DeepVQ: A Deep Network Architecture for Vector Quantization

Dang-Khoa Le Tan* Huu Le* Tuan Hoang* Thanh-Toan Do Ngai-Man Cheung†
Singapore University of Technology and Design (SUTD)
* indicates equal contribution †ngaiman_cheung@sutd.edu.sg

Abstract

Vector quantization (VQ) is a classic problem in signal processing, source coding and information theory. Leveraging recent advances in deep neural networks (DNN), this paper bridges the gap between a classic quantization problem and DNN. We introduce – for the first time – a deep network architecture for vector quantization (DeepVQ). Applying recent binary optimization theory, we propose a training algorithm to tackle binary constraints. Notably, our network outputs binary codes directly. As a result, DeepVQ can perform quantization of vectors with a simple forward pass, and this overcomes the exponential complexity issue of previous VQ approaches. Experiments show that our network is able to achieve encouraging results and outperforms recent deep learning-based clustering approaches that have been modified for VQ. Importantly, our network serves as a generic framework which can be applied for other networks in which binary constraints are required.

1. Introduction

Vector quantization (VQ) is a classical and important problem in source coding and information theory [6, 2]. Given a vector of source symbols of length D: $\mathbf{x} \in \mathbb{R}^D$, the problem considers representing $\mathbf{x}$ by one of the K reproduction vectors $\mathbf{x}' \in \mathbb{R}^D$. The primary goal of vector quantization is data compression: Given the input $\mathbf{x}$, at the encoder, the index $\mathbf{b}$ of the corresponding reproduction vector is computed; this index can be can be represented in binary using $L = \log_2 K$ bits in a transmission/storage system. At the decoder, the reproduction vector (*codeword*) $\mathbf{x}'$ corresponding to the index $\mathbf{b}$ is used as the reconstruction of the input $\mathbf{x}$.

VQ is motivated by the fundamental result of Shannon's rate-distortion theory [6, 2]: better performance can always be achieved by coding vectors instead of scalars, even if the sequence of source symbols are independent random variables. In particular, VQ is theoretically optimal: for sufficiently large D, at a given rate, the distortion (mean-squared-error MSE[1] between $\mathbf{x}$ and $\mathbf{x}'$) is arbitrarily close to the minimum distortion as stated in Shannon's rate-distortion theory. However, this information-theoretic result is *asymptotic* and *non-constructive*. Previous methods for VQ cannot use very long block size due to practical constraints, therefore performance is suboptimal.

Our contribution is to propose a novel deep neural network to address the classical VQ problem that overcomes the complexity issues of previous VQ methods. Our proposed Deep Vector Quantization (DeepVQ) is a new autoencoder design that directly outputs binary index of the codeword (i.e., $\mathbf{b}$) with a simple forward pass. As the network outputs binary code *directly*, there is no need to store reproduction vectors. DeepVQ addresses the scalability issue of previous VQ methods, enabling long block size to be used to achieve good performance. Furthermore, a deep network can learn intrinsic and robust features, and our approach has much better generalization capability to handle out-of-sample data. Training DeepVQ is very challenging as this involves non-smooth objective function and binary constraints. We attack it using alternating optimization and very recent binary optimization theory [16]. Note that DeepVQ is not only specific to the quantization problem; it is applicable when there involves integer constraints in the objective function of training a DNN, and image hashing is a notable example [3].

2. Related works

K-means (Generalized Lloyd algorithm) is the prime method for VQ. Given a set of training data $\{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^D$, the K-means algorithm partitions the N points into K clusters. Each vector is represented by its nearest centroid in $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_K]$. The training of K-means is achieved by minimizing the following objective function:

$$\begin{aligned} \min_{\mathbf{C},\{\mathbf{s}_i\}} \ & \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{C}\mathbf{s}_i\|_2^2, \\ \text{s.t.} \ & \ \mathbf{s}_i \in \{0,1\}^K, \ \ \|\mathbf{s}_i\|_1 = 1 \quad \forall i, \end{aligned} \tag{1}$$

where $\mathbf{s}_i$ is the assignment vector, which contains only one 1 element and the remaining $(K-1)$ elements are 0, corresponding to the data point $\mathbf{x}_i$; $\|\cdot\|_1$ denotes the l_1 norm. However, K-means based VQ has exponential complexity in encoding (computation and memory) and decoding (memory). Specifically, suppose L is the code size, and K-means needs to determine $K = 2^L$ centroids. This prevents K-means from using long block size. For example, to use $D = 784$ (e.g., $\mathbf{x}$ is a 28×28 image), and at a

[1] We focus on MSE although other distortion may be used.

rate of 0.05 bits per sample, this requires code of $L = 40$ bits, or $K = 2^{40}$ centroids. It is impractical to store in the encoder/decoder such large number of centroids, which are required in coding process. Furthermore, the number of training data required and the complexity of training is impractical for K-means. Various improvements of K-means have been proposed to address the complexity issues (e.g. product quantization) [5], but they are sub-optimal in performance. Notably, our DNN approach is fundamentally different from K-means for VQ design.

Recently, there are many works that apply deep learning to the clustering problem [15, 9, 12]. The main idea is to exploit DNN to map input data from the original high-dimensional space to the latent space with lower dimensionality. Then, K-means is applied to the latent codes. However, K-means in the latent space is sub-optimal for VQ, as Euclidean distances between data points are not preserved in the latent space. In our experiment, we show that adopting these works for VQ has sub-optimal performance. In addition, since K-means is still needed in these works, they still suffer from poor scalability when being used in VQ.

Constrained objective functions in deep learning appear not only in VQ problem but also in image hashing [3] and image compression [4, 13]. In image compression, the main barrier is the round function which is non-differentiable and thus it causes vanishing gradient. Instead of replacing the round function completely by a smooth function, [13] approximated the derivative by the identity function. [4] instead used a learnable uniform scale quantization in a convolutional autoencoder. In general, the current approach is to approximate the non-differentiable functions by smooth functions and then use SGD to train the whole network parameters. Even though these approximations help the quantization learnable via SGD, they could degrade the performance of the models as the loss function is not precise.

Recently, Do et al. [3] introduced a binary auto-encoder for image hashing, in which the task of binary optimization is approached by cyclic coordinate descent. However, their method cannot be applied to a stack of layers with nonlinearities. Recent works [11, 10] proposed novel reparameterization methods to handle the non-gradient issue for discrete random variables in Variational Auto-Encoder. VQ-VAE [14] avoids such problem by the identity function, namely copying gradients from the decoder input to encoder output. Nevertheless, our work addresses a different problem of optimizing a network with the strict binary constraint.

3. Proposed method

3.1. Network architecture

Given the set of input data $\{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^D$, VQ – in the context of data compression – seeks to encode $\mathbf{x}_i$ into its compact binary code $\mathbf{b}_i \in \{-1, 1\}^L (L < D)$. Our solution is an auto-encoder with the innermost hidden layer outputting binary code. Let $f(\mathbf{x}) : \mathbb{R}^D \mapsto \{-1, 1\}^L$ denote the encoding network, and $g(\mathbf{b}) : \{-1, 1\}^L \mapsto \mathbb{R}^D$ be the decoding network, which maps a compressed data point $\mathbf{b}_i$ back to the original space $\mathbb{R}^D$. The task of finding the set of binary codes $\{\mathbf{b}_i\}$ and the mapping functions $f(.)$ and $g(.)$ can be mathematically formulated as solving this optimization:

$$\begin{aligned} \min_{\mathbf{W}_1, \mathbf{W}_2} \quad & \sum_{i=1}^N \|g(f(\mathbf{x}_i)) - \mathbf{x}_i\|_2^2 \\ \text{s.t.} \quad & f(\mathbf{x}_i) \in \{-1, 1\}^L \ \ \forall i \in \{1, \dots, N\} \end{aligned} \tag{2}$$

Here, $\mathbf{W}_1$ and $\mathbf{W}_2$ are the parameters of the network f and g, respectively. The binary codes can be computed directly as $\mathbf{b}_i = f(\mathbf{x}_i)$. The reproduction vector can be computed as $\mathbf{x}'_i = g(\mathbf{b}_i)$. Comparing to K-means, no storing of centroid is needed with our approach. During encoding, no searching of codeword is required. Significantly, in contrast to the conventional auto-encoder, the output of the encoder in our proposed method is constrained to be binary, i.e., $f(\mathbf{x}_i) = \mathbf{b}_i \in \{-1, 1\}^L$, which in fact can be interpreted as an index of its corresponding centroid $g(\mathbf{b}_i)$.

The encoder of our network consists of three hidden layers and one binary layer as the the encoder's output. In this work, we let the decoder and encoder share the same weights, thus the decoder is mirrored from the encoder. Since we aim at providing a generic framework for multiple types of input data, the layers in our network are chosen to be fully connected in this work. We use $tanh$ as activation functions for the layers. Our method can later be modified with other structures for other types of input signals, e.g., convolutional layers for images.

3.2. Optimization method

Solving (2) is a challenging task, since in addition to the fact that f and g can be heavily non-convex, one must also deal with the binary constraint at the encoder's output. Consequently, (2) cannot be tackled directly by well-known smooth optimization methods. To the best of our knowledge, we are the first to propose a network architecture where f and g are non-linear and the binary constraints are strictly enforced during the training process.

First, to simplify notation, we reformulate (2) in the matrix form. Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be the matrix that contains all the input data, where the $i-$th row of $\mathbf{X}$ represents the data $\mathbf{x}_i$. Similarly, let $\mathbf{B}$ be the matrix that contains all the encoded binary codes corresponding to the input data $\mathbf{X}$. Note that henceforth, f and g are considered as functions that acts on every single row of the input matrix. With the newly introduced notations, (2) can now be written as

$$\begin{aligned} \min_{\mathbf{W}_1, \mathbf{W}_2, \mathbf{B}} \quad & \|g(\mathbf{B}) - \mathbf{X}\|_F^2, \\ \text{s.t} \quad & \mathbf{B} \in \{-1, 1\}^{N \times L}, f(\mathbf{X}) = \mathbf{B}. \end{aligned} \tag{3}$$

We propose to tackle (3) by penalty method. Specifically, by incorporating constraint $f(\mathbf{X}) = \mathbf{B}$ into the cost function, together with a penalty parameter γ , we have:

$$\begin{aligned} \min_{\mathbf{W}_1,\mathbf{W}_2,\mathbf{B}} \quad & \|g(\mathbf{B}) - \mathbf{X}\|_F^2 + \gamma\|f(\mathbf{X}) - \mathbf{B}\|_F^2, \\ \text{s.t} \quad & \mathbf{B} \in \{-1, 1\}^{N\times L} \end{aligned} \tag{4}$$

Observe that (4) contains three sets of parameters $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{B}$, where only $\mathbf{B}$ is constrained to be binary. We attack (4) using alternating optimization. Particularly, one can iteratively fix $\mathbf{B}$ and update $\mathbf{W}_1, \mathbf{W}_2$, then update $\mathbf{B}$ with $\mathbf{W}_1, \mathbf{W}_2$ fixed.

Updating $\mathbf{W}_1$, $\mathbf{W}_2$: With $\mathbf{B}$ fixed, (4) becomes an unconstrained optimization problem

$$\min_{\mathbf{W}_1,\mathbf{W}_2} \quad \|g(\mathbf{B}) - \mathbf{X}\|_F^2 + \gamma\|f(\mathbf{X}) - \mathbf{B}\|_F^2. \tag{5}$$

Since the cost function of (5) is differentiable, it can be solved by popular gradient-descent based optimization methods.

Updating $\mathbf{B}$: Starting from (4), if $\mathbf{W}_1$ and $\mathbf{W}_2$ are fixed, $\mathbf{B}$ can be updated by solving

$$\begin{aligned} \min_{\mathbf{B}} \quad & \|g(\mathbf{B}) - \mathbf{X}\|_F^2 + \gamma\|f(\mathbf{X}) - \mathbf{B}\|_F^2 \\ \text{s.t} \quad & \mathbf{B} \in \{-1, 1\}^{N\times L} \end{aligned} \tag{6}$$

Unlike (5), as noted previously, (6) cannot be tackled directly by gradient-based solvers due to the binary restriction. To resolve this, inspired by [16], we develop a new optimization technique for (6). More specifically, we first introduce an auxiliary variable $\mathbf{V} \in \mathbb{R}^{N\times L}$ and reformulate (6) equivalently as follows:

$$\begin{aligned} \min_{\mathbf{B},\mathbf{V}} \quad & \|g(\mathbf{B}) - \mathbf{X}\|_F^2 + \gamma\|f(\mathbf{X}) - \mathbf{B}\|_F^2 \\ \text{s.t.} \quad & -\mathbf{1}_{N\times L} \le \mathbf{B} \le \mathbf{1}_{N\times L}, \\ & \langle \mathbf{B}, \mathbf{V}\rangle = NL, \|\mathbf{V}\|_F^2 \le NL, \end{aligned} \tag{7}$$

with $\mathbf{1}_{N\times L}$ represents a matrix of size $N \times L$ in which all elements are equal to one, and $\langle ., .\rangle$ denotes the inner product of two matrices.

Note that besides the bi-linear constraint $\langle \mathbf{B}, \mathbf{V}\rangle = NL$, the other two constraints in (7) are convex constraints. This observation encourages us to employ one more time the penalty method to solve (7). Specifically, with the penalty parameter ρ, and let $h(\mathbf{B}) = \|g(\mathbf{B}) - \mathbf{X}\|_F^2 + \gamma\|f(\mathbf{X}) - \mathbf{B}\|_F^2$, the penalty problem to solve (7) can be written as

$$\begin{aligned} \min_{\mathbf{B},\mathbf{V}} \quad & h(\mathbf{B}) + \rho\left(\langle \mathbf{B}, \mathbf{V}\rangle - NL\right)^2 \\ \text{s.t.} \quad & -\mathbf{1}_{N\times L} \le \mathbf{B} \le \mathbf{1}_{N\times L}, \\ & \|\mathbf{V}\|_F^2 \le NL, \end{aligned} \tag{8}$$

Similarly, (8) can also be optimized by alternatively updating $\mathbf{B}$ and $\mathbf{V}$. Each sub-problem can be solved by gradient-descent based techniques. Due to space limit, the optimization details and convergence proof will be discussed in details in our extended version of this paper.

4. Experiments

In this section, we compare our method with K-means and its variants, namely PQ [8] and OPQ [5] which are able to encode with long code lengths. In addition, we also compare with the recent method Deep Clustering Network (DCN) [15]. DCN uses an auto-encoder to learn latent representations which are suitable for clustering using the K-means algorithm. The objective function of DCN is as following:

$$\begin{aligned} \min_{\substack{\mathbf{W}_1,\mathbf{W}_2\\ \mathbf{C},\{\mathbf{s}_i\}}} \quad & \sum_{i=1}^{N}\left(\|g(f(\mathbf{x}_i)) - \mathbf{x}_i\|_2^2 + \frac{\lambda}{2}\|f(\mathbf{x}_i) - \mathbf{C}\mathbf{s}_i\|_2^2\right) \\ \text{s.t.} \quad & \mathbf{s}_i \in \{0, 1\}^k, \|\mathbf{s}_i\|_1 = 1 \quad \forall i. \end{aligned} \tag{9}$$

In DCN, the encoder, e.g. $f(\cdot)$, is not constrained to produce binary codes. It is worth noting that DCN tries to conduct clustering in the latent space, which may be distorted and does not preserve the pairwise distances in the original space. This may not be suitable for VQ. To handle this issue, we propose to modify the DCN network such that during the training process, it uses the centroids learned in the latent space instead of the outputs of the encoder to reconstruct the original input. Specifically, the modification makes the loss function of DCN become:

$$\begin{aligned} \min_{\substack{\mathbf{W}_1,\mathbf{W}_2\\ \mathbf{C},\{\mathbf{s}_i\}}} \quad & \sum_{i=1}^{N}\left(\|g(\mathbf{C}\mathbf{s}_i) - \mathbf{x}_i\|_2^2 + \frac{\lambda}{2}\|f(\mathbf{x}_i) - \mathbf{C}\mathbf{s}_i\|_2^2\right) \\ \text{s.t.} \quad & \mathbf{s}_i \in \{0, 1\}^k, \|\mathbf{s}_i\|_1 = 1 \quad \forall i. \end{aligned} \tag{10}$$

This modification is denoted as Modified DCN (MDCN).

We use the common **MNIST** dataset for our evaluation. The MNIST dataset comprises of 70k handwritten grayscale images of 10 digits. We adopt the standard data split of 60k training images and 10k testing images in our experiments. Each image is represented by a 784-D grayscale feature vector by using its intensity.

4.1. Implementation details

All datasets are normalized into the range of $[-1, 1]$. The proposed method is implemented by Python, and we design the deep auto-encoder model by Theano. In our proposed DeepVQ, the encoder includes 3 fully connected layers with the $tanh$ activation function followed by batch normalization [7]. The sizes of the hidden layers are: 500, 255, L, where L is the latent dimension. The decoder plays a role as an inverse of the encoder; hence, we share weights between the encoder and the decoder, and decoder layers are symmetric to encoder layers. To initialize the parameters of DeepVQ, we apply the layer-wise pre-training method [1]. To update the network, we use SGD with momentum where learning rate $\eta = 10^{-5}$ and the momentum parameter $\alpha = 0.9$. We train 3 epochs in which we set batch

Table 1. MSE of training and testing data on MNIST dataset

	Training					Testing (lower is better)				
L	10	12	15	24	32	10	12	15	24	32
DeepVQ	**0.1439**	**0.1358**	0.1254	**0.1101**	**0.1123**	**0.1525**	**0.1438**	**0.1335**	**0.1162**	**0.1180**
OPQ [5]	0.2324	0.2138	0.2016	0.1495	0.1220	0.2303	0.2123	0.2001	0.1492	0.1240
PQ [8]	0.2649	0.2406	0.2313	0.1654	0.1330	0.2672	0.2390	0.2293	0.1645	0.1341
DCN [15]	0.9386	0.8935	0.8560	–	–	0.9361	0.8947	0.8961	–	–
MDCN	0.9113	0.8902	0.8558	–	–	0.9112	0.8922	0.8754	–	–
K-means	0.1786	0.1609	**0.0421**	–	–	0.1791	0.1704	0.1658	–	–

size at 128 for each configuration. Regarding the penalty optimization, we solve Equation 8 with $\rho = 0.05$ and 10 iterations in each batch. The penalty parameter $\gamma = 1$ over all settings. For other compared methods, we use the published code and recommended parameters from the authors.

4.2. Results

Table 1 shows the MSE of the training and testing data of the MNIST dataset. As shown in Table 1, our proposed method significantly outperforms DCN and the modified DCN methods with large margins in the MNIST dataset. Note that DCN applies K-means on the low-dimensional latent representations. Even though K-means works very well in low-dimensional space, DeepVQ still manages to achieve better performances than DCN. At $L = 15$ we observe overfitting with using K-means, but this does not happen in our proposed DeepVQ network. In fact, this is because our network does not necessarily produce 2^L different centroids which obviously causes the overfitting problem in K-means. At higher code lengths, our method gets better MSE than PQ and OPQ. Furthermore, our approach is very computationally-efficient for encoding. Additional results will be discussed in the extended version.

5. Conclusion

We introduce the deep network architecture for the vector quantization task, named DeepVQ. Specifically, we train an auto-encoder in which the encoder is constrained to produce binary code and the decoder plays the role of generating codewords from indexes on-the-fly. The experiments show that our method can achieve competitive or more favorable quantization errors in comparison to the common used method, e.g. K-means, and recent deep learning approaches. Moreover, our proposed method, in fact, can be applied to other applications which require the binary constraint on any layer of DNNs. In the extended version, we will introduce the convergence proof: binary constraints are strictly enforced using our proposed training algorithm.

References

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

[2] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2006.

[3] T.-T. Do, D.-K. Le Tan, T. T. Pham, and N.-M. Cheung. Simultaneous feature aggregating and hashing for large-scale image search. In *CVPR*, 2017.

[4] T. Dumas, A. Roumy, and C. Guillemot. Autoencoder based image compression: can the learning be quantization independent? 2018.

[5] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *TPAMI*, 36(4):744–755, 2014.

[6] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[8] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, pages 117–128, 2011.

[9] P. Ji, T. Zhang, H. Li, M. Salzmann, and I. Reid. Deep subspace clustering networks. In *NIPS*, 2017.

[10] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*. 2017.

[11] J. T. Rolfe. Discrete Variational Autoencoders. In *ICLR*. 2017.

[12] U. Shaham, K. Stanton, H. Li, B. Nadler, R. Basri, and Y. Kluger. Spectralnet: Spectral clustering using deep neural networks. 2018.

[13] L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. 2017.

[14] A. van den Oord, O. Vinyals, and k. kavukcuoglu. Neural discrete representation learning. In *NIPS*, 2017.

[15] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *ICML*, 2017.

[16] G. Yuan and B. Ghanem. An exact penalty method for binary optimization based on MPEC formulation. In *AAAI*, 2017.