

Use of Artificial Neural Network for Land Use Land Cover Classification of UAV Acquired Imagery

THESIS SUBMITTED TO

Symbiosis Institute of Geoinformatics



FOR PARTIAL FULFILLMENT OF THE M. Sc. DEGREE

By

Marcia Chen
PRN14070241023

(Batch 2014-16)

Symbiosis Institute of Geoinformatics

Symbiosis International University

5th Floor, Atur Centre, Gokhale Cross Road,

Model Colony, Pune – 411016.

CERTIFICATE

Certified that this thesis titled '**Use of Artificial Neural Network for Land Use Land Cover Classification of UAV Acquired Imagery**' is a bonafide work done by **Miss Marcia Chen**, at Webonise Lab and Symbiosis Institute of Geoinformatics, under our supervision.

Supervisor Internal

Col. B. K. Pradhan,
Professor,
Symbiosis Institute of Geoinformatics,
Pune.

Supervisor External

Prateek Srivastava,
Director – Business Development,
PrecisionHawk,
Pune.

CONTENTS

I.	Acknowledgement.....	4
II.	List of Figures.....	5
III.	List Tables.....	6
IV.	Abbreviation list.....	7
V.	Preface.....	8
1.	Introduction.....	9
1.01	Need for a Solution.....	9
1.02	Objective.....	9
1.03	About Artificial Neural Network.....	10
1.04	Usefulness and Capabilities.....	10
2.	Literature Review.....	11
2.01	Machine Learning Through the Ages.....	11
2.02	History of Neural Networks.....	12
2.03	Analogy between Human and Artificial Neural Nets.....	13
2.04	Neural Network Model.....	15
2.05	Architecture of Artificial Neural Network.....	15
2.06	The Perceptron: The Fundamentals.....	16
2.07	Multi-Layer Perceptron (MLP).....	17
2.08	The Perceptron Learning Algorithm.....	20
2.09	Artificial Neural Networks and Gradient Descent Algorithm.....	21
2.010	Non Linear Activation Functions.....	23
2.011	Learning Strategies in Neural Network.....	25
2.012	Learning Mechanisms in Neural Network.....	25
2.013	Backpropagation Algorithm.....	32
2.014	Working with UAV acquired data.....	33
3.	Methodology.....	36
3.01	Outline.....	36
3.02	Modules.....	37
3.03	Software.....	38
3.04	Development.....	39
4.	Result.....	49
5.	Discussion.....	51
6.	Conclusion.....	52
7.	References.....	53
8.	Glossary.....	58

9. Appendix.....	60
9.01 Appendix-I: Data Used.....	60
9.02 Appendix-II: UAV Hardware Specifications.....	62

ACKNOWLEDGEMENT

As this project comes to a close, I would like to look back and express my gratitude towards all those people who helped me in my endeavor.

To begin with, I would like to thank my external supervisor and mentor, Mr. Prateek Srivastava, Director – Business Development at PrecisionHawk, who devoted his valuable time and wisdom in guiding me in this project. Without his expertise, this project would not have been the same.

I would also like to extend my gratitude towards Mr. Yogesh Mukhi, GIS and Remote Sensing Expert at PrecisionHawk, for providing his technical expertise and know-how in the same.

I appreciate the guidance provided to me by Mr. Anand Jadhav, Senior Technical Lead at PrecisionHawk, who gave his useful insights on data management and processing for this project.

Further, I would thank the faculty of Symbiosis Institute of Geoinformatics, Pune, namely Dr T. P. Singh, Dr Navendu Chowdhury and Col B. K. Pradhan, without whom my knowledge about GIS and its application in the various domains would not have been clear.

Lastly, I would like to thank my mother, Mrs. Roselind Chen, for always being there and providing moral support throughout.

LIST OF FIGURES

Figure 2.01: Machine Learning: Time Line (*Hu, 2013*)

Figure 2.02: Analogy between human neurons and artificial nets (*Vinícius Gonçalves Maltarollo, January 16, 2013*)

Figure 2.03: Network Layers (*Neural Networks, 2013*)

Figure 2.04: Feedforward ANN

Figure 2.05: Feedback ANN

Figure 2.06: A Perceptron

Figure 2.07: Single Layer Multilayer Perceptron

Figure 2.08: MLP with Two Hidden Layers (*Neural Network, 2016*)

Figure 2.09: Learning process in a parametric system (*Rojas, 1996*)

Figure 2.10: Global Minimum

Figure 2.11: Local and Global Maxima and Minima (*Wikipedia, Maxima and Minima, 2016*)

Figure 2.12: Illustration of Gradient Descent(*Gradient descent, 2016*)

Figure 2.13: Sigmoid Function (*Weisstein, 2016*)

Figure 2.14: Graphical Representation of the Tan Hyperbolic Function (*Weisstein, Hyperbolic Tangent, 2016*)

Figure 2.15: Illustration of Error Correction Learning (*Bennamoun*)

Figure 2.16: Outliers: Cause of Faulty Classification

Figure 2.17: Architecture of the competitive learning mechanism (*Zipser, 1985*).

Figure 2.18: Boltzmann Machine (*Boltzmann machine, 2016*)

Figure 2.19: PrecisionHawk Lancaster 5 (*Lancaster, 2016*)

Figure 2.20: PrecisionHawk Lancaster Rev 3 (*Reich, 2015*)

Figure 3.01: Multi Layer Perceptron for MLP

Figure 3.01: Work Flow of the Multi Layer Perceptron in MLP Tool

Figure 3.03: Plugin Builder

Figure 3.04: OSGeo4WShell

Figure 3.04: MLP Classifier Plugin: First Look

Figure 3.05: MLP Classifier: Non-Functional Window

Figure 3.06: Designing the UI using the QT Creator

Figure 3.07: The MLP Tool Interface

Figure 3.08: Graphical Representation of the Achieved Error Rate after Training

Figure 3.09: Green-Red Vegetation Index

Figure 3.10: Urban Index

Figure 4.01: Classification of UAV Acquired Dataset with the MLP Tool

Figure 4.02: Manually Classified LULC Map

Figure 4.03: Bar Graph showing the Pixel Count in 50 Iterations, 100 Iterations and Manually Classified LULC Map

Figure 9.01: Orthomosaic of the Area

Figure 9.02: Number of overlapping images computed for each pixel of the orthomosaic.

LIST OF TABLES

Table 2.01: An Analogy between Human and Artificial Neurons

Table 3.01: Ranges of Classes

Table 9.01: XYZ Accuracy of the GCPs

Table 9.02: Hardware specifications of PrecisionHawk Lancaster Rev 3

ABBREVIATION LIST

AI	Artificial Intelligence
ANN	Artificial Neural Network
GRVI	Green Red Vegetation Index
GUI	Graphical User Interface
LULC	Land Use Land Cover
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
SLP	Single Layer Perceptron
SVM	Support Vector Machine
UAV	Unmanned Aerial Vehicle
UI	User Interface

PREFACE

Classification of remotely sensed data has been widely used to generate thematic Land-Use Land-Cover (LULC) inventories for a range of applications including urban planning, agricultural crop characterization, and forest ecosystem classification. In response, a number of different classification approaches have been developed to accomplish such tasks. Most notable have been classification approaches based on Artificial Neural Networks (ANNs). ANNs were originally designed as pattern-recognition and data analysis tools that mimic the neural storage and analytical operations of the brain. ANN approaches have a distinct advantage over statistical classification methods in that they are non-parametric and require little or no *a priori* knowledge of the distribution model of input data. Additional superior advantages of ANNs include parallel computation, the ability to estimate the non-linear relationship between the input data and desired outputs, and fast generalization capability. Many previous studies on the classification of images have confirmed that ANNs perform better than traditional classification methods in terms of classification accuracy, such as maximum likelihood classifiers.

Thus, this paper will focus on applying neural network machine learning methods to UAV acquired images for the purpose of automatic detection and classification. The aim is to formulate a tool which will produce LULC maps with marginal errors.

INTRODUCTION

With large number of data sources coming into picture today, we have been blessed with a rich and varied repository for almost every part of the Earth. However, due to its surmounting quantity and size, we may sometimes be unable to smartly process and utilize them. Land use Land cover (LULC) Classification is one such output which we have not been able to automatically produce yet. Hence, the need for an innovative tool that can enable us to do so.

1.01 Need for a Solution

At large, it has been seen, that most organizations perform and produce Land Use Land Cover (LULC) maps manually. Although manual classification is considered more efficient as compared to machine classification due to the ability of humans for superior pattern recognition and categorization (*Prasad S. Thenkabail, 2015*), it surely could pose some serious drawbacks, namely:

- **Time consuming:** At a time where everything is fast paced and there is a demand for good quality at a reasonable time, classifying data manually can be very time consuming.
- **No guarantee of reproducibility:** Since manual classification relies largely on the interpretability of an individual, it may vary from person to person, providing no assurance of 100% similarity in classifying standards.
- **More losses than gains:** Although manual classification can be cost effective, in cases where classification standards vary from one dataset to another, redoing them again and again will not only prove to be a waste of time but also could incur losses for that organization (*Management Association, 2013*).

By training a machine to classify datasets, we will not only be able to achieve uniformity, but will also be able to save time and assure reproducibility, hence, increasing efficiency to a great extent. Artificial Neural Network will help us to do just that.

1.02 Objective

The basic objective of this project is to instill the essence of “*Smart Processing*” rather than labor-intensive, grueling work. This tool aims to cater to the needs of many for reducing manual work and classifying datasets in comparatively shorter period of time, whereby also providing uniformity and a certain amount of reliability.

1.03 About Artificial Neural Network

Artificial neural networks (ANNs) are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning (*Wikipedia, Artificial neural network*).

1.04 Usefulness and Capabilities

- **Exploits non- linearity:** It can help in solving non-linear problems, unlike the single layer perceptron.
- **Input/output mapping:** Its learning ability sets it apart from other computational techniques.
- **Adaptivity:** It can adapt free parameters to its surroundings. In terms of humans, the free parameters could refer to the synapses.
- **Evidential response:** It takes decisions with a measure of confidence.
- **Fault tolerance:** Since a large number of neurons are involved, the failure of one does not result in the failure of the complete system. Thus, it exhibits graceful degradation.
- **VLSI Implementation:** It features VLSI (Very Large Scale Implementation) qualities.
- **Neurobiological Analogy:** It is motivated by human analogy.

Thus, with these advantages in hand, we hope to build a Python tool in QGIS which will, to an extent, automate the Land use Land cover classification procedures, keeping in mind the importance of accuracy and time.

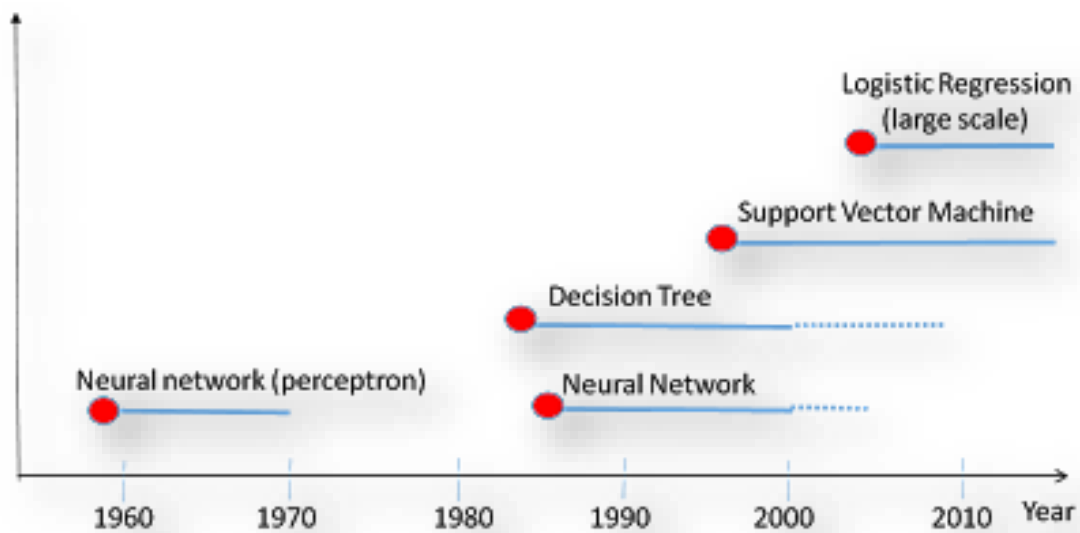
2.01 Machine Learning Through the Ages

“Those who cannot remember the past are condemned to repeat it.”

- George Santayana (Santayana, 1905-1906)

The development of machine learning is an integral part of the development of artificial intelligence. In the early days of AI, people were interested in building machines that mimic human brains. The perceptron model was invented in 1957, and it generated over optimistic view for AI during 1960s. After **Marvin Minsky** pointed out the limitation of this model in expressing complex functions, researchers stopped pursuing this model for the next decade.

Figure 01: Machine Learning: Time Line (Hu, 2013)



In 1970s, the machine learning field was dormant, when expert systems became the mainstream approach in AI. The revival of machine learning came in mid-1980s, when the decision tree model was invented and distributed as software. The model can be viewed by a human and is easy to explain. It is also very versatile and can adapt to widely different problems. It is also in mid 1980s multi-layer neural networks were invented, with enough hidden layers; a neural network can express any function, thus overcoming the limitation of perceptron. We see a revival of the neural network study.

Both decisions trees and neural networks see wide application in financial applications such as loan approval, fraud detection and portfolio management. They are also applied to a wide-range of industrial process and postal office automation (address recognition).

Machine learning saw rapid growth in 1990s, due to the invention of World-Wide-Web and large data gathered on the Internet. The fast interaction on the Intern called for more automation and more adaptivity in computer systems. Around 1995, SVM was proposed and have become quickly adopted. SVM packages like libSVM, SVM light make it a popular method to use.

After the year 2000, Logistic regression was rediscovered and re-designed for large scale machine learning problems. In the ten years following 2003, logistic regression has attracted a lot of research work and has become a practical algorithm in many large-scale commercial systems, particularly in large Internet companies (*Hu, 2013*).

2.02 History of Neural Networks

The study of the human brain is thousands of years old. With the advent of modern electronics, it was only natural to try to harness this thinking process. The first step toward artificial neural networks came in 1943 when **Warren McCulloch**, a neurophysiologist, and a young mathematician, **Walter Pitts**, wrote a paper on how neurons might work. They modeled a simple neural network with electrical circuits.

Reinforcing this concept of neurons and how they work was a book written by **Donald Hebb**. The *Organization of Behavior* was written in 1949. It pointed out that neural pathways are strengthened each time that they are used (*Artificial Neural Networks Technology*).

W.S. McCulloch, W. Pitts described the first Neural Network Model and **F. Rosenblatt** (Perceptron) and **B. Widrow** (Adaline) developed the first training algorithm (*J. Vieira, 1997*).

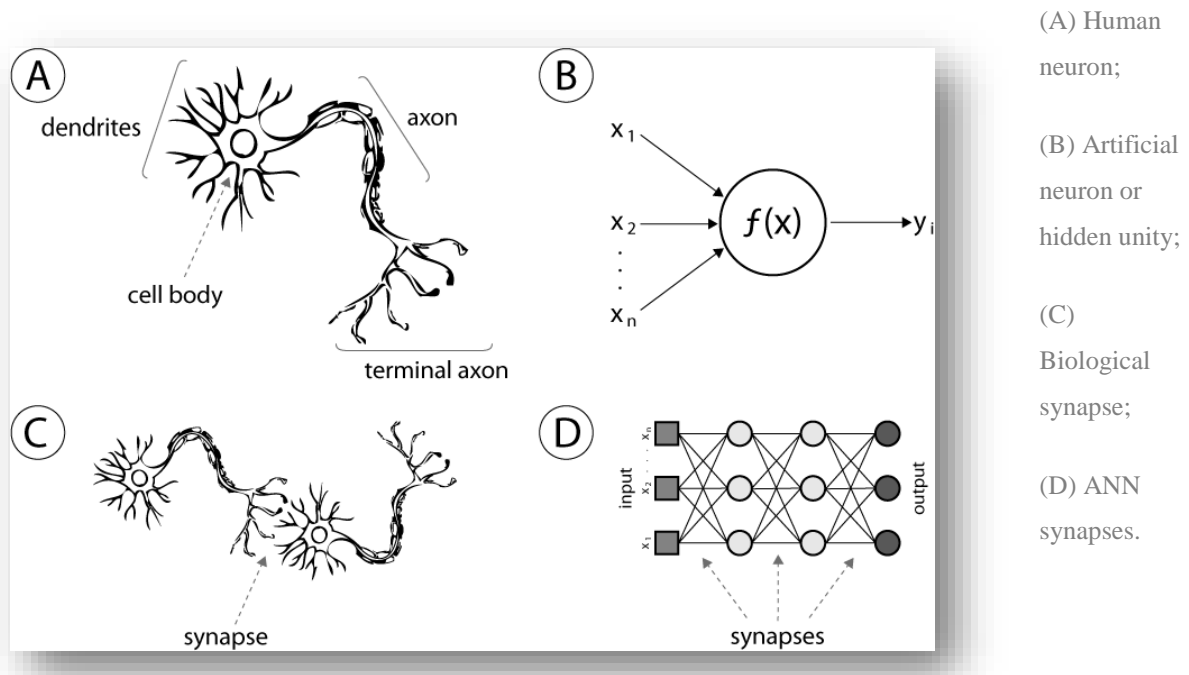
A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects (*Haykin, 1994*). Firstly, knowledge is acquired by the network through a learning process and secondly, interneuron connection strengths known as synaptic weights

are used to store the knowledge. Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store, and utilize experiential knowledge (Zurada, 1992).

2.03 Analogy Between Human and Artificial Neural Nets

Artificial neural nets were originally designed to model in some small way, the functionality of the biological neural networks, which are a part of the human brain. Our brains contain about 10^{11} neurons. Each biological neuron consists of a cell body, a collection of dendrites which bring electrochemical information into the cell and an axon which transmits electrochemical information out of the cell.

Figure 2.02: Analogy between human neurons and artificial nets
(Vinícius Gonçalves Maltarollo, January 16, 2013)



A neuron produces an output along its **axon** i.e. it fires when the collective effect of its inputs reaches a certain threshold. The axon from one neuron can influence the **dendrites** of another neuron across junctions called **synapses**. Some synapses will generate a positive effect in the dendrite, i.e. one which encourages its neuron to fire, and others will produce a negative effect, i.e. one which discourages the neuron from firing. A single neuron receives inputs from perhaps 10^5 synapses. It is still not clear exactly how our brains learn and remember but it appears to be associated with the interconnections between the neurons (i.e. at the synapses).

Artificial neural nets try to model this low level functionality of the brain. This contrasts with high level symbolic reasoning in artificial intelligence which tries to model the high level reasoning processes of the brain. When we think, we are conscious of manipulating concepts to which we attach names (or symbols) e.g. for people or objects. We are not conscious of the low level electrochemical processes which are going on underneath. The argument for the neural net approach to AI is that, if we can model the low level activities correctly, the high level functionality may be produced as an emergent property.

A single software artificial neuron consists of a processing element which has a number of **input** connections, each with an associated **weight**, a **transfer function** which determines the output, given the weighted sum of the inputs, and the **output** connection itself.

An artificial neural network is a network of interconnected neurons. The network may be trained by adjusting the weights associated with the connections in the net to try and obtain the required outputs for given inputs from a training set.

Note that the threshold values and the weights can be adjusted together by adding an extra connection to each neuron with an input value of -1 and a weight representing the threshold. The neuron then fires if the sum is greater than zero.

It can be seen that there is an analogy between biological (human) and artificial neural nets. The analogy is summarized below:

Table 2.01: An Analogy between Human and Artificial Neurons

Human	Artificial
Neuron	Processing Element
Dendrites	Combining Function
Cell Body	Transfer Function
Axons	Element Output
Synapses	Weights

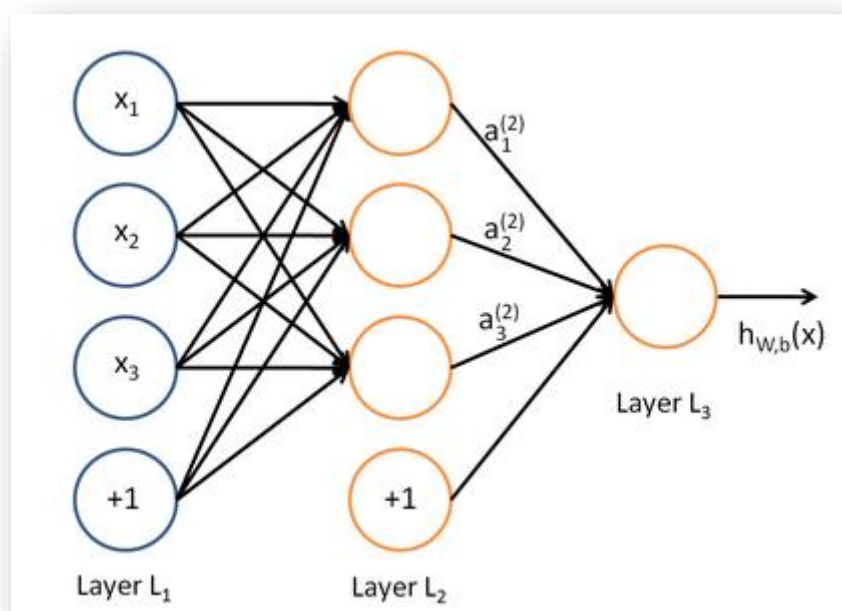
However, it should be stressed that the analogy is not a strong one. Biological neurons and neuronal activity are far more complex than might be suggested by studying artificial neurons. Real neurons do not simply sum the weighted inputs and the dendritic mechanisms in biological systems are much more elaborate. Also, real neurons do not stay *on* until the inputs change and the outputs may encode information using complex pulse arrangements (*Lewis*).

Today, researchers have progressed to such an extent so as to have built **the world's first artificial neuron that is capable of mimicking the function of an organic brain cell**—including the ability to translate chemical signals into electrical impulses, and communicate with other human cells. **Richter-Dahlfors** and her team have now managed to create an artificial neuron that can mimic this function, and they have shown that it can communicate chemically with organic brain cells even over large distances (*MacDonald, 2015*).

2.04 Neural Network Model

A neural network is put together by hooking together many of our simple "neurons," so that the output of a neuron can be the input of another. For example, here is a small neural network:

Figure 2.03: Network Layers (*Neural Networks, 2013*)



In this figure, we have used circles to denote the inputs to the network. The circles labeled "+1" are called **bias units**, and correspond to the intercept term. The leftmost layer of the network is called the **input layer**, and the

rightmost layer the **output layer** (which, in this example, has only one node). The middle layer of nodes is called the **hidden layer** because its values are not observed in the training set. We also say that our example neural network has 3 **input units** (not counting the bias unit), 3 **hidden units**, and 1 **output unit** (*Neural Networks, 2013*).

2.05 Architecture of Artificial Neural Network

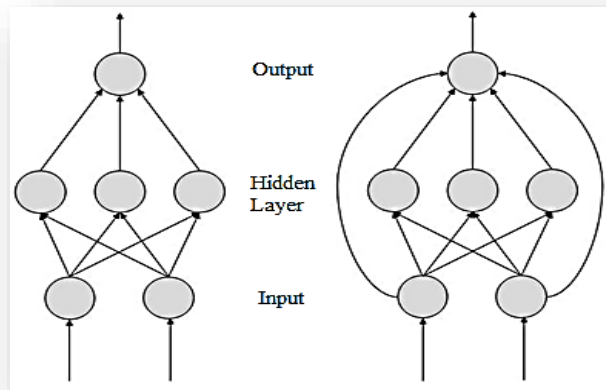
There are two Artificial Neural Network topologies – **Feedforward** and **Feedback**.

(a) Feedforward ANN:

In a Feedforward ANN, the connections between units do not form cycles. It usually produces a response to an input quickly.

The information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation, recognition and classification. They have fixed inputs and outputs (*Tutorials Point*).

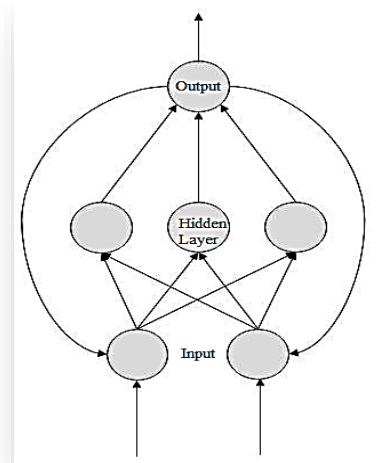
Figure 2.04: Feedforward ANN



(b) Feedback ANN:

Feedback networks can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations (*Christos Stergiou*).

Figure 2.05: Feedback ANN

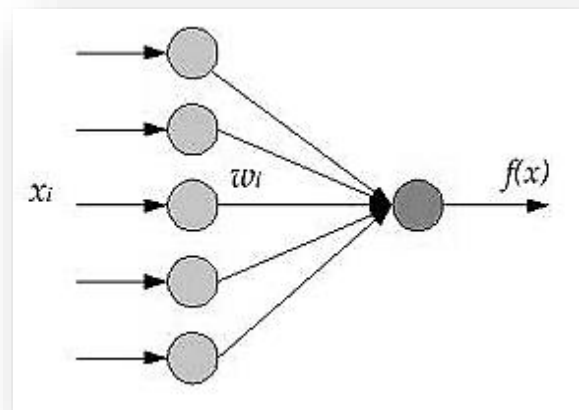


2.06 The Perceptron: The Fundamentals

The perceptron was first introduced by **F. Rosenblatt** in 1958 (*Types of Neural Nets*).

A perceptron is a unit that computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights and then possibly putting the

Figure 2.06: A Perceptron



output through some non-linear function called the **activation function** (*Multilayer Perceptron in Python, 2014*).

The output of perceptron can be expressed as:

$$f(x) = G(W^T x + b)$$

Where:

- (x) : the input vector;
- (W,b) : the parameters of perceptron;
- (f) : the non-linear function.

The perceptron is trained (i.e., the weights and threshold values are calculated) based on an **iterative training phase** involving training data. Training data are composed of a list of input values and their associated desired output values. In the training phase, the inputs and related outputs of the training data are repeatedly submitted to the perceptron.

The perceptron calculates an output value for each set of input values. If the output of a particular training case is labeled 1 when it should be labeled 0, the threshold value (theta) is increased by 1, and all weight values associated with inputs of 1 are decreased by 1. The opposite is performed if the output of a training case is labeled 0 when it should be labeled 1. No changes are made to the threshold value or weights if a particular training case is correctly classified (*Leverington, 2009*).

The learning process in a perceptron is supervised and the net is able to solve basic logical operations like **AND** or **OR**. It is also used for **pattern classification purposes**. More complicated logical operations (like the XOR problem) cannot be solved by a Perceptron (*Types of Neural Nets*).

2.07 Multi-Layer Perceptron (MLP)

Multilayer Perceptron (MLP) was first introduced by **M. Minsky** and **S. Papert** in 1969 and is classified as a type of Artificial Neural Network: the computation is performed using a set of (many) simple unit with weighted connections between them. Furthermore, there are **learning algorithms** to set the values of the weights and the same basic structures (with different weight values) are able to perform many tasks (*Gales, 2015*).

It is an **extended perceptron** and has one or more hidden neuron layers between its input and output layers. Due to its extended structure, a Multi-Layer Perceptron is able to solve every logical operation, including the **XOR problem** (*Types of Neural Nets*). Intermediate layers usually have tan hyperbolic (tanh) or the sigmoid function (defined here by a ‘**HiddenLayer**’ class) as **activation function**.

The **number of hidden layers** determines the decision boundaries that can be generated. In choosing the number of layers, the following considerations are made:

- Multi-layer networks are harder to train than single layer networks.
- A two layer network (one hidden) with sigmoidal activation functions can model any decision boundary.

Two layer networks are most commonly used in pattern recognition (the hidden layer having sigmoidal activation functions) (*Gales, 2015*).

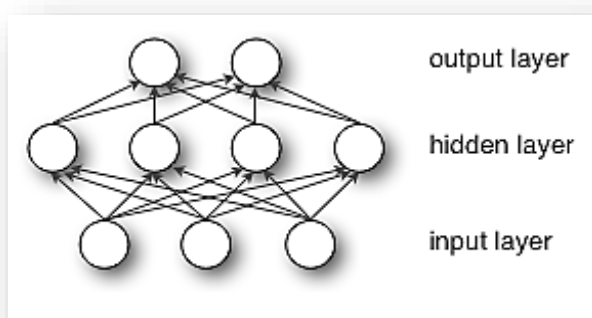
An MLP can be viewed as a logistic regression classifier where the input is first transformed using a learnt non-linear transformation ϕ . This transformation projects the input data into a space where it becomes linearly separable. This intermediate layer is referred to as a **hidden layer**. A single hidden layer is sufficient to make MLPs a **universal approximator** (*Multilayer Perceptron, 2016*).

The Model

a) Single Layer Multilayer Perceptron

An MLP (Multi Layer Perceptron) with a single hidden layer can be represented graphically as follows:

Figure 2.07: Single Layer Multilayer Perceptron



Formally, a one-hidden-layer MLP is a function

$$f: R^D \rightarrow R^L$$

Where D is the size of input vector x and L is the size of the output vector $f(x)$, such that, in matrix notation:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x)))$$

With bias vectors $b^{(1)}$, $b^{(2)}$; weight matrices $W^{(1)}$, $W^{(2)}$ and activation functions G and s (*Multilayer Perceptron, 2016*).

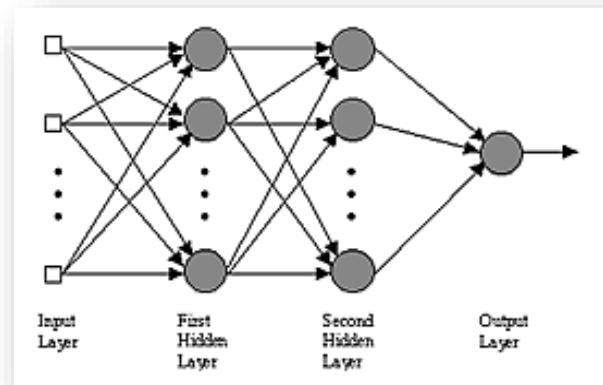
b) MLP with Multiple Hidden Layers

Multi-layer perceptron allow a neural network to perform arbitrary mappings.

A 2-hidden layer neural network is shown in the given figure. The aim is to map an input vector x into an output $y(x)$. The layers may be described as:

- **Input layer:** Accepts the data vector or pattern;
- **Hidden layers:** One or more layers. They accept the output from the previous layer, weighs them, and pass through a, normally, non-linear activation function;
- **Output layer:** Takes the output from the final hidden layer, weighs them and possibly passes through an output non-linearity to produce the target values.

Figure 2.08: MLP with Two Hidden Layers (*Neural Network, 2016*)



The MLP and many other neural networks learn using an algorithm called **backpropagation**. With backpropagation, the input data is repeatedly presented to the neural network. With each presentation the output of the neural network is compared to the desired output and an error is computed. This error is then fed back (backpropagated) to the neural network and used to adjust the weights such that the error decreases with each iteration and the neural model gets closer and closer to producing the desired output. This process is known as "**training**" (*Neural Network, 2016*).

Applications of MLP:

MLP has been used in a wide area for various applications, all of which can be stratified as pattern classification, function approximation or prediction. Pattern classification is the configuration of patterns into groups of patterns having the same set of properties. It is well

known that MLPs are universal in the sense that they can approximate any continuous non-linear function arbitrarily well on a compact interval.

As a result, MLP became popular in order to parameterize nonlinear models and classifiers, often leading to improved results compared to classical, (V. Cherkassky, 1998) MLP has proved to be a very effective tool for the classification of remote-sensing images. However, the training of such a classifier, by using data with very different a priori class probabilities (imbalanced data), is very slow.

It is an effective method which describes a learning technique aimed at speeding up the training of a MLP also the classification becomes stable (with respect to initial weights) when applied to imbalanced data (L. Bruzzone, 1997). Various efforts are being made in order to optimize the fault tolerance of MLP in pattern classification problems. Fault tolerance is a frequently cited advantage of ANN. SLP was considerably less fault tolerant than any of the MLPs, including one with fewer adjustable weights (M. D. Emmerson, 1993).

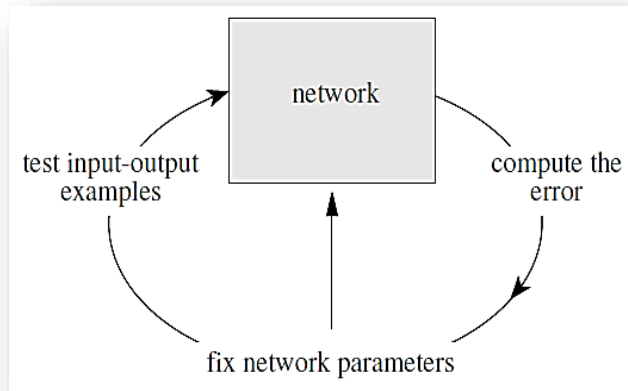
MLP has been applied within the field of air-quality prediction. According to the work of **Yi and Prybutok**, MLP helped in predicting ozone concentration on the surface of an industrial area in North America (J. Yi, 1996). Weather forecasting is a difficult task to be undertaken. In 1996, **Marzban and Stumpf** predicted the existence of tornadoes, using MLP. This approach outperformed other techniques including discriminant analysis, logistic regression and rule based algorithm (Marzban, 1996). MLP was used for many applications such as predicting monsoon and rainfall (Ceccatto, 1994), distinguishing clouds and ice or snow in Polar Regions (R. M. Welch, 1992), interpret satellite imagery for identifying cyclones, war fronts, and weather conditions (Tag, 1992). MLP can act as a useful tool to implement various other applications such as: paper currency recognition, the diagnosis of low back pain and sciatica, heart disease and cancer, stock market prediction, prediction of daily global solar radiation, handwritten character recognition, image classification, object recognition, feature extraction and many more (Anil Kumar Goswami, 2014).

2.08 The Perceptron Learning Algorithm

A learning algorithm is **an adaptive method** by which a network of computing units self-organizes to implement the desired behavior. This is done in some learning algorithms by presenting some examples of the desired input-output mapping to the network. A correction step is executed iteratively until the network learns to produce the desired response. The

learning algorithm is a closed loop of presentation of examples and of corrections to the network parameters, as shown below:

Figure 2.09: Learning process in a parametric system (Rojas, 1996)



In some simple cases, the weights for the computing units can be found through a sequential test of stochastically generated numerical combinations. However, such algorithms which look blindly for a solution do not qualify as “learning”. A learning algorithm must adapt the network parameters according to previous experience until a solution is found, if it exists

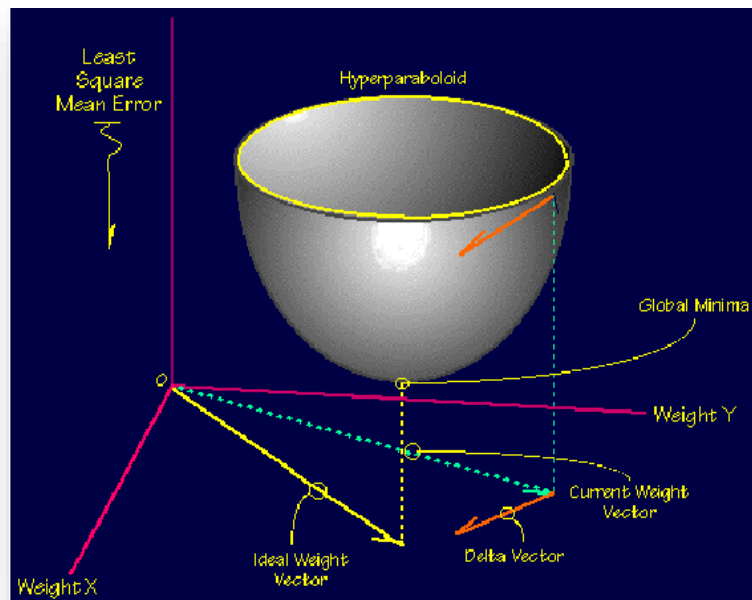
(Rojas, 1996).

2.09 Artificial Neural Networks and Gradient Descent Algorithm

Global and Local Minimum

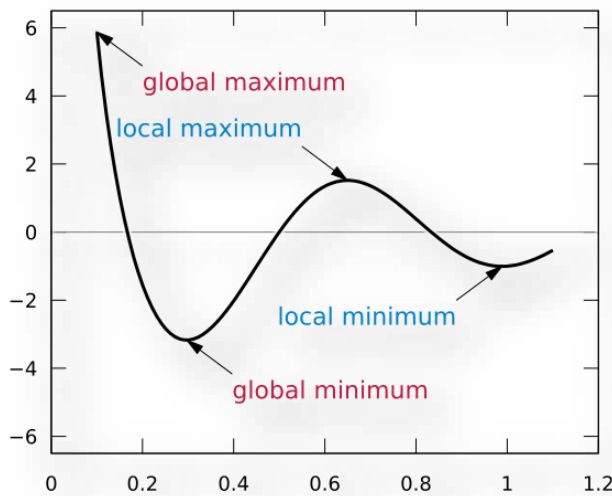
The global minimum is a theoretical solution with the lowest possible error. The error surface itself is a hyperparaboloid but is seldom 'smooth' as is depicted in the given figure. Indeed, in most problems, the solution space is quite irregular with numerous 'pits' and 'hills' which may cause the network to settle down in a 'local minimum' which is not the best overall solution.

Figure 2.10: Global Minimum



Since the nature of the error space cannot be known *a priori*, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the 'speed' (Beta-coefficient) and the 'momentum' of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Figure 2.11: Local and Global Maxima and Minima (Wikipedia, Maxima and Minima, 2016)



Local and global maxima and minima for $\cos(3\pi x)/x$, $0.1 \leq x \leq 1.1$

Gradient Descent Algorithm

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the *positive* of the gradient, one approaches a local maximum of that function; the

procedure is then known as **gradient ascent**.

Gradient descent is based on the observation that if the multi-variable function $F(x)$ is defined and differentiable in a neighborhood of a point a , then $F(x)$ decreases *fastest* if one goes from a in the direction of the negative gradient of F at a , $-\nabla F(a)$. It follows that, if

$$b = a - \gamma \nabla F(a)$$

If γ is small enough, then $F(a) \geq F(b)$. In other words, the term $\gamma \nabla F(a)$ is subtracted from a because we want to move against the gradient, namely down toward the minimum. With this observation in mind, one starts with a guess x_0 for a local minimum of F , and considers the sequence x_0, x_1, x_2, \dots such that:

$$x_{n+1} = x_n - \gamma n \nabla F(x_n, n) \geq 0,$$

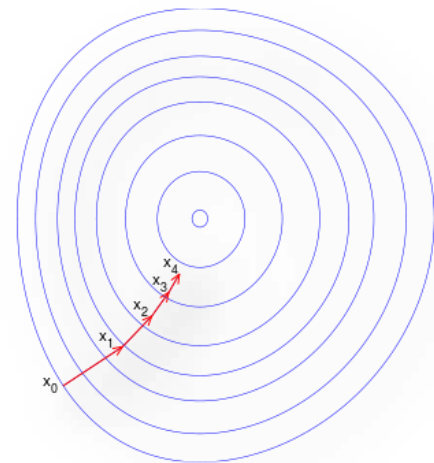
We have:

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \dots,$$

So hopefully the sequence (x_n) converges to the desired local minimum. Note that the value of the *step size* γ is allowed to change at every iteration. With certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ (e.g., chosen via a line search that satisfies the Wolfe conditions), convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

This process is illustrated in the figure to the right. Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient *descent* leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal (Gradient descent, 2016).

Figure 2.12: Illustration of Gradient Descent (Gradient descent, 2016)



2.10 Non Linear Activation Functions

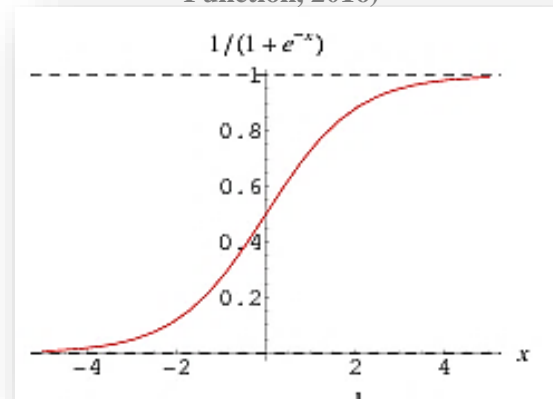
The operation of an artificial neural network is to sum up the product of the associated weight and the input signal and produce an output or activation function. For the input unit this activation function is the identity function. The neuron of a particular layer gets the same type of activation function. In almost all cases, non-linear activation functions are used (K.Vijayarekha).

Some of the activation functions commonly used for neurons is given below:

Sigmoid function

Sometimes S shaped functions called Sigmoid functions or Logistic functions are used as activation functions which are found useful. Logistic and hyperbolic tangent functions are commonly used sigmoid functions. The sigmoid functions are extensively used in back propagation neural networks because it reduces the burden of complication involved during training phase (*K.Vijayarekha*).

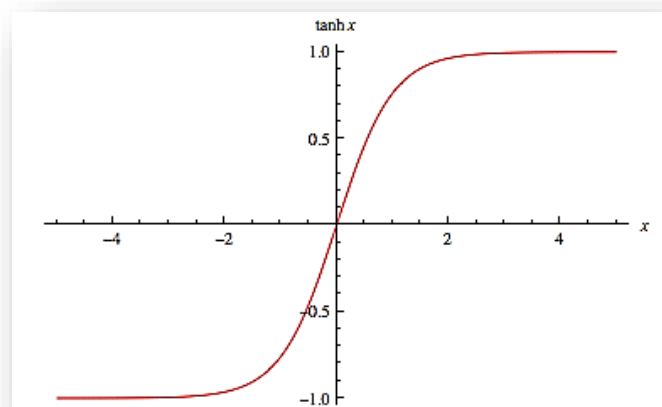
Figure 2.13: Sigmoid Function (Weisstein, Sigmoid Function, 2016)



Tan Hyperbolic Function

Though the logistic sigmoid has a nice biological interpretation, it turns out that the logistic sigmoid can cause a neural network to get “stuck” during training. This is due in part to the fact that if a strongly-negative input is provided to the logistic sigmoid, it outputs values very near zero. Since neural networks use the feed-

Figure 2.14: Graphical Representation of the Tan Hyperbolic Function (Weisstein, Hyperbolic Tangent, 2016)



forward activations to calculate parameter gradients, this can result in model parameters that are updated less regularly than we would like, and are thus “stuck” in their current state.

An alternative to the logistic sigmoid is the hyperbolic tangent, or tanh function:

$$\begin{aligned}\partial \tanh(z) &= \frac{\sinh(z)}{\cosh(z)}, \\ &= \frac{e^z - e^{-z}}{e^z + e^{-z}}\end{aligned}$$

Like the logistic sigmoid, the tanh function is also sigmoidal (“s”-shaped), but instead outputs values that range (-1, 1). Thus, strongly negative inputs to the tanh will map to negative outputs. Additionally, only zero-valued inputs are mapped to near-zero outputs. These properties make the network less likely to get “stuck” during training (*Stansbury, 2016*).

2.11 Learning Strategies in Neural Network

ANNs are capable of learning and they need to be trained. There are several learning strategies:

- **Supervised Learning:** It involves a teacher that is scholar than the ANN itself. For example, the teacher feeds some example data about which the teacher already knows the answers.
- **Unsupervised Learning:** It is required when there is no example data set with known answers. For example, searching for a hidden pattern. In this case, clustering i.e. dividing a set of elements into groups according to some unknown pattern is carried out based on the existing data sets present.
- **Reinforcement Learning:** This strategy built on observation. The ANN makes a decision by observing its environment. If the observation is negative, the network adjusts its weights to be able to make a different required decision the next time.

2.12 Learning Mechanisms in NN

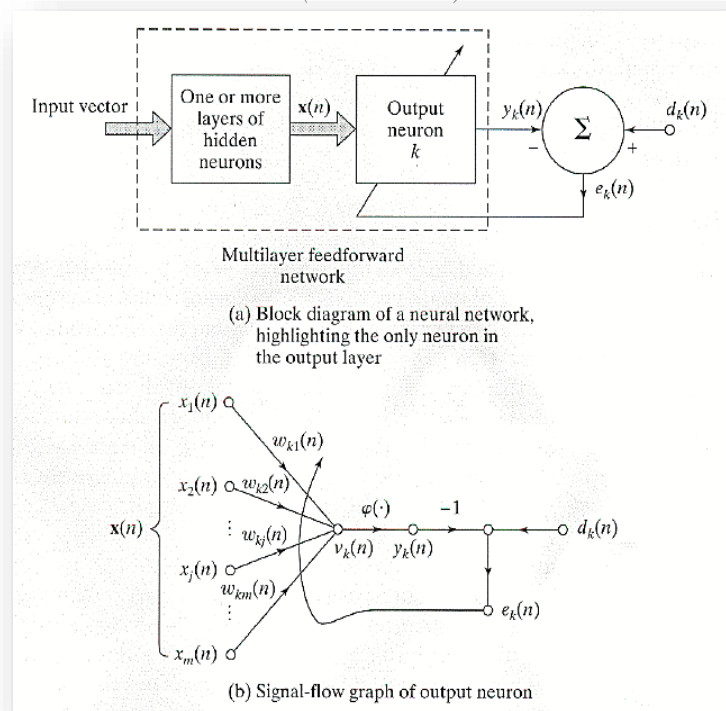
There are five basic learning mechanisms in Neural Network:

1. Error Correction Learning

In this mechanism, the learning takes place by iteratively finding out the error and consistently adjusting the weights such that at the last iteration, the error is zero. It includes a step-by-step adjustment until system reaches steady state and the synaptic weights are stabilized.

Thus, at time step 'n', the difference between the desired output via input 'k' (d_k) and the

Figure 2.15: Illustration of Error Correction Learning (Bennamoun)



actual output $y_k(n)$ will result in the error (e_k):

$$e_k(n) = d_k(n) - y_k(n)$$

The cost function ' $E(n)$ ' or the '**Index of Performance**' can be formulated using:

$$E(n) = 1/2 \sum e^2(n)$$

Minimization of this error can be done using the '**Widrow-Hoff Rule**' or '**Delta Rule**'.

$$\Delta W_{kj}(n) = \eta e_k(n) \cdot x_j(n)$$

Where:

η is the learning rate.

Thus, after correction, the formula for the **updated synaptic weight** will be as follows:

$$W_{kj}(n+1) = W_{kj}(n) + \Delta W_{kj}(n)$$

Here the adjustment is proportional to the product of error signal and the input signal error-correction learning is local. It is the learning rate η that determines the stability or the convergence.

2. Memory Based Learning

In memory-based learning, all (or most) of the past experiences are explicitly stored in a large memory of correctly classified input-output examples:

$$\{(x_i, d_i)\}_{i=1}^N$$

Where x_i denotes an input vector and d_i denotes the corresponding desired response. When classification of a test vector x_{test} (not seen before) is required, the algorithm responds by **retrieving** and **analyzing** the training data in a "local neighborhood" of x_{test} .

All memory-based learning algorithms involve 2 essential Ingredient (which make them different from each other):

- Criterion used for defining local neighbor of \mathbf{x}_{test} .
- Learning rule applied to the training examples in local neighborhood of \mathbf{x}_{test} .

Nearest Neighbor Rule (NNR): When we feed a new vector \vec{X}_{test} , it finds, from the memory, which of these memories is going to be the closest to the new vector. The Euclidean distance is found from \vec{X}_{test} and each \vec{X}_i . The vector $\mathbf{X}'_N \in \{\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_N\}$ is the nearest neighbor of \mathbf{X}_{test} if:

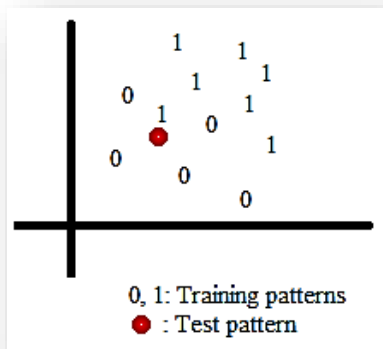
$$\min_i d(\vec{X}_i, \vec{X}_{test}) = d(\vec{X}'_N, \vec{X}_{test})$$

Where \mathbf{X}'_N is the class of \mathbf{X}_{test} . The corresponding d is going to be the response for \vec{X}_{test} (Bennamoun).

k-Nearest Neighbor Rule: It is a variant of the Nearest Neighbor Rule and identifies the k classified patterns that lay nearest to \vec{X}_{test} for some integer k . Also, \vec{X}_{test} is assigned to the class that is most frequently represented in the k nearest neighbors to \vec{X}_{test} .

Let us assume some patterns to be '0' and '1'. Thus after arranging these patterns we gain the following graph:

Figure 2.16: Outliers: Cause of Faulty Classification



In the given graph, we see the distribution of patterns '0' and '1'. There is a test pattern (marked in red) present which has to be classified. Ideally this test pattern should be classified as '0'. However, due to the presence of an 'outlier' (in this case pattern '1', lying closest to the test pattern), the test pattern will be classified as belonging to pattern '1' which will lead to faulty classification.

To overcome this flaw, the k-Nearest Neighborhood rule is used where not only the nearest neighbor but the nearest neighborhood is identified.

3. Hebbian Learning

According to Hebb, "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes place in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" (Hebb, 1949).

In other words:

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e. synchronously), then the strength of that synapse is selectively increased.
2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated so that chance coincidences do not build up connection strengths.

Thus, if the cross product of output and input (or correlation) is positive, it results in an increase of the weight, otherwise the weight decreases. It can be seen that the output is strengthened in turn for each input presented.

The Hebbian synapse is characterized by the following characteristics:

- Time dependent:
 - Depend on exact time of occurrence of two signals
- Local:
 - Locally available information is used
- Interactive mechanism:
 - Learning is done by two signal interaction
- Conjunctive or correlational mechanism:
 - Co-occurrence of two signals

Hebbian learning is found in Hippocampus of the human brain.

4. Competitive Learning:

This is an **unsupervised network training** and is applicable for an ensemble of neurons (e.g. a layer of p neurons), not for a single neuron. In this type of learning, the output neurons of NN compete to become active. Only a single neuron is active at any one time. Neurons learn to specialize on ensembles of similar patterns. Therefore, they become **feature detectors**.

Competitive learning is a rule based on the idea that only one neuron from a given iteration in a given layer will fire at a time. Weights are adjusted such that only one neuron in a layer, for instance the output layer, fires. Competitive learning is useful for classification of input patterns into a discrete set of output classes. The “winner” of each iteration, element i^* , is the element whose total weighted input is the largest (Artificial Neural Networks/Competitive

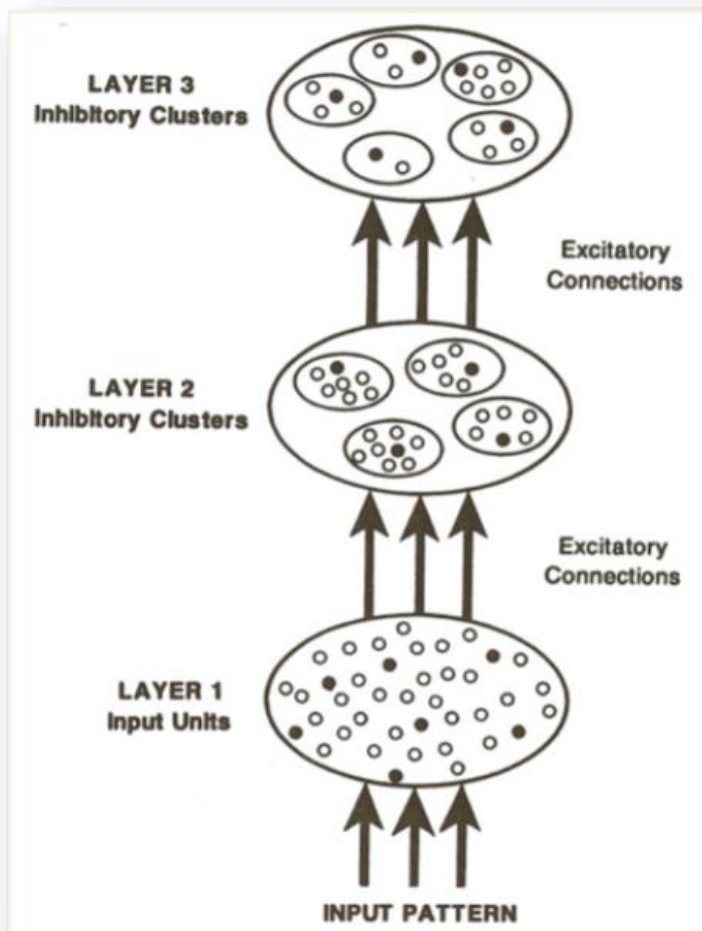
Learning, 2013). Using this notation, one example of a competitive learning rule can be defined mathematically as:

$$w_{ij}[n + 1] = w_{ij}[n] + \Delta w_{ij}[n]$$

$$\Delta w_{ij} = \begin{pmatrix} \eta(x_i - w_{ij}) & \text{if } i = j \\ 0 & \text{otherwise} \end{pmatrix}$$

Competitive learning takes place in a context of sets of hierarchically layered units. Units are represented in the diagram as dots. Units may be active or inactive. Active units are represented by filled dots, inactive ones by open dots. In general, a unit in a given layer can receive inputs from all of the units in the next lower layer and can project outputs to all of the

Figure 2.17: Architecture of the competitive learning mechanism (Zipser, 1985).



units in the next higher layer. Connections between layers are excitatory and connections within layers are inhibitory. Each layer consists of a set of clusters of mutually inhibitory units. The units within a cluster inhibit one another in such a way that only one unit per cluster may be active. We think of the configuration of active units on any given layer as representing the input pattern for the next higher level. There can be an arbitrary number of such layers. A given cluster contains a fixed number of units, but different clusters

can have different numbers of units (Zipser, 1985).

There are many variants to the basic competitive learning model. **Von der Malsburg** (von der Malsburg, 1973), **Fukushima** (Fukushima, 1975), and **Grossberg** (Grossberg, 1976)

among others, have developed competitive learning models. In this section we describe the simplest of the many variations. The version we describe was first proposed by Grossberg (*Grossberg, 1976*) and is the one studied by **Rumelhart** and **Zipser**. This version of competitive learning has the following properties:

- The units in a given layer are broken into several sets of non-overlapping clusters. Each unit within a cluster inhibits every other unit within a cluster. Within each cluster, the unit receiving the largest input achieves its maximum value while all other units in the cluster are pushed to their minimum value. We have arbitrarily set the maximum value to 1 and the minimum value to 0.
- Every unit in every cluster receives inputs from all members of the same set of input units.
- A unit learns if and only if it wins the competition with other units in its cluster. The winner that wins the competition is called “**winner-takes-all**”.
- A stimulus pattern S_j consists of a binary pattern in which each element of the pattern is either active or inactive. An active element is assigned the value 1 and an inactive element is assigned the value 0.
- Each unit has a fixed amount of weight (all weights are positive) that is distributed among its input lines. The weight on the line connecting to unit i on the upper layer from unit j on the lower layer is designated w_{ij} . The fixed total amount of weight for unit j is designated $\sum_j w_{ij} = 1$. A unit learns by shifting weight from its inactive to its active input lines. If a unit does not respond to a particular pattern, no learning takes place in that unit. If a unit wins the competition, then each of its input lines gives up some portion ϵ of its weight and that weight is then distributed equally among the active input lines. Mathematically, this learning rule can be stated

$$\Delta w_{ij} = \begin{pmatrix} 0 & \text{if unit } i \text{ loses on stimulus } k \\ \epsilon \frac{\text{active}_{jk}}{\text{nactive}_k} - \epsilon w_{ij} & \text{if unit } i \text{ wins on stimulus } k \end{pmatrix}$$

Where active_{jk} is equal to 1 if in stimulus pattern S_k , unit j in the lower layer is active and is zero otherwise, and nactive_k is the number of active units in pattern S_k (thus, $\text{nactive}_k = \sum_j \text{active}_{jk}$) (Competitive Learning).

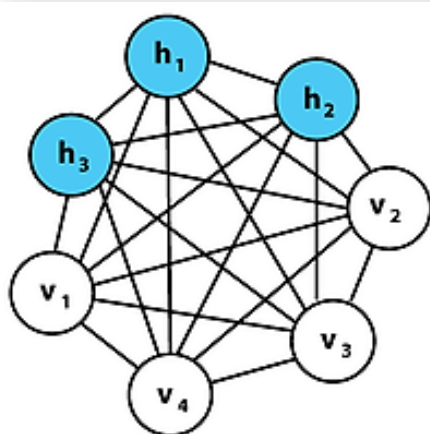
5. Boltzmann Learning

Boltzmann learning is a **stochastic learning algorithm**, derived from statistical mechanics. It is similar to error-correction learning and is used during supervised training. In this algorithm, the state of each individual neuron, in addition to the system output, are taken into account. In this respect, the Boltzmann learning rule is significantly slower than the error-correction learning rule. Neural networks that use Boltzmann learning are called Boltzmann machines.

Boltzmann learning is similar to an error-correction learning rule, in that an error signal is used to train the system in each iteration. However, instead of a direct difference between the result value and the desired value, we take the difference between the probability distributions of the system (*Artificial Neural Networks/Boltzmann Learning, 2010*).

The **Boltzmann machine** is the Neural Network basis of Boltzmann learning. A Boltzmann machine is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off. Boltzmann machines have a simple learning algorithm (*G. E. Hinton, 1983*) that allows them to discover interesting features that represent complex regularities in the training data (*Hinton, 2007*).

Figure 2.18: Boltzmann Machine
(Boltzmann machine, 2016)



A graphical representation of a Boltzmann machine with a few weights labeled. Each undirected edge represents dependency and is weighted with weight w_{ij} . In this example there are 3 hidden units (blue) and 4 visible units (white).

A Boltzmann machine, like a Hopfield network, is a network of units with an "energy" defined for the network. It also has binary units, but unlike Hopfield nets, Boltzmann machine units are stochastic. The global energy, E , in a Boltzmann machine is identical in form to that of a Hopfield network:

$$E = - \left(\sum_{i < j} w_{ij} s_i s_j + \sum_i \theta_i s_i \right)$$

Where:

- w_{ij} : The connection strength between unit j and unit i .
- s_i : The state, $s_i \in \{0, 1\}$,
- θ_i : The bias of unit i in the global energy function. ($-\theta_i$ is the activation threshold for the unit.)

The connections in a Boltzmann machine have two restrictions:

- $w_{ii} = 0 \quad \forall i$. (No unit has a connection with itself.)
- $w_{ij} = w_{ji} \quad \forall i, j$. (All connections are symmetric.)

Often the weights are represented in matrix form with a symmetric matrix W , with zeros along the diagonal.

The units in the Boltzmann Machine are divided into '**visible**' units, V , and '**hidden**' units, H . The visible units are those which receive information from the 'environment', i.e. our training set is a set of binary vectors over the set V . The distribution over the training set is denoted as $P^+(V)$.

Remarkably, this learning rule is fairly biologically plausible because the only information needed to change the weights is provided by "local" information. That is, the connection (or synapse biologically speaking) does not need information about anything other than the two neurons it connects (*Boltzmann machine, 2016*).

2.13 Backpropagation Algorithm

The Backpropagation Algorithm is a learning algorithm used by neural nets with **supervised learning**. It is a special form of the **delta learning rule** (BackPropagation). The backpropagation algorithm was originally introduced in the 1970s, but its importance was not fully appreciated until a famous 1986 paper by **David Rumelhart, Geoffrey Hinton** and **Ronald Williams**. The paper describes several neural networks where backpropagation works far faster than earlier approaches to learning, making it possible to use neural nets to

solve problems which had previously been insoluble. Today, the backpropagation algorithm is the workhorse of learning in neural networks (*Nielsen, 2015*).

The backpropagation algorithm trains a given feed-forward multilayer neural network for a given set of input patterns with known classifications. When each entry of the sample set is presented to the network, the network examines its output response to the sample input pattern. The output response is then compared to the known and desired output and the error value is calculated. Based on the error, the connection weights are adjusted. The backpropagation algorithm is based on **Widrow-Hoff delta learning rule** in which the weight adjustment is done through **mean square error** of the output response to the sample input (*Velasquez, 1998*). The set of these sample patterns are repeatedly presented to the network until the error value is minimized (*Kawaguchi, 2006*).

In this project, a combination of Multilayer Perceptron along with Backpropagation will be used to train and classify the available data to produce an LULC map with the help of training sets that have gone through Feature Scaling.

2.14 Working with UAV acquired data

Remotely gathered data is available from a range of sources (Satellite and Aerial Photography) and data collection techniques and is often the only type of data that is not always easily found within the public domain. This is largely due to the fact that most of this data is acquired by equipment that is expensive to build and maintain.

Despite its high bandwidth, coverage over a large geographical area and proven to be cheaper over long distances (*Principles of Data Communications: Media Characteristics*), satellite images have several drawbacks. Some are listed as below:

- **Accessibility:** Apart from a few selected satellites, most of the high resolution satellite data have to be purchased or a request has to be made for it to be accessible for educational purposes.
- **Affordability:** If required by organizations for specific applications, then purchasing satellite images seems reasonable. However, for research and educational purposes, purchasing the same does not seem viable by any means.

- **Resolution:** This point overlaps with accessibility; such that higher resolution data is usually unavailable free of cost and the data which is available has very poor resolution, which may not serve the purpose.
- **Time factor:** At times even if individuals opt for purchase of satellite images, the time taken to place the order as well as receiving it eventually places a lot of hindrances in the progress of the project for which it is required. Also, most satellites have another disadvantage of revisit time, such that they can only visit a certain geographical area after a span of time.
- **Noise and Interference:** Data wise, satellite images suffer from noise such as speckle, cloud cover, haze, glare and dust. Due to this data has to go through atmospheric and radiometric corrections for them to be useful.

However, in recent times, due to the emergence of the **UAV** or the **Unmanned Aerial Vehicle**, popularly known as the **Drone**, these data issues have largely been taken care of. They may not be cover as much area as a satellite may cover in flight but they ensure frequent revisits, easy availability and high resolution data (Up to an accuracy of 0.7cm at a flying altitude of 50m, PrecisionHawk Lancaster Rev 5) which may have diverse applications in various different domains, including agriculture, mining, urban planning, defense, energy and utilities, emergency response and forestry to name a few.

In addition, they may even venture into areas which may otherwise prove to be hazardous for human beings. They have the capability of staying in the air for prolonged period of time, performing a precise, repetitive raster scan of a region, day-after-day, night-after-night in complete darkness, or, in fog, under

computer control, performing a geological survey, visual or thermal imaging of a region and even measuring cell phone, radio, or, TV coverage over any terrain (UAVS, 2016).

Figure 2.19: PrecisionHawk Lancaster Rev 5 (Lancaster, 2016)



The data being used for this project has been acquired by

PrecisionHawk

Lancaster Rev 3 in 2015

(Further details about the data used in this project

have been attached in

Appendix-I and

Appendix-II of this

project). Acclaimed as Market Leaders in the Unmanned Aerial Platform based Survey Solutions Industry, PrecisionHawk has one of kind equipment and a variety of **field swappable plug-and-play sensors** which include visual, thermal, multispectral and LIDAR sensors. It is the first to have LIDAR sensor on aerial platform in India.

Claimed by many as “the future of data collection”, UAV systems and services are here to stay. With **Amazon** (Stevenson, 2016) and **Dominos** (Sachdeva, 2016) trying their best to integrate UAV service delivery for extending their services, today the drone market is infectiously moving from the defense domain to civilian applications and have the capability to take over the aerial data market in a big way.



METHODOLOGY

3.01 Outline

This section describes the overall methodology of development of the plugin named ‘MLP’ developed based on ANN approach. Here, training and testing dataset have been created keeping in mind the objective of LULC classification.

A FeedForward Multi Layer Perceptron has been implemented and backpropagation algorithm has been used to provide training to the training datasets. As mentioned earlier, Backpropagation is a well-known algorithm for learning of ANN.

The basic structure of the network consists of a three layer neural network, namely an **input layer** comprising of two nodes, a **hidden layer** comprising of three nodes and one **output layer** having one node, i. e. the classified land use land cover map.

In all neural networks the three fundamentals that one must follow are:

1. Build it: Create the skeletal structure of the network with all nodes and their connections well-defined for training.
2. Train it: Using a suitable training algorithm, train the network to perform certain function.
3. Test it: Test the trained network on the testing dataset.

Similarly, the MLP tool, too, uses the same fundamentals for its formation. These modules will be explained in greater details further in this document.

Figure 3.02: Multi Layer Perceptron for MLP

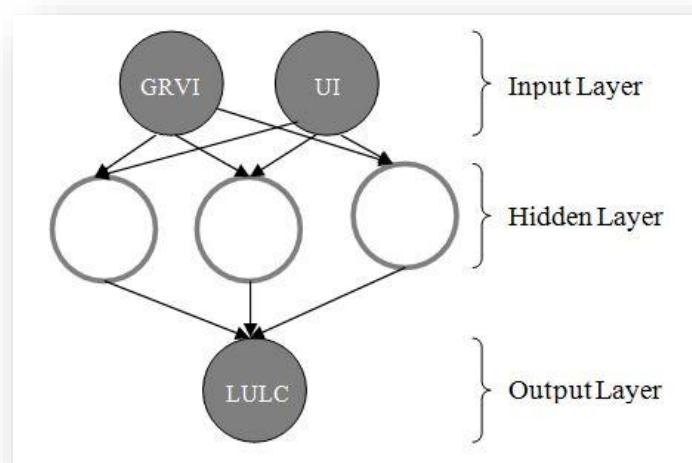
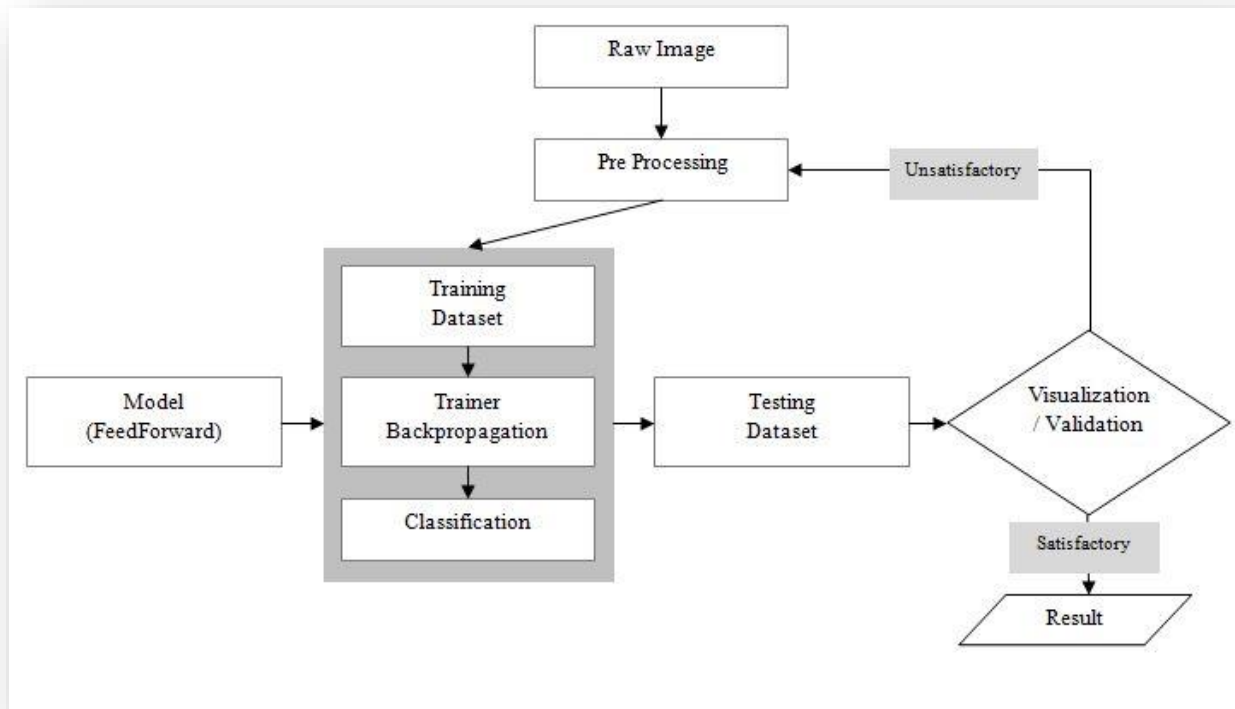


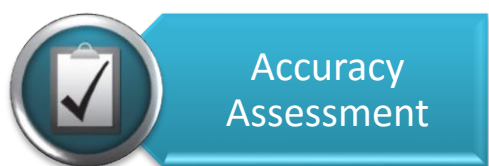
Figure 3.03: Work Flow of the Multi Layer Perceptron in MLP Tool



The above figure depicts a block diagram of the MLP tool consisting of various components i.e. MLP Training, MLP Testing and MLP Working blocks. It shows the flow of data and connectivity among various sub modules of MLP.

3.02 Modules

The development of the tool can be divided into four basic modules. They are as follows:



1. Python Tool Development

a. User Interface (UI) of the Tool:

The basic UI of the tool will be designed keeping in mind the following criteria:

- Ease of Use
- Clarity
- Integration with the software.

b. **Logic Development:** The code for the tool for using Multilayer Perceptron as the Artificial Neural Network for

classification of the dataset is the main module. This module governs the behavior of the tool.

2. Training Set

Another important module in this project is the building of the training set. It has a great bearing on the results because its accuracy will determine the accuracy of the datasets that will be classified using this tool. The more intricate the training, the better the results will be.

3. Accuracy Assessment

In the end, after testing the tool, the resultant classified dataset will be assessed for its accuracy using accuracy assessment techniques such as the Kappa Accuracy Test or the Confusion Matrix.

3.03 Software

The above mentioned modules will be developed using several software, namely:



1. Python Tool Development: This can further be divided into the following:

a. UI Design: The UI of the tool has been designed in “**QT Creator IDE**” which is a fully-stocked cross-platform integrated development environment for easy creation of connected devices, UIs and applications (QT Creator IDE).



b. Logic development: Using the “**NumPy**” library of **Python**, the logic will be developed in the **Notepad++** text editor. Python is a programming language that lets you work more quickly and integrate your systems more effectively (Python).



2. Building Training Sets: The training sets for training the dataset for accurate classification using artificial neural network will be built in **QGIS Wien 2.8.7**. QGIS is a cross-platform free and open-source desktop geographic information system (GIS) application that provides data viewing, editing, and analysis (Wikipedia, QGIS).

3. **Accuracy Assessment:** The accuracy of the dataset will be evaluated using a well-known accuracy test such as the “**Kappa Accuracy Test**”.

3.04 Development

1. **Python Tool Development:** The development of the tool will be elaborated below.

- a. **UI Design:** At first, the basic plugin was created in Quantum GIS 2.7.8 Wien version. For this several initial steps had to be performed, as shown below:

Python Bindings for Qt

Since the plugin was being developed in Python, the python bindings for Qt needed to be installed. For building plugins, the pyrcc4 command-line tool was needed. Since the tool was developed in a Windows platform, the **OSGeo4W network** installer was downloaded and **Express Desktop Install** was chosen. The QGIS package was installed. After installation, the pyrcc4 tool could be accessed via the **OSGeo4W Shell**.

A Text Editor or a Python IDE

Any kind of software development requires a good text editor or an IDE (Integrated Development Environment). For our tool, **Notepad++** editor on Windows was used.

Plugin Builder plugin

This helpful QGIS plugin creates all the necessary files and the boilerplate code for a plugin. This plugin was installed and used.

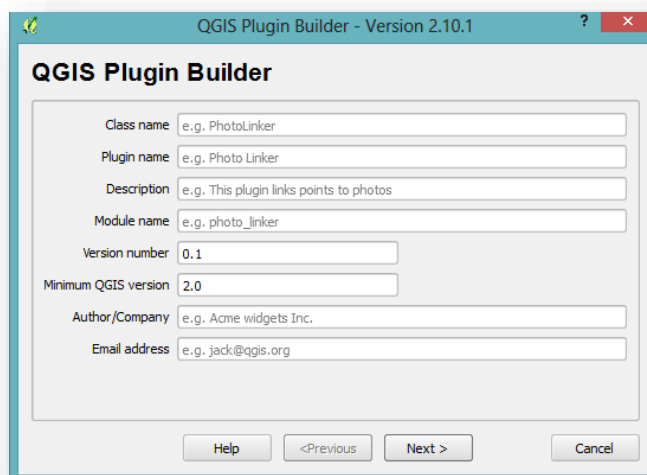
Plugins Reloader plugin

This is another helper plugin which allows iterative development of plugins. Using this plugin, one can change the plugin code and have it reflected in QGIS without having to restart QGIS every time. Even this plugin was installed.

Procedure

- On opening QGIS, the ‘**Plugin Builder**’ was installed from the already available plugins. The Plugin Builder creates a QGIS plugin template for use as a starting point in plugin development. Once installed, it can be found on the list of plugins in our ‘**Plugin**’ drop down menu. Using it, the basic framework of a plugin can be created.

Figure 3.03: Plugin Builder



- On clicking on the Plugin Builder, a window asking for details will appear. The *Class name* will be the name of the Python Class containing the logic of the plugin. This will also be the name of the folder containing all the plugin files. The *Plugin name* is the name under which the plugin will appear in the *Plugin Manager*. A description can be added in the *Description* field. The *Module name* will be the name of the main python file for the plugin. The version numbers were left to default values. The *Text for menu item* value will be how the users will find the plugin in QGIS menu. Name and email address was specified in the appropriate fields. The *Menu* field will decide where the plugin item is added in QGIS. Since our plugin is for raster data, Raster was selected. Next, a directory for the plugin had to be chosen.

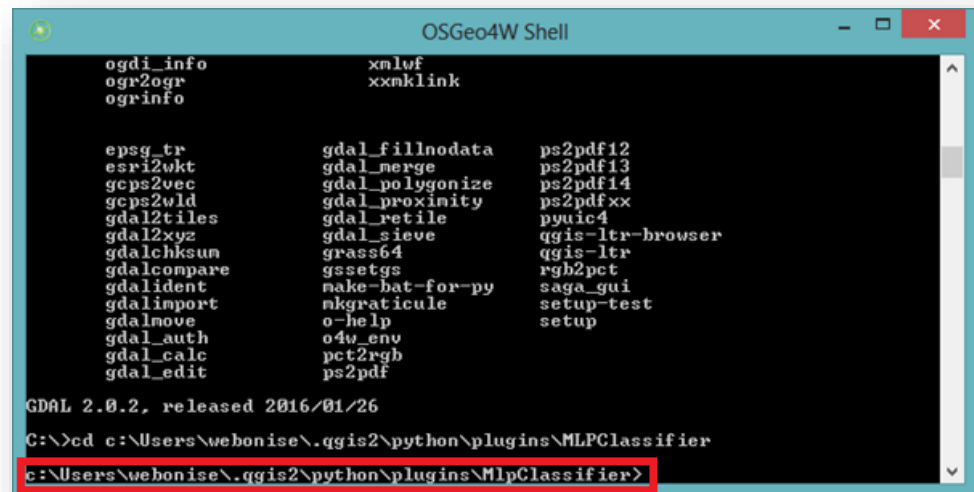
c:\Users\username\.qgis2\python\plugins

Browsing to the QGIS python plugin directory on your computer the respective folder was created. A confirmation dialog box appeared,

confirming the formation of the plugin. This led to the formation of the basic tool and its supporting files.

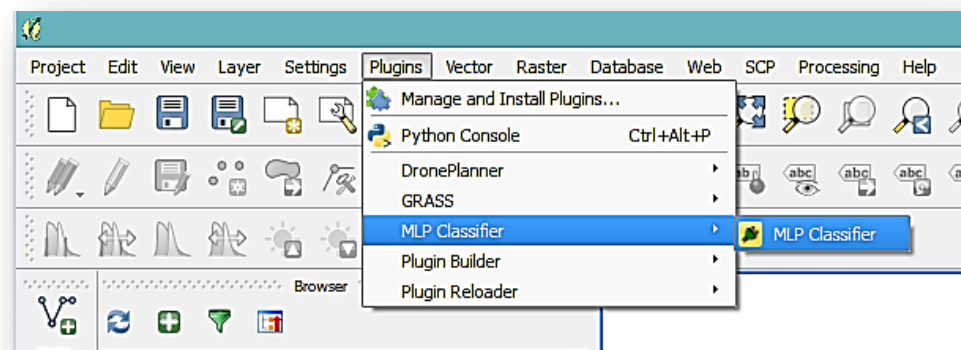
- Before using the newly created plugin, the 'resources.qrc' file that was created by Plugin Builder was compiled by launching the *OSGeo4W Shell*.

Figure 3.04: OSGeo4WShell



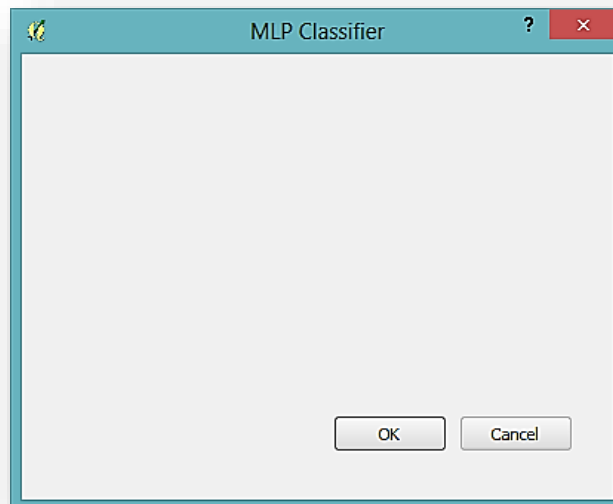
- By browsing to the plugin directory where the output of Plugin Builder was created and typing 'make', the **pyrcc4** command was run that was installed as a part of the Qt bindings for Python.
- Here is the first look of the created plugin:

Figure 3.05: MLP Classifier Plugin: First Look



On clicking on this plugin, as of now, a blank non-functional window will be displayed shown below:

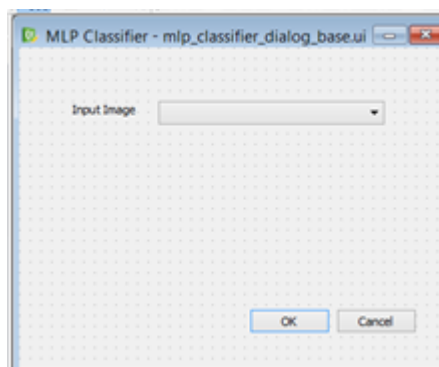
Figure 3.06: MLP Classifier: Non-Functional Window



- In order to design the window of the MLP Classifier plugin, the software ‘**QT Creator**’ was used. The already created file ‘**mlp_classifier_dialog_base.ui**’ was opened from the repository of files created by the Plugin Builder.

Thus, the end product is as follows:

Figure 3.07: The MLP Tool



b. Logic Development

The logic for the MLP tool has been developed using the following libraries:

- **PyBrain:** PyBrain or **Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library**, is a modular Machine Learning Library for Python. Its goal is to offer flexible, easy-to-use yet still powerful algorithms for Machine Learning Tasks and a variety of

predefined environments to test and compare your algorithms (Welcome to Pybrain).

- **Matplotlib:** Matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB®* or Mathematica®), web application servers, and six graphical user interface toolkits (Matplotlib).
- **NumPy:** NumPy is the fundamental package for scientific computing with Python. It contains among other things:
 - A powerful N-dimensional array object
 - Sophisticated (broadcasting) functions
 - Tools for integrating C/C++ and Fortran code
 - Useful linear algebra, Fourier transform, and random number capabilities (NumPy).

As mentioned above, PyBrain facilitates its users to develop neural networks with great ease. PyBrain, as its written-out name already suggests, contains algorithms for neural networks, for reinforcement learning (and the combination of the two), for unsupervised learning, and evolution.

The MLP Tool

For this tool, we created a FeedForward model of Multi Layer Perceptron with Backpropagation training algorithm.

The FeedForward model, as mentioned earlier, consists of three layers, namely an **input layer** comprising of two nodes, a **hidden layer** comprising of three nodes and one **output layer** having one node, i. e. the classified land use land cover map.

The number of nodes in every layer was determined in the following manner:

- **Nodes in the input layer:** The two nodes in the input layer comprise of the two indices used for training the network, i. e., Green-Red Vegetation Index (GRVI) and Urban Index (UI).
- **Nodes in the hidden layer:** The number of nodes in this layer has been determined based on the number of classes that the LULC map will be classified into, i. e., three classes, vegetation, urban and mixed classes.
- **Nodes in the output layer:** The node in this layer is the output classified LULC map which will be the product after the testing dataset has been trained by the MLP network.

The Network

By importing the ‘FeedForward’ from the PyBrain library, the network has been formed. The three different layers in the network have been formed by importing the ‘LinearLayer’ class for the input and output layers and the ‘SigmoidLayer’ class for the hidden layer. In order to establish a connection between these layers, the ‘FullConnection’ class has been imported.

The Training

The training set has been formed by building datasets after importing the ‘SupervisedDataSet’ class. While forming these datasets, the ranges of every class have been specified such that:

```
ds = SupervisedDataSet(2, 1)
ds.addSample((0, 0), (0)),
ds.addSample((5, 5), (1)),
ds.addSample((75, 75), (2)),
```

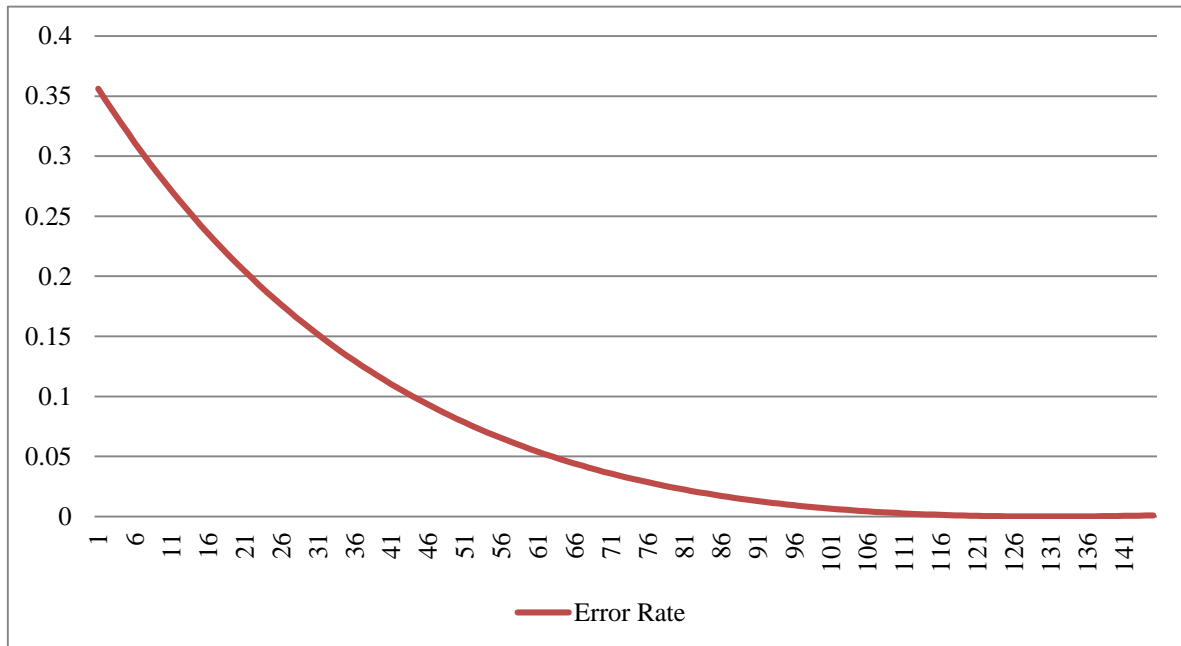
Here, the range of every class is mentioned and the target output is also specified.

Once the training dataset has been formed, we train the network with the same so as to enable the MLP network to accurately classify the raw ortho image into the aforementioned three distinct classes. To train, we use the ‘trainUntilConvergence’ method which will ensure that the network keeps on

training until it achieves the global minimum, or the lowest error rate with reference to the given training and target datasets.

Once the training is done, we print a line graph showing the error rates of our network. This is how it appears:

Figure 3.08: Graphical Representation of the Achieved Error Rate after Training



From the above graph we, can visualize that the error rate converges at 0 after 141 iterations. Now, the net is ready for testing.

The Testing

Once the training is complete and a desirable error rate is achieved, the trained network is tested using 'n.activate()' method. This causes the trained network to classify the given raw ortho image into the three different classes.

2. Building Training Sets

The training setshave been built in **QGIS Wien 2.8.7**. Several small patches of the UAV acquired dataset have been taken for the same reason. Given below are the steps followed for building the training sets:

- Using the **Green-Red Vegetation Index (GRVI)**, three distinct classes could be defined, namely, the urban section comprising of the settlements and the roads,

the vegetation class and the mixed class. The formula to compute GRVI has been given as follows:

$$GRVI = \frac{Green - Red}{Green + Red}$$

- Similarly, the **Urban Index (UI)** makes use of the red and blue band of the RGB sensor data and provides distinct differentiation between the vegetation, urban segments and the mixed class. The formula to compute UI is given as follows:

$$UI = \frac{Red - Blue}{Red + Blue}$$

- Using these two indices, the three major classes have been formed and fed to the training dataset.

Given below are the resulting GRVI and UI rasters after being applied to the orthomosaic of the UAV acquired data.

Figure 3.09: Green-Red Vegetation Index

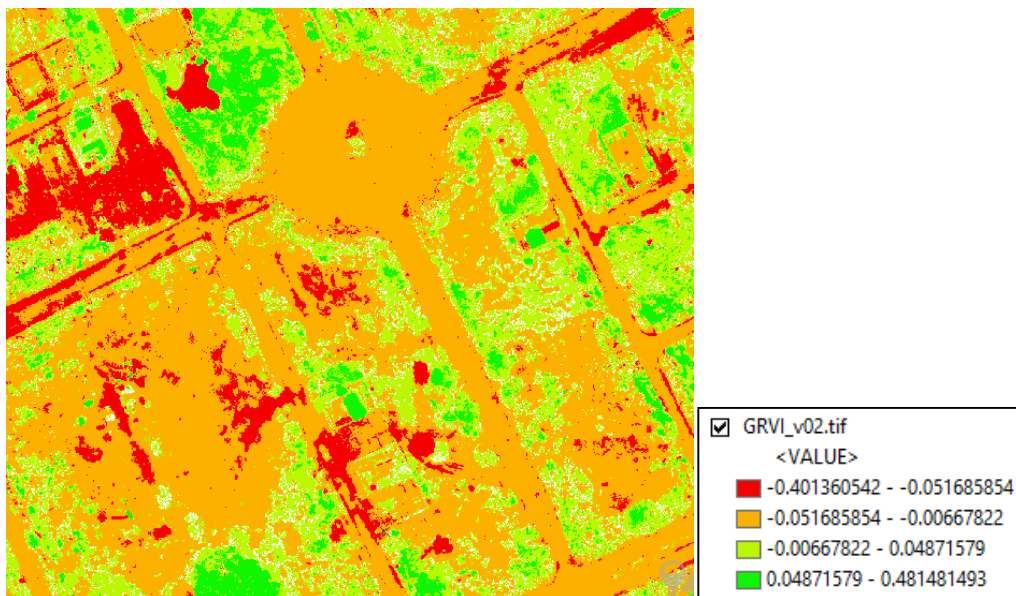
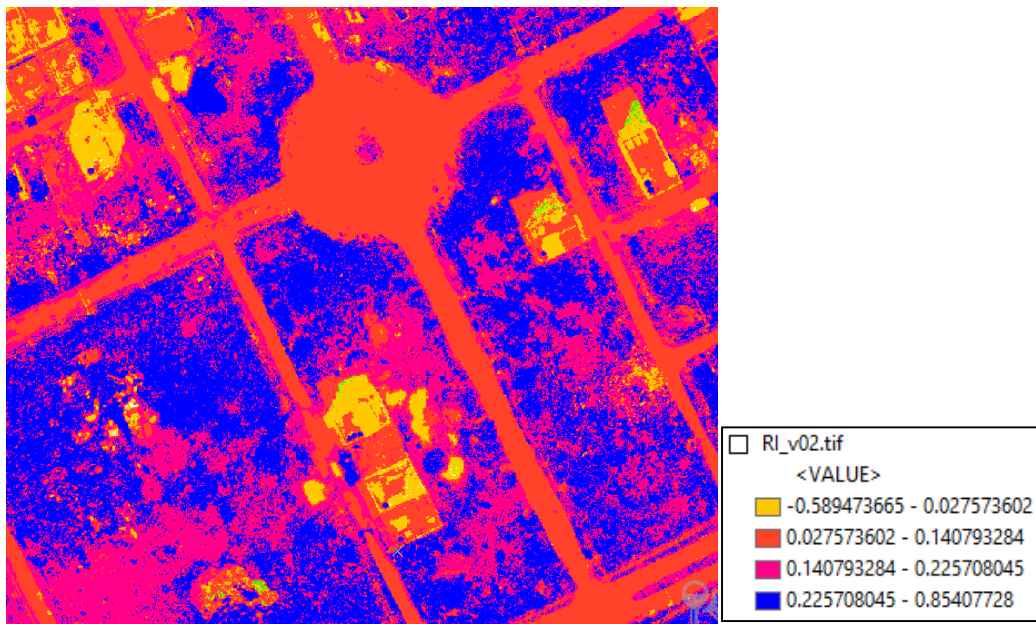


Figure 3.10: Urban Index



These values have further been normalized using the Min-Max Normalization Technique, which applies the following formula:

$$X_{norm} = \frac{X_i - X_{min}}{X_{max} - X_{min}}$$

Thereafter, we get these values within a range of 0 to 1.

Thus, the three classes and their respective ranges have been given below:

Table 3.01: Ranges of Classes

Index	Vegetation		Mixed		Urban	
	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum
GRVI	-0.0067	0.0487	-0.0067	-0.0517	-0.0517	-0.4013
UI	0.2257	0.8541	0.0276	0.1408	-0.5894	0.0276
Class	0		1		2	

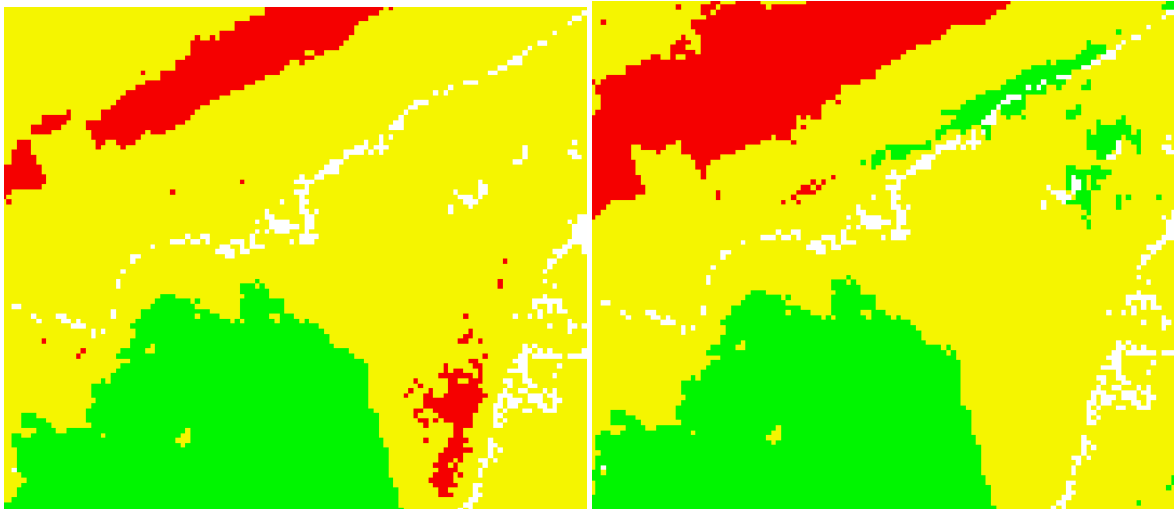
3. Accuracy Assessment

At last we test the accuracy of our MLP tool using the ‘Kappa Accuracy Test’. **Cohen's kappa coefficient** is a statistic which measures inter-rater agreement for qualitative (categorical) items. It is generally thought to be a more robust measure than simple percent agreement calculation, since κ takes into account the agreement occurring by chance.

RESULT

After running the raw dataset through the MLP tool the results that have been acquired are as follows:

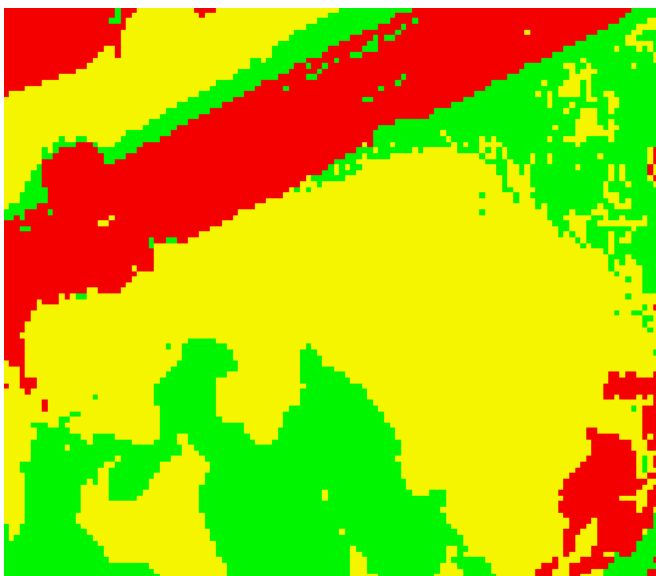
Figure 4.01: Classification of UAV Acquired Dataset with the MLP Tool



With 50 Iterations

With 100 Iterations

Figure 4.02: Manually Classified LULC Map



Legend	
	Vegetation Class
	Mixed Class
	Urban Class

As seen above, with the increase in iterations, the accuracy of classes made by the MLP tool increases. As such we ran the dataset twice through the tool; once with 50 iterations and once with 100 iterations. As the map above denotes, the one with 100 iterations came closest to the one which had been classified manually. Given below is a graph showing the pixel counts in every case:

Figure 4.03: Bar Graph showing the Pixel Count in 50 Iterations, 100 Iterations and Manually Classified LULC Map

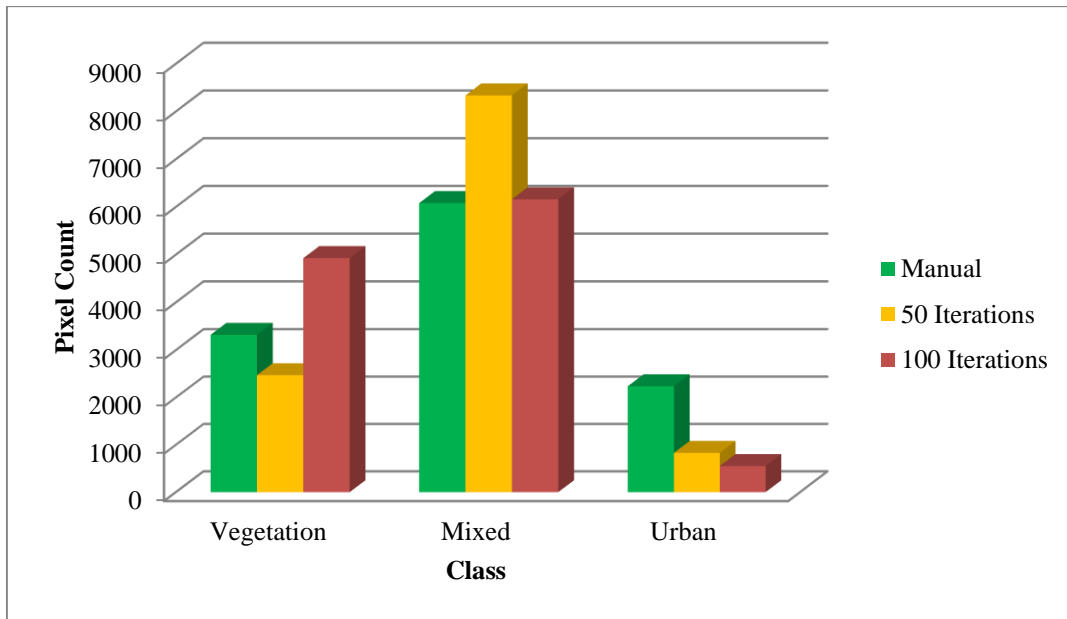


Table 4.01: Pixel Count

	Manual Classification	50 Iterations	100 Iterations
Vegetation	3320	2471	4927
Mixed	6081	8337	6160
Urban	2237	830	551

Thus even through the pixel count we can see that the classification done by the MLP tool in 100 iterations comes closer to the manually classified LULC map as compared to the one with 50 iterations.

Hence, we can say that with greater number of iterations we can increase the accuracy further.

DISCUSSION

With the main goal being removal of manual intervention in the formation of Land Use Land Cover Maps, the MLP tool has by far sorted out a majority of this issue. As shown in the previous section, the classification results have been fairly accurate. Of course, there is vast scope in improvement of its performance and quality. This can be ensured further with intense training of our training sets and more sophisticated neural network structure. For this, more number of training sets will have to be fed and classes will have to be defined in greater detail.

Let us look into the advantages that the GIS industry can plough from this tool.

- **Time efficient:** As is always the case, the LULC map is not an end product. It is further utilized and processed to yield a number of thematic maps. Thus, by accelerating the production of this very product, the production of other products dependent on it can also be accelerated to a greater extent. Hence, saving valuable time.
- **Scope for increased accuracy:** With a much more intense training net, the classification can attain high accuracy results. This will further improve the quality and also increase its productivity.
- **Set standards for classification:** As mentioned in the introduction of this document, one of the greatest disadvantages of manual creation of the LULC map is the differing standards of classification due to differences in individual understanding and knowledge, this can be easily overcome by the MLP tool. Thus, ensuring uniformity and reliability.
- **Increased reproducibility:** As it is a neural network that will be classifying the images and giving the outputs, scope of reproducibility of the LULC map of the same area can be increased greatly. This, too, will help in standardization and quality check of the maps.

Further, using Black Box Optimization, the problems of over fitting can be overcome, ensuring a very efficient net that solves the classification problem.

CONCLUSION

The MLP tool by far, has achieved what it had set forth. However, there is a vast scope for improvement, innovation and optimization for this tool. Backed with Artificial Neural Network, this tool will, in the near future, not only be capable of resolving the resounding LULC classification problems, but will also be able to perform image recognition, feature extraction and pattern recognition.

Keeping what we have covered so far in the functionalities of this tool, I would further go on to say that there is a lot in store for this tool for its future prospects. A few could be as follows:

- **Solve real-life problems:** With the ability to do feature extraction, this tool could help environmentalists and miners in a great way.
- **Ease of use for the user:** The user need not be an expert in Artificial Neural Network. All that will be required from his end would be an input image and with just a few clicks, he will be able to produce an LULC map, which would have otherwise taken him at least 2 days to produce.
- **Integration with all GIS software:** As of now, this tool is only available on QGIS Wien 2.8.7. However, its integration with not only all versions of QGIS but also with different GIS software such as the ArcMap, ERDAS and ENVI.

Thus, with these future prospects in mind, I will be continuing with the development of this tool to make it even more progressive, practical and easier to use.

REFERENCES

1. Anil Kumar Goswami, S. G. (2014). Automatic Object Recognition from Satellite Images using Artificial Neural Network. *International Journal of Computer Applications* .
2. *Artificial Neural Networks Technology*. (n.d.). Retrieved from DOD DACS Home Page: <http://psych.utoronto.ca/users/reingold/courses/ai/cache/neural4.html>
3. *Artificial Neural Networks/Boltzmann Learning*. (2010, November 2). Retrieved from WikiBooks:
https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Boltzmann_Learning
4. *Artificial Neural Networks/Competitive Learning*. (2013, February 3). Retrieved from WikiBooks:
https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Competitive_Learning
5. *BackPropagation*. (n.d.). Retrieved from <http://rfhs8012.fh-regensburg.de/~saj39122/jfroehl/diplom/e-g.html#BackPropagation>
6. Bennamoun, P. M. *Neural Computation*.
7. Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.
8. *Boltzmann machine*. (2016, March 11). Retrieved from Wikipedia:
https://en.wikipedia.org/wiki/Boltzmann_machine
9. Ceccatto, H. D. (1994). Predicting Indian monsoon rainfall: a neural network approach. *Climate Dynamics, vol. 10* .
10. Christos Stergiou, D. S. *Neural Networks*.
11. *Competitive Learning*. (n.d.). Retrieved from
<https://web.stanford.edu/group/pdplab/pdphandbook/handbookch7.html>
12. D. E. Rumelhart, G. E. (1985). *Learning internal representations by error propagation*. ICS Report 8506, California University, San Diego, La Jolla, Institute for Cognitive Science.
13. D. E. Rumelhart, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: The MIT Press, vol. 1.
14. *Erdas Imagine* . (n.d.). Retrieved from DataONE: <https://www.dataone.org/software-tools/erdas-imagine>
15. Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. In K. Fukushima, *Biological Cybernetics*.

16. G. E. Hinton, a. T. (1983). Optimal Perceptual Inference. In a. T. G. E. Hinton, *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. Washington DC.
17. Gales, M. (2015). *Multi-Layer Perceptrons*. University of Cambridge.
18. *Gradient descent*. (2016, February 7). Retrieved from Wikipedia:
https://en.wikipedia.org/wiki/Gradient_descent
19. Grossberg, S. (1976). Adaptive pattern classification and universal recoding: Part I: Parallel development and coding of neural feature detectors. In S. Grossberg, *Biological Cybernetics*.
20. Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. NY: Macmillan, Phi Learning Pvt. Ltd.
21. Hinton, G. E. (2007). *Boltzmann machine*. Retrieved from Scholarpedia:
http://www.scholarpedia.org/article/Boltzmann_machine
22. Hu, J. (2013, April 26). *About Data Mining*. Retrieved from About Data Mining:
<http://www.aboutdm.com/2013/04/history-of-machine-learning.html>
23. J. Vieira, F. M. (1997). *Neuro-Fuzzy Systems: A Survey*. A Study on Video Browsing Strategies, Technical Report, University of Maryland at College Park.
24. J. Yi, R. P. (1996). A neural network model forecasting for prediction of maximum ozone concentration in an industrialized urban area. In *Environmental Pollution*, vol. 92 no. 3.
25. K.Vijayarekha, D. (n.d.). Activation Functions.
26. Kawaguchi, K. (2006). *Backpropagation Learning Algorithm*.
27. L. Bruzzone, S. B. (1997). Classification of imbalanced remote-sensing data by neural networks. In *Pattern Recognition Letters*, vol. 18 no. 11.
28. *Lancaster*. (2016). Retrieved from PrecisionHawk:
<http://www.precisionhawk.com/lancaster>
29. Leverington, D. (2009). *A Basic Introduction to Feedforward Backpropagation Neural Networks*. Retrieved from Texas Tech University:
http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html
30. Lewis, P. (n.d.). *Analogy between human and artificial neural nets*. Retrieved from
<http://users.ecs.soton.ac.uk/phl/ctit/nn/node2.html>
31. M. D. Emmerson, R. I. (1993). Determining and Improving the Fault Tolerance of Multilayer Perceptrons in a Pattern-Recognition Application. In *IEEE Transactions on Neural Networks*, vol. 4 no. 5.

32. MacDonald, F. (2015, June 29). *Scientists have built artificial neurons that fully mimic human brain cells* . Retrieved from Science Alert:
<http://www.sciencealert.com/scientists-build-an-artificial-neuron-that-fully-mimics-a-human-brain-cell>
33. Management Association, I. R. (2013). *Image Processing: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools and Applications*. Information Science Reference (An Imprint of IGI Global).
34. Marzban, G. J. (1996). A neural network for tornado prediction based on Doppler radar derived attributes. *Journal of Applied Meteorology*, vol. 35 .
35. *Matplotlib*. (n.d.). Retrieved from Matplotlib: <http://matplotlib.org/>
36. *Multilayer Perceptron*. (2016). Retrieved from Deep Learning:
<http://deeplearning.net/tutorial/mlp.html>
37. *Multilayer Perceptron*. (2016, April 4). Retrieved from Deep Learning:
<http://deeplearning.net/tutorial/mlp.html>
38. *Multilayer Perceptron in Python*. (2014, October 9). Retrieved from Code Project:
<http://www.codeproject.com/Articles/821348/Multilayer-Perceptron-in-Python>
39. *Neural Network*. (2016). Retrieved from Mu Sigma: http://www.mu-sigma.com/analytics/thought_leadership/caffe-cerebral-neural-network.html
40. *Neural Networks*. (2013, April 6). Retrieved from stanford.edu:
http://ufldl.stanford.edu/wiki/index.php/Neural_Networks
41. Nielsen, M. A. (2015). How the backpropagation algorithm works. In M. A. Nielsen, *Neural Network and Deep Learning*. Determination Press.
42. *NumPy*. (n.d.). Retrieved from Numpy: <http://www.numpy.org/>
43. Prasad S. Thenkabail, P. (2015). *Remotely Sensed Data Characterization, Classification, and Accuracies*. CRC Press, Taylor & Francis Group.
44. *Principles of Data Communications: Media Characteristics*. (n.d.). Retrieved from HN Computing: http://www.sqa.org.uk/e-learning/NetTechDC01CCD/page_39.htm
45. *Python*. (n.d.). Retrieved from Python: <https://www.python.org/about/>
46. *QT Creator IDE*. (n.d.). Retrieved from QT Creator IDE: QT Creator IDE
47. R. D. Reed, R. J. (1999). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA: The MIT Press.
48. R. M. Welch, S. K. (1992). Polar cloud classification using AVHRR imagery - an inter comparison of methods. *Journal of Applied Meteorology*, vol. 31 .

49. Reich, L. (2015, June 06). *PrecisionHawk Media*. Retrieved from PrecisionHawk Media: <http://media.precisionhawk.com/topic/faa-approved-usaa-insurance-drone/>
50. Rojas, R. (1996). *Neural Networks*. Berlin: Springer-Verlag.
51. Sachdeva, S. (2016, March 18). *New Zealand could become first country to use Domino's pizza delivery robot*. Retrieved from Stuff.co.nz: <http://www.stuff.co.nz/business/78022236/new-zealand-could-become-first-country-to-use-pizza-delivery-robot>
52. Santayana, G. (1905-1906). *The Life of Reason* .
53. Stansbury, D. (2016, Spetember 8). *Derivation: Derivatives for Common Neural Network Activation Functions*. Retrieved from The Clever Machine: <https://theclevermachine.wordpress.com/tag/tanh-function/>
54. Stevenson, B. (2016, May 05). *Amazon looks towards UAV delivery service integration*. Retrieved from FightGlobal Aviation Connected: <https://www.flightglobal.com/news/articles/amazon-looks-towards-uav-delivery-service-integratio-424928/>
55. Tag, J. E. (1992). Towards automated interpretation of satellite imagery for navy shipboard applications. *Bulletin of the American Meteorological Society*, vol. 73 no. 7 .
56. *Tutorials Point*. (n.d.). Retrieved from Artificial Intelligence - Neural Networks: http://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm
57. *Types of Neural Nets*. (n.d.). Retrieved from <http://rfhs8012.fh-regensburg.de/~saj39122/jfroehl/diplom/e-12-text.html>
58. UAVS. (2016). *Advantages of UAS*. Retrieved from Unmanned Aerial Vehicle Systems Association: <https://www.uavs.org/advantages>
59. V. Cherkassky, F. M. (1998). *Learning from Data: Concepts, Theory and Methods*. New York: Wiley.
60. Velasquez, G. (1998). *A Distributed Approach to a Neural Network Simulation Program*. El Paso: Master's thesis, The University of Texas .
61. Vinícius Gonçalves Maltarollo, K. M. (January 16, 2013). Applications of Artificial Neural Networks in Chemical Problems. In K. M. Vinícius Gonçalves Maltarollo, *Applications of Artificial Neural Networks in Chemical Problems*.
62. von der Malsburg, C. (1973). *Self-organizing of orientation sensitive cells in the striate cortex*. Kybernetik.

63. Weisstein, E. (2016, March 22). *Hyperbolic Tangent*. Retrieved from Wolfram Math World: <http://mathworld.wolfram.com/HyperbolicTangent.html>
64. Weisstein, E. (2016, March 22). *Sigmoid Function*. Retrieved from Wolfram Math World: <http://mathworld.wolfram.com/SigmoidFunction.html>
65. *Welcome to Pybrain*. (n.d.). Retrieved from Pybrain: <http://pybrain.org/>
66. Wikipedia. (n.d.). *Artificial neural network*. Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Artificial_neural_network
67. Wikipedia. (2016, March 25). *Maxima and Minima*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Maxima_and_minima
68. Wikipedia. (n.d.). *QGIS*. Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/QGIS>
69. Zipser, D. E. (1985). *Feature Discovery by Competitive Learning*. Ablex Publishing.
70. Zoran Sevarac, M. K. *Getting Started With Neuroph*.
71. Zurada, J. M. (1992). *Introduction to Artificial Neural Systems*. Boston: PWS Publishing Company.

GLOSSARY

Activation: A node's level of activity; the result of applying the activation function to the net input to the node. Typically this is also the value the node transmits.

Asynchronous: Process in which weights or activations are updated one at a time, rather than all being updated simultaneously.

Generalization: The ability of a NN to produce reasonable responses to input patterns that is similar, but not identical, to training patterns.

Graphical User Interface: A type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation.

Inhibitory connection: Connection link between two neurons such that a signal sent over this link will reduce the activation of the neuron that receives the signal. This may result from the connection having a negative weight, or from the signal received being used to reduce the activation of a neuron by scaling the net input the neuron receives from other neurons.

Iteration: The act of repeating a process, either to generate an unbounded sequence of outcomes, or with the aim of approaching a desired goal, target or result. Each repetition of the process is also called an 'iteration' and the result of one iteration is used as the starting point for the next iteration.

Java: A general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.

Land cover: The observed (bio)physical cover on the earth's surface.

Land use: The arrangements, activities and inputs people undertake in a certain land cover type to produce, change or maintain it.

Local and Global Minima and Maxima: In mathematical analysis, the maxima and minima (the respective plurals of maximum and minimum) of a function, known collectively as extrema (the plural of extremum), are the largest and smallest value of the function, either

within a given range (the local or relative extrema) or on the entire domain of a function (the global or absolute extrema)

Neuron: Also known as a **neurone** or **nerve cell**. It is an electrically excitable cell that processes and transmits information through electrical and chemical signals.

Perceptron: An algorithm for supervised learning of binary classifiers: functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another.

Synapse: In the nervous system, a **synapse** is a structure that permits a neuron (or nerve cell) to pass an electrical or chemical signal to another neuron.

Synchronous updates: All weights are adjusted at the same time.

Test set: The ensemble of “input-desired” response data used to verify the performance of a trained system. This data is not used for training.

Training epoch: One cycle through the set of training patterns.

Training set: The ensemble of “inputs” used to train the system for a supervised network. It is the ensemble of “input-desired” response pairs used to train the system.

User Interface (UI): The design of user interfaces for machines and software, such as computers, home appliances, mobile devices and other electronic devices, with the focus on maximizing the user experience. The goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals.

Validation set: The ensemble of samples that will be used to validate the parameters used in the training (not to be confused with the test set which assesses the performance of the classifier).

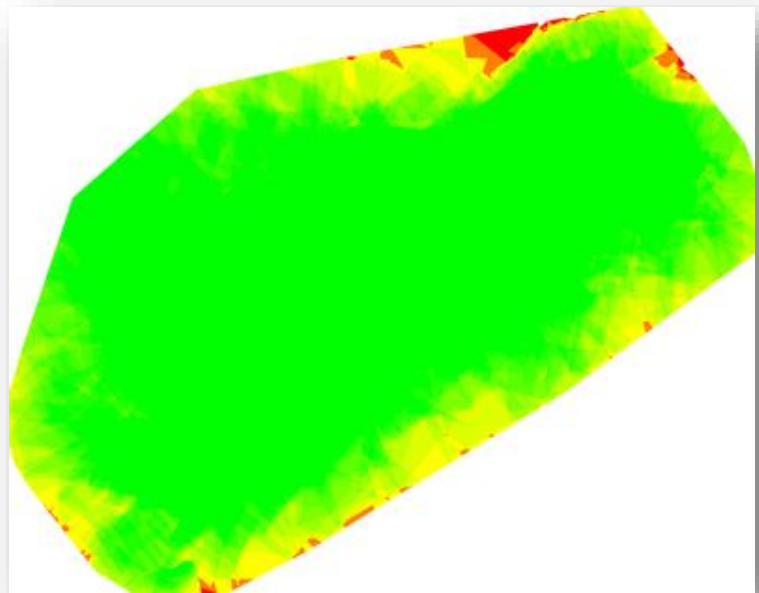
APPENDIX-I: USED DATA

The data used for creating training sets and testing is an orthomosaic of an urban area having an average GSD of 2.43 cm.



Figure 9.01: Orthomosaic of the Area

Figure 9.02: Number of overlapping images computed for each pixel of the orthomosaic.




Number of overlaps:  1 2 3 4 5+

Table 9.01: XYZ Accuracy of the GCPs

GCP Name	Accuracy XY/Z [m]	Error X [m]	Error Y [m]	Error Z [m]	Projection Error [pixel]	Verified/Marked
mtp0 (3D)	0.200/ 0.200	-0.131	0.223	-1.294	1.025	4 / 4
mtp1 (3D)	0.200/ 0.200	0.087	-0.391	0.110	1.651	8 / 8
mtp2 (3D)	0.200/ 0.200	0.093	0.200	0.768	2.348	7 / 7
mtp3 (3D)	0.200/ 0.200	0.400	0.076	0.583	0.571	4 / 4
mtp4 (3D)	0.200/ 0.200	-0.210	-0.073	-0.062	1.145	4 / 4
mtp5 (3D)	0.200/ 0.200	-0.059	-0.019	-0.551	0.971	7 / 7
mtp6 (3D)	0.200/ 0.200	-0.188	-0.012	0.445	1.190	6 / 6
Mean [m]		- 0.001071	0.000612	0.000014		
Sigma [m]		0.198665	0.190737	0.667893		
RMS Error [m]		0.198668	0.190738	0.667893		

APPENDIX-II: UAV HARDWARE SPECIFICATIONS

Table 9.02: Hardware specifications of PrecisionHawk Lancaster Rev 3

Hardware	CPU: Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz RAM: 59GB GPU: Cirrus Logic GD 5446 (Driver: unknown)
Operating System	Linux 3.13.0-61-generic x86_64
Camera Model Name	NIKON1J4_1NIKKOR10mmf/2.8_10.0_5232x3488 (RGB)
Image Coordinate System	WGS84
Ground Control Point (GCP) Coordinate System	WGS84
Output Coordinate System	WGS84 / UTM zone 43N
Keypoints Image Scale	Full, Image Scale: 1
Advanced: Matching Image Pairs	Aerial Grid or Corridor
Advanced: Matching Strategy	Use Geometrically Verified Matching: yes
Advanced: Keypoint Extraction	Targeted Number of Keypoints: Automatic
Advanced: Calibration	Calibration Method: Standard, Internal Parameters Optimization: Leading, External Parameters Optimization: All, Rematch: no