Learning Vector Quantization*

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

November 18, 2010
Technical Report: CA-TR-20101118-1

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 Learning Vector Quantization algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Learning, Vector, Quantization`

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 Learning Vector Quantization algorithm using the standardized template.

2 Name

Learning Vector Quantization, LVQ

3 Taxonomy

The Learning Vector Quantization algorithm belongs to the field of Artificial Neural Networks, and more broadly to the field of Computational Intelligence. Learning Vector Quantization is related to the Kohonen Self-Organizing Map which is a similar algorithm with the addition of a connections between the neurons. Additionally, LVQ is a baseline technique that was defined with a few variants LVQ1, LVQ2, LVQ2.1, LVQ3, OLVQ1, and OLVQ3 as well as many third-party extensions and refinements too numerous to list.

4 Inspiration

The Learning Vector Quantization algorithm is related to the Self-Organizing Map which is in turn inspired by the self-organizing capabilities of neurons in the visual cortex.

5 Strategy

The information processing objective of the algorithm is to prepare a set of codebook (or prototype) vectors in the domain of the observed input data samples and to use these vectors to classify unseen examples. An initially random pool of vectors is prepared which are then exposed to training samples. A winner-take-all strategy is employed where one or more of the most similar vectors to a given input pattern are selected and adjusted to be closer to the input vector, and in some cases, further away from the winner for runner's up. The repetition of this process results in the distribution of codebook vectors in the input space which approximate the underlying distribution of samples from the test dataset.

6 Procedure

Vector Quantization is a technique from signal processing density functions are approximated with prototype vectors for applications such as compression. Learning Vector Quantization is similar in principle, although the prototype vectors are learned through a supervised winner-take-all method.

Algorithm 1 provides a high-level pseudo-code for preparing codebook vectors using the Learning Vector Quantization method. Codebook vectors are initialized to small floating point values, or sampled from an available dataset. The Best Matching Unit (BMU) is the codebook vector from the pool that has the minimum distance to an input vector. A distance measure between input patterns must be defined. For real-valued vectors, this is commonly the Euclidean distance:

$$dist(x, c) = \sum_{i=1}^{n} (x_i - c_i)^2 \tag{1}$$

where n is the number of attributes, x is the input vector and c is a given codebook vector.

Algorithm 1: Pseudo Code for the Learning Vector Quantization algorithm (LVQ1).

Input: ProblemSize, InputPatterns, $iterations_{max}$, $CodebookVectors_{num}$ $learn_{rate}$
Output: CodebookVectors

CodebookVectors $\leftarrow$ `InitializeCodebookVectors(`$CodebookVectors_{num}$, ProblemSize`)`;
for $i = 1$ **to** $iterations_{max}$ **do**
  $Pattern_i \leftarrow$ `SelectInputPattern(InputPatterns)`;
  $Bmu_i \leftarrow$ `SelectBestMatchingUnit(`$Pattern_i$, CodebookVectors`)`;
  foreach $Bmu_i^{attribute} \in Bmu_i$ **do**
    if $Bmu_i^{class} \equiv Pattern_i^{class}$ **then**
      $Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} + learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$
    else
      $Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} - learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$
    end
  end
end
return CodebookVectors;

7 Heuristics

- Learning Vector Quantization was designed for classification problems that have existing data sets that can be used by supervise the learning by the system. The algorithm does not support regression problems.
- LVQ is non-parametric, meaning that it does not rely on assumptions about that structure of the function that is is approximating.
- Real-values in input vectors should be normalized such that $x \in [0, 1)$.
- Euclidean distance is commonly used to measure the distance between real-valued vectors, although other distance measures may be used (such as dot product), and data specific distance measures may be required for non-scalar attributes.
- There should be sufficient training iterations to expose all the training data to the model multiple times.
- The learning rate is typically linearly decayed over the training period from an initial value to close to zero.
- The more complex the class distribution, the more codebook vectors that will be required, some problems may need thousands.
- Multiple passes of the LVQ training algorithm are suggested for more robust usage, where the first pass has a large learning rate to prepare the codebook vectors and the second pass has a low learning rate and runs for a long time (perhaps $10\times$ more iterations).

8 Code Listing

Listing 1 provides an example of the Learning Vector Quantization algorithm implemented in the Ruby Programming Language. The problem is a contrived classification problem in a 2-dimensional domain $x \in [0, 1], y \in [0, 1]$ with two classes: 'A' ($x \in [0, 0.4999999], y \in [0, 0.4999999]$) and 'B' ($x \in [0.5, 1], y \in [0.5, 1]$).

The algorithm was implemented using the LVQ1 variant where the best matching codebook vector is located and moved toward the the input vector if it is the same class, or away if the classes differ. A linear decay was used for the learning rate that was updated after each pattern was exposed to the model. The implementation can easily be extended to the other variants of the method.

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

def generate_random_pattern(domain)
  classes = domain.keys
  selected_class = rand(classes.length)
  pattern = {}
  pattern[:class_number] = selected_class
  pattern[:class_label] = classes[selected_class]
  pattern[:vector] = random_vector(domain[classes[selected_class]])
  return pattern
end

def initialize_vectors(domain, num_vectors)
  classes = domain.keys
```

```
  codebook_vectors = []
  num_vectors.times do
    selected_class = rand(classes.length)
    codebook = {}
    codebook[:class_label] = classes[selected_class]
    codebook[:vector] = random_vector([[0,1],[0,1]])
    codebook_vectors << codebook
  end
  return codebook_vectors
end

def euclidean_distance(v1, v2)
  sum = 0.0
  v1.each_with_index do |v, i|
    sum += (v1[i]-v2[i])**2.0
  end
  return Math.sqrt(sum)
end

def get_best_matching_unit(codebook_vectors, pattern)
  best, b_dist = nil, nil
  codebook_vectors.each do |codebook|
    dist = euclidean_distance(codebook[:vector], pattern[:vector])
    best,b_dist = codebook,dist if b_dist.nil? or dist<b_dist
  end
  return best
end

def update_codebook_vector(bmu, pattern, lrate)
  bmu[:vector].each_with_index do |v,i|
    error = pattern[:vector][i]-bmu[:vector][i]
    if bmu[:class_label] == pattern[:class_label]
      bmu[:vector][i] += lrate * error
    else
      bmu[:vector][i] -= lrate * error
    end
  end
end

def train_network(codebook_vectors, domain, problem_size, iterations, learning_rate)
  iterations.times do |iter|
    pattern = generate_random_pattern(domain)
    bmu = get_best_matching_unit(codebook_vectors, pattern)
    lrate = learning_rate * (1.0-(iter.to_f/iterations.to_f))
    puts "> train lrate=#{lrate} got=#{bmu[:class_label]}, exp=#{pattern[:class_label]}"
    update_codebook_vector(bmu, pattern, lrate)
  end
end

def test_network(codebook_vectors, domain)
  correct = 0
  100.times do
    pattern = generate_random_pattern(domain)
    bmu = get_best_matching_unit(codebook_vectors, pattern)
    correct += 1 if bmu[:class_label] == pattern[:class_label]
  end
  puts "Finished test with a score of #{correct}/#{100} (#{(correct/100)*100}%)"
end

def run(domain, problem_size, iterations, num_vectors, learning_rate)
  codebook_vectors = initialize_vectors(domain, num_vectors)
  train_network(codebook_vectors, domain, problem_size, iterations, learning_rate)
  test_network(codebook_vectors, domain)
```

```
end

if __FILE__ == $0
  problem_size = 2
  domain = {"A"=>[[0,0.4999999],[0,0.4999999]],"B"=>[[0.5,1],[0.5,1]]}
  learning_rate = 0.3
  iterations = 1000
  num_vectors = 20

  run(domain, problem_size, iterations, num_vectors, learning_rate)
end
```

Listing 1: Learning Vector Quantization algorithm in the Ruby Programming Language

9 References

9.1 Primary Sources

The Learning Vector Quantization algorithm was described by Kohonen in 1988 [8], and was further described in the same year by Kohonen [6] and benchmarked by Kohonen, Barna, and Chrisley [4].

9.2 Learn More

Kohonen provides a detailed overview of the state of LVQ algorithms and variants (LVQ1, LVQ2, and LVQ2.1) [3]. The technical report that comes with the LVQ_PAK software (written by Kohonen and his students) provides both an excellent summary of the technique and its main variants, as well as summarizing the important considerations when applying the approach [5]. The seminal book on Learning Vector Quantization and the Self-Organizing Map is "Self-Organizing Maps" by Kohonen, which includes a chapter (Chapter 6) dedicated to LVQ and its variants [7].

10 Conclusions

This report described the Learning Vector Quantization algorithm using the standardized template. The algorithm implementation and heuristics for this algorithm were based on past work by the author for the WEKA machine learning workbench, available at http://wekaclassalgos.sourceforge.net.

11 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] T. Kohonen. Improved versions of learning vector quantization. In *IJCNN International Joint Conference on Neural Networks*, volume 1, pages 545–550. IEEE Press, 1990.

[4] T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks: benchmarking studies. In *IEEE International Conference on Neural Networks*, volume 1, pages 61–68, 1988.

[5] T. Kohonen, J Hynninen, J. Kangas, J. Laaksonen, and K. Torkkola. Lvq-pak: The learning vector quantization program package. Technical Report A30, Helsinki University of Technology, Laboratory of Computer and Information Science, Rakentajanaukio, 1996.

[6] Teuvo Kohonen. An introduction to neural computing. *Neural Networks*, 1(1):3–16, 1988.

[7] Teuvo Kohonen. *Self-Organizing Maps*. Springer, 1995.

[8] Tsang Kohonen. Learning vector quantization. *Neural Networks*, 1:303, 1988.