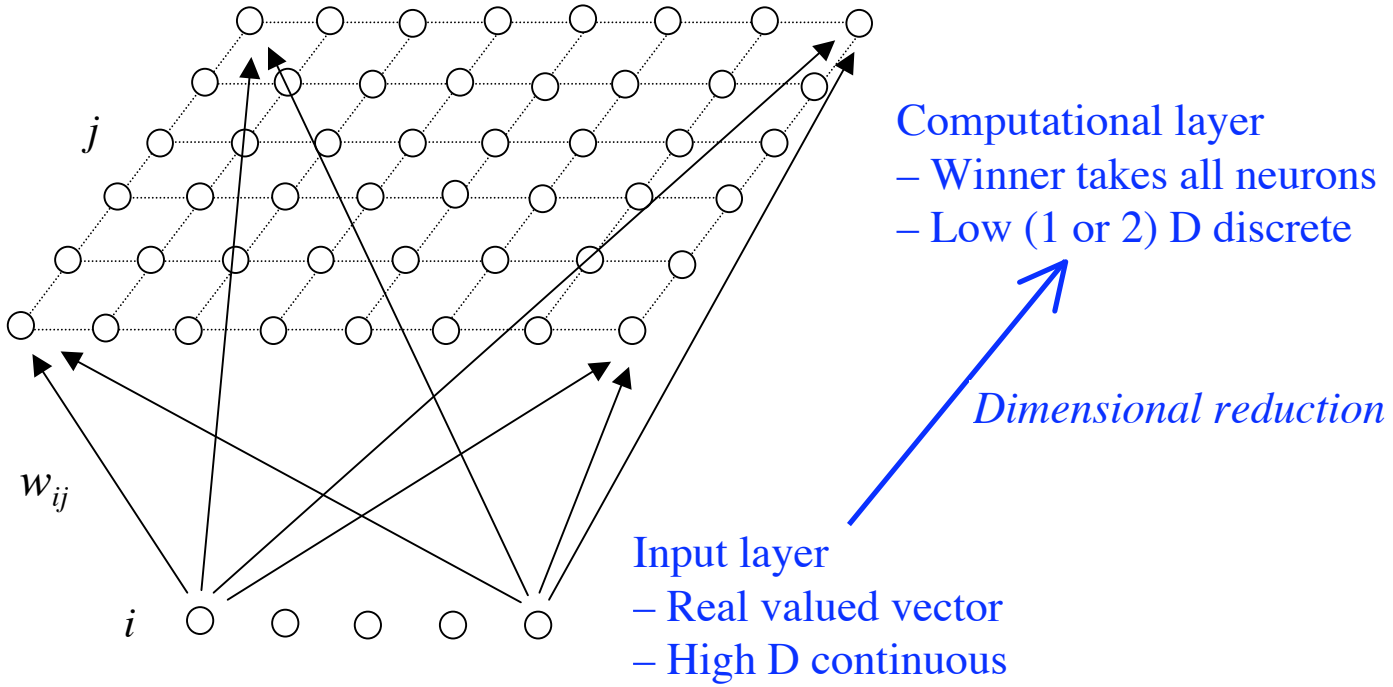
Learning Vector Quantization

Neural Computation : Lecture 18

1. SOM Architecture and Algorithm
2. Vector Quantization
3. The Encoder-Decoder Model
4. Generalized Lloyd Algorithms
5. Relation between SOMs and Noisy Encoder-Decoders
6. Voronoi Tessellation
7. Learning Vector Quantization (LVQ)

Architecture of a Self Organizing Map

We continue to study the properties of the SOM 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 the input patterns exist as vectors, to a low dimensional spatially ***discrete output space***, which is formed by arranging the computational neurons into a grid.

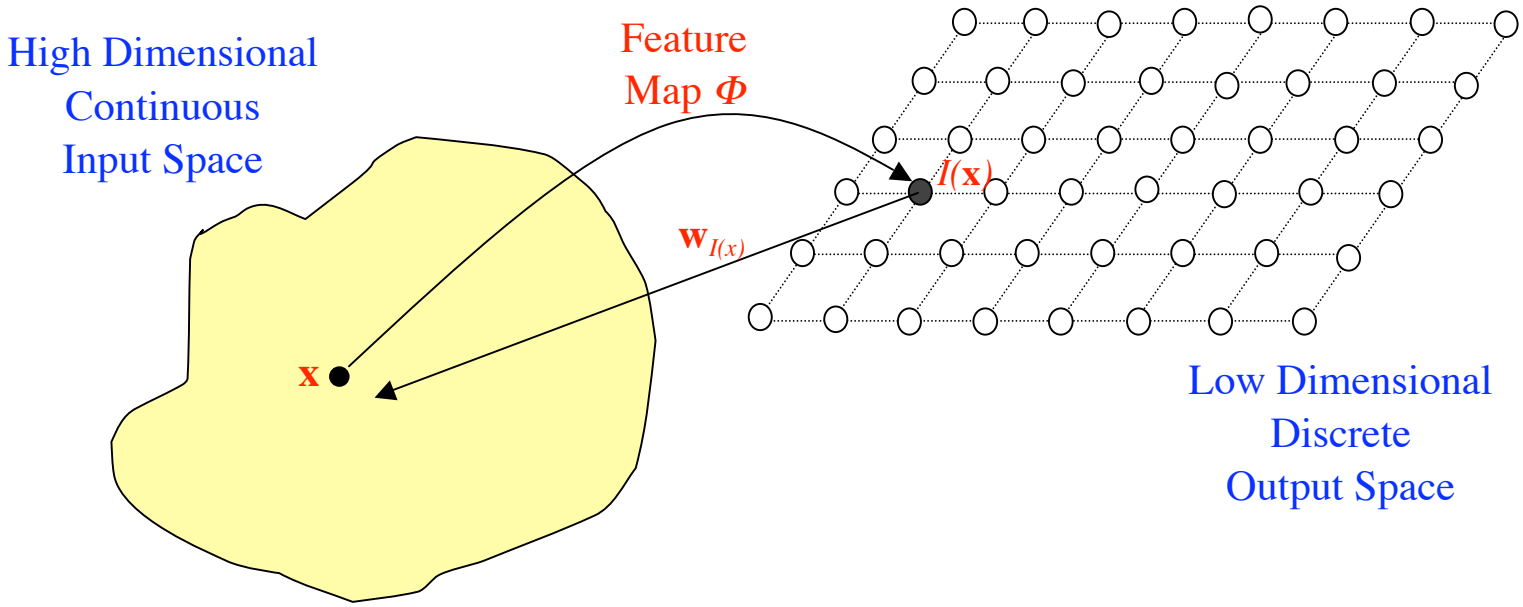
The stages of the SOM algorithm that achieves this can be summarised as follows:

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 representative 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 iterative process converges 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 single winning representative neuron $I(\mathbf{x})$ in the output space, and its 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.

Vector Quantization

It has already been 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 to facilitate ***dimensionality reduction*** or ***data compression***.

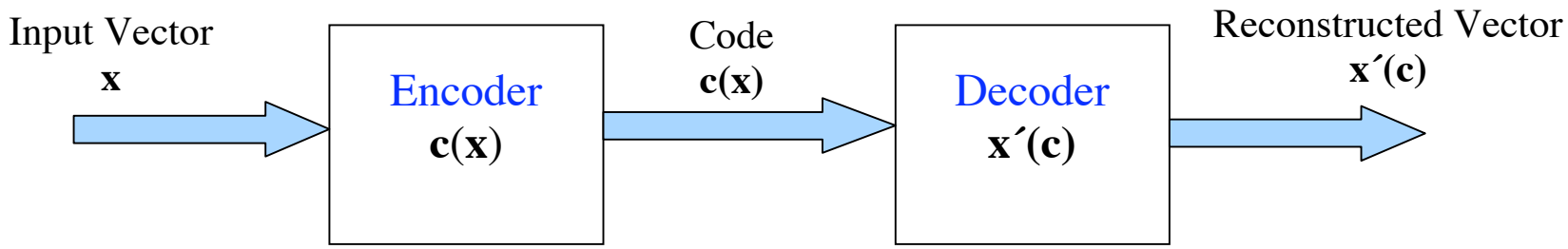
In effect, the error of the vector quantization approximation is the total squared distance

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

between the input vectors $\{\mathbf{x}\}$ and their representatives $\{\mathbf{w}_{I(\mathbf{x})}\}$, and we clearly wish to minimize this. It can be shown 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 $\mathbf{c}(\mathbf{x})$ acts as an encoder of the input vector $\mathbf{x}$, and $\mathbf{x}'(\mathbf{c})$ acts as a decoder of $\mathbf{c}(\mathbf{x})$, then we can attempt to get back to $\mathbf{x}$ with minimal loss of information:



Generally, the input vector $\mathbf{x}$ will be selected at random according to some probability density function $p(\mathbf{x})$. Then the optimum encoding-decoding scheme is determined by varying the functions $\mathbf{c}(\mathbf{x})$ and $\mathbf{x}'(\mathbf{c})$ 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 $\mathbf{x}$ and decoder $\mathbf{x}'(\mathbf{c})$, choose the code $\mathbf{c} = \mathbf{c}(\mathbf{x})$ to minimize the squared error distortion $\|\mathbf{x} - \mathbf{x}'(\mathbf{c})\|^2$.

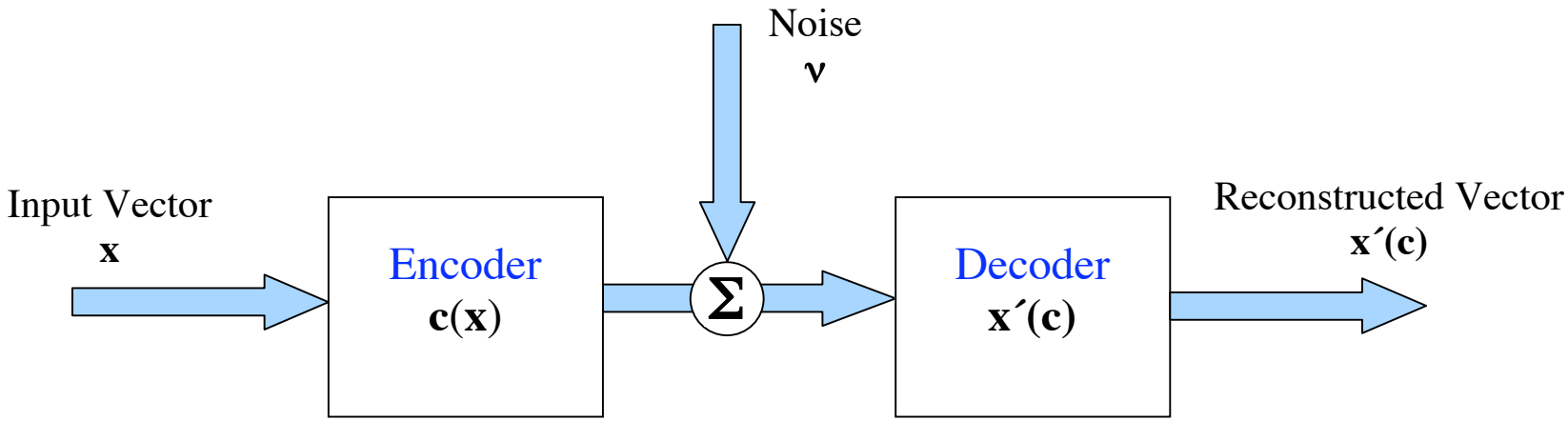
Condition 2. Given the code $\mathbf{c}$, compute the reconstruction vector $\mathbf{x}'(\mathbf{c})$ as the centroid of those input vectors $\mathbf{x}$ that satisfy Condition 1.

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. Note the clear correspondence with the two alternating phases of the K-Means Clustering Algorithm.

The Noisy Encoder – Decoder Model

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



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

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

Conditions for Minimal Expected Distortion D_v

The contribution to the total distortion D_v corresponding to a single data point $\mathbf{x}$ is

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

The partial derivative of the total distortion D_v with respect to the decoder $\mathbf{x}'(\mathbf{c})$ is

$$\frac{\partial D_v}{\partial \mathbf{x}'(\mathbf{c})} = -2 \sum_{\mathbf{x}} \pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))(\mathbf{x} - \mathbf{x}'(\mathbf{c})) = -2 \int d\mathbf{x} p(\mathbf{x}) \pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))(\mathbf{x} - \mathbf{x}'(\mathbf{c}))$$

and at the minimum of the distortion this partial derivative will be zero, i.e.

$$\int d\mathbf{x} p(\mathbf{x}) \pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))(\mathbf{x} - \mathbf{x}'(\mathbf{c})) = 0$$

which can be solved for $\mathbf{x}'(\mathbf{c})$ to give the decoder:

$$\mathbf{x}'(\mathbf{c}) = \int d\mathbf{x} p(\mathbf{x}) \pi(\mathbf{c} - \mathbf{c}(\mathbf{x})) \mathbf{x} \Big/ \int d\mathbf{x} p(\mathbf{x}) \pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))$$

Minimizing the Expected Distortion D_v

Generally, given the decoder $\mathbf{x}'(\mathbf{c})$, the optimal encoder $\mathbf{c}(\mathbf{x})$ for each data point $\mathbf{x}$ can be found straightforwardly using a simple nearest neighbour approach.

The easiest way to arrive at the optimal decoder $\mathbf{x}'(\mathbf{c})$ is usually to follow a standard gradient descent approach. The relevant partial derivative for a single data point $\mathbf{x}$ is

$$\frac{\partial D_v(\mathbf{x})}{\partial \mathbf{x}'(\mathbf{c})} = -2\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))(\mathbf{x} - \mathbf{x}'(\mathbf{c}))$$

so appropriate gradient descent updates of the decoder to minimize the distortion are

$$\Delta\mathbf{x}'(\mathbf{c}) = -\eta\frac{\partial D_v(\mathbf{x})}{\partial \mathbf{x}'(\mathbf{c})} \approx \eta\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))(\mathbf{x} - \mathbf{x}'(\mathbf{c}))$$

Since the optimal decoder depends on the encoder, and the optimal encoder depends on the decoder, an alternating series of updates will be required to end up with the minimum expected distortion D_v as in the K-Means Clustering Algorithm.

The Generalized Lloyd Algorithm with Noise

Thus, to minimize the expected distortion D_ν the encoder and decoder are iteratively updated to satisfy the conditions of the modified ***Generalized Lloyd Algorithm***:

Condition 1. Given the input vector $\mathbf{x}$ and decoder $\mathbf{x}'(\mathbf{c})$, 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 $\mathbf{x}'(\mathbf{c}) = \int d\mathbf{x}p(\mathbf{x})\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))\mathbf{x} \Big/ \int d\mathbf{x}p(\mathbf{x})\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))$.

At each iteration, Condition 1 is satisfied using a simple nearest neighbour approach for $\mathbf{c}(\mathbf{x})$, and then gradient descent updates $\Delta\mathbf{x}'(\mathbf{c}) = \eta\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))(\mathbf{x} - \mathbf{x}'(\mathbf{c}))$ are applied to the reconstruction vector $\mathbf{x}'(\mathbf{c})$ until Condition 2 is satisfied. Note that if the noise density function $\pi(\boldsymbol{\nu})$ is set to be the Dirac delta function $\delta(\boldsymbol{\nu})$, that is zero everywhere except at $\boldsymbol{\nu} = 0$, the conditions here reduce to those in the no noise case considered previously.

Relation between a SOM and Noisy Encoder–Decoder

It is now easy to see that a direct correspondence can be established between the SOM algorithm and the noisy encoder-decoder model:

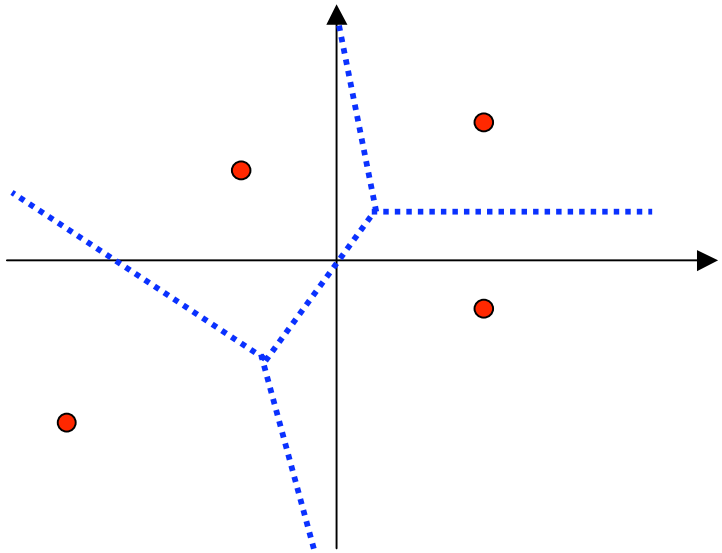
Noisy Encoder-Decoder Model	SOM Algorithm
Encoder $\mathbf{c}(\mathbf{x})$	Best matching neuron $I(\mathbf{x})$
Reconstruction vector $\mathbf{x}'(\mathbf{c})$	Connection weight vector $\mathbf{w}_j$
Probability density function $\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))$	Neighbourhood function $T_{j,I(\mathbf{x})}$

and that this relation provides a proof that the SOM algorithm is a vector quantization algorithm that generates an optimal approximation of the input space.

Note that the topological neighbourhood function $T_{j,I(\mathbf{x})}$ in the SOM algorithm maps to a noise probability density function, so a Gaussian form for it can be justified.

Voronoi Tessellation

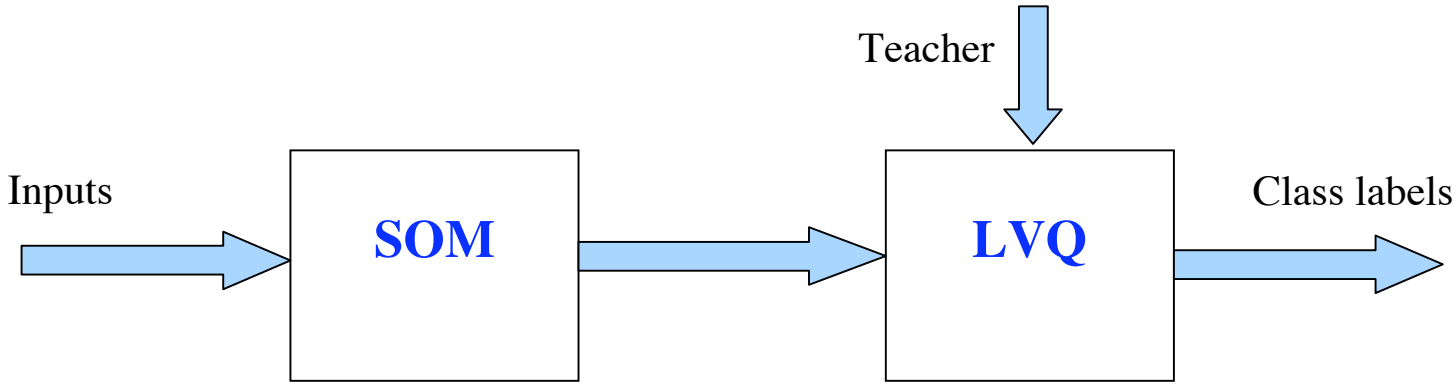
A vector quantizer with minimum encoding distortion is called a ***Voronoi quantizer*** or ***nearest-neighbour quantizer***. The input space is partitioned into a set of ***Voronoi or nearest neighbour cells*** each containing an associated ***Voronoi or reconstruction vector***:



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 a good set of basis function centres and widths in ***RBF networks***.

Learning Vector Quantization (LVQ)

Learning Vector Quantization (LVQ) is a supervised version of vector quantization that can be used when labelled input data is available. 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 winning representative weights/Voronoi vectors $\{\mathbf{w}_j\}$.

The additional factor is that the input data points have associated class information. This means the known classification labels of the inputs can be used to find the best classification label for each $\mathbf{w}_j$, i.e. for each Voronoi cell, by simply determining for each cell which class has the largest number of input instances within it.

It then follows that each new input without a class label can be classified simply by assigning it 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. The 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, one achieves better classification than from 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.

Numerous other variations on this theme also 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. Then we looked at general encoder-decoder models and noisy encoder-decoder models, and the Generalized Lloyd Algorithms 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-1999: Sections 9.5, 9.7, 9.8, 9.10, 9.11
2. Beale & Jackson: Section 5.6
3. Gurney: Section 8.3.6
4. Hertz, Krogh & Palmer: Section 9.2
5. Ham & Kostanic: Sections 4.1, 4.2, 4.3