Learning Vector Quantization (LVQ)

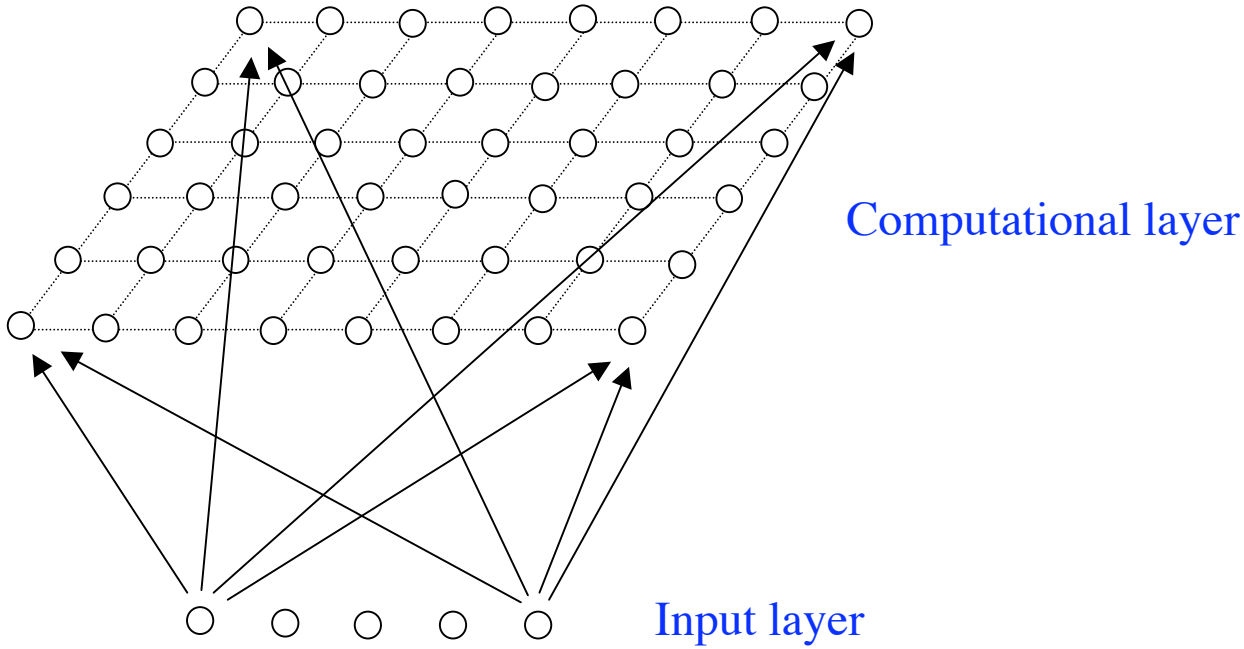
Introduction to Neural Computation : Guest Lecture 2

© John A. Bullinaria, 2007

1. The SOM Architecture and Algorithm
2. What is Vector Quantization?
3. The Encoder-Decoder Model
4. Relation between a SOM and Noisy Encoder-Decoder
5. Voronoi Tessellation
6. Learning Vector Quantization (LVQ)

The Architecture a Self Organizing Map

We shall concentrate on the SOM system known as a ***Kohonen Network***. This has a feed-forward structure with a single computational layer of neurons arranged in rows and columns. Each neuron is fully connected to all the source units in the input layer:



A one dimensional map will just have a single row or column in the computational layer.

The SOM Algorithm

The aim is to learn a ***feature map*** from the spatially ***continuous input space***, in which our input vectors live, to the low dimensional spatially ***discrete output space***, which is formed by arranging the computational neurons into a grid.

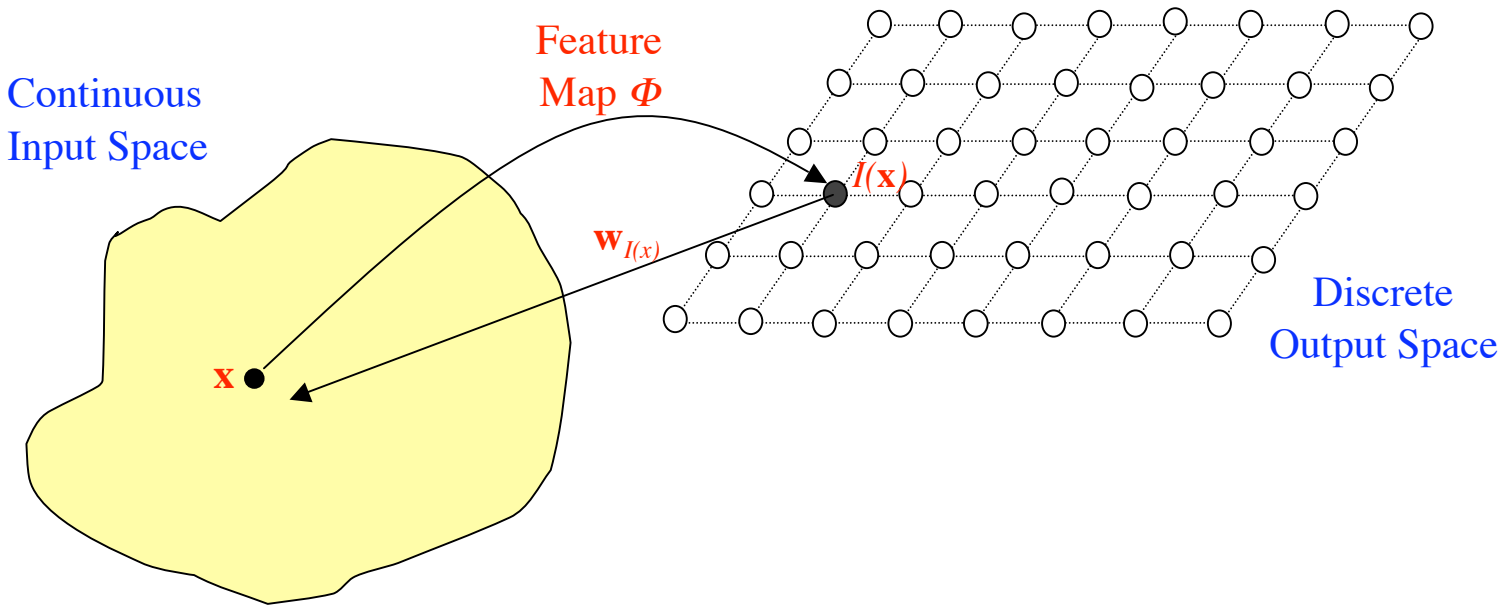
The stages of the SOM algorithm that achieves this can be summarised as follows:

1. ***Initialization*** – Choose random values for the initial weight vectors $\mathbf{w}_j$.
2. ***Sampling*** – Draw a sample training input vector $\mathbf{x}$ from the input space.
3. ***Matching*** – Find the winning neuron $I(\mathbf{x})$ that has weight vector closest to the input vector, i.e. the minimum value of $d_j(\mathbf{x}) = \sum_{i=1}^{D}(x_i - w_{ji})^2$.
4. ***Updating*** – Apply the weight update equation $\Delta w_{ji} = \eta(t)\, T_{j,I(\mathbf{x})}(t)\, (x_i - w_{ji})$ where $T_{j,I(\mathbf{x})}(t)$ is a Gaussian neighbourhood and $\eta(t)$ is the learning rate.
5. ***Continuation*** – keep returning to step 2 until the feature map stops changing.

This leads to a feature map with numerous useful properties.

Properties of the Feature Map

Once the SOM algorithm has converged, the feature map displays important statistical characteristics of the input space. Given an input vector $\mathbf{x}$, the feature map Φ provides a winning neuron $I(\mathbf{x})$ in the output space, and the weight vector $\mathbf{w}_{I(\mathbf{x})}$ provides the coordinates of the image of that neuron in the input space.



Properties: Approximation, Topological ordering, Density matching, Feature selection.

What is a Vector Quantization?

We have already noted that one aim of using a Self Organizing Map (SOM) is to encode a large set of input vectors $\{\mathbf{x}\}$ by finding a smaller set of "representatives" or "prototypes" or "code-book vectors" $\{\mathbf{w}_{I(\mathbf{x})}\}$ that provide a good approximation to the original input space. This is the basic idea of ***vector quantization*** theory, the motivation of which is ***dimensionality reduction*** or ***data compression***.

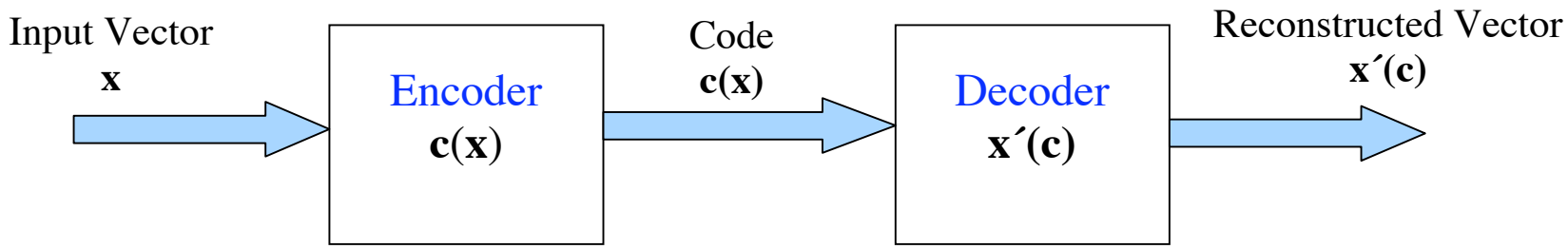
In effect, the error of the vector quantization approximation is the total squared distance

$$D = \sum_{\mathbf{x}} \left\| \mathbf{x} - \mathbf{w}_{I(\mathbf{x})} \right\|^2$$

between the input vectors $\{\mathbf{x}\}$ and their representatives $\{\mathbf{w}_{I(\mathbf{x})}\}$, and we clearly wish to minimize this. We shall see that performing a gradient descent style minimization of D does lead to the SOM weight update algorithm, which confirms that it is generating the best possible discrete low dimensional approximation to the input space (at least assuming it does not get trapped in a local minimum of the error function).

The Encoder – Decoder Model

Probably the best way to think about vector quantization is in terms of general ***encoders*** and ***decoders***. Suppose **c**(**x**) acts as an encoder of the input vector **x**, and **x´(c)** acts as a decoder of **c**(**x**), then we can attempt to get back to **x** with minimal loss of information:



Generally, the input vector **x** will be selected at random according to some probability function $p(\mathbf{x})$. Then the optimum encoding-decoding scheme is determined by varying the functions **c**(**x**) and **x´(c)** to minimize the ***expected distortion*** defined by

$$D = \sum_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}'(\mathbf{c}(\mathbf{x}))\|^2 = \int d\mathbf{x} p(\mathbf{x}) \|\mathbf{x} - \mathbf{x}'(\mathbf{c}(\mathbf{x}))\|^2$$

The Generalized Lloyd Algorithm

The necessary conditions for minimizing the expected distortion D in general situations are embodied in the two conditions of the ***generalized Lloyd algorithm***:

Condition 1. Given the input vector $\mathbf{x}$, choose the code $\mathbf{c} = \mathbf{c}(\mathbf{x})$ to minimize the squared error distortion $\|\mathbf{x} - \mathbf{x}'(\mathbf{c})\|^2$.

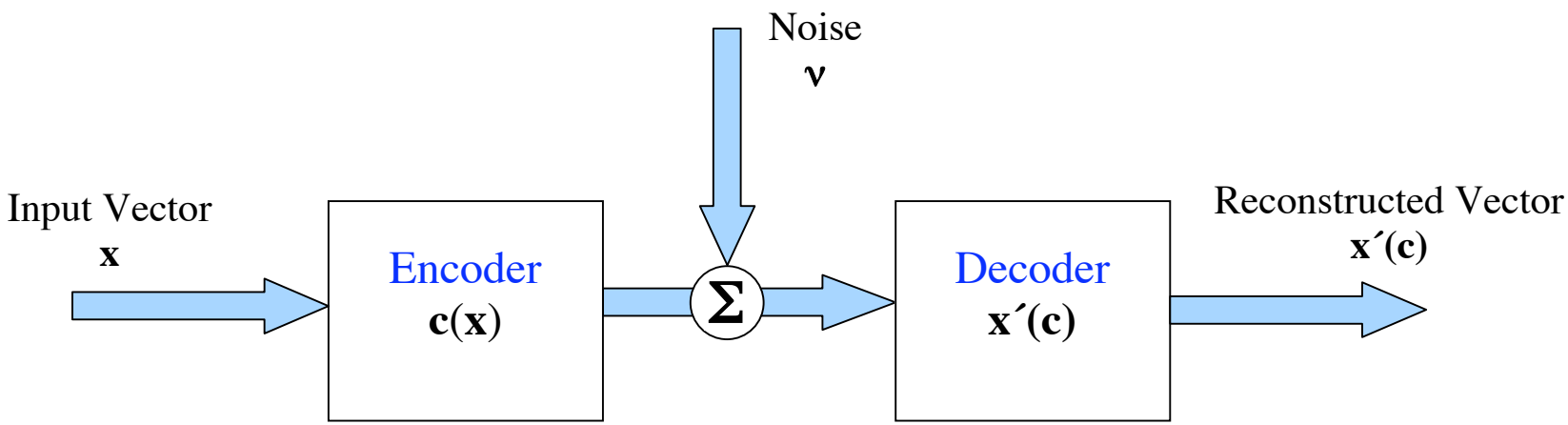
Condition 2. Given the code $\mathbf{c}$, compute the reconstruction vector $\mathbf{x}'(\mathbf{c})$ as the centroid of those input vectors $\mathbf{x}$ that satisfy condition 1.

To implement vector quantization, the algorithm works in batch mode by alternately optimizing the encoder $\mathbf{c}(\mathbf{x})$ in accordance with condition 1, and then optimizing the decoder in accordance with condition 2, until D reaches a minimum.

To overcome the problem of local minima, it may be necessary to run the algorithm several times with different initial code vectors.

The Noisy Encoder – Decoder Model

In real applications the encoder-decoder system will also have to cope with noise in the communication channel. We can treat the noise as an additive random variable $\boldsymbol{\nu}$ with probability density function $\pi(\boldsymbol{\nu})$, so the model becomes



It is not difficult to see that the expected distortion is now given by

$$D_v = \sum_{\mathbf{x}} \sum_{\boldsymbol{\nu}} \|\mathbf{x} - \mathbf{x}'(\mathbf{c}(\mathbf{x}) + \boldsymbol{\nu})\|^2 = \int d\mathbf{x} p(\mathbf{x}) \int d\boldsymbol{\nu} \pi(\boldsymbol{\nu}) \|\mathbf{x} - \mathbf{x}'(\mathbf{c}(\mathbf{x}) + \boldsymbol{\nu})\|^2$$

The Generalized Lloyd Algorithm with Noise

To minimize the modified expected distortion D_ν we can compute the relevant partial derivatives and use the modified ***generalized Lloyd algorithm***:

Condition 1. Given the input vector $\mathbf{x}$, choose the code $\mathbf{c} = \mathbf{c}(\mathbf{x})$ to minimize the distortion measure $\int d\boldsymbol{\nu}\pi(\boldsymbol{\nu})\|\mathbf{x} - \mathbf{x}'(\mathbf{c}(\mathbf{x}) + \boldsymbol{\nu})\|^2$.

Condition 2. Given the code $\mathbf{c}$, compute the reconstruction vector $\mathbf{x}'(\mathbf{c})$ to satisfy $\mathbf{x}'(\mathbf{c}) = \int d\mathbf{x}p(\mathbf{x})\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))\mathbf{x} \Big/ \int d\mathbf{x}p(\mathbf{x})\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))$.

If we set the noise density function $\pi(\boldsymbol{\nu})$ to be the Dirac delta function $\delta(\boldsymbol{\nu})$, that is zero everywhere except at $\boldsymbol{\nu} = 0$, these conditions reduce to the conditions we had before in the no noise case. We can usually approximate Condition 1 by a simple nearest neighbour approach, and then we can determine that the appropriate iterative updates of the reconstruction vector $\mathbf{x}'(\mathbf{c})$ for condition 2 are $\Delta\mathbf{x}'(\mathbf{c}) \sim \pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))(\mathbf{x} - \mathbf{x}'(\mathbf{c}))$.

Relation between a SOM and Noisy Encoder–Decoder

We can now see that there is a direct correspondence between the SOM algorithm and the noisy encoder-decoder model:

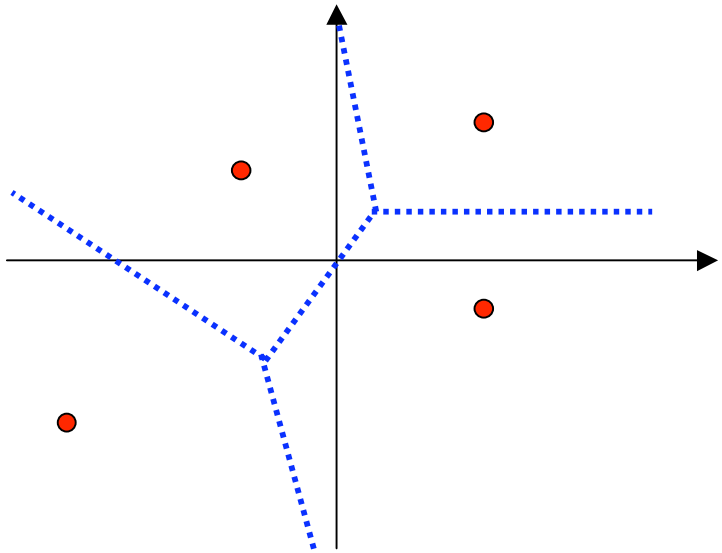
Noisy Encoder-Decoder Model	SOM Algorithm
Encoder $\mathbf{c}(\mathbf{x})$	Best matching neuron $I(\mathbf{x})$
Reconstruction vector $\mathbf{x}'(\mathbf{c})$	Connection weight vector $\mathbf{w}_j$
Probability density function $\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))$	Neighbourhood function $T_{j,I(\mathbf{x})}$

This provides us with a proof that the SOM algorithm is a vector quantization algorithm which provides a good approximation to the input space.

Note that the topological neighbourhood function $T_{j,I(\mathbf{x})}$ in the SOM algorithm has the form of a probability density function.

Voronoi Tessellation

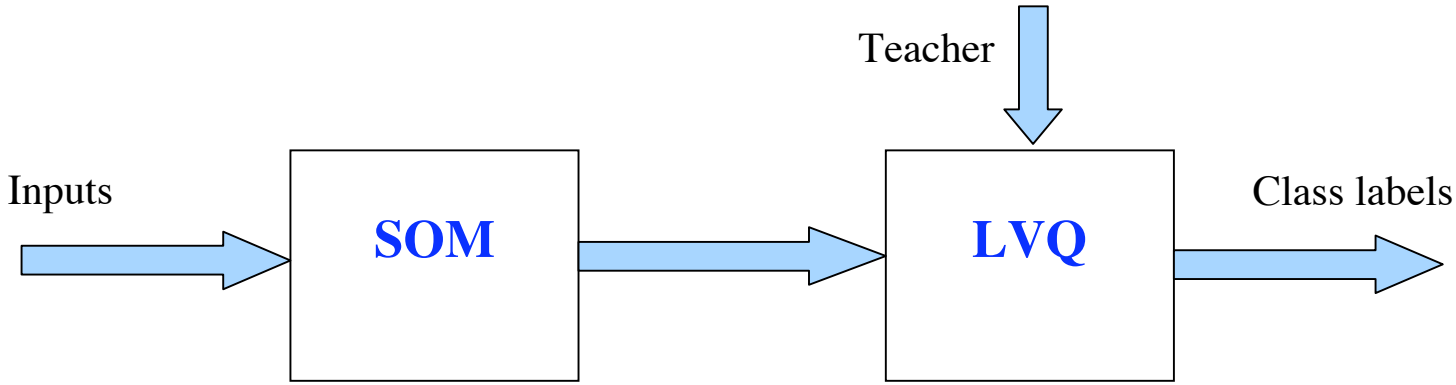
A vector quantizer with minimum encoding distortion is called a ***Voronoi quantizer*** or ***nearest-neighbour quantizer***. The input space is partitioned into a set of ***Voronoi or nearest neighbour cells*** each containing an associated ***Voronoi or reconstruction vector***:



The SOM algorithm provides a useful method for computing the Voronoi vectors (as weight vectors) in an unsupervised manner. One common application is to use it for finding good centres (input to hidden unit weights) in RBF networks.

Learning Vector Quantization (LVQ)

Learning Vector Quantization (LVQ) is a supervised version of vector quantization that can be used when we have labelled input data. This learning technique uses the class information to reposition the Voronoi vectors slightly, so as to improve the quality of the classifier decision regions. It is a two stage process – a SOM followed by LVQ:



This is particularly useful for ***pattern classification*** problems. The first step is feature selection – the unsupervised identification of a reasonably small set of features in which the essential information content of the input data is concentrated. The second step is the classification where the feature domains are assigned to individual classes.

The LVQ Approach

The basic LVQ approach is quite intuitive. It is based on a standard trained SOM with input vectors $\{\mathbf{x}\}$ and weights/Voronoi vectors $\{\mathbf{w}_j\}$.

The new factor is that the input data points have associated class information. This allows us to use the known classification labels of the inputs to find the best classification label for each $\mathbf{w}_j$, i.e. for each Voronoi cell. For example, by simply counting up the total number of instances of each class for the inputs within each cell.

Then each new input without a class label can be assigned to the class of the Voronoi cell it falls within.

The problem with this is that, in general, it is unlikely that the Voronoi cell boundaries will match up with the best possible classification boundaries, so the classification generalization performance will not be as good as possible. The obvious solution is to shift the Voronoi cell boundaries so they better match the classification boundaries.

The LVQ Algorithm

The basic LVQ algorithm is a straightforward method for shifting the Voronoi cell boundaries to result in better classification. It starts from the trained SOM with input vectors $\{\mathbf{x}\}$ and weights/Voronoi vectors $\{\mathbf{w}_j\}$, and uses the classification labels of the inputs to find the best classification label for each $\mathbf{w}_j$. The LVQ algorithm then checks the input classes against the Voronoi cell classes and moves the $\mathbf{w}_j$ appropriately:

1. If the input $\mathbf{x}$ and the associated Voronoi vector/weight $\mathbf{w}_{I(\mathbf{x})}$ (i.e. the weight of the winning output node $I(\mathbf{x})$) have the same class label, then move them closer together by $\Delta\mathbf{w}_{I(\mathbf{x})}(t) = \beta(t)(\mathbf{x} - \mathbf{w}_{I(\mathbf{x})}(t))$ as in the SOM algorithm.
2. If the input $\mathbf{x}$ and associated Voronoi vector/weight $\mathbf{w}_{I(\mathbf{x})}$ have the different class labels, then move them apart by $\Delta\mathbf{w}_{I(\mathbf{x})}(t) = -\beta(t)(\mathbf{x} - \mathbf{w}_{I(\mathbf{x})}(t))$.
3. Voronoi vectors/weights $\mathbf{w}_j$ corresponding to other input regions are left unchanged with $\Delta\mathbf{w}_j(t) = 0$.

where $\beta(t)$ is a learning rate that decreases with the number of iterations/epochs of training. In this way we get better classification than by the SOM alone.

The LVQ2 Algorithm

A second, improved, LVQ algorithm known as LVQ2 is sometimes preferred because it comes closer in effect to Bayesian decision theory.

The same weight/vector update equations are used as in the standard LVQ, but they only get applied under certain conditions, namely when:

1. The input vector $\mathbf{x}$ is incorrectly classified by the associated Voronoi vector $\mathbf{w}_{I(\mathbf{x})}$.
2. The next nearest Voronoi vector $\mathbf{w}_{S(\mathbf{x})}$ does give the correct classification, and
3. The input vector $\mathbf{x}$ is sufficiently close to the decision boundary (perpendicular bisector plane) between $\mathbf{w}_{I(\mathbf{x})}$ and $\mathbf{w}_{S(\mathbf{x})}$.

In this case, *both* vectors $\mathbf{w}_{I(\mathbf{x})}$ and $\mathbf{w}_{S(\mathbf{x})}$ are updated (using the *incorrect* and *correct* classification update equations respectively).

Various other variations on this theme exist (LVQ3, etc.), and this is still a fruitful research area for building better classification systems.

Overview and Reading

1. We began with an overview of SOMs and vector quantization.
2. We then looked at general encoder-decoder models and noisy encoder-decoder models, and the generalized Lloyd algorithm for optimizing them. This led to a clear relation between SOMs and vector quantization.
3. We ended by studying learning vector quantization (LVQ) from the point of view of Voronoi tessellation, and saw how the LVQ algorithm could optimize the class decision boundaries generated by a SOM.

Reading

1. Haykin: Section 9.5, 9.7, 9.8, 9.10, 9.11
2. Beale & Jackson: Sections 5.6
3. Gurney: Section 8.3.6
4. Hertz, Krogh & Palmer: Sections 9.2
5. Ham & Kostanic: Section 4.1, 4.2, 4.3