Chapter-4
The cognitive radio (CR) technology appears as an attractive solution to effectively allocate the radio spectrum among the licensed and unlicensed users. With the CR technology the unlicensed users take the responsibility of dynamically sensing and accessing any unused channels (frequency bands) in the spectrum allocated to the licensed users. As spectrum sensing consumes considerable energy, predictive methods for inferring the availability of spectrum holes can reduce energy consumption of the unlicensed users to only sense those channels which are predicted to be idle. Prediction-based channel sensing also helps to improve the spectrum utilization for the unlicensed users.

This chapter emphasizes on the different channel prediction schemes that were being used in order to predict the behaviour of the channel.

4.1 CHANNEL STATE PREDICTION-MEANING

Channel state prediction in cognitive wireless networks involves the analysis of the radio environment and evaluation of certain parameters which can be used to make the best decision for optimal communication. In this context, we can talk of CSI, i.e. Channel State Information which refers to the channel properties that affect the propagation of signal from the transmitter to the receiver.

In cognitive radios, CSI is estimated at the receiver and usually quantized and fed back to the transmitter via feedback channel. The cognitive module at the transmitter end receives the information and based on the history of signal transmitted and received, it trains itself using the machine-learning algorithms. The more comprehensive its test database, the better is it trained to predict the channel behaviour and this allows the cognitive radio to exploit the resources at its disposal.

The CSI acquisition is practically limited by the rate at which the channel conditions change. The available CSI may in general comprise of the instantaneous CSI with some quantization/estimation error and the statistical CSI.
4.2 CHANNEL STATE PREDICTION-NEED

Channel selection is one of the most important functions of the cognitive engine in any cognitive radio. To assign a channel to the cognitive radio, it has to be seen that the channel in question is vacant (spectrum sensing will not be discussed) and it should be free for a longer period of time based on the channel history available in database. We are interested in creating a model of our wireless channel so that we can better understand the communication being carried on in it. Thus, predicting the next state of the channel is very important and we will focus on that aspect.

The CSI makes it possible to adapt transmissions to current channel conditions, which is crucial for achieving reliable communication with high data rates. There are many methods that make use of the CSI and empower the cognitive radio to set up a communication link which can adapt the transmitted signal to the channel characteristics and thereby optimize the received signal to achieve low bit error rates.

4.3 METHODS OF CHANNEL STATE PREDICTION

Evaluating a mobile wireless network is a challenging problem because the quality of the wireless link varies unpredictably over time and space. To have a better understanding of the behaviour of the wireless links and to optimize the protocols at various levels of the network hierarchy, we require cognitive functionalities. The Encyclopaedia of Computer Science [116] discusses a three-point computational view of cognition:

i. Mental states and processes intervene between input stimuli and output responses.

ii. The mental states and processes are described by algorithms.

iii. The mental states and processes lend themselves to scientific investigations.

Moreover, the interdisciplinary study of cognition is concerned with exploring general principles of intelligence through a synthetic methodology termed learning by understanding [117]. It is the usage of this technique for predicting and simulating the channel conditions that plays an important role in understanding network protocols and application behaviour.
4.4 CHANNEL STATUS PREDICTOR STRUCTURE AND PROCEDURE

The channel status predictor is a part of the channel selector. The channel selector is composed of a channel manager, channel environment evaluator and channel status predictor. The figure 4.1 shows the overview of the channel selector.

![Figure 4.1: Overview of Channel Selector](image)

The function of channel environment evaluator is to measure the SNR (or CINR) for a specific channel and evaluate the environment based on the measured value. So, the evaluator gives a grade to that environment. Also, the channel manager combines the results of both a channel status predictor and channel environment evaluator. Thereafter, the manager sorts the total evaluated values and outputs to CE.

Traffic prediction aims at forecasting future traffic as precisely as possible, based on the measurement history [118]. In a CR context the prediction aims at determining idle times in PU traffic to be used by secondary transmissions. In addition, history information gives valuable information to the sensing process. A CR system can abandon some channels after a certain learning time if it decides that the band is used almost all the time. It is not reasonable to waste resources to the bands that cannot offer communication possibilities. The energy efficiency is better if the system concentrates only on channels that seem to have long enough idle times.
We assume that the total available spectrum is divided into multiple primary channels to be sensed and used by cognitive radios. Each channel has its own independent traffic pattern. Figure 4.2 represents the architecture for a predictive cognitive radio system. The CR collects information about the spectrum use in the different channels through spectrum sensing and stores this information into the channel history database in a binary format. Since the traffic patterns of the channels might slowly vary over time, the database should include information only over a limited time interval. Cooperative sensing may be needed to detect primary users reliably in the same area.

In order to predict the channel and to transmit the data, the following methodology is adopted.

1) All channels are sensed and the channel history database is updated with the most recent sensing information. The last sensing result is used to define the current situation in a particular channel. If the channel is free, the channel state (CS) flag is set to 0 and if not, CS = 1.

2) Based on the collected history, the traffic patterns of different channels are classified as stochastic and deterministic ones.

3) Different prediction methods apply to different traffic patterns and the method selection is made following the traffic type classification.

4) The idle time prediction uses information from three sources. The CS flag of channels is checked first. If CS = 1, the predicted idle time is 0 s. If CS = 0, the remaining idle time of these channels is estimated based on the channel history and selected prediction method for that particular channel.

5) If the channel used currently is still free, secondary transmission continues. If not, the CR switches to the channel with the longest expected remaining idle time.

6) Data is transmitted and the system goes then back to the task.
Sensing of primary channels is a periodic sampling process to determine the state (ON or OFF) of the channels at every sampling instant. The outcome of sensing is a binary sequence for each channel. This sequence tells us about the traffic that is ongoing. It has sufficient information to determine the periodicity, distribution of idle and busy times, and occupancy of the channel. Occupancy defines the fraction of the time that the primary user is transmitting in a channel.

Cognitive radio should identify the type of the traffic after a short learning period from the binary sequences gathered during that period. Initially the CR works under the assumption that the ON and OFF times are random in each channel. After the learning period is over, the CR has made a decision about the determinism or randomness of the traffic and can adapt the prediction method.

4.5 NETWORK REPRESENTATIONS

The main network comprises of various channels and users communicate based on synchronous slot arrangement. In the network, let us consider \( N_G \) number of channels where each channel is denoted by \( G_i \), for \( 0 < i \leq N_G \). Every channel is defined by the respective bandwidth represented by \( A_i \), for \( 0 < i \leq N_G \). The channel
remains in a state represented by \( \Lambda \) and the network’s traffic statistics for the channel follow discrete-time Markov process. Let there be \( N_\Lambda \) number of states, where \( N_\Lambda = 2^{N_0} \). The state denoted by \( \Lambda_i(t) \) represent the state value for \( i^{th} \) channel for time slot \( t \). The state of the channel is defined by two values of 0 or 1 representing busy and idle states which can be represented as shown in equation \( (4.1) \).

\[
\Lambda_i(t) = \begin{cases} 
0, & \text{when } i^{th} \text{ channel state is busy} \\
1, & \text{when } i^{th} \text{ channel state is idle}
\end{cases} \quad (4.1)
\]

Example illustrating the three-channel system is given in figure 4.3. Here, it is presumed that spectrum usage statistics of the network remain unaffected for \( T \) slots.

![Figure 4.3: A Sample Path of Spectrum Occupancy for a Three-Channel System](image)

The figure 4.3 shows the network eyeing for spectrum openings in the channels. Two colours of dark grey and light grey represent the channel states. Here, dark grey is used for busy or occupied state and light grey is used for idle or opportunity states. That is in time slot 1, channel 2 is occupied that is \( \Lambda_2(1) = 0 \), \( \Lambda_1(1) = 1 \) and \( \Lambda_3(1) = 1 \). The users may link or depart from the network and can use the spectrum independently without trading local information. In our case, number of channels is taken as five and number of users taken is three.

4.6 TRAINING AND PREDICTION

The user data has to be transmitted through the idle or unoccupied channels in the system. The user does not know the idle channel prior and for accomplishing the task, one of the neural network techniques is employed. Here, the channel is found out depending on the channel state prediction results. A network having \( N_G \)
number of channels represented by $G = G_1, G_2, ..., G_N$, is considered. Here, for the channel $G_i$, the respective bandwidth represented by $\lambda_i$.

The system arrives at a state $A_r$ at any time instant $\tau$ built on probabilities related with the state transitions. Each state is linked to symbols and let it be denoted by $S = \{s_1, s_2, ..., s_N\}$, where $N$ is the number of symbols associated with states. A symbol represented by $Z$ (where $Z \in S$) is emitted by state $S \in A$ depending on some probability distribution for each state transition. The flow diagram of the training and prediction is given in figure 4.4.

![Figure 4.4: Flow Diagram for Channel Status Prediction](image)

The channel states can either be 1 or 0 representing idle or busy respectively. The prediction is carried out to forecast the symbol $Z_{\tau+1}$ based on the set of previous observation. The training procedure is multi-cycle process and in each cycle, fixed numbers of states are processed for training and one state as the target. For example, when $k$ symbols are in consideration, then first $k-1$ symbol are taken as the training feature and the $n$th element as the target feature. It can be represented as shown in equations (4.2) to (4.4).

For 1st cycle:

$$\begin{align*}
\text{For 1st cycle:} & \begin{cases}
Z_1, Z_2, ..., Z_{k-1} \text{ as training features} \\
Z_k \text{ as target feature}
\end{cases} \\
\text{equation (4.2)}
\end{align*}$$

Similarly for second cycle:

For 2nd cycle:

$$\begin{align*}
\text{For 2nd cycle:} & \begin{cases}
Z_2, Z_3, ..., Z_k \text{ as training features} \\
Z_{k+1} \text{ as target feature}
\end{cases} \\
\text{equation (4.3)}
\end{align*}$$
Generalizing:

For nth cycle \( \left\{ Z_n, Z_{n+1}, \ldots, Z_{n+k-2} \right\} \) as training features
\( Z_{n+k-1} \) as target feature

(4.4)

After the training process, the probability of \( Z \) being followed by busy or idle at the \( \tau + 1 \) instant is computed. It is performed by calculating the joint probabilities of \( P(Z,1 \mid \sigma) \) and \( P(Z,0 \mid \sigma) \). Here \( \sigma \) is the set of parameters taken up by GS-LM to have maximum probability. Let the predicted value at instant \( \tau + 1 \) be represented as \( \tilde{Z}_{\tau+1} \). The prediction value is defined as given in equations (4.5) and (4.6)

\[
\text{if } P(Z,1 \mid \sigma) \geq P(Z,0 \mid \sigma); \text{ THEN } \tilde{Z}_{\tau+1} = 1 \quad (4.5)
\]

\[
\text{if } P(Z,1 \mid \sigma) < P(Z,0 \mid \sigma); \text{ THEN } \tilde{Z}_{\tau+1} = 0 \quad (4.6)
\]

In this chapter we study, analyze and review a few channel prediction schemes for spectrum sensing.

4.7 HIDDEN MARKOV MODEL (HMM) BASED CHANNEL STATUS PREDICTOR

The Hidden Markov Model (HMM) [119] is a powerful statistical tool for modeling generative sequences that can be characterized by an underlying process generating an observable sequence. HMM is used for modeling & analyzing time series or sequential data in various fields today, such as automatic speech recognition, cryptanalysis, natural language processing, computational biology, bioinformatics etc. With its prior knowledge, HMM is concerned about the unobserved sequence of hidden states and the corresponding sequence of related observation. And a Hidden Markov Model (HMM) is the most widely used method to apply machine learning to data that is represented as a sequence of observations over time.

Consider a system having \( N \) states, the set of states can be denoted as \( S = \{ S_1, S_2, \ldots, S_N \} \). At a time instant \( t \), the system enters state 'q;' depending on some probabilities associated with the state transitions. If the state transitions follow
the Markov property, then the probability of a state transition can be expressed as shown in equation (4.7).

\[ P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots ) = P(q_t = S_j | q_{t-1} = S_i) \]  \hspace{1cm} (4.7)

Suppose that the states are associated with \( M \) discrete symbols, and the set of symbols is denoted as \( V = \{ v_1, v_2, \ldots, v_M \} \). After every state transition, a symbol \( O_t (\in V) \) is emitted by state \( q_t (\in S) \) depending on some probability distribution. Suppose that only the symbol sequence is observable while the state sequence is hidden, this gives rise to the HMM. Therefore, the HMM can be formally stated as a statistical model in which the observed process is assumed to be generated in response to another stochastic process which is hidden and follows the Markov property.

In order to model the HMM, it is necessary to specify the following:

- the number of symbols, \( M \)
- the number of states, \( N \)
- the observation sequence, \( O = \{ O_1, O_2, \ldots, O_T \} \)
- the state transition probabilities, \( a_{ij} = P(q_t = S_j | q_{t-1} = S_i) \) subject to the conditions \( a_{ij} \geq 0 \) and \( \sum_{j=1}^{N} a_{ij} = 1 \)
- the symbol emission probabilities, \( b_j(v_m) = P(O_t = v_m | q_t = S_j) \) subject to the conditions \( b_j(v_m) \geq 0 \) and \( \sum_{m=1}^{M} b_j(v_m) = 1, 1 \leq j \leq N \).
- initial state distribution, \( \pi = \{ \pi_1, \ldots, \pi_N \} \), where \( \pi = P(q_1 = S_i) \) and satisfies the conditions \( \pi_i \geq 0 \) and \( \sum_{i=1}^{N} \pi_i = 1 \).

The HMM can be denoted by the notation \( \lambda = [\pi, A, B] \), where \( A \) is a \( N \times N \) state transition matrix containing the probabilities \( a_{ij} \) where \( i \) denotes the rows and \( j \) denotes the columns, and \( B \) is a \( N \times M \) emission matrix containing the probabilities \( b_j(v_m) \) where \( j \) denotes the rows and \( m \) denotes the columns.
4.7.1 HMM predictor

The HMM prediction scheme is illustrated in figure 4.5. Consider the following sequence of channel occupancies \( \{O_1, O_2, \ldots, O_T, O_{T+1}\} \) where the channel statuses busy and idle are denoted by 1 and -1, respectively. The objective of the HMM predictor is to predict the symbol \( O_{T+1} \) based on the past \( T \) observations. To predict the symbol \( O_{T+1} \), the HMM should be able to generate the observation sequence \( O = \{O_1, O_2, \ldots, O_T\} \) with maximum likelihood probability. Hence, the parameters \( \lambda = [\pi, A, B] \) are adapted to maximize the likelihood probability of generating the observation sequence, i.e., maximize the probability \( P(O|\lambda) \).

Once training is completed, the joint probability of observing the sequence \( O \) followed by a busy slot or an idle slot at instant \( T + 1 \) is calculated. In other words, the joint probabilities \( P(O, 1|\lambda) \) and \( P(O, -1|\lambda) \) are calculated. The slot occupancy at instant \( T + 1 \) is predicted according to decision rule given in equation (4.8)

\[
\text{if } P(O, 1|\lambda) \geq P(O, -1|\lambda) \text{ then } \hat{O}_{T+1} = +1 \\
\text{if } P(O, 1|\lambda) < P(O, -1|\lambda) \text{ then } \hat{O}_{T+1} = -1
\]  

(4.8)

where \( \hat{O}_{T+1} \) is the predicted value.

\[ \text{Figure 4.5: HMM Predictor Training and Prediction Flow Diagram} \]
4.7.2 HMM training

The Baum Welch Algorithm (BWA) [119] is an iterative method to estimate the HMM parameters $\lambda = [\pi, A, B]$ such that the probability $P(O|\lambda)$ is maximized.

To estimate the parameters $\lambda = [\pi, A, B]$, the BWA defines the following variables:

- **Forward variable** $a_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i|\lambda)$, for $1 \leq i \leq N$
- **Backward variable** $\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T, q_t = S_j|\lambda)$, for $1 \leq i \leq N$
- $\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda)$ for $1 \leq i, j \leq N$, the probability of being in state $S_i$ at instant $t$ and in state $S_j$ at instant $t + 1$ given the observation sequence $O$ and the model $\lambda = [\pi, A, B]$
- $\gamma_t(i) = P(q_t = S_i | O, \lambda)$ for $1 \leq i \leq N$, the probability of being in state $S_i$ at instant $t$ given the observation sequence $O$ and the model $\lambda = [\pi, A, B]$.

The estimation formulas for the parameters of the model $\lambda = [\pi, A, B]$ are expressed in terms of the variables $\xi_t(i, j)$ and $\gamma_t(i)$ as shown in equations (4.9) to (4.11)

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (4.9)$$

$$b_t(v_m) = \frac{\sum_{t=1}^{T} o_m \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \quad (4.10)$$

$$\pi_i = \gamma_1(i) \quad (4.11)$$

In Equation (4.9), the numerator represents the expected number of transitions from state $i$ to state $j$ over duration $T - 1$, while the denominator represents the expected number of times a transition is made from state $i$. The numerator in Equation (4.10) represents the expected number of transitions from state $i$ and symbol $v_m$ is observed after the transitions. In the Equations (4.9) to (4.11), $\xi_t(i, j)$ and $\gamma_t(i)$ are calculated using the equations given in (4.12) and (4.13).
The forward and backward variables in Equations (4.12) and (4.13) are calculated recursively.

The forward variable \(a_t(i)\) is calculated from the equations (4.14) to (4.16)

**Initialization:**
\[
a_1(i) = \pi_i b_i(1), \quad 1 \leq i \leq N
\]  
(4.14)

**Recursion:**
\[
a_{t+1}(j) = \left[ \sum_{i=1}^{N} a_t(i) a_{ij} \right] b_j(O_{t+1}), \quad 1 \leq i \leq N
\]
\[
1 \leq t \leq T - 1
\]  
(4.15)

**Termination:**
\[
P(O/\lambda) = \sum_{i=1}^{N} a_T(i), \quad 1 \leq i \leq N
\]  
(4.16)

The backward variable is calculated from the equations (4.17) and (4.18)

**Initialization:**
\[
\beta_T(i) = 1, \quad 1 \leq i \leq N
\]  
(4.17)

**Recursion:**
\[
\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad 1 \leq i \leq N
\]
\[
T - 1 \geq t \geq 1
\]  
(4.18)

Equation (4.16) provides the formula for the probability of observing the sequence \(O\) given the model \(\lambda = [\pi, A, B]\). The parameters \(\lambda = [\pi, A, B]\) are re-estimated using the equations for a maximum of \(K\) iterations or till the maximum \(P(O/\lambda)\) is reached.

To avoid the possibility of underflow, the forward and backward variables are scaled (or normalized) while calculating from equations (4.14) to (4.18). The scaling operation [119] can be expressed as shown in equations (4.19) and (4.20):
where \( \tilde{\alpha}_i(i) \) and \( \tilde{\beta}_i(i) \) represent the scaled forward and backward variables respectively.

The scaling coefficient \( C_s \) can be calculated as given in equation (4.21)

\[
C_s = \frac{1}{\Sigma_{s=1}^{\infty} \sigma_s(t)}
\]

Accordingly, the variables \( \xi(i, j) \) and \( \gamma(i) \) are calculated by replacing \( \alpha_i(i) \) and \( \beta_i(i) \) with their scaled versions \( \tilde{\alpha}_i(i) \) and \( \tilde{\beta}_i(i) \) in equations (4.12) and (4.13).

Because of using the same coefficients for the forward and the backward variables, the estimation formulas given in equations (4.9) to (4.11) remain unchanged. However, the likelihood probability \( P(O|\lambda) \) cannot be truly estimated due to the scaling operation. Instead, the log likelihood probability \( \log(P(O|\lambda)) \) can be calculated as given in equations (4.22) and (4.23).

\[
\log(P(O|\lambda)) = \log \left[ \sum_{i=1}^{N} \alpha_T(i) \right] = \log \left[ \sum_{i=1}^{N} (\tilde{\alpha}_T(i) / (\prod_{s=1}^{T} c_s)) \right]
\]

\[
= \log \left[ \sum_{i=1}^{N} (\tilde{\alpha}_T(i)) - \log [\prod_{s=1}^{T} c_s] \right]
\]

As \( \sum_{i=1}^{N} (\tilde{\alpha}_T(i)) = 1 \) at any time instant \( t \) due to normalization, then the equation (4.22) reduces to equation (4.23)

\[
\log(P(O|\lambda)) = - \sum_{s=1}^{T} \log(c_s)
\]

Thus, when scaling is used, the modified BWA estimates the model \( \lambda = [\pi, A, B] \) such that the log likelihood probability of generating the observation sequence \( O \), \( \log(P(O|\lambda)) \), is maximized.
During the training, the observation sequence length $T$ is set to 80, the number of states $N$ in the HMM is set to 10, and the maximum number of iterations $K$ is set to 10. Initial values of the probabilities for state transition $(A)$, state emission $(B)$, and state initial distributions $\pi$ are chosen arbitrarily.

After training, the joint probabilities $P(O,1|\lambda)$ and $P(O,-1|\lambda)$ are calculated using the equations (4.24) and (4.25)

$$P(O,1|\lambda) = \sum_{i=1}^{N} \alpha_{T+1}^1 (i)$$  \hspace{1cm} (4.24)\]

$$P(O,-1|\lambda) = \sum_{i=1}^{N} \alpha_{T+1}^{-1} (i)$$  \hspace{1cm} (4.25)\]

where $\alpha_{T+1}^1 (i)$ and $\alpha_{T+1}^{-1} (i)$ are given by equations (4.26) and (4.27)

$$\alpha_{T+1}^1 (i) = [\sum_{j=1}^{N} \alpha_{T}^0 (j) a_{ji}] b_{i} (O_{T+1} = 1), 1 \leq i \leq N$$  \hspace{1cm} (4.26)\]

$$\alpha_{T+1}^{-1} (i) = [\sum_{j=1}^{N} \alpha_{T}^0 (j) a_{ji}] b_{i} (O_{T+1} = -1), 1 \leq i \leq N$$  \hspace{1cm} (4.27)\]

Based on $P(O,1|\lambda)$ and $P(O,-1|\lambda)$, the predicted value is found using equation (4.8). When scaling is used, the log likelihood probabilities $\log(P(O,1|\lambda))$ and $\log(P(O,-1|\lambda))$ are calculated similar to equation (4.8) and the channel status at the time instant $T+1$ is predicted in favour of the maximum of the two values.

4.8 ARTIFICIAL NEURAL NETWORK (ANN) BASED CHANNEL STATUS PREDICTOR

The spectrum occupancy in most licensed user systems encountered in reality is non-deterministic in nature. Hence, it is appropriate to model such traffic characteristics using non-linear adaptive schemes. Neural networks are non-linear parametric models which create a mapping function between the input and output data. The advantage of neural networks over statistical models is that it does not require prior knowledge of the underlying distributions of the observed process. In CRNs, it is difficult to obtain the statistics of channel usage by the primary users. Therefore, the neural networks offer an attractive choice for modeling the channel status predictor. Once the neural networks are trained, the computational complexity is significantly reduced. The neural network model, Multi layer Perceptron (MLP), has been used in various applications, e.g., system identification and time series prediction [120,121].
4.8.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are neuroscience inspired computational tools that are trained using input-output data to generate a desired mapping between input stimulus to the targeted output [122]. Artificial Neural Networks [123] provide a powerful tool to analyse, model and predict. The benefit of neural networks lies that the neural networks are data driven self-adaptive methods and that they are universal functional approximations. Artificial neural networks (ANN or simply NN) are made up of artificial neurons interconnected with each other to form a programming structure that mimics the behavior and neural processing (organization and learning) of biological neurons.

The artificial neuron is the unit model of the ANN structure which gets input from all neighboring neuron and gives an output depending on its synaptic weight and activation functions. Activation function like Tan-sig function, Sigmoid function are used. Figure 4.6 shows a single neuron as processing unit.

![Figure 4.6: Single Neuron as Processing Unit](image)

A neural network consists of a pool of simple processing units, the 'neurons'. Inter-neuron connections are called synapses. Each synapse is associated with a synaptic weight. These weights are used to store knowledge which is acquired from the environment. An artificial neuron is as shown in figure 4.7 [124].
The mathematical representation of the neuron model is as shown in equations (4.28) and (4.29)

\[ u_k = \sum_{j=1}^{m} w_{kj} x_j \]  
\[ y_k = \varnothing(u_k + b_k) \]  

where \( x_1, x_2, x_3, \ldots, x_m \) are the inputs to the network, \( w_{kl}, w_{kj}, \ldots, w_{km} \) are the weights to the net where \( u_k \) is the linear combiner and \( b_k \) is the bias and \( y_k \) is the output of the neuron. A NN has to be configured such that the application of a set of inputs produces the desired set of outputs. This can be achieved by properly adjusting the weights \( w_{jk} \) of the existing connections among all \((j,k)\) neuron pairs. This process is called learning or training.

The NN learns the input–output mapping by a stepwise change of the weights with the objective to minimize the difference between the actual and desired output. In the next step the actual output vector is compared with the desired output. A learning algorithm gives the change \( \Delta w_{ij}(t) \) in the weight of a connection between neurons \( i \) and \( j \) at time \( t \) so that a network can efficiently perform a specific task. Error values are assigned to each neuron in the output layer. The error values are
back-propagated from the output layer to the hidden layers. The weights are changed so that there is a lower error for a new presentation of the same pattern. As a result of this procedure, the weights on the connections between neurons are adjusted so as to encode the actual knowledge of the NN. At that time, the NN can be used for the purpose that it was initially set up for.

The neurons are arranged in three layers the input layer, hidden layer and the output layer. Learning is accomplished by adjusting the weights till the difference between the desired output and the output of the network is negligible. The number of layers in the hidden layer and the number of neurons in each layers are changed on a hit and trial basis. Learning can be generally distinguished between supervised and unsupervised learning (with reinforcement learning being also an option). In supervised learning, the NN is fed with teaching patterns and trained by letting it change its weights according to some learning rule, the so called back-propagation rule.

Here a training algorithm is applied which helps the neural network to learn like back propagation algorithm, resilient back propagation, Levenberg algorithm, conjugate gradient, Quasi Newton algorithm etc. its variants are used. Once trained these models can be used directly to provide instant results for the purpose for which it has been developed [122]. On the contrary, in unsupervised learning the NN discovers features of the input data in a statistical manner by developing its own ways of classifying the input irritants.

The design and analysis of a Neural Network has the following phases:

- Generation of data;
- Training of the neural network;
- Testing.

The generation of the data is intended to decide the inputs and outputs which are to be trained. Training of the input parameters is done in such a way that the output of the neural network is same as the desired output. Hence this structure can be used further to predict the output in such an environment. The ANN has been
used to make the cognitive radio to learn from the environment and take decision [125, 126]. The network has two modes of operation; the training mode with validation and the testing mode.

Once the network weights and biases are initialized, the network is ready for training. The training process requires a set of network inputs and corresponding target outputs. During training the weights and biases of the network are iteratively adjusted to minimize the network performance function. The performance function for feed forward networks is mean square error (MSE) which is the average squared error between the network outputs and the target outputs. The training adjusts the connection's weights accordance with a learning algorithm, after obtaining an output from the network and comparing it with a wished output. In the testing mode data set are used to measure the performance of the network after the training.

Neural networks are normally nonlinear models and can offer the basis for setting up classification rule and performing statistical analysis. The topology of a NN plays an important role for its achievable performance. Depending on the pattern of connections that a NN uses to propagate data among the neurons, it can be classified into one over two basic (non exhaustive) categories. (a) Feed-forward NNs, where data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. (b) Recurrent NNs that contain feedback connections, which are connections extending from outputs of neurons to inputs of neurons within the same or previous layers.

In back propagation training algorithm, the training begins with random weights and each output unit computes error function during training that is used to adjust the weight to each neuron [127,128]. It is also known as steepest descent algorithm as it uses the gradient of the performance function to determine the change in weights to minimize the performance function. Equation (4.30) represents for the one step of iteration of back propagation algorithm.

\[ W_{k+1} = W_k - \alpha g_k \]  

(4.30)

where \( W_{k+1} \) is the new update weight vector, \( \alpha \) is the learning parameter and \( g_k \) is the current gradient.
The three layer feed-forward network as shown in figure 4.8 can be trained with the different back propagation algorithms considering tansigmoidal function as the activation function. The well accepted back propagation algorithms which are used to train the mobile radio classifier are as follows [125,128,129]:

a) Variable Learning Rate  
b) Resilient Back Propagation (RP)  
c) Polak-Ribiere Conjugate Gradient  
d) Quasi-Newton Algorithms  
e) Levenberg-Marquardt (LM)

In the conventional steepest descent algorithms, the learning rate set at a constant value throughout the training. But it is very crucial aspect to select a proper value of learning rate. It is also not realistic to determine the optimal setting for the learning rate before training. The performance of the steepest descent algorithm is enhanced in variable learning rate algorithm by changing the learning rate during the training process. It keeps the learning step size as large as possible maintaining the stable learning. Figure 4.8 gives a three layer feed-forward network [123].

![Figure 4.8: A Three Layer Feed-Forward Network](image)
Let the network input to \( i^{th} \) unit in \((k+1)^{th}\) layer be represented as in equation (4.31)

\[
N_{e}^{k+1}(i) = \sum_{j=1}^{N} w^{k+1}(i, j)z^{k}(j) + Ba^{k+1}(i)
\]

Where, the output of the \( k^{th} \) unit \( (z^{k}(j)) \) can be defined as given in equation (4.32)

\[
z^{k}(j) = F_{k+1}[N_{e}^{k+1}(i)]
\]

The network is to learn relations between the input-output pairs \{(\( y_{1}, \tau_{1} \), \( y_{2}, \tau_{2} \), \ldots, \( y_{q}, \tau_{q} \))\}. The performance index \((PI)\) for the network is given by equation (4.33)

\[
PI = \frac{1}{2} \sum_{r=1}^{R} (\tau_{r} - z_{r}^{N_{t}})^{T}(\tau_{r} - z_{r}^{N_{t}}) = \frac{1}{2} \sum_{r=1}^{R} e_{r}^{T}e_{r}
\]

Here, \( z_{r}^{N_{t}} \) is the output of for \( r^{th} \) input and \( e_{r} = \tau_{r} - z_{r}^{N_{t}} \) is the error for \( r^{th} \) input. The performance index is approximated by the steepest descent rule in back propagation which can be defined as in equation (4.34).

\[
PI = \frac{1}{2} e_{r}^{T}e_{r}
\]

Total sum of squares is substituted by the squared errors for a single input-output pair in steepest-descent approximation as shown in equations (4.35) and (4.36)

\[
\Delta w^{k}(i, j) = -\alpha \frac{\partial PI}{\partial w^{k}(i, j)} = -\beta \Phi^{k}(i)z^{k-1}(j)
\]

\[
\Delta z^{k}(j) = -\beta \frac{\partial PI}{\partial Ba^{k}(i)} = -\beta \Phi^{k}(i)
\]

Where, \( \Phi^{k}(i) = \frac{\partial PI}{\partial N_{e}^{k}(i)} \), \( \beta \) is the learning rate and \( \Phi \) is the sensitivity of the performance index. It can also be shown that sensitivities satisfy the relation as shown in equation (4.37)
Initially, propagate the input forward using equations (4.31) and (4.32) and subsequently, propagate the sensitivities back using (4.37). Finally, update the weights and offsets using equations (4.35) and (4.36).

4.8.2 Levenberg–Marquardt (LM) method

The Levenberg Marquardt (LM) algorithm is an approximation to the Newton method used for training ANNs. Levenberg-Marquardt back propagation algorithm is used for training the network [81, 123]. The numerical optimization techniques based Levenberg-Marquardt is the fastest and powerful method for training of moderate-sized feed-forward neural networks [130]. This optimization technique is more powerful than standard Back propagation Neural Network (BPNN). LM algorithm is very efficient and fast having also a quite good global convergence property [123]. The LM algorithm is an iterative technique that locates the minimum of a multivariate function that is expressed as the sum of squares of nonlinear real valued functions. It is fast and has stable convergence.

The Levenberg–Marquardt algorithm [131, 132] was independently developed by Kenneth Levenberg and Donald Marquardt. In the artificial neural-networks field, this algorithm is used for training networks. The LM algorithm can be thought of as a combination of the steepest descent method and the Gauss–Newton algorithm which inherits the speed advantage of the Gauss–Newton algorithm and the stability of the steepest descent method.

During training, for the networks, the weights and bias values are to be updated according to Levenberg–Marquardt optimization (trainlm). Training automatically stops when generalization stops improving, as indicated by an increase in the Mean Square Error (MSE) of the validation samples and are to be updated according to a gradient descent momentum and an adaptive learning rate method (traingdx). The Mean Squared Error (MSE) is the average squared difference between outputs and targets. Lower values are better while zero means no error. Finally, the Mean Squared Error (MSE) is to be used as a metric for measuring the neural network's performance.
The flow diagram of LM algorithm is as shown in the figure 4.9.

![Flow Diagram of Levenberg-Marquardt Training Algorithm](image)

**Figure 4.9: Flow diagram of Levenberg-Marquardt training algorithm**

Let the function to be minimized be denoted as $F_n(v)$ and it is minimized with respect to the parameter vector denoted as $v$ which can be given by equation (4.38)

$$\Delta v = -[\nabla^2 F_n(v)]^{-1} \nabla F_n(v)$$  \hspace{1cm} (4.38)

Here, $\nabla^2 F_n(v)$ is the Hessian matrix and $\nabla F_n(v)$ is the gradient matrix. The gradient matrix is taken as the sum of squares of the error function which can be calculated from equations (4.39) to (4.42)

$$F_n(v) = \sum_{i=1}^{N} e_i^2(v)$$  \hspace{1cm} (4.39)

$$\nabla^2 F_n(v) = J^T(v)J(v) + D(v)$$  \hspace{1cm} (4.40)
Where, $J()$ is the Jacobian matrix, and

\[
D(v) = \sum_{i=1}^{N} e_i(v) \nabla^2 e_i(v)
\]

\[
\Delta v = [J^T(v) J(v) + \mu I]^{-1} J^T(v) e(v)
\]

The algorithm can be explained in a brief as follows. Initially, all the inputs are given to the network and corresponding network outputs are calculated. Then the sum of squares of errors over all inputs $F_n(v)$ and the Jacobian matrix $J$ is calculated to solve $\Delta v$. The sum of squares of errors is recalculated using $v + \Delta v$. If this new sum of squares is less than that earlier computed, then $\mu'$ is reduced by $\beta$ and updated $v = v + \Delta v$ and go back to initial step. Here $\beta$ is the learning rate and $\mu'$ is the step size. If the sum of squares is not decreased, then increase $\mu'$ by $\beta$, and go back to step to solve for $\Delta v$. The algorithm is assumed to have converged when the normal of the gradient is less than some predetermined value, or when the sum of squares has been lessened to error goal. The key step in this algorithm is the computation of the Jacobian matrix. For the neural network problem the terms in the Jacobian matrix can be computed by a simple modification to the back propagation algorithm.

A drawback of the LM approach is that the ANN must have only a single output, but this can be overcome by implementing multiple networks [133].

4.9 GRAVITATIONAL SEARCH ALGORITHM

Heuristic algorithms mimic biological or physical processes. One of the newest heuristic algorithms that have been inspired by the physical laws is Gravitational Search Algorithm (GSA) [134]. In GSA; Newtonian laws of gravity and motion are applied to find the optimum solution by a set of agents called masses [135].

The gravitational search algorithm [136] is the one of the latest nature inspired algorithm to solve the optimization problems based on the Law of gravity. Many researchers has applied the gravitational search algorithm on large numbers of
problems because it requires only two parameters and having ability to find near global optimum solution and provides better results as compare to other nature inspired algorithms. The gravitational search algorithm is based on Newton’s theory. Newton’s law of gravity states that every particle attracts another article by means of some gravitational force [84].

The gravitational force between two particles is directly proportional to the product of their masses and inversely proportional to the square of the distance between them [137]. In proposed algorithm particles considered as objects and their performance has evaluated with their masses. In GSA, each particle has associated with four specifications: particle position, its inertial mass, active gravitational mass and passive gravitational mass. The position of particles provides the solution of problem while fitness function is used to calculate the gravitational and inertial masses.

In gravitational search [136], initial population is generated which is evaluated using the fitness function. Gravitational constant is later found out. Then, compute the gravitational masses, acceleration and the force. Update the particle velocity and position. Continue the process till the stopping criteria is met.

**Gravitational search procedure is given below:**

**Step 1: Agents initialization:**

The positions of the ‘s’ number of agents are initialized randomly as shown in equation (4.43)

\[ X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n), i=1,2,\ldots,s \]  
(4.43)

where \( X_i^d \) is position of the \( i^{th} \) mass in the \( d^{th} \) dimension and ‘\( n \)’ is dimension of the search space.

**Step 2: Fitness evolution and best fitness computation**

For minimization or maximization problems, the fitness evolution is performed by evaluating the best and worst fitness for all agents at each iteration is as shown in equations (4.44) and (4.45).
Minimization problems:

\[
\text{best}(t) = \min_{j \in \{1, \ldots, s\}} \text{fit}_j(t)
\]

\[
\text{worst}(t) = \max_{j \in \{1, \ldots, s\}} \text{fit}_j(t)
\]

Maximization problems:

\[
\text{best}(t) = \max_{j \in \{1, \ldots, s\}} \text{fit}_j(t)
\]

\[
\text{worst}(t) = \min_{j \in \{1, \ldots, s\}} \text{fit}_j(t)
\]

where \( \text{fit}_j(t) \) represents the fitness value of the \( j^{th} \) agent at iteration \( t \), \( \text{best}(t) \) and \( \text{worst}(t) \) represent the best and worst fitness at iteration \( t \).

Step 3: Gravitational constant \( (G) \) computation:

Gravitational constant \( G \) is computed at iteration \( t \) [135] from equation (4.46)

\[
G(t) = G_0 e^{-\alpha t/T}
\]

\( G_0 \) and \( \alpha \) are initialized at the beginning and will be reduced with time to control the search accuracy. \( T \) is the total number of iterations.

Step 4: Masses of the agents’ calculation:

Gravitational and inertia masses for each agent are calculated at iteration \( t \) using the equations (4.47) and (4.48). Here we consider \( \text{fit}_i(t) = \text{fit}_i = M_i = M \) and \( i = 1, 2, \ldots, s \).

\[
q_i(t) = \frac{\text{fit}_i(t) - \text{worst}(t)}{\text{best}(t) - \text{worst}(t)}
\]

\[
M_i(t) = \frac{q_i(t)}{\sum_{j=1}^{s} q_j(t)}
\]
where $M_{ai}$ and $M_{pi}$ are the active and passive gravitational masses respectively, while $M_{ii}$ is the inertia mass of the $i^{th}$ agent.

**Step 5: Accelerations of agents' calculation:**

Acceleration of the $i^{th}$ agents at iteration $t$ is computed using equation (4.49)

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)} = \sum_{j \in \text{kbest}, j \neq i} \text{rand}_j G(t) \frac{M_j(t)}{R_{ij}(t)} + \varepsilon \left( x_j^d(t) - x_i^d(t) \right)$$  \hspace{1cm} (4.49)

Where $F_i^d(t)$ is the total force acting on $i^{th}$ agent calculated as given in equation (4.50)

$$F_i^d(t) = \sum_{j \in \text{kbest}, j \neq i} \text{rand}_j F_{ij}^d(t)$$  \hspace{1cm} (4.50)

$F_{ij}^d(t)$ is computed using the following equation (4.51)

$$F_{ij}^d(t) = G(t) \frac{M_{ai}(t) M_{pi}(t)}{R_{ij}(t)} + \varepsilon \left( x_j^d(t) - x_i^d(t) \right)$$  \hspace{1cm} (4.51)

$F_{ij}^d(t)$ is the force acting on agent $i$ from agent $j$ at $d^{th}$ dimension and $t^{th}$ iteration. $R_{ij}(t)$ is the Euclidian distance between two agents $i$ and $j$ at iteration $t$. $G(t)$ is the computed gravitational constant at the same iteration while $\alpha$ is a small constant.

**Step 6: Velocity and positions of agents:**

Velocity and the position of the agents at next iteration $(t+1)$ are computed using the following equations (4.52) and (4.53)
where rand\textsubscript{i} and rand\textsubscript{j} are two uniformly distributed random numbers in the interval \([0, 1]\), \(\epsilon\) is a small value, \(R_{ij}(t)\) is the Euclidean distance between two agents \(i\) and \(j\), defined as \(\|X_i(t), X_j(t)\|\); \(k_{best}\) is the set of first \(K\) agents with the best fitness value and biggest mass, which is a function of time, initialized to \(K_0\) at the beginning and decreasing with time. Here \(K_0\) is set to \(s\) (total number of agents) and is decreased linearly to 1.

Step 7: Repeat steps 2 to 6

Steps 2 to 6 are repeated until the iterations reach their maximum limit. The best fitness value at the final iteration is computed as the global fitness while the position of the corresponding agent at specified dimensions is computed as the global solution of that particular problem.

The flow chart of the GSA technique is given in figure 4.10 below:

![Flowchart of Gravitational Search Algorithm (GSA)](image)

Figure 4.10: Flowchart of Gravitational Search Algorithm (GSA)
As it is mentioned earlier, the multi layered feed forward neural networks are trainable. So, many different kinds of training algorithm can be used to gain optimum adjustable parameters for the corresponding multi-layered perceptron. In some cases which are called "ill-conditioned problems", the traditional training algorithms cannot determine the adjustable parameters (weights and biases) properly [138]. Using gravitational search algorithm such difficulties can be eliminated.

4.10 MULTI USER RANDOM CHANNEL SENSING

The basic existing scheme is called random scheme, where the SU has no idea of the history of each channel and randomly chooses one accessible channel at the beginning of each time slot.

A random channel selection scheme corresponds to a situation when only instantaneous information of the channel conditions is known. A cognitive radio senses the spectrum and picks up randomly one channel among all available ones into use. The same channel is used as long as it is available. When switching is required, the next channel selection is selected randomly using the uniform distribution.

Such an approach can result in a bad channel selection since the system randomly selects channels that may be heavily used by primary users (PU) if that channel happened to be available during the sensing time called as random process. This may cause frequent service disruptions for secondary users since they have to refrain from transmission, and result in interference to primary users.