Hopfield Network*

Jason Brownlee
jasonb@CleverAlgorithms.com
The Clever Algorithms Project
http://www.CleverAlgorithms.com

November 24, 2010
Technical Report: CA-TR-20101124-1

Abstract

The Clever Algorithms project aims to describe a large number of Artificial Intelligence algorithms in a complete, consistent, and centralized manner, to improve their general accessibility. The project makes use of a standardized algorithm description template that uses well-defined topics that motivate the collection of specific and useful information about each algorithm described. This report describes the Hopfield Network algorithm using the standardized algorithm template.

Keywords: `Clever, Algorithms, Description, Optimization, Hopfield, Network`

1 Introduction

The Clever Algorithms project aims to describe a large number of algorithms from the fields of Computational Intelligence, Biologically Inspired Computation, and Metaheuristics in a complete, consistent and centralized manner [1]. The project requires all algorithms to be described using a standardized template that includes a fixed number of sections, each of which is motivated by the presentation of specific information about the technique [2]. This report describes the Hopfield Network algorithm using the standardized algorithm template.

2 Name

Hopfield Network, HN, Hopfield Model

3 Taxonomy

The Hopfield Network is a Neural Network and belongs to the field of Artificial Neural Networks and Neural Computation. It is a Recurrent Neural Network and is related to other recurrent networks such as the Bidirectional Associative Memory (BAM). It is is generally related to feedforward Artificial Neural Networks such as the Perceptron and the Back-propagation algorithm.

4 Inspiration

The Hopfield Network algorithm is inspired by the associated memory properties of the human brain.

5 Metaphor

Through the training process, the weights in the network may be thought to minimize an energy function and slide down an energy surface. In a trained network, each pattern presented to the network provides an attractor, where progress is made towards the point of attraction by propagating information around the network.

6 Strategy

The information processing objective of the system is to associate the components of an input pattern with a holistic representation of the pattern called Content Addressable Memory (CAM). This means that once trained, the system will recall whole patterns, give a portion or a noisy version of the input pattern.

7 Procedure

The Hopfield Network is comprised of a graphic data structure with weighted edges and separate procedures for training and applying the structure. The network structure is fully connected (a node connects to all other nodes except itself) and the edges (weights) between the nodes are bidirectional.

The weights of the network can be learned via a one-shot method (one-iteration through the patterns) if all patterns to be memorized by the network are known. Alternatively, the weights can be updated incrementally using the Hebb rule where weights are increased or decreased based on the difference between the actual and the expected output. The one-shot calculation of the network weights for a single node occurs as follows:

$$w_{i,j} = \sum_{k=1}^{N} v_k^i \times v_k^j \tag{1}$$

where $w_{i,j}$ is the weight between neuron i and j, N is the number of input patterns, v is the input pattern and v_k^i is the i^{th} attribute on the k^{th} input pattern.

The propagation of the information through the network can be asynchronous where a random node is selected each iteration, or synchronously, where the output is calculated for each node before being applied for the whole network. Propagation of the information continues until no more changes are made or until a maximum number of iterations has completed, after which the output pattern from the network can be read. The activation for a single node is calculated as follows:

$$n_i = \sum_{j=1}^{n} w_{i,j} \times n_j \tag{2}$$

where n_i is the activation of the i^{th} neuron, $w_{i,j}$ with the weight between the nodes i and j, and n_j is the output of the j^{th} neuron. The activation is transferred into an output using a transfer function, typically a step function as follows:

$$transfer(n_i) = \begin{cases} 1 & if \geq \theta \\ -1 & if < \theta \end{cases}$$

where the threshold θ is typically fixed at 0.

8 Heuristics

- The Hopfield network may be used to solve the recall problem of matching cues for an input pattern to an associated pre-learned pattern.
- The transfer function for turning the activation of a neuron into an output is typically a step function $f(a) \in \{-1, 1\}$ (preferred), or more traditionally $f(a) \in \{0, 1\}$.
- The input vectors are typically normalized to boolean values $x \in [-1, 1]$.
- The network can be propagated asynchronously (where a random node is selected and output generated), or synchronously (where the output for all nodes are calculated before being applied).
- Weights can be learned in a one-shot or incremental method based on how much information is known about the patterns to be learned.
- All neurons in the network are typically both input and output neurons, although other network topologies have been investigated (such as the designation of input and output neurons).
- A Hopfield network has limits on the patterns it can store and retrieve accurately from memory, described by $N < 0.15 \times n$ where N is the number of patterns that can be stored and retrieved and n is the number of nodes in the network.

9 Code Listing

Listing 1 provides an example of the Hopfield Network algorithm implemented in the Ruby Programming Language. The problem is an instance of a recall problem where patters are described in terms of a 3×3 matrix of binary values ($\in \{-1, 1\}$). Once the network has learned the patterns, the system is exposed to perturbed versions of the patterns (with errors introduced) and must respond with the correct pattern. Three patterns are used in this example, specifically a 'C', and 'L' and an 'I'.

The algorithm is an implementation of the Hopfield Network with a one-short training method for the network weights, given that all patterns are already known. The information is propagated through the network using an asynchronous method, which is repeated until no more changes in the node outputs are detected. The patterns are displayed to the console during the testing of the network, with the outputs converted from $\{-1, 1\}$ to $\{0, 1\}$ for readability.

```ruby
def random_vector(minmax)
  return Array.new(minmax.length) do |i|
    minmax[i][0] + ((minmax[i][1] - minmax[i][0]) * rand())
  end
end

def initialize_weights(problem_size)
  minmax = Array.new(problem_size + 1) {[-0.5,0.5]}
  return random_vector(minmax)
end

def create_neuron(num_inputs)
  neuron = {}
  neuron[:weights] = initialize_weights(num_inputs)
  neuron[:output] = -1
  return neuron
```

```ruby
end

def transfer(activation)
  return (activation >= 0) ? 1 : -1
end

def propagate_was_change?(neurons, vector)
  i = rand(neurons.length)
  activation = 0
  neurons.each_with_index do |other, j|
    activation += other[:weight][i]*other[:output] if i!=j
  end
  output = transfer(activation)
  change = (output==neurons[i][:output])
  neurons[i][:output] = output
  return change
end

def get_output(neurons, pattern)
  vector = pattern.flatten
  neurons.each_with_index {|neuron,i| neuron[:output] = vector[i]}
  change = propagate(neurons, vector) while change
  return Array.new(neurons.length){|i| neurons[i][:output]}
end

def train_network(neurons, patters)
  neurons.each_with_index do |neuron, i|
    for j in ((i+1)...neurons.length) do
      next if i==j
      wij = 0
      patters.each do |pattern|
        vector = pattern.flatten
        wij += vector[i]*vector[j]
      end
      neurons[i][:weights][j] = wij
      neurons[j][:weights][i] = wij
    end
  end
end

def to_binary(vector)
  return Array.new(vector.length){|i| ((vector[i]==-1) ? 0 : 1)}
end

def print_patterns(provided, expected, actual)
  p, e, a = to_binary(provided), to_binary(expected), to_binary(actual)
  p1, p2, p3 = p[0..2].join(', '), p[3..5].join(', '), p[6..8].join(', ')
  e1, e2, e3 = e[0..2].join(', '), e[3..5].join(', '), e[6..8].join(', ')
  a1, a2, a3 = a[0..2].join(', '), a[3..5].join(', '), a[6..8].join(', ')
  puts "Provided Expected    Got"
  puts "#{p1}    #{e1}     #{a1}"
  puts "#{p2}    #{e2}     #{a2}"
  puts "#{p3}    #{e3}     #{a3}"
end

def calculate_error(expected, actual)
  sum = 0
  expected.each_with_index do |v, i|
    sum += (expected[i] - actual[i]).abs
  end
  return sum
end

```

```ruby
def perturb_pattern(vector)
  perturbed = Array.new(vector.length)
  vector.each_with_index do |v,i|
    if rand() < (1.0/vector.length.to_f)*0.5
      perturbed[i] = ((vector[i]==1) ? -1 : 1)
    else
      perturbed[i] = vector[i]
    end
  end
  return perturbed
end

def test_network(neurons, patters)
  error = 0.0
  patters.each do |pattern|
    vector = pattern.flatten
    perturbed = perturb_pattern(vector)
    output = get_output(neurons, perturbed)
    error += calculate_error(vector, output)
    print_patterns(perturbed, vector, output)
  end
  error /= patters.length.to_f
  puts "Final Result: avg pattern error=#{error}"
end

def run(patters, num_inputs)
  neurons = Array.new(num_inputs) { create_neuron(num_inputs) }
  train_network(neurons, patters)
  test_network(neurons, patters)
end

if __FILE__ == $0
  # problem definition
  num_inputs = 9
  p1 = [[1,1,1],[1,-1,-1],[1,1,1]] # C
  p2 = [[1,-1,-1],[1,-1,-1],[1,1,1]] # L
  p3 = [[-1,1,-1],[-1,1,-1],[-1,1,-1]] # I
  patters = [p1, p2, p3]
  # execute the algorithm
  run(patters, num_inputs)
end
```

Listing 1: Hopfield Network algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

The Hopfield Network was proposed by Hopfield in 1982 where the basic model was described and related to an abstraction of the inspiring biological system [4]. This early work was extend by Hopfield to 'graded' neurons capable of outputting a continuous value through use of a logistic (sigmoid) transfer function [5]. An innovative work by Hopfield and Tank considered the use of the Hopfield network for solving combinatorial optimization problems, with a specific study into the system applied to instances of the Traveling Salesman Problem [6]. This was achieved with a large number of neurons and a representation that decoded the position of each position in the tour as a sub-problem on which a customized network energy function had to be minimized.

10.2 Learn More

Popovici and Boncut provide a summary of the Hopfield Network algorithm with worked examples [7]. Overviews of the Hopfield Network are provided in most good books on Artificial Neural Networks, such as [8]. Hertz, Krogh, and Palmer present an in depth study of the the field of Artificial Neural Networks with a detailed treatment of the Hopfield network from a statistical mechanics perspective [3].

11 Conclusions

This report described the Hopfield Network algorithm using the standardized algorithm template.

12 Contribute

Found a typo in the content or a bug in the source code? Are you an expert in this technique and know some facts that could improve the algorithm description for all? Do you want to get that warm feeling from contributing to an open source project? Do you want to see your name as an acknowledgment in print?

Two pillars of this effort are i) that the best domain experts are people outside of the project, and ii) that this work is subjected to continuous improvement. Please help to make this work less wrong by emailing the author 'Jason Brownlee' at `jasonb@CleverAlgorithms.com` or visit the project website at `http://www.CleverAlgorithms.com`.

References

[1] Jason Brownlee. The clever algorithms project: Overview. Technical Report CA-TR-20100105-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[2] Jason Brownlee. A template for standardized algorithm descriptions. Technical Report CA-TR-20100107-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[3] J. Hertz, Krogh A., and R. G. Palmer. *Introduction to the theory of neural computation.* Westview Press, 1991.

[4] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences of the USA*, volume 79, pages 2554–2558, April 1982.

[5] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. In *Proceedings of the National Academy of Sciences*, volume 81, pages 3088–3092, 1984.

[6] J. J. Hopfield and D. W. Tank. "neural" computation of decisions in optimization problems. *Biological Cybernetics*, 55:141–146, 1985.

[7] N. Popovici and M. Boncut. On the hopfield algorithm. foundations and examples. *General Mathematics*, 2:35–50, 2005.

[8] Raul Rojas. *Neural Networks - A Systematic Introduction*, chapter 13. The Hopfield Model. Springer, 1996.