Topological Neural Discrete Representation Learning à la Kohonen

Kazuki Irie[1*] **Róbert Csordás**[1*] **Jürgen Schmidhuber**[1,2]
[1]The Swiss AI Lab, IDSIA, USI & SUPSI, Lugano, Switzerland
[2]AI Initiative, KAUST, Thuwal, Saudi Arabia
{kazuki, robert, juergen}@idsia.ch

Abstract

Unsupervised learning of discrete representations in neural networks (NNs) from continuous ones is essential for many modern applications. Vector Quantisation (VQ) has become popular for this, in particular in the context of generative models such as Variational Auto-Encoders (VAEs), where the exponential moving average-based VQ (EMA-VQ) algorithm is often used. Here we study an alternative VQ algorithm based on Kohonen's learning rule for the Self-Organising Map (KSOM; 1982), a classic VQ algorithm known to offer two potential benefits over its special case EMA-VQ: empirically, KSOM converges faster than EMA-VQ, and KSOM-generated discrete representations form a topological structure on the grid whose nodes are the discrete symbols, resulting in an artificial version of the brain's *topographic map*. We revisit these properties by using KSOM in VQ-VAEs for image processing. In our experiments, the speed-up compared to *well-configured* EMA-VQ is only observable at the beginning of training, but KSOM is generally much more robust, e.g., w.r.t. the choice of initialisation schemes.[1]

1 Introduction

Internal representations in artificial neural networks (NNs) are continuous-valued vectors. As such, they are rarely identical in different contexts. In many scenarios, however, it is natural and desirable to treat some of these non-identical but similar vectors as representing a common discrete symbol from a fixed-size codebook/lexicon shared across various contexts. Such *discrete representations* would allow us to express and manipulate hidden representations of NNs using a set of symbols. Unsupervised learning of such discrete representations is motivated in various contexts. For example, in certain algorithmic or reasoning tasks (e.g., Liska et al. [29], Hupkes et al. [19], Csordás et al. [10]), learning of such representations may be a key for generalisation, since intermediate results in such tasks are inherently discrete.

Recently, discrete representation learning via *Vector Quantisation* (VQ) has become popular for practical reasons too. The Vector Quantised-Variational Auto-Encoders (VQ-VAEs) by van den Oord et al. [49] use VQ as a pre-processing step to *tokenise* high-dimensional data, such as images, i.e., to represent an image as a sequence of discrete symbols. The resulting sequence can then be processed by a powerful sequence processor, e.g., a Transformer variant [50, 45, 44]. Similar techniques have been applied to other kinds of data such as video [53, 56] and audio [3, 13, 47, 5]). Today, many large-scale text-to-image systems such as *DALL-E* [40], *Parti* [59], or *Latent Diffusion Models* [42], also use VQ in one way or another.

Here we study the learning rules of Kohonen Self-Organising Maps or Kohonen Maps [24, 26] as the VQ algorithm for discrete representation learning in NNs. For shorthand, we refer to the corresponding

[*]Equal contribution.
[1]Our code is public: https://github.com/IDSIA/kohonen-vae

ICML 2023 Workshop: Sampling and Optimization in Discrete Space (SODS 2023).

algorithm as *KSOM* (reviewed in Appendix A). In fact, KSOM is a classic VQ algorithm [37], and the exponential moving average-based VQ (EMA-VQ; van den Oord et al. [49], Razavi et al. [41], Kaiser et al. [21], Roy et al. [43]) commonly used today is a special case thereof (Sec. 2). KSOM is known to offer two potential benefits over EMA-VQ. First, KSOM is empirically reported to perform faster VQ (see, e.g., De Bodt et al. [11]). Second, discrete representations learned by KSOM form a *topological* structure in the pre-specified *grid* whose nodes represent the discrete symbols from the codebook. Such a grid is typically one- or two-dimensional, and symbols that are spatially close to each other on the grid represent features that are close to each other in the original input space. This *topological mapping* property has helped practitioners in certain applications to visualise/interpret their data (see, e.g. Tirunagari et al. [46]). While such a property is arguably of limited importance in today's deep learning, Kohonen's algorithm is specifically designed to achieve this property which is known in the brain as *topographic organisation*. KSOM allows to naturally achieve an artificial version thereof, as a by-product of the VQ algorithm.

We explore these properties in modern NNs by using KSOM as the VQ algorithm in VQ-VAEs [49, 41] for image processing. Importantly, we also revisit the configuration details of the baseline EMA-VQ (e.g., initialisation of EMAs). We show that proper configurations are crucial for EMA-VQ to optimally perform, while KSOM is robust, and performs well in all cases.

Appendix A provides all background materials on KSOM needed to describe our approach (Sec. 2).

2 Alternative VQ in VQ-VAEs

The general idea of this work is to replace the VQ algorithm used in standard VQ-VAEs by Kohonen's algorithm (reviewed in Appendix A). While the method can be applied to various data modalities, we focus on image processing as a representative example.

2.1 Background: VQ-VAEs

Let $d_{\text{in}}, d_{\text{emb}}, N, K$ be positive integers. A VQ-VAE [49] consists of an encoder NN, $\texttt{Enc} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{N \times d_{\text{emb}}}$, a decoder NN, $\texttt{Dec} : \mathbb{R}^{N \times d_{\text{emb}}} \to \mathbb{R}^{d_{\text{in}}}$, and a codebook of size K whose weights are $(\boldsymbol{w}_1, ..., \boldsymbol{w}_K)$ with $\boldsymbol{w}_k \in \mathbb{R}^{d_{\text{emb}}}$ for $k \in \{1, ..., K\}$. The encoder transforms an input $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ to a sequence of N embedding vectors $(\boldsymbol{e}_1, ..., \boldsymbol{e}_N) = \boldsymbol{E}$ with $\boldsymbol{e}_i \in \mathbb{R}^{d_{\text{emb}}}$ for $i \in \{1, ..., N\}$. Each of these embeddings is quantised to yield $(\texttt{VQ}(\boldsymbol{e}_1), ..., \texttt{VQ}(\boldsymbol{e}_N)) = \boldsymbol{E}'$ with $\texttt{VQ}(\boldsymbol{e}_i) \in \mathbb{R}^{d_{\text{emb}}}$ for $i \in \{1, ..., N\}$ where $\texttt{VQ}$ denotes the VQ operation. The decoder transforms the quantised embeddings $\boldsymbol{E}'$ to a reconstruction of the original input $\hat{\boldsymbol{x}} \in \mathbb{R}^{d_{\text{in}}}$. The corresponding operations can be expressed as follows.

$$(\boldsymbol{e}_1, ..., \boldsymbol{e}_N) = \texttt{Enc}(\boldsymbol{x}) \tag{1}$$

$$\text{For all } i \in \{1, ..., N\},\ k_i^* = \underset{1 \le k \le K}{\arg\min} \|\boldsymbol{e}_i - \boldsymbol{w}_k\|_2 \tag{2}$$

$$\boldsymbol{E}' = (\texttt{VQ}(\boldsymbol{e}_1), ..., \texttt{VQ}(\boldsymbol{e}_N)) = (\boldsymbol{w}_{k_1^*}, ..., \boldsymbol{w}_{k_N^*}) \tag{3}$$

$$\hat{\boldsymbol{x}} = \texttt{Dec}(\boldsymbol{E}') \tag{4}$$

The parameters of the encoder and decoder are trained to minimise the following loss:

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2 + \lambda \frac{1}{N} \sum_{i=1}^{N} \|\texttt{sg}(\boldsymbol{w}_{k_i^*}) - \boldsymbol{e}_i\|_2^2 \tag{5}$$

where the first term is the reconstruction loss, and the second term is the so-called *commitment loss*, weighted by a hyper-parameter $\lambda \in \mathbb{R}_{>0}$, which encourages the encoder to output embeddings that are close to their quantised counterparts ($\texttt{sg}$ denotes "stop gradient" operation to prevent gradients to propagate through $\boldsymbol{w}_{k_i^*}$). As noted in van den Oord et al. [49], by assuming a uniform prior over the discrete latents, the standard KL term of the VAE loss [23] can be omitted as a constant. In the reconstruction term, as the quantisation operation is non-differentiable, the straight-through estimator [18, 4] is used, i.e., gradients are directly copied from the decoder input to the encoder output.

The weights of the codebook prototypes $(\boldsymbol{w}_1, ..., \boldsymbol{w}_K)$ are trained by a variant of online mini-batch K-means algorithm which keeps track of exponential moving averages (EMAs) of two quantities for each cluster k: the sum of updates $\boldsymbol{m}_k^{(t)}$, and the count of members in the cluster $N_k^{(t)}$. Their quotient

yields the estimate of the weights at step t. Using the same notation $\mathcal{C}_k^{(t)}$ as in Appendix A.2 for the set of encoder output indices (ranging from 1 to $N * B$ where B is a positive integer denoting the batch size) that are members of cluster k at step t, and by denoting encoder outputs in the batch as $(\boldsymbol{e}_1, ..., \boldsymbol{e}_{N*B})$, it yields:

$$\boldsymbol{m}_k^{(t)} = (1-\beta)\boldsymbol{m}_k^{(t-1)} + \beta \sum_{i \in \mathcal{C}_k^{(t)}} \boldsymbol{e}_i^{(t)} \tag{6}$$

$$N_k^{(t)} = (1-\beta)N_k^{(t)} + \beta|\mathcal{C}_k^{(t)}| \tag{7}$$

$$\boldsymbol{w}_k^{(t)} = \boldsymbol{m}_k^{(t)}/N_k^{(t)} \tag{8}$$

where $(1-\beta)$ is the EMA decay typically set to 0.99 (i.e., $\beta = 0.01$). For shorthand, we refer to this algorithm as EMA-VQ.

While it is also possible to train these codebook parameters through gradient descent by introducing an extra term in the loss of Eq. 5 (batching is omitted for consistency): $1/N \sum_{i=1}^{N} \|\boldsymbol{w}_{k_i^*} - \texttt{sg}(\boldsymbol{e}_i)\|_2^2$, van den Oord et al. [49] recommend the EMA-based approach above, and many later works [41, 39] follow this practice, while in VQ-GANs [14, 58], the gradient-based approach is also commonly used.

2.2 Kohonen-VAEs

We use KSOM (Appendix A) as the VQ algorithm to learn the codebook weights in VQ-VAEs (Sec. 2.1). Essentially, we replace the EMA-VQ algorithm of Eqs. 6-8 by an *online mini-batch* version of KSOM. Such an algorithm can be obtained by introducing exponential moving averaging into the batch version of KSOM (Eqs. 19-21) with a decay of $1-\beta$. That is, we keep track of EMAs of both the weighted sum of the updates ($\boldsymbol{m}_k^{(t)}$; Eq. 19) and the weighted count of members ($N_k^{(t)}$; Eq. 20) for each cluster $k \in \{1, ..., K\}$ (where weights are the neighbourhood coefficients). Their quotient yields the estimate of the weights of codebook prototypes at step t. Using the same notations as in Sec. 2.1, i.e., $\mathcal{C}_k^{(t)}$ denotes the set of indices of encoder outputs that are members of cluster k at step t, and $(\boldsymbol{e}_1, ..., \boldsymbol{e}_{N*B})$ denotes the encoder outputs in the batch, it yields:

$$\boldsymbol{m}_k^{(t)} = (1-\beta)\boldsymbol{m}_k^{(t-1)} + \beta \sum_{j=1}^{K} \boldsymbol{A}_{j,k}^{(t-1)} \sum_{i \in \mathcal{C}_j^{(t)}} \boldsymbol{e}_i^{(t)} \tag{9}$$

$$N_k^{(t)} = (1-\beta)N_k^{(t)} + \beta \sum_{j=1}^{K} \boldsymbol{A}_{j,k}^{(t-1)} |\mathcal{C}_j^{(t)}| \tag{10}$$

$$\boldsymbol{w}_k^{(t)} = \boldsymbol{m}_k^{(t)}/N_k^{(t)} \tag{11}$$

All other aspects are kept the same as in the basic VQ-VAE (Sec. 2.1). We refer to this approach as *Kohonen-VAE*.

Remark 2.1 (Relation to EMA-VQ). If the neighbourhood is reduced to zero (see, Remark A.1), this approach falls back to the standard EMA-VQ VAEs (Sec. 2.1).

2.3 Initialisation & Updates of EMAs

Initialisation. Both the basic EMA-VQ (Sec. 2.1) and KSOM (Sec. 2.2) require to initialise two EMAs: $\boldsymbol{m}_k^{(0)}$ (Eq. 6 and 9) and $N_k^{(0)}$ (Eq. 7 and 10). This is an important detail which is omitted in the common description of VQ-VAEs. Standard public implementations of VQ-VAEs, including the official one by van den Oord et al. [49], initialise $\boldsymbol{m}_k^{(0)}$ by a random Gaussian vector, and $N_k^{(0)}$ by 0. The latter is problematic for the following reason.

In fact, standard implementations apply the updates of Eqs. 6-8 to all clusters including those that have no members in the batch—later, we show that this is another important detail for EMA-VQ. Since $N_k^{(0)}$ is initialised with 0, $N_k^{(1)}$ remains 0 at the first iteration for the clusters with no members in the first batch. To avoid division by zero, smoothing over counts $(N_1^{(t)}, ..., N_K^{(t)})$—typically additive smoothing; see, e.g., Chen & Goodman [6]—is applied to obtain smoothed counts $(\tilde{N}_1^{(t)}, ..., \tilde{N}_K^{(t)})$

such that $N_k^{(1)} = 0$ becomes $\tilde{N}_k^{(1)} = \epsilon$ where typically $\epsilon \sim 10^{-5}$. While this allows to avoid division by zero, the resulting multiplication of $\boldsymbol{m}_k^{(1)}$ by $1/\epsilon \sim 10^5$ in Eq. 8 also seems unreasonable. Our experiments (Sec. 3.1) show that this effectively results in certain sub-optimality in training. Instead, we propose $N_k^{(0)} = 1$ initialisation, which is consistent with non-zero initialisation of $\boldsymbol{m}_k^{(0)}$.

Updating EMAs of Clusters without Members. Given the update equations above (Eqs. 6-8 and 9-11), there remains the question whether the EMAs of clusters that have no members in the batch should be updated. As mentioned already in the previous paragraph, standard implementations *update all* EMAs. We conduct the corresponding ablation study in Sec. 3.1, and empirically show that, indeed, it is crucial to update all EMAs in the baseline EMA-VQ algorithm used in VQ-VAEs to achieve optimal performance.

In the next section, we'll also show that, unlike the standard EMA-VQ which is sensitive to these configurations, KSOM is robust, and performs well under any configurations.

3 Experiments

The goal of our experiments is to revisit the properties of KSOM when integrated into VQ-VAEs. In particular, we demonstrate its robustness, and analyse learned representations. Before that, we start with showing the sensitivity of the standard EMA-VQ w.r.t. various configuration details.

3.1 Sensitivity of the baseline EMA-VQ

We first present two sets of experiments for the baseline EMA-VQ, which reveal its sensitivity to initialisation and update schemes of EMAs, as discussed in Sec. 2.3.

Improving Initialisation. We start with evaluating $N_k^{(0)} = 1$ initialisation (instead of the standard $N_k^{(0)} = 0$) discussed in Sec. 2.3. Figure 1 shows the evolution of the validation reconstruction loss of VQ-VAEs trained with EMA-VQ on CIFAR-10 [27] for two runs of 10 seeds each with $N_k^{(0)} = 0$ (denoted by `N=0/run1` and `N=0/run2`), and one run of 10 seeds with $N_k^{(0)} = 1$ (denoted by `N=1`). In both runs with `N=0`, we observe a plateau at the beginning of training. The variability of the results is also high: the performance of one of them (`N=0/run1`; the *blue* curve) remains above that of the `N=1` case, even after the plateau, while the other one (`N=0/run2`, the *orange* curve) successfully reaches the performance of `N=1`. The final/best validation reconstruction losses ($1e^{-3}$) achieved by the respective configurations are: 148.7 ± 299.7 vs. 52.1 ± 0.6. In contrast, such a variability was not observed with $N_k^{(0)} = 1$.

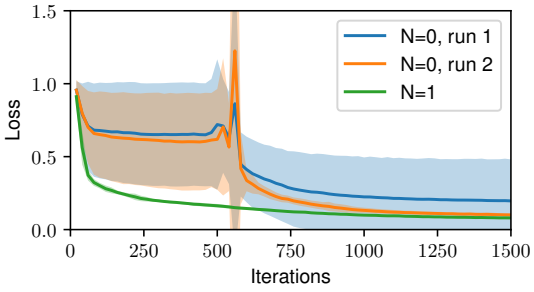


Figure 1: Evolution of validation reconstruction loss on CIFAR-10 for baseline EMA-VQ with different initialisations $N_k^{(0)}$ in Eq. 20.

Updating Clusters without Members. Now we also evaluate the effect of updating EMAs for the clusters that have no members in the batch. Table 1 shows the corresponding results. In all cases (with or without $N_k^{(0)} = 1$ initialisation), updating all clusters including those that have no member is crucial for good performance of the baseline EMA-VQ.

3.2 Reconstruction Performance and Speed

We evaluate the reconstruction performance and convergence speed of VQ-VAEs trained with KSOM. We conduct experiments on three datasets: CIFAR-10 [27], ImageNet [12], and a mixture of CelebA-HQ [22] and Animal Faces HQ (AFHQ; Choi et al. [7]). We use the basic VQ-VAE architecture [49] for CIFAR-10 and its extension VQ-VAE-2 [41] for ImageNet and CelebA-HQ/AFHQ without architectural modifications. Further experimental details can be found in Appendix B.

Table 1: Sensitivity of the baseline EMA-VQ. VQ-VAEs trained with EMA-VQ on CIFAR-10. "$N_k^{(0)}$" indicates its initialisation with 0 or 1. "Update-0" indicates whether to update clusters that have 0 members in the batch. "# Steps" is the number of training steps needed to achieve +10% of the final loss. Mean and std are computed using 10 seeds. For the $N_k^{(0)} = 0$ case, there is a high variability among results even with 10 seeds, as we report in Sec. 3.1: here, we report the result obtained using 10 *good* seeds.

$N_k^{(0)}$	Update-0	Loss ($1e^{-3}$)	# Steps ($1e^3$)
0	No	148.8 ± 11.0	13.5 ± 0.7
1	No	68.3 ± 0.8	16.3 ± 1.4
0	Yes	52.1 ± 0.6	6.8 ± 0.6
1	Yes	**51.9** ± 0.2	**5.4** ± 0.5

Table 2: Validation reconstruction loss and number of steps needed to achieve +10% and +20% of the final loss for various Kohonen-VAEs. "None" in column "Neighbours" corresponds to our *well-configured* EMA-VQ (Sec. 3.1; "$N_k^{(0)} = 1$" and "Update-0, Yes" in Table 3). 2D grid is used for both "Hard" and "Gaussian" cases. Mean and standard deviations are computed with 10 seeds for CIFAR-10 and 5 seeds each for ImageNet and CelebA-HQ/AFHQ. Image resolution is 32x32 for CIFAR-10 and 256x256 for ImageNet and CelebA-HQ/AFHQ.

	CIFAR-10			ImageNet			CelebA-HQ/AFHQ		
		# Steps ($1e^3$)			# Steps ($1e^3$)			# Steps ($1e^3$)	
Neighbours	Loss ($1e^{-3}$)	+10%	+20%	Loss ($1e^{-3}$)	+10%	+20%	Loss ($1e^{-4}$)	+10%	+20%
None	51.9 ± 0.2	5.4 ± 0.5	3.7 ± 0.5	**23.0** ± 0.4	15.2 ± 1.6	8.4 ± 0.6	18.6 ± 1.0	**17.0** ± 1.0	14.1 ± 1.4
Hard	52.1 ± 0.2	**5.2** ± 0.6	**2.5** ± 0.5	23.5 ± 0.4	**13.6** ± 2.2	**7.4** ± 0.9	**17.3** ± 0.1	**17.0** ± 1.6	**10.8** ± 1.3
Gaussian	**51.8** ± 0.2	**5.2** ± 0.6	3.0 ± 0.0	23.2 ± 0.4	14.0 ± 1.4	7.6 ± 0.6	17.5 ± 0.3	17.8 ± 1.8	11.8 ± 1.3

Comparison to Optimised EMA-VQ. We first compare models trained with KSOM with those trained using carefully configured EMA-VQ (Sec. 3.1). Table 2 summarises the results. We first observe that all methods achieve a similar validation reconstruction loss, with slight improvements obtained by KSOM over the baseline on CelebA-HQ/AFHQ. To compare the "speed of convergence," we measure the number of steps needed by each algorithm to achieve $+10\%$ and $+20\%$ of their final performance. Here "steps" correspond to the number of updates, and the batch size is the same for all methods. We observe that, indeed, KSOM tends to be faster than the basic EMA-VQ at the beginning of training, as can be seen in the column $+20\%$ (especially for the hard variant on CelebA-HQ/AFHQ). However, the baseline catches up later, and the difference becomes rather marginal at the $+10\%$ threshold: the corresponding speed up by KSOM is less than 5% relative compared to carefully configured EMA-VQ. In what follows, we show that KSOM is much more robust than EMA-VQ, and performs well under all configurations, including those that are sub-optimal for EMA-VQ.

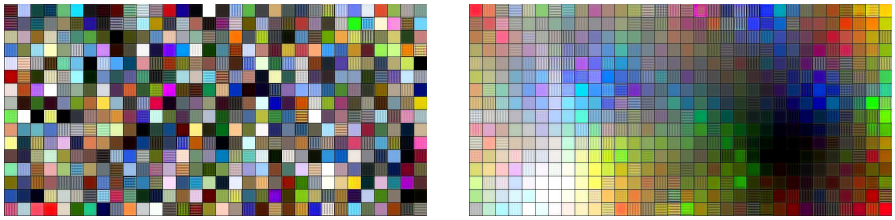
Robustness of KSOM against EMA-VQ Issues. Above we report that the performance gain (both in speed and reconstruction quality) by KSOM is rather marginal compared to our carefully configured EMA-VQ baseline obtained in Sec. 3.1. Here we compare the two approaches under various configurations. Table 3 shows the results. We observe that KSOM is remarkably robust: in all configurations, including those that are sub-optimal for EMA-VQ, KSOM achieves the same best validation loss as in the optimal configuration (Table 2). KSOM's neighbourhood updating scheme naturally fixes the problematic cases of the original EMA-VQ above. In these configurations, KSOM also generally converges faster than the baseline EMA-VQ. These results hold for both VQ-VAEs trained on CIFAR-10 and for VQ-VAE-2s trained on ImageNet and CelebA-HQ/AFHQ. In addition, in Appendix C.1, we show that KSOM also tends to improve the codebook utilisation.

3.3 Topologically Ordered Discrete Representations

Finally, we analyse the discrete representations learned by KSOM. We show that they are "topologically" ordered on the grid of indices, and consequently, reconstructed images remain close to the original ones even when we slightly shift their latent representations in the discrete index space.

Table 3: Validation reconstruction loss and number of steps needed to achieve +10% of the final loss ("# Steps"), showing the **robustness of KSOM** w.r.t. configurations that are sub-optimal for EMA-VQ. "$N_k^{(0)}$" indicates its initialisation with 0 or 1. "Update-0" indicates whether to update clusters that have 0 members in the batch. Numbers for the *hard* variant are reported (results are similar for Gaussian).

			CIFAR-10		ImageNet		CelebA-HQ/AFHQ	
	$N_k^{(0)}$	Update-0	Loss ($1e^{-3}$)	# Steps ($1e^3$)	Loss ($1e^{-3}$)	# Steps ($1e^3$)	Loss ($1e^{-4}$)	# Steps ($1e^3$)
EMA-VQ	0	No	148.8 ± 11.0	13.5 ± 0.7	44.7 ± 2.6	13.5 ± 0.7	37.3 ± 1.2	21.2 ± 3.9
KSOM			**51.8** $\pm$ 0.2	**5.2** $\pm$ 0.6	**24.4** $\pm$ 1.3	**12.6** $\pm$ 0.9	**17.4** $\pm$ 0.4	**17.6** $\pm$ 3.6
EMA-VQ	1	No	68.3 ± 0.8	16.3 ± 1.4	27.9 ± 1.0	21.2 ± 1.5	30.1 ± 2.8	22.6 ± 0.5
KSOM			**52.1** $\pm$ 0.3	**4.8** $\pm$ 0.9	**24.1** $\pm$ 1.3	**12.2** $\pm$ 1.3	**17.4** $\pm$ 0.2	**16.6** $\pm$ 1.5
EMA-VQ	0	Yes	52.1 ± 0.6	6.8 ± 0.6	**23.6** $\pm$ 0.9	15.4 ± 2.7	18.0 ± 1.0	17.6 ± 1.5
KSOM			**51.9** $\pm$ 0.2	**4.8** $\pm$ 0.6	23.7 ± 0.7	**14.2** $\pm$ 1.6	**17.2** $\pm$ 0.1	**16.8** $\pm$ 1.9



(a) EMA-VQ (b) KSOM

Figure 2: A visualization of codebook ($K = 512$) of VQ-VAEs trained on CIFAR10 with (a) EMA-VQ or (b) KSOM. The codebook of EMA-VQ obviously has no structure but serves as a reference. Similar visualisations for VQ-VAE-2 trained on ImageNet can be found in Appendix C.4.

Grid Visualisation. We first visualise how learned discrete representations are distributed over the grid. Here we use a VQ-VAE trained with KSOM (2D with hard neighbourhoods) on CIFAR-10, and proceed as follows. A discrete latent representation consists of N integers (Sec. 2.1). For each index in the codebook (corresponding to one of the nodes on the grid), we create a discrete latent representation whose N codes are all the same and equal to the corresponding index, and feed it to the decoder to obtain an output image. This results in a grid of images shown in Figure 2. We observe that each code seems to correspond to some colour, we can effectively observe several local "islands" of colours which group colours that are visually close.

Impact of Perturbation in the Discrete Latent Space. To further illustrate the presence of neighbourhoods developed by KSOM in the discrete latent space, we show images obtained by perturbing the discrete latent code representing a proper image in the index space (by adding or subtracting an integer offset to each coordinates of the code indices). Here we use VQ-VAE-2 trained on ImageNet with 2D KSOM with hard neighbourhoods. The VQ-VAE-2 [41] has two levels of discretisation: we shift all of them by the same offset on both x and y axes of the 2D grid for KSOM or directly shift the code indices for EMA-VQ. Figure 3 shows the results. Obviously, with the baseline VQ-VAE-2 trained with EMA-VQ, the output images become complete noises under such perturbations, even with an offset of one. With the KSOM-trained representations, there is a certain degree of continuity in the space of indices (as illustrated in Figure 2): the output images preserve the original contents though they become noisier as the offset increases. This illustrates the neighbourhoods learned by KSOM.

4 Discussion

Recommendations. Our first recommendation for any EMA-VQ implementation is to modify $N_k^{(0)} = 0$ to $N_k^{(0)} = 1$ (Sec. 3.1). For a more robust solution, we recommend using KSOM. Extending the standard implementation of EMA-VQ (Sec. 2.1) to KSOM (Sec. 2.2) is straightforward. While we introduce one extra hyper-parameter, shrink step τ (Eqs. 15-16), we found $\tau = 0.1$ to perform

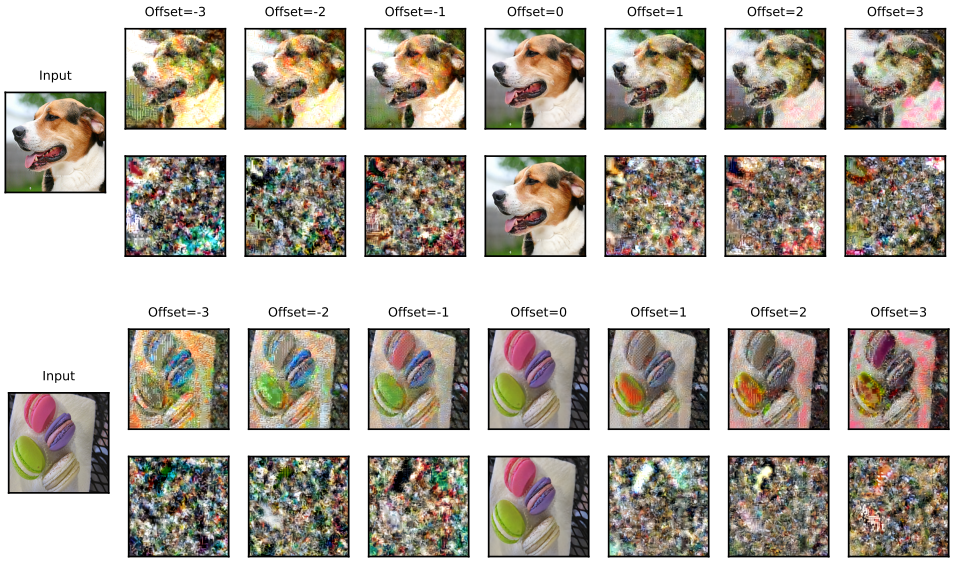


Figure 3: Effects of perturbations to the discrete latent code on reconstructed images. For each image, the **top row** shows the results for KSOM, and the **bottom row** shows those for EMA-VQ. "Offset" indicates the offset added to the indices of the latent representations.

well across all tasks. Regarding the model variations, we recommend using the 2D variant with hard neighbourhoods. Virtually, any VQ implementation should benefit from these modifications.

Related Work. The most related work is Fortuin et al. [15]'s *SOM-VAE* which is also inspired by KSOM. The core difference between the SOM-VAE and our approach is that Fortuin et al. [15] train the codebook weights by gradient descent: all neighbour weight vectors are updated to be close to the encoder output (see the last paragraph of Sec. 2.1). While this is indeed inspired by Kohonen's neighbourhoods, none of Kohonen's learning rules (Appendix A) is used. Also, the main focus of Fortuin et al. [15] is on modelling and interpreting time series. In fact, SOM-VAEs are extended by Manduchi et al. [35] for further applications to heath care. For empirical comparison of KSOM to SOM-VAE, we refer to Appendix C.3.

There are also several other works which attempt to improve the VQ algorithm. For example, Zeghidour et al. [60] propose to initialise all codebook weights using examples in the first batch. Lee et al. [28] propose residual VQ that iteratively performs VQ for improved reconstruction quality (note that Lee et al. [28] use EMA-VQ which can be directly replaced by KSOM).

Applications of VQ. Following the VQ-VAE of van den Oord et al. [49], VQ has become popular across various modalities and applications. Besides the standard use case for high-dimensional data such as images, audio, and video, VQ has been also used for texts. For example, Kaiser et al. [21], Roy et al. [43] downsample target sentences via VQ to speed up decoding in machine translation. Liu & Niehues [30] explore VQ to improve multi-lingual translation. Ozair et al. [39] applies VQ to model-based reinforcement learning/planning by quantising state-action sequences. Many text-to-image generation systems also include VQ components, e.g., Ramesh et al. [40] make use of discrete VAEs with Gumbel-softmax [20]), and Yu et al. [59] build upon VQ-GAN-2 [58]. Discrete representation learning is also motivated by out-of-distribution (or systematic) generalisation in certain tasks [31, 32, 48].

Differentiable Relaxation of KSOM. While working very well in practice, the use of the straight-through estimator (Sec. 2.1) to pass the gradients through the quantisation operation seems sub-optimal at first. For example, Agustsson et al. [1] show the possibility to perform discrete representation learning in fully differentiable NNs by using softmax with temperature annealing. In fact, we can also naturally derive a differentiable version of KSOM by replacing $\mathrm{hardmax}$ in Eq. 17 (presented in Appendix A) by the softmax function. However, in our preliminary experiments, none of

our differentiable variants with temperature annealing obtained a successful model that achieves a good reconstruction loss when the softmax is discretised for testing.

Semantic Codebook. While we demonstrate the emergence of neighbourhoods in the discrete space of codebook indices learned by KSOM, the features encoded by them remain low-level (mostly colours). While learning of more "semantic" discrete codes is out of scope here, we expect KSOM with such representations to be even more interesting, as it may potentially enable interpolation in the discrete latent space.

Other Variations of KSOM. Finally, there are several extensions of KSOM, e.g., *neural gas* [36, 16]. Hopefully, our work will inspire further research on improving discrete representation learning in modern NNs through KSOM variations or other algorithms going beyond EMA-VQ.

5 Conclusion

We revisit the learning rule of Kohonen's Self-Organising Maps (KSOM) as the vector quantisation (VQ) algorithm for discrete representation learning in NNs. KSOM is a generalisation of the exponential moving-average based VQ algorithm (EMA-VQ) commonly used in VQ-VAEs. We empirically demonstrate that, unlike the standard EMA-VQ, KSOM is robust w.r.t. initialisation and EMA update schemes. Our recipes can easily be integrated into existing code for VQ-VAEs. In addition, we show that discrete representations learned by KSOM effectively develop topological structures. We provide illustrations of the learned neighbourhoods in the image domain.

Acknowledgements

This research was partially funded by ERC Advanced grant no: 742870, project AlgoRNN, and by Swiss National Science Foundation grant no: 200021_192356, project NEUSYM. We are thankful for hardware donations from NVIDIA and IBM. The resources used for this work were partially provided by Swiss National Supercomputing Centre (CSCS) project s1145 and s1154.

References

[1] Agustsson, E., Mentzer, F., Tschannen, M., Cavigelli, L., Timofte, R., Benini, L., and Gool, L. V. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 1141–1151, December 2017.

[2] Amari, S.-I. Topographic organization of nerve fields. *Bulletin of Mathematical Biology*, 42(3): 339–364, 1980.

[3] Baevski, A., Schneider, S., and Auli, M. vq-wav2vec: Self-supervised learning of discrete speech representations. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2020.

[4] Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *Preprint arXiv:1308.3432*, 2013.

[5] Borsos, Z., Marinier, R., Vincent, D., Kharitonov, E., Pietquin, O., Sharifi, M., Teboul, O., Grangier, D., Tagliasacchi, M., and Zeghidour, N. AudioLM: a language modeling approach to audio generation. *Preprint arXiv:2209.03143*, 2022.

[6] Chen, S. F. and Goodman, J. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[7] Choi, Y., Uh, Y., Yoo, J., and Ha, J. StarGAN v2: Diverse image synthesis for multiple domains. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 8185–8194, Virtual only, June 2020.

[8] Constantinescu, A. O., O'Reilly, J. X., and Behrens, T. E. Organizing conceptual knowledge in humans with a gridlike code. *Science*, 352(6292):1464–1468, 2016.

[9] Cottrell, M., Olteanu, M., Rossi, F., and Villa-Vialaneix, N. Self-organizing maps, theory and applications. *Revista de Investigacion Operacional*, 39(1):1–22, 2018.

[10] Csordás, R., Irie, K., and Schmidhuber, J. CTL++: Evaluating generalization on never-seen compositional patterns of known functions, and compatibility of neural representations. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, Abu Dhabi, UAE, December 2022.

[11] De Bodt, E., Cottrell, M., Letremy, P., and Verleysen, M. On the use of self-organizing maps to accelerate vector quantization. *Neurocomputing*, 56:187–203, 2004.

[12] Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, Miami, Florida, USA, June 2009.

[13] Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., and Sutskever, I. Jukebox: A generative model for music. *Preprint arXiv:2005.00341*, 2020.

[14] Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 12873–12883, Virtual only, June 2021.

[15] Fortuin, V., Hüser, M., Locatello, F., Strathmann, H., and Rätsch, G. SOM-VAE: interpretable discrete representation learning on time series. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019.

[16] Fritzke, B. A growing neural gas network learns topologies. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 625–632, Denver, CO, USA, 1994.

[17] Hebb, D. O. The organization of behavior; a neuropsycholocigal theory. *A Wiley Book in Clinical Psychology*, 62:78, 1949.

[18] Hinton, G. Neural networks for machine learning. *Coursera, video lectures*, 2012.

[19] Hupkes, D., Singh, A., Korrel, K., Kruszewski, G., and Bruni, E. Learning compositionally through attentive guidance. In *Proc. Int. Conf. on Computational Linguistics and Intelligent Text Processing*, La Rochelle, France, April 2019.

[20] Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *Int. Conf. on Learning Representations (ICLR)*, Toulon, France, April 2017.

[21] Kaiser, L., Bengio, S., Roy, A., Vaswani, A., Parmar, N., Uszkoreit, J., and Shazeer, N. Fast decoding in sequence models using discrete latent variables. In *Proc. Int. Conf. on Machine Learning (ICML)*, pp. 2395–2404, Stockholm, Sweden, July 2018.

[22] Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of GANs for improved quality, stability, and variation. In *Int. Conf. on Learning Representations (ICLR)*, Vancouver, Canada, April 2018.

[23] Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *Int. Conf. on Learning Representations (ICLR)*, Banff, Canada, April 2014.

[24] Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

[25] Kohonen, T. Comparison of SOM point densities based on different criteria. *Neural Computation*, 11(8):2081–2095, 1999.

[26] Kohonen, T. *Self-organizing maps*. Springer (the first edition published in 1995), 2001.

[27] Krizhevsky, A. Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto, 2009.

[28] Lee, D., Kim, C., Kim, S., Cho, M., and Han, W.-S. Autoregressive image generation using residual quantization. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 11523–11532, New Orleans, LA, USA, June 2022.

[29] Liska, A., Kruszewski, G., and Baroni, M. Memorize or generalize? searching for a compositional RNN in a haystack. In *AEGAP Workshop ICML*, Stockholm, Sweden, July 2018.

[30] Liu, D. and Niehues, J. Learning an artificial language for knowledge-sharing in multilingual translation. In *Proc. Conf. on Machine Translation (WMT)*, pp. 188–202, Abu Dhabi, December 2022.

[31] Liu, D., Lamb, A., Kawaguchi, K., Goyal, A., Sun, C., Mozer, M. C., and Bengio, Y. Discrete-valued neural communication. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2109–2121, Virtual only, December 2021.

[32] Liu, D., Lamb, A., Ji, X., Notsawo, P., Mozer, M., Bengio, Y., and Kawaguchi, K. Adaptive discrete communication bottlenecks with dynamic vector quantization. *Preprint arXiv:2202.01334*, 2022.

[33] Lloyd, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982.

[34] MacQueen, J. Classification and analysis of multivariate observations. In *Proc. Berkeley Symp. Math. Statist. Probability*, pp. 281–297, 1967.

[35] Manduchi, L., Hüser, M., Faltys, M., Vogt, J. E., Rätsch, G., and Fortuin, V. T-DPSOM: an interpretable clustering method for unsupervised learning of patient health states. In *Proc. Conf. on Health, Inference, and Learning (CHIL)*, pp. 236–245, Virtual only, April 2021.

[36] Martinetz, T. and Schulten, K. A "neural-gas" network learns topologies. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, Espoo, Finland, June 1991.

[37] Nasrabadi, N. M. and Feng, Y. Vector quantization of images based upon the Kohonen self-organizing feature maps. In *Proc. IEEE Int. Conf. on Neural Networks (ICNN)*, volume 1, pp. 101–105, 1988.

[38] Oja, E. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[39] Ozair, S., Li, Y., Razavi, A., Antonoglou, I., van den Oord, A., and Vinyals, O. Vector quantized models for planning. In *Proc. Int. Conf. on Machine Learning (ICML)*, pp. 8302–8313, Virtual only, July 2021.

[40] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 139, pp. 8821–8831, Virtual only, December 2021.

[41] Razavi, A., van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with VQ-VAE-2. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pp. 14837–14847, Vancouver, Canada, December 2019.

[42] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10685, New Orleans, LA, USA, June 2022.

[43] Roy, A., Vaswani, A., Parmar, N., and Neelakantan, A. Towards a better understanding of vector quantized autoencoders. *OpenReview*, 2018.

[44] Schlag, I., Irie, K., and Schmidhuber, J. Linear Transformers are secretly fast weight programmers. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2021.

[45] Schmidhuber, J. Learning to control fast-weight memories: An alternative to recurrent nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, March 1991.

[46] Tirunagari, S., Bull, S., Kouchaki, S., Cooke, D., and Poh, N. Visualisation of survey responses using self-organising maps: a case study on diabetes self-care factors. In *Proc. IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–6, 2016.

[47] Tjandra, A., Sakti, S., and Nakamura, S. Transformer VQ-VAE for unsupervised unit discovery and speech synthesis: Zerospeech 2020 challenge. In *Proc. Interspeech*, pp. 4851–4855, Virtual only, October 2020.

[48] Träuble, F., Goyal, A., Rahaman, N., Mozer, M., Kawaguchi, K., Bengio, Y., and Schölkopf, B. Discrete key-value bottleneck. *Preprint arXiv:2207.11240*, 2022.

[49] van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 6306–6315, Long Beach, CA, December 2017.

[50] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 5998–6008, Long Beach, CA, USA, December 2017.

[51] von der Malsburg, C. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100, 1973.

[52] von der Malsburg, C. and Willshaw, D. J. How to label nerve cells so that they can interconnect in an ordered fashion. *Proc. the National Academy of Sciences*, 74(11):5176–5178, 1977.

[53] Walker, J., Razavi, A., and Oord, A. v. d. Predicting video with VQVAE. *Preprint arXiv:2103.01950*, 2021.

[54] Willshaw, D. J. and von der Malsburg, C. How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 194(1117):431–445, 1976.

[55] Willshaw, D. J. and von der Malsburg, C. A marker induction mechanism for the establishment of ordered neural mappings: its application to the retinotectal problem. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 287(1021):203–243, 1979.

[56] Yan, W., Zhang, Y., Abbeel, P., and Srinivas, A. VideoGPT: Video generation using vq-vae and transformers. *Preprint arXiv:2104.10157*, 2021.

[57] Yin, H. The self-organizing maps: background, theories, extensions and applications. In *Computational intelligence: A compendium*, pp. 715–762. 2008.

[58] Yu, J., Li, X., Koh, J. Y., Zhang, H., Pang, R., Qin, J., Ku, A., Xu, Y., Baldridge, J., and Wu, Y. Vector-quantized image modeling with improved VQGAN. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2022.

[59] Yu, J., Xu, Y., Koh, J. Y., Luong, T., Baid, G., Wang, Z., Vasudevan, V., Ku, A., Yang, Y., Ayan, B. K., et al. Scaling autoregressive models for content-rich text-to-image generation. *Preprint arXiv:2206.10789*, 2022.

[60] Zeghidour, N., Luebs, A., Omran, A., Skoglund, J., and Tagliasacchi, M. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507, 2021.

A Background: Kohonen Maps

Here we provide a brief but pedagogical review of Kohonen's Self-Organising Maps (KSOMs).

A.1 (Online) Algorithm

Teuvo Kohonen (1934-2021)'s Self-Organising Map [24] is an unsupervised learning/clustering algorithm which achieves both *vector quantisation* (VQ) and *topological mapping* (Sec. A.3). Let T, K, d_{in} and t denote positive integers. The algorithm requires to define a *distance function* $\delta : \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}_{\geq 0}$ as well as a *neighbourhood matrix* $\boldsymbol{A} \in \mathbb{R}^{K \times K}$ with $0 \leq \boldsymbol{A}_{i,j} \leq 1$ for all $i, j \in \{1, ..., K\}$, whose roles are specified later. We consider T input vectors[2] $(\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$ with $\boldsymbol{x}_t \in \mathbb{R}^{d_{\text{in}}}$ for $t \in \{1, ..., T\}$, and a weight matrix $\boldsymbol{W} \in \mathbb{R}^{K \times d_{\text{in}}}$ which we describe as a list of K weight vectors $(\boldsymbol{w}_1, ..., \boldsymbol{w}_K) = \boldsymbol{W}^\intercal$ with $\boldsymbol{w}_k \in \mathbb{R}^{d_{\text{in}}}$ for $k \in \{1, ..., K\}$ representing a *codebook* of size K. The algorithm clusters input vectors into K clusters where the prototype (or the centroid) of cluster $k \in \{1, ..., K\}$ is $\boldsymbol{w}_k$. At the beginning, these weight vectors are randomly initialised as $(\boldsymbol{w}_1^{(0)}, ..., \boldsymbol{w}_K^{(0)})$ where the super-script denotes the iteration step. The KSOM algorithm learns these weight vectors iteratively as follows.

For each step $t \in \{1, ..., T\}$, we process an input $\boldsymbol{x}_t$ by computing the index k^* (typically called *best matching unit*) of the weight vector that is the closest to the input $\boldsymbol{x}_t$ according to δ, i.e.,

$$k^* = \underset{1 \leq k \leq K}{\arg\min}\, \delta(\boldsymbol{x}_t, \boldsymbol{w}_k^{(t-1)}) \tag{12}$$

then the weights are updated; for all $k \in \{1, ..., K\}$,

$$\boldsymbol{w}_k^{(t)} = \boldsymbol{w}_k^{(t-1)} + \beta \boldsymbol{A}_{k^*,k}^{(t-1)} (\boldsymbol{x}_t - \boldsymbol{w}_k^{(t-1)}) \tag{13}$$

where $\beta \in \mathbb{R}_{>0}$ is the learning rate, and the super-script added to $\boldsymbol{A}_{k^*,k}^{(t)}$ indicates that it also changes over time.

Now, we need to specify the distance function δ in Eq. 12 and the neighbourhood matrix $\boldsymbol{A}^{(t)}$ in Eq. 13.

Distance function δ. A typical choice for δ is the Euclidean distance which we also use in all our experiments. Strictly speaking, δ does not have to be a metric; any kind of (dis)similarity function can be used, e.g., negative dot product is also a common choice (*dot-SOM* [26]).

Grid of Codebook Indices. In order to define the neighbourhood matrix $\boldsymbol{A}^{(t)}$, we first need to define a *grid* or lattice of the codebook indices on which *neighbourhoods* are defined. Let D denote a positive integer. We define a map $\mathcal{G} : \{1, ..., K\} \to \mathbb{N}^D$ which maps each codebook index $k \in \{1, ..., K\}$ to its Cartesian coordinates $\mathcal{G}(k) \in \mathbb{N}^D$ in the D-dimensional space representing the grid in question. Typically, the grid is 1D or 2D ($D = 1$ or 2). In the 1D case, the original index $k \in \{1, ..., K\}$ and its coordinates on the map $\mathcal{G}(k)$ are the same: $\mathcal{G}(k) = k$. In the 2D case, $\mathcal{G}(k)$ corresponds to the Cartesian (x, y)-coordinates on a 2D-rectangular grid formed by K nodes (assuming that K is chosen such that this is possible). Once $\mathcal{G}$ is defined, one can measure the "distance" between two codebook indices $i, j \in \{1, ..., K\}$ as their Euclidean distance on the grid, i.e., $\|\mathcal{G}(i) - \mathcal{G}(j)\|$. This finally allows us to define the neighbourhood: given a pre-defined threshold distance $d(\mathcal{G}) \in \mathbb{R}_{\geq 0}$, for $i, j \in \{1, ..., K\}$, i is within the neighbourhood of j on the map $\mathcal{G}$ if and only if $\|\mathcal{G}(i) - \mathcal{G}(j)\| \leq d(\mathcal{G})$.

Neighbourhood Matrix $\boldsymbol{A}^{(t)}$. In Eq. 13, the coefficient of the neighbourhood matrix $\boldsymbol{A}_{k^*,k}^{(t)}$ has the role of adapting the weights/strengths of updates for each cluster k according to its distance from the best matching unit k^* on the map $\mathcal{G}$: as can be seen in Eq. 13, $\beta \boldsymbol{A}_{k^*,k}^{(t)}$ is the effective learning rate for the update. Essentially, the best matching unit obtains the full update $\boldsymbol{A}_{k^*,k^*}^{(t)} = 1$, while the updates for all others ($k \neq k^*$) are scaled down by $0 \leq \boldsymbol{A}_{k^*,k}^{(t)} \leq 1$. In practice, there are various ways to define such an $\boldsymbol{A}^{(t)}$. Here we focus on two variants: *hard* and *Gaussian* neighbourhoods.

[2] Here we use T as both the number of inputs and iterations.

Let us first define the *hard* variant *without* dependency on the time index t. In the *hard* variant, for all $i \in \{1, ..., K\}$, $\boldsymbol{A}_{i,i} = 1$, and for all i, j such that $i \neq j$,

$$\boldsymbol{A}_{i,j} = \begin{cases} 1 & \text{if } \|\mathcal{G}(i) - \mathcal{G}(j)\| \leq d(\mathcal{G}) \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

In the 1D case, setting $d(\mathcal{G}) = 1$ yields two neighbour indices. Similarly, in the 2D case, $d(\mathcal{G}) = \sqrt{2}$ defines the eight surrounding nodes as the neighbours, which are illustrated in Figure 4.

In practice, we introduce an extra dependency on time t with the goal of *shrinking* the neighbourhood over time. For all $i \in \{1, ..., K\}$, $\boldsymbol{A}^{(t)}_{i,i} = 1$, and for all i, j such that $i \neq j$,

$$\boldsymbol{A}^{(t)}_{i,j} = \begin{cases} 1/(1 + t * \tau) & \text{if } \|\mathcal{G}(i) - \mathcal{G}(j)\| \leq d(\mathcal{G}) \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where $\tau \in \mathbb{R}_{>0}$ is a hyper-parameter representing the *shrinking step* which controls the speed of shrinking: larger τ implies faster shrinking. Shrinking in KSOM is important to obtain good performance at convergence [26].

In the alternative, *Gaussian* variant, $\boldsymbol{A}^{(t)}_{i,j}$ is expressed as an exponential function of the distance between indices on $\mathcal{G}$:

$$\boldsymbol{A}^{(t)}_{i,j} = \exp(-\|\mathcal{G}(i) - \mathcal{G}(j)\|^2 / \sigma(t)^2) \tag{16}$$

where for $\sigma(t) \in \mathbb{R}$ we take $\sigma(t)^2 = 1/(1 + t * \tau)$. One can verify that, also in this case, $\boldsymbol{A}^{(t)}$ reduces to an identity matrix when $t \to +\infty$. Figure 5 provides an illustration.

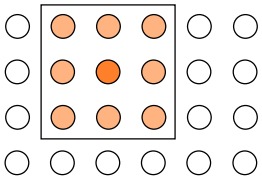
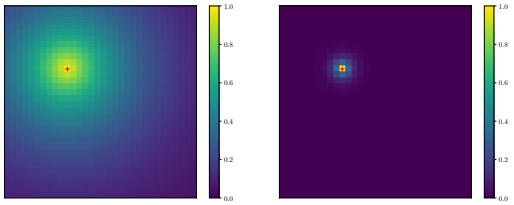


Figure 4: An illustration of *hard* neighbourhoods (Eq. 14) in the 2D case. This is a 4x6 grid with $K = 24$ nodes. Considering the left-bottom corner node as the origin (0, 0), the eight neighbours of the node (2, 2) are highlighted.



(a) Early training (b) Mid training

Figure 5: An illustration of *Gaussian* neighbourhoods in the 2D case with shrinking (Eq. 16) at two different stages of training.

Relation to Hebbian Learning. We note that the online algorithm of Eqs. 12-13 can be also interpreted (see also Yin [57] on this relation) as a variant of Hebbian learning [17]. Using the dot product based similarity function for δ in Eq. 12, and by defining $\text{hardmax} : \mathbb{R}^K \to \mathbb{R}^K$ as the function that outputs 1 for the largest entry of the input vector, and 0 for all others, we can express Eqs. 12-13 in a fully matrix-form by defining a one-hot vector $\boldsymbol{y}_t \in \mathbb{R}^K$,

$$\boldsymbol{y}_t = \text{hardmax}(\boldsymbol{W}_{t-1}\boldsymbol{x}_t) \tag{17}$$

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} + (\beta \boldsymbol{A}^{(t-1)}\boldsymbol{y}_t) \otimes (\boldsymbol{x}_t - \boldsymbol{W}^{\intercal}_{t-1}\boldsymbol{y}_t) \tag{18}$$

where $\otimes$ denotes outer product. With the neighbourhood reduced to zero, this is essentially the *winner-take-all* Hebbian learning. While this relation is not central to this work, we come back to this when we discuss the differentiable version of KSOM later in Sec. 4.

In passing, we also note that the last term in Eqs. 13 and 18 corresponds to Oja [38]'s *forgetting term* which is not part of the original 1982 algorithm [24] but has been added later (see, e.g., Kohonen [26]).

A.2 Batch Algorithm & Relation to K-means

The algorithm described above is an *online* algorithm which updates weights after every input. The *batch* version thereof (Kohonen [25, 26]; see also Cottrell et al. [9]), which takes into account all data points $(\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$ for a single update of $\{\boldsymbol{w}_1, ..., \boldsymbol{w}_K\}$, can be defined as follows.

At each iteration step t, we compute the best matching unit for each input x_i for all $i \in \{1, .., T\}$ (we now use the sub-script i to index data points not to confuse it with the iteration index t). Each input x_i is a *member* of one of the K clusters. The results can be summarised for each cluster $k \in \{1, .., K\}$ as the set $\mathcal{C}_k^{(t)}$ containing indices of its members at step t. We denote its cardinality by $|\mathcal{C}_k^{(t)}|$. The batch algorithm updates the weight vector $\boldsymbol{w}_k^{(t)}$ for cluster k as the average of its members and their neighbours weighted by the neighbourhood coefficients. That is, $\boldsymbol{w}_k^{(t)}$ is computed as the quotient of the sum $\boldsymbol{m}_k^{(t)}$ of all inputs belonging to the corresponding cluster k and their neighbours weighted by the neighbourhood coefficients, and the corresponding weighted count $N_k^{(t)}$, i.e.,

$$\boldsymbol{m}_k^{(t)} = \sum_{j=1}^{K} \boldsymbol{A}_{j,k}^{(t-1)} \sum_{i \in \mathcal{C}_j^{(t)}} \boldsymbol{x}_i \tag{19}$$

$$N_k^{(t)} = \sum_{j=1}^{K} \boldsymbol{A}_{j,k}^{(t-1)} |\mathcal{C}_j^{(t)}| \tag{20}$$

$$\boldsymbol{w}_k^{(t)} = \boldsymbol{m}_k^{(t)} / N_k^{(t)} \tag{21}$$

Remark A.1 (Relation to K-means). In the case where the *neighbourhood is reduced to zero*, i.e., for all t, $\boldsymbol{A}_{i,i}^{(t)} = 1$ and for all i, j such that $i \neq j$, $\boldsymbol{A}_{i,j}^{(t)} = 0$, Eqs. 19-21 reduce to the standard *K-means* algorithm [33]. Similarly, in such a case, Eqs. 12-13 reduce to an *online K-means* algorithm [34].

For deep learning applications, the algorithm needs to be both *online* and *mini-batch*; we discuss the corresponding extensions in Sec. 2.

A.3 Topographical Maps in the Brain as Motivation

The sub-sections above describe how KSOM performs clustering, i.e., *vector quantisation*. Here we discuss another property of this algorithm which is *topological mapping*.

It is known that there are multiple levels of *topographical maps* in the brain. For example, different regions of the brain specialise to different types of sensory inputs (vision, audio, touch, etc), and e.g., within the somatosensory part, regions that are responsible for different parts of the body are *ordered* according to the anatomical order in the body. A famous illustration of this is the "sensory homunculus."

The design of KSOM is inspired by such topographical maps. Many have proposed computational mechanisms to achieve such a property in the 1970s/80s [51, 52, 55, 54, 2]. Kohonen [24] achieves this by introducing the concept of neighbourhoods between the (output) neurons. In the algorithm above (Sec. A.1), all output neurons first compete against each other (Eq. 12) to yield a winner neuron. Then, the update is distributed to neurons that are spatially close to the winner through the coefficients of the neighbourhood matrix (Eqs. 13 and 19). As a result, clusters whose indices are spatially close on the grid are encouraged to store inputs that are close to each other in the feature space.

This is an unconventional feature for artificial NNs, since unlike the biological ones, artificial NNs do not have any physical constraints; there is no geometry nor distance between neurons. KSOM's neighbourhoods introduce such a structure. From the machine learning perspective, the resulting topological ordering has limited practical benefits. Even if it may potentially facilitate interpretation via direct visualisation, other embedding visualisation tools could fit the bill equally well. From the neuroscience perspective, however, it may be a property that contributes in filling the gap between artificial NNs and the biological ones (see also Constantinescu et al. [8]).

B Experimental Details

Datasets. We use three datasets (one of them is a mixture of two standard datasets): CIFAR-10 [27], ImageNet [12], and a mixture of CelebA-HQ [22] and Animal Faces HQ (AFHQ; Choi et al. [7]). The number of images and the resolution we use are: 60 K and 32x32, 1.3 M and 256x256, 43 K (the sum of 28 K for CelebA-HQ and 15 K for AFHQ) and 256x256, respectively. For any further details, we refer the readers to the original references.

Model Architectures. We use the basic configuration (including hyper-parameters) of VQ-VAE [49] for CIFAR-10 and that of VQ-VAE-2 [41] for ImageNet and CelebA-HQ/AFHQ from the original papers. The only change we introduce is the learning algorithm to train the codebook weights (Sec. 2.2). Again, for any further details on the model architectures, we refer the readers to the original references above. The dimension of latent representation for VQ-VAE on CIFAR-10 is 8x8x64, i.e, $N = 8 \times 8 = 64$ and $d_{\text{emb}} = 64$ using our notations of Sec. 2.1. VQ-VAE-2 has two codebooks ("top" and "bottom") with respective embedding dimensions of 32x32x64 ($N = 32 \times 32$) and 64x64x64 ($N = 64 \times 64$) with $d_{\text{emb}} = 64$ for both. We use the codebook size of $K = 512$ everywhere for all datasets.

Reference Baseline Implementations. We looked into several public implementations of the baseline VQ-VAEs, including the original implementation of VQ-VAE/EMA-VQ `https://github.com/deepmind/sonnet/blob/v2/sonnet/src/nets/vqvae.py`, its PyTorch re-implementation `https://github.com/zalandoresearch/pytorch-vq-vae` (see also `https://github.com/lucidrains/vector-quantize-pytorch`), and VQ-VAE-2 `https://github.com/rosinality/vq-vae-2-pytorch`. The GitHub link to our code can be found on the first page.

C Extra Experimental Results

Here we provide several extra experimental results which we could not report in the main text due to space limitation.

C.1 Codebook Utilisation

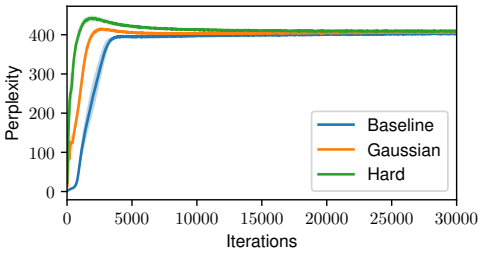
Here we show that KSOM also helps boosting the codebook utilisation of VQ-VAE and VQ-VAE-2. As a measure of codebook utilisation, we compute *perplexity* for each batch as $\exp(\sum_{k=1}^{K} p(k) \log(p(k)))$ where $p(k) = C(k)/(N * B)$ with $C(k)$ denoting the number of times the code k is used in the batch (with a batch size B and the number of embeddings N; the same notation as in the main text). Figure 6 shows the evolution of perplexity on all datasets (for VQ-VAE-2s, we show that for the two codebooks). With the only exception of the "bottom" codebook of VQ-VAE-2 trained on CelebA-HQ/AFHQ, overall, we observe that the code utilisation of KSOM variants tends to be higher than that of the baseline EMA-VQ. In particular, for the "top" codebook of VQ-VAE-2 trained on CelebA-HQ/AFHQ (Figure 6 (d)), the hard KSOM's mean perplexity exceeds 300 while the baseline EMA-VQ's is below 100.

C.2 Ablation Studies

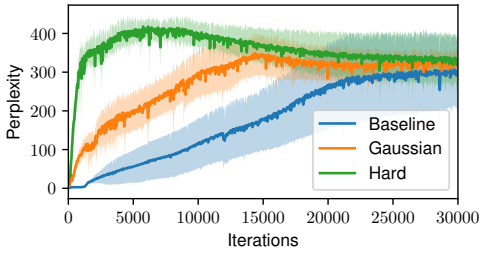
In the main text, we show that the performances of the hard and Gaussian variants are rather close (Table 2). We also conduct ablation studies to compare 1D vs. 2D grids, and different values of shrinking step $\tau \in \{1, 0.1, 0.01\}$, and observe that these different variations yield rather similar performance. For the sake of completeness, Table 4 presents the corresponding results. In terms of neighbourhoods, as expected, we find that the variants with small shrinking steps τ of 0.01 or 0.1 yield smoother codebook grids than those obtained with $\tau = 1$.

C.3 Comparison to Gradient-based Codebook Learning Methods

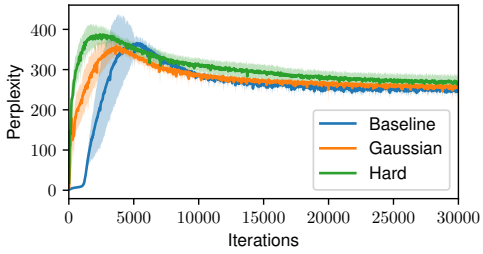
Here we compare KSOM to gradient-based codebook learning methods, including SOM-VAE [15]. Table 5 shows the results. The final reconstruction loss is similar for the EMA baseline, SOM-VAE, and KSOM, but KSOM converges the fastest (significantly faster than SOM-VAE). We also confirm that the EMA-based codebook learning outperforms the gradient-based variant (first row) as noted by the original authors of VQ-VAE (see, `https://twitter.com/avdnoord/status/1001853279649910784?lang=en`). We focus on KSOM because it is the natural extension of EMA which is recommended over the gradient-based variant (while SOM-VAE is a natural extension of the gradient-based variant).



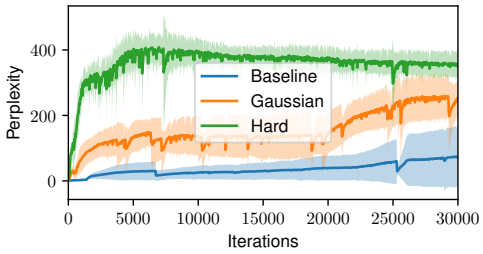
(a) CIFAR-10, VQ-VAE



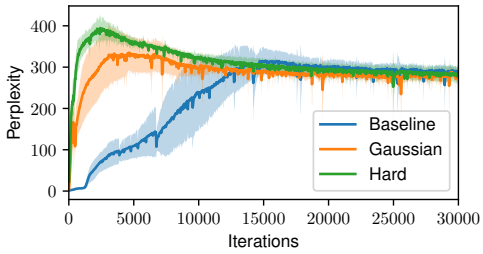
(b) ImageNet, VQ-VAE-2, Top



(c) ImageNet, VQ-VAE-2, Bottom



(d) CelebA-HQ/AFHQ, VQ-VAE-2, Top



(e) CelebA-HQ/AFHQ, VQ-VAE-2, Bottom

Figure 6: Evolution of perplexity (codebook utilisation) as a function of training iterations. The codebook size is $K = 512$ in all cases. "Baseline" is the standard EMA-VQ. "Hard" and "Gaussian" indicate the corresponding neighbourhood type for KSOM.

C.4 More Visualisations for VQ-VAE-2

Grid Visualisation for VQ-VAE-2. In Figure 2 in the main text, we visualise the 2D grid representation of the codebook for a VQ-VAE trained on CIFAR-10. Here we show similar visualisations for VQ-VAE-2 trained on ImageNet. One complication for such visualisations for VQ-VAE-2 is that it has two codebooks ("top" and "bottom"). We therefore cannot exhaustively visualise all combinations. We show a few of them by fixing all codes in either the top or bottom representations, and varying the others. Figure 7 shows the results. While hue/value/saturation change for different values of fixed top or bottom code, we again observe many "islands" of similar colours grouped together.

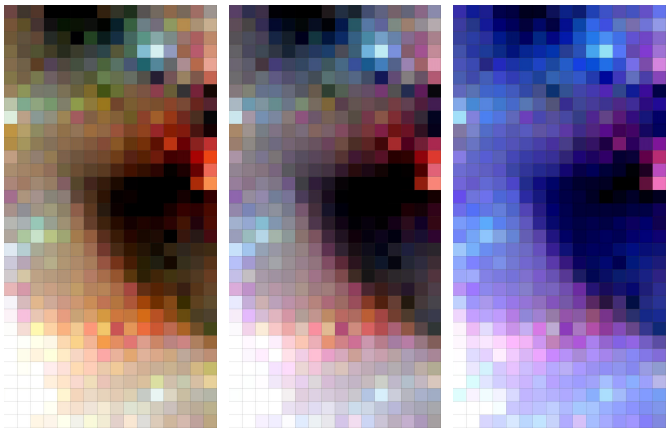
Perturbing only one of the two latent representations in VQ-VAE-2. In Figure 3 in the main text, we apply an offset to all indices in the two latent representations ("top" and "bottom") of VQ-VAE-2. Here we show the effect of perturbing only one of the two latent representations, i.e., we keep one of the "top" or "bottom" latent representations fixed (to that of a proper image), and apply offsets to the other one. Figures 8 and 9 show the corresponding results. Since one of the two latent representations is fixed, the "contents" of the original image is somewhat preserved in all cases. The comparison is interesting in the case where we perturb the top latent code: in the KSOM case, the changes of top latent code seem to gradually affect the hue/value of the image.

Table 4: Ablation studies on the 1d/2D grid and shrinking step size τ. Validation reconstruction loss and number of steps needed to achieve +10% and +20% of the final loss. Mean and standard deviations are computed with 10 seeds for CIFAR-10 and 5 seeds each for CelebA-HQ/AFHQ. Image resolution is 32x32 for CIFAR-10 and 256x256 for CelebA-HQ/AFHQ. The best numbers for "hard" and "Gaussian" groups are highlighted in **bold** separately.

		CIFAR-10			CelebA-HQ/AFHQ		
			# Steps ($1e^3$)			# Steps ($1e^3$)	
Neighbours	τ	Loss ($1e^{-3}$)	+10%	+20%	Loss ($1e^{-4}$)	+10%	+20%
Hard 1D	0.1	**52.0** $\pm$ 0.2	5.3 $\pm$ 0.5	3.1 $\pm$ 0.3	17.4 $\pm$ 0.2	**15.8** $\pm$ 0.8	**10.8** $\pm$ 1.5
Hard 2D	1	52.1 $\pm$ 0.4	**4.7** $\pm$ 0.7	2.6 $\pm$ 0.5	17.4 $\pm$ 0.2	18.0 $\pm$ 1.6	11.4 $\pm$ 1.1
Hard 2D	0.1	**52.0** $\pm$ 0.3	4.9 $\pm$ 0.3	**2.4** $\pm$ 0.5	**17.3** $\pm$ 0.2	18.0 $\pm$ 1.6	12.4 $\pm$ 1.1
Hard 2D	0.01	52.1 $\pm$ 0.3	5.5 $\pm$ 0.7	3.0 $\pm$ 0.0	17.6 $\pm$ 0.3	17.0 $\pm$ 2.9	12.4 $\pm$ 2.1
Gaussian 1D	0.1	51.9 $\pm$ 0.4	5.2 $\pm$ 0.4	3.1 $\pm$ 0.3	17.8 $\pm$ 0.5	18.0 $\pm$ 2.7	**11.6** $\pm$ 1.3
Gaussian 2D	1	51.9 $\pm$ 0.2	5.3 $\pm$ 0.5	3.3 $\pm$ 0.5	18.3 $\pm$ 0.4	**16.4** $\pm$ 1.9	11.8 $\pm$ 1.8
Gaussian 2D	0.1	51.8 $\pm$ 0.3	**5.1** $\pm$ 0.3	**3.1** $\pm$ 0.3	17.6 $\pm$ 0.2	18.6 $\pm$ 2.1	13.2 $\pm$ 0.8
Gaussian 2D	0.01	**51.7** $\pm$ 0.2	5.4 $\pm$ 0.5	3.6 $\pm$ 0.5	**17.5** $\pm$ 0.5	18.4 $\pm$ 2.6	11.8 $\pm$ 1.3

Table 5: Validation reconstruction loss and number of steps needed to achieve +10% of the final loss ("# Steps") on **CIFAR-10**.

Codebook Learning Algorithm	Loss ($1e^{-3}$)	# Steps ($1e^3$)
Gradient-based	69.4 $\pm$ 0.8	14.3 $\pm$ 0.9
EMA	51.9 $\pm$ 0.2	5.4 $\pm$ 0.5
SOM-VAE (Gradient-based)	51.7 $\pm$ 0.3	7.3 $\pm$ 0.5
KSOM	52.1 $\pm$ 0.2	5.2 $\pm$ 0.6



(a) Top=(0, 0) (b) Top=(18, 15) (c) Top=(27, 3)

(d) Bottom=(0, 0) (e) Bottom=(18, 15) (f) Bottom=(27, 3)

Figure 7: Codebook visualisations for VQ-VAE-2 trained on ImageNet with 2D-Hard KSOM. The description in the caption specifies which of the top or bottom latent representations is fixed to which coordinates.

(a) KSOM, shift top representation

(b) KSOM, shift bottom representation

(c) EMA-VQ Baseline, shift top representation

(d) EMA-VQ Baseline, shift bottom representation

Figure 8: Effects of perturbing only one of the top or bottom latent representations in VQ-VAE-2 (“dog”)

(a) KSOM, shift top representation

(b) KSOM, shift bottom representation

(c) EMA-VQ Baseline, shift top representation

(d) EMA-VQ Baseline, shift bottom representation

Figure 9: Effects of perturbing only one of the top or bottom latent representations in VQ-VAE-2 (“macaron”)