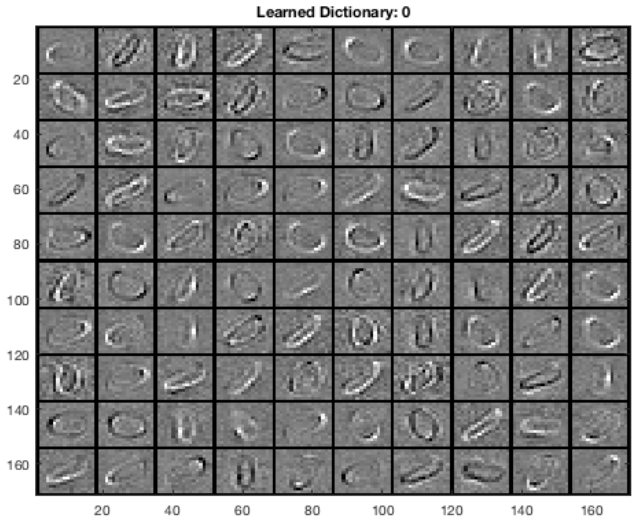

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

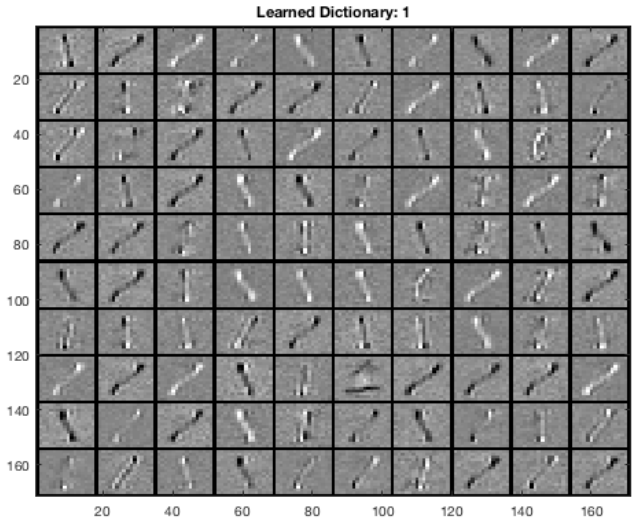

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

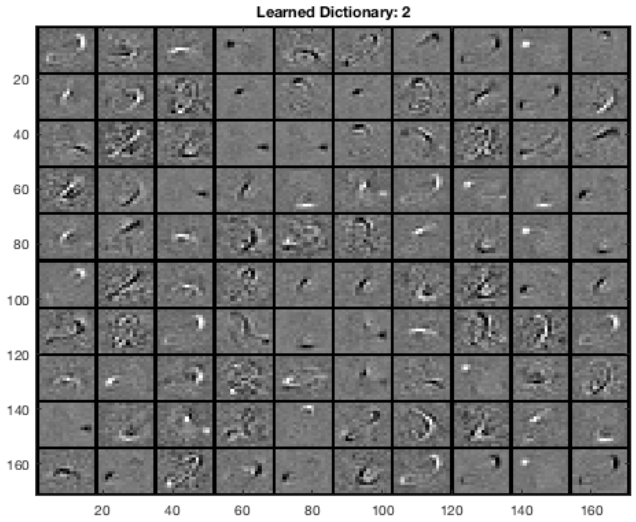

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

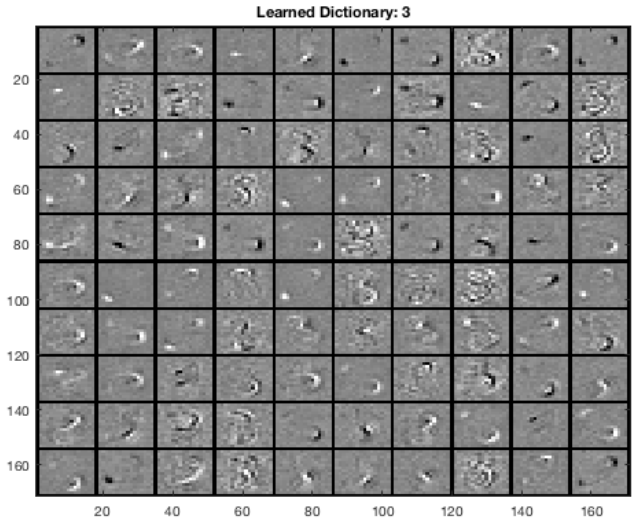

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

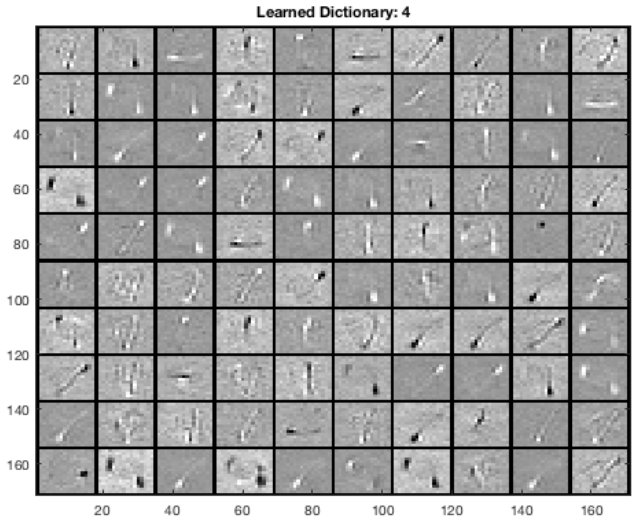

Example MNIST Digits
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16

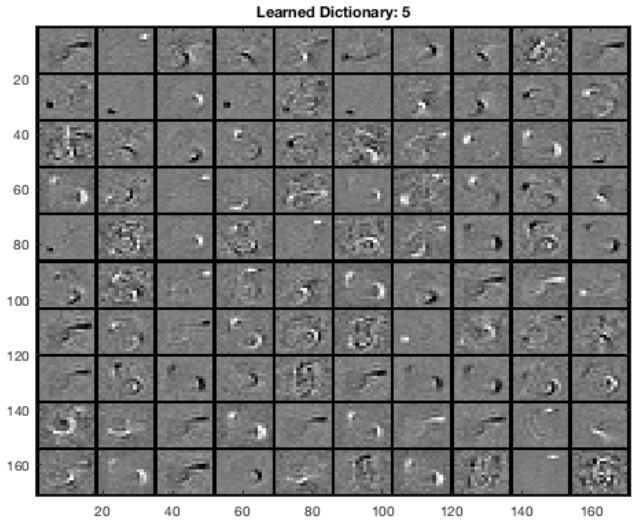

Learned Dictionary: 0
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160

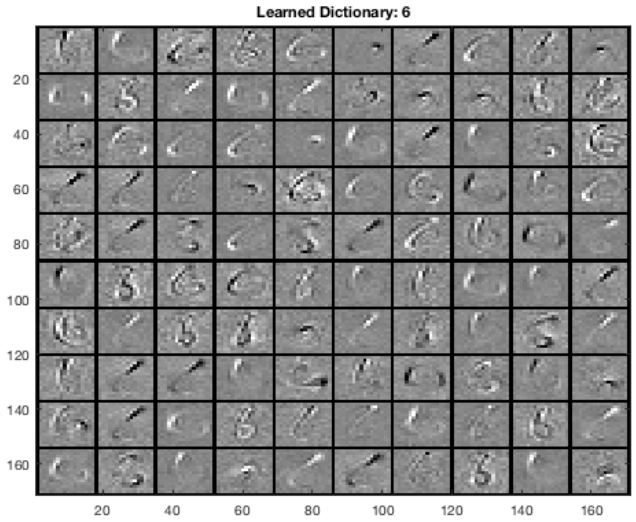

Learned Dictionary: 1
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160

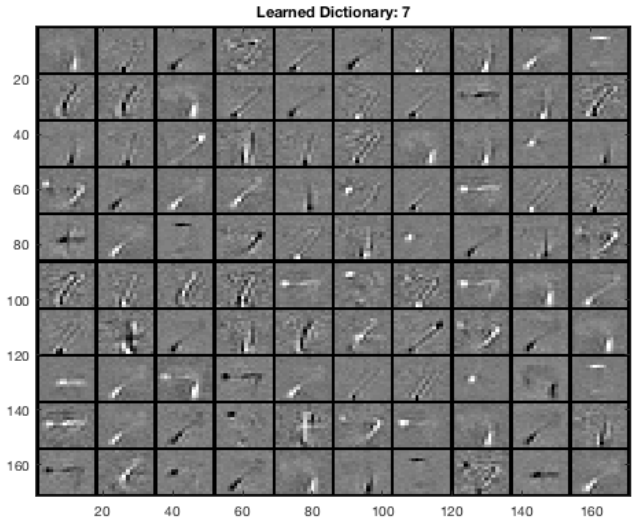

Learned Dictionary: 2
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160

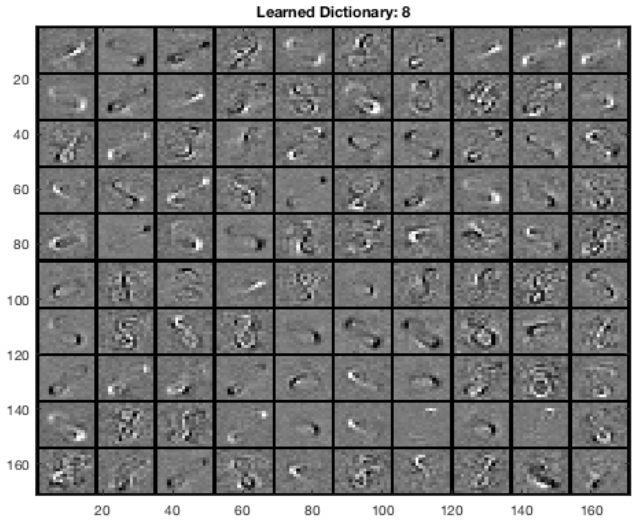

Learned Dictionary: 3
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160

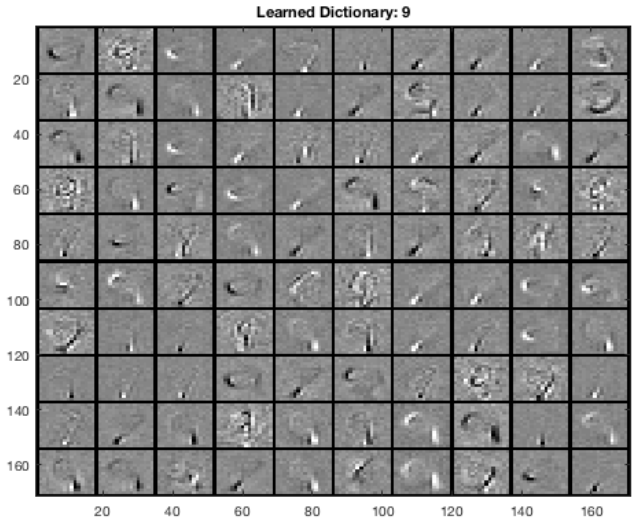

Learned Dictionary: 4
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160

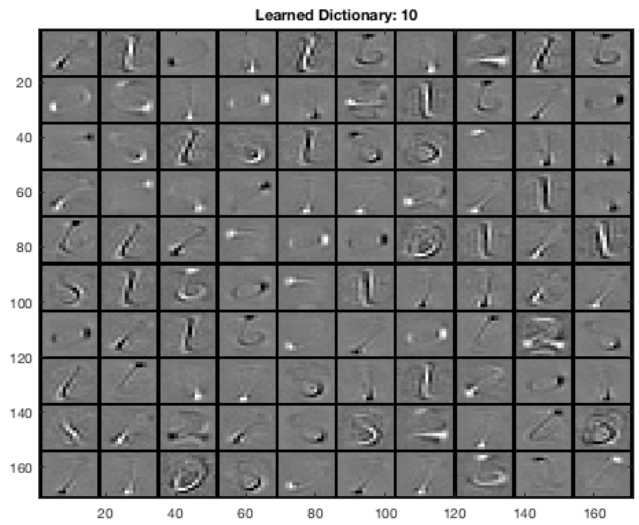

Learned Dictionary: 5
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160


Learned Dictionary: 6
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160


Learned Dictionary: 7
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160


Learned Dictionary: 8
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160


Learned Dictionary: 9
20
40
60
80
100
120
140
160
20
40
60
80
100
120
140
160



2.10 BASIS PURSUIT

```
function [a] = BP(k,D,y)

cvx_begin quiet;

        l=size(D);

        variable a(l(2));

        minimize( norm(D*a-y) );

        subject to;

        norm(a,1) <= k;
```

```
cvx_end;
```

2.11 MATCHING PURSUIT

```
function [a] = MP(k,D,y)

n=size(D);
D1=zeros(n);
r=y;

for k=1:k
    [i,j] = max(abs(D'*r));
    D1(:,j)=D(:,j);
    D(:,j)=0;
    a = D1 \ y;
    r = y - D1*a;

end;
```

2.12 LOCALLY COMPETITIVE ALGORITHMS

```
a=u.*(abs(u) > s);

u = 0.9 * u + 0.01 * (b - G*a);

W = W + (1/(batch_size*10))*((X-W*a)*a');
```

```
function [a, u] = LCA(y, D, lambda)

h=0.01;

u = zeros(size(D,2),1);

for i=1:300

    a = ( u - sign(u).*(lambda) ) .* ( abs(u) > (lambda) );
    u =   u + h * ( D' * ( y - D*a ) - u - a  ) ;
end
```

2.13 BIOLOGICAL MOTIVATION

"It is very important to appreciate that a retina is already thinking about the light." Richard Feynman

Hopfield pointed out that, in many physical systems, collective properties emerge that are robust against changes to model details (e.g., collisions generate sound waves regardless of the atomic configuration) [73]. These properties arise from the nature of the flow in phase space produced by the processing algorithm, which does not appear to be strongly dependent on precise model details [73]. Hopfield suggested that the "computational" ability of large collections of neurons may be a spontaneous consequence of the collective interactions among a large number of simple neurons, citing the neural architecture observed in the brains of higher level organisms as evidence [73]. Most neurons are capable of generating a train of action potentials, propagating pulses of electrochemical activity, when the average potential across their

membrane rises well above its normal resting value [73]. The mean rate at which these action potentials are generated increases with the mean membrane potential in a smooth fashion [73]. The biological information sent from one neuron to another can be attributed to this average firing rate of the presynaptic neuron over a short period. As a result, we can neglect the details of individual action potentials and regard the aforementioned relationship between the mean firing rate and the mean membrane potential of a neuron as that of a smooth input-output [73]. Hopfield concluded that, to understand the spontaneous computational capabilities and emergent collective effects of such a system, we must focus our attention on the nonlinearity of this input-output relationship. [73].

Natural images tend to be highly structured, with strong correlations in neighboring pixel values as well as repetition of spatial and temporal patterns within and across images. Thus, the initial coding of the human visual system (i.e., the photoreceptor array) is highly inefficient because each cell encodes luminance independently. Recent theoretical and experimental advances suggest that a primary goal of the visual system is to compress this initial visual input via a 'sparse code' [115] which can leverage the structure inherent to natural scenes into a more efficient representation. Sparse codes employ a generative model of the visual input with the constraint that only a small number of neurons are active for any given stimulus. Such a coding scheme carries potential metabolic advantages, because it reduces resource-intensive neural spiking. In addition, by converging on an efficient code, sparseness may serve to make the statistical structure of the environment explicit, which may carry computational advantages with regard to later processing [116].

Beginning with the retina itself, and extending to the thalamic lateral geniculate

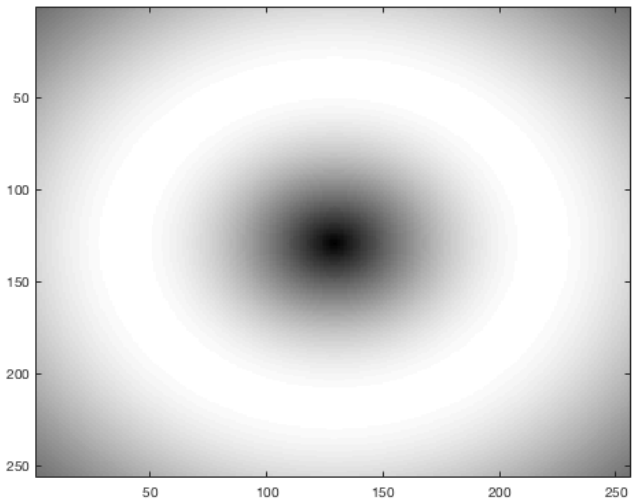
nucleus, the center-surround receptive-field responses of early visual neurons has been theorized as performing a whitening of the initial input such that neuronal responses are highly de-correlated.[7]

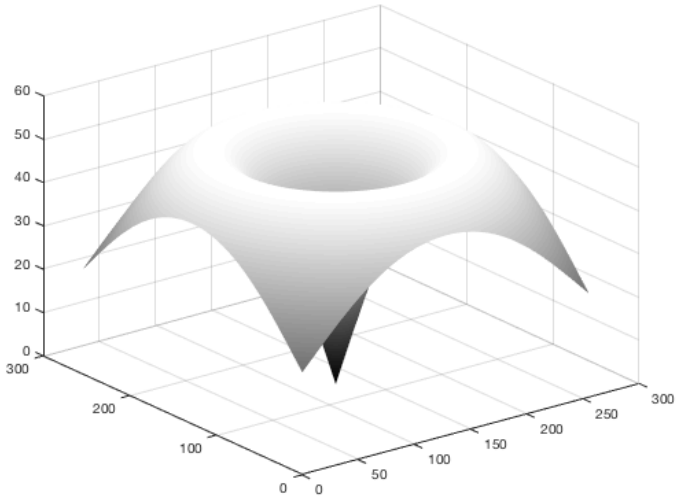
```
function HahnWhiten_Image

cd('/Users/williamedwardhahn/Documents/MATLAB')

I = double(imread('cameraman.tiff'));

I1=Hahn_scale(Hahn_Whiten1(I));

%Plots
imagesc(I)
colormap(gray)
set(gcf,'color','w')
snapnow;

imagesc(I1)
colormap(gray)
set(gcf,'color','w')
snapnow;

surf(I)
colormap(gray)
set(gcf,'color','w')
```

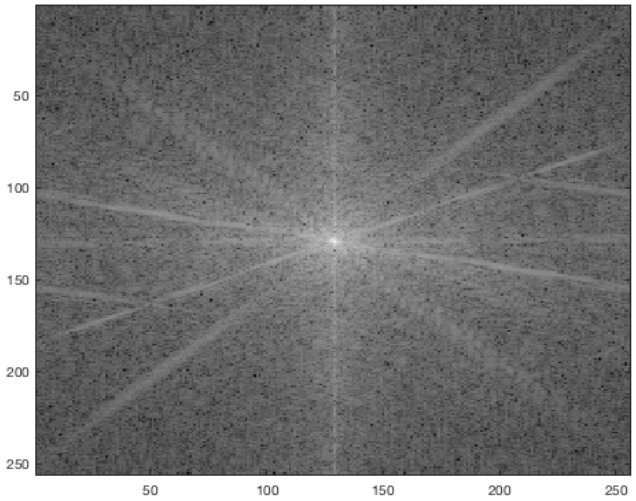
```
set(gcf,'RendererMode','manual')
shading interp
snapnow;

surf(I1)
colormap(gray)
set(gcf,'color','w')
set(gcf,'RendererMode','manual')
shading interp
snapnow;

end



function I=Hahn_Whiten1(I)

[N N]=size(I);

[fx fy] = meshgrid(-N/2:N/2-1);
[theta rho]=cart2pol(fx,fy);

f=rho.*exp(-0.5*(rho/(0.7*N/2)).^2);

FI=fftshift(fft2(I));
```

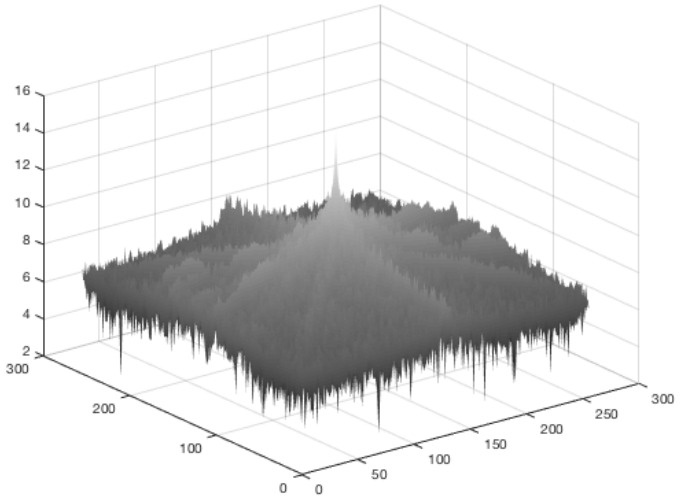
```
I = ifft2(fftshift(f.*FI));

imagesc(f)
colormap(gray)
set(gcf,'color','w')
snapnow;

surf(f)
colormap(gray)
shading interp
set(gcf,'color','w')
set(gcf,'RendererMode','manual')
snapnow;

imagesc(log(abs(FI)))
set(gcf,'color','w')
colormap(gray)
snapnow;

surf(log(abs(FI)))
colormap(gray)
shading interp
set(gcf,'color','w')
set(gcf,'RendererMode','manual')
snapnow;
```

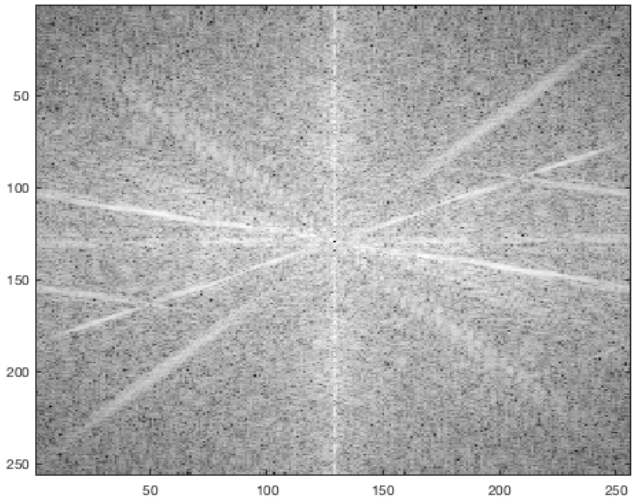
```
imagesc(log(abs(f.*FI)))
colormap(gray)
snapnow;

surf(log(abs(f.*FI)))
colormap(gray)
shading interp
set(gcf,'color','w')
set(gcf,'RendererMode','manual')
snapnow;

end
```

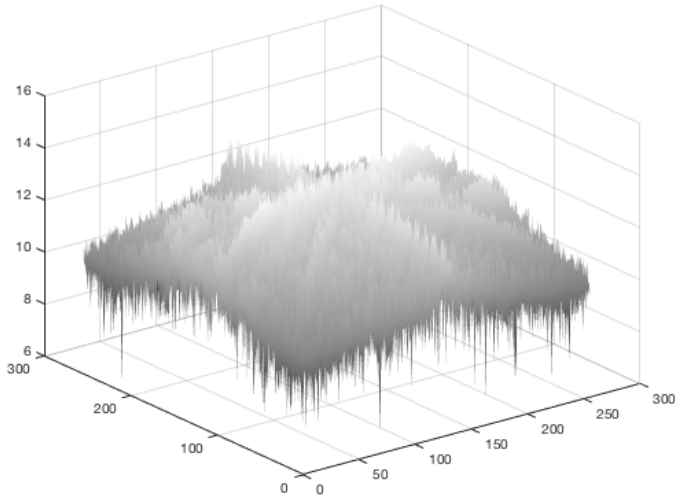



60
50
40
30
20
10
0
300
200
100
0
0
50
100
150
200
250
300


50
100
150
200
250
50
100
150
200
250


16
14
12
10
8
6
4
2
300
200
100
0
0
50
100
150
200
250
300

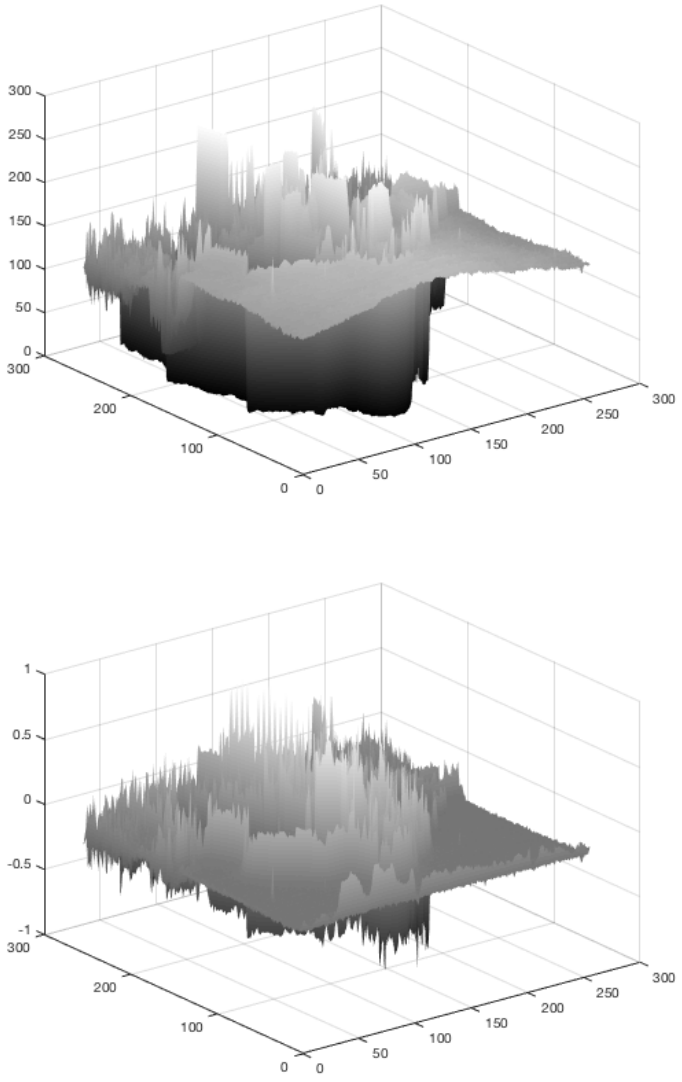

50
100
150
200
250
50
100
150
200
250




```
function x=Hahn_scale(x)


originalMinValue = (min(min(x)));

originalMaxValue = (max(max(x)));

desiredMin = -1;

desiredMax = 1;

desiredRange = desiredMax - desiredMin;

x = desiredRange * ((x) - originalMinValue) / (originalMaxValue - originalMinValue


end
```


50
100
150
200
250
50
100
150
200
250


50
100
150
200
250
50
100
150
200
250



A critical extension of these simpler mechanisms are the cortical structures of primary visual cortex, or V1, which received direct input from the thalamus. In humans, V1 is a relatively massive structure with high cell density, vastly outnumbering those in the afferent thalamic projections. As first delineated by Hubel and Weisel in their now classic experiments on animal cortex, neurons in these cortical structures generate a retinotopically organized map with many neurons, showing a high degree of selectivity to specific orientations, sizes and directions of motion. Later analyses

demonstrated that these response characteristics are better modeled by a Gabor function, which are composed of a sinusoidal function multiplied by a Gaussian, yielding a filter that may be localized in both space and frequency [81]. These neurophysiological observations have been shown to have psychophysical implications, evidenced by adaptation [14], masking [92], and summation [135] between gratings with similar orientation and spatial frequency characteristics, suggesting the existence of narrowly tuned 'channels' in the human visual system akin to those found in other mammalian species.

In the decades since these initial discoveries, much work has suggested that these response characteristics are the result of early experience and thus represent a tuning to the natural statistics of the world in which an organism develops. Blakemore and Cooper [15] showed that animals raised in an environment containing strictly horizontally or vertically orientated stripes showed profound neural and behavioral deficits in response to edges of the opposite orientation. In humans, visually-evoked potential (VEP) techniques have demonstrated that orientation selectivity does not emerge until around six weeks of age [8].

How does perceptual experience give rise to V1 filter characteristics? A significant advance in this regard came from the computational approach of Olshausen and Fields [115], who found that a generative model trained on natural images with a sparsity constraint yielded features that were highly similar to those characterized by the wavelet-like responses of V1 neurons. This strongly supports the idea that visual tuning is a product of sparse coding. More recently, sparse coding has been observed experimentally in additional biological sensory systems including visual, auditory, olfactory, and motor systems. In the context of neural coding, sparsity could provide

a mechanism for metabolically-efficient signal processing.

Sparse coding has received substantial attention in neuroscience and computer science, particularly computer vision, because 1) parsimonious nature of sparse encoding provides for a metabolically efficient implementation, 2) the mathematical structure of sparse signals allows for sub-nyquist sampling, and 3) computational sparse neural networks have begun to outperform existing methods for speech, image processing and natural language processing.

Physiological and psychophysical evidence suggest that early visual cortex compresses the visual input on the basis of spatial and orientation-tuned filters. Recent computational advances have suggested that these neural response characteristics may reflect a sparse coding architecture, in which a small number of neurons need to be active for any given image, yielding critical structure latent in natural scenes.

2.14 SPARSE CODING AND VISUAL CORTEX

Sparse coding is a common method used to extract and collect features in data. It is essentially a class of unsupervised methods for learning sets of over-complete bases to represent data efficiently [110]. Natural signals contain few active features, represented by a small number of non-zero code elements, and can be well-approximated by a subset of these elements from an overcomplete dictionary [133]. This method of sparse approximation is the foundation for many modern sensing and signal processing applications [133]; this is mainly due to its ability to efficiently and accurately represent a signal, as digital systems waste time and energy digitizing information that is eventually discarded during compression [133].

Sparse models are particularly useful in scientific applications, such as biomarker discovery in genetic or neuroimaging data, where the interpretability of a predictive model is essential [128]. Another advantage of sparsity is that it substantially improves the cost efficiency of signal processing [128]. These populations are typically very overcomplete, allowing great flexibility in the representation of a stimulus [133]. Research has found compelling evidence that the properties of V1 population responses to natural stimuli may be the result of a sparse approximation [133]. It has been shown that Vl receptive fields are consistent with optimizing the coefficient sparsity when encoding natural images [133]. Vl recordings, in response to natural scene stimuli, show activity levels (corresponding to the coefficients a_m) becoming sparser as neighboring units are also stimulated [133].

2.15 UNSUPERVISED FEATURE LEARNING

A typical pipeline in machine learning for image classification is to hand an engineer an algorithm for feature extraction first. Dictionary learning refers to a class of algorithms for unsupervised feature learning, i.e. given unlabeled examples (photos) the system will first learn what features to extract to best represent the data (photos). Dictionaries, also known as filter-banks or codebooks, are often used in signal processing for image compression and representation. Dictionary elements, the column vectors of a dictionary matrix, go by a variety of names including neurons, atoms, features, filters, and receptive fields [**?**]. The dictionary learning framework leads to state-of-the-art results for many image (and video) processing tasks [101]. One approach to choose D is from a known set of transforms (Steerable wavelet, Curvelet, Contourlets, Bandlets) [101]; yet, applications in which the dictionary is adapted and learned from data could yield a more compact representation [100]. Here we describe

how to use LCA coupled with a Hebbian update rule to learn a pseudo-overcomplete dictionary. The DL-LCA algorithm is easy to implement with few hyper-parameter to tune. The network associates each node with an element of a dictionary. A popular algorithm for sparse code inference is the Iterative Shrinkage and Thresholding Algorithm [64]. LCA, is a continuous-time, biologically relevant form of ISTA [64]. ISTA is a well-known digital solver for static sparse recovery, whose iteration is a first-order discretization of the LCA differential equation [10]. In terms of network implementation, the dictionary elements are updated by simple Hebbian learning between the outputs computed for each data vector (a_i) and the resulting residual image [115].

A sparse model consists of a dictionary

$$\hat{\mathbf{a}} = \arg\min_{\mathbf{a}} ||\mathbf{a}||_0 \quad \text{s.t.} \quad \mathbf{x} = \mathbf{D}\mathbf{a} \tag{2.107}$$

where $||\mathbf{a}||_0$ is the ℓ_0 pseudo-norm, which counts the number of non-zero basis coefficients. The dictionary matrix is comprised of m feature vectors denoted as ϕ_i.
Dictionary learning is the task of finding the unique $\mathbf{D}$ that yields the sparsest representation for each set of signals. The dictionary is constructed to have m vectors $\{\phi_m\}$ that span the space $\mathbb{R}^n$. Choosing $m > n$ establishes an over complete dictionary. Hebb's rule states that neurons that fire together, wire together. Represented in component form as a difference equation,

$\Delta\mathbf{w} \;=\; \eta\, y(\mathbf{x}_n)\mathbf{x}_n$

[141]

"Techniques from sparse signal representation are beginning to see significant impact in computer vision, often on non-traditional applications where the goal is not just to obtain a compact high-fidelity representation of the observed signal, but also to extract semantic information. The choice of dictionary plays a key role in bridg-

ing this gap: unconventional dictionaries consisting of, or learned from, the training samples themselves provide the key to obtaining state-of-the art results and to attaching semantic meaning to sparse signal representations. Understanding the good performance of such unconventional dictionaries in turn demands new algorithmic and analytical techniques" [164].

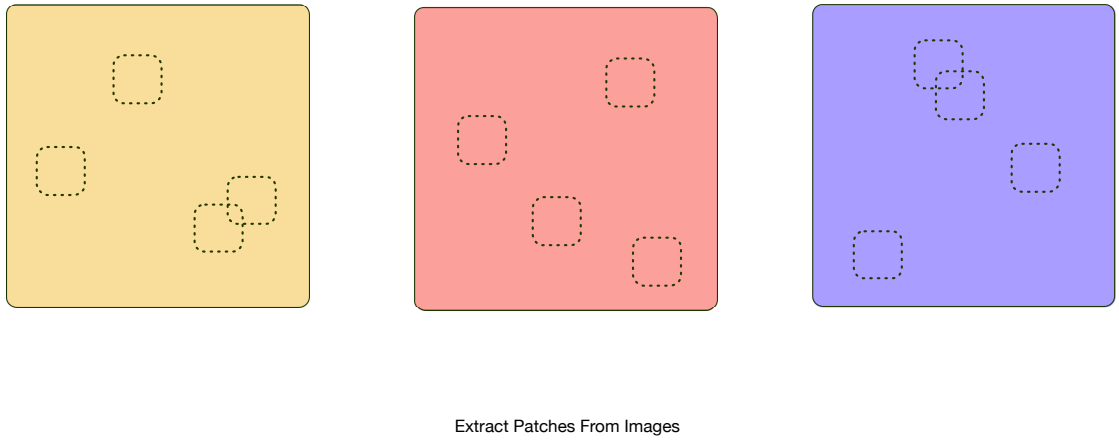


Figure 2.2

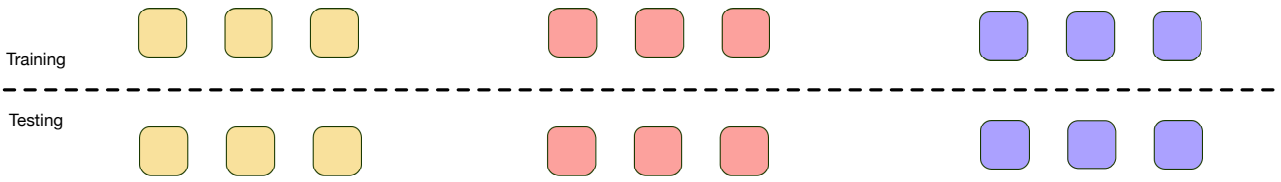
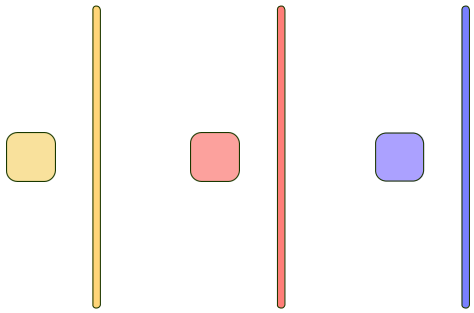


Figure 2.3

A sparse model consists of a dictionary (aka codebook, filterbank, frame)

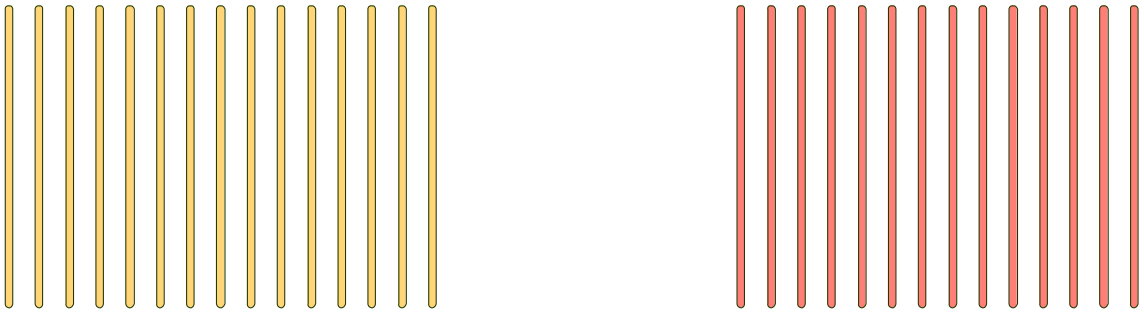
$$\hat{\mathbf{a}} = \arg\min_{\mathbf{a}} ||\mathbf{a}||_0 \quad \text{s.t.} \quad \mathbf{x} = \mathbf{D}\mathbf{a} \tag{2.108}$$

Where $||\mathbf{a}||_0$ is the ℓ_0 pseudo-norm, which counts the number of non-zero basis coefficients. The dictionary matrix is comprised of m feature vectors denoted as ϕ_i,



Vectorize Patches

Figure 2.4



Vectorized Patches K=1

Vectorized Patches K=2

Figure 2.5

Pseudo-Overcomplete Sparse Dictionary K=1

Pseudo-Overcomplete Sparse Dictionary K=2

Figure 2.6

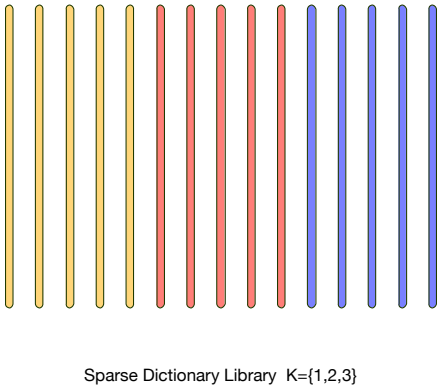


Figure 2.7

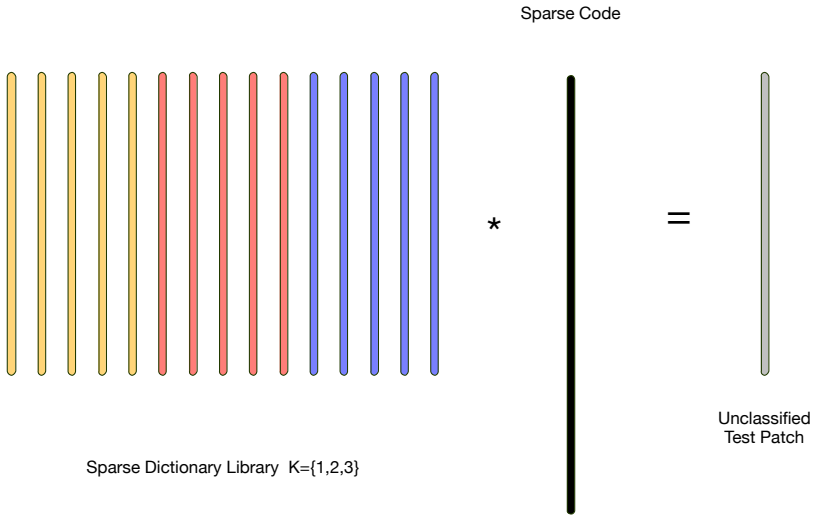


Figure 2.8

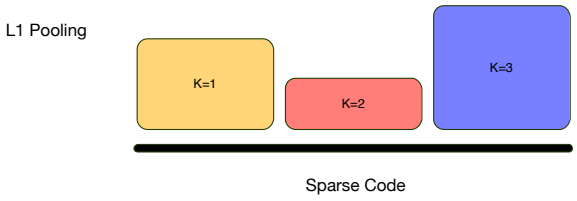


Figure 2.9

$$\mathbf{D} = \{\phi_1, \phi_2, \phi_i, \ldots, \phi_n\} \tag{2.109}$$

Dictionaries, also known as filter-banks or codebooks, are used in signal processing for image representation and compression. Dictionary elements, the column vectors of a dictionary matrix, go by a variety of names including neurons, atoms, features, filters, and receptive fields. Traditional dictionaries include the Fourier and wavelet basis. Fourier basis expansion provides for the compression of band-limited signals (resolution-limited images). A signal in $\mathbb{R}^n$ is said to be sparse if in a given basis expansion most of the coefficients are zero. A signal is sparse in the Fourier domain if there are a small number of nonzero coefficients in the signal's Fourier expansion. Natural images are sparse in the wavelet domain and thus are compressible. JPEG 2000 takes advantage of this wavelet basis to reduce the number of bits required to send and store images. Videos of frame differences (Frame k+1 - Frame k) are canonically sparse in the sense that for many natural videos most pixels do not change much from frame to frame. The graphic in Figure 1 represents a sequence of several frame differences. The frame differences are broken into manageable chunks called patches, illustrated in Figure 2.

The labeled patches are the input data (stimuli) $x_i \in \mathbb{R}^n$, where n represents the pixel dimension of the frame. These frame difference image patches are categorized according to their labeled class. Each category of patches can then become inputs used to develop a unique dictionary per category.

Dictionary learning is the task of finding the unique $\mathbf{D}$ that yields the sparsest representation for each set of signals. The dictionary is constructed to have m vectors $\{\phi_m\}$ that span the space $\mathbb{R}^n$. Choosing $m > n$ establishes an overcomplete dictio-

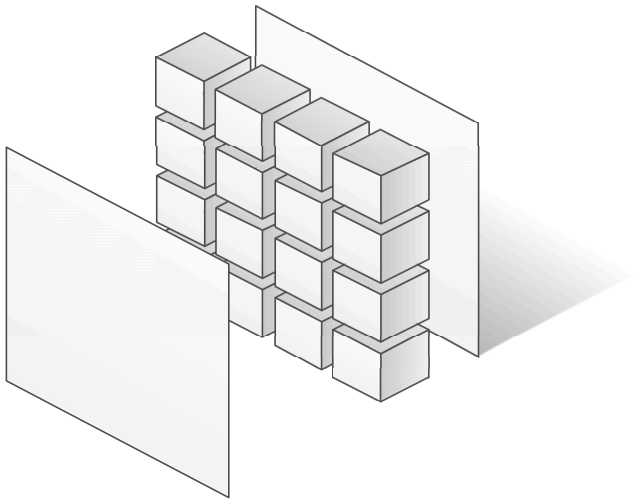


Figure 2.10: Graphic of video frames grouped into patches.

nary. In the construction process the input signals are represented as combinations of the adapted dictionary vectors. When the dictionary is overcomplete, there are an infinite number of possible representations:

$$s_i = \sum_{i=1}^{m} a_i \phi_i \tag{2.110}$$

In optimal sparse approximation, the goal is to maximize the total number of the a_i coefficients set to zero. This task is equivalent to minimizing the ℓ_0 norm:

$$\min_a \quad ||\mathbf{a}||_0 \quad \text{subject to} \quad s = \sum_{i=1}^{m} a_i \phi_i \tag{2.111}$$

This combinatorial optimization problem is NP-hard. However, it has been shown using computational mathematics, that convex relaxation of the ℓ_0 norm to the ℓ_1 norm essentially accomplishes the same goal and is practical to implement. The task becomes:

$$\min_{\mathbf{a}\in\mathbb{R}^m} ||\mathbf{Da}-\mathbf{x}||^2+\lambda||\mathbf{a}||_1 \tag{2.112}$$

a: is the code we want

x: is the data we have

D: is adapted

2.16 ATOMIC DECOMPOSITION

Atomic decomposition techniques fall into two main categories: relaxation techniques (basis pursuit) and greedy methods (matching pursuit). For a given signal the goal is to select a sparse set atoms from a given dictionary that will represent the input signal. This subset selection problem is NP-hard, and thus we must either resort to greedy heuristics or relax the ℓ_0 pseudo-norm constraint by replacing it with the ℓ_1 norm, reducing the problem to convex programing[19] While greedy or locally optimal solutions sometimes provide appropriate codes, they are unstable to small input perturbations and thus are not plausible models for neural mechanisms. More recently, locally competitive models have been proposed that achieve a sparse representation in a more neurologically plausible fashion. LCAs have been shown to provide greater stability than greedy methods in response to input perturbations.

```
function Hahn_AtomicDecomp()


close all
clear all
clc
```

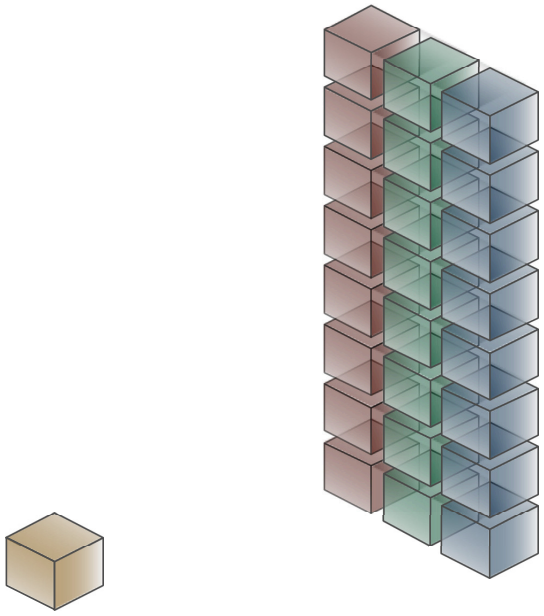


Figure 2.11: Visualization of three dictionaries with an unlabeled test patch waiting to be labeled.

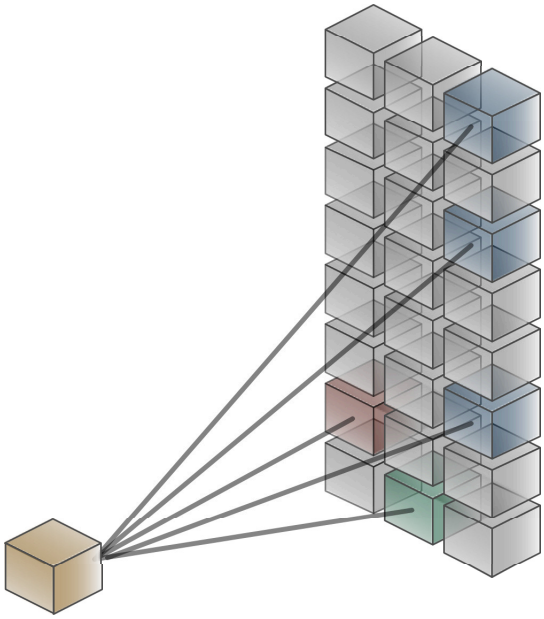


Figure 2.12: Sparse decomposition of input patch.

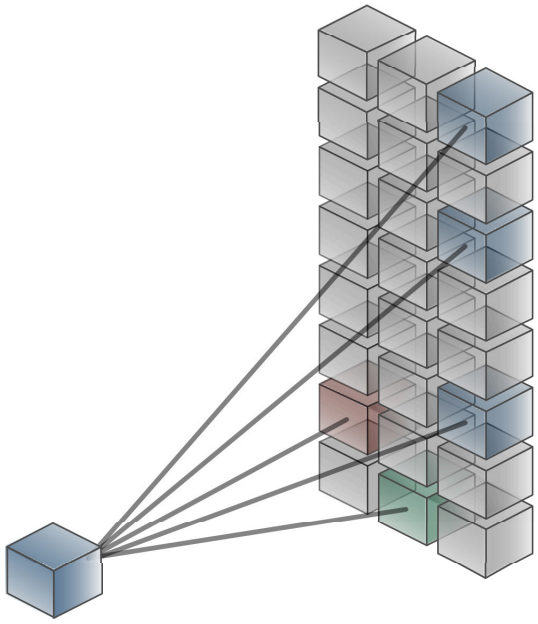


Figure 2.13: Test patch is labeled using ℓ_1 pooling of sparse decomposition.

```
load patches.mat
load dict512.mat

D=Wp';

for i = 4 : 4

    y=data(:,i);
    yy=reshape(y,16,16);

    figure(1)
    subplot(411)
```

```
imagesc(yy,[0,.5])
title('Original Patch')
colormap(gray)


a=BP(10,D,y);


subplot(412)
imagesc(reshape(D*a,16,16),[0,.5])
title('Basis Pursuit Sparse Approximation Patch')
colormap(gray)


subplot(413)
imagesc(yy-reshape(D*a,16,16),[0,.5])
title('Basis Pursuit Sparse Approximation Error')
colormap(gray)

subplot(414)
hist(a,100)
snapnow

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
figure(2)
colormap(jet)
subplot(411)
imagesc(yy,[0,.5])
title('Original Patch')
colormap(gray)

a=MP(10,D,y);

subplot(412)
imagesc(reshape(D*a,16,16),[0,.5])
title('Matching Pursuit Sparse Approximation Patch')
colormap(gray)

subplot(413)
imagesc(yy-reshape(D*a,16,16),[0,.5])
title('Matching Pursuit Sparse Approximation Error')
colormap(gray)

subplot(414)
hist(a,100)
snapnow


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

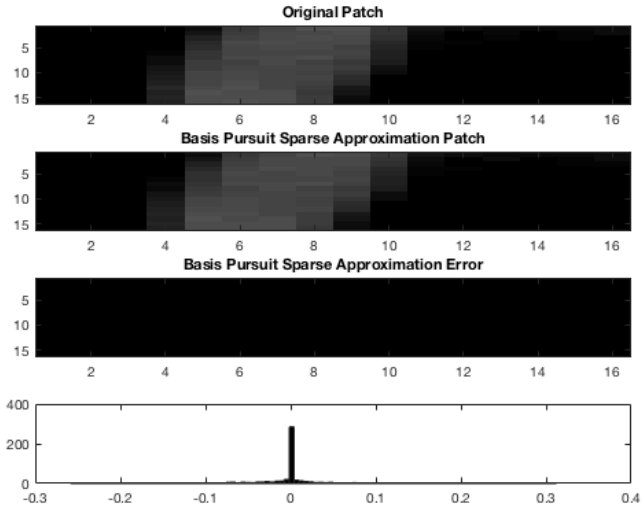
```
    figure(3)
    colormap(jet)
    subplot(411)
    imagesc(yy,[0,.5])
    title('Original Patch')

    a=LCA(y,D,0.01);

    subplot(412)
    imagesc(reshape(D*a,16,16),[0,.5])
    title('LCA Sparse Approximation Patch')
    colormap(gray)

    subplot(413)
    imagesc(yy-reshape(D*a,16,16),[0,.5])
    title('LCA Sparse Approximation Error')
    colormap(gray)


    subplot(414)
    hist(a,100)
    snapnow


end
```

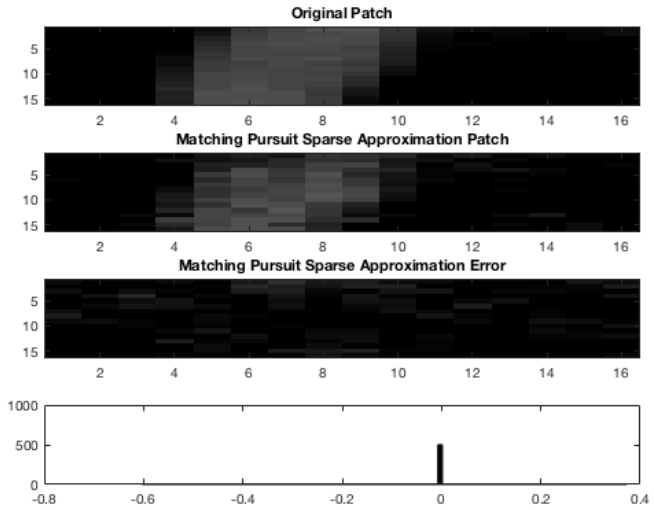
```
end




function [a, u] = LCA(y, D, lambda)



t=.01;
h=.0001;

d = h/t;
u = zeros(size(D,2),1);



for i=1:300


    a = ( u - sign(u).*(lambda) ) .* ( abs(u) > (lambda) );


    u =   u + d * ( D' * ( y - D*a ) - u - a  ) ;
```

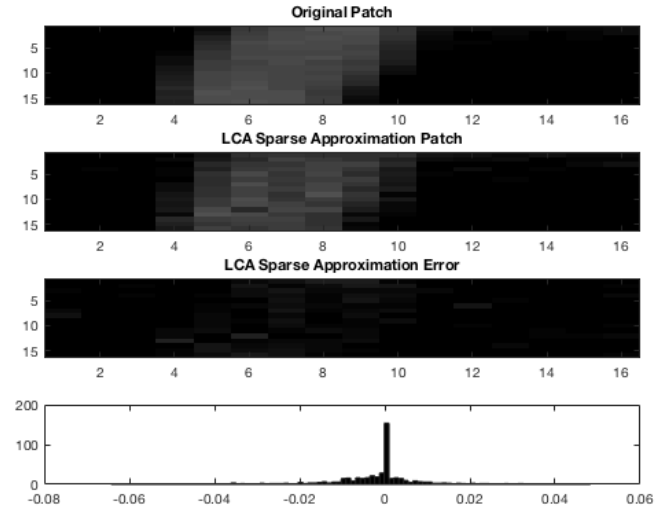
```
end




end



function [a] = MP(k,D,y)

n=size(D);
D1=zeros(n);
r=y;

for k=1:k


    [i,j] = max(abs(D'*r));

    D1(:,j)=D(:,j);

    D(:,j)=0;

    a = D1 \ y;
```

```
    r = y - D1*a;


end;


end


function [a] = BP(k,D,y)


cvx_begin quiet;

l=size(D);

variable a(l(2));

minimize( norm(D*a-y) );

subject to;
```

```
norm(a,1) <= k;

cvx_end;



end
```








```
function Hahn_Thesis_AtomicLCA1()

close all

clc

clear all


load patches.mat
```

```
load dict512.mat

D=Wp';


imagesc(filterplot(data(:,1:484)))
colormap(gray)
title('Natural Image Patches')


imagesc(filterplot(D(:,1:484)))
colormap(gray)
title('Sparse Filters Learning from Natural Image Patches')


for i=[1 2 4 13 14]

    y=data(:,i);
    y=y-mean(y(:));
    y=y/std(y(:));

    yy=reshape(y,16,16);

    colormap(jet)
    subplot(4,2,1)
    imagesc(yy)
```

```
title('Original Patch')
colormap(gray)



a=LCA(y(:),D,0.5);


subplot(4,2,3)
imagesc(reshape(D*a,16,16))
title('LCA Sparse Approximation Patch')
colormap(gray)



subplot(4,2,5)
imagesc(yy-reshape(D*a,16,16))
title('LCA Sparse Approximation Error')
colormap(gray)


subplot(4,2,7)
hist(a,100)
title('Histogram of LCA Sparse Coefficients')
```

```
subplot(4,2,2)
surf(yy)
title('Original Patch')
zl1=zlim;
colormap(gray)

subplot(4,2,4)
surf(reshape(D*a,16,16))
title('LCA Sparse Approximation Patch')
zlim(zl1)
colormap(gray)

subplot(4,2,6)
surf(yy-reshape(D*a,16,16))
title('LCA Sparse Approximation Error')
zlim(zl1)
colormap(gray)


subplot(4,2,8)
bar(a)
title('LCA Sparse Coefficients')


drawnow()
snapnow;
```

```
end
```

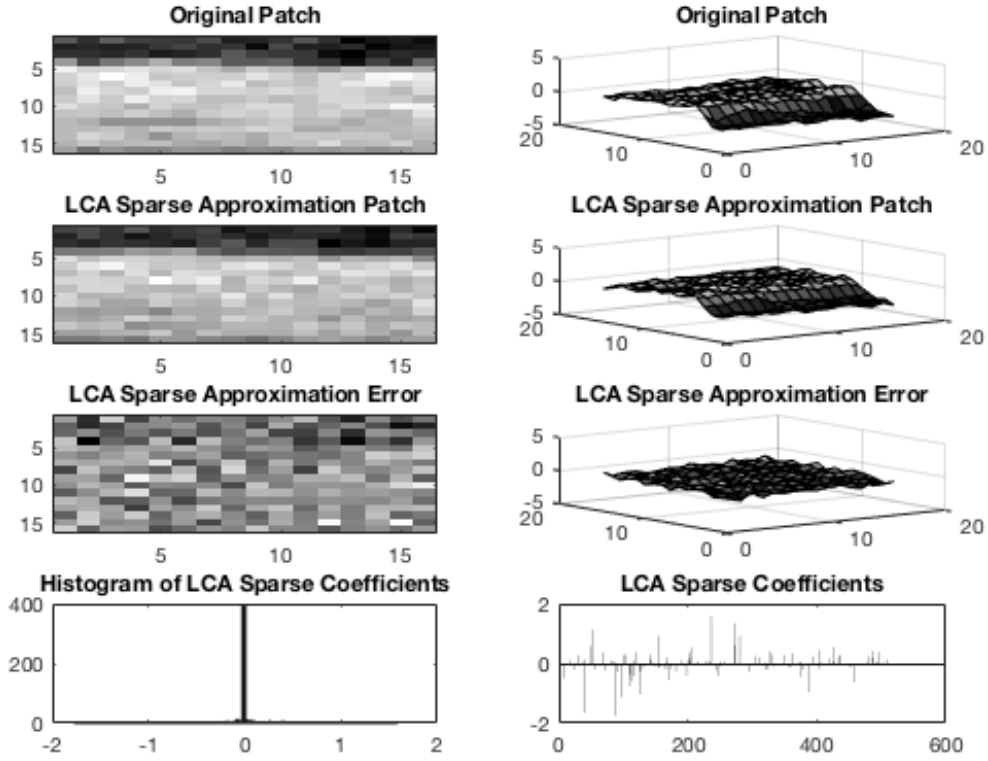
```
end
```

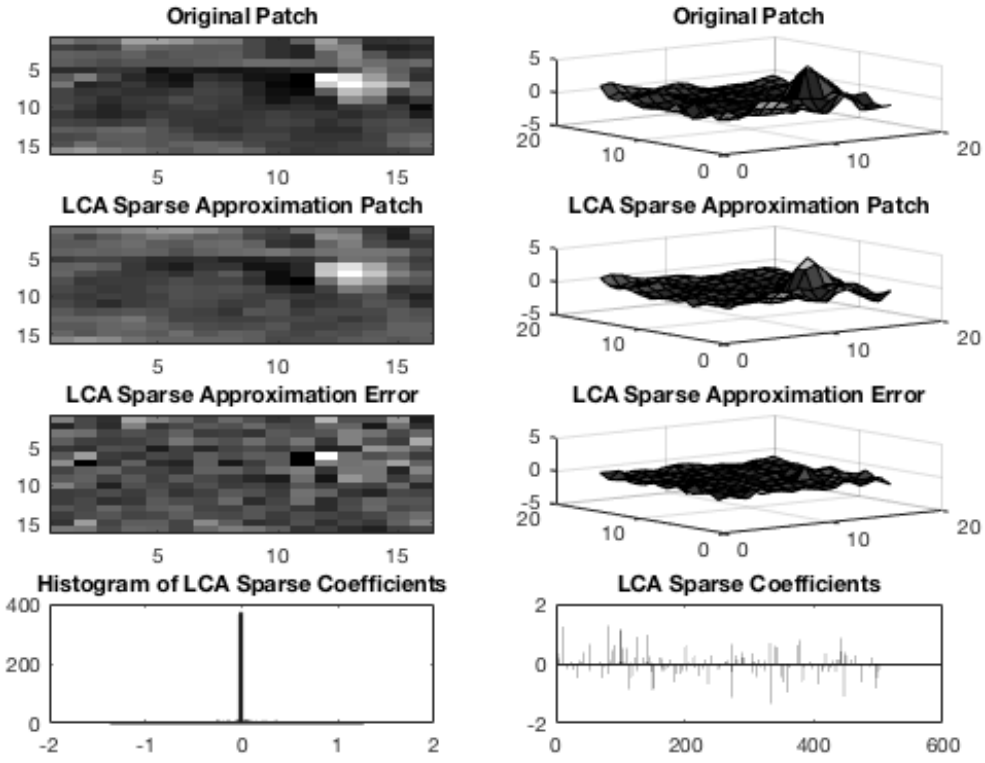
```
function [a, u] = LCA(y, D, lambda)

n=300;

t=.01;
h=.0001;

d = h/t;
u = zeros(size(D,2),1);

Dt=D';

G=Dt*D;
```

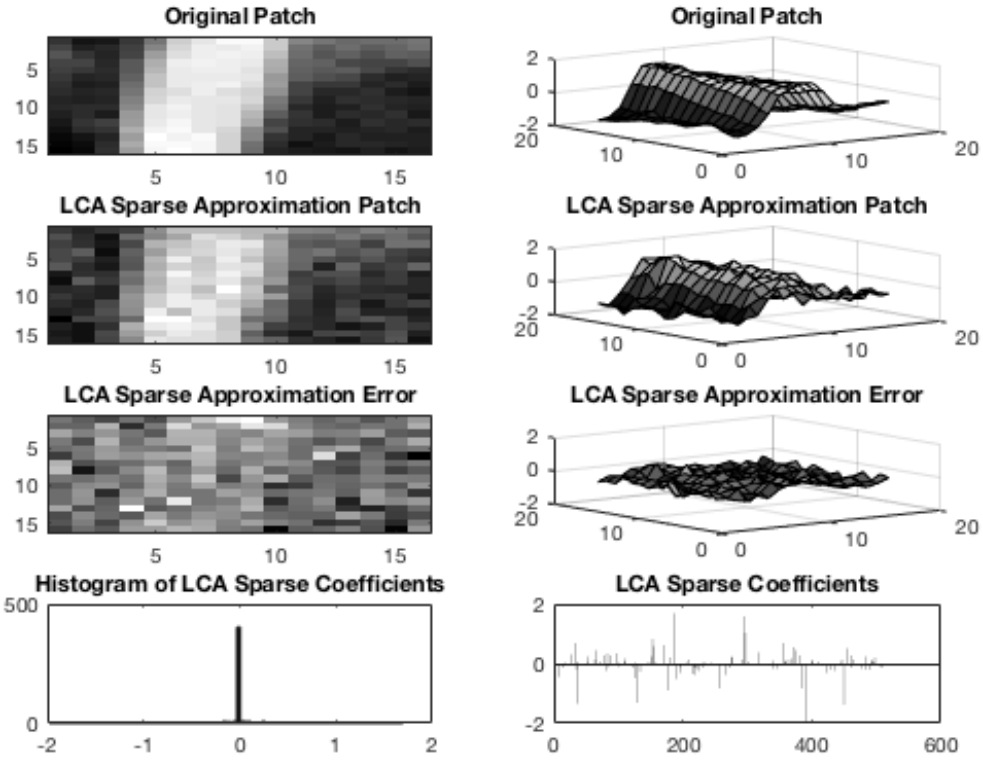
```
I=eye(size(G));

G=G-I;


for i=1:n


    a = ( u - sign(u).*lambda ) .* ( abs(u) > lambda );


    u =  u + d * (- u -G*a + Dt*y ) ;


end







end
```

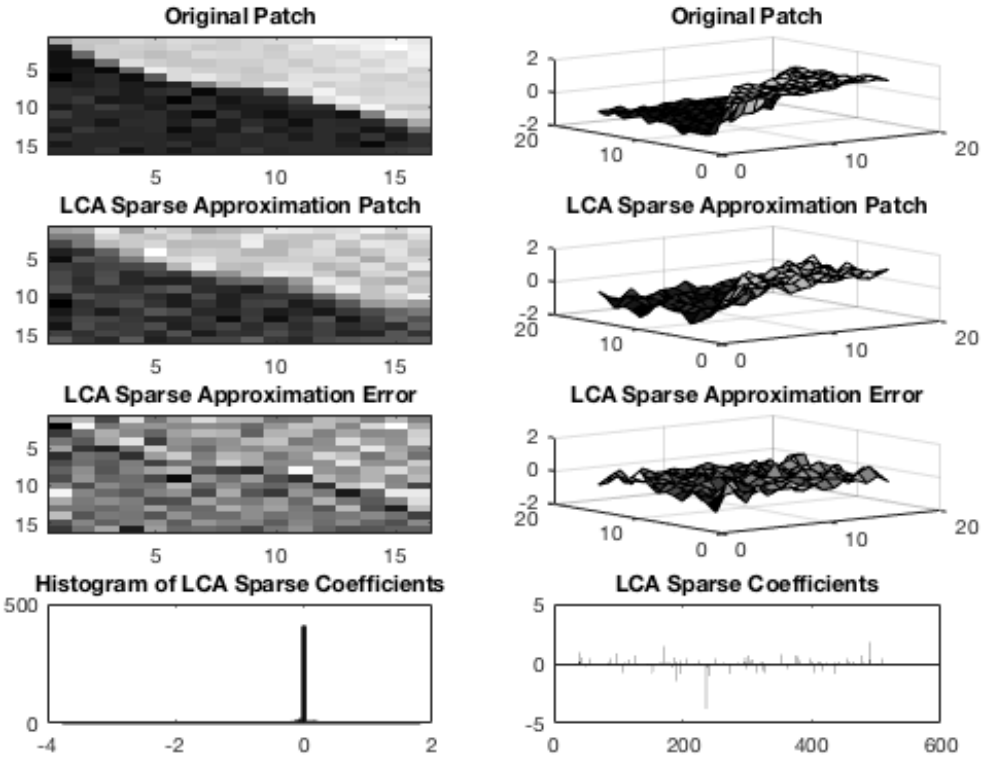
```
function [D] = filterplot(X)

X=X';

[m n] = size(X);



w = round(sqrt(n));
h = (n / w);

c = floor(sqrt(m));
r = ceil(m / c);

p = 1;

D = - ones(p + r * (h + p),p + c * (w + p));

k = 1;
for j = 1:r
    for i = 1:c
        D(p + (j - 1) * (h + p) + (1:h), p + (i - 1) * (w + p) + (1:w)) = reshape(
        k = k + 1;
    end

end
```

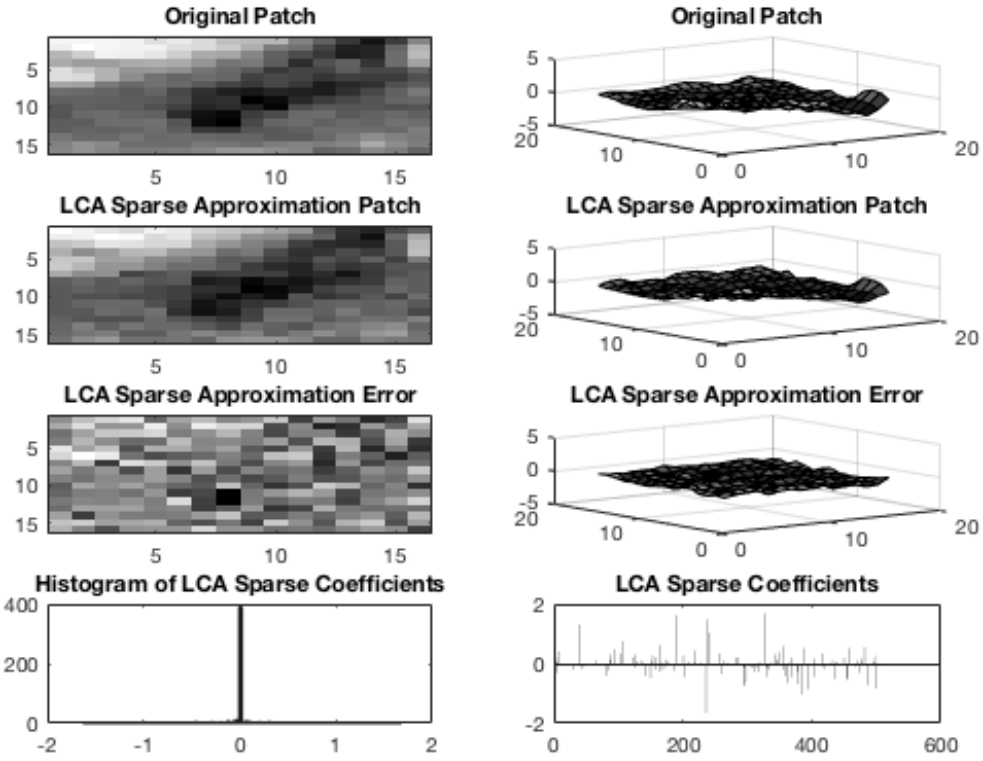
```
end
```


Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
Histogram of LCA Sparse Coefficients
Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
LCA Sparse Coefficients


Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
Histogram of LCA Sparse Coefficients
Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
LCA Sparse Coefficients


Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
Histogram of LCA Sparse Coefficients
Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
LCA Sparse Coefficients


Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
Histogram of LCA Sparse Coefficients
Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
LCA Sparse Coefficients


Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
Histogram of LCA Sparse Coefficients
Original Patch
LCA Sparse Approximation Patch
LCA Sparse Approximation Error
LCA Sparse Coefficients

Chapter 3

Sparse Filtering

3.1 SPARSE FILTERING

Sparse filtering is an unsupervised feature learning technique that, when trained on natural images, produces receptive fields similar to those found in visual cortex. Sparse filtering has the advantage of learning the feature distribution directly. Unlike many other deep learning models, such as deep belief networks, restricted Boltzmann machines and auto-encoders, sparse filtering has a simple implementation with no hyper-parameters to tune.

Sparse filtering networks make no special considerations for the particular dataset or data type; thus they can be extended to handle many data types, including hyper-spectral and multimodal (i.e. video with sound). In sparse filtering, the objective is a normalized sparsity penalty, where the response of the filters (i.e. dictionary atoms) are divided by the norm of all the filters. This is relevant in visual neuroscience to the work on local contrast normalization.

The sparse filtering technique defines an objective function that is minimized in order to build the dictionary $\mathbf{D}$, which resembles receptive fields of cortical neurons. The Objective Function is to minimize the sum of normalized entries of the feature value

matrix. On each iteration:

1. Normalize Across Rows
2. Normalize Across Columns
3. Objective Function = Sum of the Normalized Entries

Let $\mathbf{F}$ be the feature value matrix to be normalized, summed, and minimized. The components

$$f_j^{(i)} \tag{3.1}$$

represent the j^{th} feature value (j^{th} row) for the i^{th} example (i^{th} column), where

$$f_j^{(i)} = \mathbf{w}_j^T \mathbf{x}^{(i)} \tag{3.2}$$

Here, the $\mathbf{x}^{(i)}$ are the input patches and $\mathbf{W}$ is the weight matrix. Initially random, the weight matrix is updated iteratively in order to minimize the Objective Function.

In the first step of the optimization scheme,

$$\widetilde{\mathbf{f}}_j = \frac{\mathbf{f}_j}{||\mathbf{f}_j||_2} \tag{3.3}$$

Each feature row is treated as a vector, and mapped to the unit ball by dividing by its ℓ_2-norm. This has the effect of giving each feature approximately the same variance.

The second step is to normalize across the columns, which again maps the entries to the unit ball. This makes the rows about equally active, introducing competition between the features and thus removing the need for an orthogonal basis. Sparse filtering prevents degenerate situations in which the same features are always active. [111]

$$\hat{\mathbf{f}}^{(i)} = \frac{\widetilde{\mathbf{f}}^{(i)}}{||\widetilde{\mathbf{f}}^{(i)}||_2} \tag{3.4}$$

The normalized features are optimized for sparseness by minimizing the ℓ_1 norm. That is, minimize the Objective Function, the sum of the absolute values of all the entries of $\mathbf{F}$. For datasets of M examples we have the sparse filtering objective:

$$\text{minimize} \quad \sum_{i=1}^{M} \left|\left|\hat{\mathbf{f}}^{(i)}\right|\right|_1 = \sum_{i=1}^{M} \left|\left| \frac{\widetilde{\mathbf{f}}^{(i)}}{||\widetilde{\mathbf{f}}^{(i)}||_2} \right|\right|_1 \tag{3.5}$$

The sparse filtering objective is minimized using a Limited-memory Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, a common iterative method for solving unconstrained nonlinear optimization problems.[19]

Chapter 4

Locally Competitive Algorithms

4.1 LCA FOUNDATION

LCA is a parallel dynamical system for computing sparse representations of data. LCA is a class of algorithms based on thresholds and local competition to solve a family of sparse approximation problems [133].When the system is presented with an input image s(t), the collection of nodes evolve according to fixed dynamics [133] and achieve a sparse representation of the input in a neurologically plausible fashion. LCAs have been shown to provide greater stability than greedy methods in response to input perturbations. Sparse approximation is a difficult non-convex optimization problem that is at the center of much research in mathematics and signal processing [133]. There are several sparse approximation methods that do not fit into the two primary approaches of pure greedy algorithms or convex relaxation but involve computations that would be very difficult to implement in a parallel, distributed architecture [133]. According to Rozell, "LCA coefficients for temporal (video) sequences demonstrate inertial properties that are both qualitatively and quantitatively more regular, i.e., smoother and more predictable, than the coefficients produced by greedy algorithms" [133].

In this work, we use a Locally Competitive Algoithm (LCA), a neurologically-inspired

method of attaining a sparse representation of a signal. The architecture for LCA contains properties essential for a neurally plausible sparse coding system, and the dynamics of LCA also behave in a manner similar to those observed in neural systems [133]. One benefit of LCA is that it minimizes an energy function that combines reconstruction mean-squared error (MSE), which describes the data, and a cost function (constraint) on neural activity [133]. LCA node dynamics are expressed by a set of non-linear ordinary differential equations, which are congruous to the previously described Hopfield network [133].

In LCA, each node is comprised of a source of an internal state signal and a thresholding element [133]. When an input is received by a node, it causes the membrane potential of the node to charge up like a leaky integrator. The change in membrane potential is proportional to resemblence of the input to a given dictionary element and the amount of inhibition received from other nodes [133]. When the membrane potential charges sufficiently and rises over a threshold, an action potential will be produced for extracellular signaling. Super-threshold responses inhibit weak neighboring nodes through lateral connections, making the search for a sparse approximation more energy efficient [133]. This efficiency is primarily due to the unidirectional nature of this inhibition of weaker nodes by active nodes, preventing the weaker nodes from becoming active and producing a counter-inhibition [133]. Furthermore, this lateral inhibition resembles the behavior observed in many retinal cells, where a postsynaptic potential in one retinal cell produces a hyperpolarization of the membrane potential of neighboring cells via depolarization of inhibitory horizontal and amacrine cells. Similar to LCA and sparse coding, the inhibitory mechanisms found in these cells essentially reduce the stimulus representations across arrays of retinal cells so that the output of the ganglion cells is an approximation of the dimensions of the original

stimulus [?]. LCA, unlike retinal cells, doesn't require extra nodes to accomplish this inhibition, thus reducing the complexity and size of the network.

4.2 HARDWARE INTERPRETATION OF LCA

It is possible to recreate Hopfield's model of a neuron, with smooth input-output relationship between the mean firing rate and the mean membrane potential, using simple circuit elements. Due to the inherent robustness of the network, implementation of such a model using integrated circuits could allow for the construction of chips that would be less susceptible to element failure and soft-failure than normal circuits [73]. Hopfield pointed out that such chips would be wasteful of gates but could be much larger than conventional designs at a given yield. It is their asynchronous parallel processing capability that further prompted him to suggest these chips could be of great value when applied to some special classes of computational problems [73]. When applied to signal and image processing, for example, the input features must be appropriately coded before the network can perform these abstract calculations. For example, feature extraction should have previously been performed [73]. It is perhaps possible, then, that this model can provide insight into the categorization or memorization of a Gestalt from a collection of features [73].

There are classes of physical systems whose spontaneous behavior can be used as a form of general (and error-correcting) content-addressable memory [73]. One of these, the LCA model, could be easily constructed from integrated circuit hardware as previously mentioned [73]. Physical systems can be potentially useful for memory if they contain states that tend toward local points of stability from nearby regions. In addition, a physical system's utility depends on whether any prescribed set of states

can readily be made the stable states of the system [73]. In LCA, the weights of the synapses connecting two nodes are updated in a process called training. Each weight is updated one at a time, which is more biologically plausible than the sychronous update method. This update rule also confers upon the network the ability to settle to a stable state. Hebbian learning is the method used to update the weights between two nodes during training such that two nodes that are on or off at the same time have higher weights.

LCAs can be implemented using a parallel network of simple elements that match well with parallel analog computational architectures, including analog circuits and sensory cortical areas such as V1 [133]. Another benefit of LCA is that is can be used to reconstruct compressively sensed images [133]. Compressive sensing (CS) is a technique used to capture a signal by sampling only 5-10 percent of the original image. [133]. CS reconstruction requires a large memory and many calculations, as it solves an optimal sparse approximation problem [133]. LCA, an analog way of finding sparse representations, can be modified for compressive sensing reconstruction as well [133]. Here we demonstrate an application of LCA to the recognition of butterfly classes in heavily cluttered natural images [86].

4.3 DYNAMICAL SYSTEM FOR SPARSE RECOVERY

There are simple systems of nonlinear differential equations that settle to the solution of

$$\min_{\mathbf{x}} \ \lambda||\mathbf{x}||_1 \ + \ \frac{1}{2}||\mathbf{Da}-\mathbf{x}||_2^2 \tag{4.1}$$

The LCA is a neurologically inspired system which settles to the solution of the above.[131] Developed to represent the "equation of motion" for a slice through a cortical column, LCAs are systems of nonlinear differential equations that settle to a minima of a given ℓ_1 regularized least squares optimization problem.

Here we use LCA for atomic decomposition: given an input signal x and a pseudo-overcomplete dictionary $\mathbf{D}$, the LCA returns a sparse vector α such that $\mathbf{D}\alpha \approx x$.

The three main components of LCA are leaky integration, nonlinear activation and inhibition/excitation networks [132]. Input to the LCA equations are a stimulus pattern and dictionary, and the output is a sparse code, i.e. a vector of dictionary coefficients. This set of coefficients can now be used as a feature vector for machine learning and classification.

The LCA model approximates the input $\mathbf{x}$ as a linear combination of receptive fields (dictionary elements, or feature columns). $\mathbf{x}$ is approximated as $\hat{\mathbf{x}}$, the product of a sparse vector $\mathbf{a}$ multiplied by receptive fields,

$$\hat{\mathbf{x}}(t) = \sum_m a_m(t)\phi_m \tag{4.2}$$

The sparse coefficient vector $\mathbf{a}$ is determined by solving the LCA differential equations.

$$\dot{v}_m(t) = \frac{1}{\tau}\left[b_m(t) - v_m(t) - \sum_{n \neq m} G_{m,n}a_n(t)\right] \tag{4.3}$$

A nonlinear threshold function is needed for LCA to convert a membrane potential

$\mathbf{v}$ into a firing rate,

$$a_m = T_m(\mathbf{v}) = \begin{cases} 0, & \mathbf{v} \leq \lambda \\ \\ \mathbf{v}, & \mathbf{v} > \lambda \end{cases} \tag{4.4}$$

$b_m(t)$ represents the similarity between the m^{th} receptive field and input stimulus, measured with an inner product,

$$b_m(t) = \langle \phi_m, \mathbf{x}(t) \rangle \tag{4.5}$$

$G_{m,n}$ measures the similarity between any two receptive fields ϕ_m and ϕ_n with an inner product,

$$G_{m,n} = \langle \phi_m, \phi_n \rangle \tag{4.6}$$

Note that the receptive fields, ϕ_m, are the columns of the dictionary $\mathbf{D}$, i.e. the feature vectors. Inhibition allows stronger nodes to prevent weaker nodes from becoming active, which results in a sparse solution. Specifically, the inhibition signal from the active node m to any other node n is proportional to the activity level a_m and to the inner product between the node receptive fields.

Originally inspired by visual cortex, LCA has been shown to provide stable atomic decompositions with dynamic inputs. Implementable on reconfigurable analog hardware, LCAs could provide the ultra-efficient high-performance computing techniques needed for future computer vision applications.

A Hopfield network is a (continuous or discrete) dynamical system for which a Lyapunov function relates system evolution to the geometry of a corresponding energy

surface [**?**]. Locally competitive algorithms (LCA) are continuous Hopfield(-like) networks for which the optimization problems of sparse representation are (weak) Lyapunov functions [**?**]. Further analysis demonstrates robust global convergence properties [**?**]. The LCA system demonstrates robust convergence properties which suggest it may be feasibly implementable for efficiently solving large-scale problems [**?**]. Network implementations of LCA systems suggest intriguing connections with neuroscience [**?**]. When the internal state of a node becomes significantly large, the node becomes active and produces an output signal used to represent the stimulus and inhibit other nodes [133]. Nodes in a population continually compete with neighboring units using lateral inhibition to calculate coefficients representing an input in an overcomplete dictionary [133]. LCA dynamical system is stable to guarantee that a physical implementation is well behaved [133].

The internal state signal in each node is calculated as a function of said matching signal received at said node and weighted outputs of all other nodes [133]. Excitatory input current is proportional to how well the image matches with the node's receptive field [133]. This output coefficient is the result of an activation function applied to the membrane potential [133]. The nodes best matching the stimulus will have internal state variables that charge at the fastest rates and become active soonest [133]. This can be modeled as a network of coupled leaky integrators. "An imaging system using VLSI to implement LCAs as a data collection front end has the potential to be extremely fast and energy efficient."[133]. In LCA, "time and energy resources would only be spent digitizing coefficients that are a critical component in the signal representation." [133]. LCA can implimented in hardware, for example, on FPGAs or memristor arrays [141].

Chapter 5

LCA Compressed Sensing

```
function Hahn_Thesis_CS_LCA_1D


close all

clear all

clc


P = 1; %Number of Signals


N = 1000; % Sparse Signal

x = zeros(N,P);


K = 5;  % #Non-Zeros



x(randi(P*N,[1 P*K]))=sign(randn(1,P*K));



M = 100; % #Samples
```

```
D = sign(randn(M,N));

% Compressed Measurements
y = D*x;


imagesc(y)
snapnow;

imagesc(D)
snapnow;

imagesc(x)
snapnow;



% Minimum l2 (Energy) Solution
x0 = D \ y;


% % Minimum l1 Solution Linear Programing
% x1 = l1eq_pd(x0, D, [], y, 1e-3);


% Minimum l1 Solution LCA
```

```
x2 = LCA2(y,D);



sum(sum(abs(x-x0)))

sum(sum(abs(x-x2)))



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Plots



plot(x)

axis([0, N, -2, 2])

title('Original Signal')

snapnow;


plot(x0)

axis([0, N, -2, 2])

title('Minimum l2 Reconstruction')

snapnow;


plot(x2)

axis([0, N, -2, 2])

title('Minimum l1 Reconstruction Locally Competitive Algorithm')

snapnow;
```

```
plot(x2-x)
axis([0, N, -2, 2])
title('Locally Competitive Algorithm Error')
snapnow;


plot(x0-x)
axis([0, N, -2, 2])
title('Minimum l2 Reconstruction Error')
snapnow;

end



function a=LCA(x,D)

lambda=4;
h = 0.005;

u = zeros(size(D,2),1);


for i=1:100
```

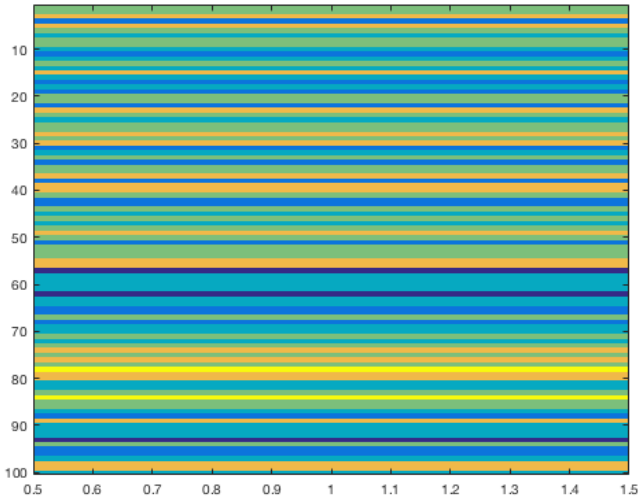
```
    a = ( u - sign(u).*(lambda) ) .* ( abs(u) > (lambda) );

    u = u + h * ( D' * ( x - D*a ) - u - a ) ;

end

a=round(a);

end


function a=LCA2(x,D)

lambda=4;

G = D'*D - eye(size(D,2));

b = D'*x;

u = zeros(size(D,2),size(x,2));
```

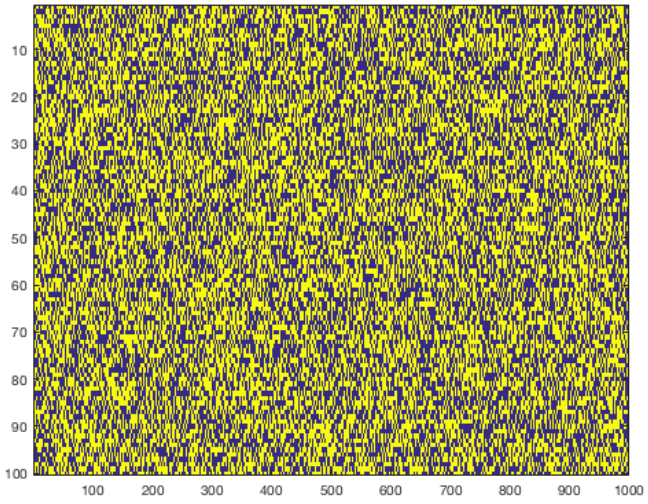
```
for i =1:1000

    a = ( u - sign(u).*(lambda) ) .* ( abs(u) > (lambda) );

    u = 0.99 * u + 0.01 * (b - G*a);

end


a=a.*(abs(a) > 0.5);

end
```

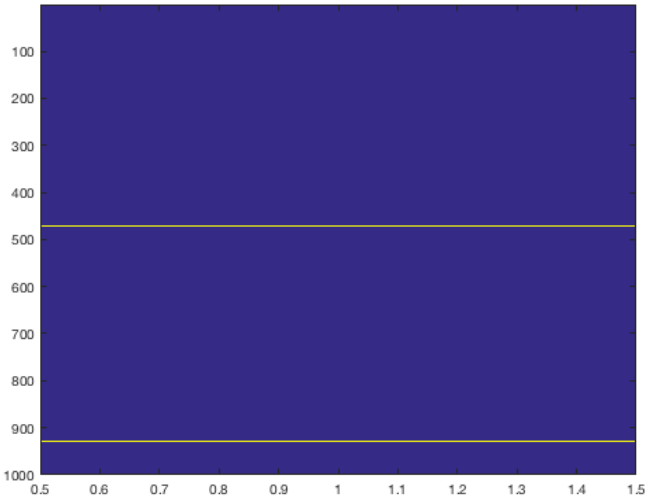
```
function a=LCA3(x,D)


G = D'*D - eye(size(D,2));

b = D'*x;

u = zeros(size(D,2),20);
```

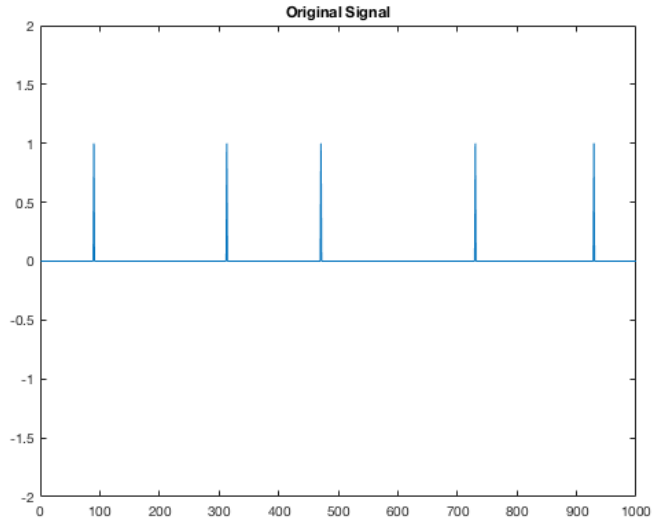
```
for i =1:100

    a=u.*(abs(u) > 4);

    u = 0.9 * u + 0.01 * (b - G*a);

end
```

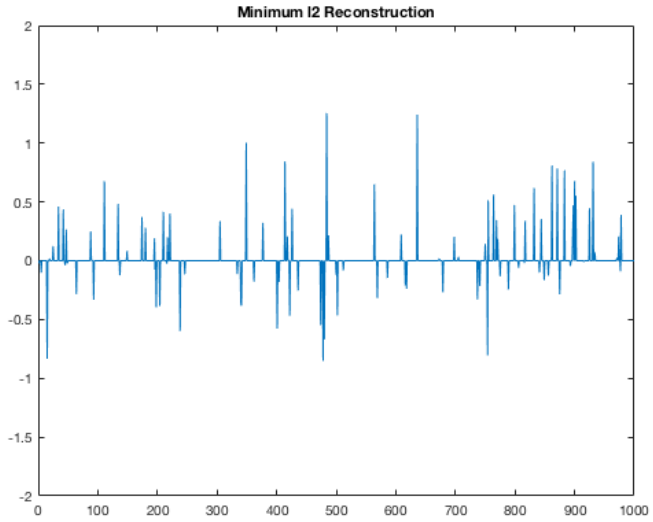
```
end
ans =

   38.9076


ans =

    0.2003
```

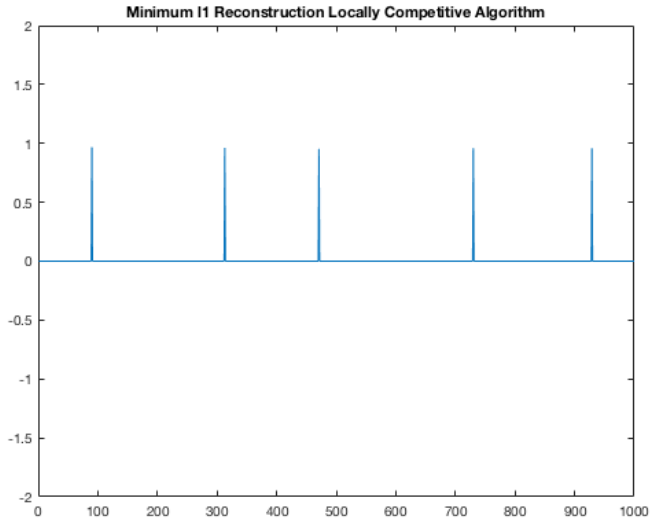

10
20
30
40
50
60
70
80
90
100
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5

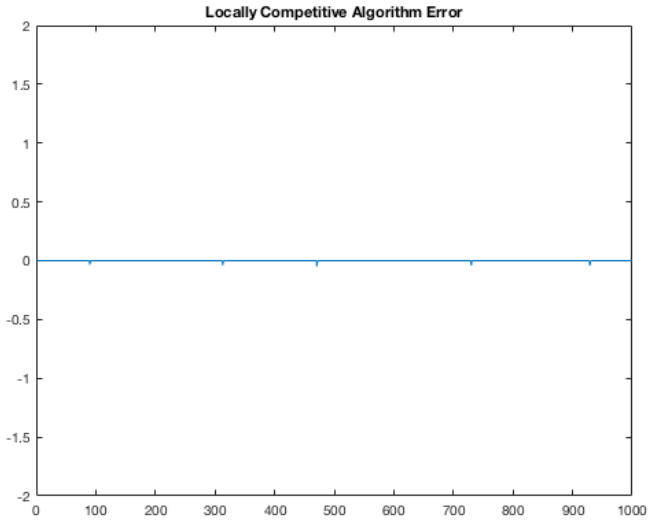

10
20
30
40
50
60
70
80
90
100
100
200
300
400
500
600
700
800
900
1000

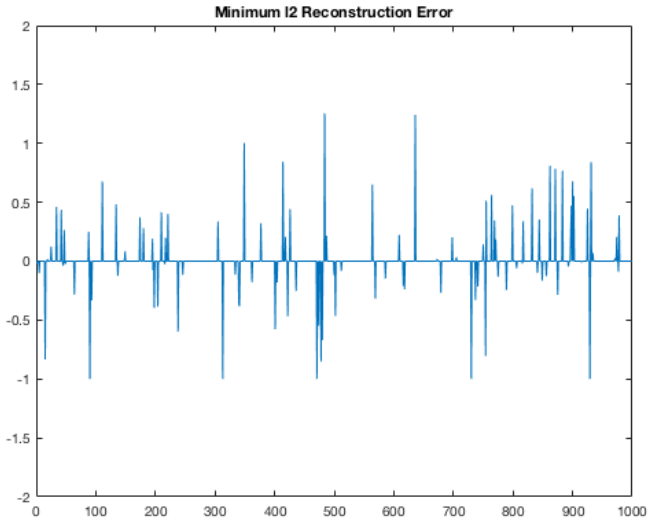

100
200
300
400
500
600
700
800
900
1000
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5

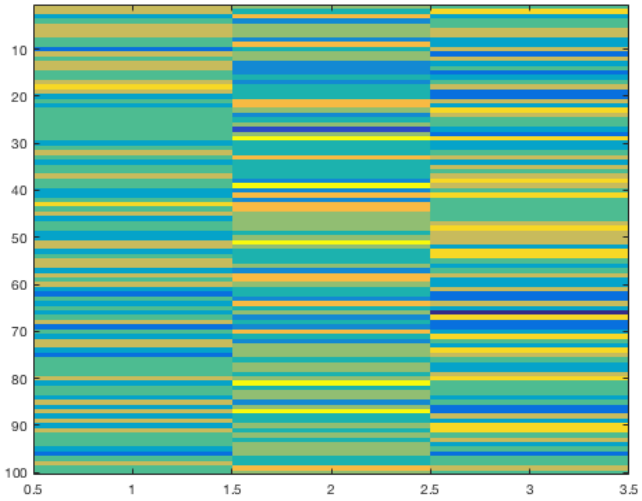

Original Signal
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

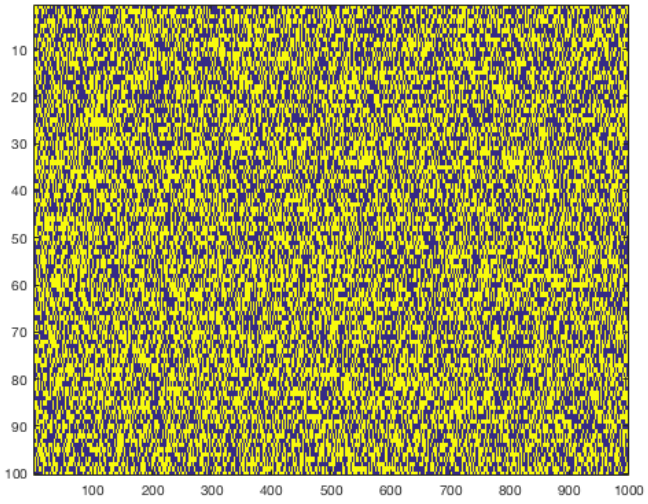

Minimum l2 Reconstruction
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

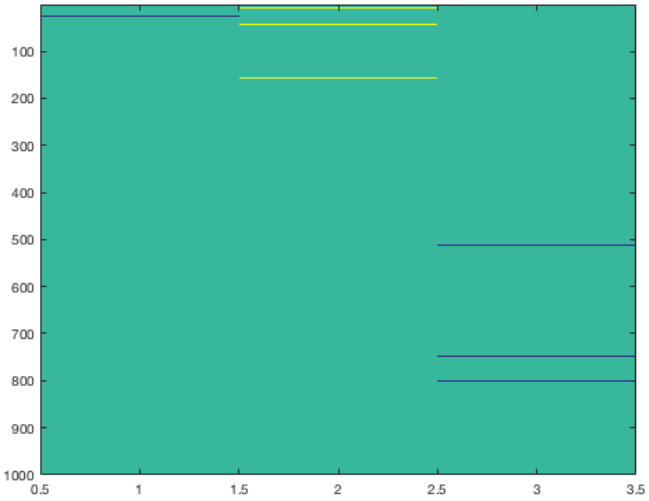

Minimum l1 Reconstruction Locally Competitive Algorithm
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

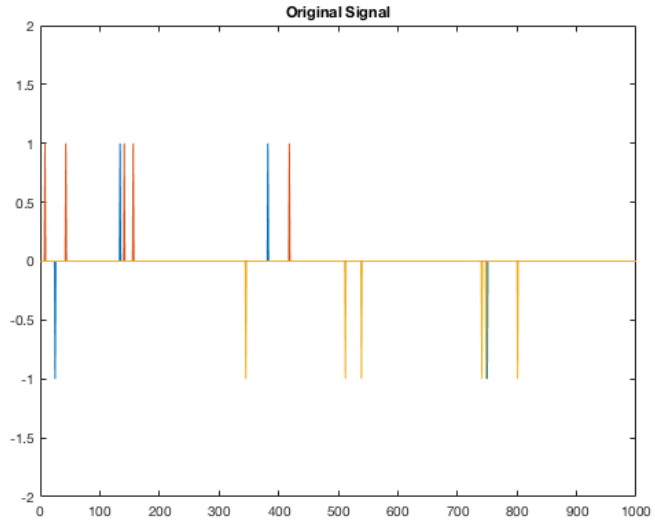

Locally Competitive Algorithm Error
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

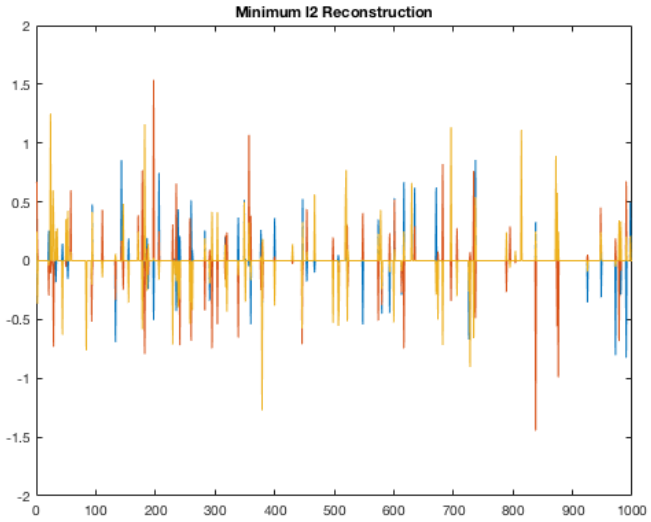

Minimum l2 Reconstruction Error
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

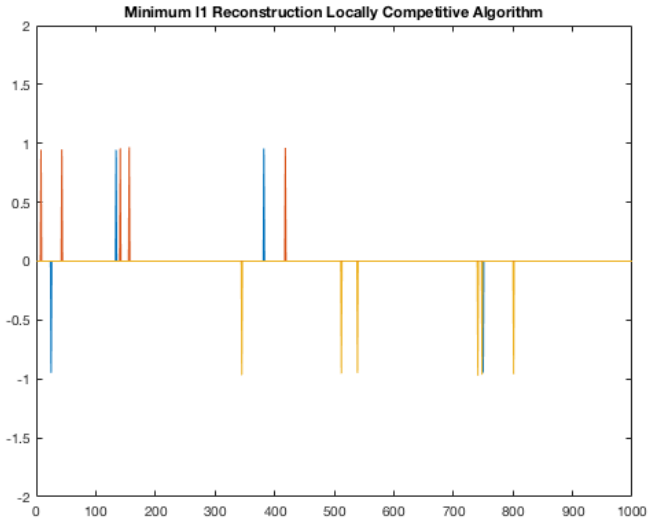

10
20
30
40
50
60
70
80
90
100
0.5
1
1.5
2
2.5
3
3.5

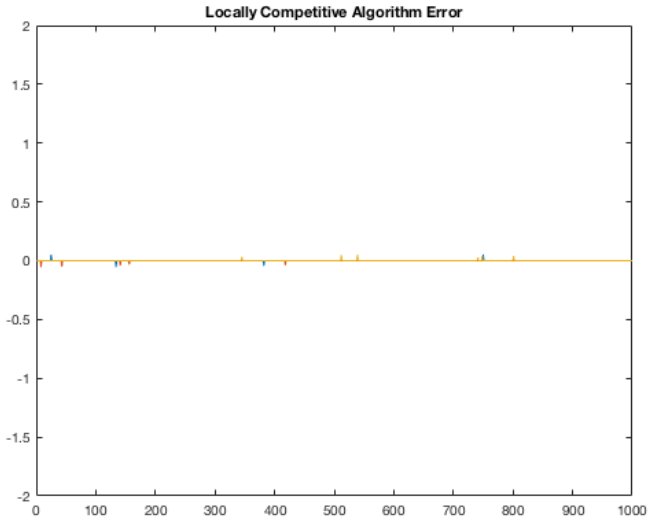

10
20
30
40
50
60
70
80
90
100
100
200
300
400
500
600
700
800
900
1000

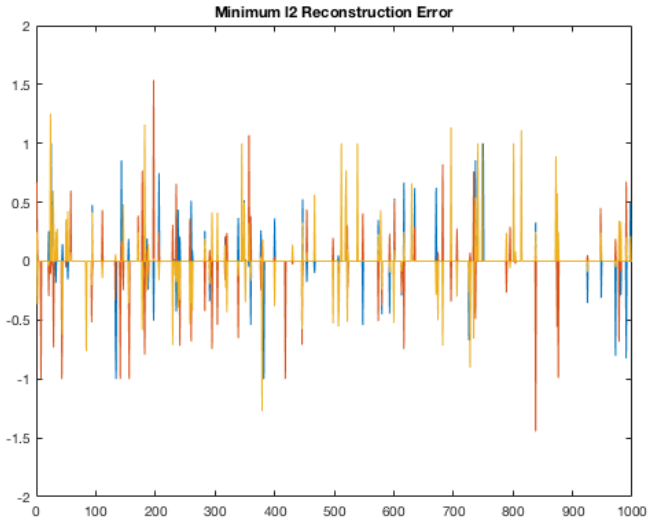

100
200
300
400
500
600
700
800
900
1000
0.5
1
1.5
2
2.5
3
3.5

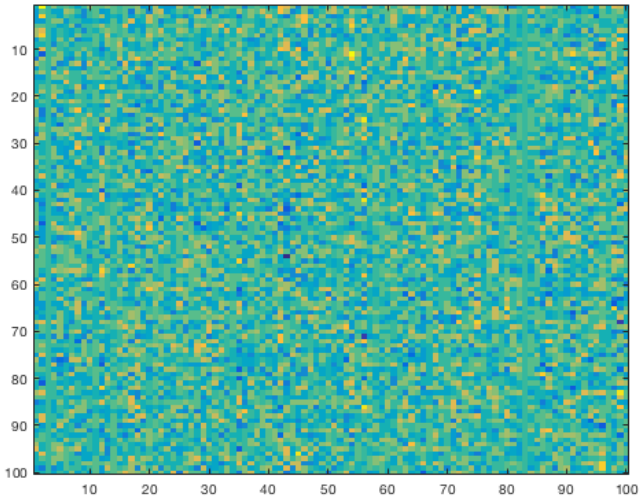

Original Signal
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

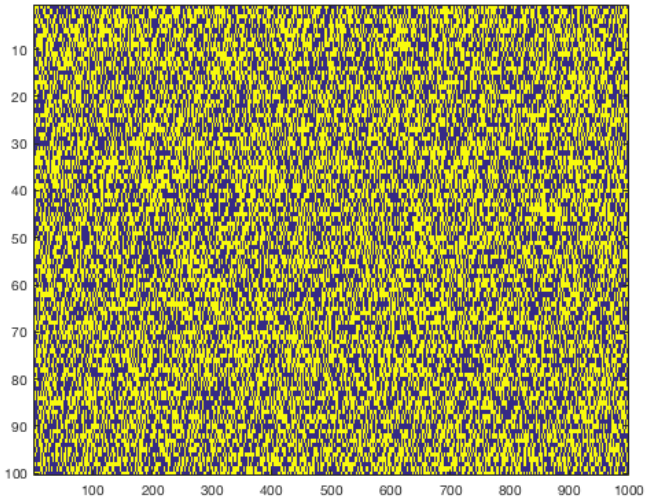

Minimum l2 Reconstruction
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

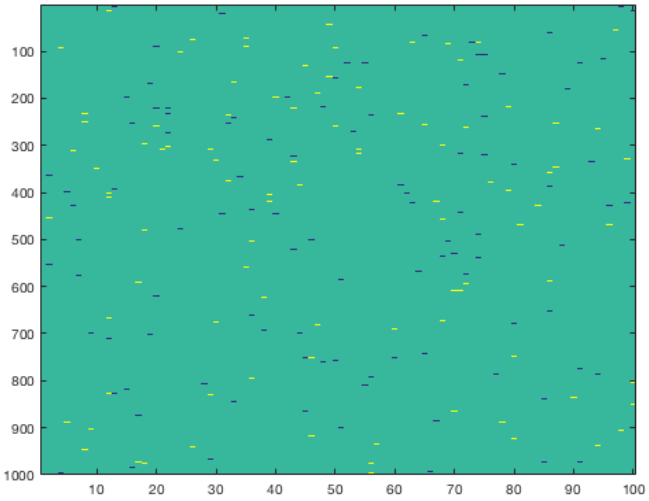

Minimum l1 Reconstruction Locally Competitive Algorithm
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

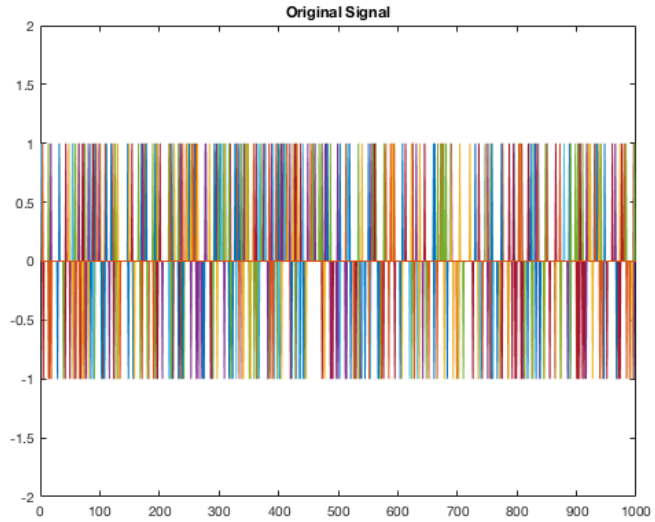

Locally Competitive Algorithm Error
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

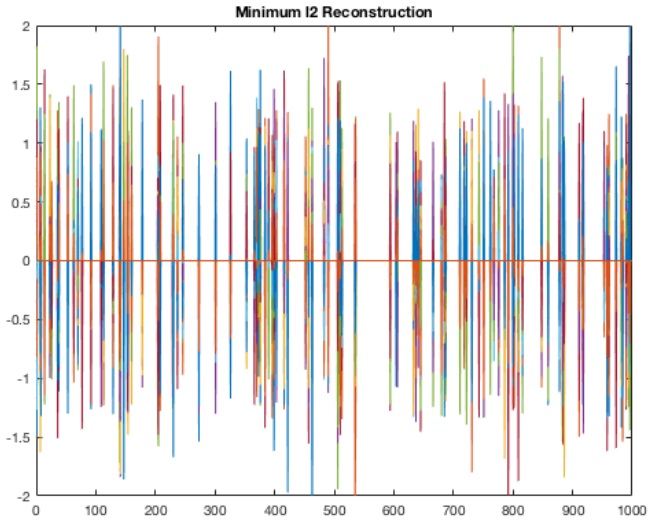

Minimum l2 Reconstruction Error
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000

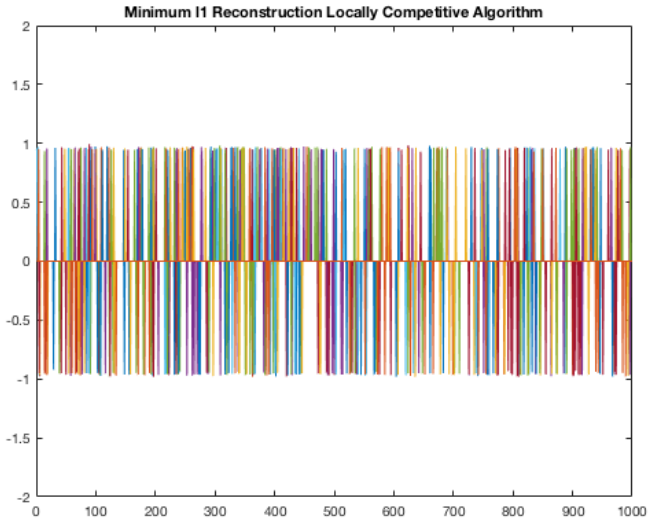

10
20
30
40
50
60
70
80
90
100
10
20
30
40
50
60
70
80
90
100

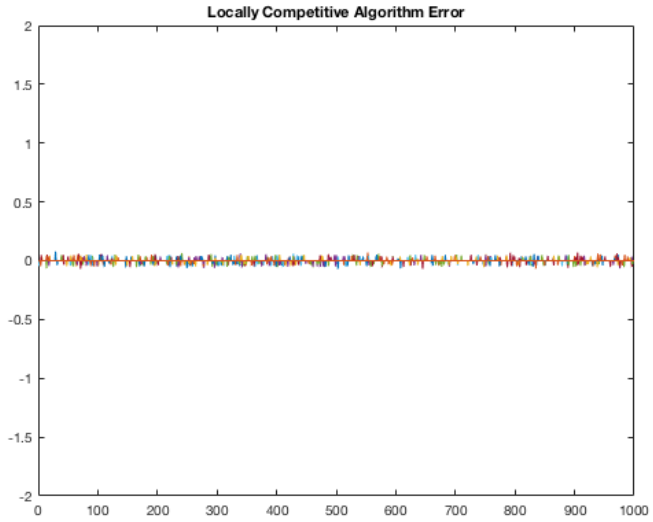

10
20
30
40
50
60
70
80
90
100
100
200
300
400
500
600
700
800
900
1000

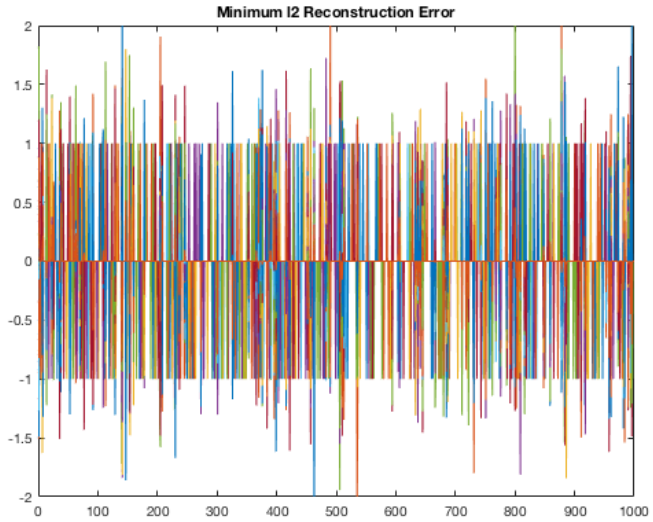

100
200
300
400
500
600
700
800
900
1000
10
20
30
40
50
60
70
80
90
100


Original Signal
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000


Minimum l2 Reconstruction
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000


Minimum l1 Reconstruction Locally Competitive Algorithm
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900
1000





Compressed Audio:

```
Fs = 40000;

t = (1:Fs/8)'/Fs;

x = cos(2*pi*200*t)+cos(2*pi*440*t);


n = length(x);
```

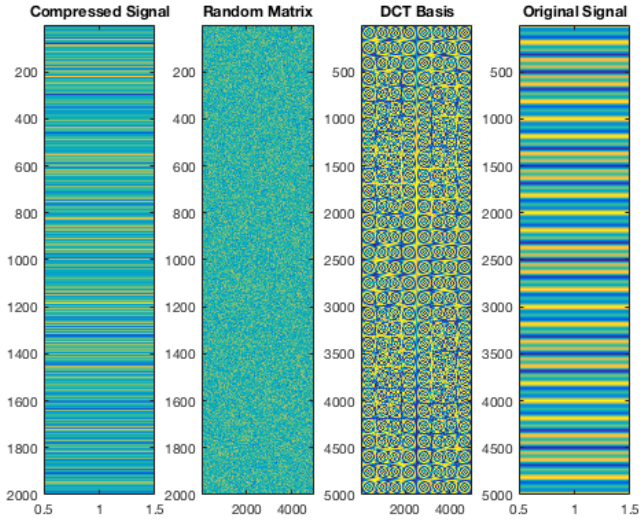
```
m = 2000; %number of samples

Phi = randn(m,n);
Psi = idct(eye(n));

y = Phi*x;

Theta=Phi*Psi;

lambda=2;
t = 1;
h = 0.0001;
d = h/t;
D = Theta;

u = zeros(size(D,2),1);

for i=1:100

    a = ( u - sign(u).*(lambda) ) .* ( abs(u) > (lambda) );

    u = u + d * ( D' * ( y - D*a ) - u - a ) ;

end

s2=a;
```

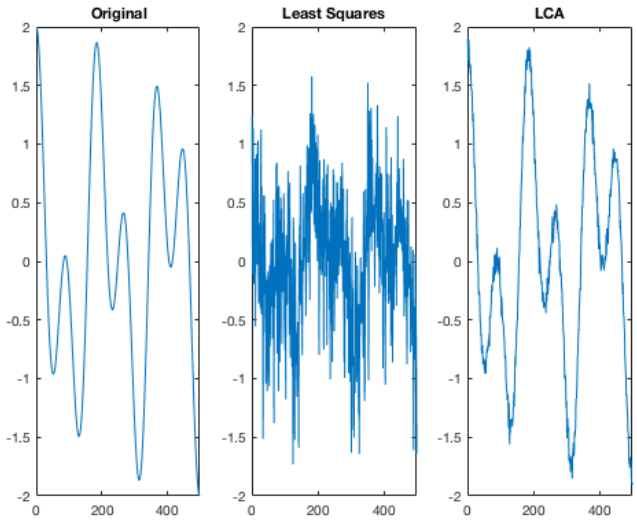
```
s1 = pinv(Theta)*y;


x1=Psi*s1;
x2=Psi*s2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Visualization:

```
figure(1)
subplot(141)
imagesc(y)
title('Compressed Signal')

subplot(142)
imagesc(Phi)
title('Random Matrix')

subplot(143)
imagesc(Psi)
title('DCT Basis')

subplot(144)
imagesc(x)
title('Original Signal')
```

```
figure(2)
%figure('name','Compressive sensing image reconstructions')
subplot(1,3,1), plot(x(1:500)), title('Original'),
axis([0, 500, -2, 2])
subplot(1,3,2), plot(x1(1:500)), title('Least Squares'),
axis([0, 500, -2, 2])
subplot(1,3,3), plot(x2(1:500)), title('LCA'),
axis([0, 500, -2, 2])
colormap jet
```





Sonification

```
sound([x],Fs)
pause(1)
sound([x1],Fs)
pause(1)
sound([x2],Fs);
```

"The uncertainty principle protects quantum mechanics. Heisenberg recognized that if it were possible to measure the momentum and the position simultaneously with a greater accuracy, the quantum mechanics would collapse. So he proposed that it must be impossible. Then people sat down and tried to figure out ways of doing it, and nobody could figure out a way to measure the position and the momentum of anythinga screen, an electron, a billiard ball, anythingwith any greater accuracy. Quantum mechanics maintains its perilous but accurate existence." Richard Feynman[52]

Chapter 6

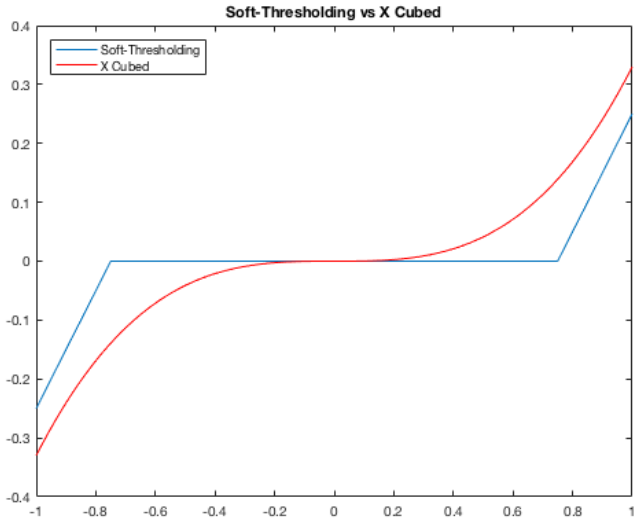
X^3 Dictionary Learning

It has been shown that"a third-order power law best described the transformation of membrane potential to firing rate"[82] "Experimental data, though, seem to favor a third-power law"[82]

"In 7 out of 10 neurons a third order power law best described the data. In one animal, no singleparameter model provided a good fit. In another, a sixth-order nonlinearity even closer to exponential than third order was required. In the last, exponential and third-power models could not be distinguished statistically."[82]

```
lambda=0.75;

u=-1:0.01:1;

a = ( u - sign(u).*(lambda) ) .* ( abs(u) > (lambda) );

plot(u,a)

hold on

plot(u,0.33*u.^3,'r')
```

```
title('Soft-Thresholding vs X Cubed')
legend('Soft-Thresholding', 'X Cubed','Location','NorthWest')
```



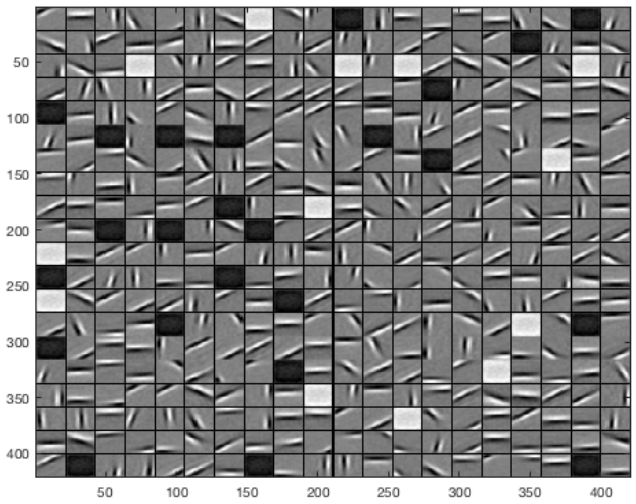

```
function Hahn_Thesis_LCA_Dictionary_Simple_2D

load('IMAGES.mat')

I=IMAGES;

patch_size=400;
neurons=400;
batch_size=1000;

% k=0.1; %not needed with cubic sparsity function
```

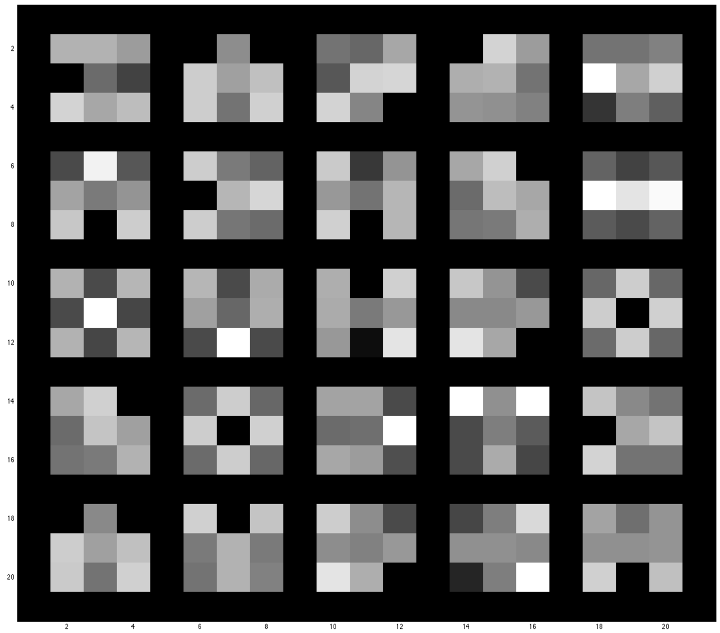
```
W = single(randn(patch_size, neurons)); %Initialize Random Dictionary

for j=1:1000

    X=create_batch(I,patch_size,batch_size);

    W = nc(W); %Normalize Columnns

    a = W'*X; %Feature Activations
    a = nc(a); %Normalize Columnns

    a=0.3*a.^3;   %Cubic function acts as threshold

    W = W + ((X-W*a)*a'); %Update Dictionary

end



imagesc(filterplot(W))
colormap(gray)
drawnow()
snapnow;

end

function X = nc(X)
%normalizecolumns
```

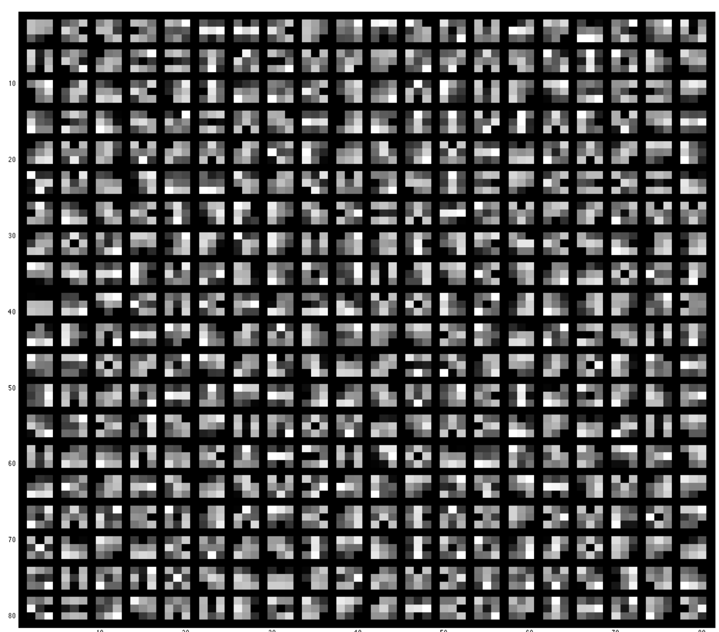
```
X = ones(size(X,1),1)*sqrt(ones./sum(X.*X)).*X;

end

% http://www.cns.nyu.edu/~david/courses/perceptionGrad/Readings/Carandini-NRN2012.
%Normalization as a canonical neural computation



function X=normcol(X)

X=X*diag(1./sqrt(sum(X.^2,1)));

end

function I=create_batch(Images,patch_size,batch_size)

[imsize, imsize, num_Images] = size(Images);

border=10;
patch_side = sqrt(patch_size);

I = zeros(patch_size,batch_size);

im_num= ceil(num_Images * rand());
```

```
for i=1:batch_size

    row = border + ceil((imsize-patch_side-2*border) * rand());
    col = border + ceil((imsize-patch_side-2*border) * rand());

    I(:,i) = reshape(Images(row:row+patch_side-1, col:col+patch_side-1, im_num),[p



end



end

function [D] = filterplot(X)

X=X';

[m n] = size(X);



w = round(sqrt(n));
h = (n / w);

c = floor(sqrt(m));
r = ceil(m / c);
```

```
p = 1;


D = - ones(p + r * (h + p),p + c * (w + p));


k = 1;
for j = 1:r
    for i = 1:c
        D(p + (j - 1) * (h + p) + (1:h), p + (i - 1) * (w + p) + (1:w)) = reshape(
        k = k + 1;
    end


end


end
```







6.0.1 2D Block LCA

%%

%--%

```
%
% Machine Perception and Cognitive Robotics Laboratory
%
%     Center for Complex Systems and Brain Sciences
%
%               Florida Atlantic University
%
%-------------------------------------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------%
%
% Locally Competitive Algorithms Demonstration
% Using natural images data, see:
% Rozell, Christopher J., et al.
% "Sparse coding via thresholding and
% local competition in neural circuits."
% Neural computation 20.10 (2008): 2526-2563.
%
%-------------------------------------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Hahn_Thesis_LCA_Dictionary_Simple_2D_Block

clear all
close all
clc
```

```
load('IMAGES.mat')

I=IMAGES;

patch_size=400;
neurons=400;
batch_size=1000;

% k=0.1; %not needed with cubic sparsity function

W = randn(patch_size, neurons); %Initialize Random Dictionary

for j=1:3000

    X=create_batch(I,patch_size,batch_size);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    W = normcol(W); %Normalize Columnns


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    a = W'*X; %Feature Activations

    a = nc(a); %Normalize Columnns
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%     a = a.*(abs(a) > k); % Hard Threshold

%     a = sign(a).*((abs(a)-k)+abs((abs(a)-k)))/2; %Soft Threshold

    a=0.3*a.^3;   %Cubic function acts as threshold

    if j > 100
        a=blocksparse_vec(a,10);
        a=blocksparse_vec(a,5);
    end


    W = W + ((X-W*a)*a'); %Update Dictionary

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end


    imagesc(filterplot(W))
    colormap(gray)
    drawnow()
    snapnow
```

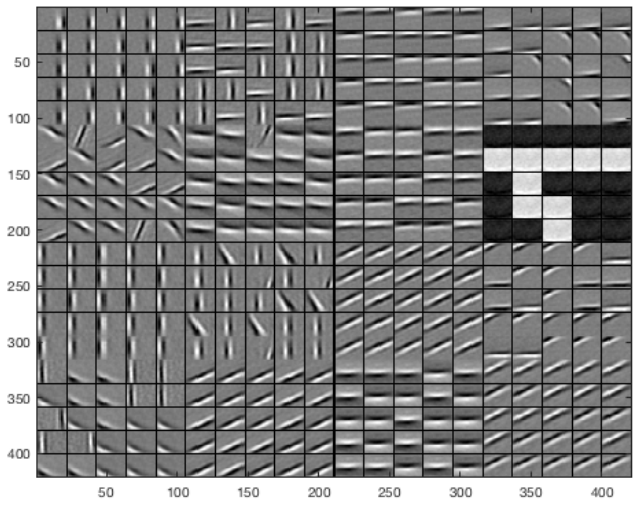
```
end


function X = nc(X)
%normalizecolumns

X = ones(size(X,1),1)*sqrt(ones./sum(X.*X)).*X;

end


function X=normcol(X)

X=X*diag(1./sqrt(sum(X.^2,1)));

end


function XX = blocksparse_vec(XX,blocksize)

s=size(XX(:,1));
n=sqrt(s(1));
```

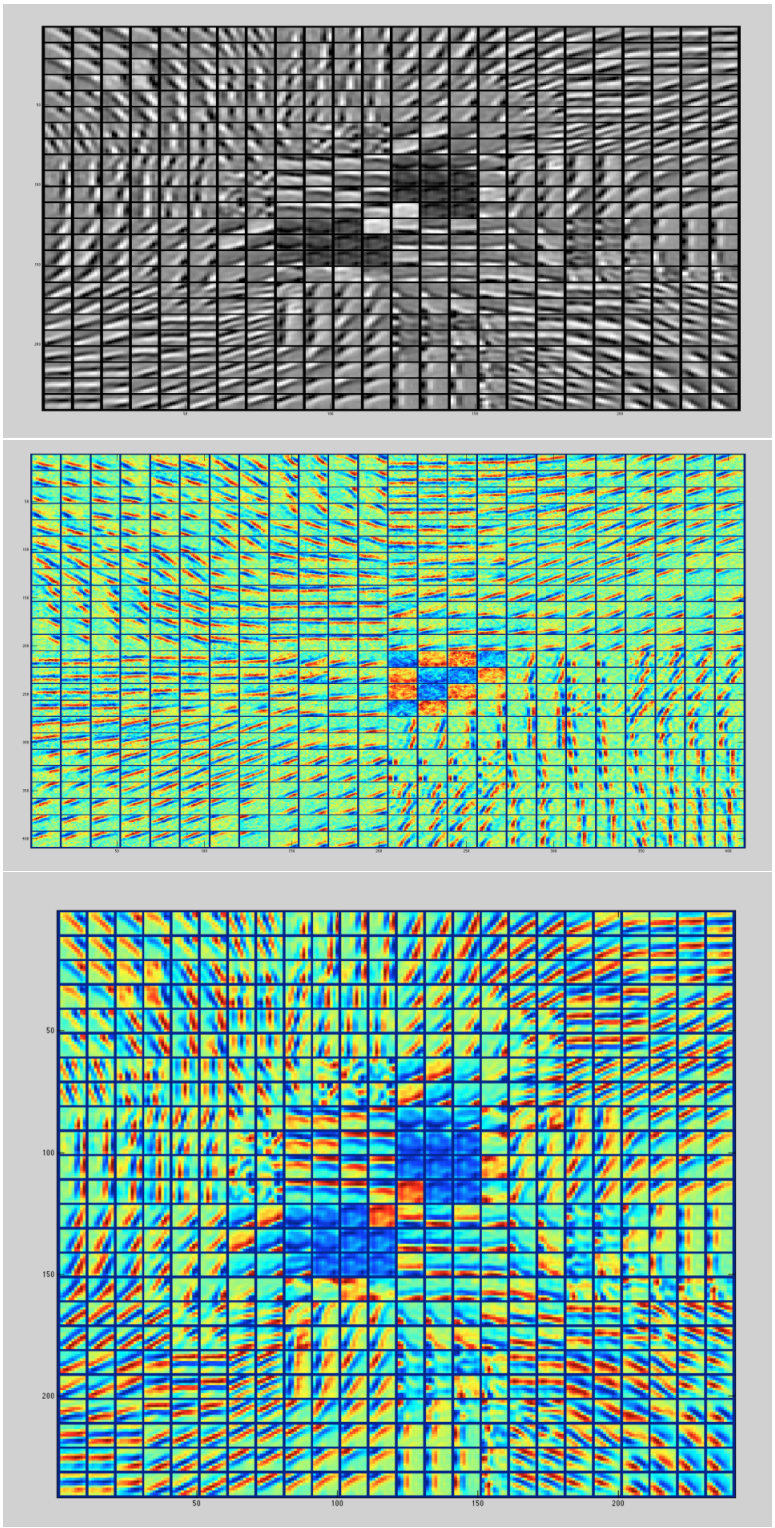
```
blocks=n/blocksize;

A = reshape(1:n^2,[n n])';
B = im2col1(A,[n/blocks n/blocks]);

for i=1:size(XX,2)

    X=reshape(XX(:,i),[n n]);

    [m1 m2]=max(sum(abs(X(B)),1));

    D=0*B;

    D(:,m2)=X(B(:,m2));

    XX(:,i)=reshape(im2col1(D,[blocksize blocks]),s);

end

end



function E = im2col1(A,blocksize)

r = blocksize(1);
c = blocksize(2);
```

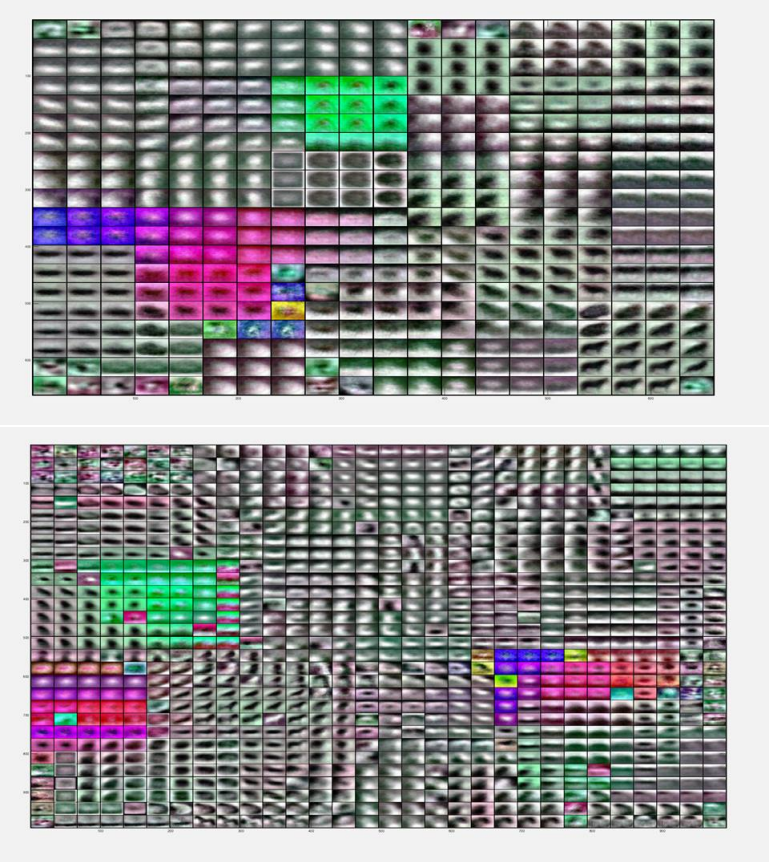
```
e = r*c;

B = zeros(size(A,1)+((mod(size(A,1),r)~=0)*(r - mod(size(A,1),r))),size(A,2)+((mod
B(1:size(A,1),1:size(A,2)) = A;

C = reshape(B,r,size(B,1)/r,[]);
D = reshape(permute(C,[1 3 2]),size(C,1)*size(C,3),[]);
E = reshape(permute(reshape(D,e,size(D,1)/e,[]),[1 3 2]),e,[]);

return;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function I=create_batch(Images,patch_size,batch_size)

[imsize, imsize, num_Images] = size(Images);

border=10;
patch_side = sqrt(patch_size);

I = zeros(patch_size,batch_size);

im_num= ceil(num_Images * rand());
```

```
for i=1:batch_size

    row = border + ceil((imsize-patch_side-2*border) * rand());
    col = border + ceil((imsize-patch_side-2*border) * rand());

    I(:,i) = reshape(Images(row:row+patch_side-1, col:col+patch_side-1, im_num),[p



end



end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [D] = filterplot(X)

X=X';

[m n] = size(X);



w = round(sqrt(n));
h = (n / w);

c = floor(sqrt(m));
```

```
r = ceil(m / c);


p = 1;


D = - ones(p + r * (h + p),p + c * (w + p));


k = 1;
for j = 1:r
    for i = 1:c
        D(p + (j - 1) * (h + p) + (1:h), p + (i - 1) * (w + p) + (1:w)) = reshape(
        k = k + 1;
    end


end


end
```




50
100
150
200
50
100
150
200

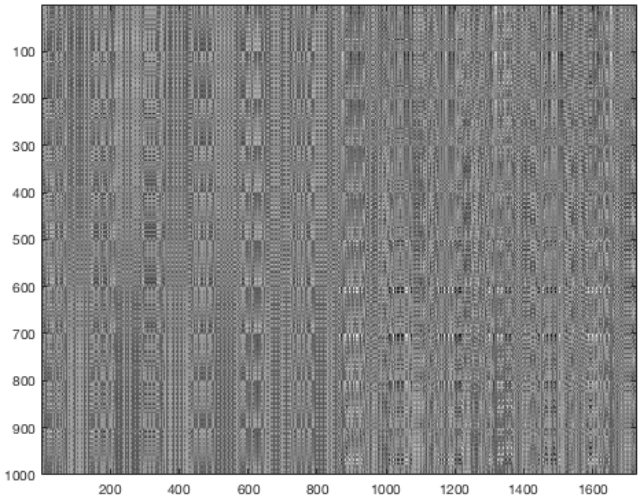


6.0.2 3D Block LCA

```
function Hahn_Thesis_LCA_Dictionary_Simple_3D

clear all

close all

clc


k=0.01;

patch_size=1000;

neurons=1728%256;

batch_size=100;
```

```
W = randn(patch_size, neurons);

load('MRI_ABIDE_Patches.mat')

for j=1:3000

j
    r=randperm(size(patches,2));
    X=patches(:,r(1:batch_size));
    X=whiten_patches(X);

    W = nc(W);



    a = W'*X;

    a = nc(a);

    a = a.^3;

    dW = ((X-W*a)*a');



    if j > 10

        dW =  patch2block(dW);
```

```
        dW =  block_sparse(dW,60);
        dW =  block2patch(dW);

    end

    W = W + 1000*dW;

end



    imagesc(nc(W))
    colormap(gray)
    snapnow()

neurons =

        1728
```




```
W = nc(W);
C = patch2block(W);
cd('/Users/williamedwardhahn/Desktop/thesis')
save_nii(make_nii(C),'3Dpatches_block6020.nii')
end
```

```
function B=patch2block(P)%patches matrix to 3D block cube

%1000 Rows, 1000 Columns In
%100,100,100 Volume Out;
%patchsize=[10,10,10];
%stride=[10,10,10];
load('3D_Volume_Key.mat')
B=zeros(120,120,120);
```

```
B(K)=P;

end


function P=block2patch(B)%3D block cube to patches matrix
%100,100,100 Volume In
%1000 Rows, 1000 Columns Out
patchsize=[10,10,10];
stride=[10,10,10];
P=video2col2(B,patchsize,stride);

end

function C=block_sparse(A,b)

% b=20;

patchsize=[b,b,b];
stride=[b,b,b];

B=video2col2(A,patchsize,stride);
K=video2col2(reshape(1:length(A(:)),size(A)),patchsize,stride);
C=zeros(size(A));
[B1,B2]=sort(sum(abs(B),1),'descend');
B(:,B2(2:end))=0;%winner take all
```

```
C(K)=B;

end


function X = nc(X)
%normalizecolumns

X = ones(size(X,1),1)*sqrt(ones./sum(X.*X)).*X;

end


function X=whiten_patches(X)

% Whiten Images in Matlab
% http://xcorr.net/2013/04/30/whiten-images-in-matlab/

X = bsxfun(@minus,X,mean(X)); %remove mean
fX = fft(fft(X,[],2),[],3); %fourier transform of the images
spectr = sqrt(mean(abs(fX).^2)); %Mean spectrum
X = ifft(ifft(bsxfun(@times,fX,1./spectr),[],2),[],3); %whitened X


end
```

```
function X=nc2(X)
%normalizecolumns
X = X*diag(1./sqrt(sum(X.^2,1)));
end
```

6.1 COLOR IMAGE SPARSE DICTIONARY LEARNING USING LCA

To demonstrate the utility of this work we choose the "challenging application of identifying butterflies in natural imagery" [86]. This database contains 619 images of seven different classes of butterflies (Papilionoidea): Admiral, Black Swallowtail, Machaon, Monarch, Peacock, and Zebra.

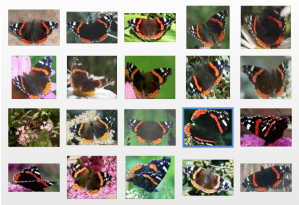


Figure 6.1: Admiral butterfly sample images [86].

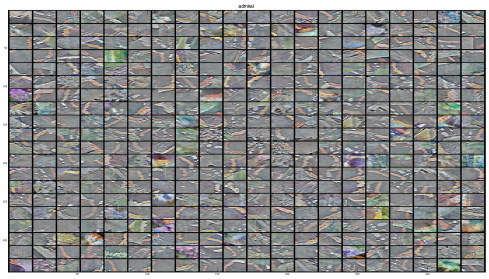


Figure 6.2: Admiral butterfly sample image patches.

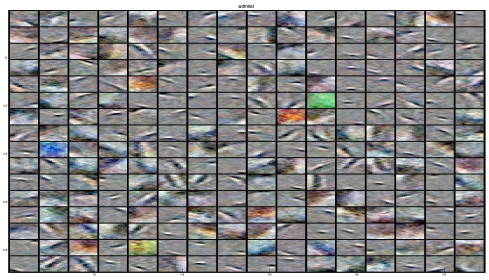


Figure 6.3: Dictionary filters trained on admiral image patches.

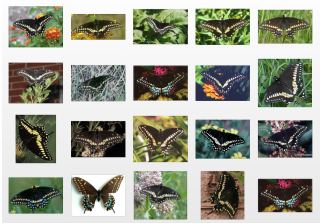


Figure 6.4: Black swallowtail butterfly sample images [86].

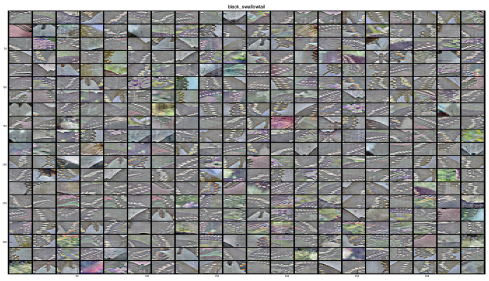


Figure 6.5: Black swallowtail butterfly sample image patches.

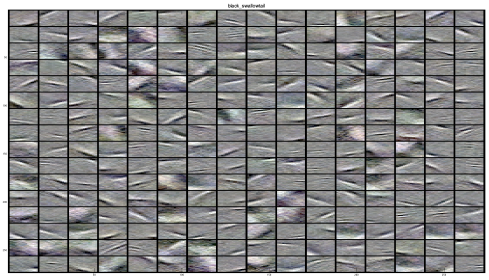


Figure 6.6: Dictionary filters trained on black swallowtail image patches.

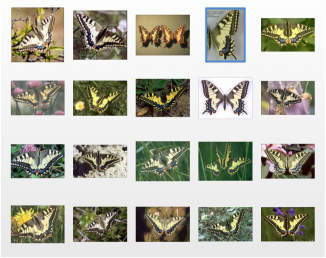


Figure 6.7: Machaon butterfly sample images [86].

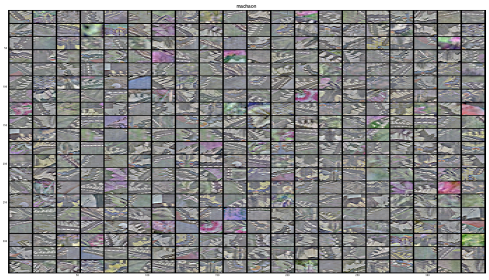


Figure 6.8: Machaon butterfly sample image patches.

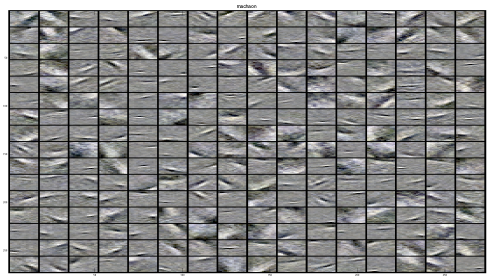


Figure 6.9: Dictionary filters trained on machaon butterfly image patches.

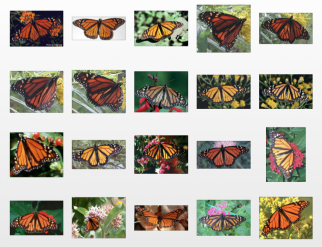


Figure 6.10: Monarch butterfly sample images [86].

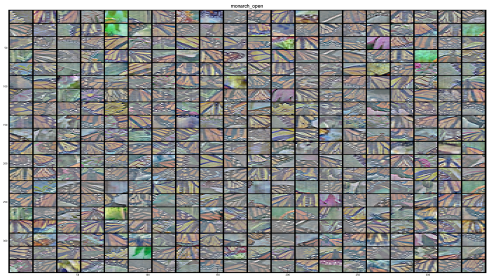


Figure 6.11: Monarch butterfly sample image patches.

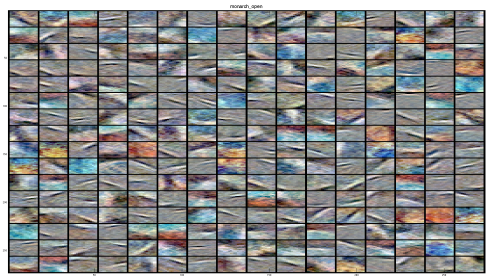


Figure 6.12: Dictionary filters trained on monarch butterfly image patches.

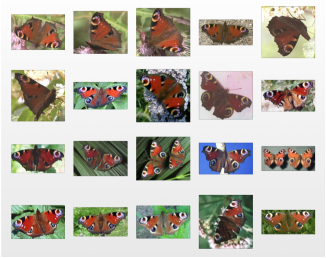


Figure 6.13: Peacock butterfly sample images [86].

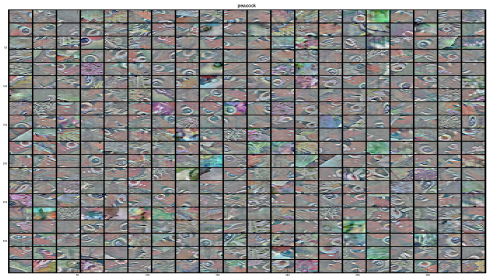


Figure 6.14: Peacock butterfly sample image patches.

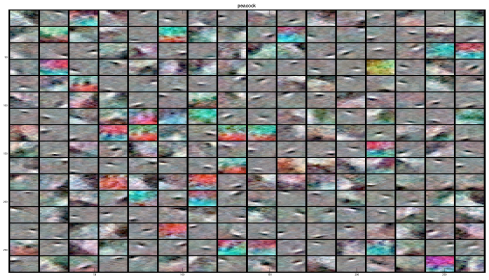


Figure 6.15: Dictionary filters trained on peacock butterfly image patches.

Figure 6.16: Zebra butterfly sample images [86].

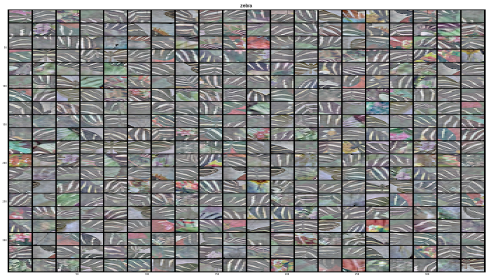


Figure 6.17: Zebra butterfly sample image patches.

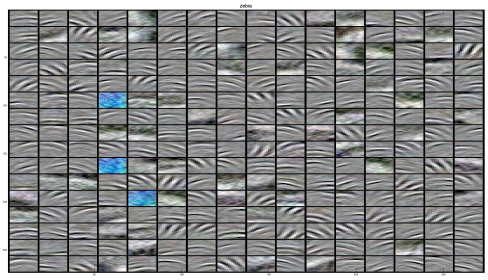


Figure 6.18: Dictionary filters trained on zebra butterfly image patches.

The images are color JPEG, of variable resolution, extremely diverse in terms of size and quality, motion blur, lack of focus, resampling and compression artifacts are common [86]. "It is crucial to point out that butterfly recognition is beyond the capabilities of many current state-of-the-art recognition systems" [86]. Insect classification will be of great interest to robotic farming systems and general pest control applications.

Chapter 7

Video Classification using Sparse Filtering and Locally Competitive Algorithms

---

Here we present a novel neural network architecture combining a sparse filter model and locally competitive algorithms (LCAs), and demonstrate the networks ability to classify human actions from video. We applied this architecture to train a classifier on categories of motion in human action videos. Inputs to the network were small 3D patches taken from frame differences in the videos. Dictionaries were derived for each action class and then activation levels for each dictionary were assessed during reconstruction of a novel test patch. Overall, classification accuracy was at 95 %. We discuss how this sparse modeling approach provides a natural framework for multi-sensory and multimodal data processing including RGB video, RGBD video, hyper-spectral video, and stereo audio/video streams.

7.1 DATA

To demonstrate the utility of the model, we trained sparse filtering dictionaries on videos of humans in action and then classified unseen videos using LCA. The videos are from a subset of the action database courtesy of KTH, Royal Institute of Technology, in Stockholm, Sweden.[140] A single action was performed by a single person in each video. We selected 279 videos, with variations including many different actors,

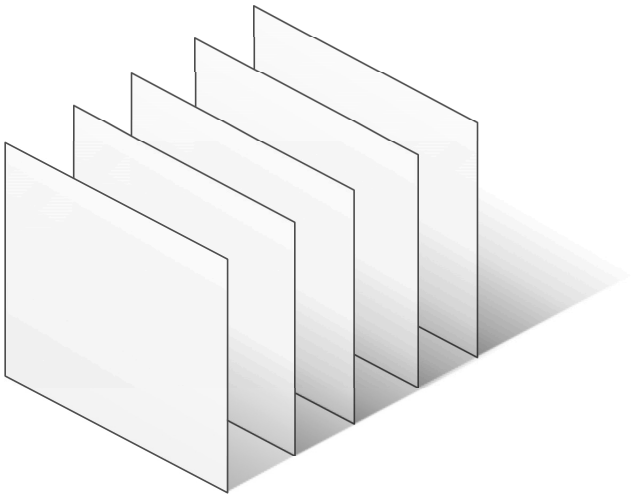


Figure 7.1: Graphic of video frames. Each slice represents the difference between two consecutive frames.

multiple lighting conditions, and dynamic zooming. Filmed with a static camera in 8 bit grayscale, the neutral environments included a variety of indoor and outdoor lighting, with many natural variations and shadows. The average video length was four seconds and 100 frames (25 fps). The resolution was 160 x120 pixels. Inputs to the network were small space-time volumes (3D patches), consisting of blocks of 15 pixels x 15 pixels x 7 frames, taken from frame differences of the videos. (see Figs. 6 - 7.) The space-time patches were sorted by their signal energy, and the top 1000 patches from each video were selected for further processing.[140]

7.2 TRAINING PHASE

We implemented Sparse Filtering to train a library of three separate dictionaries, one for each of three action classes: walking, running, and hand-waving.[28]

Fifty atoms were tuned for each dictionary, (see Fig. 8). The sparse filtering objective error was lowered using a quasi-Newton strategy with a small number of iterations

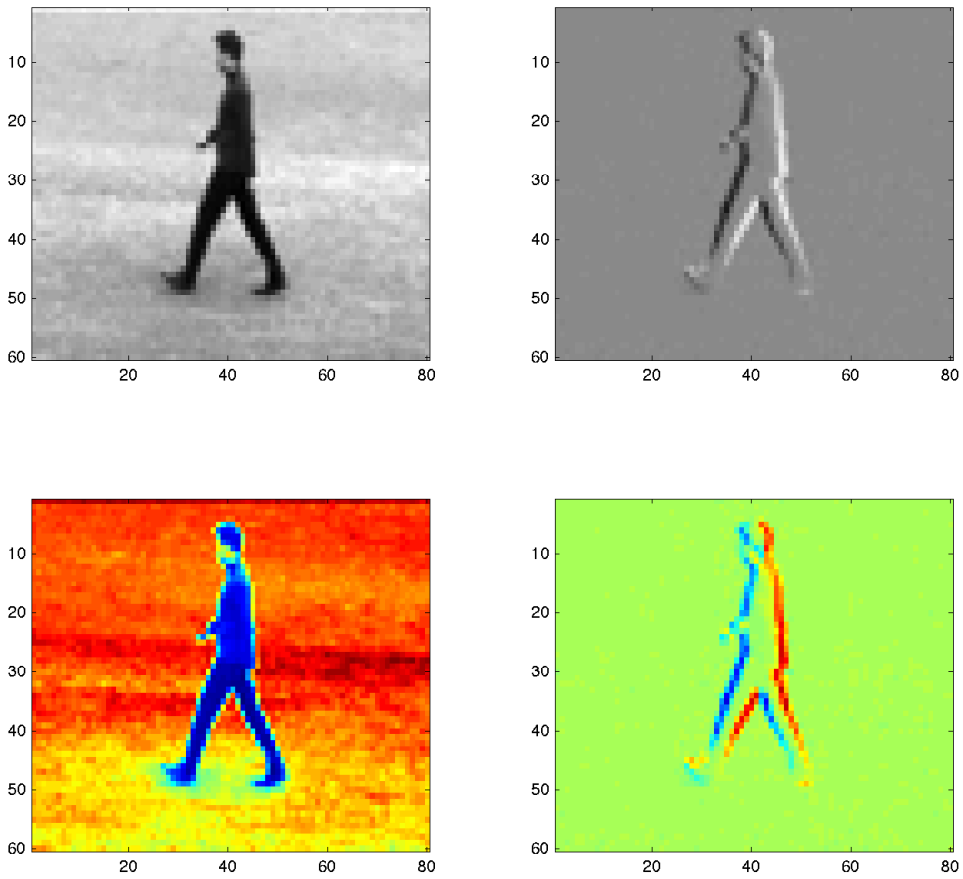


Figure 7.2: Frame differences from KTH dataset, walking action. Image size 60x80 pixels. False coloring for visualization to emphasize spatio-temporal structure.

(< 10) to prevent overfitting.[139] The library was formed by concatenating the three action class dictionaries into a single dictionary consisting of 150 dictionary atoms (i.e. filters, receptive fields). The training was based on patches from 150 videos. All experiments were performed in Matlab.

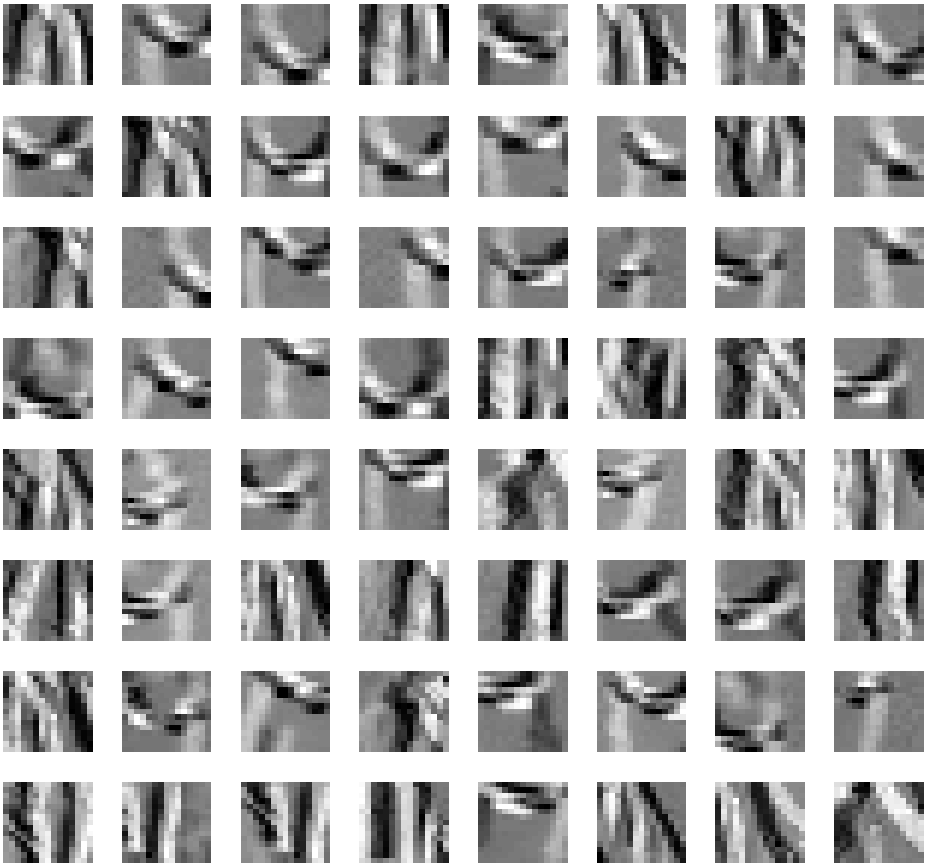


Figure 7.3: Sixty-four sample raw patches from frame differences. First frame shown only. Patch size 15x15x7.

7.3 TESTING PHASE

A total of 129 testing videos were decomposed into patches (15x15x7) as the training videos. The library was used to sort unlabeled patches back into the three categories with the LCA network. Originating from Hubel and Wiesel's work on complex cells in the visual cortex, the idea of feature pooling, which combines the responses of multiple feature detectors, is now standard practice in computer vision.[18]
Our network undergoes ℓ_1 pooling (summing the absolute values) for each of the three known dictionary sections, resulting in a new vector that describes the contribution of each action class to the reconstruction. (see Figs. 3 - 5.) The low dimensional (classes = 3) vectors that resulted from the pooling were used to train a second set of dictionaries for each action class.

The output from the second layer of LCA was pooled over the action classes, and the patch class was assigned in a winner-take-all fashion. Hence the test video class was determined by the majority patch label. We trained and tested on a three-class subset of the KTH video dataset with 129 tests videos and the network correctly labeled 123 videos. See the confusion matrix below.

7.4 CONCLUSION

We have described how to derive spatial and orientation-tuned filters computationally by combining the unsupervised feature learning technique, known as Sparse Filtering, with the atomic decomposition technique, known as Locally Competitive Algorithms. We have shown this combination has utility in the video processing domain as an efficient mechanism for managing multispectral images and video datasets. Both Sparse Filtering and LCA are neurologically inspired; thus, in addition to their utility in video

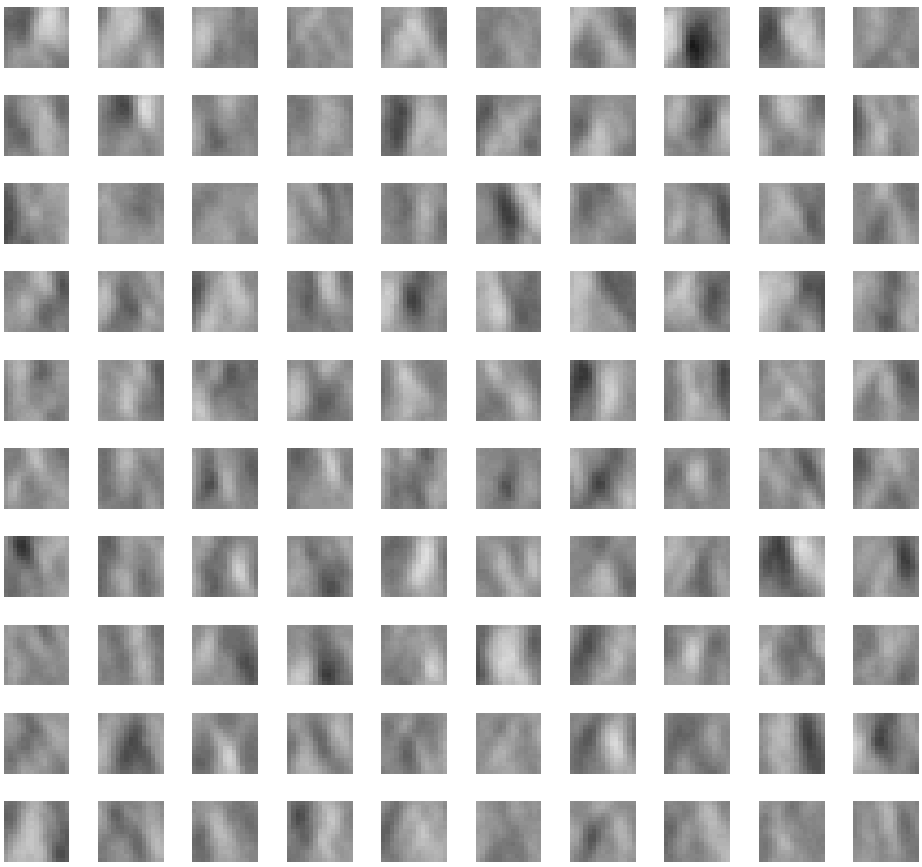


Figure 7.4: Sixty-four dictionary atoms trained by Sparse Filtering on patches of frame differences. Each patch represents the first frame of a spatio-temporal receptive field, and each pixel represents the dendritic weight for that portion of the receptive field. First frame shown only. Patch size 15x15x7.

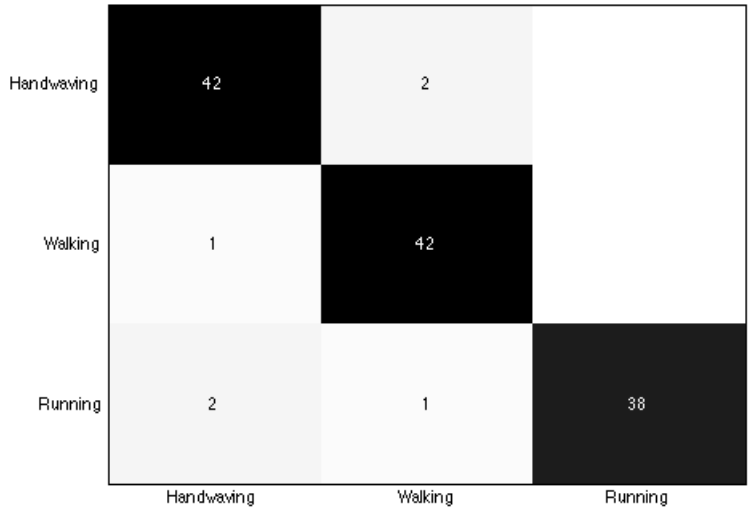


Figure 7.5: Confusion matrix for three action video categories: handwaving, walking, and running.

processing, they provide valuable new frameworks and insights into theoretical visual neuroscience. Future work will include extending dictionary learning to multispectral and multimodal datasets, including RGB video, RGBD video, hyperspectral video, and multi-sensory streams. Implementations in graphics processing units (GPUs) or field programmable analog arrays (FPAAs) could provide real-time systems for industries such as medical imaging analysis and autonomous vehicles navigation. Given that sparse filtering requires no data-specific preprocessing and no hyper-parameter tuning it would seem an ideal candidate to model receptive neurons in hardware for the purposes of unsupervised feature learning.

BIBLIOGRAPHY

[1] Scott Aaronson. *Quantum computing since Democritus*. Cambridge University Press, 2013.

[2] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines*. New York, NY; John Wiley and Sons Inc., 1988.

[3] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):92, 2013.

[4] John Allman, Francis Miezin, and E McGuinness. Stimulus specific responses from beyond the classical receptive field: neurophysiological mechanisms for local-global comparisons in visual neurons. *Annual review of neuroscience*, 8(1):407–430, 1985.

[5] Philip W Anderson et al. More is different. *Science*, 177(4047):393–396, 1972.

[6] Sanjeev Arora, Rong Ge, Tengyu Ma, and Ankur Moitra. Simple, efficient, and neural algorithms for sparse coding. *arXiv preprint arXiv:1503.00778*, 2015.

[7] Joseph J Atick and A Norman Redlich. Towards a theory of early visual processing. *Neural Computation*, 2(3):308–320, 1990.

[8] Janette Atkinson, Bruce Hood, John Wattam-Bell, Shirley Anker, and Johanna Tricklebank. Development of orientation discrimination in infancy. *Perception*, 17(5):587–595, 1988.

[9] Aurèle Balavoine. Implementation of the locally competitive algorithm on a field programmable analog array. 2009.

[10] Aurele Balavoine, Christopher Rozell, and Justin Romberg. Discrete and continuous-time soft-thresholding for dynamic signal recovery. 2015.

[11] Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. The johnson-lindenstrauss lemma meets compressed sensing. *Submitted manuscript, June*, 2006.

[12] Richard G Baraniuk and Michael B Wakin. Random projections of smooth manifolds. *Foundations of computational mathematics*, 9(1):51–77, 2009.

[13] Caroline Blais, Martin Arguin, and Frédéric Gosselin. Human visual processing oscillates: Evidence from a classification image technique. *Cognition*, 128(3):353–362, 2013.

[14] Colin Blakemore and F W Campbell. On the existence of neurones in the human visual system selectively sensitive to the orientation and size of retinal images. 1969.

[15] Colin Blakemore and Grahame F Cooper. Development of the brain depends on the visual environment. 1970.

[16] Avrim Blum. Random projection, margins, kernels, and feature-selection. In *Subspace, Latent Structure and Feature Selection*, pages 52–68. Springer, 2006.

[17] Giuseppe Boccignone and Mario Ferraro. Modelling gaze shift as a constrained random walk. *Physica A: Statistical Mechanics and its Applications*, 331(1):207–218, 2004.

[18] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 111–118, 2010.

[19] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[20] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[21] Neil Bruce and John Tsotsos. Saliency based on information maximization. *Advances in neural information processing systems*, 18:155, 2006.

[22] Samuel Butler. Darwin among the machines. 1863.

[23] Gyorgy Buzsaki. *Rhythms of the Brain*. Oxford University Press, USA, 2009.

[24] Emmanuel J Candè and Michael B Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, 2008.

[25] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

[26] Ryan T Canolty, Karunesh Ganguly, Steven W Kennerley, Charles F Cadieu, Kilian Koepsell, Jonathan D Wallis, and Jose M Carmena. Oscillatory phase coupling coordinates anatomically dispersed functional cell assemblies. *Proceedings of the National Academy of Sciences*, 107(40):17356–17361, 2010.

[27] Matteo Carandini and David J Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51–62, 2012.

[28] Alexey Castrodad and Guillermo Sapiro. Sparse modeling of human actions from motion imagery. *International journal of computer vision*, 100(1):1–15, 2012.

[29] James R Cavanaugh, Wyeth Bair, J Anthony Movshon, et al. Nature and interaction of signals from the receptive field center and surround in macaque v1 neurons. *Journal of neurophysiology*, 88(5):2530–2546, 2002.

[30] Gregory J Chaitin. *Thinking about Godel and Turing: Essays on Complexity 1970-2007*. World scientific, 2007.

[31] L Andrew Coward. Towards a theoretical neuroscience. In *Towards a Theoretical Neuroscience: from Cell Chemistry to Cognition*, pages 389–393. Springer, 2013.

[32] Heitor F Credidio, Elisângela N Teixeira, Saulo DS Reis, André A Moreira, and José S Andrade Jr. Statistical patterns of visual search for hidden objects. *Scientific reports*, 2, 2012.

[33] Francis Crick and J Clark. The astonishing hypothesis. *Journal of Consciousness Studies*, 1(1):10–16, 1994.

[34] John G Daugman. Brain metaphor and brain theory. 2001.

[35] Peter J Denning. Great principles of computing. *Communications of the ACM*, 46(11):15–20, 2003.

[36] Agnès Desolneux, Lionel Moisan, and Jean-Michel Morel. *From gestalt theory to image analysis: a probabilistic approach*, volume 34. Springerverlag New York, 2008.

[37] Gordana Dodig Crnkovic. Physical computation as dynamics of form that glues everything together. *Information*, 3(2):204–218, 2012.

[38] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[39] David Donoho and Jared Tanner. Observed universality of phase transitions in high-dimensional geometry, with implications for modern data analysis and signal processing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 367(1906):4273–4293, 2009.

[40] David L Donoho and Philip B Stark. Uncertainty principles and signal recovery. *SIAM Journal on Applied Mathematics*, 49(3):906–931, 1989.

[41] David L Donoho, Yaakov Tsaig, Iddo Drori, and Jean-Luc Starck. Sparse solution of underdetermined systems of linear equations by stagewise orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 58(2):1094–1121, 2012.

[42] Marco F Duarte, Mark A Davenport, Dharmpal Takhar, Jason N Laska, Ting Sun, Kevin E Kelly, Richard G Baraniuk, et al. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25(2):83, 2008.

[43] Robert J Durrant and Ata Kabán. Random projections as regularizers: Learning a linear discriminant ensemble from fewer observations than dimensions. In *ACML*, pages 17–32, 2013.

[44] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[45] Gerald M Edelman and Joseph A Gally. Reentry: a key mechanism for integration of brain function. *Frontiers in integrative neuroscience*, 7, 2013.

[46] Albert Einstein. On the generalized theory of gravitation. *Scientific American*, 182:13–17, 1950.

[47] Yonina C Eldar and Gitta Kutyniok. *Compressed sensing: theory and applications*. Cambridge University Press, 2012.

[48] Steve K Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, P Merolla, S Chandra, et al. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *International Joint Conference on Neural Networks (IJCNN). IEEE*, 2013.

[49] Ruogu Fang, Tsuhan Chen, Dimitris Metaxas, Pina Sanelli, and Shaoting Zhang. Sparsity techniques in medical imaging. *Computerized Medical Imaging and Graphics*, 46:1, 2015.

[50] Edward A Feigenbaum, Julian Feldman, et al. *Computers and thought*. New York, 1963.

[51] Chrisantha Fernando and Sampsa Sojakka. Pattern recognition in a bucket. In *Advances in Artificial Life*, pages 588–597. Springer, 2003.

[52] Richard P Feynman, Robert B Leighton, Matthew Sands, and EM Hafner. The feynman lectures on physics; vol. i. *American Journal of Physics*, 33(9):750–752, 1965.

[53] David J Field, Anthony Hayes, and Robert F Hess. Contour integration by the human visual system: Evidence for a local association field. *Vision research*, 33(2):173–193, 1993.

[54] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[55] Pascal Fries. A mechanism for cognitive dynamics: neuronal communication through neuronal coherence. *Trends in cognitive sciences*, 9(10):474–480, 2005.

[56] Pascal Fries, John H Reynolds, Alan E Rorie, and Robert Desimone. Modulation of oscillatory neuronal synchronization by selective visual attention. *Science*, 291(5508):1560–1563, 2001.

[57] Steve FURBER. Bio-inspired massively-parallel computation. *Parallel Computing: On the Road to Exascale*, 27:3, 2016.

[58] Fabrizio Gabbiani, Holger G Krapp, Christof Koch, and Gilles Laurent. Multiplicative computation in a visual neuron sensitive to looming. *Nature*, 420(6913):320–324, 2002.

[59] Dashan Gao and Nuno Vasconcelos. Bottom-up saliency is a discriminant process. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–6. IEEE, 2007.

[60] V Gaudet and A Rapley. Iterative decoding using stochastic computation. *Electronics Letters*, 39(3):299–301, 2003.

[61] Rafal Goebel, Ricardo G Sanfelice, and A Teel. Hybrid dynamical systems. *Control Systems, IEEE*, 29(2):28–93, 2009.

[62] Irving John Good. Speculations concerning the first ultraintelligent machine. *Advances in computers*, 6(99):31–83, 1965.

[63] R Paul Gorman and Terrence J Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural networks*, 1(1):75–89, 1988.

[64] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 399–406, 2010.

[65] William Edward Hahn, Stephanie Lewkowitz, Daniel C Lacombe Jr, and Elan Barenholtz. Deep learning human actions from video via sparse filtering and locally competitive algorithms. *Multimedia Tools and Applications*, 74(22):10097–10110, 2015.

[66] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of operations research*, 13(2):311–329, 1988.

[67] Biao Han, Hao Zhu, and Youdong Ding. Bottom-up saliency based on weighted sparse coding residual. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 1117–1120. ACM, 2011.

[68] Jonathan Harel, Christof Koch, Pietro Perona, et al. Graph-based visual saliency. *Advances in neural information processing systems*, 19:545, 2007.

[69] Darrall Henderson, Sheldon H Jacobson, and Alan W Johnson. The theory and practice of simulated annealing. In *Handbook of metaheuristics*, pages 287–319. Springer, 2003.

[70] Ewald Hering. On memory as a universal function of organized matter. 1870.

[71] RF Hess, A Hayes, and DJ Field. Contour integration and cortical processing. *Journal of physiology, Paris*, 97(2-3):105, 2003.

[72] JP Holdren, E Lander, and H Varmus. Report to the president and congress: Designing a digital future: federally funded research and development in networking and information technology. *Presidents Council of Advisors on Science and Technology*, 2010.

[73] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[74] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[75] Guang-Bin Huang. What are extreme learning machines? filling the gap between frank rosenblatts dream and john von neumanns puzzle. *Cognitive Computation*, 7(3):263–278, 2015.

[76] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004.

[77] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of Neurophysiology*, 1965.

[78] Laurent Itti and Pierre Baldi. A principled approach to detecting surprising events in video. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 631–637. IEEE, 2005.

[79] Laurent Itti and Christof Koch. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision research*, 40(10):1489–1506, 2000.

[80] Laurent Itti, Christof Koch, Ernst Niebur, et al. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

[81] Judson P Jones and Larry A Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of neurophysiology*, 58(6):1233–1258, 1987.

[82] Matthias S Keil. Emergence of multiplication in a biophysical model of a wide-field visual neuron for computing object approaches: Dynamics, peaks, & fits. In *Advances in Neural Information Processing Systems*, pages 469–477, 2011.

[83] James Kennedy. Particle swarm optimization. In *Encyclopedia of Machine Learning*, pages 760–766. Springer, 2010.

[84] Christof Koch. Scientific life: Trendstalk. *Trends in Cognitive Sciences*, 16(9), 2012.

[85] Laurent Larger et al. Photonic information processing beyond turing: an opto-electronic implementation of reservoir computing. 2012.

[86] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Semi-local affine parts for object recognition. In *British Machine Vision Conference (BMVC'04)*, pages 779–788. The British Machine Vision Association (BMVA), 2004.

[87] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[88] Stéphane Leduc and W Deane Butcher. The mechanism of life. *Archives of the Roentgen Ray*, 16(1):30–31, 1911.

[89] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2006.

[90] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

[91] Tai Sing Lee. Computations in the early visual cortex. *Journal of Physiology-Paris*, 97(2):121–140, 2003.

[92] Gordon E Legge and John M Foley. Contrast masking in human vision. *JOSA*, 70(12):1458–1471, 1980.

[93] Yin Li, Yue Zhou, Lei Xu, Xiaochao Yang, and Jie Yang. Incremental sparse saliency detection. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3093–3096. IEEE, 2009.

[94] Edo Liberty. Fast random projections. 2007.

[95] Antoine Liutkus, David Martina, Sébastien Popoff, Gilles Chardon, Ori Katz, Geoffroy Lerosey, Sylvain Gigan, Laurent Daudet, and Igor Carron. Imaging with nature: Compressive imaging using a multiply scattering medium. *Scientific reports*, 4, 2014.

[96] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[97] Computing Machinery. Computing machinery and intelligence-am turing. *Mind*, 59(236):433, 1950.

[98] Bruce J MacLennan. A review of field computation. *Knoxville: Department of Electrical Engineering of Computer Science. University of Tennessee*, 2007.

[99] Bryan Magee. *Confessions of a philosopher: a personal journey through Western philosophy from Plato to Popper*. Random House Digital, Inc., 1999.

[100] Julien Mairal, Francis Bach, and Jean Ponce. Sparse modeling for image and vision processing. *arXiv preprint arXiv:1411. [100]3230*, 2014.

[101] Julien Mairal, Francis Bach, Jean Ponce, et al. Foundations and trends in computer graphics and vision. *Foundations and Trends in Computer Graphics and Vision*, 8(2-3):85–283, 2014.

[102] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 689–696. ACM, 2009.

[103] Marc Massar and Serge Massar. Mean-field theory of echo state networks. *Physical Review E*, 87(4):042809, 2013.

[104] John McCarthy. Epistemological problems of artificial intelligence. *Readings in Artificial Intelligence*, page 459, 1987.

[105] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[106] Christine Métin and Douglas O Frost. Visual responses of neurons in somatosensory cortex of hamsters with experimentally induced retinal projections to somatosensory thalamus. *Proceedings of the National Academy of Sciences*, 86(1):357–361, 1989.

[107] Yoan Miche, Benjamin Schrauwen, and Amaury Lendasse. Machine learning techniques based on random projections. In *ESANN*, 2010.

[108] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

[109] Richard D Neidinger. Introduction to automatic differentiation and matlab object-oriented programming. *SIAM review*, 52(3):545–563, 2010.

[110] Andrew Ng, Jiquan Ngiam, Chuan Y. Foo, Yifan Mai, and Caroline Suen. UFLDL Tutorial. *http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial*, 2010.

[111] Jiquan Ngiam, Zhenghao Chen, Sonia A Bhaskar, Pang W Koh, and Andrew Y Ng. Sparse filtering. In *Advances in Neural Information Processing Systems*, pages 1125–1133, 2011.

[112] John Nolan. *Stable distributions: models for heavy-tailed data*. Birkhauser, 2003.

[113] Atsushi Nomura, Makoto Ichikawa, Koichi Okada, Hidetoshi Miike, Tatsunari Sakurai, and Yoshiki Mizukami. Image edge detection with discretely spaced fitzhugh-nagumo type excitable elements. In *Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET), 2011 Joint 3rd Int'l Workshop on*, pages 1–8. IEEE, 2011.

[114] Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

[115] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

[116] Bruno A Olshausen and David J Field. Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4):481–487, 2004.

[117] Dylan Paiton, Sheng Lundquist, William Shainin, Xinhua Zhang, Peter Schultz, and Garrett Kenyon. A deconvolutional competitive algorithm for building sparse hierarchical representations. In *proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) on 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 535–542. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[118] Hélène Paugam-Moisy, Régis Martinez, and Samy Bengio. Delay learning and polychronization for reservoir computing. *Neurocomputing*, 71(7):1143–1158, 2008.

[119] Panagiotis C Petrantonakis and Panayiota Poirazi. A compressed sensing perspective of hippocampal function. *Frontiers in systems neuroscience*, 8, 2014.

[120] Kenneth C Pohlmann. *Principles of digital audio*. Butterworth-Heinemann, 1985.

[121] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, DTIC Document, 1989.

[122] F.L. Putzrath. Processing apparatus utilizing simulated neurons, October 12 1965. US Patent 3,211,832.

[123] Anna S Rakitianskaia and Andries Petrus Engelbrecht. Training feedforward neural networks with dynamic particle swarm optimisation. *Swarm Intelligence*, 6(3):233–270, 2012.

[124] Louis B Rall. Automatic differentiation: Techniques and applications. 1981.

[125] Robert S Rand and DeLiang Wang. A biologically inspired neural oscillator network for geospatial analysis. In *Defense and Security Symposium*, pages 62470N–62470N. International Society for Optics and Photonics, 2006.

[126] Ronald A Rensink. Seeing, sensing, and scrutinizing. *Vision research*, 40(10):1469–1487, 2000.

[127] Ronald A Rensink and James T Enns. Preemption effects in visual search: evidence for low-level grouping. *Psychological review*, 102(1):101, 1995.

[128] Irina Rish and Genady Grabarnik. *Sparse Modeling: Theory, Algorithms, and Applications*. CRC Press, 2014.

[129] Kamil Rocki. Towards machine intelligence. *arXiv preprint arXiv:1603.08262*, 2016.

[130] Edmund T Rolls and Alessandro Treves. The neuronal encoding of information in the brain. *Progress in neurobiology*, 95(3):448–490, 2011.

[131] Christopher Rozell, Don Johnson, Richard Baraniuk, and Bruno Olshausen. Locally competitive algorithms for sparse approximation. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 4, pages IV–169. IEEE, 2007.

[132] Christopher J Rozell, Don H Johnson, Richard G Baraniuk, and Bruno A Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural computation*, 20(10):2526–2563, 2008.

[133] C.J. Rozell, D.H. Johnson, R.G. Baraniuk, B.A. Olshausen, and R.L. Ortman. Analog system for computing sparse codes, 2010. US Patent 7,783,459.

[134] Giovanni Russo and Jean Jacques E Slotine. Global convergence of quorum-sensing networks. *Physical Review E*, 82(4):041919, 2010.

[135] Murray B Sachs, Jacob Nachmias, and John G Robson. Spatial-frequency channels in human vision. *JOSA*, 61(9):1176–1186, 1971.

[136] RF Salazar, NM Dotson, SL Bressler, and CM Gray. Content-specific fronto-parietal synchronization during visual working memory. *Science*, 338(6110):1097–1100, 2012.

[137] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[138] Juergen Schmidhuber. Don't forget randomness is still just a hypothesis. *Nature*, 439(7075):392–392, 2006.

[139] Mark Schmidt, Glenn Fung, and Romer Rosales. Optimization methods for l1-regularization. *University of British Columbia, Technical Report TR-2009*, 19, 2009.

[140] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE, 2004.

[141] Peter F Schultz, Dylan M Paiton, Wei Lu, and Garrett T Kenyon. Replicating kernels with a short stride allows sparse reconstructions with fewer independent kernels. *arXiv preprint arXiv:1406.4205*, 2014.

[142] Ivan W Selesnick. Sparse signal restoration. *Proceedings available online at url: http://cnx. org/content/m32168/latest*, 2010.

[143] Thomas Serre. *Learning a dictionary of shape-components in visual cortex: comparison with neurons, humans and machines*. PhD thesis, Massachusetts Institute of Technology, 2006.

[144] Wolf Singer and Charles M Gray. Visual feature integration and the temporal correlation hypothesis. *Annual review of neuroscience*, 18(1):555–586, 1995.

[145] Guoshen Yu Jean-Jacques Slotine. Visual grouping by neural oscillator networks.

[146] Alfred Smee. *Elements of electro-biology, or the voltaic mechanism of man; of electro-pathology, especially of the nervous system; and of electro-therapeutics*. 1849.

[147] Olaf Sporns, Giulio Tononi, and Gerald M Edelman. Modeling perceptual grouping and figure-ground segregation by means of active reentrant connections. *Proceedings of the National Academy of Sciences*, 88(1):129–133, 1991.

[148] Alessandra Staglianò, Gabriele Chiusano, Curzio Basso, and Matteo Santoro. Learning adaptive and sparse representations of medical images. In *Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging*, pages 130–140. Springer, 2010.

[149] Steven H Strogatz. *Sync: The emerging science of spontaneous order*. Hyperion, 2003.

[150] Nicolas Tabareau, Jean-Jacques Slotine, and Quang-Cuong Pham. How synchronization protects from noise. *PLoS computational biology*, 6(1):e1000637, 2010.

[151] Alan M Turing. Intelligent machinery, a heretical theory. *The Turing Test: Verbal Behavior as the Hallmark of Intelligence*, page 105, 1948.

[152] Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[153] Alan Mathison Turing and B Jack Copeland. *The essential Turing: seminal writings in computing, logic, philosophy, artificial intelligence, and artificial life, plus the secrets of Enigma*. Oxford University Press, USA, 2004.

[154] Mauro Ursino, Elisa Magosso, and Cristiano Cuppini. Recognition of abstract objects via neural oscillators: interaction among topological organization, associative memory and gamma band synchronization. *Neural Networks, IEEE Transactions on*, 20(2):316–335, 2009.

[155] I Vilovic, Nikša Burum, and D Milic. Using particle swarm optimization in training neural network for indoor field strength prediction. In *ELMAR, 2009. ELMAR'09. International Symposium*, pages 275–278. IEEE, 2009.

[156] William E Vinje and Jack L Gallant. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science*, 287(5456):1273–1276, 2000.

[157] John Von Neumann. The general and logical theory of automata. *Cerebral mechanisms in behavior*, 1(41):1–2, 1951.

[158] Edward Wallace, Hamid Reza Maei, and Peter E Latham. Randomly connected networks have short temporal memory. *Neural computation*, 25(6):1408–1439, 2013.

[159] DeLiang Wang and Peter Chang. An oscillatory correlation model of auditory streaming. *Cognitive neurodynamics*, 2(1):7–19, 2008.

[160] DeLiang Wang and David Terman. Image segmentation based on oscillatory correlation. *Neural Computation*, 9(4):805–836, 1997.

[161] DeLiang L Wang and Guy J Brown. Separation of speech from interfering sounds based on oscillatory correlation. *Neural Networks, IEEE Transactions on*, 10(3):684–697, 1999.

[162] Wei Wang and Jean-Jacques E Slotine. On partial contraction analysis for coupled nonlinear oscillators. *Biological cybernetics*, 92(1):38–53, 2005.

[163] Weinstein Wolfram. Smooth function.

[164] John Wright, Yi Ma, Julien Mairal, Guillermo Sapiro, Thomas S Huang, and Shuicheng Yan. Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, 98(6):1031–1044, 2010.

[165] Santiago Ramn y Cajal. The structure and connexions of neurons. *Nobel Lecture*, December 12, 1906.

[166] Xin-She Yang, Slawomir Koziel, and Leifur Leifsson. Computational optimization, modelling and simulation: Recent trends and challenges. *Procedia Computer Science*, 18:855–860, 2013.

[167] Shih-Cheng Yen and Leif H Finkel. Extraction of perceptually salient contours by striate cortical networks. *Vision research*, 38(5):719–741, 1998.

[168] Guoshen Yu and J-J Slotine. Visual grouping by neural oscillator networks. *Neural Networks, IEEE Transactions on*, 20(12):1871–1884, 2009.

[169] Ying Yu, Bin Wang, and Liming Zhang. Saliency-based compressive sampling for image signals. *Signal Processing Letters, IEEE*, 17(11):973–976, 2010.

[170] Jiangye Yuan, DeLiang Wang, Bo Wu, Lin Yan, and Rongxing Li. Automatic road extraction from satellite imagery using legion networks. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 3471–3476. IEEE, 2009.

[171] Liang Zhao, Fabricio A Breve, Marcos G Quiles, and Roseli AF Romero. Visual selection and shifting mechanisms based on a network of chaotic wilson-cowan oscillators. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 5, pages 754–762. IEEE, 2007.

[172] Konrad Zuse. *Der Computer*. Springer, 1984.