

Liquid Computing: A Real Effect

ABSTRACT

A novel approach to temporal pattern recognition was recently introduced in the literature, called liquid computing. The model used was very complex, thus the present study uses a simplified version that is easier to understand. By exploring properties of the system with the simple setup, a better understanding of the complex version can be reached. A liquid computer needs a liquid memory and readout. In the present study, a liquid based on damped coupled oscillators was used. The liquid was a finite element model of water and the wave produced by throwing a rock into it. The readout was a 3-layered back-propagation network. The findings show that even a simplified version of the liquid computer can successfully classify patterns presented serially in time, with or without noise, with or without jitter, and can generalize its training in some cases. It is also shown to correctly learn timing information about its' input. The liquid computing framework is therefore shown to be a powerful tool in the analysis of temporal signals using neural networks.

BACKGROUND

Liquid computing is a technique recently described in the literature (1,2). It provides a theoretical framework for a neural network that can process signals presented temporally. The system is comprised of two subcomponents, the “liquid” and the “readout”. The former acts as a decaying memory, while the latter acts as the main pattern recognition unit (see Figure 3 and 4). To understand the system, a simple analogy is used. Imagine a cup of water. Now picture a small rock thrown in (figure 1). The water generates ripples that are mathematically related both to characteristics of the rock, as well as characteristics of the pool at the moment that the rock was thrown in. A camera takes still images of the water's ripple patterns. A computer then analyzes these still pictures. The result is that the computer should know something about the rock that was thrown in. For example, it should about how long ago the rock was thrown. The rock represents a single bit from an input stream, or an action potential. The water is the liquid memory. The computer functions as the readout.

The implementation used in the literature was very sophisticated. It involves a complex column of three dimensionally arranged leaky integrate and fire neurons for the liquid. The column has a certain amount of random recurrent connections, which give it the property of memory. Some of the connections are specified as inhibitory, others as excitatory. Neurons were assigned a probability of synapse with other neurons, based on three-dimensional proximity to each other. Meanwhile, the readout was calculated in a variety of ways, including a simple perceptron and a back-propagation neural network.

In trying to understand the workings of the setup used in the literature, one may become overwhelmed by the complexity of the system used to describe a seemingly simple concept. The analogy used above with rocks and water can be implemented more directly, thus simplifying the system and making it more accessible for analysis.

METHODS

A variety of models were used to explore the possibilities that arise with the liquid state machine framework. The majority of the work went into exploring substrate for the liquid, and the overall system capabilities. Code for the back-propagation algorithm was modified from a Matlab code for a simpler network (personal communication, Scott Beardsley). All other code was original, and developed in Matlab.

The first experiment was to show that a simple one-dimensional “pond” could model a liquid. The pond had N elements. Each element was coupled horizontally to its neighbors as well as vertically to itself from the immediate past. The coupling equation was simply that of a spring, $a = -kx$, where a =acceleration, k =constant, and x = the vertical displacement of the element from position zero (see Figure 5). The sum of all couplings would determine the overall acceleration. This was used to compute the velocity. The velocity was added to the current position, and then scaled by a decay factor, b . This process was repeated for each element at each time step. The results looked very realistic, especially as N increased.

Once the liquid was working, two were designed in parallel to process somewhat similar inputs. Each pool was given three rocks to be thrown at random times into random locations, but all before time step 100. The first rock pattern was referred to as pattern A, and the second as pattern B. The resulting pools were then used as inputs to a simple perceptron with an error rule. Because poor results were obtained, a three-layered back propagation network (3) was used instead (see Figure 2). The network was chosen to be three-layered feed forward, with as many inputs as the size of the liquid, 20 units in the hidden layer, and 1 unit in the output layer. The same network was used for all subsequent experiments.

Next, the same setup was used with the addition of a rock thrown in at a randomly chosen time after the trained pattern had been presented. The question asked was could the trained network distinguish between patterns A and B after the extra rock was thrown in. This was equivalent to asking if noise were introduced into the liquid, would it still preserve enough information so that the readout could classify correctly.

Again the 1D pool was used with pattern A vs. pattern B, this time with the original five random rock throw times modified by a small random amount. The experiment tried to measure how well the system performed if the pattern presented differed from the trained pattern by the small jitter.

Next the system was tested for the ability to measure how much time passed from the beginning of the rock throw. A single rock was thrown in a random location, and the pool values were computed. The network was then trained to respond with linearly increasing values as the time progressed. This was also done using quadratic increases in target responses, and quadratic decreases.

Using the same principles as the 1D pond, a 2D pond was developed as well. In this case, neighbors were surrounding each element left, right, up, down, and on the diagonals. Weights were modified to reflect the Cartesian distance between the center and the neighbor, so for instance the left neighbor would get a weight of 1, but the upper left neighbor would get a weight of $1/\sqrt{2}$. This setup was modeled in 3D by using the value of each element as the height. In that way, image sequences that resembled a real

pond were generated.

The 2D pond was then modified to map its outputs to a sigmoidal activation function, similar to the one used by the back propagation network of the readout. This was done to bring the model closer to its more advanced relative, the neuron. By using a sigmoidal activation function, the elements of the 2D pond model could begin to crudely approximate neurons, which fire action potentials in an all or none fashion.

RESULTS

Figure 6 shows the results of classification for two input patterns. The system was trained to identify the patterns starting from the end of the last of the pebbles thrown in. It is clear that even before the last pebble, the system correctly classifies the patterns, showing generalization.

Figure 7 shows that when noise is added to the system, it recovers well and can still perform with reasonable correctness.

Figure 8 shows that when jitter is added to the input signal, it can still recover the correct classification to a reasonable degree.

Figure 9 shows the results from the quadratic timing experiments. The linear timing experiments appeared to learn a more quadratic looking output, so it was deemed not as useful as the quadratic. The system in both cases is generalizing past the training data, but the further away from the training it gets, more noise creeps in.

Figure 10 shows the 2D pond after several rocks have been thrown in, and a rock was just thrown into the middle.

Figure 11 shows the sigmoidal function which was introduced into the 2D pond model to bring it closer to a neuron model.

DISCUSSION

The fact that a simple 1D pond was able to classify pattern presented temporally shows that the theory behind liquid state computation is in principle correct. The systems used showed a tendency towards generalization, which furthers the argument that such a setup is useful. Although this was not explicitly explored here, others (1,2) report that multiple features can be simultaneously extracted from a single liquid using specialized readouts. Still, that was implicitly shown by using the same underlying substrate (the 1D pond) for multiple unrelated learning tasks.

As the ponds grew in complexity it slowly began to take on features of realistic neurons. By the time it had two dimensions and a sigmoidal activation function, it was already approximating a basic neural network from start to finish. Each element of the finite element model in the 2D pond could be considered a neuron. Each neuron would need recurrent connections to account for calculations based on previous values. It would also need lateral local connections to surrounding neighbors. The constant k could be thought of as a synaptic weight. Thus the function of each neuron was to calculate the weighted sum of a series of values based on present values from neighbors, as well as past self values due to recurrent connections. The weighted sum would then pass through a sigmoidal activation function, similar to what occurs at an axon hillock. At the readout stage, the liquid was read into computational units that had similar properties: they would take a weighted sum of inputs, run them through a sigmoidal activation function, and use

that as the outputs. In short, the entire liquid state machine at represented a simplified model of a neuronal microcircuit.

There are some problems with the simplified model. It doesn't account for the large diversity of neurons found in biological experiments. This is addressed using the more complex models (1). It also doesn't deal with the question of channel noise or synaptic noise, both potentially significant contributors to neuronal behavior. Finally, neither the liquid nor the readout truly model the temporal aspects of real neurons. It is assumed for simplicity that a neuron simply takes the weighted sums of inputs and passes it through a sigmoidal activation function. The weights are assumed to be constant always in the liquid, and the weights are assumed to be constant after training in the readout. In reality, the neuron can propagate action potentials backwards, reassign synaptic weights continuously, and experience strange effects such as long term potentiation and long term depression.

It would be interesting to further link these simple models to the more complex ones now being investigated (1,2). Perhaps some of the shortcomings of these models could be avoided by using optimized code that accounts for the various realistic possibilities and stronger computational power.

CONCLUSION

After careful examination of the liquid state machine as a potentially biologically plausible way for small groups of neurons to classify patterns presented temporally, it must be concluded that the technique has strong merit. Even in a simplified form, it can correctly classify patterns presented temporally, it can generalize beyond its training in multiple ways, and it can learn a variety of types of tasks with nothing more than a change in the training.

REFERENCES

1. W. Maass, T. Natschläger, and H. Markram. *Real-time computing without stable states: A new framework for neural computation based on perturbations*. Neural Computation, 14(11):2531-2560, 2002.
2. T. Natschläger, W. Maass, and H. Markram. *The "liquid computer": A novel strategy for real-time computing on time series*. Special Issue on Foundations of Information Processing of TELEMATIK, 8(1):39-43, 2002.
3. Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1995) Ch8: Learning Internal Representations by Error Propagation, Parallel Distributed Processing: Explorations in the Microstructure of Cognition Vol. 1 Foundations. The MIT Press pp318-362.

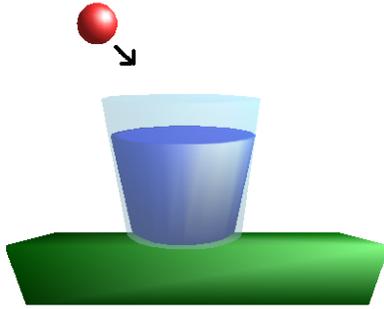


FIGURE 1: The pebble in the cup analogy. The liquid of a liquid state machine can be thought of as a liquid in a cup. The pebble can be thought of as the input to the system. The ripples in the water of the cup can be thought of as the memory of the system, which can be sampled at various times by the readout to determine facts about the past.

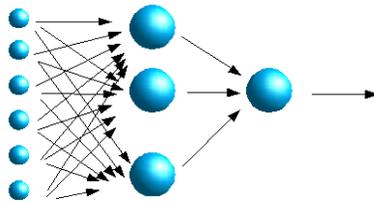


FIGURE 2: A schematic of a back-propagation network. Each small sphere represents an input; the group of them are called the input layer. Arrows show synaptic connections, each of which has a separate associated weight or strength. The large spheres in the middle represent the hidden layer. The hidden layer neurons are connected to the output layer, which contains only one neuron.

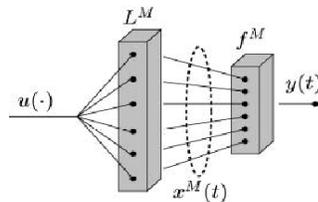


FIGURE 3: The general setup of a liquid state machine. Input comes as a vector into the liquid. The liquid dynamically changes, and is sampled by the readout. In turn, the readout has an output that (from Maass et al, 2002).

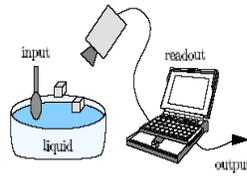


FIGURE 4: The general setup of a liquid state machine is that inputs (spoon, sugar cubes) are put into the liquid. The liquid is perturbed, and these changes are recorded by a camera. The camera output is fed to a computer capable of recognizing patterns presented spatially (from Natschläger et al, 2002)

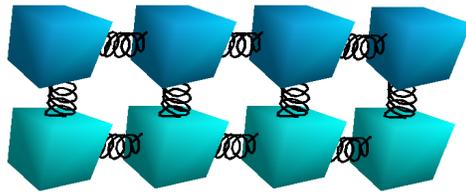


FIGURE 5: The framework used to develop a liquid for this report. The finite element model of the liquid allowed each element to be calculated in exactly the same way. Each element was considered to have a spring coupling its movement to the element next to it to the left or right. There are also vertical connections, which exist between present elements (light blue) and the same elements one time step ago (darker blue).

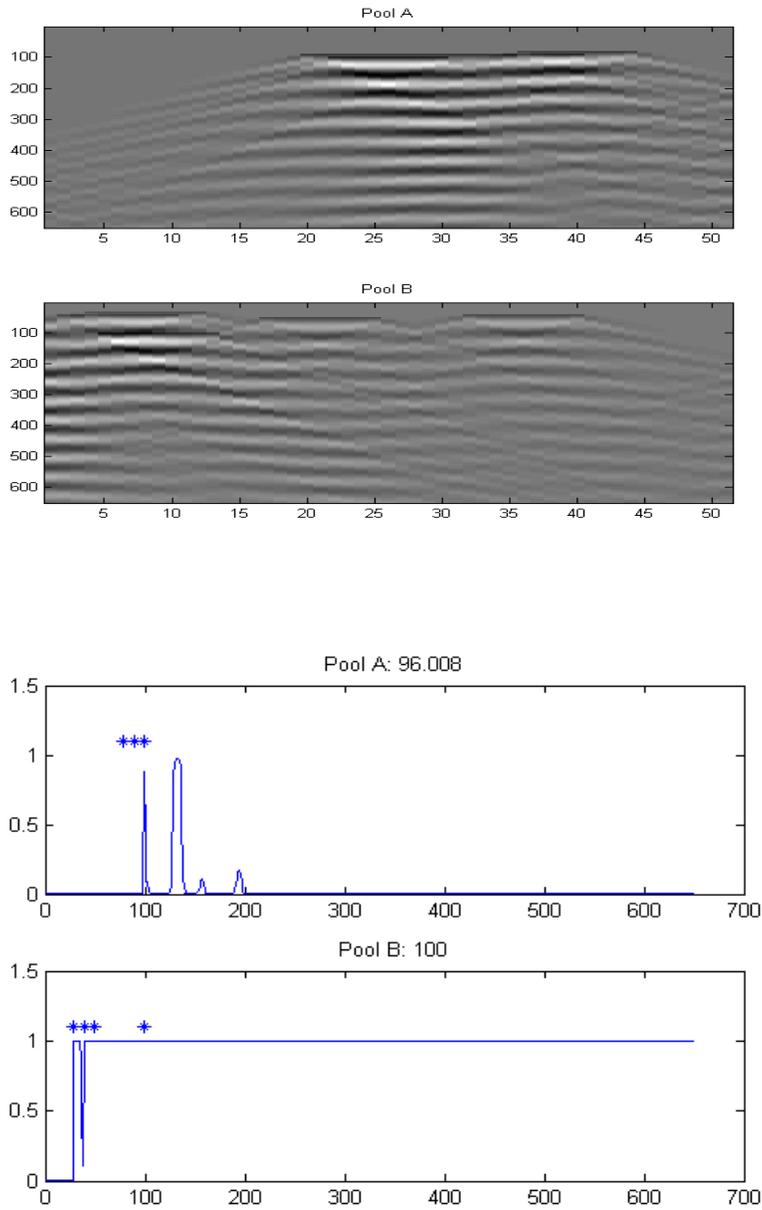


FIGURE 6: Both upper two plots have time as the y-axis, and space along the 1D pond on the x-axis. Three rocks were chosen to drop at five random times between 0 and 100, and random locations for each of the two plots. Clearly, the two patterns formed have differences when viewed in this way. The two lower plots show the value of the output of the neural network as a function of time. It was trained to output 0 when pattern A was presented, and 1 when pattern B was presented. Stars represent times when rocks were dropped in. The percentage correct is listed above each graph.

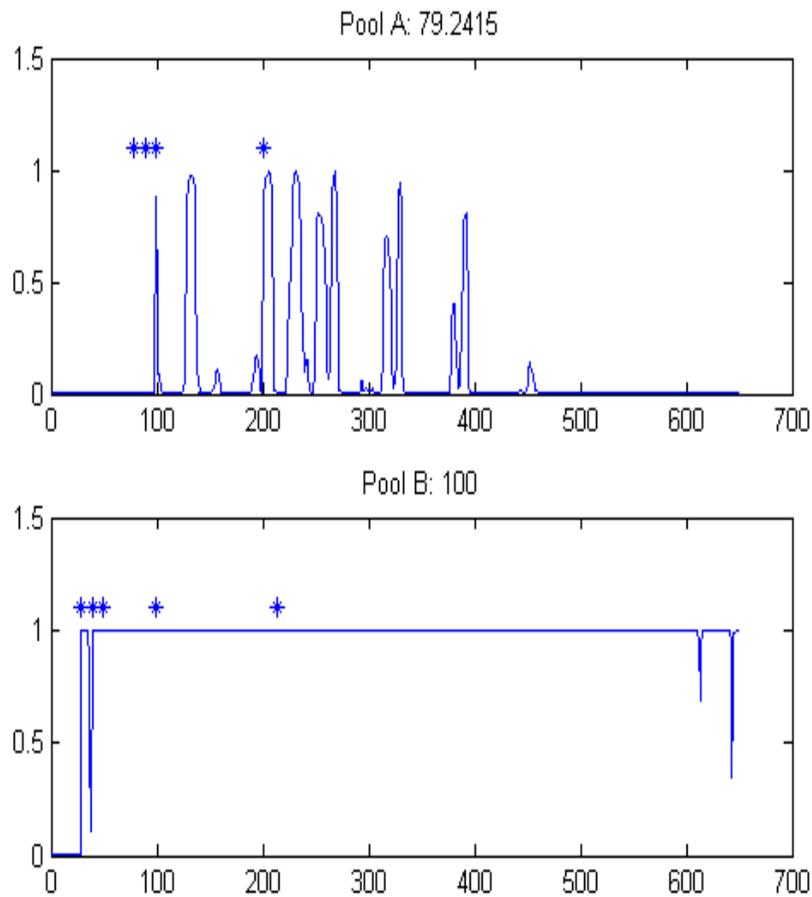


FIGURE 7: Shown above is an example of an extra rock being thrown in 100 time steps after the last possible rock from pattern A or B. The system is trained to recognize the difference between pattern A and B before the extra rock is thrown in. This way, the extra rock represents noise. If the network can still classify the patterns correctly, then it can generalize in this case. As can be seen, generalization occurs here.

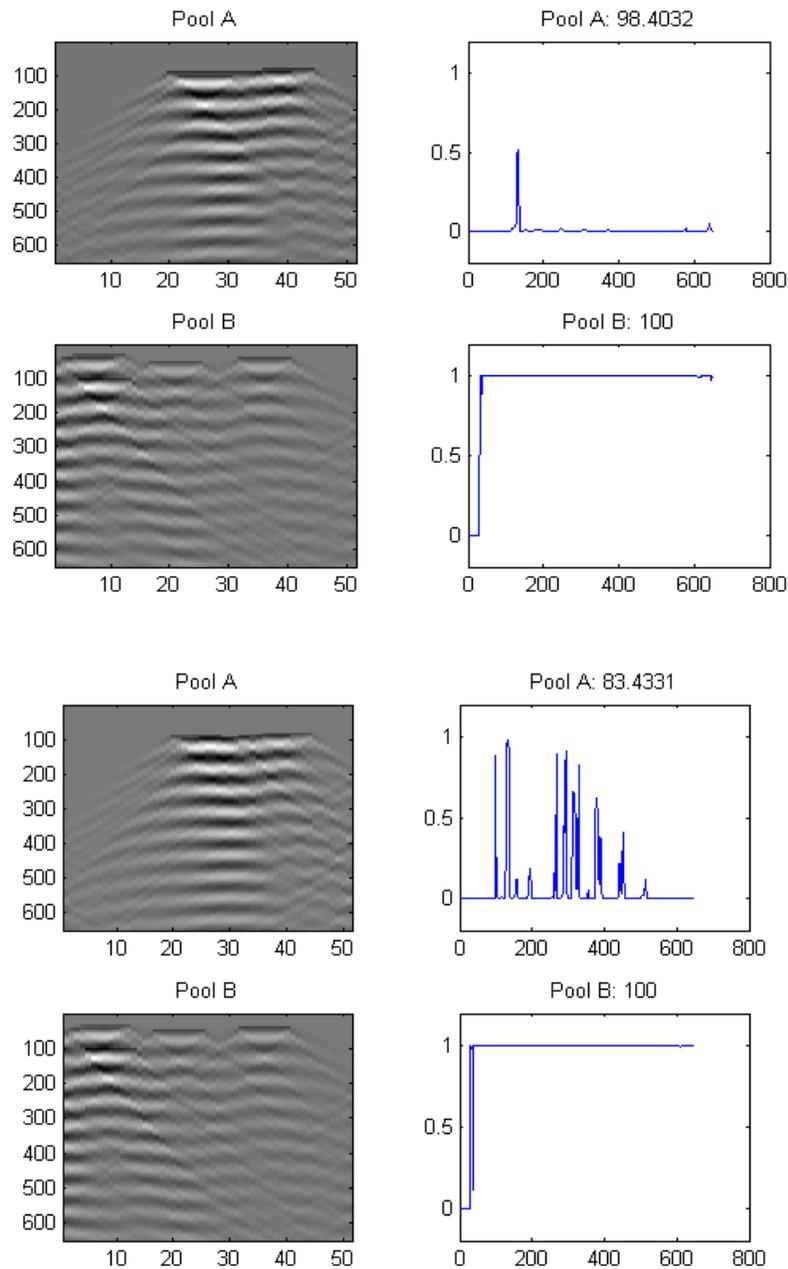


FIGURE 8: The four upper graphs represent a trained liquid state machine to classify pool A vs. B. The outputs are very accurate in this case. The four lower graphs represent the same system but with inputs that are moved slightly in time by a random amount. This “jitter” is another form of noise. The system can still classify the two patterns, though there is a noticeable reduction in performance.

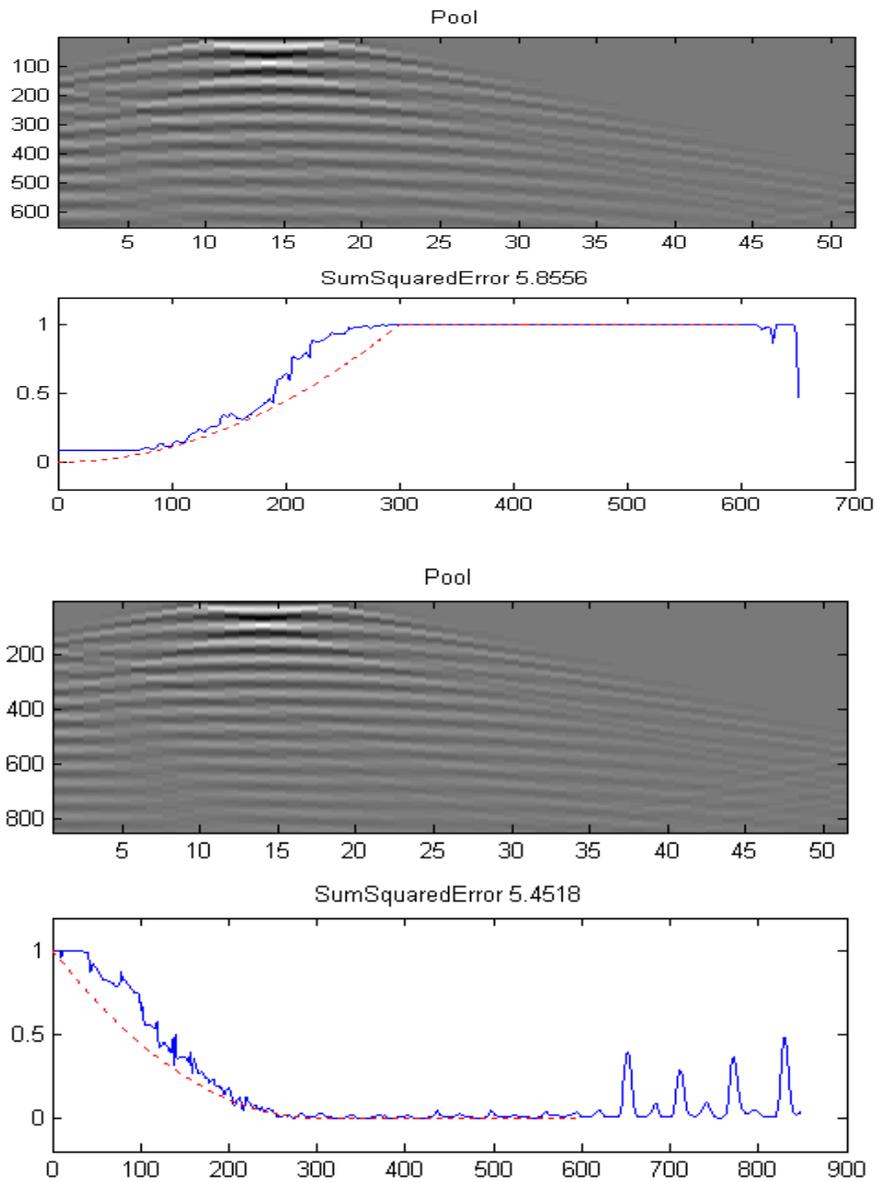


FIGURE 9: Two attempts were made to correctly identify the timing of a liquid since the most recent rock throw. The target was a quadratic increasing or decreasing. Note the ringing that starts after the training data (red dashed line).

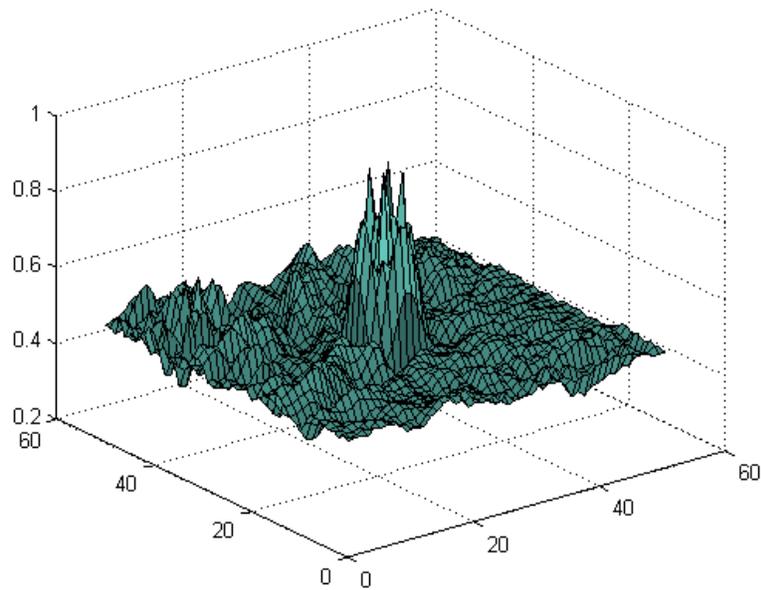


FIGURE 10: The 2D pond after a rock was thrown to the middle of it. Previously, several other rocks were also thrown in.

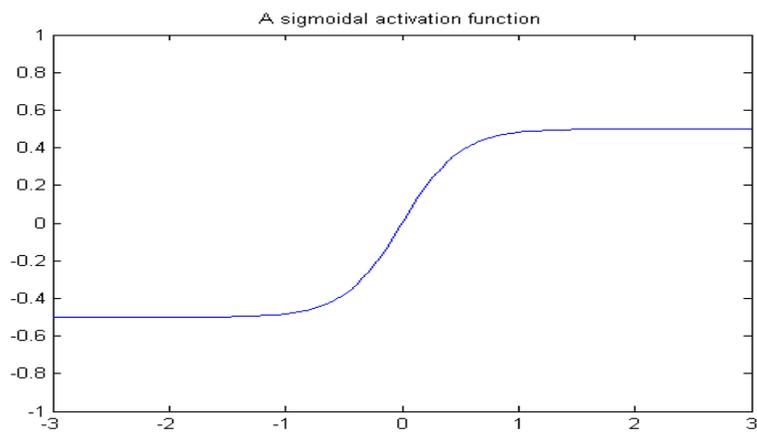


FIGURE 11: A sigmoidal activation function. This curve was used to scale the outputs of each element in the 2D pond model, thus allowing the system to more closely resemble neurons which have a threshold above which they fire action potentials, and below which they have small graded potentials.