2012

A reinforcement-based brain computer interface to translate EEG patterns into commands

Abhimanyu Sheshashayee
Bard College

A reinforcement-based brain computer interface to translate EEG patterns into commands

Senior Project submitted to

The Division of Science, Mathematics, and Computing

Of Bard College

by

Abhimanyu Venkatraman Sheshashayee

Annandale-on-Hudson, New York

May 2012

Acknowledgements

The author would like to thank Professor Sven Anderson, Professor Frank Scalzo, Professor Keith O'Hara, and the many friends and colleagues who helped and contributed to this project.

Dedicated to R., C., and T., for all the joy you brought to my world.

Abstract

A Brain Computer Interface (BCI) is a computer-based system that attempts to translate thoughts into computational commands. Thought is represented by the brain-activity of the user, which is recorded using electroencephalography (EEG) or magnetoencephalography (MEG). The system learns to identify input patterns and to associate them with computer commands.

In this study, a BCI is designed that includes a pattern recognizer (PR) for feature extraction, and a hill-climbing optimizer for translating signals into commands.

Evaluation of the pattern recognizer demonstrates the ability to correctly recognize patterns that vary by $\pm 13\%$ in spatial scale or with up to $\pm 2.5\%$ noise. Both creation and identification of patterns is 95% accurate for input data having no more than 2% noise, degrading to approximately 40% accuracy for 3% noise. Increasing the threshold of spatial scale tolerance resulted in approximately the same net error, but with an increase in the ratio of type 1 errors to type 2 errors. Similarly, decreasing the threshold of spatial scale tolerance resulted in a decrease in the ratio of type 1 errors to type 2 errors. A decrease in error was elicited when the system was given enough memory and data that the signal processor could compensate for noise by generating large numbers of memories.

Contents

Introduction

Neurons in the brain perform a variety of specialized cellular functions and interactions. When sensing, communicating with one another, or controlling muscles, neurons propagate ionic waves across their membranes, generating electrical and magnetic fields. These fields can be recorded using electroencephalography (EEG) and magnetoencephalography (MEG). The sum total of all of these electromagnetic signals represents part of brain activity. Brain activity is dependent on the types, numbers, locations, and metabolic levels of different neurons. Thus, brain activity can be considered to be a representation of thought (Wolpaw, et al., 2002).

A brain computer interface (BCI) is a system that converts thoughts into computational commands. This involves a computational system responding to specific patterns of brain activity by issuing desired commands. In essence, a BCI is a system that bypasses the nervous system to allow control of external devices using signals generated by the brain (Anderson & DeVries, 2010).

Research into BCI technology is, for most of the part, still in its infancy. Although only dating from the 1970s, BCI research has nevertheless pioneered many fascinating discoveries in a relatively short time. These have included recreating stimuli from the visual field of cats (Stanley, et al., 1999), and simulated motor control of a robotic arm by macaque monkeys (Carmena, et al., 2003).

BCI research has used both intrusive and non-intrusive mechanisms to build communication pathways between computational systems and the brain. In invasive brain-computer interfacing, electrodes are surgically inserted into neural tissue to record activity of single and multiple neurons. In non-invasive brain computer interfacing, the electrical fields (EEG) or magnetic fields (MEG, fMRI) generated as a result of brain activity are used instead.

This data is then interpreted by computational systems, which attempt to respond to the acquired information appropriately (Friedman, et al., 2010). Modern brain-computer interfaces, such as the Emotiv EPOC, record and interpret large quantities of EEG data in order to respond to a small set of thought-processes of the user (Ramírez-Cortes, et al., 2010).

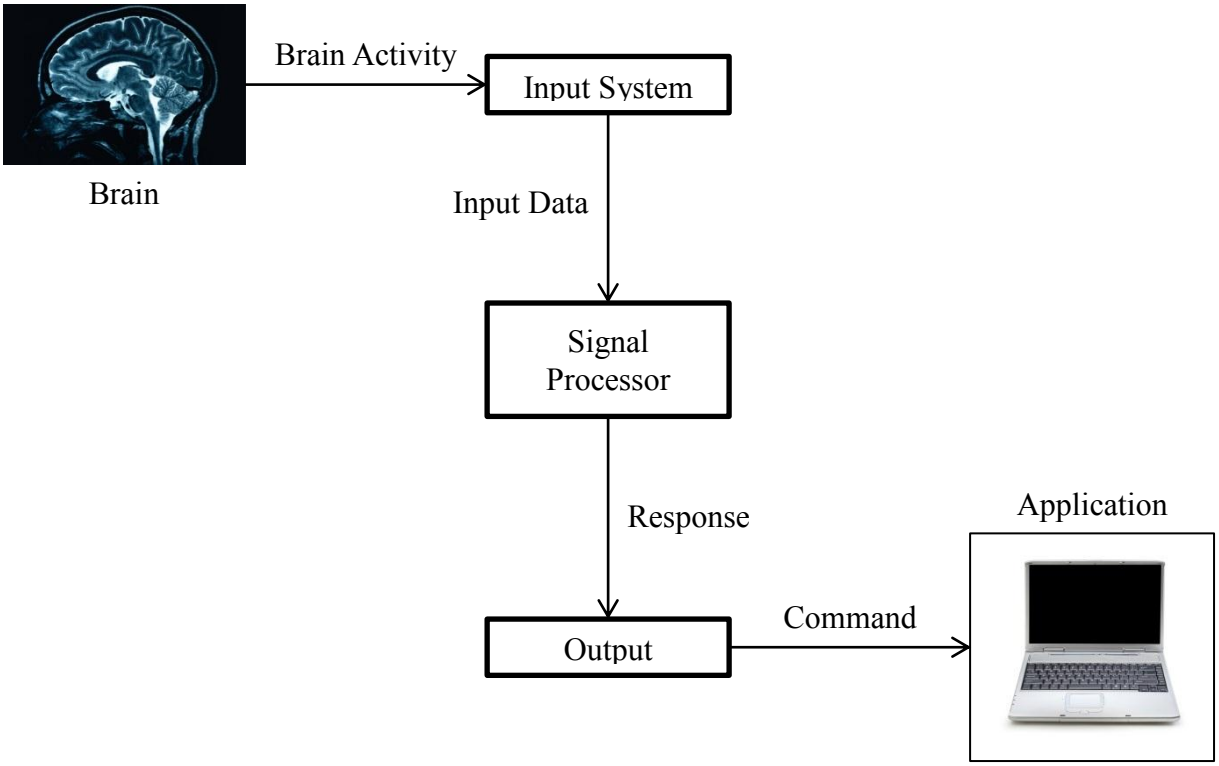


Figure 1 – Structure of a brain-computer interface.

The goal of this study is to design a BCI, and to begin the construction and analysis of its components. A BCI has three primary components (Figure 1). The first is the input system. The input system obtains input data, which is representative of brain activity. The second component is the signal processor (SP). The input data is processed, and an appropriate response is generated. Finally, the third component is the output system, which uses the SP's response to return computational commands. (Wolpaw, et al., 2002)

The input system of the BCI can use tools such as EEG and MEG to represent the magnitudes of electrical and magnetic fields observed at different locations around the cranium. These magnitudes are expressed in units of voltage and tesla, respectively. The input data is in the form of an array of numerical values, with each element representing the magnitude of the recorded field at a specific electrode on the cranium.

The obtained data is then analyzed, by identifying specific features of the signal and associating those features with certain responses. The translation can be performed in a number of ways: clustering algorithms, spatiotemporal filtering, frequency processing, independent component analysis, and reinforcement learners can be used to evaluate brain activity data (Lotte, et al., 2010).

The output system maps the resultant responses from the SP onto a set of application-specific commands. The nature of the map is dependent on the control system that the BCI implements. A proportional control system will parse the elements of the SP's response through one or more functions, the output of which determine the magnitudes of the commands to be performed. Alternatively, a hysteresis control system will only trigger a command if one or more elements of the SP's response exceed a specific threshold. Whatever the case, the commands to be performed are finally relayed to the application (Anderson & DeVries, 2010).

An example of the implementation of a BCI would be a control system for an automated wheel-chair. This system would use EEG as the input system, a clustering-algorithm based SP, and an output system that controls the motion of the wheelchair, with commands such as move forward, move backward, turn left, and turn right. The input provided to the SP would include the input data obtained from the input system, as well as any training parameters that might indicate when the SP is supposed to learn a new mental directive from the user of the system and when it

is supposed to identify previously learned mental directives. The SP would generate a response, which would be sent to the output system of the BCI.

The user would have to be seated in the wheelchair, wearing the EEG headset. In order to train the system to respond to the user's thoughts, the user would have to undergo a brief training procedure, wherein the user is prompted for the mental directive that they want to represent each action. For this example, if the user wanted to train the system to perform the command 'turn left', then the user would specify that the system enter training mode. Then, the user could think of a specific mental directive that the user wanted to map to the command turning left. Suppose that that mental directive was the visualization of an arrow pointing left. So now, the user would imagine an arrow pointing left, so the BCI would record the user's EEG activity while the user was imagining the arrow and map that recording to a specific response.

Once the training is complete, the user can control the motion of the wheelchair by experiencing the thoughts that were mapped onto the commands, allowing the BCI to interpret those mental directives, and return the respective commands. So, if the user imagined an arrow pointing left, the EEG data from the user would be fed to the SP. Now, if the SP successfully identified similarities between the new stream of data from the user and the stored recording of the user imagining an arrow pointing left, then it would return whatever response had been mapped to the stored recording. In this case, the response would be the string "left", which would be sent to the output system. Having received the response "left", the output system would perform the appropriate command: cause the motor controlling the right wheel of the wheelchair to turn forwards for a second, while the left wheel is stationary.

The primary challenge for any BCI design is that the recording of brain activity is subject to a variety of uncontrollable factors: from ambient electromagnetic fields, to the complexity of the

electromagnetic fields being generated by the body. Thus, the input data contains a great deal of stochastic noise. Moreover, there is also very little uniformity in the structure of the input data across different individuals. That is, each individual has different patterns of brain activity that need to be analyzed and evaluated individually (Wolpaw, et al., 2000). Even so, the resolution of input data is not a limiting factor to the effectiveness of a BCI (Wolpaw, 2010). As will be shown, the operation and performance of the SP is the most significant contributing factor to the performance of the BCI.

Signal Processing

The SP of a BCI obtains input data from the aforementioned input system, analyzes that data, and finally returns a response, which is ultimately mapped onto a set of computational commands. The number of inputs and their ranges is pre-determined by the application, as is the number of possible responses and the possible magnitudes of those responses.

The SP receives a steady stream of input data. While a majority of this data might be meaningless, there are some parts of the data that require a specific response. The goal of the SP is to determine when the input data warrants response, as well as the appropriate response for the same (Wolpaw, et al., 2002).

Thus, the operation of a SP involves two essential parts (Figure 2). The first part is feature extraction, which identifies and isolates the important features of the input data stream. The primary purpose of feature extraction is to recognize when a response is required from the SP. Now, when the user of the BCI experiences a thought for which the BCI is to return a command, the SP should return the response that begets that command. Since brain activity is representative of thought, and the input data is a subset of brain activity, the experiencing of a thought can be considered to be a pattern of brain activity, which can in turn be considered to be a sequence of input data points. Hence, the feature extractor must identify specific sequences of input data points, for which a response is generated. Feature extraction also usually allows the system to represent the raw input data in forms that are both memory-efficient as well as more conducive to rapid analysis. The identified features are compiled into a ‘feature vector’.

The second part of signal processing is translation, which involves using the feature vector to determine an optimal response. Having determined that a response is required for some part of the input data stream, the system approximates the ideal response to that given feature.

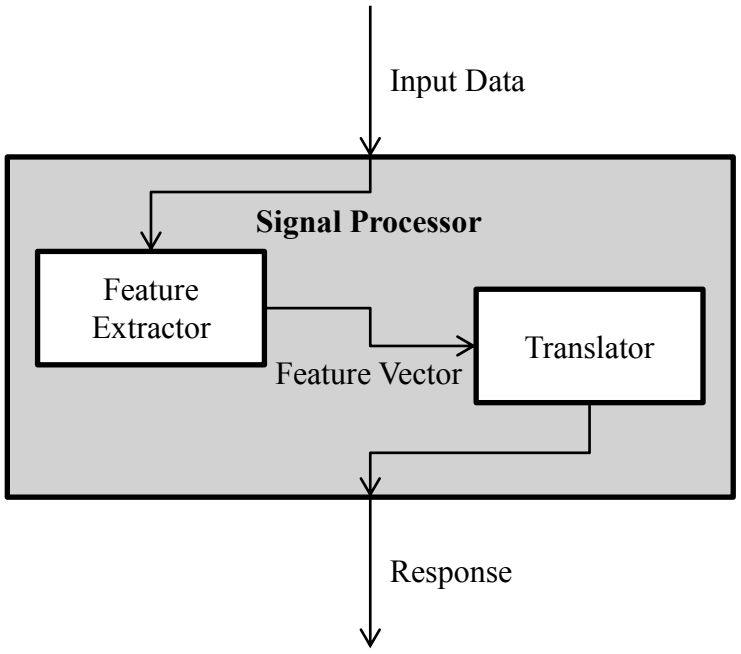


Figure 2 – Structure of a signal processor.

Different systems implement different types of signal processing. Lotte, et al. (2010), describes a signal processing algorithm, comprised of five parts. The first part is a pre-processing step, which modifies the input signal to reduce noise and enhance different elements of the input data. This is followed by feature extraction – which identifies aspects of the input data such as the power of signals in specific frequency bands – in order to prepare the feature vector. The feature vector is then classified, using a variety of different methods, such as linear discriminant analysis. Following this, the classified feature vector is translated into a response, which will later be converted into a command. Finally, the system is given feedback on its performance. Usually, feedback is in the form of reward, which is a value that represents a measure of the correctness of a past response. Reward need not represent how appropriate the current response is, since feedback provided by a user will most likely be delayed. However, reward can be used by a reinforcement learning SP to adjust future responses.

OpenViBE is an integrated development environment (IDE); a software application used to develop BCI programs. Thus, OpenViBE does not have a fixed algorithm for the various components of the SP. Rather, the IDE allows for a the development of different types of algorithms to be used for feature extraction and translation. However, the aforementioned publication (Lotte, et al., 2010) lists a couple of examples, each of which implements their own algorithms, from pre-processing to feedback. One example builds a very simple BCI system that responds to imaginary movements of the subject's left hand. The pre-processor in this system enhances specific EEG frequency bands (μ (~8 Hz to ~13 Hz) and β (~13 Hz to ~30 Hz)) over the motor cortices. This is followed by the use of a technique called logarithmic Band Power for feature extraction. These features are classified using linear discriminant analysis, a response is generated, and feedback is obtained.

Another type of SP uses an artificial neural network to extract features (Zhang, et al.). A back-propagating network, with a three-layer structure (12 input, 22 hidden, and 2 output nodes) is used to generate feature vectors. The feature vectors are then parsed through a support vector machine in order to classify them. Finally, the classification of the feature vectors is used by the translator to determine the resultant response.

The BCI designed in this study implements a reinforcement learning SP. Since the SP uses reinforcement learning, the system requires some form of feedback. This feedback is provided to the system as reward that is appended to the input data. The SP has two modules: a pattern recognizer (PR) that handles feature extraction, and a hill-climbing optimizer that performs the translation of the feature vector into a response.

The operation of the SP occurs in iterations, each of which corresponds to a discrete time step. Changes to system states are event driven: an iteration is triggered when a new input is

obtained. When a new iteration begins, the system receives both the new input, as well as the time t at which the input was obtained. This is then parsed by the pattern recognizer, which identifies patterns in the input data. These patterns are then used by the hill-climbing optimizer to predict a response that maximizes the expected reward for the system. Finally, the response is returned, and the iteration is complete (Figure 3).

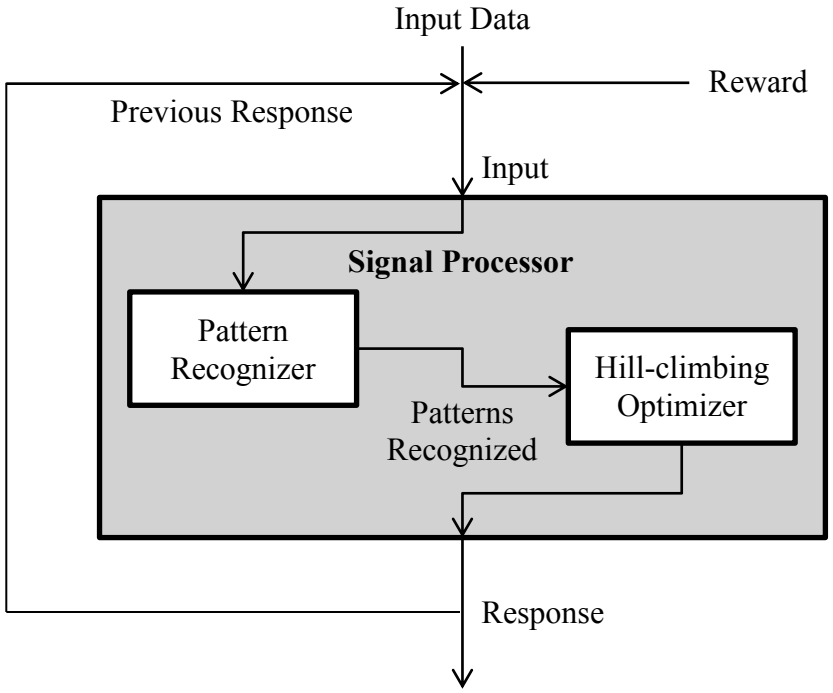


Figure 3 – Structure of the BCI SP designed in this study.

The input is a vector of real values between 0 and 1. Each element of the input vector represents the magnitude of activity of a single receptor, from 0% to 100%. The input is composed of three distinct parts. The first is the input data, the second is the previous response, and the third is the reward. The data represents the portion of the input that is obtained from the input system of the BCI. The previous response is just the response generated by the SP at the end of the last iteration. The previous response is looped back into the input in order to provide

the system with proprioceptive feedback. This becomes especially salient during the translation stage of the signal processing. Finally, the reward represents the external feedback, quantifying how appropriate past responses have been. The system's goal is to maximize the reward obtained.

The PR identifies the occurrence of specific sequences of inputs (hereafter an input pattern) in the stream of inputs (Figure 4). Due to the aforementioned stochastic noise that is present in the input data, the PR has to recognize data points in the stream of inputs which trace a function of the same shape as the input pattern, when plotted against time. Thus, the PR has to first construct a best-fit representation of both the input pattern and stream of inputs, and then determine if the former representation occurs within the latter (Figure 5).

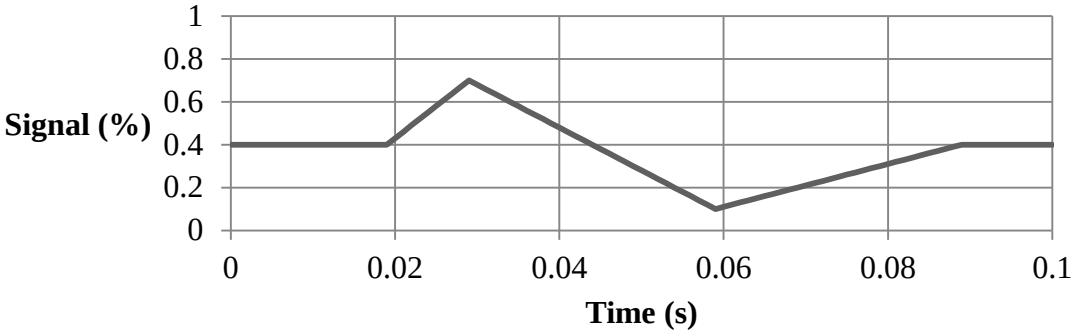


Figure 4 – An input pattern to be identified.

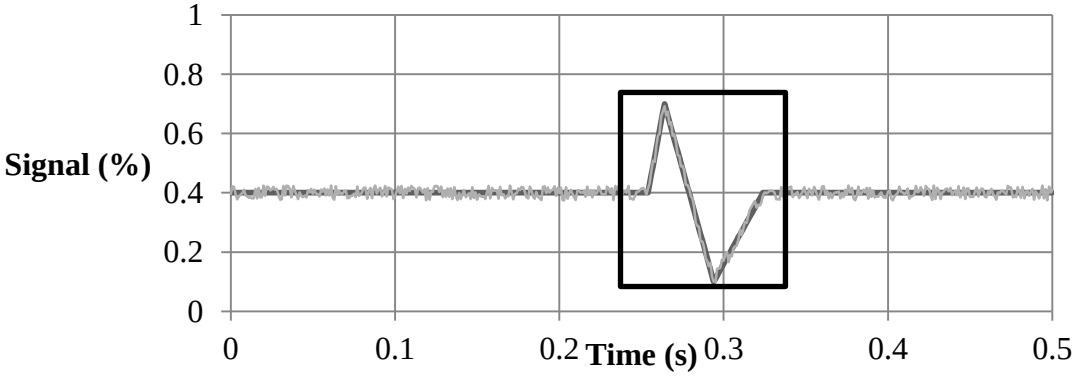


Figure 5 – A portion of the stream of inputs (light gray) and its best-fit representation (dark gray). The input pattern from Figure 4 can be identified within (outlined).

Since the recognition of existing patterns is dependent on their re-occurrence in the stream of inputs, it is possible that any recurring portion of the stream of inputs has the potential to be recognized as an input pattern. Thus, new input patterns are identified by locating recurring sequences of data. Due to memory constraints, the context of the recurrence should be limited. Therefore, a new input pattern is determined to be a portion of the best-fit representation of the input data that recurs within a certain period of time.

Once the system is able to identify and memorize patterns, the system needs to return an optimal response for the pattern. This is done by the translator – a predictive hill-climbing algorithm that uses linear interpolation to optimize for reward.

Whenever the onset of multiple patterns is detected (as described in the next section), the system checks if:

1) The patterns diverge at a point that has different values for an element of the previous response component of the input.
2) The patterns have different reward values.

If both of these conditions are true, this suggests that the difference between the responses might have affected the reward obtained by the system (Figure 6). So, the translator linearly interpolates the diverging patterns (Figure 7). Then, the next time the onset of these patterns is detected, the system uses the generated linear interpolation, and scales the response to a value that is predicted to maximize the expected reward (Figure 8).

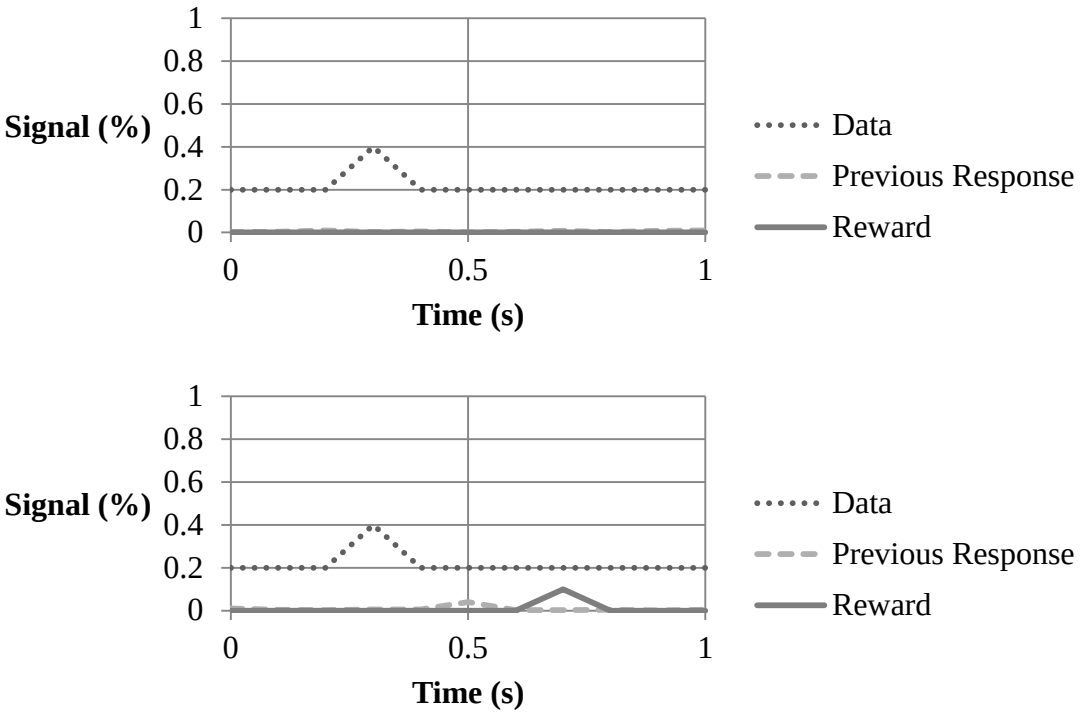


Figure 6 – Two patterns with the same onset, but which diverge at a point with different values for the previous response.

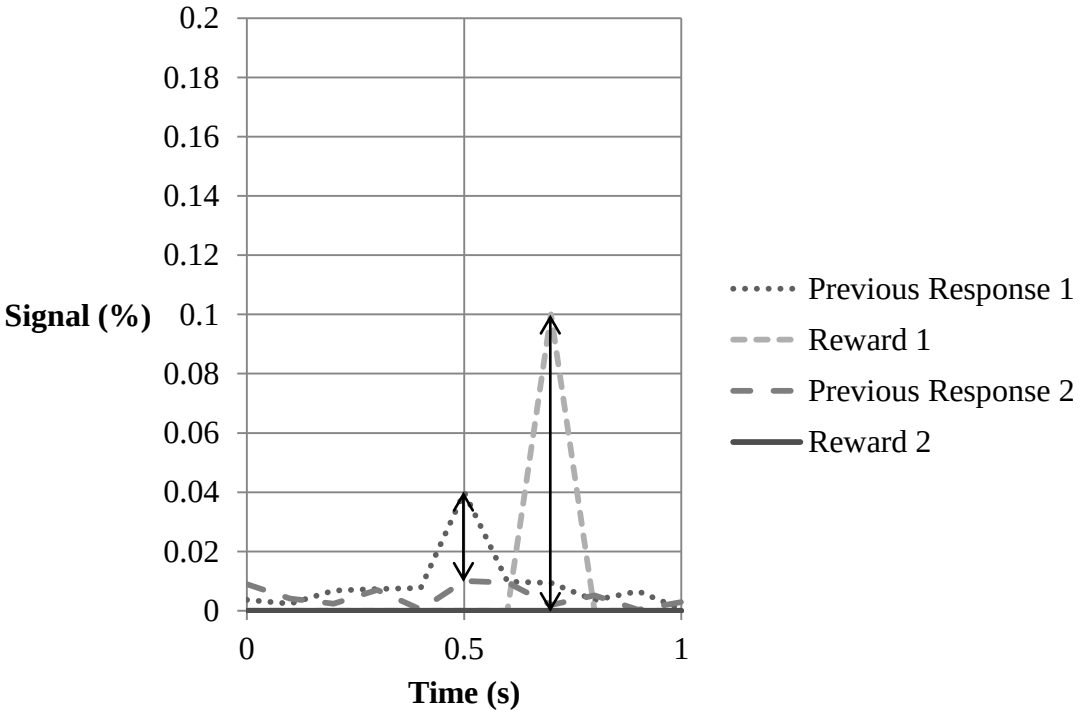


Figure 7 – Linear interpolation between diverging patterns. $\frac{\Delta Reward}{\Delta Response} \cong \frac{0.1}{0.04} \cong 2.5$.

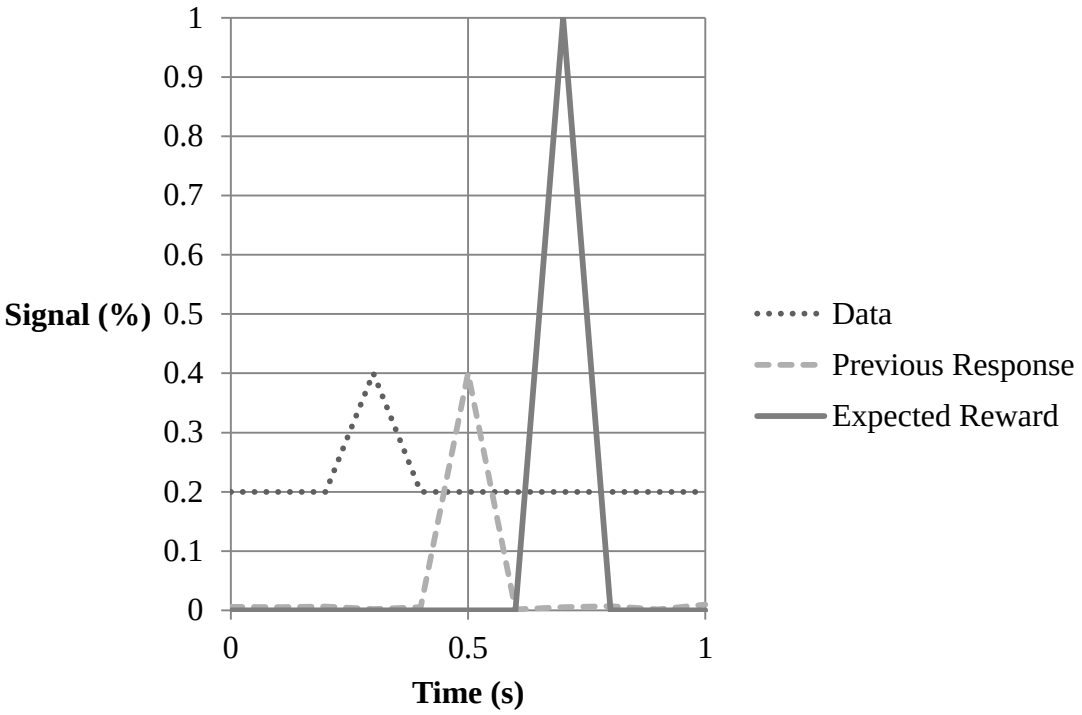


Figure 8 – Scaling the output maximizes the expected reward.

However, if the patterns do not diverge at a previous response element, or if the divergent patterns do not have different reward values, then the system modifies the response vector slightly, in order to generate a pattern which has a different reward value, so that the minor change can be used to triangulate the exact influence that the slight change in the response has on the rest of the pattern, especially reward.

The complete operation of the translator:

1) Observes inputs and identifies input patterns.
2) If multiple patterns branch out at an output element and have different reward values, go to step 4, otherwise, go to step 3.
3) Explore solutions for output by generating a small alteration in the output vector.

4) Linearly interpolate divergent patterns and different reward values, and use the interpolation to predict optimal output.

At the end of each iteration, the system returns a response, which is also a vector of real values between 0 and 1. The response vector is used by the BCI output system, and is mapped onto the set of commands available to the BCI. Thus, the response vector determines the action that the system performs.

Pattern Recognition – Overview

The goal of the PR is to represent input patterns, identify the occurrence of those input patterns in the stream of input data, and learn new patterns from the stream of input data. The representation of input patterns should be such that the system can efficiently identify the onset of an input pattern, predict the course of the pattern (the approximate values of the input elements that it encompasses), and determine when the pattern ends. The PR should also be capable of learning new input patterns by locating recurring sequences of data, as mentioned earlier.

The PR is inspired by Hebbian learning. In accordance with Hebb's rule, the simultaneous activation of neurons increases the synaptic strength between those two neurons. The PR categorizes the input data into discrete pieces, called memories, and then creates associations between groups of memories. Extending Hebb's rule, the associations are based on the relative time of occurrence of different memories, and are strengthened by the repeated reoccurrence of the same sequence of memories with the same relative time difference.

A memory is an abstract data structure that represents a displacement in time and input. Memories are stored in separate nodes, which can link to other memories. Links between memories are assembled using two arrays of pointers. The link array in a memory points to child memories, which are associated by the parent memory. Another array, called the priming link array, points a child memory to its parent. Together, these pointers are used to assemble doubly-linked trees, which represent associations between memories, and thus represent input patterns.

Each memory has a property called strength, which represents how often the memory occurs. Every time a memory is encountered, its strength is increased by a factor that decreases as the strength approaches 100%. Also, memories decay, losing a constant amount of strength per

second. Hence, if a memory is not refreshed often enough, it will become weaker, and eventually become eligible for replacement. The modification of strength is:

$$strength = \begin{cases} strength + \big((maxStrength - strength) * \gamma\big) - (\theta * \Delta t) & if\ the\ memory\ is\ encountered \\ strength - (\theta * \Delta t) & otherwise \end{cases}$$

Where γ is a constant for strength reinforcement, θ is a constant for strength decay, and Δt is the change in time since the last iteration.

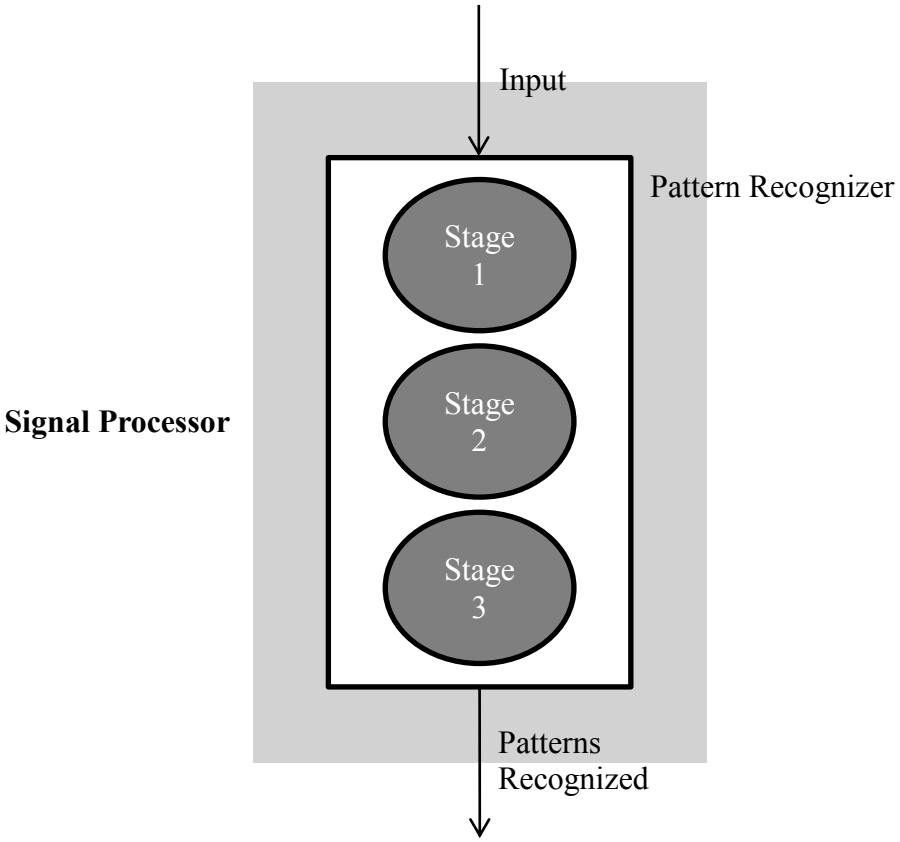


Figure 9 – Structure of the pattern recognizer.

The PR is divided into three stages (Figure 9).

In the first stage, the system constructs the individual pieces of the best-fit representation of the input data (Figure 10). The system keeps track the current slope, which represents the slope of

recent input data. For the very first iteration of the SP, the current slope is set to a default starting value. When the PR receives an input data point, it checks if the new input data point fits the current slope. If it does, it updates and extends the current slope (Figure 11), and then notifies any memories that contain the piece of the best-fit representation being constructed. If the data point does not fit the current slope (Figure 12), the second stage is triggered.

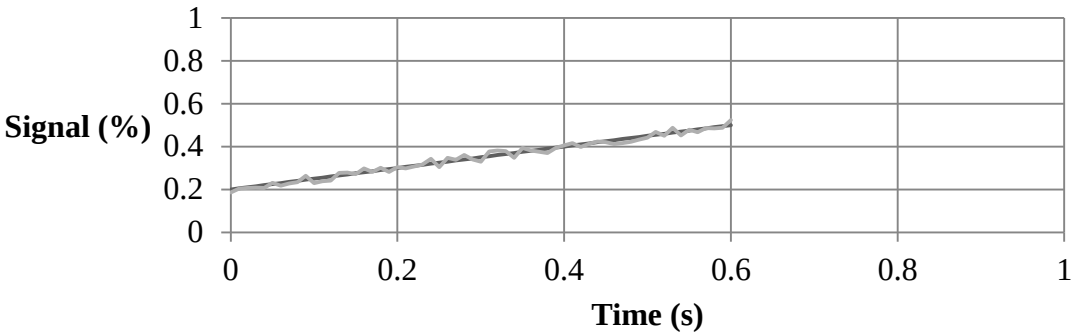


Figure 10 – A portion of the stream of inputs (light gray) and the current slope (dark gray).

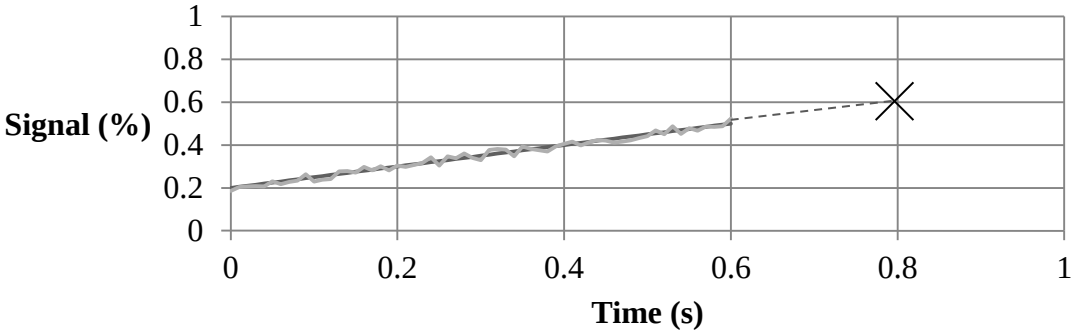


Figure 11 – Encountering a new data point (represented by an X) that fits the current slope.

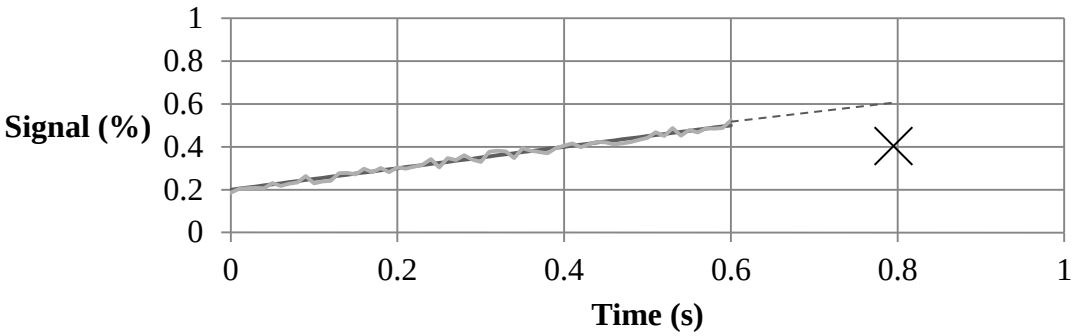


Figure 12 – Encountering a new data point (represented by an X) that does not fit the current slope.

When an input data point is inconsistent with the current slope, it means that the input data point belongs to a new slope. Thus, the linear piece that was being constructed in the first stage is complete, and is stored as a new memory by the second stage of the PR. The new piece is assigned to a free or weak memory node, overwriting a non-existent or weak memory with a new one.

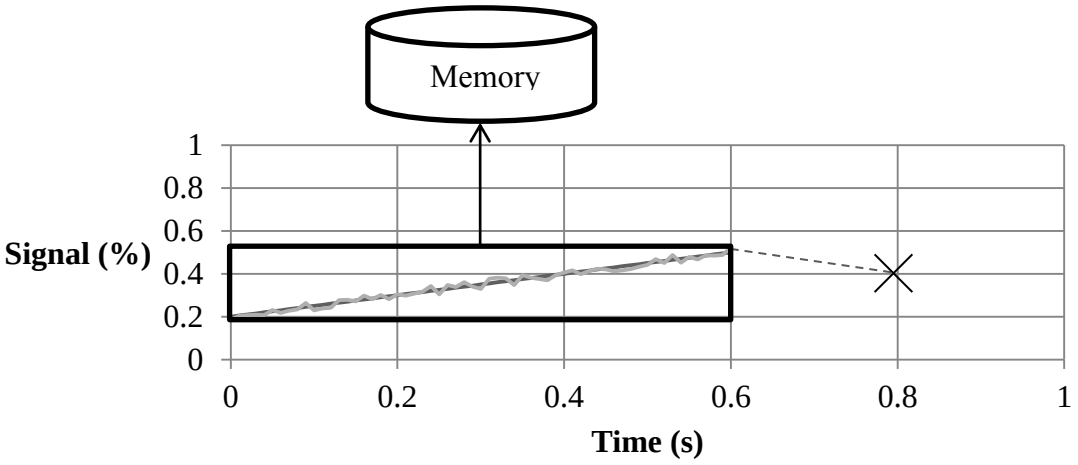


Figure 13 – Consolidating the old slope into a memory.

The third stage involves building associations between the memories, based on their relative time of occurrence. This stage also involves strengthening previously constructed

associations that have recurred. Nodes of the same type are used for the association, but with the slight difference that each of these memories is linked to the nodes whose association they represent (Figure 14).

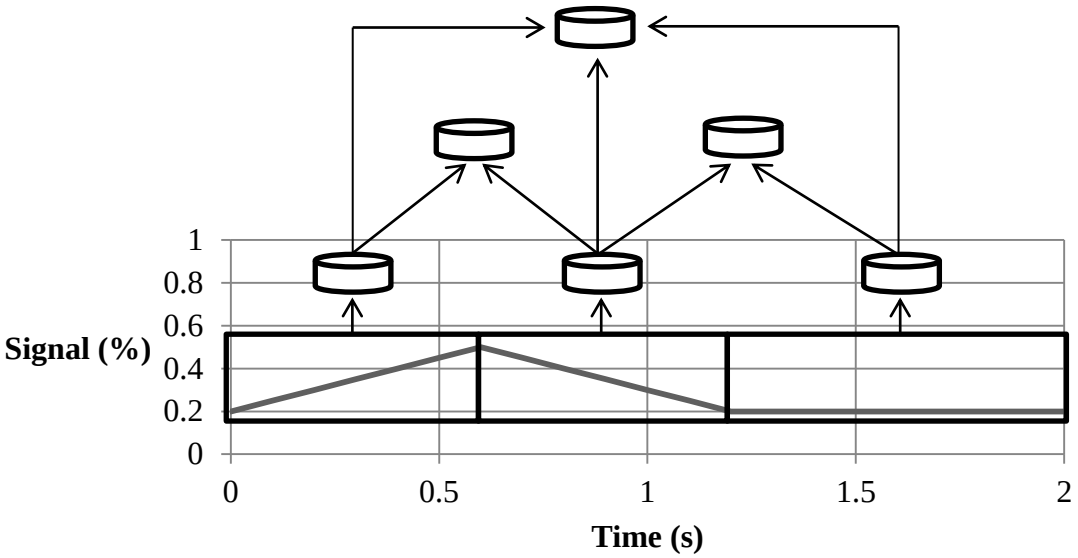


Figure 14 – Associating memories, which are pieces of the best-fit representation of the input data steam.

The first stage of the pattern recognition begins by processing the raw input. When the system is given an input vector I, each element of I is sent to a slope-detection node. The purpose of the slope-detection node is to evaluate the slope of the sequence of values from the linked input element. At each iteration, the initial node determines if the given input fits the current slope. This is done by determining what the expected input is, and then taking the absolute value of the difference between the expected input and the actual input. If this deviation is greater than a predetermined threshold, then the new input does not fit the current slope.

If the new input fits the current slope, then the new slope is incorporated into the current slope, so as to better represent the new input. This is done by determining, based on time, how

much the newly calculated slope should contribute to the overall slope, and then by combining the slopes based on that multiplier (Pseudo-code 1).

```
Slope.incorporate(newDeltaTime, newDeltaInput, newNetDeltaTime, newNetDeltaInput)
{
     //Calculate the value that determines how much a new input contributes to the
current slope
     hysteresis = newDeltaTime / newNetDeltaTime;

     //Calculate the value for the expected change in input
     expectedDeltaInput = ((deltaInput / deltaTime) * newNetDeltaTime);

     //Modify the values of the change in time and change in input
     deltaTime = newNetDeltaTime;
     deltaInput = (expectedDeltaInput * hysteresis) + (newNetDeltaInput * (1 –
hysteresis));
}
```

Pseudo-code 1 – The algorithm for incorporating a new input data point into the current slope.

Once this is done, the existing memories are checked to see if the current slope exists in them. If it does, the respective memories are primed. If the new input does not fit the current slope, then the current slope is passed to the second stage of the system, where it is recorded as a memory. After this, the current slope is set to the instantaneous slope, as calculated over the current iteration (Pseudo-code 2).

```
Stage1(time, input)
{
     //Calculate the change in the variables due to the new iteration
     deltaTime = time – previousTime;
     deltaInput = input – previousInput;
     netDeltaTime = time – currentSlope.startTime();
     netDeltaInput = input – currentSlope.startInput();

     //Calculate the predicted input value
     expectedInput = currentSlope.startInput() + ((currentSlope.deltaInput() /
currentSlope.deltaTime()) * netDeltaTime);

     //Check if the input is approximately equal to the expected input
     if(expectedInput ~= input)
     {
          //If so, then incorporate the new input into the current slope
```

```
          currentSlope.incorporate(deltaTime, deltaInput, netDeltaTime,
netDeltaInput);

          //Check if the slope has occurred previously
          for each(i in previousMemories)
          {
               if(i.getSlope() ~= currentSlope)
               {
                    //If a memory contains a slope that is approximately equal to
the current slope, prime that memory
                    i.prime()
               }
          }

     }
     else
     {
          //If not, then send the current slope to stage 2…
          Stage2(currentSlope);

          //…then, set the current slope to the instantaneous slope
          currentSlope = Slope(deltaTime, deltaInput);
     }
}
```

Pseudo-code 2 – Stage 1 of the pattern recognizer.

The second stage of the system is triggered when a new memory is identified. First, a check is made, to see if the newly identified memory has occurred previously. All the memory nodes are queried, and if the memory has been encountered previously – that is, if a previously existing memory of nearly the same displacement in input and time is found – then the node with the stored memory broadcasts the fact, and the strength of the memory is reinforced. If it has not been encountered previously, then the node with the weakest memory is overwritten with the new memory, which is given an initial strength, so that it isn't immediately overwritten again (Pseudo-code 3).

```
Stage2(slope)
{
     //Create the new memory using the given slope
     newMemory = Memory(slope);

     //Check if the new memory has occurred previously
```

```
     if(previousMemories.contains(newMemory))
     {
          //If so, increase the strength of the existing memory
          existingMemoryIndex = previousMemories.getIndexOf(newMemory);
          previousMemories[existingMemoryIndex].increaseStrength();
     }
     else
     {
          //If not, then overwrite the weakest memory with the new memory
          weakestMemoryIndex = previousMemories getWeakestMemoryIndex();
          previousMemories[weakestMemory] = newMemory;
          previousMemories[weakestMemory].strength = initialStrength;
     }

     //Run stage 3 with the new memory
     Stage3(newMemory);
}
```

Pseudo-code 3 – Stage 2 of the pattern recognizer.

The third stage of the system is automatically triggered by the second stage. In the third stage, the nodes query each other for the existence of memories that subsume this memory and any other memory. If memories such as this exist, their strength is reinforced. If no memory such as this exists, but the time since the last occurrence of that memory is less than the duration of the memory, then new memories are created. The creation of a new memory which associates two memories adds the new memory to the priming link array of the two memories being associated. Once the new memory is created, the current memory 'primes' all the memories that associate it with other memories, via the priming link array. This notifies the associating memories, which allows them to determine if an association has reoccurred (Pseudo-code 4). The identification of a recurring association is imperative for the operation of the translator.

```
Stage3(memory)
{
     //Juxtapose the new memory with every existing memory…
     for (i in previousMemories)
     {
          //…that has occurred within a period that is proportional to the duration
of the current memory
          if (i.timeSinceLastOccurence < (periodMultiplier * memory.duration))
```

```
          {
               //Make the new memory, which associates the two specified memories
               newMemory = Memory(i,memory);

               //Check if the new memory has occurred previously
               if (previousMemories.contains(newMemory))
               {
                    //If so, increase the strength of the existing memory
                    existingMemoryIndex = previousMemories.getIndexOf(newMemory);
                    previousMemories[existingMemoryIndex].increaseStrength();
               }
               else
               {
                    //If not, then overwrite the weakest memory with the new memory
                    weakestMemoryIndex = previousMemories getWeakestMemoryIndex();
                    previousMemories[weakestMemory] = newMemory;
                    previousMemories[weakestMemory].strength = initialStrength;
               }
          }
     }

     //Prime all the linked memories
     for (i in memory.primingLink)
     {
          i.prime();
     }
}
```

Pseudo-code 4 – Stage 3 of the pattern recognizer.

The priming algorithm involves a simple feed-forward mechanism that, when triggered in a memory, causes all the other memories that associate the first memory with other memories to be primed in turn. This priming is important for indicating the signs of recurrence of a memory, which is used in the exploration and prediction modules of the translator (Pseudo-code 5).

```
Memory.prime()
{
     //Prime all the linked memories
     for (i in primingLink)
     {
          i.prime();
     }
}
```

Pseudo-code 5 – Priming algorithm of the memory nodes.

Pattern Recognition – Evaluation

In evaluating the PR, the system was given a number of simple, pre-determined input patterns, to test if the system could learn and identify those patterns. This meant that each pattern would have to be given to the system at least twice, so that the system could detect the recurring pattern and process it appropriately.

When it was ascertained that the system could learn and recognize perfectly similar patterns, the system was then given patterns which were subject to spatial scaling and stochastic noise.

Spatial scaling was done by proportionally altering the distances between data points in a pattern. The tests for spatial scaling were done by scaling one of the patterns, while leaving the other intact, and then recording the effect this modification of the stream of inputs had on the recognition of patterns. The purpose of testing spatial scaling was to determine how different a pattern in the stream of inputs could be from the learnt pattern, before the PR decided that the former pattern was not the latter pattern. The tolerance of spatial scaling affects how easily a PR can identify a pattern, as well as how difficult it is for the PR to differentiate between two similar patterns.

Stochastic noise was generated by adding uniformly distributed random variation, proportional to the range of input values (that is, 0% to 100%), to each data point. The tests for noise were done by adding a limited amount of noise to all the data in the input stream, and then recording the effect that this modification had on the recognition of patterns. The purpose of testing noise was to determine how well the PR could operate when the data was subject to random error. The tolerance of spatial scaling indirectly affects how well the PR can identify

noisy data, as the effect of the noise is to distort the scale (and for sufficient noise, even alter the number) of the pieces of the generated best-fit representation of the stream of inputs.

The following is an example of one of the patterns used to evaluate the operation of the PR. The PR is set to being with 5 dead memories (a dead memory has 0 strength, and is thus immediately overwritten by any new memory), and is provided a sequence of inputs (Figure 15).

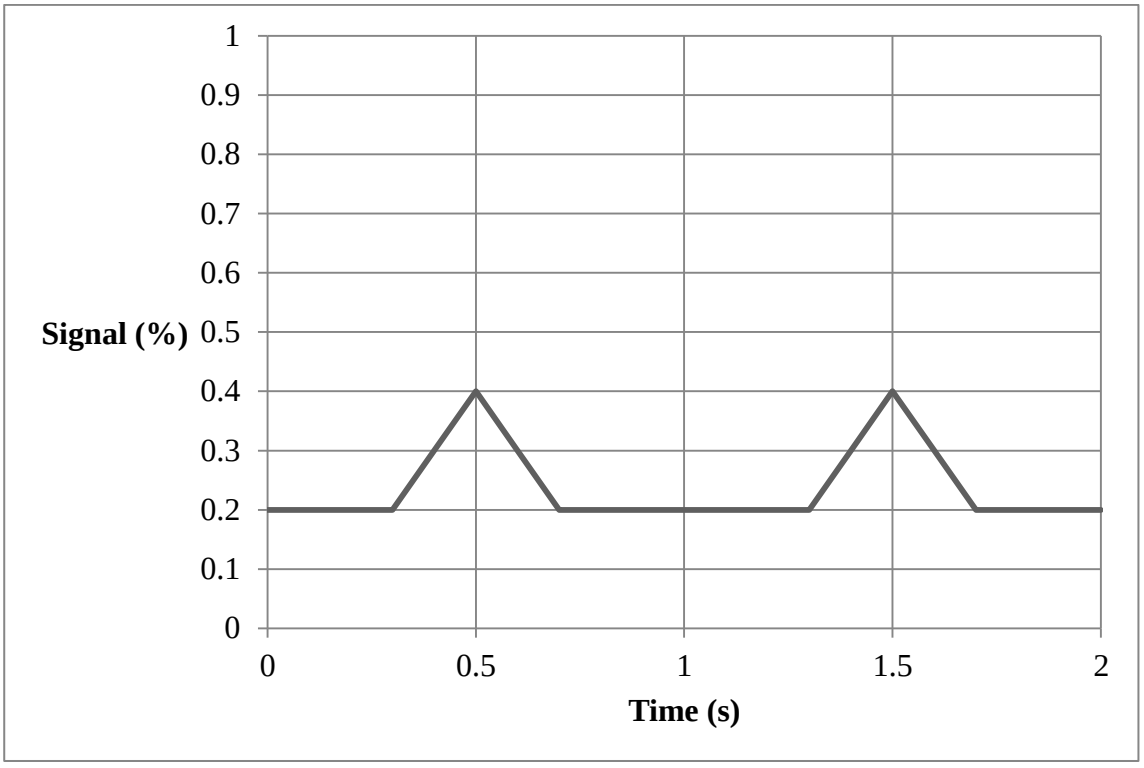


Figure 15 – Control sequence of inputs for evaluating system operation.

This input sequence depicts two spikes. Both spikes have the same dimensions (beginning at 0.2 units of amplitude, peaking at 0.4 units in amplitude, ending at 0.2 units of amplitude, spanning 0.4 seconds). There is a gap of 0.6 seconds between the end of the first spike and the start of the second. This input sequence is fairly simple, and provides a basis for the operation of the pattern recognition module.

When provided the control sequence of inputs, the pattern recognition module identifies the beginning and ending of three recurring sequences of inputs, indicated by three offset lines (Figure 16). Each offset line indicates the priming of a different memory. The priming of the memory begins at the point when the memory is first detected, and ends when the end of the memory is detected. The detection of a memory is triggered at the iteration immediately after the significant change in slope, as the priming occurs only after the detection of a significant change in the rate of change of input.

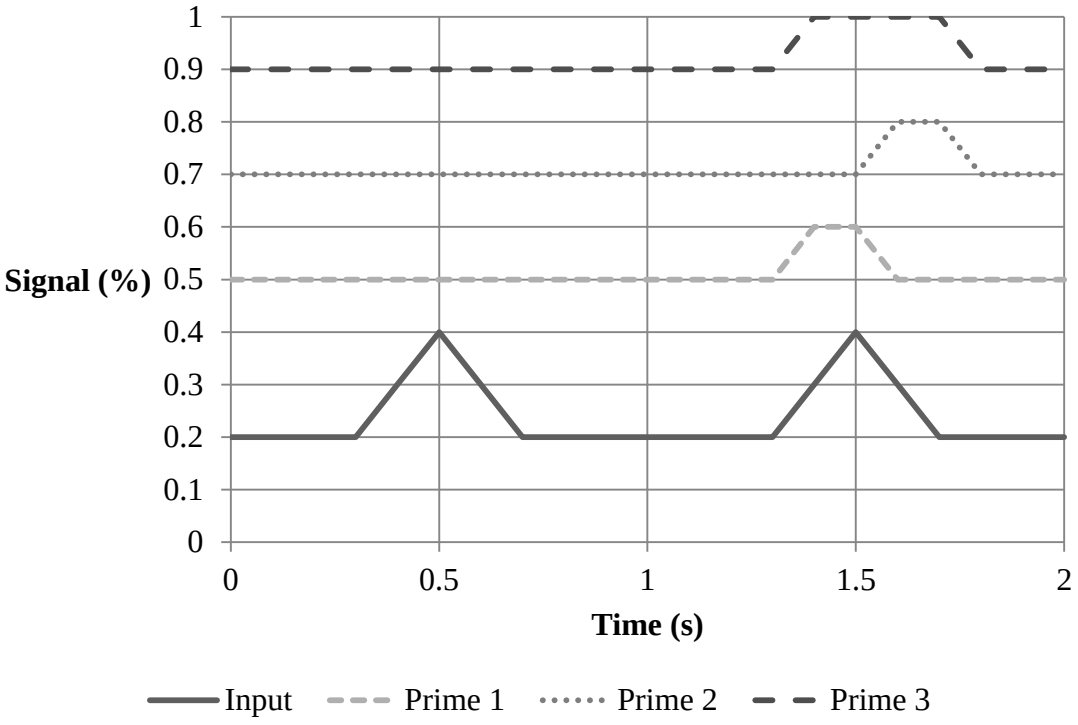


Figure 16 – Identifying patterns in the control sequence of inputs.

When provided with a sequence of inputs that resemble the control sequence, but with the first spike increased in scale by 5% (Figure 17), the identification of recurring patterns is the same.

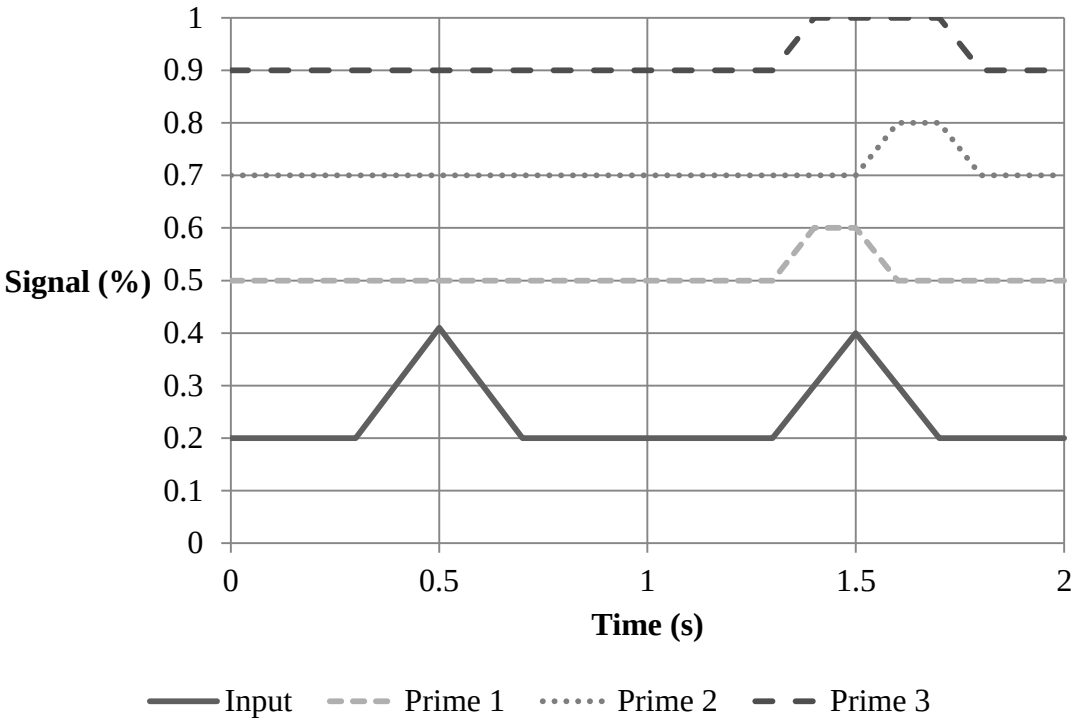


Figure 17 – Identifying patterns for a 5% difference in scale.

When increased in scale by 10% (Figure 18), the identification of recurring patterns is still the same.

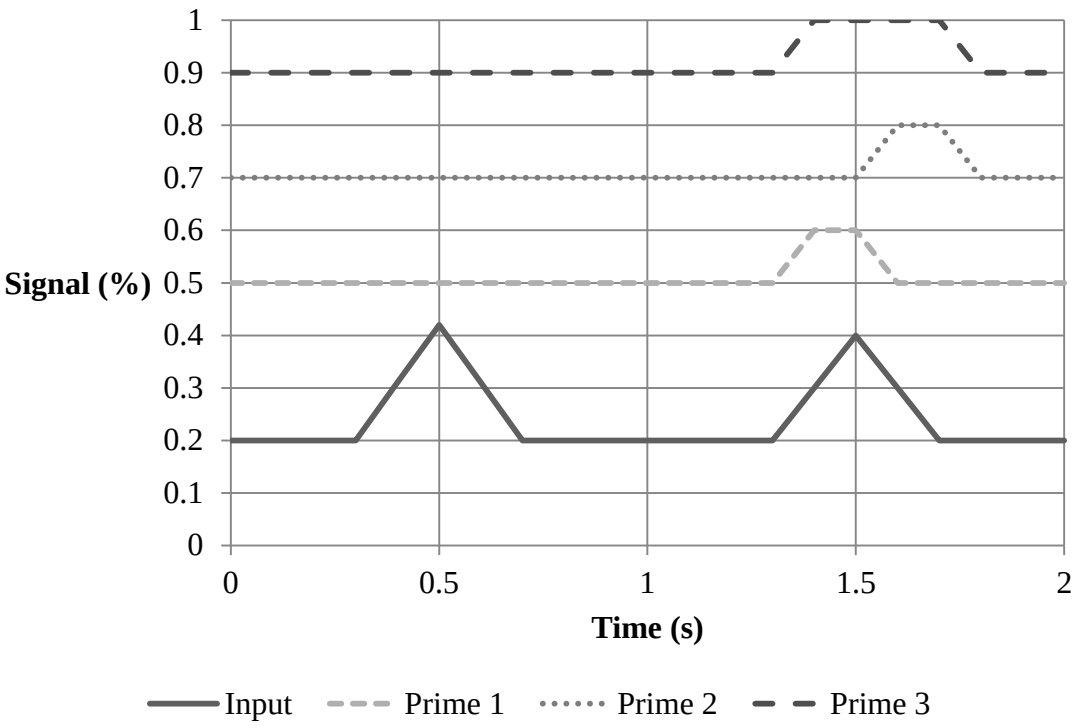


Figure 18 – Identifying patterns for a 10% difference in scale.

However, when scaled by 15% (Figure 19), no recurring patterns are identified. This is because the pieces of the best-fit representation of the second spike do not fall within the threshold of similarity when compared with the pieces of the best-fit representation of the first spike.

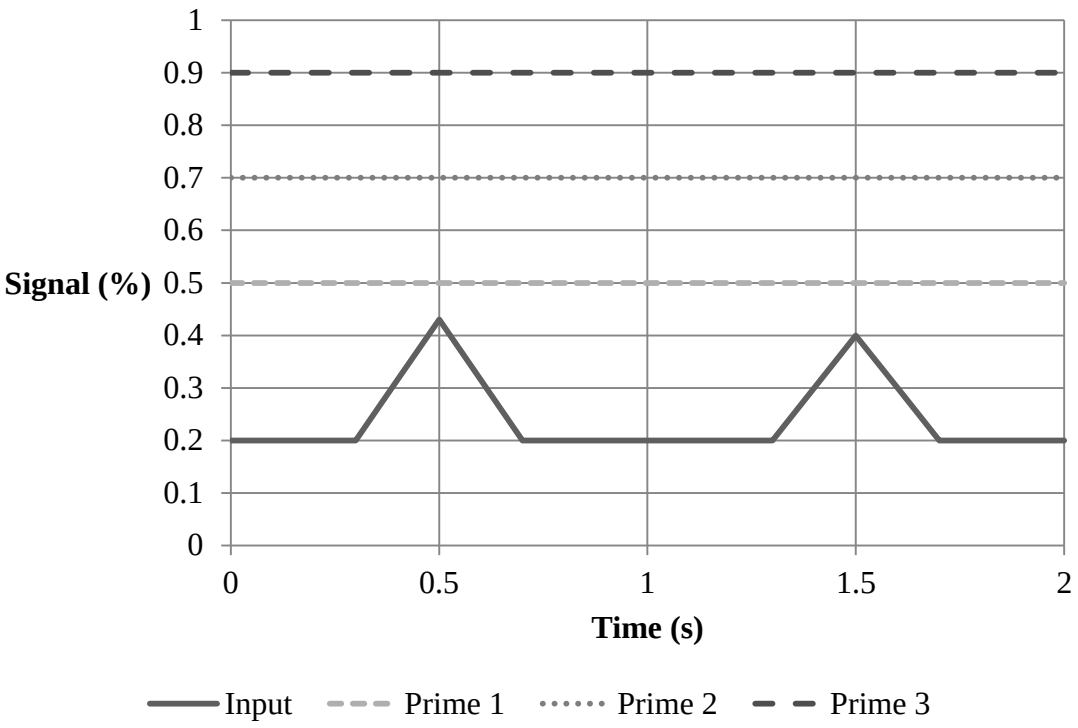


Figure 19 – No pattern identification for a 15% difference in scale.

With this degree of scale invariance, the pattern recognition is now evaluated for operation when subject to noise. When provided with a sequence of inputs that resembles the control sequence, but with a 1% distortion due to noise (Figure 20), the pattern recognition is similar to the control.

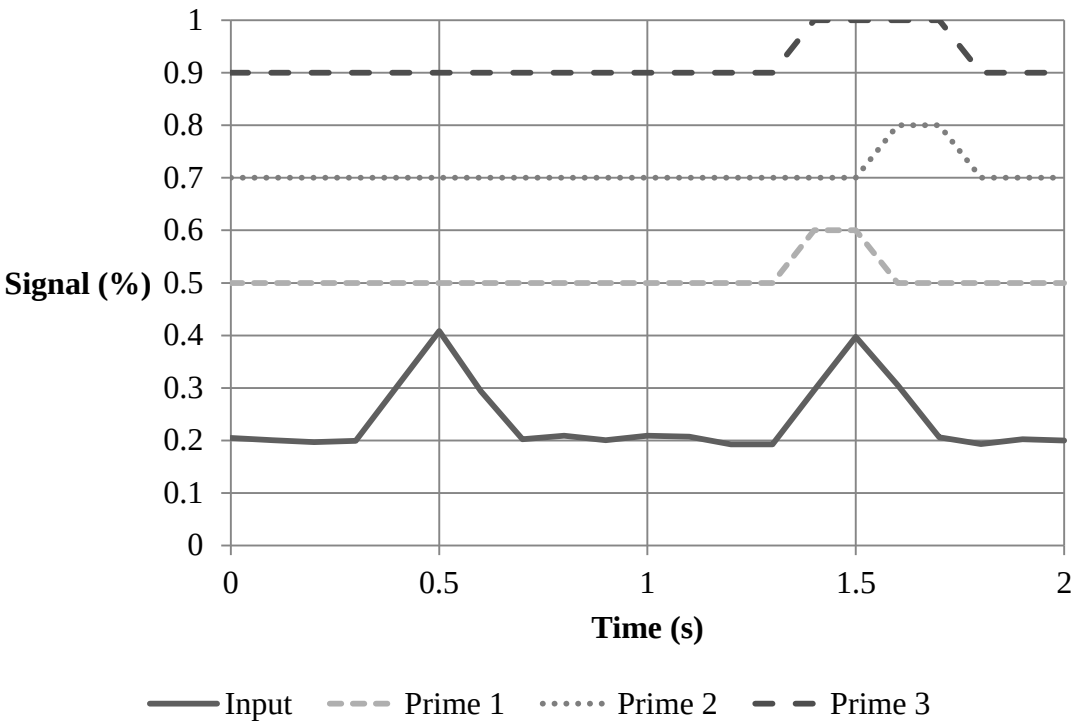


Figure 20 – Identifying patterns for a 1% distortion due to noise.

When provided with a 5% distortion due to noise (Figure 21), the pattern recognition breaks down, almost completely. A recurring pattern is found in this case, but it is a random artifact of the noise, since it is a priming response to a pattern, but does not match the output from the control test.

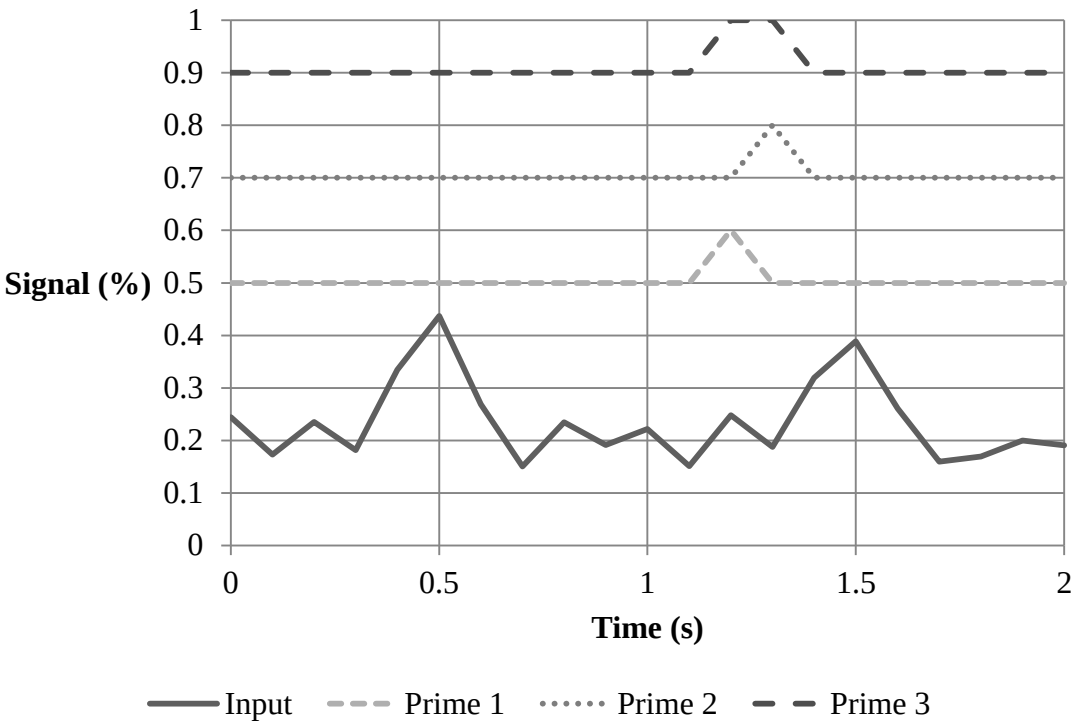


Figure 21 – Erroneous identification of patterns for a 5% distortion due to noise.

For the current degree of scale invariance, a 2% distortion due to noise (Figure 22) results the identification of recurring patterns akin to that of the control group.

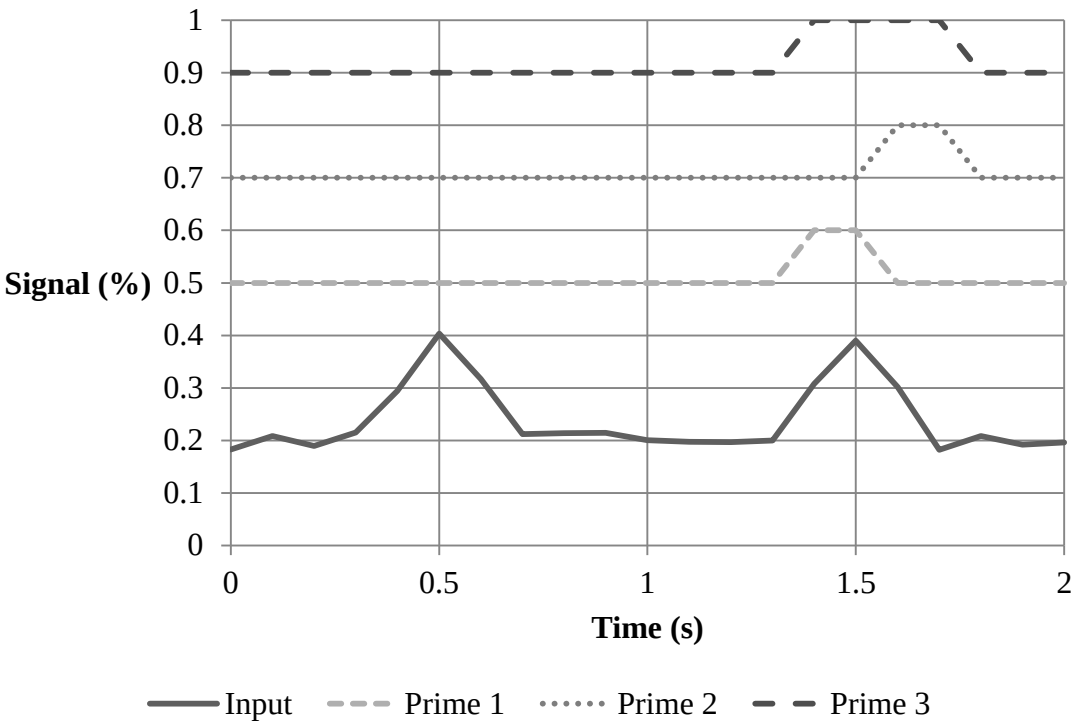


Figure 22 – Identifying patterns for a 2% distortion due to noise.

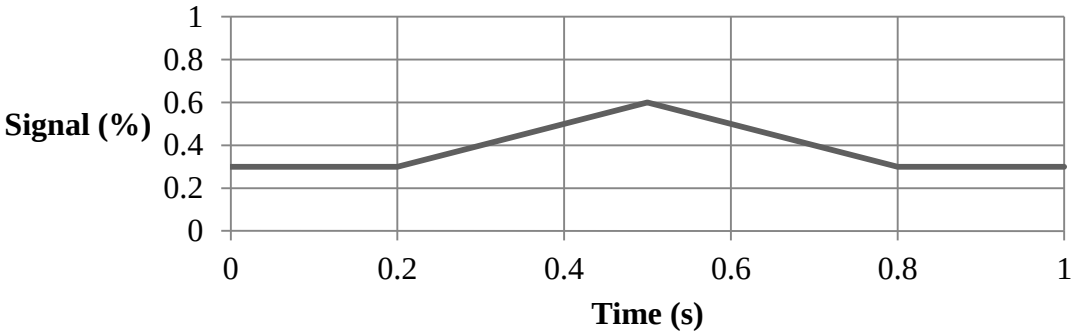
This evaluation was performed for many other patterns, with similar results. Having done this evaluation, and having ascertained that the PR is, at the very least, operational; the system can now be tested for performance in recognition of patterns, subject to noise.

Pattern Recognition – Performance

The performance of the PR was gauged using four tests. The first is a test of learning. The second, third, and fourth are tests of identification. All the tests were used to measure the accuracy of the recognition of patterns when noise is added to the input data.

In the first test, the system was provided two consecutive streams of data. The first stream contains a pre-determined pattern. The second stream is a copy of the first stream, but with added noise. Finally, the system is checked to see if the pattern was learned. The goal of this test is to determine the degree to which learning is invariant to noise.

The test begins with the system being trained on one of the three template data streams (Figure 23). This is followed by being trained on a data stream that resembles the template, but with added noise (Figure 24).



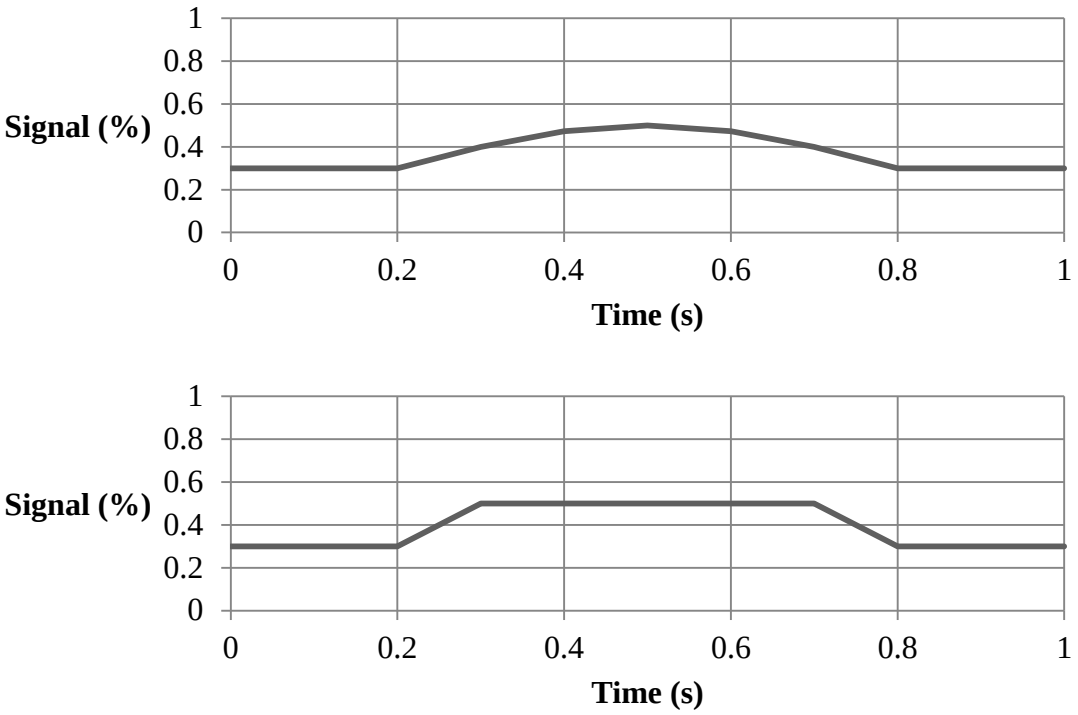


Figure 23 – Test 1, templates.

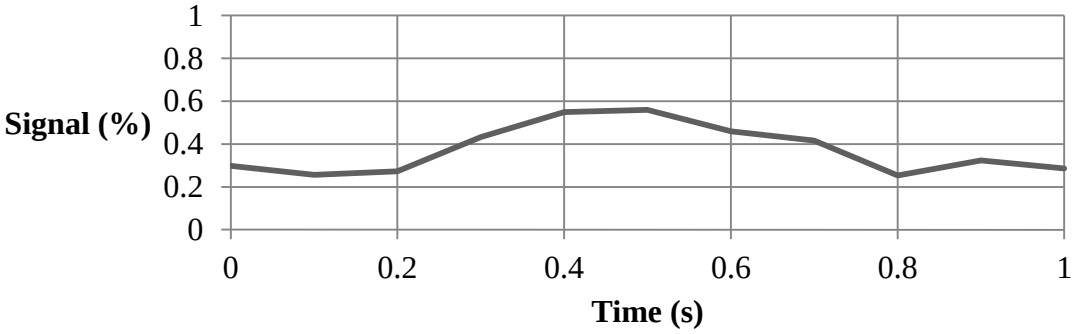


Figure 24 – Test 1, template 1 with 5% noise.

Test 1 was performed for 0%, 0.5%, 1%, 1.5%, 2%, 2.5%, 3%, 3.5%, 4%, 4.5%, and 5% noise. Each degree of noise was tested with 3000 trials, using each template 1000 times. The results showed that learning was 100% accurate for 0% noise, with above 95% accuracy from 0.5% to 2% noise, and reduced sharply in accuracy for greater than or equal to 3% noise (Table 1) (Figure 25).

Noise	Successful Trials	Accuracy
0	3000	100%
0.5	2947	98%
1	2931	98%
1.5	2940	98%
2	2864	95%
2.5	2415	81%
3	1415	47%
3.5	727	24%
4	308	10%
4.5	35	1%
5	5	0%

Table 1 –Test 1, results.

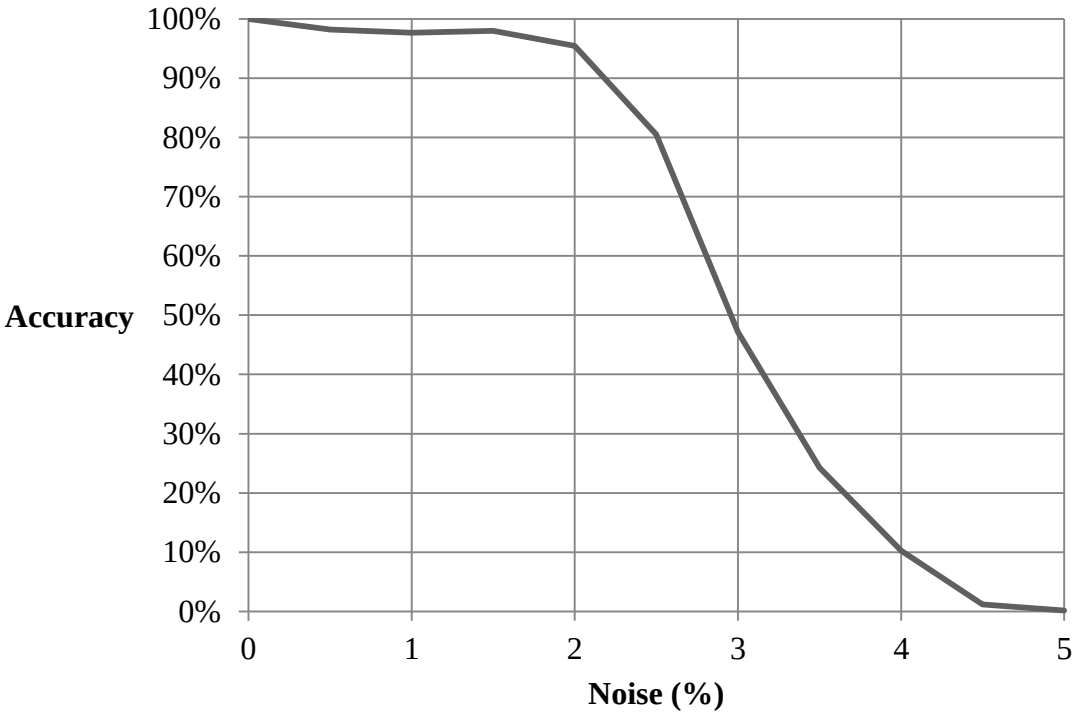


Figure 25 –Test 1, results.

In the second test, the system was provided a stream of data that contains a set of recurring patterns. It is then determined how many of these patterns the system is able to correctly identify.

The number of patterns identified, divided by the number of patterns present in the data is considered the accuracy of pattern recognition. The input data is then subject to increasing degrees of noise, and the accuracy is thusly recorded.

There are two kinds of errors that the system can encounter, when identifying patterns. The first is the identification of a pattern that is not in the template. This type of error is called a false positive, or type 1 error. This can happen, as was seen in the evaluation of the PR when subject to 5% noise, when the PR deems an artifact of the noise to be the recurrence of a pattern. The second type of error is the inability to identify a pattern that was present in the template. This type of error is called a false negative, or type 2 error. This happens when the PR encounters a pattern that was in the template but has been so distorted by noise that the PR is unable to ascertain that that portion of the stream of inputs resembles one of the patterns in the template.

The test begins with the system being trained on the template data stream (Figure 26). After this, the new-pattern-learning subsystem within the PR is turned off, to prevent the PR from treating artifacts of the noise as brand new patterns. This is followed by the PR being trained on a stream of input data that, at random points, contains sections of the template, subject to varying degrees of noise (Figure 27).

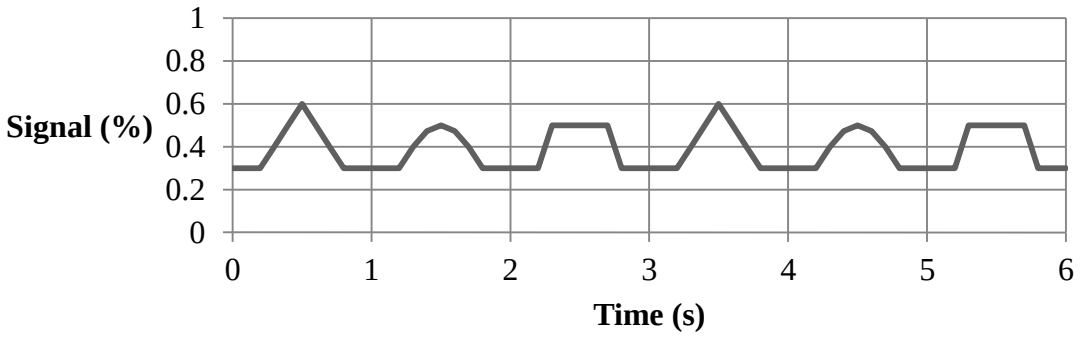


Figure 26 – Test 2, template.

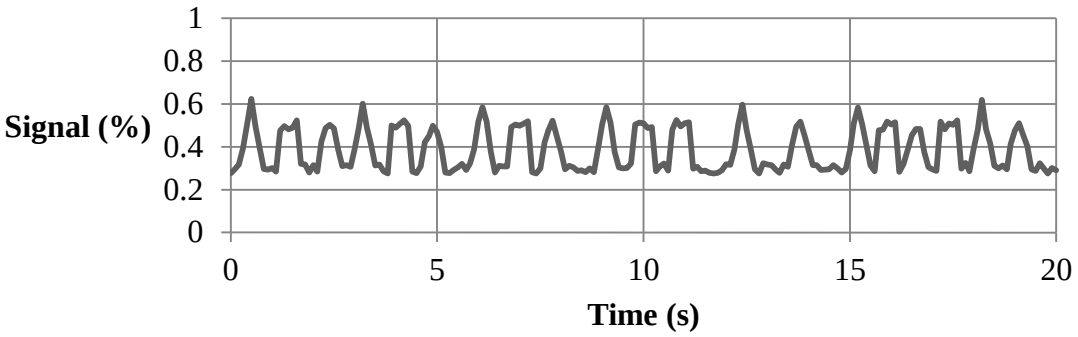


Figure 27 – Test 2, stream of input data with 2.5% noise.

Test 2 was performed for 0%, 0.5%, 1%, 1.5%, 2%, 2.5%, 3%, 3.5%, 4%, 4.5%, and 5% noise. Each stream of input data was 20 seconds long, with 20 patterns to identify. Each degree of noise was tested with 5000 trials. The results showed that learning was 100% accurate for 0% noise, with above 95% accuracy from 0.5% to 2% noise, and reduced sharply in accuracy for greater than or equal to 3% noise (Table 2) (Figure 28).

Noise	Successfully Identified Patterns	False Positives	False Negatives	Accuracy
0	100000	0	0	100%
0.5	97355	1311	1334	97%
1	96342	165	3493	96%
1.5	96232	1917	1851	96%
2	95083	412	4505	95%
2.5	72325	16018	11657	72%
3	32872	22946	44182	33%
3.5	12842	13055	74103	13%
4	2492	55496	42012	2%
4.5	381	35198	64421	0%
5	292	52932	46776	0%

Table 2 – Test 2, results.

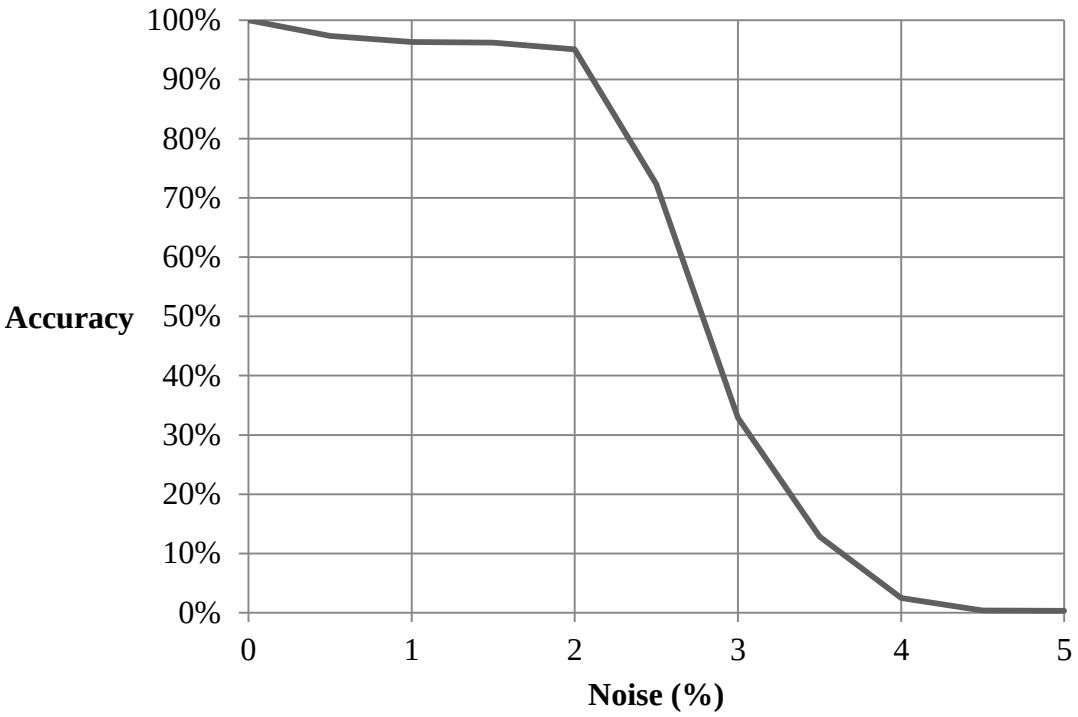


Figure 28 – Test 2, results.

The third test was an extension of the second test, but with the simultaneous modification of the threshold of scale tolerance. Test 3 was performed with the same parameters of noise, stream length, number of patterns, and number of trials per degree of noise as test 2. However, test 3 also performed for thresholds of scale tolerance that were 50%, 75%, 100%, 125%, and 150% the threshold of scale tolerance in test 2. The results for accuracy were similar to the results of test 2: learning was 100% accurate for 0% noise, with above 95% accuracy from 0.5% to 2% noise, and reduced sharply in accuracy for greater than or equal to 3% noise. This was the case for every degree of the threshold of scale tolerance. However, the ratio of type 1 errors to type 2 errors changed as the threshold of scale tolerance (Table 3) (Table 4) (Figure 29) (Figure 30).

False Positives

	Threshold of Scale Tolerance				
Noise	**50%**	**75%**	**100%**	**125%**	**150%**
0	0	0	0	0	0
0.5	609	873	1334	1609	1889
1	187	370	447	547	651
1.5	1393	2838	3370	3752	3724
2	1415	2696	3215	3901	4804
2.5	9282	13267	19996	25139	27534
3	13200	19411	26993	36289	40595
3.5	26930	34056	52281	63229	81815
4	43503	64732	80382	97139	97203
4.5	40587	70282	82695	99259	99405
5	42945	60756	78324	99465	99393

Table 3 – Test 3, results 1.

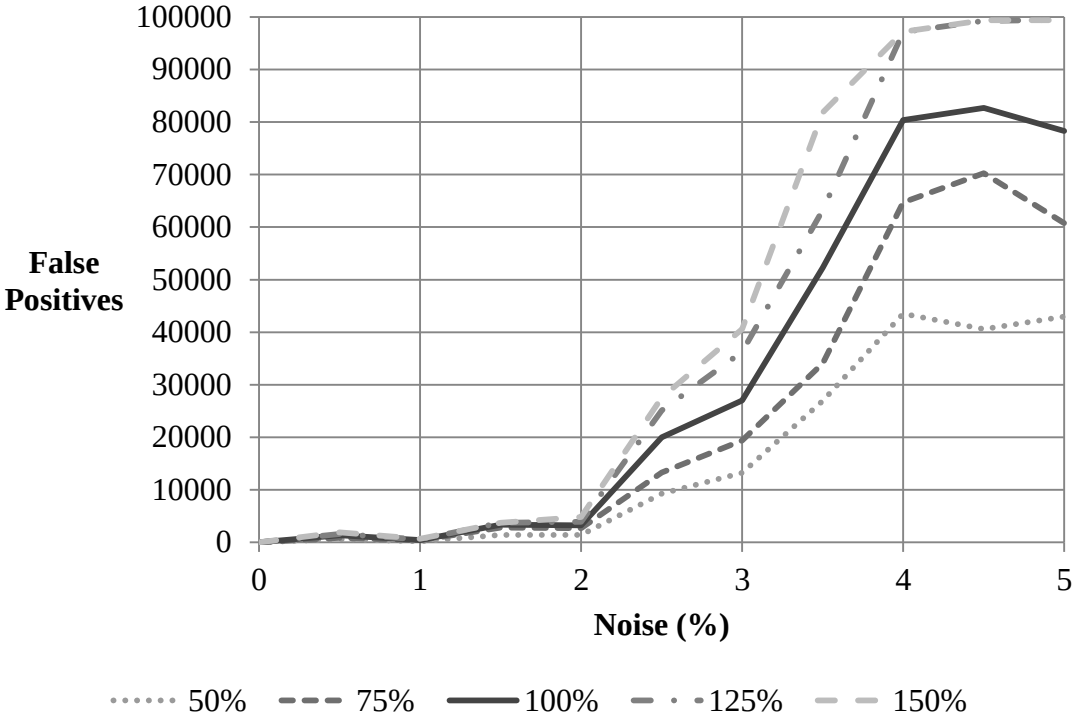


Figure 29 – Test 3, results 1.

False Negatives					
	Threshold of Scale Tolerance				
Noise	**50%**	**75%**	**100%**	**125%**	**150%**

0	0	0	0	0	0
0.5	2036	1772	1311	1036	756
1	3471	3288	3211	3111	3007
1.5	2375	930	398	16	44
2	3502	2221	1702	1016	113
2.5	18393	14408	7679	2536	141
3	53928	47717	40135	30839	26533
3.5	60228	53102	34877	23929	5343
4	54005	32776	17126	369	305
4.5	59032	29337	16924	360	214
5	56763	38952	21384	243	315

Table 4 – Test 3, results 2.

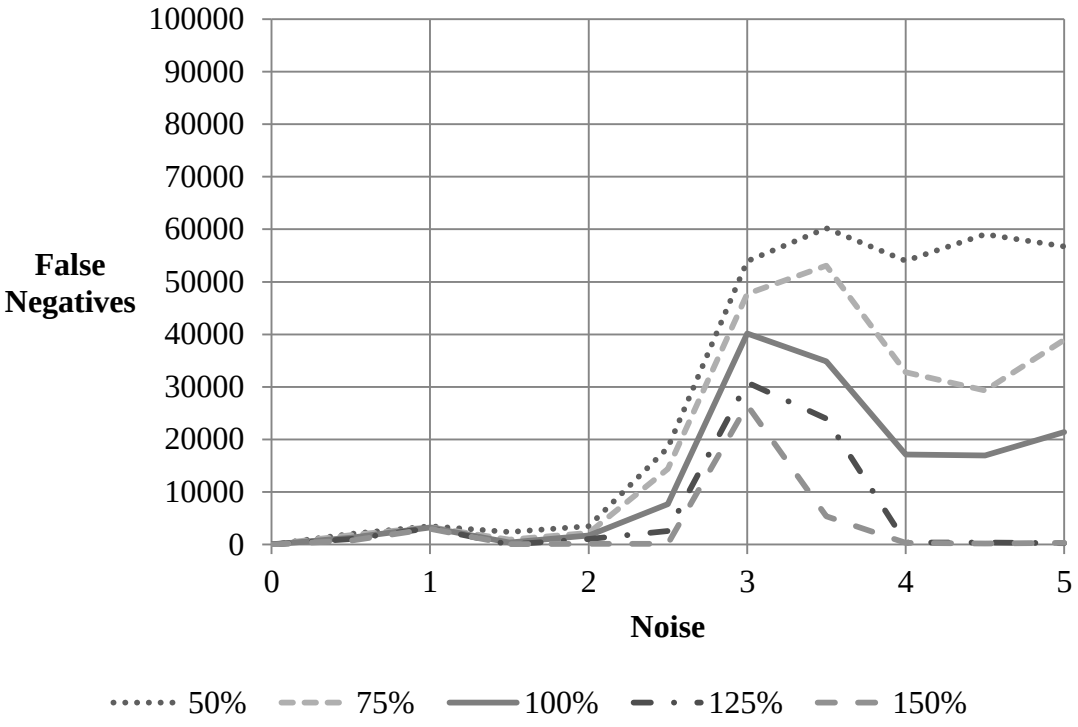


Figure 30 – Test 3, results 2.

The fourth test was also an extension of the second test, but this time, the system was provided with multidimensional inputs. The PR was run by feeding each element of the input through a separate slope-detection node. All the slope-detection nodes were connected to the same set of memory nodes.

Test 4 was performed for 2 to 5 nodes. The results were nearly identical to the results of test 2 for all numbers of nodes. This was to be expected, since the system was designed for multi-dimensional inputs.

Analysis

From the results of the performance tests, it can be seen that the system has rather specific limitations on tolerance of noise.

The evaluation of the operation of the PR indicated that it is capable of identifying learned patterns and learning to identify new patterns. A variety of different types of patterns were evaluated, with different levels of complexity. The recognition of patterns was tolerant of scale variance and the addition of noise, but to a limit. This limit seems to maintain for both simple and complex patterns. However, any more noise than the limit, and the accuracy of the PR deteriorates very quickly.

The first test of performance of the PR demonstrated that the system was capable of learning patterns with the same levels of accuracy as in the evaluation. This result was not particularly significant, but it was necessary to check this before moving on to other tests.

The second test provided a wealth of information. First, the accuracy of pattern recognition being equivalent to the results of the evaluation indicated that the system is performing as expected. Also, the system was found to be performing extremely well within a rather specific set of bounds. Furthermore, the nature of the errors produced at higher levels of noise shed light on exactly how the noise was affecting the operation of the PR. This relation between the types of error was further explored in the next test.

In the third test, it was found that changing the threshold of tolerance did not significantly change the accuracy of the PR. Rather, increasing the threshold of tolerance was resulting the in the generation of many more false positive errors than previously, while reducing the number of false negative errors. This was because the system was recognizing random triangular shapes in the stream of inputs as the triangular pattern from the template upon which the system was

trained, while also being able to recognize patterns through more distortion. Similarly, as the threshold of tolerance was reduced, many of the patterns that were previously recognized even when distorted by noise were no longer considered patterns by the system, but the various artifacts of the noise were being correctly ignored as well.

The fourth test was also done to verify the correct operation of the system. Since the PR had been designed to accept multidimensional inputs, the fourth test was done in order to ascertain that it was capable of doing so. Since the performance of the system in the fourth test was much like the performance of the second test, it was concluded that the system was performing as expected.

An additional test was tentatively performed. This final test mirrored test 2, but with the added condition that the learning system was not disabled before presenting the system with the stream of inputs. This resulted in a marked increase in accuracy, but also resulted in the memory nodes of the system being rewritten many times over. Providing the system with more memories improved the performance substantially. A linear relation was found between the reduction in error (or the increase in accuracy) and the number of memories available to the system.

Altogether, the information from these tests suggest that the PR is capable of supporting the feature extraction required for the translator. It would be useful to explore the feasibility of implementing this PR, as well as to investigate the assembly of the rest of the SP, and eventually, the entire BCI.

Future Research

Further research will probably entail:

1) Additional testing of the performance of the PR. Different conditions can be explored, beginning with the scaling of the system's performance relative to the number of nodes. Other useful tests would include comparing the performance of this PR to other PRs that operate similarly.
2) The construction and evaluation of the translator, using the results of the PR. The translator should be tested in the same manner as the PR, to gauge its operation and performance.
3) The decentralization of the SP. The PR and translator were originally designed to be parts of a single neural network artificial intelligence. In the evaluation of the PR, it was decoupled from the translator and flattened out into a serial system. However, it is still quite possible to return the system to its neural network structure. Once this is done, the system can be evaluated in in terms of algorithmic efficiency; properly evaluating the benefits of a distributed system versus a linear system.
4) Finally, the assembly and evaluation of the complete BCI, using input data from human EEG.

Bibliography

Anderson, N. R., & DeVries, E. M. (2010). Brain Computer Interface (BCI) Tools Developed in a Clinical Environment. *American Journal of Electroneurodiagnostic Technology*, 187 - 198.

Carmena, J. M., Lebedev, M. A., Crist, R. E., O'Doherty, J. E., Santucci, D. M., Dimitrov, D. F., . . . Nicolelis, M. A. (2003). Learning to Control a Brain–Machine Interface for Reaching and Grasping by Primates. *PLoS biology, 1*(2), E42.

Friedman, D., Leeb, R., Pfurtscheller, G., & Slater, M. (2010). Human-Computer Interface Issues in Controlling Virtual Reality With Brain-Computer Interface. *Human-Computer Interaction, 25*, 67 - 93.

Lotte, F., Renard, Y., Gibert, G., Congedo, M., Maby, E., Delannoy, V., . . . Lecuyer, A. (2010). OpenViBE: An Open-Source Software Platform to Design, Test, and Use Brain-Computer Interfaces in Real and Virtual Environments. *Presence: Teleoperators & Virtual Environments, 19*(1), 35 - 53.

Ramírez-Cortes, J. M., Alarcon-Aquino, V., Rosas-Cholula, G., & Gomez-Gil, P. (2010). P-300 Rhythm Detection Using ANFIS Algorithm and Wavelet Feature Extraction in EEG Signals. *Proceedings of the World Congress on Engineering and Computer Science.* San Francisco. Retrieved April 25, 2011, from Emotiv - Brain Computer Interface Technology: http://www.emotiv.com/

Stanley, G. B., Li, F. F., & Dan, Y. (1999,, September 15). Reconstruction of Natural Scenes from Ensemble Responses in the Lateral Geniculate Nucleus. *The Journal of Neuroscience, 19*(18), 8036-8042.

Wolpaw, J. R. (2010). Brain-Computer Interface Research Comes of Age: Traditional Assumptions Meet Emerging Realities. *Journal of Motor Behavior, 42*(6), 351-353.

Wolpaw, J. R., Birbaumer, N., Heetderks, W. J., McFarland, D. J., Peckham, P. H., Schalk, G., . . . Vaughan, T. M. (2000, June). Brain–Computer Interface Technology: A Review of the First International Meeting. *IEEE Transactions on Rehabilitation Engineering, 8*(2), 164-173. Retrieved from http://www.ocf.berkeley.edu/~anandk/neuro/BCI%20Overview.pdf

Wolpaw, J. R., Birbaumer, N., McFarland, D. J., Pfurtscheller, G., & Vaughan, T. M. (2002). Brain-computer interfaces for communication and control. *Clinical Neurophysiology*, 767-791.

Zhang, X.-D., Wang, Y.-X., Li, Y.-N., & Zhang, J.-J. (n.d.). *An Approach for Pattern Recognition of EEG Applied in Prosthetic Hand Drive.* Retrieved from International Institute of Informatics and Systemics: www.iiis.org/CDs2010/CD2010IMC/CCCT_2010/PapersPdf/TA865CT.pdf