Straightening Out the Straight-Through Estimator: Overcoming Optimization Challenges in Vector Quantized Networks

Minyoung Huh [1] **Brian Cheung** [1 2] **Pulkit Agrawal** [1] **Phillip Isola** [1]

Abstract

This work examines the challenges of training neural networks using vector quantization using straight-through estimation. We find that a primary cause of training instability is the discrepancy between the model embedding and the code-vector distribution. We identify the factors that contribute to this issue, including the codebook gradient sparsity and the asymmetric nature of the commitment loss, which leads to misaligned code-vector assignments. We propose to address this issue via affine re-parameterization of the code vectors. Additionally, we introduce an alternating optimization to reduce the gradient error introduced by the straight-through estimation. Moreover, we propose an improvement to the commitment loss to ensure better alignment between the codebook representation and the model embedding. These optimization methods improve the mathematical approximation of the straight-through estimation and, ultimately, the model performance. We demonstrate the effectiveness of our methods on several common model architectures, such as AlexNet, ResNet, and ViT, across various tasks, including image classification and generative modeling.

Project page: `minyoungg.github.io/vqtorch`

1. Introduction and related works

Vector-quantization (Gray, 1984) enables deep neural networks to learn discrete representations by quantizing features into clusters referred to as "code-vectors" or "codes". Vector-Quantization (VQ) is a parametric online K-means algorithm (Caron et al., 2018) that has an explicit bias for compression and competition, which serves as a good prior for learning disentangled features for downstream tasks. To name a few, VQs have shown impressive results on image generation (Van Den Oord et al., 2017; Ramesh et al., 2021; Esser et al., 2020; Chang et al., 2023), image representation learning (Caron et al., 2020), speech generation (Dhariwal et al., 2020), speech representation learning (Chung et al., 2020), and even decision-making (Ozair et al., 2021). While powerful, vector-quantized networks (VQNs) are notoriously difficult to optimize and require esoteric knowledge to efficiently train them. Hence, constructing algorithms to improve training stability has been a topic of great interest.

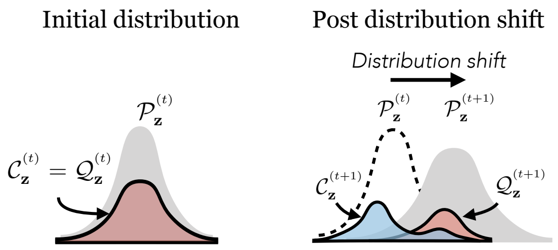
Initial distribution
Post distribution shift
Distribution shift
$\mathcal{P}_{\mathbf{z}}^{(t)}$
$\mathcal{C}_{\mathbf{z}}^{(t)} = \mathcal{Q}_{\mathbf{z}}^{(t)}$
$\mathcal{P}_{\mathbf{z}}^{(t)}$
$\mathcal{P}_{\mathbf{z}}^{(t+1)}$
$\mathcal{C}_{\mathbf{z}}^{(t+1)}$
$\mathcal{Q}_{\mathbf{z}}^{(t+1)}$

Figure 1: **Illustration of internal codebook covariate shift:** During training, the embedding distribution $\mathcal{P}_{\mathbf{z}}$ drifts from initialization. When the model undergoes a distributional shift, the codebook $\mathcal{C}_{\mathbf{z}}$ (blue) is misaligned with $\mathcal{P}_{\mathbf{z}}$. The code-vectors that have assignments are denoted with $\mathcal{Q}_{\mathbf{z}}$ (red) and initialized to overlap with $\mathcal{C}_{\mathbf{z}}$. With training, $\mathcal{Q}_{\mathbf{z}}$ diverge and are misaligned with $\mathcal{P}_{\mathbf{z}}$. Code-vectors without assignment do not receive gradients and are no longer trained, which leads to bifurcation in the codebook distribution and ultimately leads to index collapse.

First introduced in context of generative models (Van Den Oord et al., 2017), the vector-quantization layer in VQNs maps the continuous embedding (or feature representation) $\mathcal{P}_{\mathbf{z}}$ into a discrete embedding $\mathcal{Q}_{\mathbf{z}}$ using the codebook $\mathcal{C}_{\mathbf{z}}$. As the discretization function is not continuously differentiable, a widely used technique to optimize VQNs is via straight-through estimation. This bypasses the non-differentiable discretization function (Bengio et al., 2013), allowing it to be optimizable by standard deep learning libraries (Paszke et al., 2019). Of course, straight-through estimating a selection function has negative ramifications on training. Among many training challenges, the most well-documented is model collapse or"index collapse" wherein only a small fraction of codes are used during training. While the root cause of the collapse is not well understood, there have been abundant efforts to mitigate index

[*]Equal contribution [1]MIT CSAIL [2]MIT BCS. Correspondence to: Minyoung Huh <minhuh@mit.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

collapse: EMA (Van Den Oord et al., 2017), codebook reset (Łańcucki et al., 2020; Zeghidour et al., 2021; Dhariwal et al., 2020),probabilistic/stochastic re-formulation (Roy et al., 2018; Takida et al., 2022), equipartition assumption using optimal transport (Asano et al., 2020), and many more (See Section 3). While many of these methods directly reduce the extent of index collapse, to the best of our knowledge, there has not been any work that investigates the root cause of the instability in the first place. Hence, our work aims to systematically investigate the cause of the model collapse and provide methods to address the common pitfalls that stem from unstable optimization. Concretely, our contributions are as follows:

- We provide new insights into understanding VQ networks by formulating commitment loss as a divergence measure. Doing so allows us to understand better why the divergence occurs.
- To reduce this divergence, we propose an affine reparameterization of the code-vectors that can better match the moments of the embedding representation. This alone drastically reduces the model collapse.
- Lastly, We provide a set of improvements on the existing optimization techniques, such as alternating optimization and synchronized commitment loss. Both these methods are simple and more mathematically correct update rules that result in improvements over the standard approach.

2. Preliminaries

We denote x as a scalar, $\mathbf{x}$ as a vector, X as a matrix, $\mathcal{X}$ as a distribution or a set, $f(\cdot)$ as a function, $F(\cdot)$ as a composition of functions, and $\mathcal{L}(\cdot)$ for loss function.

▷ Deep neural networks
A feed-forward neural network is defined as a composition of parametric linear functions f_i (*e.g.* fully-connected, convolutional layer) and non-linearities σ (*e.g.* ReLU):

$$\hat{\mathbf{y}} = \underbrace{f_n \circ \sigma \circ \cdots \circ \sigma \circ f_{i+1}}_{G} \circ \underbrace{f_i \circ \sigma \circ \cdots \circ f_2 \circ \sigma \circ f_1}_{F} \circ \mathbf{x} \quad (1)$$

$$= G(F(\mathbf{x})) \quad (2)$$

In the context of generative modeling, $F(\cdot)$ is referred to as the encoder and $G(\cdot)$ as the decoder. The network is trained by minimizing the empirical risk $\mathcal{L}_{\text{task}}(\cdot)$ with dataset $\mathcal{D}$:

$$\min_{F,G} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}} \left[\mathcal{L}_{\text{task}}\left(G(F(\mathbf{x})), \mathbf{y}\right)\right] \quad (3)$$

▷ Vector-quantized networks
A vector-quantized network (VQN) is a neural-network consisting of a vector-quantization layer $h(\cdot,\cdot)$:

$$\hat{\mathbf{y}} = G(h(F(\mathbf{x}), C)) = G(h(\mathbf{z}_e, C)) = G(\mathbf{z}_q) \quad (4)$$

The VQ layer $h(\cdot)$ quantizes the embedding $\mathbf{z}_e = F(\mathbf{x})$ by selecting a vector from a collection of m vectors. The individual vector $\mathbf{c}_i$ is referred to as the code-vector, the index i as the code, and the collection of the code-vectors as the codebook $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots \mathbf{c}_m\}$. Here on out, we omit writing the codebook C in the quantization function $h(\cdot)$ for notational convenience. In the quantization function $h(\cdot)$, $\mathbf{z}_e$ is quantized into $\mathbf{z}_q$ by assigning a code-vector from the codebook C using a distance measure $d(\cdot,\cdot)$:

$$\mathbf{z}_q = \mathbf{c}_k, \quad \text{where} \quad k = \arg\min_j d(\mathbf{z}_e, \mathbf{c}_j) \quad (5)$$

Euclidean distance is the standard distance measure for $d(\cdot,\cdot)$ (Van Den Oord et al., 2017). We denote the set associated to $\mathbf{z}_e$, $\mathbf{z}_q$ and $\mathbf{c}$ with $\mathcal{P}_{\mathbf{z}}$, $\mathcal{Q}_{\mathbf{z}}$ and $\mathcal{C}_{\mathbf{z}}$, respectively, with $\mathcal{Q}_{\mathbf{z}} \subseteq \mathcal{C}_{\mathbf{z}}$. Without loss of generality, we assume these sets are constructed from an underlying distribution.

The quantized embedding is then used to predict the output $\hat{\mathbf{y}} = G(\mathbf{z}_q)$, and the loss is computed with the target $\mathbf{y} : \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$. For images, VQ is generally performed on each spatial location of the tensor, where the channel dimension is used to represent the vector (*e.g.* each spatial location (i,j) of $\mathbf{z}_e \in \mathbf{R}^{h\times w\times c}$ is quantized $\mathbf{z}_e[i,j] \in \mathbf{R}^c \xrightarrow{h(\cdot)} \mathbf{z}_q[i,j] \in \mathbf{R}^c$).

Akin to standard training, the objective of VQNs is to minimize the empirical risk:

$$\min_{F,G,h} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}} \left[\mathcal{L}_{\text{task}}\left(G(h(F(\mathbf{x}))), \mathbf{y}\right)\right] \quad (6)$$

The equation above is not continuously differentiable. To differentiate through the $\arg\min$ operator in $h(\cdot)$, a *straight-through* estimation (Bengio et al., 2013) is applied:

$$\frac{\partial\mathcal{L}}{\partial F} = \frac{\partial\mathcal{L}}{\partial\hat{\mathbf{y}}}\frac{\partial\hat{\mathbf{y}}}{\partial\mathbf{z}_q}\underbrace{\cancel{\frac{\partial\mathbf{z}_q}{\partial\mathbf{z}_e}}}_{\text{straight through}}\frac{\partial\mathbf{z}_e}{\partial F} \approx \frac{\partial\hat{\mathcal{L}}}{\partial F} \quad (7)$$

To ensure the straight-through estimation is accurate, the codebook and the encoder representations are pulled together using a *commitment loss*:

$$\mathcal{L}_{\text{cmt}}(\mathbf{z}_e, \mathbf{z}_q) = (1-\beta)\cdot d\left(\mathbf{z}_e, \text{stop\_gradient}\left[\mathbf{z}_q\right]\right) \quad (8)$$
$$+\beta \cdot d\left(\text{stop\_gradient}\left[\mathbf{z}_e\right], \mathbf{z}_q\right) \quad (9)$$

Here, $\beta \in [0,1]$ is a scalar that trades off the importance of updating $\mathbf{z}_e$ and $\mathbf{z}_q$ (*e.g.* large β implies more emphasis on the codebook to adapt towards the encoder). Then for some scalar α that weighs the commitment loss, a differentiable pseudo-objective is minimized:

$$\min_{F,G,h} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}} \left[\mathcal{L}_{\text{task}}\left(G(h(F(\mathbf{x}))), \mathbf{y}\right) + \alpha\cdot\mathcal{L}_{\text{cmt}}(\mathbf{z}_e, \mathbf{z}_q)\right] \quad (10)$$

A good rule of thumb is to set $\alpha = 10$ and $\beta = 0.9$ when using euclidean distance $d_{l_2}(\mathbf{z}_e, \mathbf{z}_q) := \frac{1}{2}\|\mathbf{z}_e - \mathbf{z}_q\|_2^2$ (Van Den Oord et al., 2017).

▷ Updating codebook using EMA
Instead of using the commitment loss, another popular approach is to use exponential moving average (EMA) to train the codebook:

$$\mathbf{z}_q^{(t+1)} \leftarrow (1-\gamma)\cdot \mathbf{z}_q^{(t)} + \gamma \cdot \mathbf{z}_e^{(t)} \tag{11}$$

While EMA is proposed as a training trick in lieu of commitment loss, it is easy to see that it is equivalent to the commitment loss optimized with SGD when $\beta = 1$ (see Appendix A.2); where the EMA decay constant is the learning rate $\gamma = \eta$. This equivalence has been commonly overlooked with the exception of (Łańcucki et al., 2020).

3. On the trainability of VQ networks

It is well known that VQNs perform poorly when the number of actively used codes is small (Kaiser et al., 2018). This is referred to as "index collapse" and is the bottleneck in training VQNs. Thus, there have been abundant efforts to construct algorithms that recover and prevent models from collapsing. We discuss a few popular approaches below:

▷ Stochastic sampling
Roy et al. (2018); Kaiser et al. (2018); Sønderby et al. (2017) proposed to use stochastic sampling and probabilistic relaxation to VQ. Since then, it has become a valid alternative method for training VQNs (Williams et al., 2020; Lee et al., 2022). Taking one of the most recent work as an example, Takida et al. (2022) argues that determinism is the main cause of the codebook collapse and proposes to sample codes from a categorical distribution proportional to the negative distance:

$$p(\mathbf{z}_q = \mathbf{c}_j | \mathbf{z}_e) \propto e^{-d(\mathbf{z}_e - \mathbf{c}_j)/\tau} \tag{12}$$

where τ is a scalar indicating the temperature. The temperature is often annealed to make the model deterministic at convergence. Stochastic sampling can be a bottleneck as it requires computing and storing the full distance matrix. Note that the original idea of sampling to encourage diversity can be rooted back to Kohonen (1990).

▷ Repeated K-means
Łańcucki et al. (2020) explicitly ensures all code-vectors are active by running K-means at every epoch. The naive implementation of repeated K-means forces all codes to be re-initialized. Depending on the noise sensitivity of the decoder, it can lead to large spikes in model performance, where both the encoder and the decoder have to readjust to the newly introduced codes. When decaying the learning rate, the model can no longer adapt to the new codes, and the performance stars to degrade.

Method	K-means		$\mathcal{N}_{\text{kaiming}}$	
	Active	Accuracy	Active	Accuracy
None	81.4	67.0	6.8	59.3
Stochastic	87.8	67.3	16.3	60.8
Rep. KM	99.2	63.7 (67.4)	99.0	62.7 (68.6)
Replace (LRU)	99.8	69.3	99.9	68.6

Table 1: **Collapse prevention:** Comparison of index collapse prevention mechanisms proposed in prior works. We report the classification performance using ResNet18 on ImageNet100 when initializing the code-vectors with K-means and $\mathcal{N}_{\text{kaiming}}$. All models use 1024 codes with vector-size of 512.

▷ Replacement policy
Zeghidour et al. (2021); Dhariwal et al. (2020) proposed to replace the dead codes with a randomly sampled embedding vector. Replacement policies require careful tuning, and we find that using a least-recently-used (LRU) policy with a lifespan of 20 iteration works the best – if the code does not get used for 20 training iterations, it gets replaced with a random embedding vector. When using replacement policies, the active codes are left unchanged, and the overall performance of the model does not degrade.

As shown in Table 1, these aforementioned works result in better performance with an improved number of actively used codes. However, these methods address the symptoms of the collapse by replacing inactive codes rather than resolving why they became inactive in the first place. In this work, we investigate the source of the index collapse by analyzing the optimization dynamics of the codes and how it affects the model. We find that the divergence in the model representation causes the index collapse, and this divergence causes erroneous model updates.

3.1. Commitment loss is an asymmetric loss

VQ layers often diverge throughout training and fail to recover the codes that are no longer actively used (see Appendix A.6). While having good initial conditions, such as an improved initialization scheme (see Appendix A.7), can improve index collapse, ensuring good codebook usage is hard to maintain throughout training. To understand why dead codes are hard to recover from, we need to revisit the commitment loss. One can rewrite commitment loss as an average over distance $d(\cdot)$ computed between an aligned set of points in $\mathcal{P}_\mathbf{z}$ and $\mathcal{C}_\mathbf{z}$:

$$\min_{\mathcal{C}_\mathbf{z}} D(\mathcal{P}_\mathbf{z}, \mathcal{C}_\mathbf{z}) = \frac{1}{|\mathcal{P}_\mathbf{z}|} \sum_{\mathbf{z}_i \in \mathcal{P}_\mathbf{z}} \min_{\mathbf{c}_j \in \mathcal{C}_\mathbf{z}} d(\mathbf{z}_i, \mathbf{c}_j) \tag{13}$$

Here, when $d(\cdot)$ is a Bregman divergence (*e.g.* l_2 used in

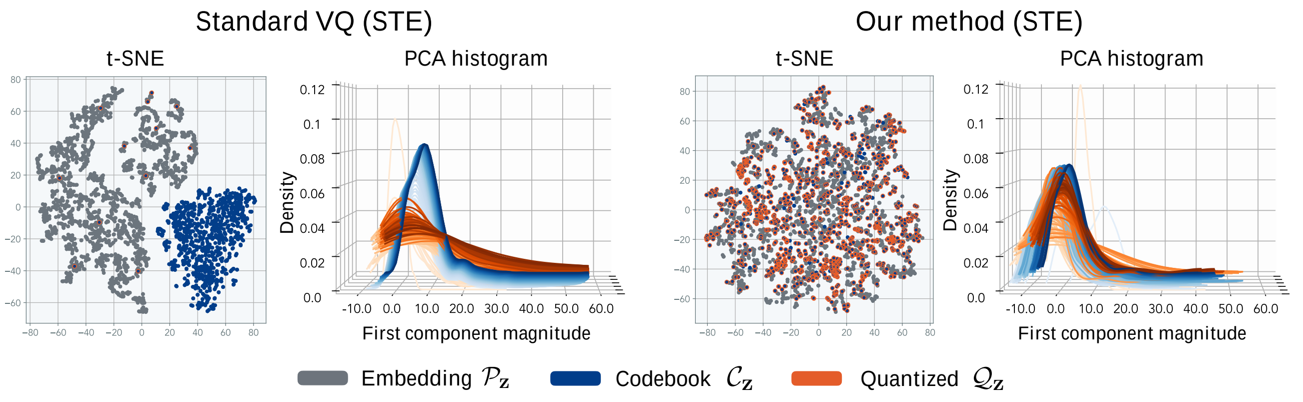

Figure 2: **Divergence vs Accuracy:** We visualize the divergence between $\mathcal{P}_z$, $\mathcal{Q}_z$ and $\mathcal{C}_\mathbf{z}$ on ResNet18 during training. ResNet18 is trained to solve ImageNet100 classification and the codes are initialized using K-means. We sample 2048 embedding the vectors from $\mathcal{P}_\mathbf{z}$ and use the full $\mathcal{C}_\mathbf{z}$ and $\mathcal{Q}_\mathbf{z}$. We embed the vectors associated to the 10th training iteration using tSNE. We also compute distribution shifts throughout training by computing the histogram on the PCA projections. Here the lighter color indicates early iteration in training. The standard approach results in a bifurcation of the codebook. On the right, we show the result of our method using affine re-parameterization, which leads to better distribution matching.

commitment loss is a Bregman divergence), then the distance $D(\mathcal{P}_\mathbf{z}, \mathcal{C}_\mathbf{z})$ can be seen as an average divergence over an aligned set of $\mathcal{P}_\mathbf{z}$ and $\mathcal{C}_\mathbf{z}$ (Banerjee et al., 2005).

This divergence function is non-symmetric and is computed over $\mathcal{P}_\mathbf{z}$ but is minimized with respect to $\mathcal{C}_\mathbf{z}$. Rewriting the commitment loss as an average divergence makes it easier to see why it is susceptible to model collapse. The divergence results in a many-to-one mapping, with the set of selected codes forming $\mathcal{Q}_\mathbf{z}$. Here the disjoint set $\mathcal{C}_\mathbf{z} \setminus \mathcal{Q}_\mathbf{z}$ does not receive any gradient and is not trained. This is analogous to computing reverse-KL in probability measure (Ghosh, 2018), where any sample that falls outside the support of the measure $\mathcal{P}$, does not receive gradients (see Appendix A.10 for further discussion).

Since the subset of codes $\mathcal{Q}$ is used to minimize the commitment loss, and *not* $\mathcal{C}$, $\mathcal{Q}$ learns a "mode-seeking" behavior. This implies that once the codes are not selected, they will likely remain unselected in the future. Note that even if we initialize the code-vectors to overlap in distribution with the embedding perfectly, the code-vectors can be dropped during optimization for various reasons, including stochasticity in training and non-stationary model representation $\mathcal{P}_\mathbf{z}$.

We further demonstrate the impact of the divergence between the codebook and encoder embeddings by visualizing how they diverge in practice. In Figure 2 (left), we train ResNet18 (He et al., 2016) on the ImageNet100 (Russakovsky et al., 2015; Tian et al., 2020) dataset, initialized with K-means, and visualize $\mathcal{P}\mathbf{z}$, $\mathcal{Q}\mathbf{z}$, and $\mathcal{C}_\mathbf{z}$ using dimension reduction methods after several optimization steps. Using t-SNE, we observe that only a few code-vectors are active, with more than 95% of the codes not being selected and trained. We also compute the histogram of the vectors by projecting them into the first PCA component. The histogram shows that the distribution quickly diverges after a few iterations of training. The experiment highlights the vulnerability of VQN optimization, where a sudden shift in the encoder embedding causes severe misalignment. Once misaligned, it often stays misaligned throughout optimization, with few active codes representing the embedding representation. A more desirable outcome can be observed in Figure 2 (right), where the codebook can closely match the embedding distribution. We discuss how to achieve better distribution matching in Section 4.1.

3.2. Gradient estimation gap

When the model embedding diverges from the codebook distribution, the quantization function yields sub-optimal code assignments. This sub-optimal assignment results in a sudden increase in the average quantization error (see Appendix A.6). As VQNs rely on straight-through estimation, the accuracy of the gradient updates in the encoder becomes dependent on the precision of the quantization function.

A good quantization function $h(\cdot)$ is one that can preserve the necessary information of $\mathbf{z}_e$ given a finite set of vectors. The resulting quantization vector can be represented as $\mathbf{z}_q = \mathbf{z}_e + \epsilon$ where ϵ is the residual error vector resulting from the quantization function. When $\epsilon = \mathbf{0}$, there exists no straight-through estimation error, and the model acts as if there is no quantization function. Of course, with a finite set of codes, a lossless quantization function is non-trivial to achieve when training on a large dataset. To measure the gradient deviation from the lossless quantization function, we define the gradient gap as:

$$\Delta_{\text{gap}}(F|h) = \left\| \frac{\partial \mathcal{L}_{\text{task}}(G(\mathbf{z}_e))}{\partial F(\mathbf{x})} - \frac{\partial \mathcal{L}_{\text{task}}(G(\mathbf{z}_e + \epsilon))}{\partial F(\mathbf{x})} \right\| \quad (14)$$

The gradient gap measures the difference between the gradient of the non-quantized model and the quantized model. When $\Delta_{\text{gap}} = \mathbf{0}$, the gradient descent using STE is guaranteed to minimize the loss. When the gap is large, no guarantees can be made. This gradient gap can be made small when (1) the quantization error is small and (2) the decoder function $G(\cdot)$ is smooth. To see this, consider the case when $\mathbf{z}_e$ and $\mathbf{z}_q$ are equivalent ($\epsilon = \mathbf{0}$), then the estimation gap is $\Delta_{\text{gap}} = \mathbf{0}$. When they are not equivalent and $G(\cdot)$ is K-Lipschitz smooth, then the estimation error is proportionately bounded by the quantization error $K \cdot d(\mathbf{z}_e, \mathbf{z}_q)$.

While regularizing for the smoothness of the network is task and model-dependent, the quantization error is a controllable design choice that users can improve upon. One approach is to ensure the quantization error is small at initialization using K-means (Łańcucki et al., 2020; Zeghidour et al., 2021; Karpathy, 2021)(see Appendix A.7). Another approach is to further improve gradient estimates by ensuring the quantization error is small throughout training. This can be done by improving the optimization algorithm itself. In Section 4.2, we propose an improved optimization algorithm to reduce the gradient estimation gap.

The gradient gap provides a measure of the goodness of the STE and is a useful tool for mathematical intuition. However, there is a caveat to be aware of when using it in practice. When VQNs go through index-collapse, there is a sharp spike in Δ_{gap} but eventually, it becomes very trivial for the model to achieve $\Delta_{\text{gap}} = 0$ as the encoder function is encouraged to predict the few remaining codes via the commitment loss. Where with fewer active codes, the easier it becomes to predict. Hence, one should be cautious when using the gradient gap as a measure to compare models.

4. Improved techniques for VQNs

The previous section emphasized contributing factors of index collapse and how minimizing the divergence at initialization leads to improved performance and codebook usage. This section explores methods to reduce codebook divergence and improve optimization.

4.1. Minimizing internal codebook covariate shift with shared affine parameterization

In Section 3, we observed that methods such as resampling resulted in improved performance with a higher number of active codes. We hypothesize that replacement methods work well because the model representation $\mathcal{C}_\mathbf{z}$ eventually ends up resembling the representation of $\mathcal{P}_\mathbf{z}$ by consistently resampling code-vectors, albeit a slow process that requires resampling at almost every iteration. In light of this observation, we propose a more efficient method to match the distribution of $\mathcal{P}_\mathbf{z}$. But first, we describe why misalignment occurs in the first place.

The misalignment between the internal representation of the network's layers is referred to as an internal covariate shift. VQ layers are also prone to this internal covariate shift, where the consistently changing internal representation $\mathcal{P}_\mathbf{z}$ creates a misalignment with the codebook distribution $\mathcal{C}_\mathbf{z}$. We refer to this misalignment as *internal codebook covariate shift*. Generally, internal covariate shifts can be minimized by directly matching the moments of the distributions, or in the case of (Ioffe & Szegedy, 2015), the distributions are whitened to a univariate Gaussian.

Using vector-quantization adds a layer of complexity that compounds with the existing internal covariate shift. While the linear layers receive dense gradients from the objective, the codebook receives sparse gradients. This implies that when the internal representation $\mathcal{P}_\mathbf{z}$ is updated, not only does it require much longer for $\mathcal{C}_\mathbf{z}$ to catch up but also if the update in $\mathcal{P}_\mathbf{z}$ is too large (*e.g.* large learning rates), the assignment can be severely misaligned (see Figure 2 and Appendix A.9).

To reduce gradient sparsity and encourage the codebook to update faster towards the embedding, we propose an affine reparameterization of the code-vector with a shared global mean and standard deviation.

$$\mathbf{c}^{(i)} = \mathbf{c}_{\text{mean}} + \mathbf{c}_{\text{std}} * \mathbf{c}^{(i)}_{\text{signal}} \quad (15)$$

Here $\mathbf{c}^{(i)}_{\text{signal}}$ is the original code-vector, and $\mathbf{c}_{\text{mean}}, \mathbf{c}_{\text{std}}$ are the shared affine parameters with the same $\dim(\mathbf{c}^{(i)}_{\text{signal}})$.

The affine parameters can either be learned through gradient descent or computed via the exponential moving average over $\mathbf{z}_e$ and $\mathbf{z}_q$ statistics (see Appendix A.12). Note, that under the Gaussian assumption on $\mathcal{P}_\mathbf{z}$ and $\mathcal{C}_\mathbf{z}$, matching the moments is equivalent to minimizing the KL divergence (Kurz et al., 2016). The reparameterization allows gradients to flow through the unselected code-vectors through the affine parameters. Although we make no specific Gaussian assumption on the parameterization, it is easy to extend our method to better capture complex distributions by assigning distinct affine parameters to each codebook subset.

4.2. Alternated optimization

In Section 3.2, we showed that the error in the gradient update is proportional to the quantization error. Hence, we are interested in ensuring that divergence stays well-behaved during optimization. Here it is important to note that when there is an index collapse, the model trivially achieves zero

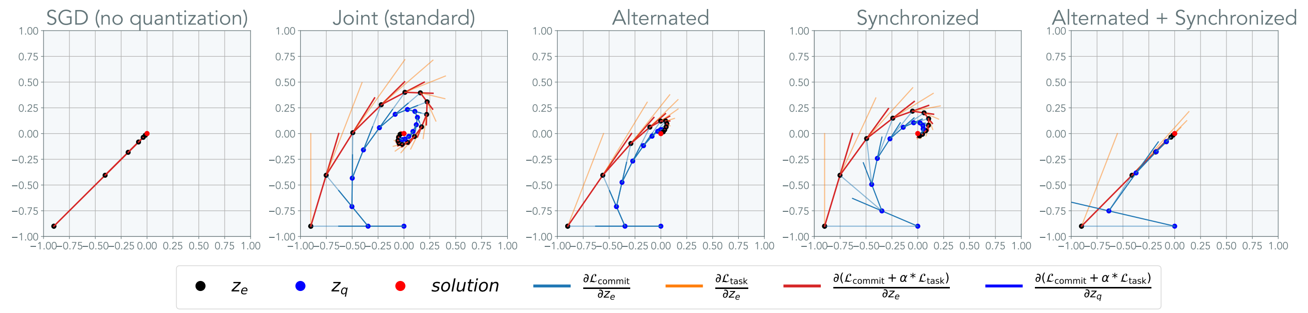

Figure 3: **Codebook update dynamics on toy experiment:** Optimization dynamics of vector-quantization on a toy setup. The experiment above uses a single code vector with a stationary target for $\mathbf{z}_e$ (red). A euclidean loss is computed with respect to the code-vector $\mathbf{z}_q$ (blue), and the resulting gradient is used to update the embedding $\mathbf{z}_e$ using the straight-through approximation (black). A commitment loss is applied to the $\mathbf{z}_q$ and $\mathbf{z}_e$ using l_2 distance. All methods are optimized using SGD with the same fixed learning rate of 0.1. NoVQ optimizes the embedding without the quantization function. Joint optimizes the embedding with $\mathcal{L}_{\text{task}}$ and $\mathcal{L}_{\text{commit}}$ together. Alternated uses first optimizes the codebook assignment and then optimizes the model with the task loss, with single iteration for each step. Lookahead objective predicts the trajectory of $\mathbf{z}_e$ and updates $\mathbf{z}_q$ towards it. The trajectory creates a large spiral for standard VQ training due to the straight-through approximation error. The alternated optimization minimizes this approximation error, reducing the extent of the spiral. Note that the approximation error is also caused by the code-vector representation, which is a historical moving average of the embedding with a delay. The lookahead optimizer reuses the gradient from $\mathcal{L}_{\text{task}}$ to better synchronize the code-vector representation and accelerate convergence.

gradient error as the commitment loss is easily minimized. This is not the setting we are interested in, and we assume that we are operating on a well-behaved regime.

For any arbitrary task $\mathcal{L}_{\text{task}}$, the underlying objective of a VQN is to minimize the empirical loss while learning a good codebook representation.

$$\min_{F,G,h} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}_{\text{train}}} \left[\mathcal{L}_{\text{task}}\left(G(h(F(\mathbf{x}))), \mathbf{y}\right)\right] \quad (16)$$

The objective function above is not continuously differentiable; therefore, Eqn. 10 is used as a surrogate objective. However, the gradient computed from the surrogate objective is a biased estimate of the true gradient and can result in undesirable optimization dynamics. This is illustrated in Figure 3 with a toy setup. The dynamical error induced from the optimization can be traced to the straight-through estimation, in which the surrogate gradient deviates from the true gradient proportional to the quantization error. Hence, updating the network when the quantization error is big can lead to an erroneous model update. To reduce the quantization error, we propose an alternating optimization algorithm:

$$\min_{h} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}} \left[\mathcal{L}_{\text{commit}}\left(h(F(\mathbf{x})), F(\mathbf{x})\right)\right] \quad (17)$$

$$\min_{F,G} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}} \left[\mathcal{L}_{\text{task}}\left(G(h(F(\mathbf{x}))), \mathbf{y}\right)\right] \quad (18)$$

The algorithm above resembles that of online K-means with a non-linear encoder and decoder. Where Eqn. 17 optimizes the K-mean clusters, and Eqn. 18 optimizes the model given the new cluster assignment. We know that when $\mathcal{L}_{\text{commit}} \to 0$, $h(\cdot)$ acts as an identity function under stationary F and G. Then, under fixed h, both F and G can be optimized with close to zero estimation error. This can be repeated until the small quantization error assumption is broken.

Of course, optimizing the inner term till convergence is computationally expensive. Fortunately, we find that it is not necessary to wait till convergence (see Appendix A.11). In practice, alternating the inner term even for a single iteration yields good performance. In the toy setting of Figure 3, we visualize how the alternating optimization performs when using a single update for each term.

4.3. One-step behind or in synchronous step?

The codebook updated with the commitment loss is a historical average of the model representation. Writing out the gradient update for commitment loss for $\beta = 1$:

$$\mathbf{z}_q^{(t+1)} \leftarrow (1-\eta)\cdot \mathbf{z}_q^{(t)} + \eta \cdot \mathbf{z}_e^{(t)} \quad (19)$$

The historical average does not account for the current representation but only up to the previous representation. Therefore, computing the gradient with respect to the historical average implies that the model receives a “delayed” gradient. To reduce the delay in the $\mathbf{z}_q$ representation, we desire the code-vector to include a running average of the most recent representation:

$$\mathbf{z}_q^{(t+1)} \leftarrow (1-\eta)\cdot \mathbf{z}_q^{(t)} + \eta \cdot \mathbf{z}_e^{(t+1)} \quad (20)$$

The above equation is tractably computable as the gradient of $\mathbf{z}_q$ is used to update $\mathbf{z}_e$. Hence, an explicit equation for the *synchronized update rule* is:

$$\mathbf{z}_q^{(t+1)} \leftarrow \underbrace{(1-\eta)\cdot \mathbf{z}_q^{(t)} + \eta \cdot \mathbf{z}_e^{(t)}}_{\text{original commitment loss}} + \eta^2 \cdot \frac{\partial \mathcal{L}_{\text{task}}}{\partial \mathbf{z}_q} \tag{21}$$

Using the equation above, the code-vectors take a step in the direction of the encoder representation using the gradient of the task loss. In python, this requires a minor change to the existing implementation of straight-through estimation:

```
z_q = z + (z_q − z).detach() + ν ∗ (z_q − (z_q).detach())
```

Where ν is a scalar to decide whether we want a pessimistic or an optimistic update. We find that the effectiveness of ν depends on the model architecture.

5. Results

5.1. Classification

We apply our methods to ImageNet100 (Tian et al., 2020) classification. ImageNet100 is a subset of the ImageNet-1K dataset (Russakovsky et al., 2015), consisting of 100 classes and approximately 100,000 images. The training details are provided in the Appendix A.1.

Our results, presented in Table 2, demonstrate the effectiveness of our proposed methods. All models were initialized using the K-means clustering algorithm. We also report the perplexity of the model on the test dataset, which is defined as $2^{H(p)}$, where $H(p)$ is the entropy over the codebook likelihood. A higher perplexity implies a uniform assignment of codes. While having very low perplexity is associated with index collapse, having more does not necessarily imply better performance – having a high perplexity on a task that has more codes than necessary indicates redundancy. Our results using affine re-parameterization largely improves index-collapse, and the use of synchronized and alternating training methods further improves the overall performance of the model. We further compare our results against the use of the least-recently-used (LRU) replacement policy and observed our method to outperform or perform comparably. A combination of all these methods results in the largest improvement in performance. We observed using l_2 normalization to hurt performance on classification. We suspect that removing the magnitude component of the embedding hurts models that use magnitude-sensitive objectives (*e.g.* soft-max cross-entropy loss).

	Affine	Sync.	Alt.	Replace	Accuracy ↑	Perplexity ↑
AlexNet	-	-	-	-	47.2	11.2
	✓	-	-	-	55.7 (+8.5)	648.1
	-	✓	-	-	49.2 (+2.0)	10.7
	-	-	✓	-	52.8 (+5.6)	65.7
	-	-	-	✓	54.4 (+7.2)	657.6
	✓	✓	✓	✓	**57.9 (+10.7)**	**753.8**
ResNet18	-	-	-	-	64.1	32.4
	✓	-	-	-	70.4 (+6.3)	300.0
	-	✓	-	-	66.0 (+1.9)	60.1
	-	-	✓	-	67.5 (+3.4)	54.2
	-	-	-	✓	68.1 (+4.0)	**334.5**
	✓	✓	✓	✓	**71.0 (+6.9)**	306.6
ViT(T)	-	-	-	-	48.6	112.9
	✓	-	-	-	52.8 (+4.2)	299.0
	-	✓	-	-	51.3 (+2.7)	88.8
	-	-	✓	-	53.1 (+4.5)	95.6
	-	-	-	✓	54.4 (+5.8)	**598.8**
	✓	✓	✓	✓	**56.7 (+8.1)**	592.2

Table 2: **Classification:** The effect of how our methods affect the final performance on classification. All methods are initialized with K-means.

5.2. Generative modeling

We further apply our method to CelebA (Liu et al., 2015) and CIFAR10 (Krizhevsky et al., 2009) generative modeling tasks. We adopt the training framework from previous works, and the details can be found in the appendix (Appendix A.1). We compare the performance of our method against existing techniques such as VQVAE (Van Den Oord et al., 2017), SQVAE (Takida et al., 2022), and Gumbel-VQVAE (Karpathy, 2021; Esser et al., 2020) using MSE as well as LPIPS perceptual loss (Zhang et al., 2018). SQVAE requires $4\times$ more memory footprint than all other methods as it requires storing the full distance matrix along with the computation graph. Both baselines using l_2 normalization and least-recently-used (LRU) replacement policy largely improve training stability and reconstruction performance of generative models. When these methods are applied jointly with ours, we observe the best improvement.

In Figure 4, we run generative modeling using MaskGIT (Chang et al., 2022). We plot the rFID (reconstruction-FID) (Takida et al., 2022) and the FID (Heusel et al., 2017) during training. rFID measures the FID on the reconstructed images from the auto-encoder over the test set. We do not use perceptual or discriminator loss to train the network, and for rFID/FID, we use 5000 generated samples.

5.3. Warmup and normalization can be helpful

One way to mitigate the divergence between the codebook and the embedding distribution is by constraining the representation to be within a bounded measure space. This limits the range of movement of the embedding distribution, facilitating alignment between the codebook and the embedding distribution throughout training. Common techniques for this include l_2 normalization (Yu et al., 2022), batch-normalization (Łańcucki et al., 2020), and assuming a restricted distribution (Takida et al., 2022) in probabilistic VQNs. However, these techniques improve stability at the cost of reduced model expressivity. Alternatively, one can ensure the updates of $\mathcal{P}_{\mathbf{z}}$ to be small in order for the code-

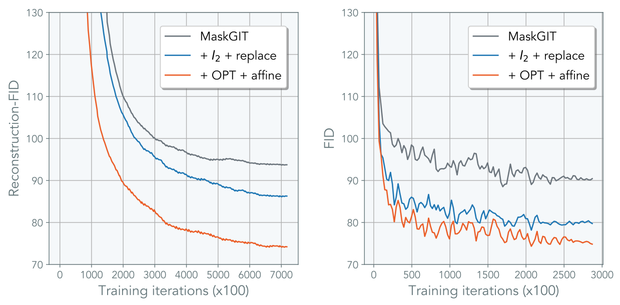

Figure 4: **MaskGIT FID training curves:** MaskGIT (Chang et al., 2022) trained on CelebA with only reconstruction loss. We report rFID (left) and FID (right) training curves. We use a slimmed-down version of MaskGIT: VQGAN using 32 channels instead of 128 and transformer using 8 blocks instead of 24.

	Method	MSE (10^{-3}) ↓	Perplexity ↑	LPIPS (10^{-1}) ↓
CIFAR10	VQVAE	5.65	14.0	5.43
	VQVAE + l_2	3.21	57.0	3.64
	VQVAE + replace	4.07	109.8	4.30
	VQVAE + l_2 + replace	3.24	115.6	3.56
	SQVAE	3.36	**769.3**	3.99
	Gumbel-VQVAE	6.16	20.3	5.73
	VQVAE + Affine	5.15	69.5	5.18
	VQVAE + OPT	4.73	15.5	4.82
	VQVAE + Affine + OPT	4.00	79.3	4.36
	VQVAE + Affine + OPT + replace	1.81	290.9	2.56
	VQVAE + Affine + OPT + replace + l_2	**1.74**	608.6	**2.27**
CELEBA	VQVAE	10.02	16.2	2.71
	VQVAE + l_2	6.49	188.7	1.82
	VQVAE + replace	4.77	676.4	1.55
	VQVAE + l_2 + replace	4.93	861.7	1.47
	SQVAE	9.17	769.1	2.68
	Gumbel-VQVAE	7.34	96.7	2.30
	VQVAE + Affine	7.47	112.6	2.22
	VQVAE + OPT	7.78	30.5	2.25
	VQVAE + Affine + OPT	6.60	186.6	1.82
	VQVAE + Affine + OPT + replace	**3.84**	650.4	**1.35**
	VQVAE + Affine + OPT + replace + l_2	4.42	**872.6**	1.36

Table 3: **Generative modeling reconstruction:** Comparison between various methods on image reconstruction task. All methods use the same base architecture used in (Takida et al., 2022). The metrics are computed on the test set and hyper-parameters are tuned for each model.

book to catch up. To do so without hindering convergence speed, we find a learning rate scheduler with warmup to work very well. In Appendix A.4, we show how using cosine learning rate decay with linear warmup (Loshchilov & Hutter, 2017; Goyal et al., 2017) improves both the model performance and model perplexity.

5.4. Ablation on alternating optimization

In Appendix A.11, we measure how varying the number of inner and outer loop iterations affects classification performance. We find that by increasing the number of inner loop iterations by 8, we observed an 11.09% improvement over the baseline and 5.51% over the version that uses a single iteration of the inner step. On the other hand, we do not find increasing the outer loop to help. When combining all our methods, we observed that setting the inner loop iteration to 1-2 suffice. In this experiment, we evenly divide the training mini-batch into sub-mini-batches for each iteration of the expectation and maximization steps. This ensures that the number of images the model observes is equal across all experiments. Furthermore, when choosing to only optimize h for a single iteration, it is possible to update both the inner and outer term in a single forward pass, as the commitment loss $\mathcal{L}_{\text{commit}}$ does not depend on the task loss $\mathcal{L}_{\text{task}}$. As a result, the computational overhead for a single fused pass is $1.05\times$, and $2\times$ with four inner loop iterations (For smaller models like AlexNet, the overhead is less, roughly $1.5\times$).

5.5. Further reducing sparsity in VQNs

Irrespective of the initial alignment between the codebook $\mathcal{C}_z$ and the embedding distribution $\mathcal{P}_z$, a certain degree of divergence between these distributions is inevitable. This is particularly true for loss functions that are either unbounded (e.g., hinge loss) or have non-saturating gradients (e.g., logistic/exponential losses), where the weights grow inversely proportional to the loss. This effect is more pronounced in networks that lack normalization. Despite the use of affine reparameterization of the code-vectors, achieving 100% utilization is non-trivial. To further reduce the sparsity in the codebook update, one can directly improve the architectural design choices that contribute to sparsity. Specifically, factors such as image size, batch size, and the number of pooling layers have a significant effect on VQN performance, as the number of code-vector selections directly depends on these variables. In Appendix A.5, we demonstrate that there is a significant degradation in performance when reducing the image size from 256×256 to 128×128, resulting in an over 20% reduction in performance. This indicates the importance of training design choices is VQN.

6. Conclusion

Discretization has played a significant role in many fields, such as analog-to-digital communication and modern computing. Once representations are made discrete, various techniques from information theory can be applied to manipulate them for benefits such as compression, error correction, and robustness. Discrete representations can also be broken down into independent parts, allowing for the development of composable symbolic representations. In this work, we proposed a set of techniques that addresses several known challenges in optimizing vector-quantized models. Through our proposed methods, we were able to demonstrate improved model performance. While symbolic representation learning is still in its early stages, our optimization techniques provide insight for designing better models in the future.

7. Acknowledgement

We want to thank our lab members for their helpful feedback. Minyoung Huh was funded by ONR MURI grant N00014-22-1-2740. Brian Cheung is supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216. Minyoung would like to further thank Wei-Chiu Ma, Lucy Chai, and Eunice Lee.

References

Asano, Y. M., Rupprecht, C., and Vedaldi, A. Self-labelling via simultaneous clustering and representation learning. In *International Conference on Learning Representations*, 2020.

Banerjee, A., Merugu, S., Dhillon, I. S., Ghosh, J., and Lafferty, J. Clustering with bregman divergences. *Journal of machine learning research*, 6(10), 2005.

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

Caron, M., Bojanowski, P., Joulin, A., and Douze, M. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 132–149, 2018.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems*, 33:9912–9924, 2020.

Chang, H., Zhang, H., Jiang, L., Liu, C., and Freeman, W. T. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11315–11325, 2022.

Chang, H., Zhang, H., Barber, J., Maschinot, A., Lezama, J., Jiang, L., Yang, M.-H., Murphy, K., Freeman, W. T., Rubinstein, M., et al. Muse: Text-to-image generation via masked generative transformers. *arXiv preprint arXiv:2301.00704*, 2023.

Chen, X., Hsieh, C.-J., and Gong, B. When vision transformers outperform resnets without pre-training or strong data augmentations. In *International Conference on Learning Representations*, 2022.

Chung, Y.-A., Tang, H., and Glass, J. Vector-quantized autoregressive predictive coding. In *Interspeech*, 2020.

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

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Ghosh, D. Kl divergence for machine learning. `https://dibyaghosh.com/blog/probability/kldivergence.html`, 2018.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Gray, R. Vector quantization. *IEEE Assp Magazine*, 1(2): 4–29, 1984.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

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 *International Conference on Machine Learning*, pp. 2390–2399. PMLR, 2018.

Karpathy, A. deep-vector-quantization. `https://github.com/karpathy/deep-vector-quantization`, 2021.

Kohonen, T. Improved versions of learning vector quantization. In *1990 ijcnn international joint conference on Neural networks*, pp. 545–550. IEEE, 1990.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kurz, G., Pfaff, F., and Hanebeck, U. D. Kullback-leibler divergence and moment matching for hyperspherical probability distributions. In *2016 19th International Conference on Information Fusion (FUSION)*, pp. 2087–2094. IEEE, 2016.

Łańcucki, A., Chorowski, J., Sanchez, G., Marxer, R., Chen, N., Dolfing, H. J., Khurana, S., Alumäe, T., and Laurent, A. Robust training of vector quantized bottleneck models. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. IEEE, 2020.

Lee, D., Kim, C., Kim, S., Cho, M., and Han, W.-S. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2022.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Ozair, S., Li, Y., Razavi, A., Antonoglou, I., Van Den Oord, A., and Vinyals, O. Vector quantized models for planning. In *International Conference on Machine Learning*, pp. 8302–8313. PMLR, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pp. 8821–8831. PMLR, 2021.

Roy, A., Vaswani, A., Neelakantan, A., and Parmar, N. Theory and experiments on vector quantized autoencoders. *arXiv preprint arXiv:1805.11063*, 2018.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 2015.

Sønderby, C. K., Poole, B., and Mnih, A. Continuous relaxation training of discrete latent variable image models. In *Beysian DeepLearning workshop, NIPS*, volume 201, 2017.

Takida, Y., Shibuya, T., Liao, W., Lai, C.-H., Ohmura, J., Uesaka, T., Murata, N., Takahashi, S., Kumakura, T., and Mitsufuji, Y. Sq-vae: Variational bayes on discrete representation with self-annealed stochastic quantization. *arXiv preprint arXiv:2205.07547*, 2022.

Tian, Y., Krishnan, D., and Isola, P. Contrastive multiview coding. In *European conference on computer vision*, pp. 776–794. Springer, 2020.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Williams, W., Ringer, S., Ash, T., MacLeod, D., Dougherty, J., and Hughes, J. Hierarchical quantized autoencoders. *Advances in Neural Information Processing Systems*, 33: 4524–4535, 2020.

Yan, W., Zhang, Y., Abbeel, P., and Srinivas, A. Videogpt: Video generation using vq-vae and transformers. *arXiv preprint arXiv:2104.10157*, 2021.

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 *International Conference on Learning Representations*, 2022.

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.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

A. Appendix

A.1. Training details

▷ VQ configuration and implementation

We use 1024 codes for all our experiments. $1024 \sim 4096$ is the typical codebook size used in prior works (Esser et al., 2020; Yan et al., 2021). Using 4096 codes does improve the performance slightly. We do not apply weight decay on the codebooks. For VQ hyper-parameters, we use $\alpha = 5$ and $\beta = 0.9 \sim 0.995$ with mean-squared error for the commitment loss. The performance starts to degrade outside this range of β. Note that using higher $\beta = 1.0$ is equivalent to the EMA update. We implemented our own VQ algorithm to improve the run-time and memory efficiency. Standard implementation does not fit in memory for ImageNet training on standard commercial GPUs as the number of vectors grows close to 1 million for a given mini-batch. To mitigate this, we implemented our VQ algorithm using divide-and-conquer, which runs significantly faster and is more memory efficient compared to the naive implementation that allocates a contiguous memory to compute pair-wise distance. At a high level, our algorithm divides the batch of vectors into smaller chunks. For each chunk, batch matrix multiply (cdist in PyTorch) is applied for each subset and top-K reduction operation simultaneously. This is implemented in Python, and there is room for additional speed-up by implementing it in CUDA and parallelization via vmap. To further reduce computational and memory footprint, the distance is computed in half-precision, and the resulting code index is used to query the vector from float precision. Doing so adds no numerical imprecision to the existing model.

For affine-reparameterization, we use the variant with learnable affine parameters, which we find to be easier to implement and more stable in practice. We find that the optimal learning rate scale for the affine parameters needs to be tuned based on the model architecture as the norm of the magnitudes grows differently for each model during training. For learnable affine parameters, this can be easily implemented in Python via:

$$\texttt{scale} = 1 + \texttt{aff\_lr\_scale} * \texttt{codebook\_scale}$$
$$\texttt{bias} = \texttt{aff\_lr\_scale} * \texttt{codebook\_bias}$$
$$\texttt{codebook} = \texttt{scale} * \texttt{codebook} + \texttt{bias}$$

To be robust to the affine-parameter learning rate scale, we recommend using norm constraint (e.g., max norm constraint, norm clipping) or placing the VQ layer after an explicit normalization layer.

▷ Classification

For AlexNet and ResNet18, we follow the design choices in https://github.com/pytorch/examples/blob/main/imagenet. The training configuration can be found in Table 4 and Table 5. For ViT(T), we started with the hyper-parameters recommended by (He et al., 2022) and re-tuned the hyper-parameters on the baseline VQ model. For ViT model codebase, we use the official PyTorch TorchiVison repository https://github.com/pytorch/vision/blob/main/torchvision/models/vision_transformer.py. The ViT(T) configuration is from https://github.com/rwightman/pytorch-image-models/blob/main/timm/models/vision_transformer.py. The architecture configuration is shown in Table 7 and the training configuration is shown in Table 6. ViT on ImageNet100 performs worse than ResNet18 as ViT does not perform very well on small datasets. This is a well-known observation (Chen et al., 2022). We apply data augmentation to the original image resolution and resize them to 224×224 for training.

▷ AlexNet quantization

For AlexNet, we quantize the features after the convolutional layers and before the fully-connected layers.

config	value
optimizer	SGD with momentum
base learning rate	0.01
weight decay	1e-4
optimizer momentum	0.9
training epochs	90
learning rate scheduler	step
step epochs	30, 60
augmentations	RandomResizedCrop
$\mathcal{L}_{\text{cmt}}$ weight α	5
$\mathcal{L}_{\text{cmt}}$ tradeoff β	0.95
sync ν	2
affine lr scale	10

Table 4: AlexNet image classification training configuration.

▷ ResNet18 quantization

For ResNet18, we quantize after the second macroblock. This is after `layer2` in the TorchVision repository. This is roughly the halfway point in the ResNet18. We found this model architecture to be insensitive ν, possibly due to batch-normalization.

▷ ViT quanitzation

ViT does not directly operate on image pixels but on image patches. Hence, the quantization cannot be applied to pixels. For our experimental setup, we tokenize non-overlapping patches of size 16×16 resulting in a total of $14 \times 14 = 196$ input tokens. These individual tokens are quantized. Note that this is often much fewer than the number of embedding vectors used in CNNs (*e.g.* feature embedding of $32 \times 32 = 1024$ embedding vectors) and may be the reason why they suffer more from vector-quantization. We apply the quantization after the 6th transformer block. Train-

config	value
optimizer	SGD
base learning rate	0.1
weight decay	1e-4
optimizer momentum	0.9
training epochs	90
learning rate scheduler	step
step epochs	30, 60
augmentations	RandomResizedCrop
$\mathcal{L}_{\texttt{cmt}}$ weight α	5
$\mathcal{L}_{\texttt{cmt}}$ tradeoff β	0.98
sync ν	0.2
affine lr scale	1

Table 5: ResNet18 image classification training configuration.

ing ViT without replacement policy is extremely finicky, which requires careful hyper-parameter tuning. When using replacmenet policy, it becomes more robust to wide-range of hyper-parameters. We recommend re-tuning the configuration when using it jointly with replacement policy.

config	value
optimizer	AdamW
base learning rate	2e-4
weight decay	0.03
optimizer momentum	$(0.9, 0.95)$
training epochs	90
learning rate scheduler	Cosine
warmup epochs	10
augmentations	RandomResizedCrop
$\mathcal{L}_{\texttt{cmt}}$ weight α	5
$\mathcal{L}_{\texttt{cmt}}$ tradeoff β	0.995
sync ν	0.01
affine lr scale	10

Table 6: ViT-Tiny image classification training configuration.

config	value
patch size	16×16
embedding dim	192
depth	12
num heads	3
feedforward dim	768
activation	GELU

Table 7: ViT-Tiny model architecture.

▷ **Generative modeling**
For generative modeling, we use backbone architecture from (Takida et al., 2022) with 64 channels. The training configuration is listed in Table 8. We use MSE for reconstruction loss, and we do not use any perceptual or discriminative loss. For CIFAR10, we use an image size of 32×32, and for CelebA, we use an image size of 128×128. For MaskGIT (Chang et al., 2022), we followed the author's original codebase and reimplemented it in PyTorch. To ensure that we can fit the model in a commercial GPU (24GB), we reduced the number of channels by 1/4th (32 channels) for the auto-encoder and used an 8-layer transformer instead of 24. The code resolution factor is 8, resulting in an 8×8 code map. A transformer is trained on the vectorized code. See (Chang et al., 2022) for the method details.

config	value
optimizer	AdamW
base learning rate	1e-4
weight decay	1e-4
optimizer momentum	$(0.9, 0.95)$
training epochs	90
learning rate scheduler	Cosine
warmup epochs	10
augmentations	None

Table 8: Generative modeling, auto-encoder configuration for CIFAR10 and CelebA.

▷ **Baseline**
For **SQVAE** (Takida et al., 2022), we use the Gaussian-SQVAE (4th-variant), which is the best-performing variant for unconditional generative modeling that uses a diagonal covariance matrix. We used the architecture from the official codebase and replicated their stochastic quantization layer to interface our framework. We anneal the temperature till convergence with a geometric decay. We re-tuned the learning rate and the weighting of the loss. We find the default scaling of $1 \sim 0.1$ to work, setting it any higher did not train. SQVAE has an entropy term that encourages diversity in the codebook along with the ELBO objective. We observed that the model can easily collapse if the temperature is annealed too fast and, therefore, requires much longer to converge.

Gumbel-VQ was a variation proposed in the public repository by (Karpathy, 2021) and was also been used by (Esser et al., 2020) in their codebase. Similar to (Takida et al., 2022), Gumbel-VQ minimizes the ELBO; however, unlike standard VQ methods, which compute the distance across all code vectors, Gumbel-VQ predicts a distribution over the code without making any explicit comparisons. The model then uses the Gumbel-softmax (Jang et al., 2017) trick to sample from the distribution. We tried various hyper-parameters for the ELBO loss weight {5e-1, 5e-2, 5e-3, 5e-4, 5e-5} and found 5e-3 to work the best.

A.2. EMA and commitment loss

We show the equivalence between EMA and the commitment loss. Let $\beta = 1$ for the commitment loss with MSE for d:

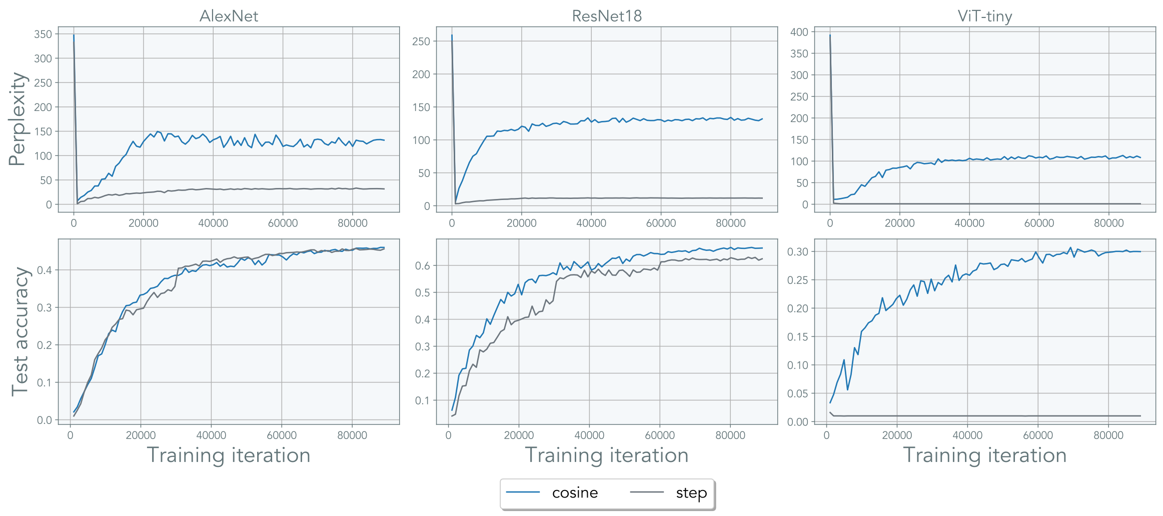

Figure 5: **Warmup improves perplexity:** We plot the perplexity and test accuracy for methods trained with and without a linear warmup. All methods were initialized with K-means.

$$\mathcal{L}_{\text{vq}}(\mathbf{z}_e^{(t)}, \mathbf{z}_q^{(t)}) = \frac{1}{2}\|\mathbf{z}_e^{(t)} - \mathbf{z}_q^{(t)}\|_2^2 \tag{22}$$

The gradient of the commitment loss is computed with respect to the codes $\mathbf{z}_q$ is:

$$\frac{\partial \mathcal{L}_{\text{vq}}(\mathbf{z}_e^{(t)}, \mathbf{z}_q^{(t)})}{\partial \mathbf{z}_q^{(t)}} = \frac{1}{2}\|\mathbf{z}_e^{(t)} - \mathbf{z}_q^{(t)}\|_2^2 \tag{23}$$

$$= \mathbf{z}_e^{(t)} - \mathbf{z}_q^{(t)} \tag{24}$$

$$\tag{25}$$

Then the update for $\mathbf{z}_q^{(t+1)}$ is:

$$\mathbf{z}_q^{(t+1)} = \mathbf{z}_q^{(t)} - \eta \frac{\partial \mathcal{L}_{\text{vq}}(\mathbf{z}_e^{(t)}, \mathbf{z}_q^{(t)})}{\partial \mathbf{z}_q^{(t)}} \tag{26}$$

$$= \mathbf{z}_q^{(t)} - \eta \cdot (\mathbf{z}_e^{(t)} - \mathbf{z}_q^{(t)}) \tag{27}$$

$$= (1-\eta) \cdot \mathbf{z}_e^{(t)} + \eta \cdot \mathbf{z}_q^{(t)} \tag{28}$$

Letting $\eta = \gamma$ and using SGD to optimize the code-vectors, we recover the EMA update rule in Eqn. 11

A.3. Gradient estimation gap

To compute the gradient estimation gap in Eqn. 14, we compute 2 forward passes. Once with the quantization function and once without. Let $g^{(l)}$ be the gradient without the quantization function for layer $l \in L$ and $\hat{g}^{(l)}$ be the gradient gap computed without the quantization error. Then the total average gradient error is $\sum_{l \in L} \|g^{(l)} - \hat{g}^{(l)}\|_2^2$. We visualize the gradient error of VQVAE training for the baseline model and our model that uses alternated optimization Figure 6.

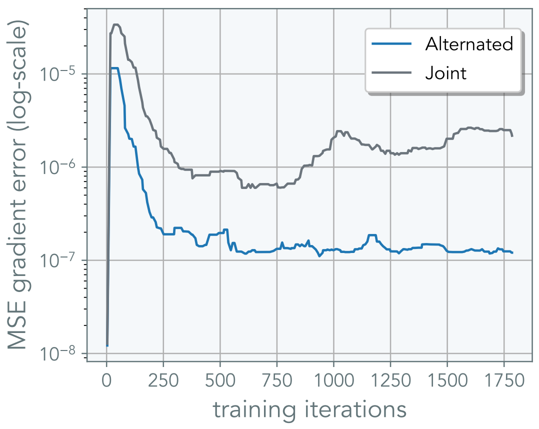

Figure 6: **Gradient estimation gap:** STE estimation gap comparing standard joint optimization versus alternated optimization. The y-axis is log-scale.

The VQ-layers use l_2 normalization, and the gradient gap is computed in log-scale – hence the gradient gap is much larger than it appears in the figure.

A.4. Warmup improves perplexity and performance

As mentioned in Section 5.3, we found using a warmup to significantly improve codebook perplexity. This, in turn, results in better performance. In Figure 5, we compare VQNs trained with and without warmup. All methods are initialized using K-means. The baseline model uses a step scheduler. For ViT, the model collapses when we do not use a linear warmup. We hypothesize that a small learning rate allows the code-vectors to catch up to the model representa-

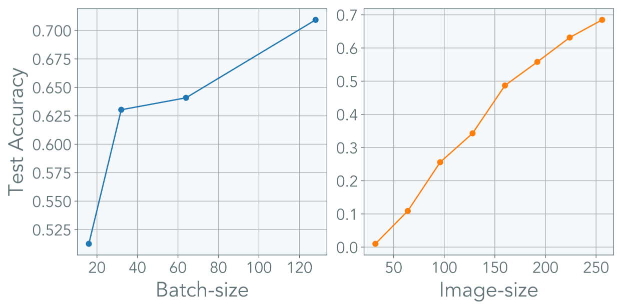

Figure 7: **How activation rate affects performance:** The performance of VQNs is tied to the selection likelihood of the code-vectors $p_{\textsf{activate}}$. Here we visualize how the batch-size and the image-size affects the model performance. VQNs that have a low activation ratio perform signifcantly worse than standard networks. On the right, we convert the x-axis into $p_{\textsf{activate}}$ using Eqn. A.5

.

tion, while using a large learning rate at initialization causes misalignment between code-vectors and embedding vectors, resulting in index collapse.

A.5. Sensitivity of VQ models

Vector-quantization using the standard commitment loss result in sparse gradients by design. Therefore, it is imperative to isolate design factors that would contribute to this sparsity. Since sparsity is directly correlated with the selection rate of the code vectors, we can write out the selection probability under a simplifying assumption of i.i.d selection rate. For a VQN that operates on images, the likelihood that a code-vector $\mathbf{c}_i$ will activate at-least k times is:

$$p(\mathbf{c}_i \text{ activate atleast } k \text{ times}) = \\ 1 - \sum_{j=0}^{k-1} \binom{bhw/2^{n_{\textsf{pool}}}}{j} \left(\frac{1}{\mathcal{C}}\right)^j \left(1 - \frac{1}{\mathcal{C}}\right)^{bhw/2^{n_{\textsf{pool}}}-j} \quad (29)$$

Where h and w are the image dimensions and $n_{\textsf{pool}}$ is the number of 2×2 pooling layers, and b is the batch size. The equation above is simply a summation of the binomial distribution over the selection probability. Here it becomes apparent that the image size, the batch size, the codebook size, and the number of pooling layers all directly affect the selection likelihood. While it is impossible to study the effect of these factors in perfect isolation, in Figure 7, we show how performance degrades compared to the non-quantized network.

A.6. Index Collapse

Index collapse is a phenomenon associated with the under-utilization of the codebook. While code-vectors become inactive throughout training, it is common to see a sudden collapse in code-vector usage early in training. In Figure 9, we visualize the index collapse on ResNet18 trained with K-means initialization. At initialization, all codes are actively used, with the gradient gap and divergence being close to 0. Here the divergence is measured between $D(\mathcal{C}_{\mathbf{z}}, \mathcal{Q}_{\mathbf{z}})$, which measures the bifurcation of the codebook distribution. After a single iteration of SGD update, the code-vector starts to diverge, with the number of active codes dropping below 50%. The resulting misalignment causes a large spike in the gradient error gap. The divergence continues to grow with the number of active codes approaching 1% utilization. With fewer active codes, the encoder learns a degenerate solution of predicting these few remaining code-vectors.

	Method	FID ↓
CELEBA	MaskGIT*	90.4
	+ l_2	81.5
	+ replace	79.7
	+ Affine + OPT (ours)	**74.8**

Table 9: **MaskGIT generation:** FID on CelebA image generation using MaskGIT. *We use a significantly smaller capacity architecture to make the training feasible.

A.7. The effect of VQ initialization

To improve gradient estimates, one can ensure the quantization error is small. To do so, one can ensure that the quantization error is small at initialization by choosing an appropriate weight initialization scheme. Unfortunately, Initializing the codebooks requires a priori knowledge of the model architecture and data distribution, as the distribution associated with the input embedding $\mathbf{z}_e$ is hard to calculate ahead of time nor may not have a closed form distribution to sample from. For example, a VQ layer placed after a ReLU function implies that code-vectors in the negative half-space will never be sampled.

To mitigate this issue, it is a common practice to use data-dependent initialization such as K-means. These methods precisely capture the distribution of $\mathbf{z}_e$ and better distributes the likelihood individual codebooks will be sampled. In fig X, we show the relationship between quantization error at initialization and the final performance of the model. Moreover, we find that the quantization error at initialization is a strong indicator for index-collapse / under-utilization of code-vectors – a phenomenon in which the number of active code-vectors at convergence is significantly smaller than the one we started with. As shown in Figure 8, a good initialization scheme mitigates index-collapse and leads to favorable performance.

A.8. Generation with MaskGIT

We extend our results on generative modeling using MaskGIT (Chang et al., 2022) on CelebA. In Table 9, the

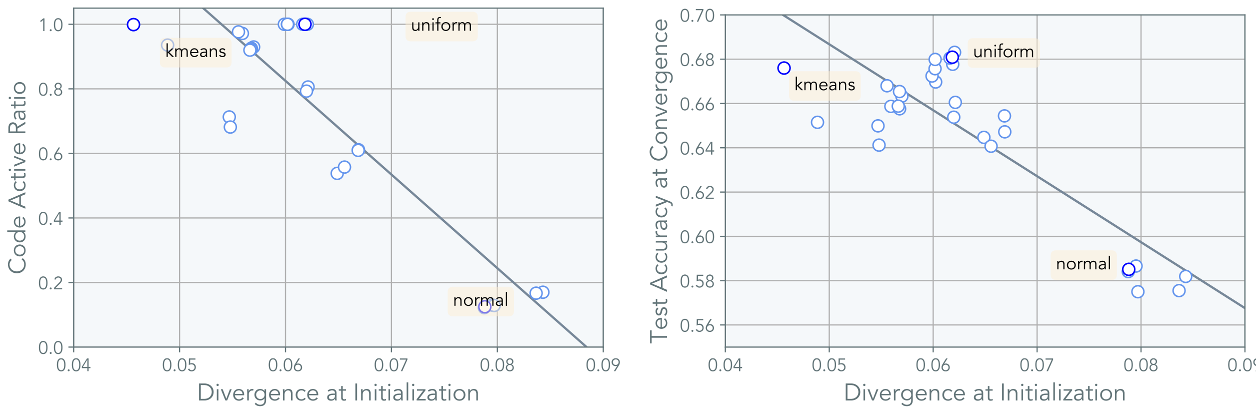

Figure 8: **Divergence vs Accuracy:** We visualize how the divergence between $\mathcal{P}_z$ and $\mathcal{Q}_z$ affect the performance of the model at convergence. We use various standard initialization schemes and provide some labels (the full list is reported in the appendix). We observed a linear relationship between the divergence at initialization and performance at convergence. We found that the quantization error at initialization determines the effect of codebook collapse.

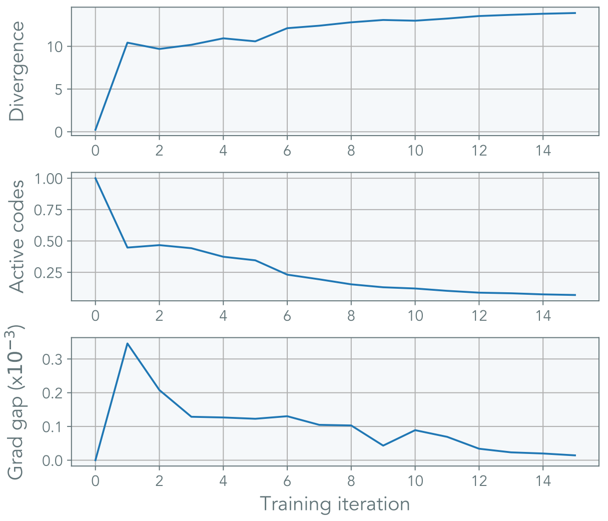

Figure 9: **Codebook collapse early on in training:** The dynamics of vector-quantized ResNet18 in the first 16 iteration of training. The divergence code is measured between $D(\mathcal{C}_{\mathbf{z}}, \mathcal{Q}_{\mathbf{z}})$. The model is initialized using K-means. Despite having ideal conditions, the model starts to degrade after a single iteration of SGD update.

generation results improve from the baseline by 15.6 FID and 4.9 FID from the best-performing variation. We did not scale to images to compute FID. In Figure 11, we visualize randomly sampled images from the model. No samples were cherry sampled.

A.9. Affine reparameterization on toy setting

The use of vector quantization adds to an already existing issue of internal covariate shift. When the internal representation, $\mathcal{P}\mathbf{z}$, is updated, it takes a longer time for the codebook, $\mathcal{C}\mathbf{z}$, to catch up. Furthermore, if the update in $\mathcal{P}\mathbf{z}$ is too large, for example, due to a high learning rate, the assignment of codes to embeddings can become severely misaligned. To visualize this, consider a toy example illustrated in Figure 10 where the embedding distribution, $\mathcal{P}\mathbf{z}$, and the codebook distribution, $\mathcal{C}_{\mathbf{z}}$, undergo a drift (left). As a result, in the next iteration, only a small fraction of the codes are chosen and updated, while the rest remain unchanged. This misalignment in the codebook and embedding distributions leads to suboptimal model performance. Using affine reparameterization can mitigate this by allowing gradients to flow through all code-vectors implicitly via the shared parameters. The toy example uses the learnable parameter variant.

When implementing affine reparameterization, we observed better performance when using accumulated statistics. We accumulate the statistics with a momentum of 0.1. We compute the moments of both $\mathbf{z}_e$ and $\mathcal{C}$ and compute the appropriate shift the match the moments of $\mathbf{z}_e$. We reduce the weighting of the commitment loss to $\alpha = 2$. When using the learnable parameters variant, while it performs slightly worse, the implementation only requires a 2 line change.

A.10. Discussion and intuition of the commitment loss

We discuss the difficulty of achieving perfect assignments using commitment loss. First, the upper bound of the commitment loss is given by:

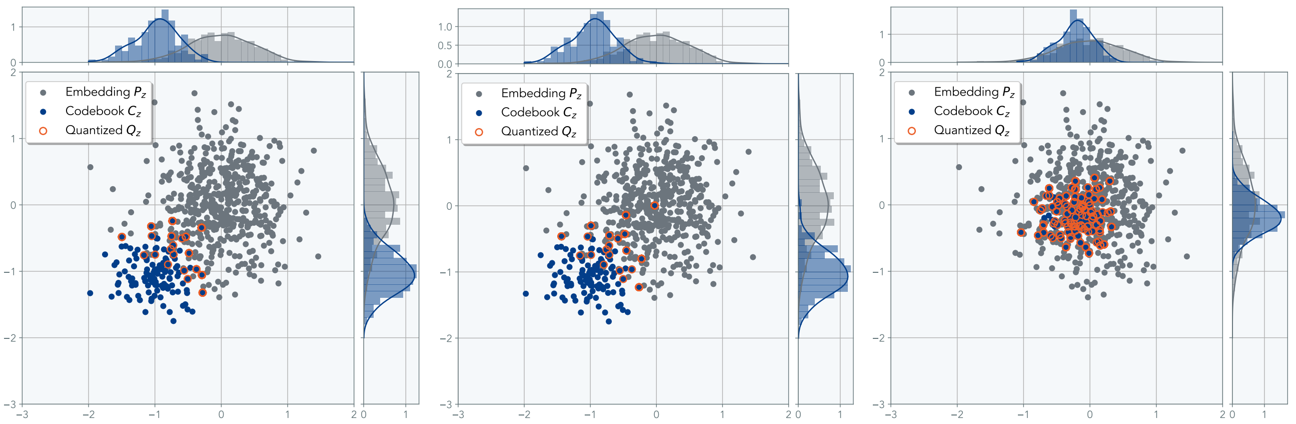

Figure 10: **Minimizing internal codebook covariate shift with affine parameterization:** Consider a toy example in 2D with the embedding distribution $\mathcal{P}_{\mathbf{z}} \sim \mathcal{N}([0, 0]^{\mathsf{T}}, 0.5 \cdot \mathbf{I})$ in grey and codebook distribution $\mathcal{C}_{\mathbf{z}} \sim \mathcal{N}([-1, -1]^{\mathsf{T}}, 0.3 \cdot \mathbf{I})$ in blue, and the selected quantized points $\mathcal{Q}_{\mathbf{z}}$ highlighted in orange. Here $|\mathcal{P}_{\mathbf{z}}| = 512$ and $|\mathcal{Q}_{\mathbf{z}}| = 128$. We visualize the distribution along with the density after 20 codebook updates using the standard commitment loss with $\beta = 0$ (same as the EMA-update variant) and a learning rate of 0.1. During initialization, only few samples are selected by the embedding distribution. Therefore, when using standard parameterization, more than 90% of the codes are not updated. For affine-parameterization, we consider a learnable variant. When using affine-parameterization, even if the codes are not selected, all the codes receive gradients and are able to better match the embedding distribution.

$$\mathcal{L}_{\text{commit}} = \frac{1}{2|\mathcal{P}_{\mathbf{z}}|} \sum_{\mathbf{z}_i \sim \mathcal{P}_{\mathbf{z}}} \min_{\mathbf{c}_j \sim \mathcal{C}} \|\mathbf{z}_i - \mathbf{c}_j\|^2 \tag{30}$$

$$\leq \min_{\mathbf{c}_j \sim \mathcal{C}} \frac{1}{2|\mathcal{P}_{\mathbf{z}}|} \sum_{\mathbf{z}_i \sim \mathcal{P}_{\mathbf{z}}} \|\mathbf{z}_i - \mathbf{c}_j\|^2 \tag{31}$$

$$= \frac{1}{2} \min_{\mathbf{c}_j \sim \mathcal{C}} \mathbb{E}_{\mathcal{P}_{\mathbf{z}}} \left[\|\mathbf{z}_i - \mathbf{c}_j\|^2\right] \tag{32}$$

$$= \frac{1}{2} Var(\mathcal{P}_{\mathbf{z}}) \tag{33}$$

Where the minimum is achieved when $c_j = \mu(\mathcal{P}_{\mathbf{z}})$, the mean of the embedding. By moving the minimization function outside, we optimize with respect to a single code vector.

The tight lower bound of the commitment loss can be achieved by solving an assignment problem. Let $\mathcal{B} = \{B_0, B_1, \ldots, B_m\}$, where $|\mathcal{C}| = m$. Each cluster ball B_i is a cluster associated with the code-vector c_i. Using this notation, we rewrite the commitment loss as:

$$\underset{\mathcal{B}=\{B_0,B_1,\ldots,B_m\}}{\arg\min} \frac{1}{2|\mathcal{P}_{\mathbf{z}}|} \sum_{i=1}^{m} \sum_{\mathbf{z}_i \sim B_i} \|\mathbf{z}_i - \mathbf{c}_j\|^2 \tag{34}$$

$$\underset{\mathcal{B}=\{B_0,B_1,\ldots,B_m\}}{\arg\min} \frac{1}{2|\mathcal{P}_{\mathbf{z}}|} \sum_{i=1}^{m} |B_i| \cdot Var(B_i) \tag{35}$$

Here we see that minimizing the commitment loss is equivalent to computing a set of balls $\mathcal{B}$ with the mean of each ball being $\mu(B_i) = \mathbf{c}_i$ such that the sum of the weight variance of all the balls is minimized. Solving this assignment problem NP-hard and cannot be trivially attained. Note that all existing algorithm for solving K-means is a heuristics. The above objective solves a global optimization assignment problem while optimizing with SGD is a greedy local update. With abuse of notation, define $B(\mathbf{c}_j) = \{\forall\ \mathbf{z}_i \in \mathcal{P}_{\mathbf{z}} \text{ where } \mathbf{c}_j = \arg\min(\|\mathbf{z}_i - \mathbf{c}_j\|^2)\}$ to be the set of points that is closest to the center $\mathbf{c}_j$. Then the commitment loss we optimize in practice is:

$$\min_{\mathbf{c}_j} \frac{1}{2|\mathcal{P}_{\mathbf{z}}|} \sum_{i=1}^{m} \sum_{\mathbf{z}_i \sim B(\mathbf{c}_j)} \|\mathbf{z}_i - \mathbf{c}_j\|^2 \tag{36}$$

With the update rule for the code vectors being:

$$\mathbf{c}_j^{(t+1)} \leftarrow \mathbf{c}_j^{(t)} - \eta \sum_{\mathbf{z}_i \in B(\mathbf{c}_j)} (\mathbf{z}_i - \mathbf{c}_j) \tag{37}$$

The update rule states that each code vector moves toward the mean of its current ball. Note that the cardinality of the ball can be zero; in such case, there is no gradient for the code-vector. Hence, in the worst case where $B(\mathbf{c}_j) = \emptyset$ for all $j \neq i$ for some i, we achieve the upper bound. Once the ball becomes empty, the code-vectors do not receive any gradients. To achieve the lower bound, one must hope that there is good coverage over $\mathcal{P}_{\mathbf{z}}$, and by minimizing the loss, we recover the optimal assignment. This highlights the difficulty of achieving the optimal assignment using the commitment loss.

A.11. Alternating optimization ablation

In Table 10, we compare how changing the number of inner and outer optimization steps affects model performance. We

	Inner. step	Outer. step	Accuracy	β
AlexNet	joint (baseline)		46.94 (+0.0)	0.9
	1	1	52.54 (+5.58)	0.9
	2	1	55.13 (+8.19)	0.9
	4	1	57.45 (+10.51)	0.95
	8	1	**58.03 (+11.09)**	0.98
	2	2	55.09 (+8.15)	0.95
	4	4	55.29 (+8.35)	0.95

Table 10: **Alternating optimization ablation:** We vary the number of the inner and outer loops for alternated training. All models observe the same number of training examples by splitting the mini-batch.

find that the inner step of $\times 8$ works the best. Increasing it further results in a negligible gain in performance.

A.12. Affine parameterization using EMA

When accumulating batch statistics for affine parameters, we compute an exponential moving average over the mean and variance of $\mathbf{z}_e$ and $\mathbf{z}_q$ with momentum m:

$$\mu_{\textsf{ema}}(\mathbf{z}_e) \leftarrow m \cdot \mu(\mathbf{z}_e) + (1-m) \cdot \mu_{\textsf{ema}}(\mathbf{z}_e) \tag{38}$$

$$\sigma^2_{\textsf{ema}}(\mathbf{z}_e) \leftarrow m \cdot \sigma^2(\mathbf{z}_e) + (1-m) \cdot \sigma^2_{\textsf{ema}}(\mathbf{z}_e) \tag{39}$$

$$\mu_{\textsf{ema}}(\mathbf{z}_q) \leftarrow m \cdot \mu(\mathbf{z}_q) + (1-m) \cdot \mu_{\textsf{ema}}(\mathbf{z}_q) \tag{40}$$

$$\sigma^2_{\textsf{ema}}(\mathbf{z}_q) \leftarrow m \cdot \sigma^2(\mathbf{z}_q) + (1-m) \cdot \sigma^2_{\textsf{ema}}(\mathbf{z}_q) \tag{41}$$

$$\tag{42}$$

We then use these statistics to normalize the $\mathbf{z}_q$. We first center the code-vectors and then re-normalize them back to the embedding moments:

$$\mathbf{z}_q \leftarrow \frac{\sigma_{\textsf{ema}}(\mathbf{z}_e)}{\sigma_{\textsf{ema}}(\mathbf{z}_q)}(\mathbf{z}_q - \mu_{\textsf{ema}}(\mathbf{z}_q)) + \sigma_{\textsf{ema}}(\mathbf{z}_e) \tag{43}$$

The affine parameters then correspond to:

$$\mathbf{c}_{\textsf{mean}} = \frac{\sigma_{\textsf{ema}}(\mathbf{z}_e)}{\sigma_{\textsf{ema}}(\mathbf{z}_q} \tag{44}$$

$$\mathbf{c}_{\textsf{std}} = \sigma_{\textsf{ema}}(\mathbf{z}_e) - \frac{\sigma_{\textsf{ema}}(\mathbf{z}_e)}{\sigma_{\textsf{ema}}}\mu_{\textsf{ema}}(\mathbf{z}_q) \tag{45}$$

The momentum m acts as the learning rate for the affine parameters. We find $m \in [0.01, 0.1]$ to be a good starting point for most models.

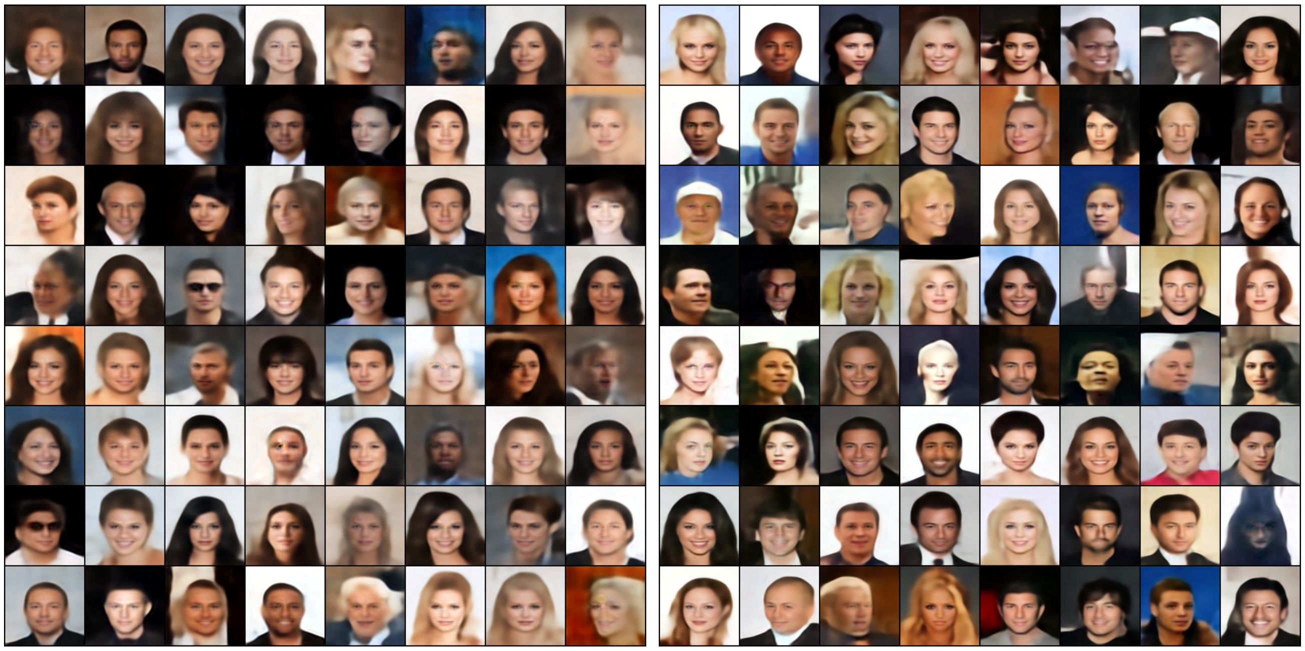

Figure 11: **MaskGIT on CelebA:** Generation result on 128×128 CelebA images. On the left we have the standard MaskGIT training, on the right we have MaskGIT with our proposed method.