Methodology and Techniques for Building Modular Brain-Computer Interfaces

by

Jason Cummer
B.Sc., University of Lethbridge, 2006

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Methodology and Techniques for Building Modular Brain-Computer Interfaces

by

Jason Cummer
B.Sc., University of Lethbridge, 2006

ABSTRACT

Commodity brain-computer interfaces (BCI) are beginning to accompany everything from toys and games to sophisticated health care devices. These contemporary interfaces allow for varying levels of interaction with a computer. Not surprisingly, the more intimately BCIs are integrated into the nervous system, the better the control a user can exert on a system. At one end of the spectrum, implanted systems can enable an individual with full body paralysis to utilize a robot arm and hold hands with their loved ones [28, 62]. On the other end of the spectrum, the untapped potential of commodity devices supporting electroencephalography (EEG) and electromyography (EMG) technologies require innovative approaches and further research. This thesis proposes a modularized software architecture designed to build flexible systems based on input from commodity BCI devices. An exploratory study using a commodity EEG provides concrete assessment of the potential for the modularity of the system to foster innovation and exploration, allowing for a combination of a variety of algorithms for manipulating data and classifying results.

Specifically, this study analyzes a pipelined architecture for researchers, starting with the collection of spatio temporal brain data (STBD) from a commodity EEG device and correlating it with intentional behaviour involving keyboard and mouse input. Though classification proves troublesome in the preliminary dataset considered, the architecture demonstrates a unique and flexible combination of a liquid state machine (LSM) and a deep belief network (DBN). Research in methodologies and techniques such as these are required for innovation in BCIs, as commodity devices, processing power, and algorithms continue to improve. Limitations in terms of types of classifiers, their range of expected inputs, discrete versus continuous data, spatial and temporal considerations and alignment with neural networks are also identified.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Star Trek, YouTube, Piper and Gerri Elder, for supporting me in the low moments.

Yvonne Coady, for mentoring, support, encouragement and patience.

NeuroDevNet, Mitacs and Cebas Visual technology, for funding me one way or another.

I do not think there is any thrill that can go through the human heart like that felt by the inventor as he sees some creation of the brain unfolding to success... such emotions make a man forget food, sleep, friends, love, everything.
Nikola Tesla

DEDICATION

Thanks to Basil for keeping my lap warm while I write my thesis. Thanks to Gerri Elder for providing me a future.

Chapter 1

Introduction

There are numerous reasons why the lure of brain-computer interfaces (BCI) is attracting consumer attention. Not only could the technology be critical for people relying on assistive technology, but in a "hands-free" world of mobile technology, ubiquitous interaction could be accomplished if button pushing and key pressing did not rely on manual dexterity. Reading electric fields with electroencephalography (EEG) is one way to record brain activity. Once this activity has been processed and analyzed, *trigger events* can be isolated and associated with different intentional actions. The intentional actions would align directly with mouse and keyboard activity within a conventional system. The reason for this is that the neural activity in the motor cortex that generates electric fieldsis the same activity that propagates to the muscles which actuate the fingers. Each trigger event needs to be mapped in the appropriate way to influence the environment.

Specifically, the critical components in building a BCI include (see Figure 1.1):

1. Large amounts of data output from high resolution EEGs.

2. Transformation of this into a form such that trigger events can be identified.

3. Classification of trigger events.

4. Mapping of trigger events into environmental effectors.

Currently, analyzing brain waves and extracting large amounts of data from high resolution EEGs has proven to be challenging for all but very simple classification problems. Even well known and robust classification algorithms appear to be lacking in this domain. Type one errors, which are rejections of the null hypothesis when it

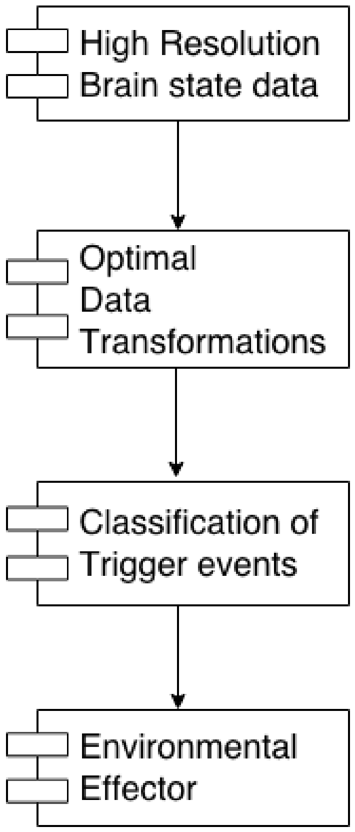

High Resolution Brain state data
Optimal Data Transformations
Classification of Trigger events
Environmental Effector

Figure 1.1: The abstract critical components for a BCI. Some form of capturing the state of the brain is required. Then, based on how you captured your data and what signals you are looking for, the brain state is transformed into an optimal form for classification. Finally, environmental effectors are created, capable of producing your desired outcome (Drawn by Jason Cummer).

is in fact true, are common. This means an activity is not performed, even when the user did intend it. Similarly, type two errors, or failures to reject a null hypothesis that is false, are especially fatiguing and discouraging. That is to say that when a user is just relaxing and an event occurs, the user's anxiety level increases. When the user is trying unsuccessfully to do something in a way that they have done it before, it can tire them out and depress them. This is a big problem for using brain signals as environmental effectors outside of biological systems. Man made systems, which are not wired in like a natural neural network, are not always intuitive.

Though many commodity devices are starting to populate the landscape and can handle very simple input events, this thesis focuses on the problem of how to architect a framework for exploring sophisticated BCIs. The goal is to allow for flexibility and sustainability in the approach as the technology rapidly evolves. This framework accommodates a wide range of non-traditional input devices, such as commodity EEG devices, several methodologies for transformation and representation of data, a range of approaches for the identification of trigger events, any number of classification algorithms and mappings to output behaviours.

In the use-case provided in the thesis, we provide a preliminary exploration with a commodity EEG, domain specific recording software, customized transformation software to generate spike trains, a Liquid State Machine (LSM) [56], and more customized software to map LSM output to Deep Belief Network (DBN) [36] input, see Figure 1.2 for LSM / DBN visualization.

The rest of this thesis is organized in the following way: Chapter 2 provides an overview of the background and related work. Chapter 3 covers the general architecture of such a system and the specific instance we used. Chapter 4 establishes the setup of an example using an LSM and an overview of the results. Chapter 5 is a discussion of the example system, specifically identifying how modularity allows for isolation in troubleshooting and analysis. Chapter 6 is a summary and conclusion chapter that also covers future work.

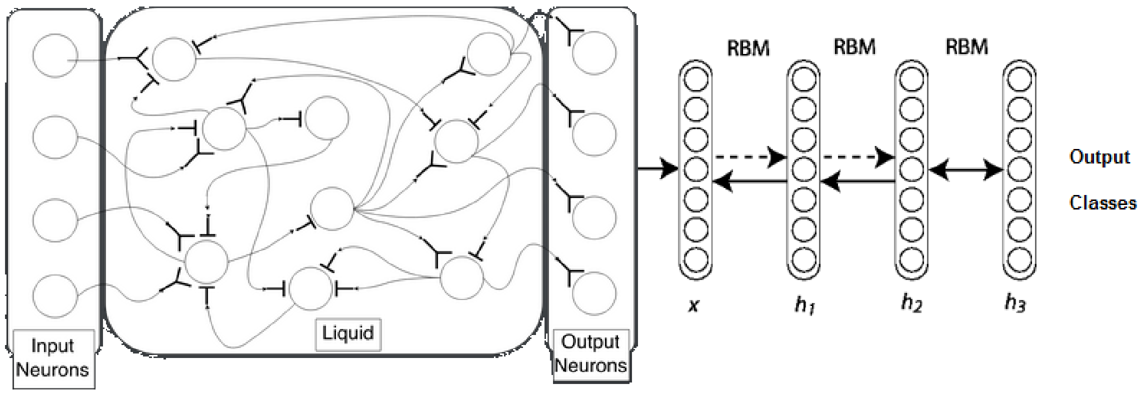


Figure 1.2: This figure shows the combination of the LSM and the DBN in series. The LSM is on the left, and the DBN is on the right. Arrows represent information flow. (Drawn and adapted by Jason Cummer) [64]

Chapter 2

Background and Related Work

This chapter covers the background information relating to brain-computer interfaces (BCI). Some of the various ways of scanning the brain are reviewed. The preparation of electroencephalography (EEG) data is explained. The example BCI system is introduced. Certain algorithms in the discipline of machine learning are covered as they relate to the test system. Artificial neural networks (ANN) are overviewed in a general sense and two types of ANN that are specific to this thesis are covered in more detail.

2.1 Brain Scanning

A number of neurological disorders can leave a person with diminished ability to communicate and function in the world. Individuals with Parkinson's disease, amyotrophic lateral sclerosis (ALS), or locked-in syndrome are in this category [43]. To complete day to day actions, we require both gross and fine movements of our limbs. The tools humans use to interface with the world change these gross and fine movements. There is evidence that the brain can easily remap its sense of self to utilize these tools [34, 25].

Research has shown that it is possible to gather the signals that the brain makes when it initiates action planning and the execution of movements [54]. There are a number of methods for collecting network activity in the brain; these range from subcellular alteration to entirely noninvasive methods. Here we provide an overview of some pros and cons associated with a variety of methods.

Optogenetics [29] - An invasive technology which requires the injection of a gene editing technology (genetic engineering) to add genes to the brain's cells and alter the brain's genome. It also requires the addition of hardware into the cranium to read and write signals to the brain. The advantages that optogenetics brings are high spatial temporal resolution of both input and output signals from cells [37]. These signals can be targeted to specific populations of cells. The cell based targeting is based on promoter sequences in the genome of the cells. The accurate targeting used to insert genes into the right promoter sequences can be achieved with TALENs (transcription activator-like effector nucleases) or zinc finger nuclease based genetic engineering [60]. Specific cells produce internal signals or receive external signals that can activate proteins. Some of these proteins bind to the promoter regions involved in regulating other proteins. When genes added for optogenetics are on these promoter regions they will be transcribed and produce the proteins for optogenetic interfacing. The targeting of specific cells allows researchers to dynamically alter the network dynamics of an organism. Ultimately the network dynamics can be used to drive an effector for a BCI.

Microelectrode Array [54] - A set of electrodes, usually set up in a grid pattern, that is implanted into the brain. Microelectrode arrays have the advantage of being able to record in both high temporal and spatial resolution. With this high temporal and spatial resolution, it is possible to correlate patterns of neurons firing with body movements. Aspects such as limb trajectories and positions can be determined with the recorded brain activity from the microelectrode array. Local field potentials give the electrodes access to neurons in a volume of 65 to 350 μm [33, 47] and a slow ionic current in 0.5 to 3 mm [40]. Many electrodes are needed to record network dynamics from a given region because the individual electrodes have a limited sensing range. This is why many electrodes are combined in an array, forming the microelectrode array. If high spatial temporal resolution of many brain regions is required, the microelectrode array will require more implantation sites. Currently, one of the problems with microelectrode arrays is glial scarring [58]. The cause of this scarring can be the implantation of the electrodes, the material's interactions with the tissue and the motion of the brain tissue against the implanted electrodes [24].

Positron Emission Tomography (PET) - An injected radioactive analog of a biologically active molecule [35], coupled with gamma ray detectors to observe metabolic activity in brain tissue. The brain areas that are actively processing information exhibit greater use of resources, including the radioactive nuclide (radionuclide). By scanning for gamma rays, this brain activity can be detected. One problem with such a system is the use of a relatively large and cumbersome ring of photomultiplier tubes. One would find it difficult to engage in free exploration of the world when utilizing such a large device. Radiation from the radionuclide, though it is from a short-lived radioisotope, is still able to cause cellular damage. A system like this would not be suitable for long term use as a BCI. Another problem is that PET requires a computed tomography scan to map the user's individual cortical structure. PET has low spatial resolution due to the positron traveling through the tissue of interest in an unknown path and the unknown position of the electron with which it collides. Considering the size of the machine and the side effects of radiation, PET is impractical for a personal BCI.

Functional Magnetic Resonance Imaging (fMRI) - Magnetic resonance imaging which uses a large magnet to polarize all of the spins of the electrons of an atom [39]. A smaller magnet is used to resonate an atom of choice with an appropriate resonant frequency. This causes an emission of a radio signal that is recorded and interpreted by a computer. The computer creates an image of the material in the scanner from the radio signal emissions. fMRI uses the blood oxygen level dependant (BOLD) [39] signal to detect where activity is occurring in the brain. The BOLD signal is a measure of the difference between two forms of the protein hemoglobin. The two forms are oxyhemoglobin and deoxyhemoglobin. Oxyhemoglobin is saturated with oxygen. deoxyhemoglobin is devoid of oxygen. As in PET, where neurons use glucose and have to uptake more when they are active, in fMRI the active neurons use oxygen and uptake more from the blood. This desaturates oxyhemoglobin and it becomes deoxyhemoglobin. There is a difference in the magnetic susceptibility of the these two forms of the hemoglobin protein. This is due to the presence or absence of oxygen. The oxyhemoglobin is diamagnetic, while deoxyhemoglobin is paramagnetic. Paramagnetic molecules have a stronger magnetic resonance then diamagnetic molecules. Because the brain uses oxygen at different rates in different regions, the ratio of the two forms of hemoglobin change. The difference in the magnetic resonance can be used to tell the different activity levels in these regions. This difference can be measured to map the location of the activity in the brain.

The temporal resolution of an fMRI is a couple hundred milliseconds [59] and the spatial resolution is one millimetre (mm) voxels [39]. Resolution fine enough to detect specific neuronal firing patterns is ideal, but fMRI does not give you this level of accuracy. At lower resolutions, firing patterns are less useful for interpreting brain activity [54]. You could use fMRI for the input of a BCI but it would be better to use a technology that averages the neural network activity as little as possible.

There are several problems with using fMRI as a BCI. The scanners have to be fairly large due to house a primary magnet large enough to generate a sufficiently powerful magnetic field. The strong magnetic field necessitates an area free from materials that could interfere with the magnet. Also electromagnetic shielding is needed to get the best results from the relatively weak radio signals.

Electrocorticography (ECog) [55] - An invasive technology: electrodes must be implanted in the skull, on the brain itself. A craniotomy is needed to gain access to the brain's surface and to place electrodes over the region of interest. Problems with sampling could occur if the placement of electrodes was not correct when implantation occurred. The resolution and the signal to noise ratio is better for ECog than EEG. Signals are stronger than in an EEG system, as they are not being read through the skull and other tissue between the brain and the electrodes.

Electroencephalography (EEG) [55] - The strength of electric fields are detected on the scalp using electrodes. Ideally the activity of many neuronal axons firing at the same time will produce a field that is strong enough to reach through the skull and the scalp. The strength of the electric fields over time is mapped to a graph and this yields an electroencephalograph. EEG is relatively inexpensive, has high temporal resolution and is noninvasive. A major weakness of EEG is that it has low spatial resolution [63]. Signals are averaged from areas of the brain. This is relative to the number of electrodes in the system. Adding more electrodes increases spatial resolution but also requires more signal processing: each electrode measures its own stream of data. It is also difficult to get signals that originate in sulci (furrows in the brain) as shown in Figure 2.1. Opposing electric fields from neurons in the sulci cancel each other out, making the information of the network in the sulci difficult to read. EEG has a low signal to noise ratio. This means that the signals recorded from the neuronal electric fields are small versus the other electric fields that the electrodes can pick up (unwanted noise). The opposite, a high signal to noise ratio, is when the signal you are interested in is very strong and the other signals in the same medium are weak. An example of a high signal to noise ratio is listening for thunder in a rain storm. The low signal to noise ratio is an unfortunate consequence of living in the electromagnetic soup of the industrial world. The signals of the EEG are often flooded with electromagnetic radiation from all of the electronics in our environment.

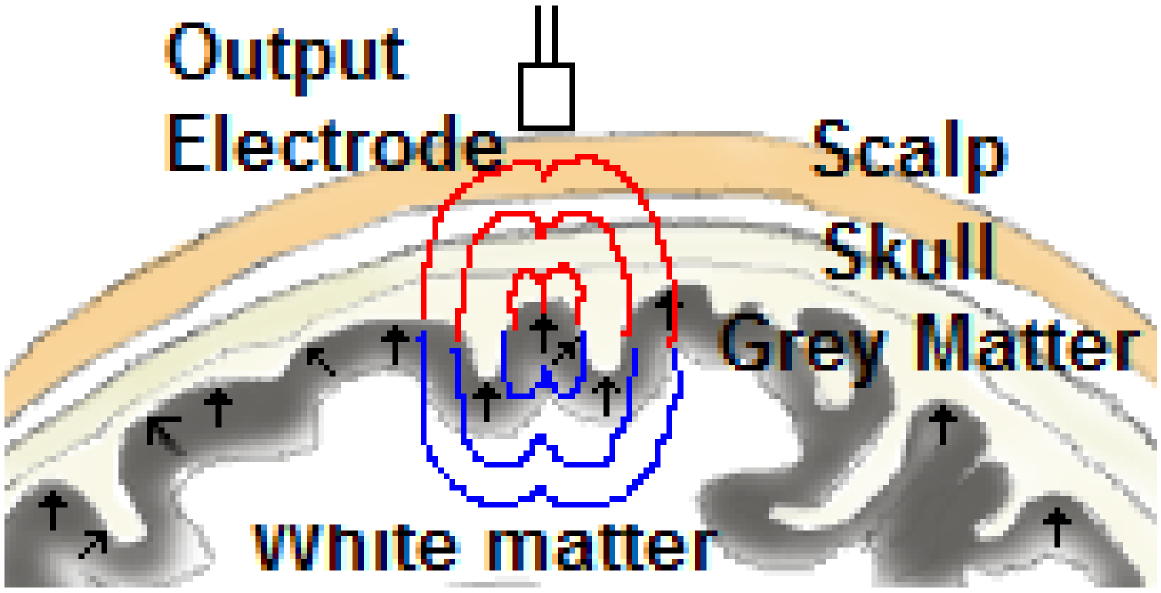


Figure 2.1: A simplified diagram of how EEG works. Ion flows generate electric fields in the neurons of the grey matter. The fields interact: diametrically opposed fields cancel each other out and electric fields with similar orientations amplify each other. If the fields (red and blue on diagram) become large enough they reach the electrode and are recorded as EEG. (Drawn by Jason Cummer).

2.2 Data Preparation

There are a few different ways to monitor the state of the mind, one of which is the EEG. This is the measurement of the electric fields on the scalp. These electric fields are the combination of the fields from the environment and the brain. Electrical fields from the brain are generated from the collective activity of excitable cellular membranes in the brain. The resultant signal that the EEG detects is one that is created by the synchronous firing of many cells. Axons that run parallel create electric fields that are additive and they are detectable on the scalp [26].

2.2.1 Artifacts

Artifacts are strange waveforms in the EEG, that do not have anything to do with the brain activity. External fields are generated from unshielded electronic equipment that humans have created. These fields can create artifacts as their potentials interact with the sensors of the EEG. Examples are the radio waves from any number of transmitters, the 60 hertz noise generated from the electrical transmission system, lights switches and motors.

Artifacts are also created from the human body. They can come from the heart (cardiac artifacts), muscles and eyes (ocular artifacts). Any time a muscle contracts it generates an electrical field. These fields are orders of magnitude larger then the electric activity detectable from the brain; their large size often dwarfs any detectable smaller fields from the brain. Electrode movement on the scalp can also cause errors in detecting the electric fields originating in the brain.

2.2.2 Emotiv EPOC

The Emotiv EPOC [31] is a commercial wireless electroencephalogram. It has fourteen electrodes which provide a total of 14 channels of data. The electric field of the scalp is sampled 128 times per second. The EPOC can function in a few different ways: cognitively, affectively, expressively, and gyroscopically.

One of the ways that researchers using EEG attempt to keep their experiments replicable is through the use of standardized electrode placement. One of the standards is the 10-20 [38] system, as shown in Figure 2.2. The location of the electrodes in this system are based on percentages of the distance from either the front and back of the scalp or the right and left side. The 10-20 locations for the Emotiv are af3, af4, f3, f4, f7, f8, fc5, fc6, t7, t8, cms, drl, p7, p8, o1, o2 [38]. These locations are easily seen in Figure 2.3

The EPOC has the ability to discern a few cognitive states. The states that come with Emotiv's software are long term excitement, short term excitement, meditation score, frustration score and boredom score. The EmoEngine analyzes those data collected while the user is mentally performing tasks on a cube that is displayed on a screen. The brain wave data are collected from all the channels. The user trains for one action at a time. The collected data are then analyzed with proprietary algorithms. Classifications are built for any of the thirteen possible actions a user can do with the cube. An example screen shot of the training cube is shown in Figure 2.4.

The actions are the six directions of movement, six types of rotation and the ability to make the block disappear. Once the user has trained the EmoEngine on these states, they can assign actions to the states. With these actions the user can interact with the computer via their thoughts.

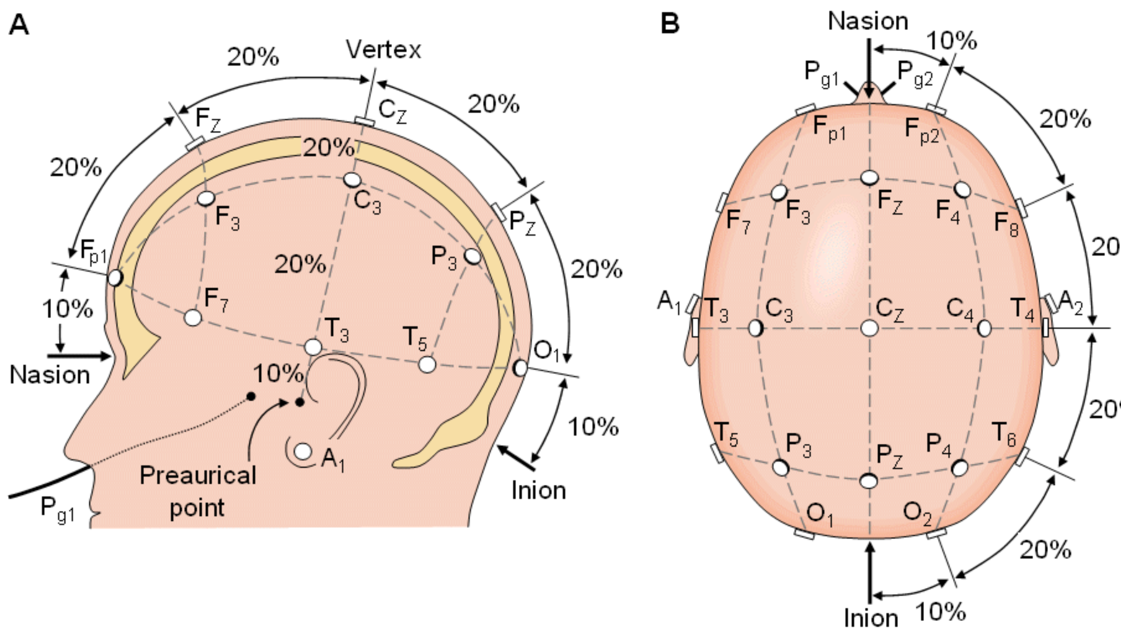


Figure 2.2: The international standard 10-20 system [19]

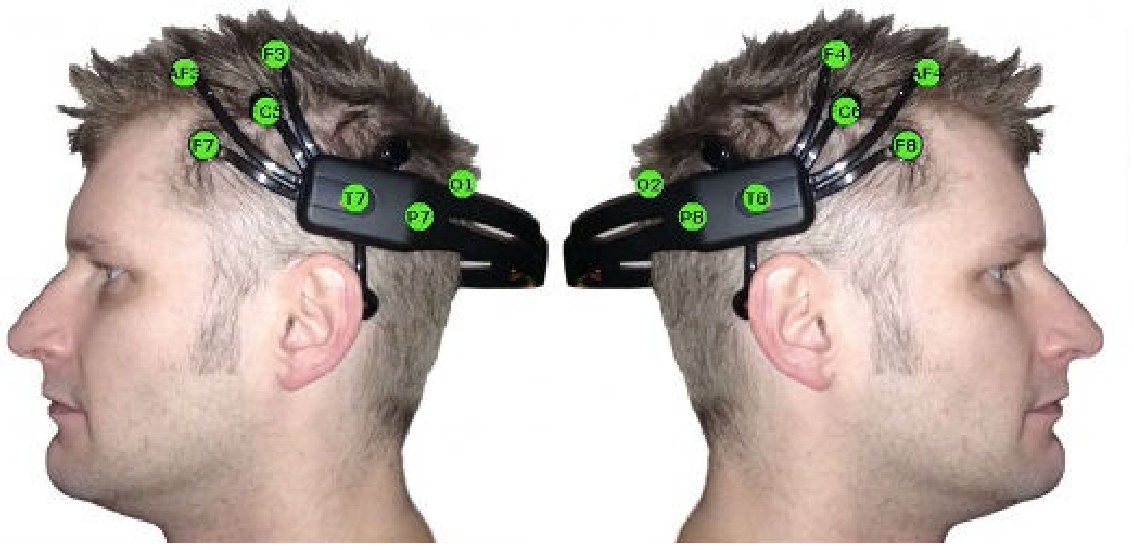
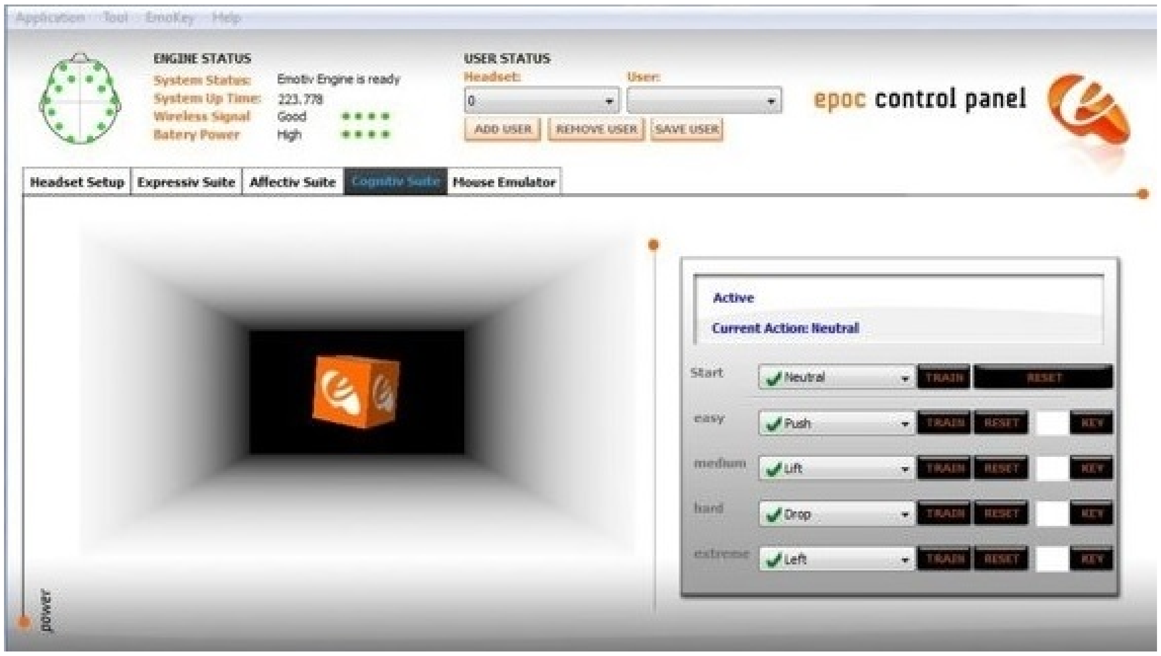


Figure 2.3: Positions of the electrodes on the Emotiv EPOC [5].

Affective states that can be detected with the EPOC are engagement, instantaneous excitement and long term excitement. A fast Fourier transform (FFT) can be used to find the component wave frequencies in the EEG. These wave frequencies have energy, also referred to as power. A group of frequencies is called a band. The specific powers of the frequency bands have been identified and correlated with



System Status: Emotiv Engine is ready
System Up Time: 223.778
Wireless Signal: Good
Batery Power: High
Headset: 0
User:

Active
Current Action: Neutral

Start: Neutral
easy: Push
medium: Lift
hard: Drop
extreme: Left

Figure 2.4: This image is the EMOTIV built in training system. It shows a few different ideas / cognitive states for training yourself to move the cube [30].

specific affective states. Many of the electrodes, especially those nearest to the face, receive large changes in voltage when the muscles of the face activate. These voltage changes can be classified to determine the expressive state of a user. The EPOC's two dimensional gyroscope allows for the position of the user's head to be used as input as well. Popular uses of the gyroscope are tracking head position and mouse control. The EPOC headset also has built in wireless communication with the host computer. This allows for more freedom when the user or subject is wearing the headset, as there are no wires to hinder them.

2.2.3 Artifact Removal: Independent Component Analysis

Artifact removal is not a focus of the exploratory study in this thesis. The Emotiv EPOC headset has some digital filters built in, but no additional filters were added. The reason for this choice is that the system used should be able to tolerate the noise of the world.

2.2.4 Pre-processing of Data

For a simple test of data mining, we used the Waikato environment for knowledge analysis (WEKA). This is a Java software package designed to help with the study and use of big data [50]. It is a collection of data mining algorithms that utilize machine learning. With WEKA and its built in neural networks, an attempt was made to find a relationship between the EEG data and the direction in which the mouse was moving. The mouse position data were used to find the cardinal and intercardinal directions of the movement.

To find these directions, the deltas for the x and the y of two mouse positions were found. The arctan function was used to get a measure of the angle of the vector that the deltas formed. The angle of the vector was then rounded to the nearest cardinal and intercardinal direction of the compass. The rounding was done based on the division of a circle into 22.5 degree segments. The two mouse position deltas that were chosen were based on the amount of time the mouse was moving. This time was related to the time between line writes in the data. The first point was taken when the first change in a mouse point was detected. If the mouse had not moved in 30 lines of data, then the mouse movement was considered to have stopped. The last point change recorded before the 30 lines was the second point used for finding the delta. An example of the number of 128ths of a second that occurred in the spaces between recorded mouse movements is shown in Appendix A.1. Thirty lines corresponds to about 234 ms. This assignment was based on an analysis of the mouse position data. The time it took me to move my mouse on to an object, for example.

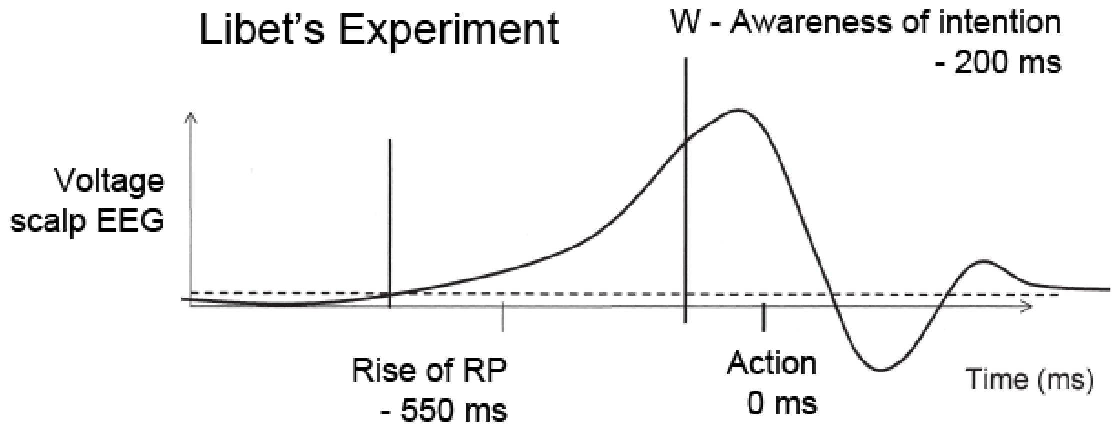


Figure 2.5: Libet's experiment showing that EEG can capture the time at which the user is generating an intent to act [1].

Date collected during the second before the mouse movement started was used for training the WEKA neural network. This approach was based on Benjamin Libet's work in the 1970s. Libet used EEGs for his experiments. His subjects wore EEG sensors and they were told to press a button or extend a finger. Libet also told them to watch a timer. When his subjects became aware that they wanted to press the button or extend a finger, they were instructed to note the time. This allowed the researchers to look at the EEG for a signal that would correlate with the button press. After analysis, Libet found that the participant reported the urge to act at 200 ms before the button press, with a margin of error of 50 ms, as shown in Figure 2.5. The EEG showed that there was a rising potential 500 ms before the button press. This means that 300 ms elapsed between the execution of the action and when the brain showed signs of initiation of the action. This means that after about a third of a second the user becomes aware of an action that they could execute [48]. Daniel Dennett's comment was "the action originally precipitated in some part of the brain, and off fly the signals to muscles, pausing en route to tell you, the conscious agent what is going on" [27]. Given that it is possible to discern from analysis of the EEG data that movements are coming, this should be a reasonable feature to look for in the data from the EPOC.

2.3 Biological Neural Networks

Information processing in animals is done with networks of neurons. In vertebrates, information processing is done in large networks of neurons called brains. Humans have advanced brains capable of simulating environments and planning for the long term future. With the help of the circuits contained within it, the brain is able to figure out what events will happen in the world. An example is the ability to catch a fly ball. The brain is able to simulate the ballistics of the ball and this allows one to move to intercept it.

The human brain is an approximately 1.5 kg mass of information processing and support cells. Here are some numbers that provide a picture of the complexity of the human brain. Genetic variation and life events in humans lead to variety in the following estimates. The human brain contains approximately 100 billion neurons. Neurons have about 1000 to 10 000 inputs per cell, leading to about 100 trillion synapses in the brain. These neurons exist in various levels of organization, somewhat related to our evolutionary history. One of the newer structures to evolve is the *neo cortex*. The neocortex is a continuous sheet of neurons 1.5 to 3 mm thick and approximately $2500cm^2$. One level of sub organization is known as a *minicolumn*, of which there are about 300 million in the human brain. These minicolumns contain about 80 to 120 neurons. These minicolumns are classification units; an example is an ocular dominance column. The minicolumns are classification units because they have one level of classification they do. They then pass on their information to the next column or to other cortical regions. The basic circuit of a minicolumn can be seen in Figure 2.6. One last interesting fact is that the brain creates new neurons which partake in learning new information. About 700 new neurons are added every day. Not many ANNs incorporate new neurons after their initial training is complete.

A feature of natural neural nets (NNN) is bottom up processing. This concept involves sensory neurons sending information into the nervous system and then that information is filtered and abstracted until it reaches the highest levels of abstraction the network is capable of. An interesting second feature is that NNN also engage in top down processing. This is when neurons that have the highest levels of abstractions send information back to the lower levels of the network. In NNN, this often leads the neurons to expect a specific type of input. The expectation and the resulting difference can be used to alter the bottom up filtering of information. This allows the network to be more flexible in the kinds of information and contexts it can process

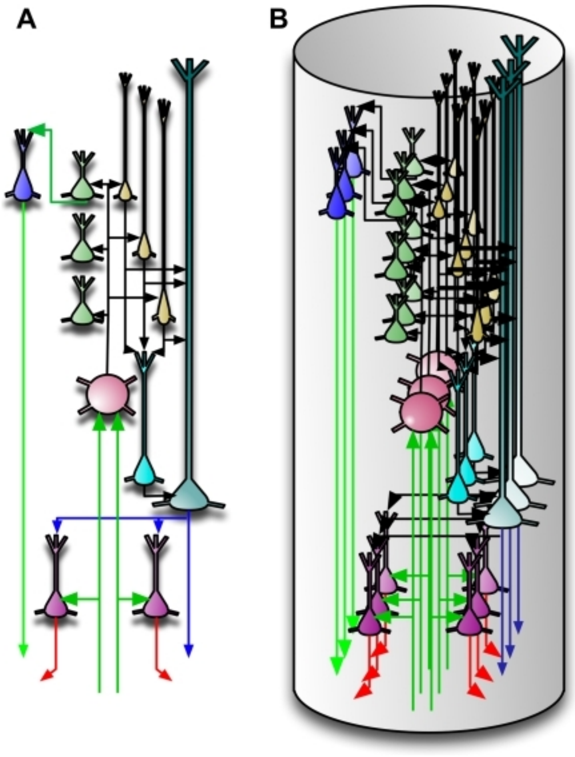


Figure 2.6: The basic wiring of the cortical column. (A) shows the simplified neurons and their connections in a cortical column. Green arrows represent connections to and from the thalamus. Internal connections are shown in black. Output to other cortical areas is shown in blue. The red connections are input and output for feedback connections. (B) is the same set of neurons as in (A) but it shows that there are more than one of these circuits in a cortical column. [32]

successfully. On the architectural level, one can see the inputs and the outputs from the minicolumns in Figure 2.7.

The brain is composed of many different higher level structures. The cerebellum is a largely repetitive structure located at the base of the brain. The cerebellum is primarily used for motor learning and fine control of motor actions. The brain stem consists of a few sub regions. At the top of the spinal cord is the *medulla oblongata*. The medulla oblongata contains many of the automatic functions to maintain homeostasis such as heart rate and breathing. Above the medulla oblongata is the *pons* which has white matter tracts to move information to and from the body. The pons also has some regulatory functions such as helping control the muscles of the face, eyes and throat, equilibrium, hearing, taste and facial sensations. The last region in

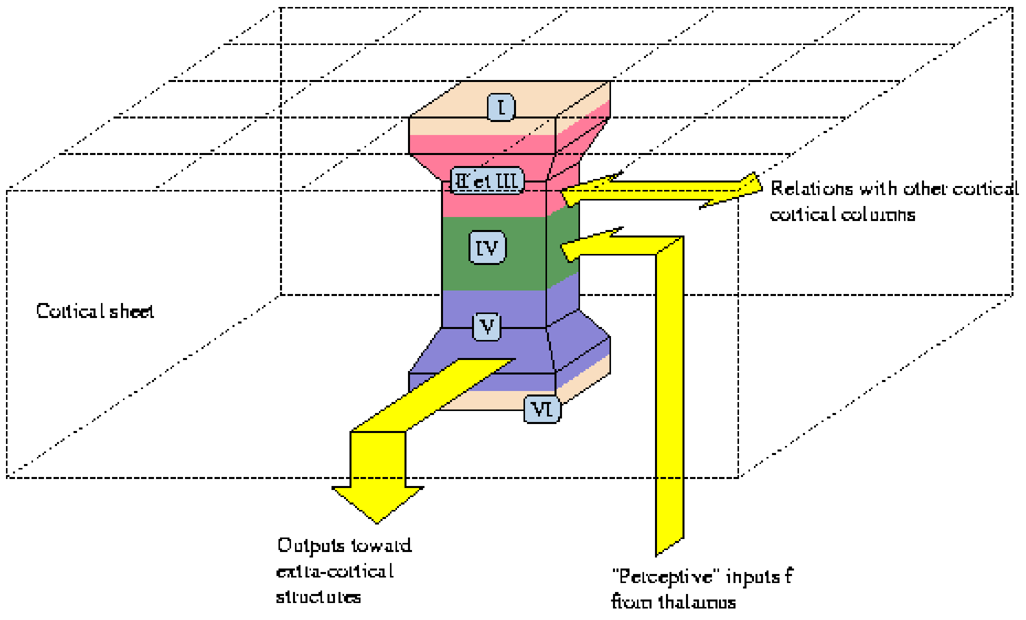

I
II et III
IV
V
VI
Relations with other cortical cortical columns
Cortical sheet
Outputs toward extra-cortical structures
"Perceptive" inputs f from thalamus

Figure 2.7: Cortical Column abstracted and showing connections. Bottom up input from the thalamus feeds into layer IV. Top down processing flows from the column from layer V. The lateral connections to other units are shown from layers II and III. These lateral connections can be viewed as both bottom up and top down processing. On the inside of the column there is also a mix of both types of processing as can be seen in Figure 2.6 [20]

the brain stem is the midbrain which oversees vision, hearing, motor control, sleep patterns and temperature regulation [44].

Physically on top of the brain is the *cerebral cortex*. The cerebral cortex is divided into four main lobes: the *occipital*, *parietal*, *temporal* and *frontal*. These regions play a role in almost everything you do as a human being. The occipital lobe takes in input from the eyes; it processes shape, colour and motion, and passes this on to secondary processing areas. The parietal lobes take information, primarily from the skin, and combine it with information from the ears and eyes in secondary processing areas. They give us our sense of the body and space, helping us deal with many spatial problems and movement. The temporal lobes take information from the ears and secondary information from the eyes, and it stores patterns. It is involved in semantic and episodic memory. The frontal lobe is the primary output for motor activity, including speech. It receives input about smell, helps to regulate attention, personality and mood and it is involved in long term planning.

The network that these regions create is known as the *connectome*. The connectome could be used as a template for creating ANNs with similar functions as the brain. Later we will look at a new neural net architecture, the NeuCube. The NeuCube has utilized the connectome to create some of its structure [42].

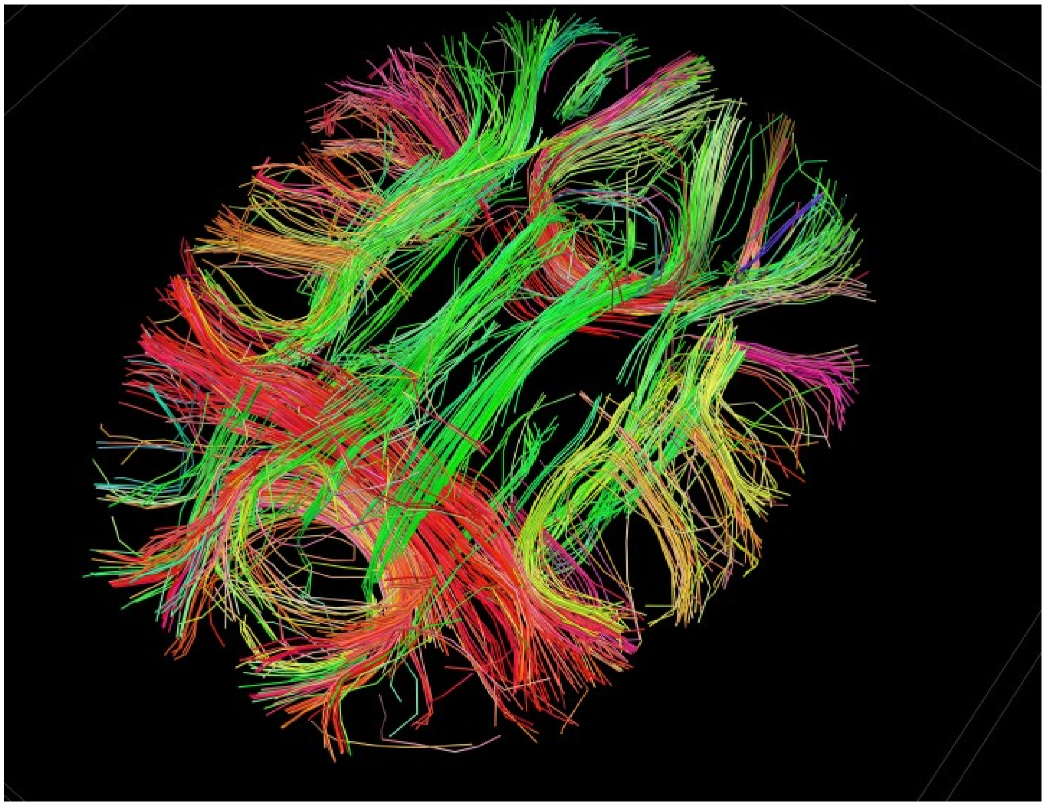


Figure 2.8: A partial representation of the human connectome [8] It displays how the long range connections from cortical columns are arranged across the brain, thus showing how the overall network of the brain is constructed. The different colours of the lines allow one to trace the different connections in the connectome.

The brain is made up of two types of cells. *Glia* and neurons are the main cells of the brain. Neurons are the main information processing units of the brain, with glia being an important partner cell to neurons. Glia cells help to maintain the environment for neurons and help regulate synaptic connections. Neurons have some specializations that allow them to process information and pass the new information to any neurons with which they share a connection.

Specialized structures known as *axons* and *dendrites* exist on neurons. Axons are extensions from the soma (cell body).Their primary purpose is to carry an action potential to the axon terminals. Action potentials will be discussed in more detail

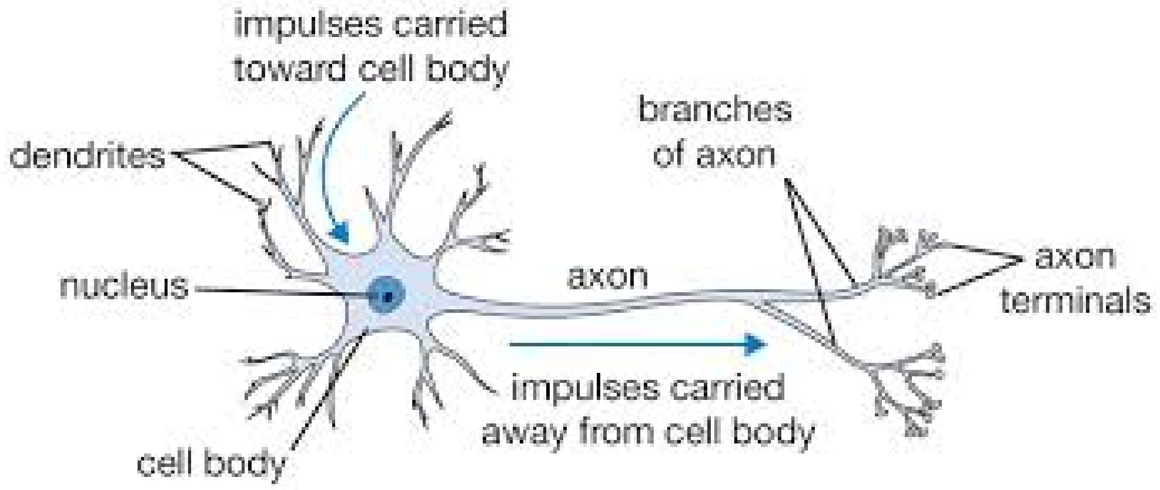

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Figure 2.9: A basic neuron, with its various constituent components [13].

below. The axons may or may not be myelinated by special glia. If they are myelinated they have a higher conduction speed. This faster conduction speed influences the networks dynamics. Higher conduction speed is due to the ability of the myelin to transmit voltage via saltatory conduction. The action potential is thus conducted past the myelinated section of the axon to a section that is not myelinated. This non myelinated section is called a node of Ranvier. At nodes of Ranvier, the action potential is regenerated via the ion channels present in the cell membrane. If the axons are entirely non myelinated then the action potential is transmitted entirely with ion channels. Dendrites are postsynaptic terminals. They work to sum the incoming information of the connected synapses.

Synapses occur between two neurons and are the location at which information is passed from one neuron's axon to its neighbour's dendrites. The synapses work either with a chemical or electrical connection (gap junction). Chemical connections are the most common in vertebrate brains. Chemical synapses work by the following mechanism: when the action potential reaches the location of a synapse, it triggers the release of neurotransmitter chemicals. The neurotransmitter diffuses across the synaptic cleft and binds to specific receptors in the postsynaptic cell. The receptors on the postsynaptic cell allow for the flow of ions which alters the voltage of their cell membranes. Electrical synapses function by allowing ions and other small cytoplasmic molecules to diffuse or flow across them. This is what affects the postsynaptic cell's membrane voltage.

2.3.1 The Action Potential

Action potentials occur when the cell's membrane voltage increases above a threshold regulated by ion channels in the cell's membrane. The voltage can increase due to an influx of Na^+ ions. When the voltage increases to +30 mV the flow of Na^+ stops. The K^+ ion channels open and potassium flows out of the cell, rapidly dropping the voltage back down to below its resting potential. As all the ion channels reset, the voltage returns to -70 mv. An action potential is shown in Figure 2.10.

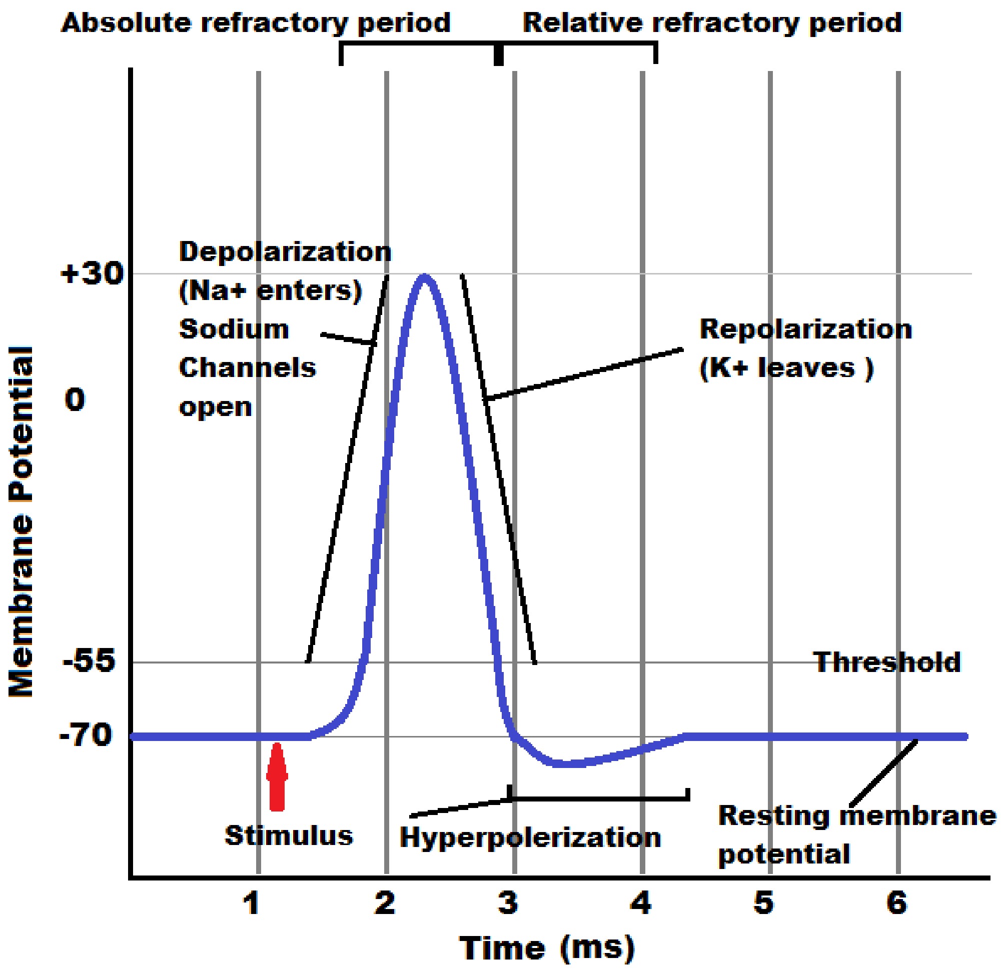


Figure 2.10: A graph of the membrane's voltage as the action potential flows through a region of the axon. (Drawn by Jason Cummer).

2.4 Machine Learning

Observing the world with sensors creates vast amounts of information. Saving all these data creates huge amounts of what is called big data. Humans can learn from these data but often times there is too much to look through in a useful time period. This has given rise to a field of study called machine learning that uses algorithms and computers on big data. Machine learning deals with how algorithms can learn about the structure of data. The algorithms can be used to learn how to extract facts from the data that are usable for a further process.

The most common of these algorithms are decision trees (DT), association, naive Bayes, clustering, support vector machines (SVM), logistic regression, multilayered perceptrons (MLP) and ANN. Amongst these algorithms, the few we are going to consider are logistic regression, MLP and ANN.

2.4.1 Supervised Learning and Unsupervised Learning

Supervised learning is like doing math problems and checking your answers. You go through the questions and then look at the back of the book. If your answer is the same as the book's answer, you move to the next question. If your answer is different, you have to go back and try to adjust your approach to the question. This is similar to how supervised learning works. There is a set of data, targets and results or classes to which the algorithm has to match the data. Over the time that the algorithm is running, it is trying alter its internal representation of the data to allow for the desired output. Once the desired output is predicted or an acceptable level of error has been achieved, the algorithm terminates. The saved model can then be used for predictions of new data from the same source.

In unsupervised learning there are no answers given for the data presented. The algorithm has to learn the structure of the data by itself.

This thesis will focus on supervised learning. The reason for supervised leaning is that the desired outputs were recorded. This allows the collected data to be used with the recorded outputs for training. Zero-one loss is used to calculate the number of errors in the prediction function against the data [15].

$$\ell_{0,1} = \sum_{i=0}^{|\mathcal{D}|} I_{f(x^{(i)}) \neq y^{(i)}}$$

$\ell_{\theta,\mathcal{D}}$ is the empirical loss of the prediction function f, parameterized by θ on dataset $\mathcal{D}$. θ is the set of all parameters for a the model. $\mathcal{D}$ is the number of dimensions. I_x is the indicator function. The definition of I_x is 1 if x is true and 0 otherwise. I_f was defined for the tutorial, as:

$$f(x) = argmax_k P(y = k|x, \theta)$$

Zero-one loss is not differentiable and uses a lot of computational power if you have thousands to millions of parameters [3]. This makes it less practical than other functions for the purpose of machine learning. negative log-likelihood loss (NLL) is a function that is similar to the zero-one loss but is differentiable. This allows it to handle thousands to millions of parameters [3]. NLL is used instead of zero-one loss gradient decent for the pre-training of the restricted Boltzmann machines (RBM). This is for training larger problems in a reasonable amount of time. The function for NLL used by Theano is as follows:

$$NLL(\theta, \mathcal{D}) = -\sum_{i=0}^{|\mathcal{D}|} logP(Y = y^{(i)}|x^{(i)}, \theta)$$

This equation finds the log of the probability of class Y, when the input of y^i is correlated with x^i.

The error of the NLL function is used in conjunction with L1 and L2 regularization to provide the gradient for gradient descent [3]. The regularization penalizes certain parameters if they are too large. This reduces nonlinearity in the model. The gradient of the error surface is calculated from the input data. The function that is supposed to represent the data is often created as a linear equation. The gradient of the function can be found by taking the derivative of the error surface.

Small alterations in the equation are introduced and a new gradient is calculated. The continued alterations of the equation hopefully allow one to converge on the global minimum. More often, several minima, maxima and saddle points are found. The equation is at its optimum for the prediction of a class when the gradient is 0 on the error surface [22].

Stochastic gradient descent (SGD) [3] is a variant of gradient descent. If the set of data are very large, iteration over all the training data takes a long time. The first step is to shuffle the training examples. Then, the first example in the randomly shuffled input examples is used to calculate a partial derivative. The partial derivative allows the algorithm to move down a gradient on the error surface of the function. The vector for the optimization function is slightly altered. This is done by adding the coefficients to the gradient, multiplied by the learning rate. The next point of the shuffled input examples is then used to repeat the process of finding the partial derivative and altering the optimization function. The advantage of the partial derivative is to speed up the gradient decent algorithm. Convergence on the minima may only take a single loop through the data, but a dozen loops is common in the literature review undertaken. It might not settle in the minima, but it is usually close enough for a satisfactory classification rate [22].

Minibatch stochastic gradient descent is a modification of SGD. The algorithm uses only a subset of the data to find the gradient. It can be some number of the previously processed data points. The error is calculated the same way as SGD but only the subset of data is used. The greater the number of data the smoother the plotted error will look. Minibatch can be faster then SGD. If the algorithm hasn't made any progress on learning in the latest batch, it can terminate. Minibatch works better with vectorization and it allows for better parallelization. The implementation that we use, Theano, is designed to use parallel processing. This makes minibatch a good choice for Theano [22].

2.5 WEKA

WEKA [50] is an open source data mining software package. It was created at the University of Waikato and named Waikato environment for knowledge analysis, hence WEKA. It contains modules for data pre-processing, classification, regression, clustering, association rules and visualization. It includes many of the commonly used machine learning algorithms such as multilayered neural networks, which was used earlier for testing the EEG data for directional information.

2.6 Artificial Neural Networks (ANN)

In "Introduction to Neural Networks in C#", written by Jeff Heaton [34], a simple multilayered neural net is described. The basic points from his book on simple nets follow. ANNs can have different numbers of hidden layers:

- No hidden layers - these are only capable of representing linear separable functions of decisions.
- One - these can approximate any function that contains a continuous mapping from one finite space to another.
- Two - these can represent an arbitrary decision boundary to a arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
- One hidden layer is practical for most any function.
- There is no theoretical reason to have more than two hidden layers.

Hidden layers are layers that are not directly observable. The input and output layers can often be seen, but hidden layers are supposed to be internal to the ANN. ANNs are also characterized by the number of neurons in the hidden layers. Too few neurons in a hidden layer leads to underfitting. This is not good for a complicated data set. Too many neurons may result in overfitting, where the ANN learns the noise of the model. With limited information, there may not be enough to train all the hidden layer's neurons. With sufficient information and too many neurons, an ANN may require too much training time.

Some ANN rules of thumb follow: The number of neurons in the hidden layer should be between the number of output and input neurons. The number of hidden neurons should be 2/3 the size of the input plus the number of output. The number of hidden neurons should be less than twice the size of the input layer. You will have to ultimately try various numbers of hidden neurons - Ray Kurzweil says evolutionary algorithms [45]. With the addition of an evolutionary algorithm, you can alter many of the parameters of the neural net in a more automated fashion. Running evolutionary algorithms can also give you specialized neural nets that may be better for some applications but not others.

2.7 Liquid State Machine (LSM)

A liquid state machine is a type of neural network known as a spiking neural network. An LSM also fits into the category of a *reservoir computing system* [49]. Reservoir computing uses a system that is dynamic and thus, information about timing in the system matters. In an LSM, the system is a number of nodes or neurons that pass information along to their downstream connections. In the case of this neural network, the downstream neurons reconnect causing recurrent connections. These recurrent connections allow information that has been presented to the network to flow back to the origin of the data and through other loops. LSMs can be created with structures, but most are created as a set of randomly connected neurons, see Figure 2.11.

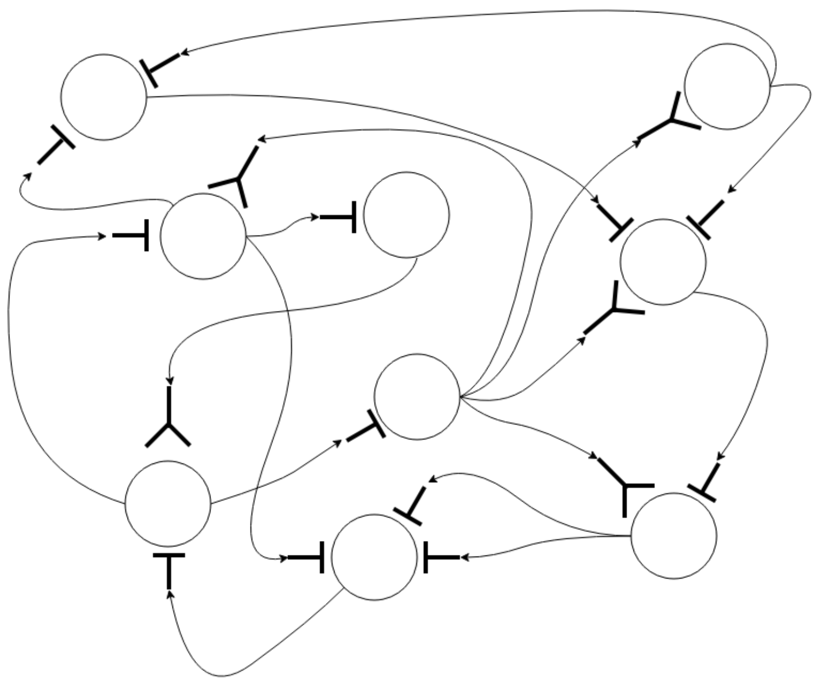


Figure 2.11: A simple example of a liquid state machine with random inhibitory and excitatory circuits. Circles are neurons. Y shapes represent excititory synapses. T shapes represent inhibitory neurons. (Drawn by Jason Cummer).

The separation property of the state vectors of an LSM is a measure of the Euclidean distance of the states of the network with different samples. This can be used to determine whether the network is able to create different states for the different input fed into it. Two states can be seen with this technique: over-stratification being that there is too little power in the input to create continuous firing patterns and pathological synchrony, which is excessive firing of too many neurons [51].

2.8 Parallel Neural Circuit Simulator (PCSIM)

PCSIM is a tool for simulating neural networks [56]. It is a software library written in C++ for speed, and has a Python interface for ease of use. The Python interface as abstracted version of the C++ functions and classes to allow the user to easily use them. It is built to use objects called network elements, which are base classes for higher order objects. Some of the higher order objects are neurons and synapses, see Figure 2.12. There is code to build and simulate a neural network based on these network elements. PCSIM also allows for the network to be run on many different machines if available. The simulation can be advance based on a number of user defined time steps or a user defined amount of time.

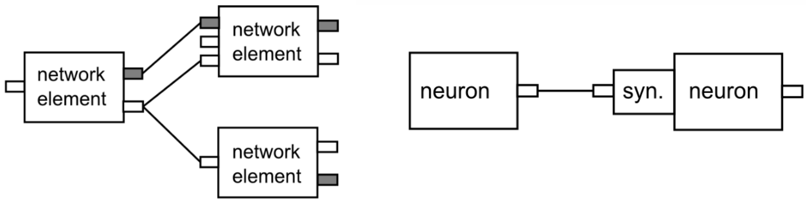


Figure 2.12: On the left is an image of the generic network element. They are the parent class to the neuron and synapse on the right [56].

It can take many seconds to run an advanced cycle of the simulation in the LSM, but this is where the liquid state machine implemented with PCSIM excels. It has been designed to be run in parallel. It can be run on the same machine in a single thread or multiple threads, or it can be run in many machines that are networked [56]. It has many different ways to create a liquid. One of them is the cubic volume. For example, the LSM used for this thesis is filled with 20 x 20 x 10 (4000) neurons in a cube pattern. The input for the LSM can either be normalized analog data or spike trains. Output is in the form of floats, being the recorded analog values of the simulation's neural membranes that were sampled at random from the liquid.

The utilization of an LSM to incorporate and compress time series data into a smaller output for the DBN was investigated for the example system.

2.9 Deep Belief Network (DBN)

DBNs are constructed as an MLP. They utilize RBMs as part of the layers in the MPL. An example of an RBM is shown in Figure 2.13.

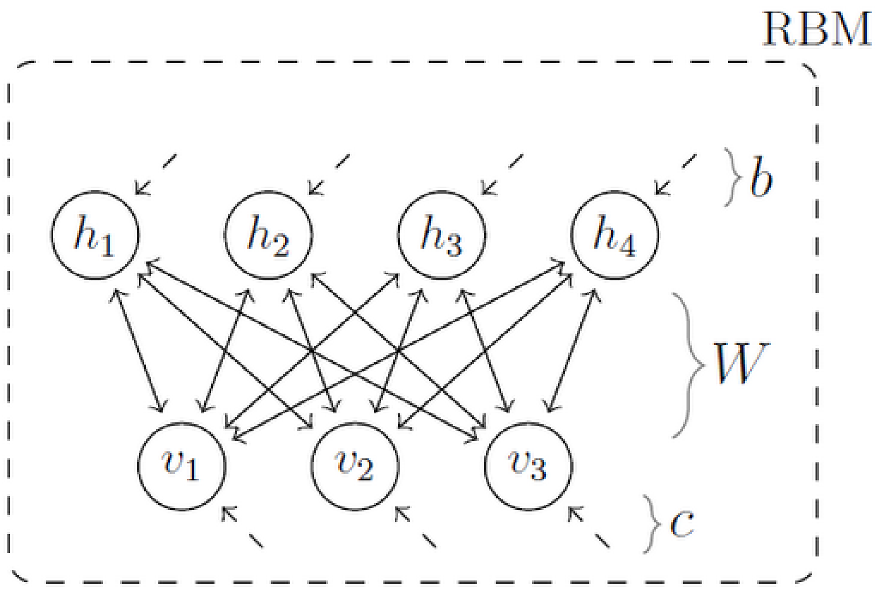


Figure 2.13: A restricted Boltzmann machine. There are two layers of neurons. One layer contains neurons with the set of h_i. This is also known as the hidden layer. The second layer, the visible layer, contains the set of neurons v_i. C is the biases for the visible layer. B is the biases for the hidden layer. The symmetric weights between the layers are indicated by W [64].

The RBM is created from two layers of the MLP (see Figure 2.14). The weights of the MLP's layers and the weights in the RBM are shared. Thus, training the RBMs will alter the weights that are shared with the MLP.

The training is done in a greedy layer-wise manner for a fixed number of epochs for each layer. The DBN training then moves onto the final stage, to train the MLP with stochastic gradient descent. If the training is successful, the DBN will be able to classify data into the target classes.

During gradient descent, the algorithm periodically check, with the validation set to test the model on the real objective function. The SGD saves the first model and its score, which is a measure of how well the model fits the training set. Upon checking again, it will save over the previously saved score and model with models that have better scores. The algorithm keeps making checks to see if it can find a better model. There is a built in patience score. Every time there is no significant

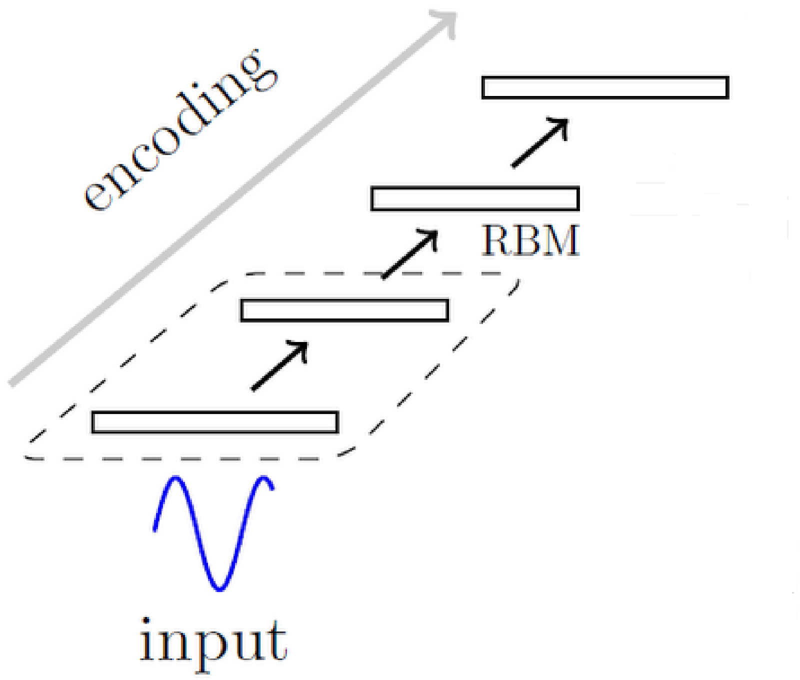


Figure 2.14: An MLP that is built with RBMs. Each set of layers would be an RBM [64].

change in the score, 1 is subtracted from the patience score. When the patience reaches zero the searching for a better model stops. Once we have our best model based on the training data, we evaluate the model with the test set. This evaluation against the test set allows us to determine if the model is still generalized enough to classify data it has not seen before [3].

The training of DBNs might not be real time, but the implementation of the filters that they create is utilizable in real time [64]. Results from the DBN are given in the form of validation error.

Theano [23] is used to dramatically decrease the training time for the DBN. Theano uses a computer's graphics processing unit (GPU) to achieve parallelization of data processing. It also contains optimizations that allow it to compile functions directly into GPU code.

2.10 Brain-Computer Interfaces

Classification of EEG signals is not perfect. If a user is trying to create a desired outcome and the machine will not detect this intent, it can be extremely frustrating and disheartening. On the other hand, if the user is doing nothing and the system is acting erroneously, it can be annoying and interruptive. Current methods to classify EEG such as MPL, SVM and DBN do not yield reliable classifications for the near ideal user interface [42].

The method presented in this thesis borrows from natural neural networks (NNN). In NNNs, recurrent connections allow for a second type of memory in the network. Put another way, there is information in the state of the network itself. Typically in ANNs, there would only be information in the weights of the connections between nodes. The information in the weights is from previous learnt input, but not current input. This knowledge of NNNs is used to attempt a better classification rate by adding a processing step into the classification.

For humans to further merge with their machines, methods that allow for fast, accurate interaction must be developed. Ideally, the system would integrate seamlessly with the user's thoughts. It is essential to have less than 250 ms for feedback to the brain in order for the brain to incorporate the information into its networks [54]. Longer than this time and the feedback loop from the BCI to the brain is too large. The larger the delay in the feedback, the more difficult learning the interface becomes.

2.11 Summary

In this chapter we started by covering brain sensing technology that ranged from intimate to topical. Next we covered NNNs from the basic units of ion channels, up to the overall network that makes up the brain and the connectome [8]. We finished the chapter with machine learning algorithms and how they can combine to form a few types of ANNs known as DBNs and LSMs. In the next chapter, we will cover the basics of the proposed architecture this thesis takes to classify user intent and demonstrate a concrete implementation of a system as a case study.

Chapter 3

Proposed Architecture and Case Study Design

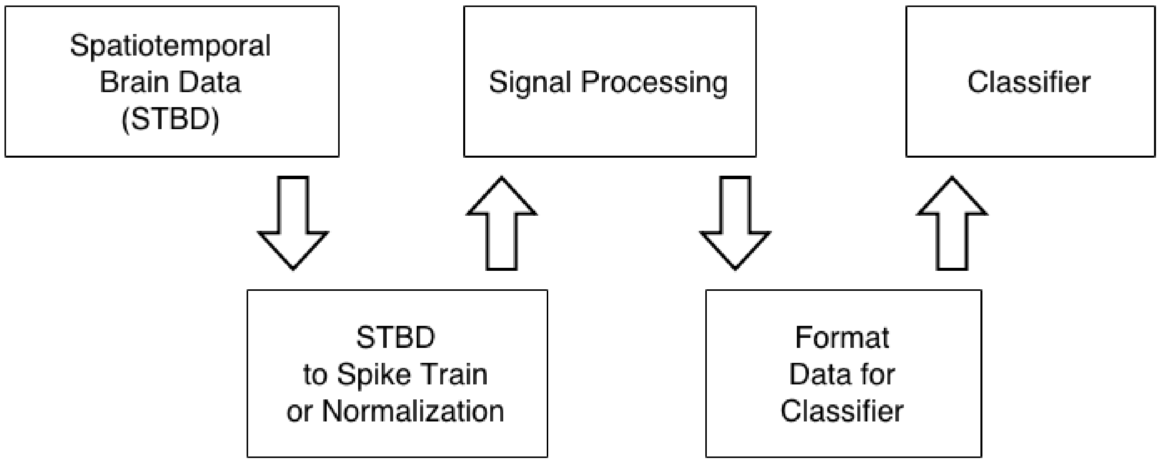

Spatiotemporal Brain Data (STBD)
Signal Processing
Classifier
STBD to Spike Train or Normalization
Format Data for Classifier

Figure 3.1: General architecture of a brain data classifier (Drawn by Jason Cummer)

The general framework for building a brain-computer interface (BCI) is shown in Figure 3.1, minus the final stage of an environmental effector. In general, a BCI will use some electroencephalography (EEG) or other device to capture the information about the user's brain. Other good example devices that could be used are microelectrode arrays, or an optogenetic interface. The brain data will then be formatted for signal processing. There might be a few different layers of signal processing depending on what is called for in your design. Eventually the data will be classified and a system interface will inform the environmental effector based on the classification results.

In order to assist others researching BCIs, one of the contributions of this thesis is the proposed modularization of the system, shown in Figure 3.1. To the best of our knowledge, no current framework exists. Though the framework we explore in our case study would take more work to completely generalize to be "plug-and-play", the overall breakdown remains intact. The temptation to perform more fine-grained integration into a monolithic system is strong in this case. The reason is that it could decrease the processing time, making this more of a real time system. However, other optimizations are possible once the efficacy has been established, such as making more sustainable software systems. The specific case study in this chapter overviews the behaviour of a liquid state machine (LSM) coupled with a deep belief network (DBN) in an exploratory experiment. In future work it would be possible to replace the DBN with a different classification component, such as an Echo State Network, and the subsequent testing could be done with a Support Vector Machine.

The case study to demonstrate the proposed architecture in this thesis uses the abilities of an LSM to help analyze EEG data. The EEG data were recorded while the user typed at a computer keyboard and used a mouse. Key presses, mouse movements and mouse clicks are the events to be associated with brain states from the EEG data. LSMs have the ability to encode the incoming data from the EEG into a spatiotemporal pattern in the liquid. A sample is taken from the network at the time of a keyboard or mouse event. The set of events and their corresponding samples is then sent to a classifier. The classifier for this case study is the DBN. Let us explore this system in increasing detail in this and the following chapter.

3.1 Overview of the Exploratory System

The system, in a more complete sense, is a commercial EEG (the Emotiv EPOC), a series of intermediate processes, the LSM and the DBN (See Figure 3.2).

The LSM is a dynamic system. This means that the system's components interact with one another and their locations or states depend on the time of the interactions in the system. For computer science, this falls into a set of computing known as *reservoir computing* [49]. With reservoir computing there are some forms of nodes that communicate with each other and interact in potentially non-linear ways. That is, they can suddenly shift the state of the system. When applied to the way that computers emulate natural neural networks (NNN), a common result is a spiking neural network (SNN). In the same way that an animal's neural network will send

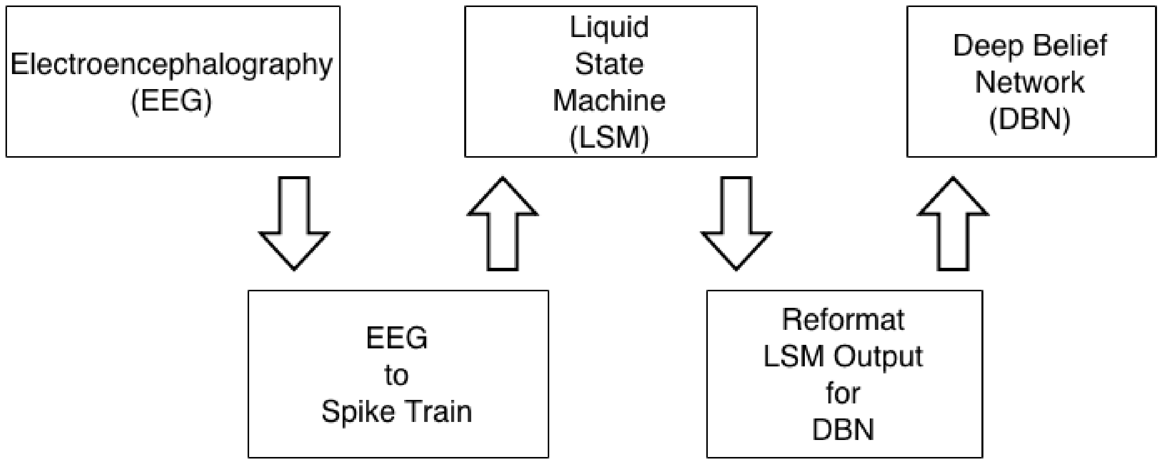

Electroencephalography (EEG)
Liquid State Machine (LSM)
Deep Belief Network (DBN)
EEG to Spike Train
Reformat LSM Output for DBN

Figure 3.2: Overview of the exploratory system (Drawn by Jason Cummer)

signals from neuron to neuron, so does an SNN. This is often a result of trying to optimize the way that computers use their internal processing resources. What happens, though, is that the network of neurons, or nodes in a computer system, is created such that they have loops of connections. Information, in the form of spikes, flows from one neuron to many neurons and loops back to its starting location, forming a highly dynamic form of memory. This highly dynamic memory has the property that information is held in the system from an input that came at an earlier time. With many of these memory loops of various sizes, information from the past is retrained by the system. As the information from different previous time points flows and interacts, new patterns emerge in the system. These patterns are used in the next stage of processing.

With the LSM, the current patterns of information in the system are of interest. In Figure 3.3, ripples from various drops of water create a specific pattern on the water's surface. There would be many ways to sample the water's surface, such as very high resolution ultrasound, a camera measuring the intensity of light at a given location or some system of mechanical floats on its surface. What is important is that a sample is taken from a randomly selected set of locations on the surface. With this set of sample measurements of the liquid's surface, associations can be made of what occurred to create this state. With the water ripple examples, you can see where the drops came from by the patterns they formed. Those patterns could be associated with some outside phenomenon. If there were enough examples, confidence in the association would increase, and it could be said that the outside phenomena x creates the state y.

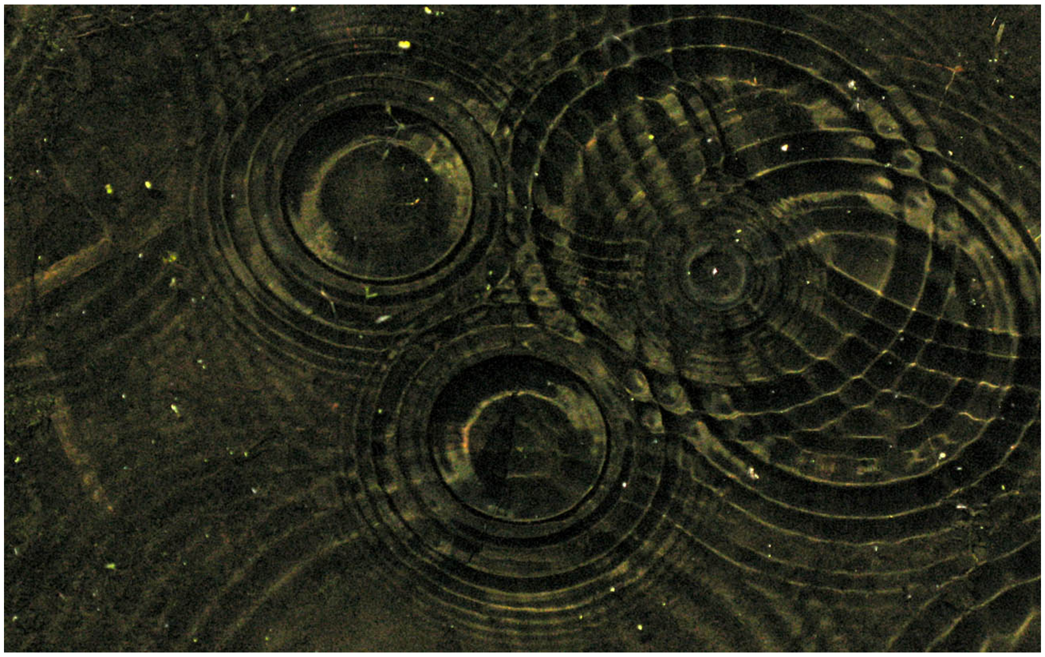


Figure 3.3: The surface of a liquid, water, after some drops of water have been added over time [11].

In the brain and in SNNs, these ripples are spikes flowing through the neurons of the system. For the LSM, there is a set of neurons that sample the state of the neurons in the liquid.

For the exploratory study in this thesis, it is important to note that the state of the network is created from a brain state that had intention. This means, as in the water example, when an outside phenomenon created a particular state, it can be associated with the phenomenon. In the case of the LSM, with inputs from the brain, the phenomenon that is being associated has a state from the intention of the user. Because the state and the intention are linked, in the LSM's case, the correlation tells the overall system that if the state was detected, the intention was desired.

To find the relationship between the state and user intent, DBNs were used. They are multilayered neural networks. They differ from deep neural networks in the way they are trained and the way they are set up. DBNs use a method of modeling the input, in our case the states of the LSM, that is learned by the DBN, layer by layer. There are two stages for training such a network. One is to have the input data modeled internally by restricted Boltzmann machines constructed of the layers of the DBN. The second is fine tuning the model's last layer to recognize the classes of the data, based on the data that have moved up through the layers of the DBN. Once this has been done, the DBN should be able to predict the final classes. In our case, the intentions of the users are recorded from electric fields created from the brain, by electrodes.

This thesis proposes a pipelined architecture for classifying user intent from EEG data. The LSM and DBN system demonstrate a concrete manifestation of this architecture. This exploratory pipelined system includes training of networks to filter the data they receive. For example, the DBN creates a set of filters that, once trained, are fast at processing the input. The LSM does not require any training so its information flow rate will remain the same. An LSM can be created in such a way that the neurons are processed at the same time at different locations. This parallelization can increase the speed of the overall system to allow it to process the input as it comes from the EEG. If the system were set up with the LSM running in parallel and existing DBN filters, it would run in real time. At this point, it might be the case that if a user was wearing the EEG headset with the system processing the data from it, the system could produce an output matching the user's intent. With one more small program the intent could be translated into computer commands for key presses and mouse output, making a system with which users could type with their thoughts.

3.1.1 Emotiv EPOC

The Emotiv EPOC, a commercial EEG headset, was used for this study. As expected in a commercial system, the resolution is less than that of systems intended for scientific research. Temporal resolution of research systems can be as high as 20 000 Hz but as low as 250 Hz [21]. The EPOC's sampling rate is 128 Hz. The spatial resolution of the EPOC is not as good as those of research systems, either, with only 14 electrodes. High density research models can have 256 electrodes. Overall, this EEG is not high quality, however, it resembles what an early functional system or prototype of a BCI might use.

3.1.2 Input Recording Software

A program was developed in C#, to receive the recorded values from the EPOC headset. The types of values were the power of the electric fields on the surface of the scalp and the user head movements, sensed with the built in gyroscopes of the EPOC headset. The recorded power values were saved to a file along with some of the user's actions. Key presses and the mouse movements were recorded with *User32.dll* on a Windows 7 computer. User32 is explained in the next section. All these data were saved to binary files for later use.

3.1.3 User32

Windows USER is the general name given to the section of the Windows operating system that receives, uses and creates interfaces for the user. The implementation of this is in the dynamic link library, User32.dll. Sixty-four bit versions of Windows have the SysWOW64 directory and this is where a modified version of the User32.dll exists for these systems [57].

Platform invoke is a feature that allows managed code to call unmanaged or native code. With platform invoke, a program can hook the functions in User32.dll [53]. Using this, the program can collect the following data from the mouse: right and left mouse click, up and down on the mouse wheel and the position of the mouse on the screen. From the keyboard, the data collected were all the letters, numbers and other keys from the keyboard. The shift keys, capital letter-lock key, tab, alt, left Windows and numlock key states are collected in a slightly different manner. The reasoning behind this was to capture the moment the key was pressed and possibly held down for interaction with other keys. For example, if the user wants to capitalize *I* in their text, the user would hold down the shift key, press the *i* key and then release it. These might all be detectable depending on the overlap of the timing. On the other hand, they may represent entirely different classes.

The recording of the control key was missing in the original set of data that was collected. The code necessary for recording the control key was added into the input recording software, but no new data have been collected with the control key recording. The absence of this key is of no concern to the exploratory study. If it is possible to train the ANN to recognize the other recorded classes it would then become useful to get more data with the control key and train a recognizer for the key.

3.1.4 Saving Data and File Formats

The initial recordings of the data were saved to custom binary files in the hope of saving hard drive space. In the end, hard drive space was not an issue for this study, but could be useful in some situations. The binary files were extracted and saved as text files of larger size. The purpose for the extraction was conversion of mouse position data to mouse movement direction data.

3.2 Spike Train Generator

Initially the use of PCSIM, and its implementation of a LSM seemed to have no way to get analog data into the LSM. I had read that it was possible with the correct combination of neurons, synapses and inputs. Later it was revealed that it was possible to do so with a StaticAnalogSynapse. The code to read in analog data is as follows:

```
net.connect(inp_nrns, nrn,
                StaticAnalogSynapse(delay = 1e-3,W = 1e-10))
```

The arrival of this information was after the completion of a program to take the EEG data and create spike trains as LSM inputs. The spike train generation is thoroughly described in Chapter 4 Exploratory Experiment and Results.

3.3 PCSIM

In Section 2.8 PCSIM was introduced and some very basics were covered. The LSM implementation is done with the network elements as neurons. It has been created as a single threaded network, rather then a distributed network. On a lower C++ level the single or multithreaded network is controlled by a simulation engine. To run the network one of two functions is called. There is a network advance and an network simulate function. “The simulation engine integrates all the network elements (typically neurons and synapses) and advances the simulation to the next time step, and uses its communication system to handle the routing and delivery of discrete and analog messages (i.e. spikes and e.g. firing rates or membrane voltages) between the connected network elements.” [56]. In this way PCSIM simulates its neural circuit.

If the neural circuit is run as a multithreaded distributed network it can run in other networked locations on many machines. The way that PCSIM implements the LSM in parallel, speeds up the simulation of the network.Some of the speed comes from optimized C++ code, allowing the network to operate efficiently. The parallelizability of the networks created with PCSIM is also a way that the networks can increase their processing speed. For this exploratory system we did not utilize this ability to run the network in parallel.

The code that I have utilized for the LSM has a cubic volume filled with 20 x 20 x 10 (4000) neurons. The input for the LSM is fourteen channels of EEG. The EEG collected data at a sampling rate of 128 Hz. With this sampling rate there is 128 sample from each of the 14 channels and a state for the mouse and all the keys every second. The encephalograms were transformed into spike trains and stored as 14 channels in a file for the LSM to use. There is also a set of data in the file that is the direction of the mouse at the time the EEG was recorded. The spike trains are fed into the LSM. The LSM is run in a way that every 128th of a second is simulated. One 128th of a second converted into a decimal is 0.007 812 5 s, or 78.125 ms. The advance step of the LSM is 0.1 ms for this setup.What this 1/128th of a second means in practical terms is that for every 8th set of 78 simulation steps, an extra net.advance(1) has to be called. The reason is that there is a remainder of 0.000 012 5 seconds. If a step in the simulation is 0.0001 s then 0.0001/0.0000125 = 8. On this extra step the network is just advanced one time unit, 0.1 ms in this case.

If the input file for the LSM has an event associated with the current simulation time, the current state of the liquid is saved to an output array.The information (output array), is saved to the format needed for the DBN. The format needed by the DBN is a pickle file and is discussed further in Section 3.4. The saved state is a set of floats and a target value. The target value is the number associated with the transformed mouse data. There are also classes for the states of many buttons that the user might be holding down. The floats are the recorded analog values of the simulation's neural membranes that were sampled at random from the liquid.

It is believed, due to the nature of technological progress, that computers will continue to become more powerful. With an increase in computational power, this network of 4000 neurons and perhaps larger LSMs can be created and simulated in real time. The second thing that could be done is that the network could be reduced in size and parameters tested to get faster results.This is explored more in Chapter 6, in the future work Section 6.5.

3.4 Reformatting LSM Output for DBN Input

The last part of the exploratory BCI system is a DBN, which processes the state of the LSM. The information about the state of the LSM is in the form of arrays of floats.

The Theano DBN uses pickled files for input. Python pickle is a form of serialization: a way to store or save Python object hierarchies without losing information. These files can be later deserialzed to reconstruct the original Python objects [10]. The internal structure is three sets of data: one set each for training, validation and testing. In each of these sets there are two lists: one for the target class and one for the matching state array. There is a small program that reformats the pickle file from the LSM into the correct format for the DBN. The format is shown in Figure 3.4.

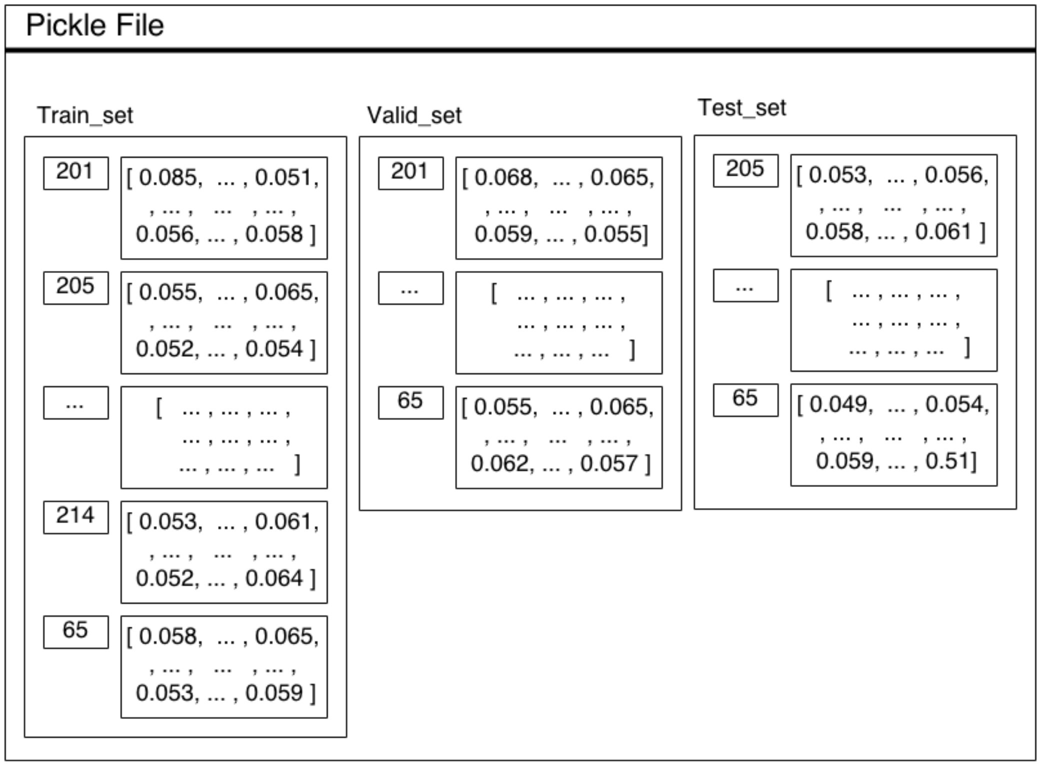


Figure 3.4: The format of the pickle file created from LSM output. The format is used as input for the DBN. Three sets of data are contained within the file: one for training, one for validation and one for testing. The training set is five times larger than the validation and testing sets. In each set there are two lists: one for the classes and one for the data corresponding to that class. (Created by Jason Cummer)

3.5 DBN

The DBN used in this study was based on the Theano version of a DBN at Deeplearning.net [4]. Not much had to be changed to make the data from the LSM flow into the DBN. The number of inputs had to be altered to match the number of elements in the state arrays from the LSM and the number of outputs had to be altered to match the classes in the output. All other parameters remained the same, initially. Minimal changes were done to tune the network. More of the details for this will be covered in Chapters 4 and 5.

3.6 Summary

The use of a recurrent neural network that utilizes spikes in a manner similar to a biological neural network was tested in this thesis as an exploratory study of the classification of EEG data. The specific implementation used was a LSM and a DBN for classification. The EEG data were transformed into a set of fourteen spike trains with a semi random conversion technique. The key press and mouse events were propagated forward to be used in the LSM and finally in a classifier. Chapter 4 will demonstrate that, though it is unclear from the results if any classification possible, the pipeline architecture is both flexible and useful for troubleshooting. Much more tweaking and investigation should be done with this combination of algorithms to find a clear result, along with a substantial set of users to execute a definitive case study.

Chapter 4

Exploratory Experiment and Results

This chapter delves in to the details of the exploratory system that attempts to read the intention of a user and translate the intent into input for a computer. We will look at the details of the system, from the computers that it was run on to the design of the pipeline architecture and an example of the system using specific components for neural networks. Finally, we will look at the results of the investigation.

4.1 Setup

The computers used in this study were quad core Intel I7s with four, eight or twelve gigabytes of random access memory (RAM) as described in Appendix A.1. The computer that ran Theano had 4 GB of RAM and two Nvidia Tesla cards with 1000 CUDA cores each. The virtual machine (VM) that was utilized, was an 8 GB machine with three cores.

The data for this project were all electroencephalography (EEG) recordings from the author, while freely operating and programing on the recording computer. This was done with the use of User32 platform invokes to collect the mouse and keyboard data. The brain's electric fields were sensed with the Emotiv EPOC headset and recorded to the computer.

These EEG data and the mouse direction data were tested in WEKA: both the direction based data and raw data were run through the naive Bayes, Bayes network and multilayered perceptron algorithms. The direction based data are comprised

Figure 4.1: The Emotiv EPOC headset [6]

of the mouse event data and one second chunks of EEG channels. These chunks are taken from the overall data such that only the second directly preceding the corresponding mouse event of the channel data is used. For the raw data, it did not contain information separated into one second chunks. It did contain the changes of direction, rather than the mouse positional data. WEKA used the raw data to create a model that represented the classes. Neural nets with greater utility had to be tried next. So preparation of these data were adapted, not for WEKA, but for the LSM and DBN.

4.1.1 Spike Train Generation

The input for the LSM needed to be in spike times for the LSM to accept it. To transform the EEG data to spike trains the value of the EEG recordings had to be normalized and brought into a range that was similar to a natural neural network (NNN). The range of EEG data is received in a range of 0 to 8000 muV. The actual numbers in the file sit around 3500 to 4000 muV. The means vary, based on each of the channels. To calculate the spike trains, the maximum firing rate of neurons must be known, though this rate is variable. For this experiment, the maximum firing rate was assumed to be 200 spikes per second. This number is from the typical action potential; it includes the relative and absolute refractory periods as recovery times between the neurons action potentials. The use of only the absolute refractory period could be utilized for recovery. Assuming the absolute refractory period is 3 ms, this would yield a maximum firing rate of approximately 333 Hz. The problem with the firing rate of 333, is that to overcome the relative refractory period's hyperpolarization you have to have a larger input to the neuron then the initial stimulus. Though, if one was trying to simulate a real neural system it would seem that a higher number of inputs would have to be accounted for to simulate overcoming the hyperpolarization of the post synaptic cell. Coming back to generating our spike trains from the EEG data; the value of the EEG is divided by the maximum value of the channel (8000), then multiplied by the maximum firing rate. If the spike rate is greater than the maximum firing rate then the firing rate is set to the maximum firing rate. The spike rate is then appended to an array of spike rates.

```
spikeRate = ( ( float( arrayIn[i][0]) / max) * 200 )
    #if it is over the largest threshold bring it into range
    if ( spikeRate > 200 ):
      spikeRate = 200
    subOutPutSpikes.append(spikeRate)
```

The array of spike rates are then multiplied by 1/128 of a second. This gives a spike rate probability for the time step of 1/128.

```
subProbList.append(spikeRateArray[i][j] * TIMESTEP)\\
```

The maximum value is 1.5625. 1.5625 is the value of the maximum spike rate divided by 1/128th of a second: (200/(1/128) = 1.5625).

The spike rate probability is then used to come up with a spike train. There is a maximum of 1.5625 spikes per time step in the initial EEG sampling rate of 128 Hz. If the spike rate probability is higher than one, a spike time is added to the output spike train. If there is a probability remainder, a random number is generated. If the random number is less than the remainder, a spike time is added to the output. Likewise, if the probability is below one, it is treated like the remainder from the greater than one example. The appropriate amounts of time are added to the time count to keep the count in data in the correct alignment with the events.

The next few lines of code below show how the time is added to the spike train.
BuildSpikeTrain

```
if spikeProbabilities[i][j] > 1:
        time += THIRDTIMESTEP
        singleChannelSpikeTrain.append(time)
        time += THIRDTIMESTEP
        spikeProbabilities[i][j]-= 1
        if (randint(0,100) * 0.01) < spikeProbabilities[i][j]:
                singleChannelSpikeTrain.append(time)
        time += THIRDTIMESTEP
elif (randint(0,100) * 0.01) < spikeProbabilities[i][j]:
        time += HALFTIMESTEP
        singleChannelSpikeTrain.append(time)
        time += HALFTIMESTEP
else:
        time += TIMESTEP
```

Spike trains are in the format of a list of spike times. The list of spike times for each channel acts as input to an input neuron in the LSM's set of input neurons.

4.2 Liquid State Machine

The LSM in this study evolved from the spatial_inh_exc_populations.py file on the main PCSIM website A.3.2. Functions that were added are as follows: Data were read into the LSM from files containing the spike trains and the targets. One of the tasks that needed to be repeated many times was reading the state. Thus, reading the state of the liquid was moved into its own function. A section in the simulation code was added at the Python level for testing for events. This section took care of how much time to advance the LSM. An attempt was made to compensate for the time drift of the remainder of the 1/128th of the simulation: the 0.000 012 5 s. If there was an event detected, the event was added to the output array in preparation to write the output pickle file.

Functions were also added for saving the output as a pickle file. This is so that the DBN would accept the file. Ultimately, the output pickle file would have to be reworked, which is covered in this Section 4.3.

The LSM was created with 4000 neurons in the liquid. The neurons in the liquid were composed of both excitatory and inhibitory neurons. The LSM was built on a cuboid grid with 20 x 20 x 10 neurons as a SpatialFamilyPopulation.The spatial family population is a population of neurons with built in spatial coordinates. The coordinates can be used to calculate the probabilities of connections based on distance. The LSM has a 4:1 ratio of excitatory to inhibitory neurons. The neurons are all leaky integrate and fire neurons. The network simulation is a single treaded local network. Details about the settings for the excitatory and inhibitory neurons are in the Appendix A.5

4.2.1 Input Neurons

Spike trains generated from the EEG data were used as input for the LSM's network. The spike trains are represented as a series of time values. The times tell the system when a spike occurs. Input neurons were created with the spike trains as a parameter which was passed into the neurons' constructors as an array of time values in milliseconds. These neurons connect to random excitatory neurons in the liquid.

4.2.2 Recording Neurons

Spike Time Recorders
The spike time recorders connect to all the neurons in the liquid and record the times at which there is a spike. The spike time recorders were left in but were not really utilized for the testing that was done. Some uses are discussed in the evaluation and future work sections.

Voltage Membrane Recorders (VMR)
The VMR neurons are connected to a random sample of 256 neurons in the liquid. For Theano's internal processing, the 256 connections make for a square matrix. A population of AnalogRecorder() neurons is created and then connected. The analog recorder neuron records the membrane voltage of a cell in the liquid so that it can be used for output later. Sample time is one millisecond.

4.2.3 Simulation of the Circuit

The time at which the simulation starts is taken. The network is then reset. The network steps through the number of steps specified in the input file, for the desired simulation time. The number of steps originally comes from number of samples there were in the EEG recording. The number of steps propagates through all the file processing steps before the LSM.

The simulation has a resolution of one millisecond. The original sample was taken at 128 samples a second. So that is 78.125 milliseconds. The LSM was told to process 78 steps (seventy eight milliseconds).

Time Compensation

Due to the remainder of the 128th of a second ($1/128 = 0.0078125$, $0.0078125/0.0001 = 0.125$), it is necessary to compensate for the difference. The difference is ($0.0001/0.0000125 = 8$) of $8 * 0.125$. A **for** loop was introduced to compensate for this drift in time. Every eight loops the program advances the simulation one more time and checks the events to see if anything will occur during this time step. That is to say every eight loops, of 78 loops, there is an an extra simulation step (net.advance(1), in code).

During the time that the LSM stops, between its 78 ms and compensation times, it looks through its list of events. The LSM checks the directions, the mouse buttons, the states of some of the keys that might be in a held state and the key codes of most of the main keys. If there is an event at this time, the LSMS writes the event class to a list and retrieves the state of the liquid based on the analog recording neurons.

The LSM can take a lot memory, and initial investigations were hampered by the virtual machine crashing after too much simulation run time. A solution that was pursued, was the clearing of the analog recorders. Their recordings and states were reset every 78 steps. This solution has allowed the virtual machine of 4 GB of memory to run the simulation. Other papers do not use files as long as the files used in this simulation, so it may never have been a problem before. The counts for the types of events are taken so that they can be verified with later processing.

4.2.4 Pickling

For the later processing of the data, a DBN is used. The DBN used in the exploratory system utilizes Theano and takes input in the form of a pickled file. There are three arrays of data for the DBN: one for training data, one for a validation, and one for testing. The arrays are setup in the following format:

```
[[class/target <int> ],[ <list> [information,<float >]]
```

4.2.5 GZip

The DBN takes its pickled input as a GZip file. After the processing of the spike trains is finished by the LSM and the input arrays are processed, the command gzip <filename> will compress the file to a .gz file.

4.3 Reprocessing for Training, Validation and Testing Sets

The ratio for the training, validation and testing set was determined by reading the pickle file for the Mixed National Institute of Standards and Technology (MNIST) data and retrieving the length of the arrays.
Train:Valid:Test
50 000:10 000:10 000
or
5:1:1

The format for all three input sets was in the following form:

```
( array(
  [
    [0., ... , 0. ],
    [0., ... , 0. ],
      ...
    [0., ... , 0. ]
        ] ,
    dtype=float32 ,
    array(
     [5,0,4, ... ,8,4,8]
     dtype=int64
    )
)
```

The data from the LSM was in one list so events had to be separated into training, validation and testing sets, as can be seen in Figure 3.4. This was done by reading in the data and counting the number of instances of all the 154 classes. If there were fewer than five instances of a given class, the count was used to determine what should be done. On a count of one, the three sets all receive the same instance of an event. Then the count for that class was increased by one. A pointer to the output arrays is kept for all the classes in the three sets; this points to the last updated position. The pointers to the last updated positions are used in the later encounters of classes to replace replicated instances of events. On a count of two, that is the second time a class code or target is found, the values in the training and validation sets are updated so they differ from the testing set. On the count of three, the only value that is updated is the value for the training data. On counts from four to seven, the test set is appended to. This makes five values for the training set and one for both testing and validation sets. This would allow for sets that might not have the ratio of five to one. There could also be a set where all three sets have the same list of floats for a given event. If there were no examples of a class, it was not included in the training.

A later revision to the reformatting was done. The reason for this was to exclude classes that did not have the required minimum number of training examples for the DBN. This is important because, if there was a class that had any overlap in the sets, the training would be compromised. It could appear that the training was better then it was, but if the trained DBN was used for classification, the results would not be as good as expected. The later revision saved all the examples into separate arrays as they were encountered. When the count reached seven, the examples were appended to the appropriate lists. This ensured that there was no overlap in the data sets and that the validation and testing would be accurate.

4.3.1 Deep Belief Network (DBN)

For the implementation of the DBN, I started with the Theano DBN which was used to classify the MNIST data set. I altered only a few settings including the number of inputs, the number of outputs and the number and sizes of the layers. The learning rate had no modifications.

The following two settings in the python code were not altered: a random number is created for numpy_rng = 123; Random number generator. Theano_rng = none.

Theano random generator; if none is given then one is generated based on a seed drawn from ’rng’.

The default input size of the DBN in the Theano “getting started” tutorial is 784 inputs. I altered this to a number that was close to the value of the outputs from the LSM, and a square, so that it can be input more easily in matrix form: 256. This number could be increased to be more similar to the 784 used above, but one would have to go back to the LSM for that modification, for the output from the LSM is the input of the DBN.

The DBN started with three layers of 1000 neurons each. The intermediate layers must contain at least one neuron each. The size of the intermediate layers is a parameter that was not experimented with much in this study. One or two layers were added or removed at a time, with minimal effect on the validation error. The size of the first layer was altered to an order of magnitude larger or smaller. This also had no effect as validation error was never below 99%.

The number of outputs for the DBN was originally 10. This was the number of targets in the MNIST data: one target for one of the integers from 0 to 9. The EEG data contained 154 classes. Most of these classes were not represented in the data fed into the DBN. Once again, if there are less than 7 examples of a class, that class is not represented. This is due to the stratification od the data.

The LSM output needed to match the DBN input, which was set to 256 input neurons. This was the first major impediment to having the LSM output be usable in the DBN input. Once the output neuron number was altered for the LSM, the output from the LSM process was able to be fed into the DBN. The validation error was still 99 to 100%.

4.4 Evaluation, Analysis and Comparisons

The remainder of the chapter shows the results of the LSM and DBN system. The results of the original MNIST data set is included. The MNIST data are handwritten digits from 0 to 9. There are 60 000 examples contained within it. It is a standard set of items with which to test classification algorithms. The MNIST data set is discussed in more detail in the next chapter. This is to contrast the results from the EEG data.

4.5 Electroencephalography Results

This section shows the results of training the DBN after processing all the data. Table 4.1 shows the parameters that were tried in DBN of the exploratory system. More files and more tuning of the network should be done, but this is discussed in the future work section.

Inverted	**Input Numbers**	**Layers**	**Neurons Out**	**Best Validation Error %**
false	256	[100]	153	99.629 630
false	256	[1000,1000]	153	99.629 630
false	256	[10 000]	153	99.444 444
false	256	[1000,1000,1000]	153	99.259 259
true	256	[1000,1000,1000]	153	99.629 630
true	256	[12 layers 1000]	153	99.25

Table 4.1: DBN training results from EEG data collected on January 18th, 2013, for the hour 13:00

One can see from the class count that there are only 12 classes that had more than seven instances in the data. Table A.4 shows the class counts for the EEG data collected on January 18th, 2013 for the hour 13:00. As a result of analyzing the numbers in this file, the number of outputs for the DBN was reduced to 12, for the classes that had representation in the data (more than seven instances), to see if this had any effect on validation error. As can be seen in Table 4.2, there is little effect.

Inverted	**Input Numbers**	**Layers**	**Neurons Out**	**Best Validation Error %**
true	256	[1000,1000,1000]	153	99.629 630
true	256	[1000,1000,1000]	12	99.629 630

Table 4.2: DBN training results from EEG data collected on January 18th, 2013, in the hour 14:00

The next input error that was discovered was that the output from the LSM should be between 0 and 1, but the actual output was between -1 and 0. The output from the sampling neurons in the LSM was the membrane voltage, which has a negative resting potential. The simple alteration of multiplying the inputs by negative one was tried, but had no apparent effect on the validation error. The only effect was the cost in the pre-training functions. Before, the cost of the pre-training had always been above zero and proceeded to numbers greater than one million. After the multiplication by negative one, the cost was closer to the MNIST data sets: the cost value started at -91 for layer 0, epoch 0 (the first of 100 training stages for each layer, epoch 0 to epoch 99) and by the end of the pre-training of layer 0, at epoch 99, the cost was -55. By contrast the MNIST data set at epoch 0, layer 0, was -98 and moved up to -68 by the 99th epoch. Ultimately the inverting of the input data had no effect on the final validation error. The results are shown in Tables 4.1 and 4.2

4.6 MNIST Results

Here is an example of some of the numbers from training the Theano DBN with the MNIST data set: In the 127th epoch, mini batch 5000/5000, the validation error was 1.36%. The best validation score occurred earlier with a result of 1.35%, with test performance 1.37%. The total time to complete the fine tuning stage on the MNIST data set was 46.85 m, much longer than the time for the EEG data sets.

4.7 More Results

Retrieving outputs from intermediate steps in the pipeline has not been implemented yet. This means that it is difficult to get results for training of specific classes, either during validation or testing of the model. There appears to be an output array that holds the testing data for verification in Theano's DBN implementation. It is a shared array, which means that it is accessible to Theano and to the graphics card. The structure of this array is not known at the time of writing, but these data should be sought after for future understanding.

Early on when I was starting to test with the DBN, the input arrays seemed to be the incorrect size for the gemm function. The reason for this was that the input number was too high for the expected input. The number of inputs did not match the expected number of inputs from the original MNIST based DBN. This caused the DBN to expect 784 inputs, but was getting 250 initially. This would cause the CUDA code to throw an exception. I added six recording neurons to the LSM, this makes it a square (16 x 16) so the matrix operations in CUDA do not have wasted space.

Changes were made to fix the LSM's output. It was passing all 78 states from the recording neurons, not just the one that was currently relevant for a particular event. The fixed LSM only passes on the state of the LSM from the millisecond at which the event is detected. Once again, this had no impact on the validation error. This fix is one that is fundamental to the design, though. Some of the data from the previous states of the LSM should be propagated into the current state. This means that only the last state need be sampled. This is one of the main reasons for using a recurrent neural network like an LSM.

While the results from the DBN may not be promising at this time for the basis of a BCI, there are lessons to be learned. The piperline architecture facilitates the investigation into possible reasons, processes and programs that may have created problems for the classifier. The parameters for the DBN may also be incorrect. The information intrinsically may not be suitable for the DBN or the combination of the LSM-DBN. It may be that there is some ability of the DBN to classify the EEG data, but it is not evident from this exploration. We will look at these sources of problems in detail in the following chapter.

4.8 Summary

EEG data has a spatial temporal nature to it that should be analyzed on a system that is designed to process spatial temporal data. One system that is designed to help the analysis of temporal data is the LSM. The use of an LSM to assist in the classification of EEG data was examined here. The LSM and the DBN, in this test, were able to work together in the sense that data could be passed from one to the other. Whether that is useful for classification in the DBN is not necessarily known.

This exploratory study shows that the LSM and DBN did not do a good job at classifying EEG data. With a validation error of approximately 99%, the system could not classify any of the target classes. The error could have come from any stage in the pipeline: the EEG, data transformation or processing, the LSM, or the DBN. The good news here is that exploring the problems is not hampered by a monolithic representation. Instead, a systematic approach to trouble shoot the results can correspond to the stages and modules in the pipelined architecture. There are plenty of directions in which to test the system and other ideas to try. A newer neural net, the NeuCube, reported results of 100% accuracy for one class in a classification of EEG data [42]. The NeuCube shows many similarities with the system considered here. The NeuCube and the future tests and ideas will be examined more in the next chapter. Possible reasons for the poor classification will also be discussed further in the next chapter, as we examine the components of this system one by one.

Chapter 5

Discussion and Trouble Shooting the Pipeline

In this chapter we look at the various potential reasons why the combination of liquid state machine (LSM) and deep belief network (DBN) did not produce a functional classification system. Figure 5.1 shows some of the modules that might have errors. All components of the system will be covered, starting from the electroencephalogram (EEG) used to collect data, moving through the stages of processing the data for both the LSM and the DBN and finally noting potential sources of error in the neural networks employed in the system. This process demonstrates the advantages of the modularity of the proposed architecture to localize trouble shooting when experimenting with different algorithms and assumptions in the system.

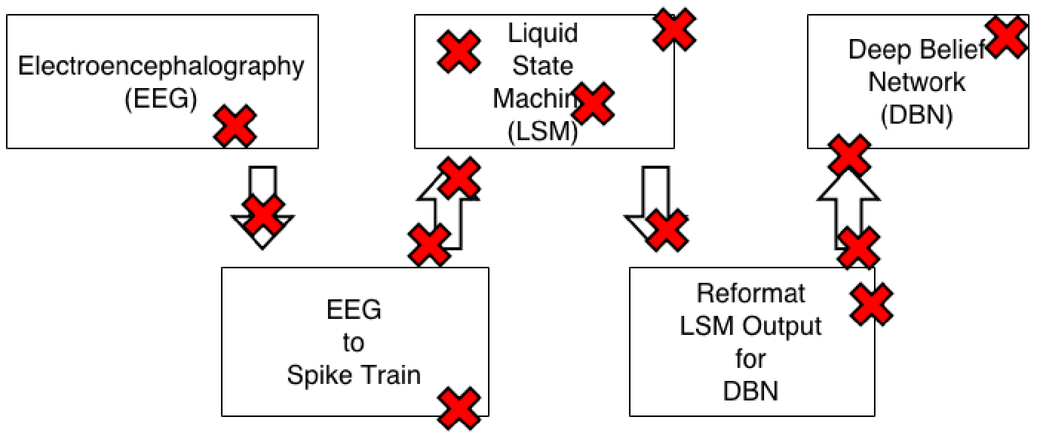

Electroencephalography (EEG)
Liquid State Machine (LSM)
Deep Belief Network (DBN)
EEG to Spike Train
Reformat LSM Output for DBN

Figure 5.1: Locations in the exploratory system that may contain faults which contribute to the system's error. (Drawn by Jason Cummer)

5.1 Systematic Remedies Aligned with Modular Decomposition

The stages which may contain errors are EEG, spike train conversion, LSM, data stratification and the DBN. With the EEG there may be spatial resolution problems, myographic interference issues or the headset needs more time to collect enough data for a state determination. The conversion of the EEG channels to spike trains might cause overstratification or pathologic synchrony. Some of the problems with the LSM could be network dynamics caused by the input or internal network properties. Data alignment might also cause the classes to be out of sync with their states. Data stratification, relating to having the correct number of examples for the training validation and testing sets, is a factor. The DBN likely needs tuning to use input data for classification and there could also be some hidden classification.A systematic examination of the possible issues of the exploratory pipeline follows.

5.1.1 Electroencephalogram

The EEG sensors may not have fine enough resolution for determining the user intent behind the commands. Without access to the proprietary algorithms that Emotiv uses for the EPOC, the spatial resolution might be too limited. The space between the brain and the electrodes of the EEG can cause distortions of the electric fields. These distortions make it difficult for classification of the signals that originate in these regions. The Emotiv EPOC uses training data for a neutral state that is 30 seconds long. Then the other states take 10 seconds to train, but may have to be trained multiple times for good results. One other issue is that EEG in general has to deal with large amplitude noise from muscle contractions. This type of artifact would overpower the signals from the brain. A module for removing artifacts and extracting what information is left in the data should be investigated.

In 1999, Karet found the typing rate of composition to be 19 words per minute [41]. The composition rate from our data was not determined from the collected data, but could be. If one assumed 19 words per minute, that is approximately 3 seconds per word. The average word length from English, at the time of writing this thesis, was 5.1 characters [17]. This gives each character 0.6192 s. This is a short amount of time to look for a general increase in brain activity in a region. In the brain, some of the regions that generate typing behaviour are the pre-motor, motor cortex and Wernicke's area. Without high resolution information from these areas the information may be averaged together. With low resolution, it would be difficult to distinguish the motor signals in these regions. The homunculus for motor movement has regions representing the hand. Within each of these regions are representations of smaller and smaller regions of the hand. As the region of the hands become smaller, their corresponding cortical representations are smaller. The difference between the action of pressing the r and t keys is small in terms of the resolution of the EEG. Despite the fact that Migal Nicolelis found that information for most processes can be found in neurons across the brain, the information may not be available to the low spatial resolutions of EEG.

Another possible fault with the EEG stage of the pipeline is all the combinations of electric fields occurring. With noise from other electronics, information can be lost. The state of the brain over longer time periods may allow for clearer data to be generated.

It may be possible to recover the information from the EEG necessary to have a real time brain-computer interface (BCI) from the EPOC. My current setup does not allow for this. The quick set of readings I collected are certainly shorter then 10 to 30 second recordings Emotiv uses. So the data collected may need to be extended. The amount of information collected in a longer time window. This is entirely possible with the LSM setup. Infact, this should already be happening. The recurrent connections in the liquid should be holding information from the previous states.An LSM though it has this ability it is more of a fading memory. It might be the case that the information in the recurrent connections is not lasting long enough. This could likely be fixed by alter the composition of the liquid. It is some what dependant on the randomness of the network's creation.

5.1.2 EEG to Binary Files and to Text Files

The functions in the EEG recording program that wrote the data to a binary file were tested during development. The functions were found to accurately write to the file. The file extraction from binary to text files was also tested during development and was found to be correct.

5.1.3 Pre-processing Text File: Mouse Point Information to Cardinal and Intercardinal Directions

User key press and mouse data may have been lost after the addition of code to include the keys into the output file. It seems unlikely that this would have happened due to the testing during development, however there was not strict unit testing. Creating some unit tests for the program would have resolved the question during development.

5.1.4 Neutral State

One potential confounding issue is the use of a neutral state for training the DBN. There was no neutral state class used in this study. The main use of a neutral state is for the classifier to have a state for when the user is idle. In the idle state, no user output would be generated. For a neutral state, there would be a class defined so that the classifier could generate an internal result that has no assigned output. It is possible to get the data for a neutral state from the existing EEG recordings. To collect the requisite data, a function can be created to keep track of the time of the EEG data, and the user's input. If the user does not enter any input for a time period, say two seconds, save that time point to a data structure (a list or array). The reason for this relates to the work from Libet (see Figure 2.2.4). If there was two seconds of time, it should be the case that there is no intentionality for any event of interest. If two more seconds passed and no input was detected, then the recorded time point could stay in a data structure. If input was detected, the time point would be removed from the data structure. When the file is to be saved for the next step in the pipeline, the information in the data structure is written to the output file. There is no need to extract the neutral state at the moment, as we are still trying get the key and mouse classifications operational. This process can be revisited if the classifier is able to extract the intended output reliably.

5.1.5 Pre-processed Data to Spike Train

The EEG channel data were converted to spike trains for LSM input by one of the programs that was developed for this study. One problem that might have occurred in this program is data misalignment of spike generation frequency.

This type of error may result in user key and mouse events aligned with incorrect brain states. These incorrect states would manifest in the liquid as the target classes not being aligned with the EEG states. After examination of the process this possibility was dismissed. The spike trains were generated based on their EEG values in the data and some constants related to how the brain generates spikes. The spike trains were generated without affecting the recorded key and mouse events. The time of the events can be found independently from the time of the spikes in the spike train. The event classes are passed on through the pipeline, as they are, in their own streams of input. Each one of the values in the stream represents 128th of a second. These values, which are the class information, are processed further in the LSM. As the LSM is simulated, the input is tested for events. If there is an event, then it is linked to the state of the liquid at the point in time of the event. If there is a data misalignment error, it is in the LSM.

There could be an error in the network dynamics of the LSM, which originate in the generation of the spike trains. If the number of spikes is too great, the network becomes saturated. This could happen if the probabilities chosen for generating a spike were too high for the LSM, see Section 4.1.1. The probabilistic way the spikes are generated seems to follow a logic that would approximate a natural spike rate. If the odds were greater than one that there would be a spike within the time period of 0.007 812 5 seconds, then one was added to the spike list. The max probability for a spike generation is 1.5625 for a value equivalent to 200 spikes per minute. There are values lower than 1.5625 but higher then 1, so after the first definite spike is generated and the definite probability of one is subtracted there is a remainder. An example would be an EEG value of 6000 muV would have a probability of generating 1.171875. This means there is one spike in that time due to the one. The one is then subtracted and the remainder is 0.171875, which is compared to a random number to see if a spike is generated. If there was a remainder, then there would be a chance for a spike in that time period, and it would be generated. The probability is tested against a random number and the result of that test generates a spike or not. If a spike is generated, it is added to the list of spike times at a fraction of 128th of a second later. Problems may be introduced if too many spikes are generated by the function used to generate and process the random number and the probability. This would result in all states associated with target classes having almost all the neurons at full excitation. If there are too many neurons firing at the same time, this has to be known so steps can be taken to ameliorate this.

One other thought is that the input from the EEG channels hovers around a central value that varies by approximately +/-100 μV. The normalization to fit this under 8000 μV results in about 2.5% variation in the signal. With this much normalization the signal might be lost in the noise and become too difficult to discriminate.

If the frequency of the spikes is too low, the network might not have sufficient power to continue firing after a stimulus stops. As with the case of too many spikes being generated, the spikes would follow the same generation method as discussed in Section 4.1.1. The manifestation of this effect could be little excitation at the time of recorded events. The resulting LSM output would have little to no variation, as all states would have same neuronal resting values.

If the probabilities of the spike generation was off then there might be too many or too few spikes. The number of spikes generated by this algorithm has not been tested. Testing should be done to determine errors introduced by the spike generation.

5.1.6 Liquid State Machine (LSM)

The LSM could have had problems that relate to data misalignment or network dynamics. The data misalignment problems, in the LSM, are related to remainder compensation or advancement cycles.

A major error discovered was the math for the main loop of the exploratory LSM program. This main loop was for finding how many times to call the network's advance function (net.advance()). In the read file function, the number of samples was taken from the first line of the file. The first input from the spike train files are a number that represents the number of total samples. That is the number of samples that were written to the file, one every 128th of a second. Ultimately this is used to calculate the amount of total time for which to simulate the LSM.

The first thing done is to find the simulation time. An example here of a large file is concurrently used to illustrate the calculation.
From the first line of a spike trains file we get TotalTimeSteps = 460 349. The level of precision is just from the initial 1/128th of a second, which is 0.007 812 5 s. To set the total simulation time (Tsim), we have Tsim = 0.007 812 5 * TotalTimeSteps. Multiply the 128th of a second by the number of time steps in the file. This yields $0.0078125s * 460349 = 3596.4765625$ seconds. The PCSIM internal setting for advancing is set to DTsim $= 1e - 4; 0.0001$ ms.

A constant was created for representing 128th of a second, it is oneTwentyEightInt= 78. To find the number of times to loop the 128th of a second simulation loop I did the following. (Tsim/DTsim) / oneTwentyEightInt , for our example that is (35964765.625)/78 = 461086.738782. The value 78 is a problem and causes an alignment error. The 0.0001 can be used to cancel out the 0.007 812 5 and that leaves 78.125 multiplied against the total time steps in the file to give 35 964 765.625. Upon conversion back to the number of simulation steps used in the loop for the net.advance(), the number of 78 was used to get the number of times to loop through the advance loop. The result from 35 964 765.625 / 78 is 461 086.738 782 0. As an integer the result is 461 086. The difference is 460 349 - 461 086 = 737. This amounts to 5.763 584 seconds over the recorded hour of time. The 5.763 584 seconds of drift divided by 60 minutes is 0.096 059 7 seconds of drift per minute. If I once again use Libet's one second of intentionality, the simulated LSM would take
$1 second/0.0960597 second/minute$ = 10.410 minutes of time to drift the one second. With the LSM operating with this drifting, the states and target classes would really start to be misaligned after approximately 10 minutes. This is obviously one of the reasons that the classification is wrong. If there are shifting examples of states from the LSM that the DBN is trying to learn it would be "confusing".

It appears that dealing with the remainder, and the incorrect math both resulted in data misalignment.

5.1.6.1 To .advance() or .simulate()

The reason for the conversion between TotalTimeSteps and total simulation time (Tsim), was because at the time of programming the LSM, the minimum advance time was assumed to be 0.0001 s. Many of the examples of PCSIM had this number with the net.advance() function. The number of times to advance the net.advance loop is an issue of how many advance cycles go into the network. There might be a minimum for PCSIM's networks, but that value of DTsim $\bar{0}$.0001 is set in the example files and not well documented. The minimum is unclear. PCSIM does not specify a minimum for this variable in any of the files I explored. The net.advance() function takes an int, and has a minimum set to one. The common use of DTsim and the default int value of one in the advance function implied a minimum. The PCSIM examples with DTsim $\bar{0}$.0001 seemed to imply convention. It would have been better to set the advance time to 0.007 812 5 s for the network, to match the EEG sampling rate.

A different way to simulate the LSM was discovered. With the same net = SingleThreadNetwork(), there is a simulate function as well: net.simulate(<time in seconds>) is the signature. An example is net.simulate(0.2) for 200 milliseconds of time. Searching for this functions in the code was not intuitive. Generally class.function() implies that function() is within the class. The function "simulate" was nowhere to be found in SingleThreadNetwork or its documentation. Upon a search through all the header files, the function was found in SimNetwork.h [12].

A better way to utilize the LSM would be to use net.simulate(0.007 812 5). This would allow for some code simplification in the main loop of the program. This simplification should allow for the removal of the data alignment error. The math above would be removed as the number of 128th samples would be the number of times to do the loop.Then the loop would simulate 0.007 812 5 s and test for the target classes, thus removing the two main errors so far identified in the LSM.

There could be more constants causing errors or other programing errors that were less obvious, which would require more experience with the PCSIM software to learn.

5.1.7 Network Dynamics

The network dynamics of the LSM might also present a problem, either with the input data or the network's setup. The setup that I used for the liquid was not altered much from the original example file [12]. The only edits to the working program were additional inputs and outputs. This mitigated problems of introducing bugs. There are the different ratios of excitatory to inhibitory neurons, as well as the strength of those connections, that could be experimented with. If the ratios are off, then either states of pathological synchrony or overstratification could occur. The use of the spike time recorders could diagnose this possibility and create feedback for training the LSM. This is discussed more in the future work section.

The spike time recorders could be used to see the states in the liquid of the LSM. PCSIM is able to use pylab to create charts of the data from the LSM. Pylab is a python library for numeric analysis, which includes the creation of charts to visualize data. These charts could be used to determine if the network was functioning correctly. Such charts should be able to determine if the network was in a state of pathological synchrony or overstratification and would help to tune the LSM. The tuning, altering the ratio of the excitatory and inhibitory neurons, may bring the network to a level where there was separation between all the classes. One could also tune the types and properties of the synapses to change the information flow in the network.

5.1.8 Robust Timing and Motor Patterns

“The main challenge has been that recurrent neural networks operating in ’high- gain’ regimes in which recurrent connections are strong enough to generate self-sustained patterns of activity are highly sensitive to noise and are often formally chaotic. Thus, although the dynamics in these networks are potentially computationally powerful, the fact that minute levels of noise can produce vastly different neural trajectories effectively abolishes their computational power because a given pattern cannot be reliably reproduced across trials.” [46]. If the noise is too much then it is a case for altering the network. Unless the noise in the EEG is just to much for the network to handle. I would hope that at least 70 percent of the data could be recovered from some classes. WEKA was able to do some classification in the earlier experiment. The results were with the raw EEG data(not normalized) with the cardinal and intercardinal mouse directions, see Appendix A.2.

One other piece of information that came later than I would have liked: I had posted to the Source Forge online help forums to find out how to get data into the LSM. I got a response, but it was after I had done my initial creation of the program. The response, was about a StaticAnalogSynapse.
$net.connect(inp\_nrns, nrn, StaticAnalogSynapse(delay = 1e-3, W = 1e-10))$
StaticAnalogSynapse would have been the method to use as it takes normalized input. Instead of creating spike trains from the original stream of EEG data, one would simply normalize it into a range the network could use. This could be done in the LSM if a function was written for it. This might also alleviate the network dynamics problems. Spikes would automatically be created from the neuron that received the analog input. These spikes would then be fed into the main liquid. By using the static analog synapse it would provide a contrast for problems arising from my spike train generation function.

5.1.9 Deep Belief Network (DBN)

The possible errors of the DBN could be insufficient data, non separable data, data that are not similar enough to learn, configuration, or perhaps the classification might not be evident.

The EEG information may not be classifiable. The MNIST set has 60 000 examples in it, for MNIST information see Section 4.4. There are 50 000 training examples in the training set and in the validation set there are 10,000. This is based on reading the MNIST file and calculating the ratio of training to test and validation to be 5:1:1. The data were then formatted according to this ratio: 2745 for a training set and 549 for the validation and test sets. The MNIST data set should have about 6000 instances of every class. That is 6000 for each of the ten numbers from 0 to 9. The data used in this study have more classes and far fewer examples. The number of training examples might be 700 for a best case of a direction (see Appendix A.10). The number of examples for the most represented classes are more than an order of magnitude smaller then the MNIST training set. The size difference between the MNIST and EEG data sets used for the DBN has likely had an effect on classification training. That effect being that there is insufficient data to train the DBN on the EEG data set.

Initially some of the same data points were used for training, validation, and test. This is not good for the final results. The reason for the overlap was an attempt to see if the DBN would even process the data, regardless of correctness of classification. Once the DBN did accept the data, the reformatting program was altered. This resulted in sets that were in a ratio of 5:1:1 that had no repeats of example states. With the new data format, the validation error results were still high (See Appendix A.12). Machine learning techniques sometimes use a technique called stratification. Stratification is when you have examples of your classes in all your training sets [61]. In this case for training set, I am referring to the training set, validation set, and testing set. Stratification was used, as there is at least one test for each of the five training examples.The classes that have seven or more examples are stratified.

Target classes might need to be represented in proportion to frequency. Depending on what was typed at the time of recording, the inclusion of certain key codes can vary in the EEG and class data files. If, say, the Z key was pressed in one file but not in another, does this mean the number of outputs from the DBN must be changed for each file? There would be 154 classes with all the key codes and the null states.

The modification to include 154 output classes did not seem to have an effect on the classification ability. The classification currently does not need the null state. An offline system was discussed in Chapter 3.

One issue with the data recording and processing is that there are classes that only have a few or even one example in the set. This is not a good representation. If the examples that come out of the LSM are not as random as initially thought, there might be a chance to add the output from various files together for the DBN. Examples from many of the files compiled into one set may make the training of the DBN more accurate. This compilation will be addressed further in the future work. The classes with low representation, if allowed to train on the DBN, would have a lesser effect on the network then other more prominent classes. The classes that have only a few examples would have the worst testing results against the test data.

Another idea is that the number of inputs to the DBN is too low. The default dimensions of the "getting started" tutorial for the Theano DBN has 784 input samples. I altered this to a number that was close to the value of the outputs from the LSM, 256, but it might be the case that giving more information to the DBN would help it classify the examples.

5.1.10 Separability

The structure of the data might not be separable. If this were the case, the samples from the different classes would have values that are too close for the DBN to classify. This could be a result of the LSM having incorrect network dynamics or the creation of spike trains that were too similar. Separability issues may be caused by the EPOC head set. The EPOC might not have recorded the data well enough. This seems a bit unlikely as the initial testing of the data with WEKA showed some positive results. The multilayered perceptron (MLP) had a true positive rate of 70% for some of the direction classes.

For the mouse direction data finding the delta between two points might be suboptimal. The number of time points sampled might have to be tweaked. If the sample was too small, the variations in the mouse position might contain noise that does not reflect the user's intent. If the number of samples included in the averaging was too large, the intent might also be lost. In this case the user might have altered the trajectory of the mouse and the intent for the earlier or later direction might not reflect the user intent that drove the movement initially.

5.1.11 Overly Dynamic

In this case the LSM creates output that is too dissimilar between target classes. The samples of the liquid's state from the same class might not be similar enough to each other to be classified with the same label. This is due to the integration and dynamic nature of the system of the LSM. The internal cause would be that the complex interactions of the neuron's spiking patterns in the liquid would always create unique patterns. If these patterns are unique then they would be hard to classify.

5.1.12 Configuration

There are a large number of ways to configure a neural net and the DBN is no exception. In this case, the DBN is not configured correctly. It might not be able to classify the data with its set up. The DBN might be too small for the complexity of the data. These possibilities would have to be explored with experimentation. There are certainly ways to alter the configuration, such as varying the number of neurons in the hidden layers. It is also possible to change the number of hidden layers. Some of these alteration have been experimented with, as described in Chapter 4, Exploratory Experiment and Results. There might also be some value in varying the learning epochs, learning rates and patience levels for the DBN. Variations for the DBN is future work.

5.1.13 Classification Not Evident

It might be the case that some part of training of the classifier worked but is obscured in the Theano layer. The Theano DBN that was utilized did not report the results from every test. The results returned were only the average of the sets of tests done on the test_set. In cases where there were disproportional class representations it would appear that there was a higher validation error then might be present for specific classes. Once again if the worst classes were in the same mini blocks as better classes, they would increase the average validation error. The ability to examine the data closer would be beneficial to the process as a whole. Looking at the evaluation results within each minibatch, class by class, is what is needed. Examining each class would allow one to find out what the classification rates for the class was. If there were some good classification accuracies for specific classes, this would partially validate this approach. As it is, there is still insufficient data to confirm or deny the DBN's ability as a classification system. The modularity of the pipeline does allow us to look at the DBN in isolation and trouble shoot the errors that originate within it.

5.2 Comparison of Classifiers

The fact that there are possible errors throughout the entire pipeline make the overall analysis of many different classifiers and their interaction with the LSM impossible at this time. It might be possible in the near future to use different classifiers to help diagnose problems in the attempted system. For example, the results from WEKA would indicate that the EEG data contains at least 70% utilizable for some data. One test that should be run is the LSM data against the collection of WEKA's classifiers. The data from the LSM would have to be put into the *arff* file format for WEKA. Once in that format, it could be run in the WEKA classifiers. If WEKA classifiers were able to classify a high percentage of the data, it would likely be that the DBN needed tuning or reconsideration. If WEKA was unable to classify the same data it had previously classified at 70% true positive, the LSM could be reconfigured and retested.

5.3 Summary

The exploratory system created from an LSM and a DBN did not produce usable results. We have now covered many of the reasons why the test system could have preformed poorly. We have also looked at how the modularization within the architecture supports a systematic approach to troubleshooting and further exploration. Looking at the data pre-processing of the EEG data, there could be fault in the spike generation density. The factors that influenced the LSM are network dynamics, creation of the liquid, and timing issues in the simulation of the network. The DBN's poor classification might have been caused by mis-aligned training and testing examples from the LSM, or the configuration of the DBN itself. In the next and last chapter we will cover the conclusions we have drawn from working with the architecture and the test system, we will also talk about what might be done in the next iteration of research for this system.

Chapter 6

Conclusions

The idea of creating a system that generates an input for a computer using the mind is very enticing. There are varying difficulties in extracting information about the intent of a user. These difficulties range from user discomfort to the risk of death. Discomfort with electroencephalography (EEG) because the best results require the user to be still for the duration of the recording. Possibly deadly infections of the brain can arise from intracranial technologies. Due to EEG's combination of temporal, spectral and spatial resolution, its non-invasiveness, relative ease of use, cost and availability, it is an acceptable system for simple brain-computer interfaces (BCI). The ways in which natural nervous systems (NNS) work to solve problems for animals is extremely varied but inspiringly powerful. One of the ways NNS add power to their processing ability is to have circuits that feed back upon themselves. In the cortex, this gives the ability to alter the filtering of the data as they come into the organism. The filtering can allow the organism to adapt its behaviour to its environmental context more rapidly.

I did not get the chance to experiment with learning synapses in a fully recurrent artificial neural network (ANN) but I did start using one. The use of PCSIM's liquid state machine (LSM) is a form of reservoir computing system that allow for recurrent connections in the ANN. The input, in the form of 14 channels of EEG data, fed into the LSM allows for the incorporation of the previous temporal information into the current state of the liquid. This would allow for past information to propagate into the future state of the liquid and add information for classifiers to differentiate.

The sampled output from the LSM was then stratified in a 5:1:1 ratio for training, validation and testing sets. This created a training set for a deep belief network (DBN). The DBN utilized a graphics processing unit (GPU) enabled system to learn the set's features in a reasonable time. The filters created by the DBN can be used in real time against the data to classify the output from the EEG. This allows for the theoretical prediction of computer input from EEG data. With the envisioned system there is still some work to be done to get both real time interactivity and classification of the data. It is hoped that some more thought and some tweaking of the system will help to resolve these issues and allow any human mind to interact with the computer with thought alone.

The goal of this thesis was to investigate whether off-the-shelf tools could be used to create a brain computer interface for input into the computer.

6.1 Data Collection & Preparation

The data were collected as if in a normal office environment. This is a good start for a system like this. Noise from electrical systems would have to be dealt with by users while they work in their typical setting. The data were collected in real time and were passed on to a set of subsequent programs. As long as the total time for feedback is less then 250 ms, the human brain will most likely be able to incorporate it easily. The programs that prepare and format the information are quick: they can process the entire hour's worth of data in a second or two. The fact that they can process the information so quickly means that the 250 ms threshold should be easy to stay below.

6.2 Real Time

The directional processor and the spike train generator could be combined into one program. They could even be linked right into the recording program. There are application programming interfaces (API) for retrieving data from the Emotiv EPOC in Python. The APIs are not created by Emotiv but are open source. If the EEG recording code was all written in Python, it would be able to be linked from the recorder to the DBN. Converting the system into one process would definitely save memory and time loading and saving. Due to the LSM's processing time, the system would still be slower than real time.

The virtual machine that ran the LSM code only used a single thread on a single processor. The creation of more threads on other processors would immediately take the time down. On the host machine, for example, it could be 8 times faster. Instead of the time it takes for one second of time being processed in 22 seconds it would be processed in 2.75 seconds. With more time to play with the distribution of the neurons in the multithreaded multicore local machine, I would expect a slightly faster time. With more than one machine or a machine with more cores such as some of the machines at WestGrid one could parallelize the program more. It might be possible to get the LSM to process in real time with a machine with 88 cores or some other cluster running the simulation.

If the virtual box can not utilize the full number of processing cycles of the current computer, I would expect the native installation of the PCSIM software to run faster as well. This needs to be investigated.

Another interesting idea would be to see how one could use a GPU for running the neurons of the LSM. Perhaps it could be done with the matrix of neural states all in the memory of the GPU. The data would be streamed into the GPU and input to the internal states of the neurons in the GPU memory. From there the output would be sampled and streamed back out or shared with Theano. The newest Nvidia cards like the Titan Z have the same memory as the host computer we used to run the LSM on a virtual box [9, 7, 16, 2]. That 12 Gb of memory on the card is greater than the guest system at 8 Gb. Overcoming the conversions to CUDA capable code might be difficult, but Theano could help with that too.

6.3 NeuCube

The NeuCube is a spiking neural network (SNN); a new type of neural net created for the EvoSpike project. The initial structure of SNN has a large role to play in how it can process the data it receives [51]. It is a reservoir computing system. The set up of SNN in the NeuCube is created from connectivity patterns in the mammalian brain. Its overall 3D architecture is built to approximate the mammalian brain. NeuCube's SNN can evolve, with the addition of new connections and new neurons. It also evolves its synaptic weights with the alteration of the leaky integrate and fire neurons. It uses unsupervised learning with the STDP technique [14]. The gene regulatory network (GRN) can modulate the learning. The GRN can regulate the expression of the four types of synapses in the NeuCube. It takes into account both fast and slow excitatory and inhibitory synapses [42]. "The human brain has the amazing capacity to learn and recall patterns that occur at different time scales, ranging from milliseconds, to - years and possibly to millions of years (e.g. genetic information, accumulated through evolution). Thus the brain can be considered the ultimate inspiration for the development of new machine learning techniques for STBD" pg 64 [42].

6.4 Analysis

The results of the exploration of this topic are null results. If the null hypothesis was that it is not possible to create a brain-computer input device with off-the-shelf components, then the results support the null hypothesis. The system of a commercial EEG recorder, spike train generator, liquid state machine, and finally deep belief network, as they were, could not accomplish the task. That is to say that the initial setting and attempts seem to indicate this. It may be the case the this type of system is fundamentally flawed and would never work. The investigations needed to prove there is a fundamental flaw are far from done and are explored more in the future work.

6.5 Future Work

One thing to be done is the testing of other classifiers. Specifically for me is the MLP in WEKA, tested with the LSM. An interesting experiment would be to use fast Fourier transform (FFT) to format the input, more like nature would. The LSM has lots of potential tests: testing more of the internal parameters, classes based on biological counterparts, aspects of the LSM's random creation, saving and reusing the LSM and perhaps the most useful for future work and testing would be the classification of an astable multivibrator. For the DBN, it would be nice to do more tweaking, test more data, save filters to see how various LSM instances compare to one another and have the ability to print more of the intermediary and final results. Other classifiers might be useful to classify all or part of the EEG data. Alternative pipelines can been seen in Figure 6.1.

6.5.1 Alternative Classifier

One way to test the system is with data that has already been tested. For this I would take the data from the LSM and use WEKA's MLP. With the same setup of the MLP, but instead of the preceding second's data that I had used, I would input the LSM output. The MLP in WEKA was able to get a true positive of 70%. I would

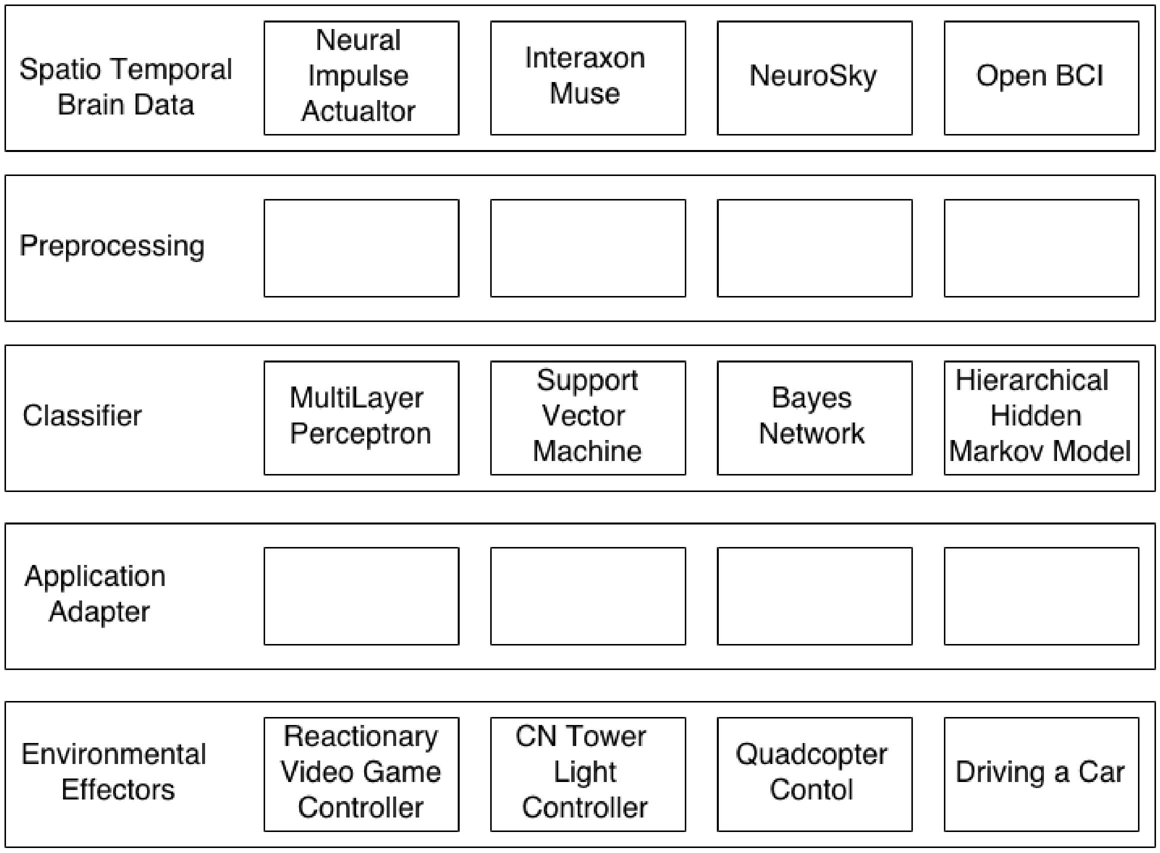


Figure 6.1: Possible alterations to the components of the pipeline. Changing the EEG input device is shown here. One could use an input device that is not an EEG, such as a microelectrode array. There are also variation in the classifiers and effectors that could be altered. (Drawn by Jason Cummer)

hope that the positive results from the multilayered perceptron (MLP) would mean that it is possible to classify the data. This could eliminate earlier steps of processing as sources of error.

6.5.2 EEG Substitution

The use of a commodity EEG system has been posited as a primary concern. If the system has unclassifiable data entered into the start of the pipeline, then there will be no classification possible at the end of the pipeline. To overcome the potential errors of the EEG, a known signal should be tested. A known signal could come out of a signal generator. The square wave, triangle wave and sine wave should all be able to be converted to spike trains and used in the pipeline. The DBN could be trained to test the value of the wave. If the results came back as having low validation error, one would know that the EEG data are not classifiable. Investigation into the reason why could be pursued, or an entirely different set of data could be explored.

6.5.3 Pre-processing

To better simulate the brain, one might use frequency mapped data for each channel. This would be more like the cochlea of many animals. The cochlea has an ability to physically separate sonic vibrations based on receptor position on the basilar membrane. A similar effect can be replicated with an FFT. The FFT data from each channel should then be used as input for the LSM. This would increase the number of inputs that the LSM would have, as there are bins for frequencies from the FFT. The number of bins is at the programmers discretion. It is estimated that there are about 3000 frequency bands that exist in the human cochlea [18]. There is likely no need to use this number of frequency bands. When you look at an FFT you can see the amount of energy in the waves that make up a frequency band. The energy in the frequency bands are often referred to as power. Most of the power from EEG is from 0 to 30 Hz, rather then the full range of 20 Hz to 20 000 Hz for optimal human hearing. One could create a different conversion algorithm to analyze the variation in the EEG signal. The Emotiv EPOC captures data about the electric fields as unsigned integers representing microVolts. The EEG values captured have a variation of +/-100 μV from a central point in the data. This variation is approximately 2.5% of the maximum value of 8000 μV used for normalization.

6.5.4 LSM

Try to use the StaticAnalogSynapse for the testing as it should be easily switched in, according to the experts of PCSIM. This would require some thought on the data inputs, but I think all that would be needed is to transpose the input arrays for the EEG and normalize them. This alteration might the fastest to find out if the LSM processing was at fault.

Use spikeTimeRecorder neurons to chart the overall network dynamics. With the data from the charts one could see if some tuning was required.

Try different ratios of excitatory to inhibitory neuron in the liquid and different types of populations of neurons rather then the SpatialFamilyPopulation. Could a training algorithm be created to help deal with these problems based on the recorded spikes?

Evaluate the LSM not just at the time of transition of the event, but include the times before. The fact that an event is coming can be told from the brain one second before the event, such as an arm movement. This has been shown [54].

There is a need to analyze multiple LSMs, as they are all randomly created with the default parameters. The random creation might not be a problem if the settings of the LSM are right. With the right set up you should get LSMs that are able to differentiate different inputs into different liquid states. This being said, there must be some LSMs created that perform better then other LSMs. If there was a set of parameters that was known to produce better machines, it would save a lot of time testing at random every time you needed something similar. Perhaps the parameters could be derived from experiments in which high performing machines were classified and analyzed deeper.

Another beneficial action would be to know if the network object can be pickled or saved in some way. This way the network created for the first file would be utilizable for subsequent files and perhaps realtime processors. Saving and reloading randomly created liquids, would allow the EEG data to be all be processed on the same network. This prevent outputs from different EEG files from coming from different temporal processing / compression. This would allow users to create larger data sets. The readings from many EEG files could be collected into one data set and run through a classifier. The classification results might be better if the problem was that the classifier needed more data.

One of the techniques used for studying neural ensembles is to build a neural drop off curve (NDC). An NDC allows a scientist to know how many neurons are important for a particular representation to transmit through a neural ensemble. Building NDCs from LSMs or other newer reservoir computing systems would allow for the reduction in size of the liquid. If one knows the size of the liquid needed for a problem, they can reduce or increase the size to optimize for their needs. Hopefully, smaller machines would come out of the analysis, as this would allow for the LSM to run faster.

A few other parameters to investigate would be the following: vary the numbers of neurons and their types in the liquid. There are a number of synapse models built into the PCSIM framework. Some types of synapses could provide better separation of states, while other types would allow learning within the LSM's set of neurons. The learning occurs in the weights between the neurons. Investigate the influence of more recording neurons on external classifiers. The ability of certain classifiers varies with particular parameters, so altering the number of neurons in the output neurons could mitigate some problems.

Another thing to build and test would be a simple LSM for testing classifiers to see if they could distinguish a binary data set. To produce the data set a simple astable multivibrator could be built. An input file or neurons could be built that would alternate at a given frequency a given number of times. Before the switch of input state, the liquid would be sampled and the samples saved and formatted to be passed on to a classifier. This would show whether the classifier could handle the data or if the outputs needed some alteration to maximize performance.

6.5.5 DBN

Further investigations would include more alterations with the structure of the DBN layers, number of neurons, various constants and training epochs. A genetic algorithm would allow the testing of many of the variables so that the resulting DBN is the one with the best suited design.

One of the test sets should have all the classes saved in it. One way to do this would run PCSIM on a large, faster computer cluster so that it could process many of the files with the same liquid.

Having sufficient training examples would allow better training. As stated earlier, MNIST had 6000 samples for each class whereas I only had about 700 samples for the best classes.

The filters from the trained DBN should be saved. This would allow testing of LSM outputs against each other, to see if they are similar enough for inputs from one LSM to be used with another. This would allow one to observe just how random are the LSMs. The saved filters could also be used for a real time system. When many filters had been saved, the best ones could be employed, based on the results of the classification. Printing out the results from each test class would allow for more detailed analysis. A breakdown of classification accuracies would show which classes need more training examples to be collected. This might also show some insight into the way the system collects data from the brain.

6.5.6 General Classifiers

Investigating other classifiers for the LSM output would be interesting. It might be the case that the classes of computer input (key or mouse events) are best analyzed with a specific classifier that is not a DBN. Other examples are the perceptron, Support Vector Machine, binary tree, and random forest. A lot more research and thought should go into the design of this idea to see if it is even a potential direction.

6.6 Future Computation Platforms

IBM has recently come out with a new processor. It is designed to be more like the brain. Instead of using Von Neumann architecture, it integrates the computation and memory to build a low power, event driven microprocessor [52]. Chips of this design are scaleable and could process the data quick and efficiently. This chip would be an ideal candidate to replace the LSM. It would be great to investigate the potential of this neuromorphic chip. It would both increase allow realtime processing and increase the capabilities of a BCI.

6.7 Final Thoughts

The exploratory system I envisioned and worked towards requires more work and has not shown initial promise. Along the way however, I was able to establish the importance of a modularization that supports both future innovation and trouble shooting better then a monolithic system would. The off-the-shelf components that were used in this particular pipeline mostly used Python as a programming language, so development was not too difficult. The components of the exploratory system are all separate at this stage. Integrating the components would decrease the time it takes for information to flow through the system. This would make it closer to real time, but compromise modularity. The modularity will allow researchers to work on the individual components of the system. One final point on the modularity is that it would allow myself or other researchers to try substituting various components in the pipeline. For example, one could remove the LSM and try an echo state network, then test the pipeline with a support vector machine. It is hopeful that all the technical difficulties can be overcome and that individuals with upper spinal cord injuries, amyotrophic lateral sclerosis or any interested individual can use BCIs to influence their environment and free their mind of its biologic boundaries.

Appendix A

Additional Information

A.1 Computers:

A.1.1 Wardenclyffe

Pentium I7 CPU, 920@2.67GHz

Intel processor identification utility:
Intel(R) Processor Identification Utility
Version: 5.00.20140627
Time Stamp: 2014/10/11 18:29:28
Operating System: 6.1-7601-Service Pack 1
Number of processors in system: 1
Current processor: #1
Active cores per processor: 4
Disabled cores per processor: 0
Processor Name: Intel(R) Core(TM) i7 920 CPU @ 2.67GHz
Type: 0
Family: 6
Model: 1A
Stepping: 5
Revision: 11
Maximum CPUID Level: B
L1 Instruction Cache: 4 x 32 KB
L1 Data Cache: 4 x 32 KB

L2 Cache: 4 x 256 KB
L3 Cache: 8 MB
Packaging: LGA1366
Enhanced Intel SpeedStep(R) Technology: Yes
MMX(TM): Yes
Intel(R) SSE: Yes
Intel(R) SSE2: Yes
Intel(R) SSE3: Yes
Intel(R) SSE4: Yes
Intel(R) AES-NI: No
Intel(R) AVX: No
Enhanced Halt State: Yes
Execute Disable Bit: Yes
Intel(R) Hyper-Threading Technology: Yes
Intel(R) 64 Architecture: Yes
Intel(R) Virtualization Technology: Yes
Intel(R) VT-x with Extended Page Tables: Yes
Expected Processor Frequency: 2.67 GHz
Reported Processor Frequency: 2.80 GHz
Expected System Bus Frequency: 133 MHz
Reported System Bus Frequency: 133 MHz
Expected QuickPath Interconnect Speed: 4.80 GT/s
Reported QuickPath Interconnect Speed: 4.80 GT/s
Expected Integrated Memory Controller Frequency: 1066 MHz
Reported Integrated Memory Controller Frequency: 1066 MHz

System Memory 12Gb
Windows 7 64 bit, service pack 1

A.1.2 Beast

CPU = Intel Core i7-2600 3.40 GHz
Memory = 4 GB
Linux Ubuntu 14.04

A.1.3 Oricle VM Virtualbox: Knoppix with PCSIM

Knoppix = special setup that includes the compiled PCSIM
CPU triple core
Memory 8 GB

Setting up guest additions (makes it easier to use) The steps I used to set it up:

1. Run your VM
2. Go to device in the VM window
3. Install guest additions *
4. You might need to add your profile to sub
5. vi sudoers or visudo **
6. Ctrl i ***
7. Under, root ALL=(ALL) ALL
8. Add <username> ALL=(ALL) ALL
9. Save and exit
10. Open the cd drive with the ISO of guest additions
11. Copy the VBoxLinuxAdditions.run to the Desktop
12. Navigate to the Desktop
13. Type : ./VBoxLinuxAdditions.run
14. It should install
15. Turn off the VM
16. In the system that you're working on you need to navigate to where you have the Oricle Virtualbox software installed.

17. Then you need to run this command:
    VBoxManage setextradata global GUI/MaxGuestResolution >width,height<
18. Where the width and height are your screen resolution. Supposedly, if you set the max resolution higher than your monitor, it will resize accordingly

* A video for this: https://www.youtube.com/watch?v=mBAbcwsKog8
** Adding yourself to su
http://askubuntu.com/questions/124166/how-do-i-add-myself-into-the-sudoers-group
*** This is to insert text in VI. I would suggest finding a list of basic VI commands.

A.2 Emotiv EPOC

A.2.1 EPOC Properties

Property	**EEG Headset**
Number of channels	14 (plus CMS/DRL references, P3/P4 locations)
Channel names (International 1020 locations)	AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, P3 (CMS), P4 (DRL), F4, F8, AF4
Sampling method	Sequential sampling. Single ADC
Sampling rate	128Hz SPS 2048$Hzinternal$
Resolution	14 bits 1 LSB $\bar{0}$.51 V (16 bit ADC, 2 bits instrumental noise floor discarded)
Bandwidth	0.2 45Hz, digital notch
Filtering	0.2 45hz, digital notch filters at 50Hz and 60Hz
Dynamic range (input referred)	256 V *pp*
Coupling mode	AC coupled
Connectivity	Proprietary wireless, 2.4GHz band
Power	LiPoly
Battery life (typical)	12 hours
Impedance Measurement	Realtime contact quality using patented system

A.2.2 Emotiv EPOC Use Procedure

The procedure to use the Emotiv EPOC is as follows. Insert electrodes by twisting them into their slots. Wet the felts on the electrodes with a saline solution of concentration 0.154 mols per litre. A common solution is contact solution or saline solution which is commercially available. Once the electrodes are wetted, open the Emotiv control panel program. In the headset setup tab there is an image of a superior view of the human head. There are a sixteen circles on the view that will be red. The red dots indicate the state of the electrodes and the connection. Black denotes no signal, red is a very poor connection, orange is a poor connection, yellow is fair and green means that the connection is good and the signals from the brain will be strong. Place the EPOC on the head and fit it snugly with the reference electrode on the temporal bone behind the ear. The solution in the felt pads on the electrodes improves contact with the scalp and allow for better conduction of the electric field to the electrode. Sometimes the hair of the user gets in the way and due to capillary action on the hairs surface the water moves from the electrode along the hair. If this happens it can be necessary to replace the solution in the electrodes. The refilling of the liquid in the felt pads might have to be done a few times. If attempted more then 5 times, check the felt pads and their connection to the EPOC. Make sure the electrodes are not corroded and that they make proper electrical contact. The user or technician can tell if the headset has a proper connection to the scalp with the Emotiv control panel. When the circles turn green then the EEG signals can be received.

A.2.3 Sample of Pre-processed Mouse Movement Data

x Delta	x(0)	y(0)	y Delta	count(1)	Direction	128ths	seconds
-2881	2881	928	-928	64	e	64	0.5
0	2881	931	-3	448	s	448	3.5
0	2881	933	-2	32	s	32	0.25
0	2881	937	-4	16	s	16	0.13
-99	2980	1084	-147	16	w	16	0.13
-125	3105	1111	-27	16	e	16	0.13
-11	3116	1099	12	16	ne	16	0.13
32	3084	1069	30	16	nw	16	0.13
12	3072	1062	7	16	nw	16	0.13
37	3035	1060	2	64	w	64	0.5
-201	3236	1056	4	16	e	16	0.13
54	3182	1063	-7	16	w	16	0.13
132	3050	1067	-4	16	w	16	0.13
0	3050	1068	-1	16	s	16	0.13
13	3037	1085	-17	16	sw	16	0.13
1	3036	1086	-1	17	sw	17	0.13

Continued on next page

Table A.1 – *Continued from previous page*

x Delta	x(0)	y(0)	y Delta	count(1)	Direction	128ths	seconds
-75	3111	1086	0	16	e	16	0.13
-61	3172	1087	-1	116	e	116	0.91
81	3091	1114	-27	15	w	15	0.12
71	3020	1130	-16	15	w	15	0.12
-127	3147	1130	0	31	e	31	0.24
-8	3155	1129	1	16	e	16	0.13
103	3052	1148	-19	15	w	15	0.12
-181	3233	1140	8	16	e	16	0.13
-17	3250	1140	0	12	e	12	0.09
0	3250	1141	-1	20	s	20	0.16
1	3249	1126	15	281	n	281	2.2
14	3235	1164	-38	4	s	4	0.03
2	3233	1177	-13	16	s	16	0.13
-12	3245	1188	-11	16	se	16	0.13
-7	3252	1197	-9	4	se	4	0.03
0	3252	1198	-1	12	s	12	0.09
0	3252	1201	-3	12	s	12	0.09
0	3252	1200	1	36	n	36	0.28
-1	3253	1200	0	4	e	4	0.03
17	3236	1148	52	12	n	12	0.09
5	3231	1059	89	32	n	32	0.25
-17	3248	1093	-34	16	se	16	0.13
-17	3265	1135	-42	4	s	4	0.03
35	3230	1136	-1	12	w	12	0.09
222	3008	1070	66	4	w	4	0.03
135	2873	1027	43	12	w	12	0.09
95	2778	975	52	12	nw	12	0.09
49	2729	960	15	4	w	4	0.03
3	2726	960	0	16	w	16	0.13
-113	2839	993	-33	15	e	15	0.12
-11	2850	993	0	1	e	1	0.01

Table A.1: Example of mouse direction for WEKA pre-processing

A.2.4 WEKA Results

Correctly classified Instances (53) 70.6667%

Incorrectly classified instances (22) 29.3333%

Kappa statistic 0.5271

Mean absolute error 0.0793

Root mean squared error 0.2411

Relative absolute error 54.6544 %

Root relative squared error 91.4514 %

Total Number of Instances (75)

Ignored Class Unknown Instances 3445

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC	Class
	0.667	0.063	0.667	0.667	0.667	0.83	North
	0	0	0	0	0	?	Northeast
	0	0	0	0	0	?	East
	0	0	0	0	0	?	Southeast
	0	0	0	0	0	?	South
	0	0	0	0	0	?	Southwest
	0.758	0.214	0.735	0.758	0.746	0.673	West
	0.667	0.2	0.69	0.667	0.678	0.665	Northwest
	0	0	0	0	0	?	?
Weighted Avg.	0.707	0.184	0.706	0.707	0.706	0.695	

Table A.2: WEKA Results for the mlp_raw16_norm_01_21_14 file.

A.3 Preprocessing Information

A.3.1 MinMax EEG from Five Files, Multiple Channels

```
min: 3838, max: 4912
min: 3605, max: 7519
min: 3921, max: 4722
min: 3972, max: 4934
min: 3639, max: 4503
min: 4191, max: 4418
min: 3762, max: 4433
min: 3768, max: 4412
min: 4183, max: 5164
min: 3933, max: 4708
min: 3815, max: 4626
min: 3953, max: 4800
min: 3681, max: 5828
min: 3434, max: 5815
min: 1325, max: 2076
min: 1468, max: 1912
```

A.3.2 MinMax All Files, All Channels

```
min: 0, max: 65431
min: 0, max: 161061
min: 0, max: 8401
min: 0, max: 107374
min: 0, max: 209720
min: 0, max: 137575
min: 0, max: 172821
min: 0, max: 157723
min: 0, max: 110732
min: 0, max: 179518
min: 0, max: 82210
```

min: 0, max: 189599
min: 0, max: 164433
min: 0, max: 203007
min: 0, max: 137575
min: 0, max: 137575

Original file for LSM The Liquid State Machine that I built was created from the file in the following link. `http://www.lsm.tugraz.at/pcsim/examples/spatial_inh_exc_populations.py`

Data Storage My data are stored on Beast. The data are also stored on a personal external hard drive.

A.4 Liquid State Machines & PCSIM

PCSIM is found here: http://www.lsm.tugraz.at/pcsim/
The Knoppix distro with PCSIM is here:
http://www.lsm.tugraz.at/pcsim/download/KnoppixPCSim.iso
The versions of PCSIM at Sourceforge: http://sourceforge.net/projects/pcsim/

A.5 Liquid State Machine

Number of neurons 4000
Number of recording neurons 256
Number of analog recording neurons was two hundred and fifty six.
network setup
time delay 1e-3
single threaded
ssimulationRNGSeed = 345678
constructionRNGSeed = 349871

Liquid excitatory neurons LifNeuron
Cm Normal Distribution 2e-10, 1e-11
Rm normal distribution 1e8 , 5e6

Vthresh, ConstantNumber(-50e-3) voltage threshold for membrane depolarization
Vresting -49e-3, resting potential of the membrane
Vrest -60e-3
H

Trefract , UniformDistribution(4.8e-3, 5.2e-3)
Vinit -60e-3

Liquid Inhibitory Neuron LifNeuron
Cm Normal Distribution 2e10, 2e11
Rm normal distribution 1e8 , 5e6
Vthresh, ConstantNumber(-50e-3) voltage threshold for membrane depolarization
Vresting -49e-3, resting potential of the membrane
Vrest -57e-3
Trefract , UniformDistribution(4.8e-3, 5.2e-3)
Vinit -57e-3

SpatialFamilyPopulation
Ratio Based Family 4:1
CuboidInterGrid3D(20,20,10). That is to say there are 4000 neurons in the liquid.

Excitatory Liquid Synapses
type: staticSpikingSynapse (W=Wexc, tau = 5e-3, delay = 1e-3)
range: EuclideanDistanceRandomConnection(ConnP, 10), ConnP = connectivity probability = 0.02
Erev_exc $\bar{0}$
Vmean = -60e-3
Wexc = (Erev_exc - Vmean) * 0.27e-9

Inhibitory Liquid Synapses
type: staticSpikingSynapse (W=Wexc, tau = 5e-3, delay = 1e-3)
range: EuclideanDistanceRandomConnection(ConnP, 10), ConnP = connectivity probability = 0.02
"The connection probability is C $exp(-|x-y|^2 lambda^2)$ where $|x-y|$ is the Euclidian distance between the SimObject's x and y."
`http://www.lsm.tugraz.at/pcsim/pyclassreference/html/pypcsim.`

`Euclidean\\DistanceRandomConnections-class.html`
Erev_inh = -80e-3
Vmean = -60e-3
Winh = (Erev_inh - Vmean) * 4.5e-9

SpikingInputNeurons
Input spike trains generated from the EEG Data.
Two hundred and fifty SpikingInputNeuron that generate spikes as output. These neurons connect to random excitatory neurons in the liquid.
static spiking synapse ($w = (Erev_exc - Vmean) * 2e9$, tau = 5e-3, delay = 1e-3)
randomConnections inpConnP = 0.01

SpikeTime Recording Neurons
net.setDistributionStrategy(DistributionStrategy.ModuloOverLocalEngines())
spike_rec_popul = SimObjectPopulation(net, SpikeTimeRecorder(), all_nrn_popul.size())
rec_conn_project = ConnectionsProjection(all_nrn_popul, spike_rec_popul, Time.ms(1)
);

Voltage Recording Neurons

```
vm_rec_nrn_popul = SimObjectPopulation(net,
                    random.sample( all_nrn_popul.idVector(),
                      nRecordNeurons ));
vm_recorders_popul = SimObjectPopulation(net, AnalogRecorder(),
                          nRecordNeurons );
for i in range(vm_recorders_popul.size()):
    net.connect( vm_rec_nrn_popul[i], 'Vm',
                  vm_recorders_popul[i], 0, Time.ms(1) );
```

A.6 Theano Deep Learning

Theano is found here: `http://deeplearning.net/software/theano/`
Theano at Github: `https://github.com/Theano/Theano`
Theano Deeplearning tutorial: `http://www.deeplearning.net/tutorial/`

Theano Deep Belief Network: `http://deeplearning.net/tutorial/code/DBN.py`
Theano Logistic regression node tutorial: `http://deeplearning.net/tutorial/logreg.html`
Theano Logistic regression node code: `http://deeplearning.net/tutorial/code/logistic_sgd.py`
Theano Multi Layer Perceptron tutorial: `http://deeplearning.net/tutorial/mlp.html`
Theano Multi Layer Perceptron code: `http://deeplearning.net/tutorial/code/mlp.py`
Theano Restricted Boltzmann MachineRBM tutorial:
`http://deeplearning.net/tutorial/rbm.html`
Theano Restricted Boltzmann MachineRBM Code: `\\http://deeplearning.net/tutorial/code/rbm.py`
Theano utils.py: `http://deeplearning.net/tutorial/code/utils.py`

A.7 MNIST Format

```
train_set
(array([[0., 0., 0., ... , 0., 0., 0.],
  [0., ... , 0.],
   ...
  [0., ... , 0.]
  ],
  dtype=float32, array([5,0,4,   , 8,4,8]dtype=int64)
)

valid_set:
(array([[0., 0., 0., ... , 0., 0., 0. ],
  [0.,   , 0.],

  [0., ... , 0.]
 ],
 dtype=float32, array([3,8,6,   ,5,6,8]dtype=int64)
)

test_set:
(array([[0., 0., 0., ... , 0., 0., 0.],
  [0.,  , 0.],
  ...
  [0., ... , 0.]
 ],
 dtype=float32, array([7,2,1,   , 4,5,6]dtype=int64)
)
```

A.8 Deep Belief Network Setup

First Runs number of layers = 3
numpy_rng = 123 ;Random number generator
Theano_rng = none. Theano random generator; if None is given one is generated based on a seed drawn from 'rng'
n_ins = 256. Dimension of the input to the DBN
hidden_layers_sizes = [1000, 1000, 1000]. Intermediate layers size, must contain at least one value.
n_outs = 10

Ubuntu 12.04.5 (Precise Pangolin) is here: `http://releases.ubuntu.com/12.04/`
Later Runs numpy_rng = 123 ;Random number generator
Theano_rng = none. Theano random generator; if None is given one is generated based on a seed drawn from 'rng'
n_ins = 256. Dimension of the input to the DBN
hidden_layers_sizes = [1000, 1000, 1000]. Intermediate layers size, must contain at least one value.
Other test contained [100], [10000], [10000,1000] [10000,1000, 1000], [1000,1000,1000,1000]
n_outs = 153 0r 154

results: Validation error was never better then 99.6%.

A.9 Class Code List

A.10 Class Code Count Example File 18-13

Class Name	Count
Null	0
North	119
Northeast	0

Continued on next page

Table A.4 – *Continued from previous page*

Class Name	**Count**
East	265
Southeast	0
South	130
Southwest	0
West	221
Northwest	0
MouseLeft Button True	0
MouseLeft Button False	0
MouseRight Button True	0
MouseRight Button False	0
Tab True	0
Tab False	0
Shift True	5
Shift False	5
Ctrl True	1
Ctrl False	1
Alt True	0
Alt False	0
LeftWindows True	0
LeftWindows False	0
Apps True	0
Apps False	0
Numlock True	0
Numlock False	0
Key Codes	
# 1	0
# 2	0
# 3	0
# 4	0
# 5	0

Continued on next page

Table A.4 – *Continued from previous page*

Class Name	**Count**
# 6	0
# 7	0
# 8	0
# 9	2
# 10	0
# 11	0
# 12	0
# 13	0
# 14	5
# 15	0
# 16	0
# 17	0
# 18	0
# 19	0
# 20	0
# 21	0
# 22	0
# 23	0
# 24	0
# 25	0
# 26	0
# 27	0
# 28	0
# 29	0
# 30	0
# 31	0
# 32	0
# 33	1
# 34	0
# 35	0

Continued on next page

Table A.4 – *Continued from previous page*

Class Name	**Count**
# 36	0
# 37	0
# 38	1
# 39	0
# 40	0
# 41	0
# 42	0
# 43	0
# 44	0
# 45	0
# 46	0
# 47	0
# 48	0
# 49	0
# 50	0
# 51	0
# 52	0
# 53	0
# 54	0
# 55	0
# 56	0
# 57	0
# 58	0
# 59	0
# 60	0
# 61	0
# 62	0
# 63	0
# 64	0
# 65	0

Continued on next page

Table A.4 – *Continued from previous page*

Class Name	Count
# 66	2
# 67	3
# 68	3
# 69	4
# 70	6
# 71	0
# 72	6
# 73	4
# 74	5
# 75	0
# 76	1
# 77	6
# 78	6
# 79	0
# 80	6
# 81	4
# 82	0
# 83	6
# 84	2
# 85	4
# 86	4
# 87	5
# 88	5
# 89	0
# 90	4
# 91	0
# 92	0
# 93	0
# 94	0
# 95	0

Continued on next page

Table A.4 – *Continued from previous page*

Class Name	Count
# 96	0
# 97	0
# 98	0
# 99	1
# 100	4
# 101	4
# 102	1
# 103	0
# 104	0
# 105	0
# 106	0
# 107	0
# 108	0
# 109	0
# 110	0
# 111	0
# 112	0
# 113	0
# 114	0
# 115	0
# 116	0
# 117	0
# 118	0
# 119	0
# 120	0
# 121	0
# 122	0
# 123	0
# 124	0
# 125	0

Continued on next page

Table A.4 – *Continued from previous page*

Class Name	Count
# 126	0
# 127	0
# 128	0

Table A.4: The number of examples for each class of input. The classes included in the DBN input are floored to the nearest 7. This is so that the data are stratified in a ratio of 5:1:1. If less than seven, the class is not represented.

A.11 Key Code Constants

`http://msdn.microsoft.com/en-ca/library/aa243025%28v=vs.60%29.aspx`

A.12 DBN Results

`https://docs.google.com/spreadsheets/d/1uFa-1Bpvz0eUQTM2yiyT_AcuKeF0uW9N6nD1yA3iYNc/edit#gid=1967290946`

pre-training layer 2 now EPOC 57

Results
epoch 1, minibatch 274/274, validation error 100.000000 %

('this_validation_loss.shape: ', ())
epoch 1, minibatch 274/274, test error of best model 100.000000 %

('len(test_losses): ', 54)
[array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),

Class Name	**Value**
Null	None State
North	201
Northeast	202
East	203
Southeast	204
South	205
Southwest	206
West	207
Northwest	208
MouseLeft Button True	209
MouseLeft Button False	210
MouseRight Button True	211
MouseRight Button False	212
Tab True	213
Tab False	214
Shift True	215
Shift False	216
Ctrl True	217
Ctrl False	218
Alt True	219
Alt False	220
LeftWindows True	221
LeftWindows False	222
Apps True	223
Apps False	224
Numlock True	225
Numlock False	226
Key Codes	1 - 128

Table A.3: The list for mapping the class codes from the LSM and DBN data to actions

```
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0)]
epoch 2, minibatch 274/274, validation error 100.000000 %

('this_validation_loss.shape: ', ())
epoch 3, minibatch 274/274, validation error 100.000000 %

('this_validation_loss.shape: ', ())
epoch 4, minibatch 274/274, validation error 99.629630 %

('this_validation_loss.shape: ', ())
epoch 4, minibatch 274/274, test error of best model 99.629630 %

('len(test_losses): ', 54)
[array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(0.9), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(0.9), array(1.0), array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0), array(1.0),array(1.0),
array(1.0), array(1.0), array(1.0), array(1.0), array(1.0)]
Optimization complete with best validation score of 99.629630 %,with test perfor-
mance 99.629630 %
The fine tuning code for file JasonsDBN.py ran for 0.08m
```

MNIST DATA
```
epoch 127, minibatch 5000/5000, validation error 1.360000 %
Optimization complete with best validation score of 1.350000 %,with test performance
1.370000 %
The fine tuning code for file DBN.py ran for 46.85m
```

A.13 Beyond Boundaries

pg 107. Broadly tuned neurons are working in concert. There are no label lined neurons in the barrel cortex. Receptive fields in the ventral posterior medial (VPM) nucleus varied through time and were dependent on the reciprocal connection to the reticular nucleus Thalamocortical neurons TC)

pg 2, 151 With 96 neurons sampled, it was possible to reproduce the cursor movement very accurately in real time.

pg 139 wiener filter - essentially a perceptron just after the summing it isn't working on a threshold for firing its output. The output from a Weiner filter is continuous.

pg 165 Based on the work done on neural drop off curves by Nicolelis et all [54] about 60 neurons should give an accurate sample for a prediction of movement. It could be the case, though, that the information fed to the neural net should be at a sufficient level of abstraction for this to work reliably.

pg 168 Single neuron insufficiency Neural dropping curves (NDC): when you have approximately one hundred neurons that is a good number for a prediction from the network of their activity. As the number of neurons is 20 to 30, it degrades quickly.

Interesting cortical columns tend to be about 150 to 300 neurons as well.[44].

pg 172 Thought:
different random samples from a LSM should have different information density

pg 263 p4 "The way the cortex responds as a whole, to an incoming stimulus or the need to produce a particular motor behaviour, depends on the global internal state of the brain at that instant; That is, ongoing brain dynamics are essential in defining the optimal solution the brain derives for the generation of any given behaviour."

A.14 Glossary

DBN - Deep Belief Network - A type of artificial neural network, composed of both a multilayer neural network and pairs of layers built as Restricted Boltzmann Machines.

LSM - Liquid State Machine - There is a pool or collection of neurons/nodes that are connected, usually at random, that form a neural network. They are spiking neural nets. Input is fed generally into a random selection of neurons and output is read from a random selection of neurons. A LSM is a form of reservoir computing.

Over-stratification - A problem with an LSM. The pattern of firing in the liquid does not continue for long after an input is injected. The amount of time is not long enough to be of use for classification.

Pathological synchrony - When there are many loops that have positive feedback. This causes overstimulation of the liquid and thus the neurons fire continuously. Classification would be more difficult or impossible if neurons are always in an on state.

Reservoir computing - A type of computing that utilizes a dynamic system to generate its output. A dynamic system being one that generates its outputs from both the input and the time values.

Separation property - A measure of the Euclidean distance of an LSM's states of the network at different times.

SGD - Stochastic Gradient Descent - An algorithm that uses the derivative of an equation to alter the constants of said equation. The purpose of which is to minimize the error in the results from the equation and the data.

STDP - Spike Time Dependent Plasticity - A form of neural plasticity that depends on how close in time two neurons fire together. If neuron one fires close in time to an input of neuron two, the strength of the connection between the neurons one and two increases.

Train set - The set of data that is sampled in the learning stage of a machine learning model of a classification problem.

Test set - A set of data like the training data (train set) that has not been involved in the training of the classifier.

Validation set - A set of data used to test the model generated my a machine learning algorithm during initial training.

WEKA - A free data mining software package written in Java by the University of Waikato. The software includes clustering, classifiers and associations as well as tools for working with data.

Bibliography

[1] Benjamin Libet. http://www.informationphilosopher.com/solutions/scientists/libet/.

[2] CPU GPU and MIC Hardware Characteristics Over Time. http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/.

[3] Deeplearning 0.1 documentation : Getting started. http://www.deeplearning.net/tutorial/gettingstarted.html#gettingstarted.

[4] Deeplearning.net Theano Deep Belief Network. http://deeplearning.net/tutorial/code/DBN.py.

[5] emokit-java GitHub.

[6] Emotiv EPOC / EPOC+. https://emotiv.com/epoc.php.

[7] Geforce Graphics Cards. http://www.nvidia.ca/object/geforce_family.html.

[8] Human Connectome Project. http://www.humanconnectomeproject.org/.

[9] List of Nvidia Graphics Cards on Wikipedia. http://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units.

[10] pickle-Python Object Serialization, howpublished=https://docs.python.org/2/library/pickle.html, url-https://docs.python.org/2/library/pickle.html.

[11] Rain's Ripple Rings. http://www.words4it.com/?tag=raindrops.

[12] SimNetwork.h from PCSIM. http://www.lsm.tugraz.at/pcsim/cppclassreference/html/classSimNetwork.html#d2e5009b3b61bbe348ada57550c7fdb6.

[13] Simple Neuron Diagram. www.wpclipart.com/medical/anatomy/cells/neuron/neuron.png.html.

[14] Spike Timing Dependent Plasticity. http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity.

[15] Theano - Negative Log Likelihood Loss. http://deeplearning.net/tutorial/gettingstarted.html#negative-log-likelihood-loss.

[16] What is the Difference Between GPU and CPU? http://allegroviva.com/gpu-computing/difference-between-gpu-and-cpu/.

[17] Wolfram Alpha Search for ”Average English Word Length”. http://www.wolframalpha.com/input/?i=average+english+word+length.

[18] Reverse-Engineering the Human Auditory Pathway in Advances in Computational Intelligence. 7311:47–59, 2012.

[19] Lóska Ádám. Emotiv EPOC - EEG Based Brain-Computer Interface. May 2011.

[20] Frdric Alexandre, Frdric Guyot, Jean-Paul Haton, and Yves Burnod. The cortical column: A new processing unit for multilayered networks. *Neural Networks*, 4(1):15 – 25, 1991.

[21] Kimitaka Anami, Takeyuki Mori, Fumiko Tanaka, Yusuke Kawagoe, Jun Okamoto, Masaru Yarita, Takashi Ohnishi, Masato Yumoto, Hiroshi Matsuda, and Osamu Saitoh. Stepping Stone Sampling for Retrieving Artifact-Free Electroencephalogram During Functional Magnetic Resonance Imaging. *NeuroImage*, 19(2):281 – 295, 2003.

[22] Yoshua Bengio. Introduction to Gradient-Based Learning. http://www.iro.umontreal.ca/~pift6266/H10/notes/gradient.html, 2014. Last updated on Apr 02, 2010.

[23] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[24] C S Bjornsson, S J Oh, Y A Al-Kofahi, Y J Lim, K L Smith, J N Turner, S De, B Roysam, and S.J.Kim W Shain and. Effects of Insertion Conditions on Tissue Strain and Vascular Damage During Neuroprosthetic Device Insertion. *Journal of Neural Engineering*, 3(3):196, 2006.

[25] M. Botvinick. Probing the Neural Basis of Body Ownership. *Science*, 305(5685):782–783, AUG 6 2004. PT: J; UT: WOS:000223104900025.

[26] Gyoergy Buzsaki, Costas A. Anastassiou, and Christof Koch. The Origin of Extracellular Fields and Currents - EEG, ECoG, LFP and Spikes. *Nature Reviews Neuroscience*, 13(6):407–420, JUN 2012. PT: J; NR: 161; TC: 179; J9: NAT REV NEUROSCI; PG: 14; GA: 944JP; UT: WOS:000304197000010.

[27] Daniel c. Dennett. *Freedom Evolves.* New York: Viking, 2003.

[28] Jennifer L Collinger, Brian Wodlinger, John E Downey, Wei Wang, Elizabeth C Tyler-Kabara, Douglas J Weber, Angus JC McMorland, Meel Velliste, Michael L Boninger, and Andrew B Schwartz. High-performance neuroprosthetic control by an individual with tetraplegia. *The Lancet*, 381(9866):557 – 564, 2013.

[29] Guillaume P. Dugu, Walther Akemann, and Thomas Knpfel. Chapter 1 - A Comprehensive Concept of Optogenetics. *Progress in brain research*, 196(0):1–28, 2012.

[30] Emotiv. *Emotiv Brain Computer Interface Emotiv EPOC User Manual.* Emotiv, 2011.

[31] Emotiv. *Emotiv EPOC UserManual.* Emotiv, 2014.

[32] Dileep George and Jeff Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology*, 5(10):e1000532, 2009.

[33] Charles M. Gray, Pedro E. Maldonado, Mathew Wilson, and Bruce McNaughton. Tetrodes Markedly Improve the Reliability and Yield of Multiple Single-Unit Isolation from Multi-Unit Recordings in cat Striate Cortex. *Journal of Neuroscience Methods*, 63(12):43 – 54, 1995.

[34] Jeff Heaton. *Introduction to Neural Networks for C#, 2Nd Edition*. Heaton Research, Inc., 2nd edition, 2008.

[35] Michael A. Heymann, Bruce D. Payne, Julien I.E. Hoffman, and Abraham M. Rudolph. Blood flow measurements with radionuclide-labeled particles. *Progress in Cardiovascular Diseases*, 20(1):55 – 79, 1977.

[36] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[37] Daniel R. Hochbaum, Yongxin Zhao, Samouil L. Farhi, Nathan Klapoetke, Christopher A. Werley, Vikrant Kapoor, Peng Zou, Joel M. Kralj, Dougal Maclaurin, Niklas Smedemark-Margulies, Jessica L. Saulnier, Gabriella L. Boulting, Christoph Straub, Yong Ku Cho, Michael Melkonian, Gane Ka-Shu Wong, D. J. Harrison, Venkatesh N. Murthy, Bernardo L. Sabatini, Edward S. Boyden, Robert E. Campbell, and Adam E. Cohen. All-optical Electrophysiology in Mammalian Neurons using Engineered Microbial Rhodopsins. *Nat Meth*, 11(8):825–833, print 2014. M3: Article.

[38] Richard W Homan, John Herman, and Phillip Purdy. Cerebral Location of International 10–20 System Electrode Placement. *Electroencephalography and Clinical Neurophysiology*, 66(4):376 – 382, 1987.

[39] A. W.; McCarthy G. Huettel, S. A.; Song. *Functional magnetic resonance imaging, 2d ed*, volume 33. 06 2009. Copyright - Copyright Book News, Inc. Jun 2009; Last updated - 2010-06-06.

[40] E. Juergens, A. Guettler, and R. Eckhorn. Visual Stimulation Elicits Locked and Induced Gamma Oscillations in Monkey Intracortical- and EEG-Potentials, but not in Human EEG. *Experimental Brain Research*, 129(2):247–259, NOV

1999. PT: J; NR: 52; TC: 84; J9: EXP BRAIN RES; PG: 13; GA: 258JE; UT: WOS:000083835400010.

[41] Clare-Marie Karat, Christine Halverson, Daniel Horn, and John Karat. Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 568–575, New York, NY, USA, 1999. ACM.

[42] Nikola K. Kasabov. NeuCube: A Spiking Neural Network Architecture for Mapping, Learning and Understanding of Spatio-Temporal Brain Data. *Neural Networks*, 52:62–76, April 2014.

[43] Bryan Kolb and Ian Q Whishaw. An introduction to brain and behavior. 2001.

[44] Bryan Kolb and Ian Q Whishaw. *Fundamentals of human neuropsychology.* Macmillan, 2009.

[45] Ray Kurzweil. *How to Create a Mind: The Secret of Human Thought Revealed.* Viking Penguin, 2012.

[46] Rodrigo Laje and Dean V. Buonomano. Robust Timing and Motor Matterns by Taming Chaos in Recurrent Neural Networks. *Nature neuroscience*, 16(7):925–U196, JUL 2013. PT: J; NR: 55; TC: 14; J9: NAT NEUROSCI; PG: 12; GA: 174UM; UT: WOS:000321180900027.

[47] Alan D. Legatt, Joseph Arezzo, and Herbert G. Vaughan Jr. Averaged Multiple Unit Activity as an Estimate of Phasic Changes in Local Nuronal Activity: Effects of Volume-Conducted Potentials. *Journal of neuroscience methods*, 2(2):203–217, 4 1980.

[48] Benjamin Libet. *Mind Time: The Temporal Factor in Consciousness.* Harvard University Press, 2004.

[49] Mantas Lukoevicius and Herbert Jaeger. Reservoir Computing Approaches to Recurrent Neural Network Training. *Computer Science Review*, 3(3):127 – 149, 2009.

[50] Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witten Mark Hall, Eibe Frank. The WEKA Data Mining Software: An Update. Volume 11, Issue 1, 2009.

[51] Jonas Matser. Structured Liquids in Liquid State Machines. 2011.

[52] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D.S. Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1–4, Sept 2011.

[53] Microsoft. How to set a Windows Hook in Visual C# .NET. `http://support2.microsoft.com/default.aspx?scid=kb;en-us;318804#3`, 2014.

[54] Miguel Nicolelis. *Beyond Boundaries: The New Neuroscience of Connecting Brains with Machines—and How It Will Change Our Lives*. Times Books, 2011.

[55] Andre Palmini. The Concept of the Epileptogenic Zone: A Modern look at Penfield and Jasper's Views on the Role of Interictal Spikes. *Epileptic Disorders*, 8:S10–S15, AUG 2006. PT: J; NR: 30; TC: 10; J9: EPILEPTIC DISORD; SU: 2; PG: 6; GA: 096ZL; UT: WOS:000241415500002.

[56] PCSIMMain. Pcsim: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python. *Frontiers in Neuroinformatics*, 3:11, 04/21 2009. J1: Front Neuroinformatics.

[57] Charles Petzold. Programming Windows Fifth Edition, 1998.

[58] Vadim S. Polikov, Michelle L. Block, Jean-Marc Fellous, Jau-Shyong Hong, and W. Monty Reichert. In vitro model of glial scarring around neuroelectrodes chronically implanted in the {CNS}. *Biomaterials*, 27(31):5368 – 5376, 2006.

[59] Stefan Posse. Advances in Neuroscience and Clinical Research using Ultra - High Speed fMRI. 2014. OHBM 2014 Annual Metting, Hamberg, Germany.

[60] Shondra M. Pruett-Miller. Chapter 29 - genome editing in somatic cells using zinc finger nucleases and transcription activator-like effector nucleases. In Jos Cibelli, John Gurdon, Ian Wilmut, Rudolf Jaenisch, Robert Lanza, Michael D. West, and Keith H.S. Campbell, editors, *Principles of Cloning (Second Edition)*, pages 369 – 378. Academic Press, San Diego, second edition edition, 2014.

[61] Sandro Saitta. Stratification for data mining. `http://www.dataminingblog.com/stratification-for-data-mining/`, 2007.

[62] Wei Wang, Jennifer L. Collinger, Monica A. Perez, Elizabeth C. Tyler-Kabara, Leonardo G. Cohen, Niels Birbaumer, Steven W. Brose, Andrew B. Schwartz, Michael L. Boninger, and Douglas J. Weber. Neural interface technology for rehabilitation: Exploiting and promoting neuroplasticity. *Physical Medicine and Rehabilitation Clinics of North America*, 21(1):157 – 178, 2010. Quality of Life.

[63] Nikolaus Weiskopf, Ranganatha Sitaram, Oliver Josephs, Ralf Veit, Frank Scharnowski, Rainer Goebel, Niels Birbaumer, Ralf Deichmann, and Klaus Mathiak. Real-Time Functional Magnetic Resonance Imaging: Methods and Applications. *Magnetic resonance imaging*, 25(6):989–1003, 7 2007.

[64] D. F. Wulsin, J. R. Gupta, R. Mani, J. A. Blanco, and B. Litt. Modeling Electroencephalography Waveforms with Semi-Supervised Deep Belief Nets: Fast Classification and Anomaly Measurement. *JOURNAL OF NEURAL ENGINEERING*, 8(3), JUN 2011.