Vector Quantization

— Lecture Notes —

Laurenz Wiskott
Institut für Neuroinformatik
Ruhr-Universität Bochum, Germany, EU

15 December 2016

— Summary —

Vector quantization can be applied to any vectorial data and is mostly used for compression. A more rare application is image posterization, for instance, where colors are discretized to a few, to get a poster effect. Compression is achieved by replacing the data vectors by a few representative vectors. If you have one thousand 100-dimensional vectors, for instance, then you can place ten reference vectors into high density regions of the vector space and represent each data vector by its closest reference vector. Instead of storing one-thousand 100-dimensional vectors, you then store only ten 100-dimensional vectors plus one thousand indices pointing to one of the reference vectors, which would be a compression from 1,000,000 floating point numbers down to 1,000 floating point numbers and 1,000 integers. The position of the reference vectors and the indices used for the data vectors are optimized to minimize the mean squared error between the data vectors and their corresponding reference vector.

1 Batch learning algorithms introduces the basic mathematical and algorithmic concepts of vector quantization under the assumption that all data is available at once: the error function, the optimal assignment of the data vectors to the reference vectors, the optimal placement of the reference vectors, and the simple but very popular k-means algorithm.

→ Video 1.1-1.5 (old 1-5), no 1.6, Exercises, Solutions

Core text and formulas are set in dark red, one can repeat the lecture notes quickly by just reading these; ♦ marks important formulas or items worth remembering and learning for an exam; ◊ marks less important formulas or items that I would usually also present in a lecture; + marks sections that I would usually skip in a lecture.

More teaching material is available at https://www.ini.rub.de/PEOPLE/wiskott/Teaching/Material/.

2 Online learning algorithms generalizes the concepts of Section 1 to the online situation, i.e. the case where the data vectors come in one by one. New concepts here are a scheduled learning rate and moving groups of reference vectors around simultaneously.

→ Videos 2.1+2.2 (old 6+7), 2.3+2.4 (old 8-10), Exercises, Solutions

Contents

These lecture notes are largely based on (Hertz et al., 1991; Rojas, 1996).

1 Batch learning algorithms

1.1 General idea

The purpose of vector quantization is to compress vectorial data. The general idea is to **find a good set of reference vectors and then replace each data vector simply by the index of its best reference vector.** If there are much fewer reference vectors than data vectors, it is much cheaper to store or transmit just the reference vectors and the index for each data vector than the data vectors themselves. Finding the optimal vector quantization has thus two subproblems: (i) finding an optimal set of reference vectors and (ii) finding the optimal index for each data vector.

1.2 Error function

The **performance** of the vector quantization **is typically measured by** the quantization error, which is **the mean Euclidean distance between the data vectors and their reference vectors.** Let $\boldsymbol{x}^\mu$ indicate the data vectors, with $\mu = 1, ..., M$, and $\boldsymbol{r}_i$ indicate the reference vectors, with $i = 1, ..., I$. The index associated with each data vector is $i^* = i^*(\mu)$, where we refer to $i^*(\mu)$ as the assignment function. The quantization error is then given by

$$\blacklozenge \quad E := \left\langle \|\boldsymbol{x}_\mu - \boldsymbol{r}_{i^*}\|^2 \right\rangle_\mu , \tag{1.1}$$

where $\langle\cdot\rangle$ indicates an averaging. Note that i^* is a function of μ.

1.3 Optimal assignment given the reference vectors

Assume the data vectors and reference vectors are given. What is the optimal assignment of the data vectors to the reference vectors, i.e. what is the otimal assignment function $i^*(\mu)$? It is pretty obvious and follows directly from equation (1.1) that **each data vector should be assigned to its nearest reference vector**, i.e.

$$\lozenge \quad i^*(\mu) := \arg\min_i \|\boldsymbol{x}_\mu - \boldsymbol{r}_i\|^2 \tag{1.2}$$

is optimal.

1.3.1 Voronoi tesselation

Instead of deciding for each data point to which reference vector it belongs, **one can also partition the space such that all points within one partition get assigned to one reference vector**. The optimal reference vector then becomes a function of space and not of data vector, but otherwise it is the same, i.e.

$$i^*(\boldsymbol{x}) := \arg\min_i \|\boldsymbol{x} - \boldsymbol{r}_i\|^2 \tag{1.3}$$

is optimal. **Such a partitioning is called *Voronoi* (or *Dirichlet*) *tesselation*** (D: Voronoi-Diagramm, Dirichlet-Zerlegung) **and the region closest to one reference vector is called *Voronoi cell*** (D: Voronoi Region)**.**

1.3.2 Delaunay triangulation

If one connects all pairs of reference vectors whose Voronoi regions share a common edge, one gets a *Delaunay triangulation* (D: Delaunay-Triangulation)**.**

A definition independent of Voronoi regions reads as follows: "For a set P of points in the (d-dimensional) Euclidean space, a Delaunay triangulation is a triangulation $DT(P)$ such that no point in P is inside the circum-hypersphere of any simplex in $DT(P)$. It is known that there exists a unique Delaunay triangulation for P, if P is a set of points in general position; that is, there exists no k-flat containing $k+2$ points nor a k-sphere containing $k+3$ points, for $1 \leq k \leq d-1$ (e.g., for a set of points in $\mathbb{R}^3$; no three points are on a line, no four on a plane, no four are on a circle, and no five on a sphere)." (Wikipedia, 2013)

1.4 Optimal reference vectors given an assignment

In general it is a hard problem to determine **the optimal reference vectors**. However**, if a fixed assignment is given,** it is easy to determine them. They simply **lie in the center of gravity of the data vectors assigned to them** as can be seen as follows. Consider just one reference i and let $\mathcal{M}_i$ be the set of data vectors assigned to it. The quantization error then simplifies to

$$\lozenge \quad E_i := \left\langle \|\boldsymbol{x}_\mu - \boldsymbol{r}_i\|^2 \right\rangle_{\mu \in \mathcal{M}_i} . \tag{1.4}$$

A condition for the optimal choice of the reference $\boldsymbol{r}_i$ is that the gradient of E_i with respect to $\boldsymbol{r}_i$ vanishes.

$$\lozenge \quad 0 \stackrel{!}{=} \operatorname{grad} E_i \stackrel{(1.4)}{=} -2\langle \boldsymbol{x}_\mu - \boldsymbol{r}_i \rangle_{\mu \in \mathcal{M}_i} \tag{1.5}$$

$$\lozenge \quad = \quad 2\left(\boldsymbol{r}_i - \langle \boldsymbol{x}_\mu \rangle_{\mu \in \mathcal{M}_i}\right) \tag{1.6}$$

$$\lozenge \quad \Longleftrightarrow \quad \boldsymbol{r}_i \quad = \quad \langle \boldsymbol{x}_\mu \rangle_{\mu \in \mathcal{M}_i} . \tag{1.7}$$

Since E_i is a quadratic error function, this is indeed the optimum.

1.5 A simple vector quantization algorithm

Now that we have a criterion for the optimal assignment given the reference vectors and for the optimal reference vectors given a fixed assignment, we can formulate **a simple iterative algorithm for vector quantization**. This algorithm is known as the k-means, LBG or generalized Lloyd algorithm.

1. **Decide on the number k of reference vectors** based the compression rate you want to achieve.
2. **Initialize the k reference vectors** to different data vectors.
3. **Find the assingment** of the data vectors to their nearest reference vectors.
4. **Move each reference vector to the mean or center of gravity** of the data assigned to it, see (1.7).
5. **Repeat** at step 3 **until convergence**.

This algorithm is simple and converges quickly. **Unfortunately, it** is not guaranteed to converge to the optimal solution but **gets easily stuck in a local optimum.** To mitigate this problem, you can start the algorithm multiple times with different initializations and just keep the best result.

1.6 Growing networks +

Bernd Fritzke has developed a growing neural gas algorithm, in which additional reference vectors are added at locations of large errors, so that the error can be reduced in a very targeted way (Fritzke, 1994).

2 Online learning algorithms

2.1 An online learning rule for vector quantization

If it is not possible to process all data simultaneously, e.g. if there is too much data, or if one prefers to process data one by one for greater biological plausibility, **one can use an online version of the algorithm in the previous section. Instead of moving a given reference vector directly to the center of gravity** of the data vector within its Voronoi region, **it gets moved by a certain fraction towards a data vector** whenever it is assigned to the reference vector. This procedure is approximately equivalent in the long run, if the fraction, determined by the so-called *learning rate* η, is small enough so that the reference vectors do not jump about too much. The algorithm now reads as follows:

1. **Initialize the reference vectors** to different data vectors.
2. **Pick a data vector $\boldsymbol{x}_\mu$ at random**.
3. **Determine the optimal reference vector $\boldsymbol{r}_{i^*}$** with (1.2)
$$i^*(\mu) := \arg\min_i \|\boldsymbol{x}_\mu - \boldsymbol{r}_i\|^2 \tag{2.1}$$
4. **Move** the optimal reference vector **a little bit towards the data vector:**
$$\Delta \boldsymbol{r}_{i^*} = \eta(\boldsymbol{x}_\mu - \boldsymbol{r}_{i^*}) \tag{2.2}$$
5. **Repeat** at 2 **until** a maximum number of steps is reached or any other measure indicates **convergence**.

2.2 Learning rates

With a constant learning rate in the online learning rule the learning is typically too slow in the initial phase and at the end the algorithm does not converge, because the reference vectors get changed indefinitely. Thus, **it might be good to consider decreasing the learning rate over time**. However, first consider the effect of a constant learning rate.

2.2.1 Constant learning rate

Consider just one reference vector $\boldsymbol{r}$ and let $\boldsymbol{x}(t)$ be a sequence of data vectors used to update $\boldsymbol{r}$. **With a constant learning rate** we have

$$\begin{aligned}
\lozenge \quad \boldsymbol{r}(0) &= \boldsymbol{r}_0\,, && (2.3)\\
\lozenge \quad \boldsymbol{r}(t) &\overset{(2.2)}{=} \boldsymbol{r}(t-1) + \eta(\boldsymbol{x}(t) - \boldsymbol{r}(t-1)) && (2.4)\\
\lozenge &= (1-\eta)\boldsymbol{r}(t-1) + \eta\boldsymbol{x}(t)\,. && (2.5)
\end{aligned}$$

By mathematical induction **we find that** for $t > 0$

$$\lozenge \quad \boldsymbol{r}(t) = (1-\eta)^t \boldsymbol{r}_0 + \eta \sum_{t'=1}^{t} (1-\eta)^{t-t'} \boldsymbol{x}(t')\,. \tag{2.6}$$

Proof:

$$\text{Base } (t=1): \quad \boldsymbol{r}(1) \overset{(2.5,2.3)}{=} (1-\eta)\boldsymbol{r}_0 + \eta\boldsymbol{x}(1) \tag{2.7}$$

$$= (1-\eta)^1\boldsymbol{r}_0 + \eta\sum_{t'=1}^{1}(1-\eta)^{1-t'}\boldsymbol{x}(t') \tag{2.8}$$

$$= \text{rhs of (2.6) for } t=1. \tag{2.9}$$

$$\tag{2.10}$$

$$\text{Induction step } (t>1): \quad \boldsymbol{r}(t) \overset{(2.5)}{=} (1-\eta)\boldsymbol{r}(t-1) + \eta\boldsymbol{x}(t) \tag{2.11}$$

$$\overset{(2.6)}{=} (1-\eta)\left((1-\eta)^{(t-1)}\boldsymbol{r}_0 + \eta\sum_{t'=1}^{t-1}(1-\eta)^{t-1-t'}\boldsymbol{x}(t')\right) + \eta\boldsymbol{x}(t) \tag{2.12}$$

$$= (1-\eta)^t\boldsymbol{r}_0 + \eta\sum_{t'=1}^{t-1}(1-\eta)^{t-t'}\boldsymbol{x}(t') + \eta(1-\eta)^{t-t}\boldsymbol{x}(t) \tag{2.13}$$

$$= (1-\eta)^t\boldsymbol{r}_0 + \eta\sum_{t'=1}^{t}(1-\eta)^{t-t'}\boldsymbol{x}(t') \tag{2.14}$$

$$= \text{rhs of (2.6) for } t>1. \tag{2.15}$$

Thus, **the influence of data vectors** picked for training **decays exponentionally into the past.** As a consequence **the system never converges** but it remains adaptive in case the data distribution changes. If the data distribution does not change, the reference vectors first move towards an equilibrium distribution and then wiggle around forever.

2.2.2 $1/t$-decaying learning rate

To avoid the problems connected with a constant learning rate one can choose a decaying learning rate. One interesting scheme is to introduce a learning rate **for each reference vector individually** and let it decay **like a harmonic sequence**

$$\blacklozenge \quad \eta(t) = \frac{1}{t}\,, \tag{2.16}$$

where t again indexes the training event for the one particular reference vector considered (one could also write $\eta_i(t_i)$ so that it is more explicit that each reference vector has its own learning rate and its own counter for how often it has been modified).

With such a decaying learning rate **we** can **find** by mathematical induction that

$$\blacklozenge \quad \boldsymbol{r}(t) = \frac{1}{t}\sum_{t'=1}^{t}\boldsymbol{x}(t')\,. \tag{2.17}$$

Thus, with such a learning rate **the reference vectors lie exactly in the center of gravity of all data vectors used to train it. However, some of the data vectors used at some time might not lie in the Voronoi region of the reference vector anymore,** because the reference vectors have moved around. Also, if the data vectors are picked at random, **some might have been picked multiple times while others have not been picked at all.** Thus, online learning is not equivalent to the batch algorithm (LBG) considered above. Finally, **this online learning does not even converge in a strict sense because the harmonic series diverges** (although this is more of a theoretical issue, because the harmonic series diverges extremely slowly, see http://www.mathematik.com/Harmonic/index.html).

2.2.3 Exponentially decaying learning rate

Another proposal for a learning rate **is**

$$\blacklozenge \quad \eta(t) = \eta_{\text{init}}(\eta_{\text{final}}/\eta_{\text{init}})^{t/t_{\text{final}}}\,, \tag{2.18}$$

which is exponentially decaying. η_{init} indicates the initial learning rate, η_{final} the final learning rate, and t_{final} the total number of steps used. Note that now there is just one **common** learning rate **for all reference vectors.**

Depending on the parameters the learning rate **tends to be larger in the beginning and smaller at the end compared to the $1/t$-decay.** This has several advantages. Firstly, **learning is more agressive in the beginning** (reference vectors jump around more), which can be interpreted as additional noise and **helps** the system **to escape a local optimum** it might have been initialized to. Secondly, **data vectors used early on loose their influence rapidly** so that in the end only data vectors within the final Voronoi region have significant influence on the position of the reference vector. Finally, since the learning rate is smaller at the end, the **learning has better convergence.** It has also been show experimentally that the exponentially decaying learning rate usually leads to better results than $1/t$ (except if there is only one reference vector, in which case $1/t$ is optimal).

2.3 Soft competitive learning

So far only the nearest reference vector got updated, which is referred to as hard competitive learning. **Some algorithms not only update the nearest reference vector but also the second and even more neighbors,** but to a lesser degree the farther away the reference vectors are from the data vector, although it is actually not the distance that matters but the rank in the list of neighbors. This can be formalized as follows.

1. **Initialize the reference vectors** to different data vectors.
2. **Pick a data vector $\boldsymbol{x}_\mu$ at random**.
3. **Determine the nearest reference vector, the second nearest, etc.** indicated by ${i^*}_1, {i^*}_2, \ldots$ etc.
4. **Move** the reference vectors **a little bit towards the data vector depending on the rank in the** nearest reference vector **list**
$$\lozenge \quad \Delta \boldsymbol{r}_{{i^*}_k} = \eta \exp(-k/\lambda)(\boldsymbol{x}_\mu - \boldsymbol{r}_{{i^*}_k}) \tag{2.19}$$
5. **Repeat** at 2 **until** a maximum number of steps is reached or any other measure indicates **convergence**.

The factor $\exp(-k/\lambda)$ weights the adaptation according to the rank k of the reference vectors. This soft competitive learning can be combined with an exponential decay of the learning rate as well as an exponential decay of the scale λ of the neighborhood function $\exp(-k/\lambda)$.

The advantage of soft competitive learning, I guess, is that **local optima are avoided to some extent and** that **learning can be faster**, since in the beginning whole groups of reference vectors are updated.

2.4 Towards a neural implementation +

So far we have considered vector quantization only as an algorithmic problem. If we want to use vector quantization for computational neuroscience we have to consider also how such an algorithm could be implemented in a neural system.

2.4.1 Euclidian distance and inner product

In the algorithmic version the Euclidian norm is used as a measure of distance between data and reference vectors. In neural systems a more natural choice is the inner product between the input vector and the weight vector. These two measures are quite different, however, if data/input vectors as well as reference/weight vectors are normalized to length 1, Euclidian distance and inner product are closely related.

$$\begin{aligned} |\boldsymbol{x}-\boldsymbol{r}|^2 &= |\boldsymbol{x}|^2 - 2\boldsymbol{x}\cdot\boldsymbol{r} + |\boldsymbol{r}|^2 && (2.20)\\ &= 2(1-\boldsymbol{x}\cdot\boldsymbol{r})\,. && (2.21)\\ &\qquad (\text{since } |\boldsymbol{x}| = |\boldsymbol{r}| = 1 \text{ by assumption}) \end{aligned}$$

This also implies

$$\begin{aligned} & \boldsymbol{x}\cdot\boldsymbol{r}_i > \boldsymbol{x}\cdot\boldsymbol{r}_j && (2.22)\\ \iff\quad & |\boldsymbol{x}-\boldsymbol{r}_i| < |\boldsymbol{x}-\boldsymbol{r}_j|\,. && (2.23) \end{aligned}$$

Thus, if input and weight vectors are normalized to one, the most excited unit is always the one with the least Euclidean distance of its weights vector to the input vector.

Even though there is a strong relationship between Euclidian distance and inner product if input and weight vectors are normalized to one, the normalization forces the data to lie on the surface of a hypersphere, which is a special topology.

2.4.2 Winner takes all

The vector quantization algorithm requires to determine the nearest reference vector. In a neural system this can be achieved by a winner-take-all mechanism, which can be implemented by mutual inhibition and self-excitation.

There exist also k-winner-take-all versions of this, where the first few winners get activated. It is difficult, though, to implement a decaying activation of the first winners. Soft-max might be an option here.

2.4.3 Adaptive resonance theory

Adaptive resonance theory (ART) is a detailed model of vector quantization, which employs new units as qualitatively new input data arrives. It is therefore similar to the growing neural gas algorithm.

3 Demo

See http://www.demogng.de/js/demogng.html.

References

Fritzke, B. (1994). Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA.

Rojas, R. (1996). *Theorie der neuronalen Netze: Eine systematische Einführung*. Springer Verlag, Berlin. 4., korregierter Nachdruck.

Wikipedia (2013). Delaunay triangulation — wikipedia, the free encyclopedia. [Online; accessed 21-February-2013].