Journal of Mathematics in Industry
a SpringerOpen Journal

RESEARCH Open Access

Using low-discrepancy points for data compression in machine learning: an experimental comparison

S. Göttlich[1]*, J. Heieck[1] and A. Neuenkirch[1]

*Correspondence:
goettlich@math.uni-mannheim.de
[1]Department of Mathematics, University of Mannheim, 68131 Mannheim, Germany

Abstract

Low-discrepancy points (also called Quasi-Monte Carlo points) are deterministically and cleverly chosen point sets in the unit cube, which provide an approximation of the uniform distribution. We explore two methods based on such low-discrepancy points to reduce large data sets in order to train neural networks. The first one is the method of Dick and Feischl (J Complex 67:101587, 2021), which relies on digital nets and an averaging procedure. Motivated by our experimental findings, we construct a second method, which again uses digital nets, but Voronoi clustering instead of averaging. Both methods are compared to the supercompress approach of (Stat Anal Data Min ASA Data Sci J 14:217–229, 2021), which is a variant of the *K*-means clustering algorithm. The comparison is done in terms of the compression error for different objective functions and the accuracy of the training of a neural network.

Mathematics Subject Classification: 41A99; 65C05; 65D15; 68T07

Keywords: Data reduction; Low-discrepancy points; Quasi-Monte Carlo; Digital nets; *K*-means algorithm; Neural networks

1 Introduction

Data reduction is a classical technique that reduces the size of a dataset while still preserving the most important information. Concepts and methods in this field include CoreSets [7], support points [19] and random subsampling [18], to mention a few. However, low-discrepancy points and quasi Monte Carlo-techniques seem to have received little attention in this context so far. They were used to create training data points for learning surrogate models in urban traffic by [4] and also similarly in [17, 20] for a variety of applications, including several types of partial differential equations. Dick and Feischl [5] proceeded differently, namely by using low-discrepancy points to compress known data for the training of neural networks. This work was the starting point of our study.

We assume that the original data, denoted by $\mathcal{X}$, is a set of N points in $[0,1)^s$. The corresponding responses, denoted by $\mathcal{Y}$, are a set of N points in $\mathbb{R}$. The objective is to predict the relationship between the s attributes of the data points from $\mathcal{X}$ and the single attribute of the responses in $\mathcal{Y}$. This is achieved through the use of a parametrized predictor function $f_\theta : [0,1)^s \to \mathbb{R}$ where $\theta \in \Theta \subseteq \mathbb{R}^p$. The quality of the predictor function f_θ is measured

© The Author(s) 2025. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

Springer

by the ℓ_2-distance between the predicted value $f_\theta(\boldsymbol{x}_n)$ and the actual value y_n for each instance $n = 1, \dots, N$, which is a classical approach for such regression problems. Hence, the error of the predictor f_θ is defined as

$$\operatorname{err}(f_\theta) := \frac{1}{N} \sum_{n=1}^{N} \left(f_\theta(\boldsymbol{x}_n) - y_n\right)^2. \tag{1.1}$$

To identify an optimal value for θ, which can be chosen from the set Θ, it may be necessary to evaluate the error function (or potentially its derivatives $\nabla_\theta^m \operatorname{err}(f_\theta)$, for values of $m \in \mathbb{N}$) on a number of occasions, resulting in a cost that is proportional to

$$\#\text{optimisation steps} \times \underbrace{\#\text{data points}}_{=N}.$$

To reduce this cost, one can use a compressed data set, denoted by $P = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_L\} \subset [0,1)^s$ with $L \ll N$, in combination with an approximation of the quadratic loss function (1.1).

In the case of P resulting from the supercompress method from [14] or the QMC-Voronoi-method, each compressed data point will be assigned to a corresponding (approximate) response, denoted by $\mathcal{W} = \{w_1, \dots, w_L\}$. The approximation of the quadratic loss function is then given by

$$\operatorname{err}(f_\theta) \approx \operatorname{app}_L^{clst}(f_\theta) := \frac{1}{L} \sum_{l=1}^{L} \left(f_\theta(\boldsymbol{z}_l) - w_l\right)^2. \tag{1.2}$$

Both approaches rely on clustering, hence we use the label *clst* for the approximate error function.

The approach of Dick and Feischl [5] proceeds in a different way. Here, for the reduced set of responses P, weights $\left\{W_{\mathcal{X},P,v,l}\right\}_{l=1}^{L}$ and $\left\{W_{\mathcal{X},\mathcal{Y},P,v,l}\right\}_{l=1}^{L} \subset \mathbb{R}$ are computed and the approximation of the loss is given by

$$\operatorname{err}(f_\theta) \approx \operatorname{app}_L^{avg}(f_\theta) := \sum_{l=1}^{L} f_\theta^2(\boldsymbol{z}_l)\, W_{\mathcal{X},P,v,l} - 2 \sum_{l=1}^{L} f_\theta(\boldsymbol{z}_l)\, W_{\mathcal{X},\mathcal{Y},P,v,l} + \frac{1}{N} \sum_{n=1}^{N} y_n^2. \tag{1.3}$$

Here v is a parameter, which will be explained later on. Since the calculation of the weights relies on an averaging procedure, we use the label *avg* for the approximate error function.

The reduced points sets, approximate responses and weights are all independent of θ, which implies that they can be calculated once at the outset and subsequently reused throughout the optimization process. Since $L \ll N$, the cost is now proportional to

$$\#\text{optimisation steps} \times \underbrace{\#\text{compressed data points}}_{=L}.$$

In order to compare the approaches, we will first apply them to some given and fixed functions f instead of f_θ, see Sect. 4.1, and will study the error of the approximate loss functions.

In a second step, we will compare the performance of the methods, when used for fitting neural networks. To this end, we will train a neural network with the original data $\mathcal{X}$ and

$\mathcal{Y}$ and then compare the prediction accuracy of the original data with that of the neural network trained with the compressed set. This will be done for the MNIST data set [16], as proposed in [5]. Our experiments support the hypothesis that for the studied problems the adaptive clustering from the supercompress approach is superior to the QMC-Voronoi method, which again performs better than the QMC-averaging approach.

The remainder of this manuscript is structured as follows: In the next section, we will give a short motivation, why low-discrepancy point sets could be beneficial for data reduction in regression problems. In Sect. 3 we describe the different data compression approaches, while Sect. 4 contains our numerical experiments. Our detailed findings and conclusion can be found in Sect. 5.

2 Low-discrepancy point sets and regression

We start by introducing the concept of low-discrepancy point sets and some corresponding results. See, e.g. Chap. 2 in [22] for further information. Let $\mathcal{P} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \subset [0,1)^s$ and

$$A([\boldsymbol{a}, \boldsymbol{b}), \mathcal{P}) = \sum_{n=1}^{N} \mathbb{1}_{[\boldsymbol{a},\boldsymbol{b})}(\boldsymbol{x}_n),$$

where $\mathbb{1}_{[\boldsymbol{a},\boldsymbol{b})}$ is the characteristic function of the interval $[\boldsymbol{a}, \boldsymbol{b}) = \prod_{i=1}^{s}[a_i, b_i)$. Hence, $A([\boldsymbol{a}, \boldsymbol{b}), \mathcal{P})$ is the number of the points of $\mathcal{P}$, which belong to $[\boldsymbol{a}, \boldsymbol{b})$. The discrepancy D_N of the point set $\mathcal{P}$ is then defined as

$$D_N(\mathcal{P}) = \sup_{\substack{\boldsymbol{a},\boldsymbol{b}\in[0,1]^s \\ \boldsymbol{a}\le\boldsymbol{b}}} \left| \frac{A([\boldsymbol{a}, \boldsymbol{b}), \mathcal{P})}{N} - \lambda_s([\boldsymbol{a}, \boldsymbol{b})) \right|$$

and measures the deviation of the empirical distribution of the points in $\mathcal{P}$ from the uniform distribution λ_s. A related quantity is the so-called star-discrepancy

$$D_N^*(\mathcal{P}) = \sup_{\boldsymbol{a}\in[0,1]^s} \left| \frac{A([0, \boldsymbol{a}), \mathcal{P})}{N} - \lambda_s([0, \boldsymbol{a})) \right|.$$

Point sets $\mathcal{P}$ with small (star-)discrepancy are suitable for the numerical integration of functions $g : [0,1]^s \to \mathbb{R}$. In particular, if g is continuous we have the error bound

$$\left| \frac{1}{N} \sum_{i=1}^{N} g(\boldsymbol{x}_i) - \int_{[0,1]^s} g(u)\, du \right| \le 4w(g; \mathcal{D}_N^*(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)^{1/s})$$

with the modulus of continuity

$$w(g; \delta) = \sup_{\substack{\boldsymbol{u},\boldsymbol{v}\in[0,1]^s \\ \|\boldsymbol{u}-\boldsymbol{v}\|_\infty \le \delta}} |g(\boldsymbol{u}) - g(\boldsymbol{v})|, \quad \delta > 0.$$

Moreover, the famous Koksma-Hlawka inequality reads as

$$\left| \frac{1}{N} \sum_{i=1}^{N} g(\boldsymbol{x}_i) - \int_{[0,1]^s} g(u)\, du \right| \le V(g)\, \mathcal{D}_N^*(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N). \tag{2.1}$$

Here and in the following $V(g)$ is the Hardy-Krause variation of g.

Now, let X be a random vector with values in $[0,1]^s$, Y be a random variable and $f_\theta : [0,1]^s \to \mathbb{R}$ with $\theta \in \Theta \subseteq \mathbb{R}^p$. Finding the parameter θ, which minimizes the expected prediction error

$$\mathcal{L}(\theta) = \mathbf{E}|f_\theta(X) - Y|^2,$$

is the classical L^2-regression problem. The loss function (1.1), i.e.

$$\operatorname{err}\left(f_\theta\right) := \frac{1}{N}\sum_{n=1}^{N}\left(f_\theta\left(\boldsymbol{x}_n\right) - y_n\right)^2,$$

can be seen as the empirical variant of the expected prediction error by assuming that $(\boldsymbol{x}_n, y_n)$, $n = 1,\dots,N$, are independent and identically distributed realizations of (X,Y). If (X,Y) has a joint Lebesgue-density $\varphi : [0,1]^s \times \mathbb{R} \to [0,\infty)$ with marginal densities $\varphi_X : [0,1]^s \to [0,\infty)$ and $\varphi_Y : \mathbb{R} \to [0,\infty)$, then we have

$$\begin{aligned}\mathcal{L}(\theta) &= \int_{[0,1]^s\times\mathbb{R}} |f_\theta(x) - y|^2 \varphi(x,y)d(x,y)\\ &= \int_{[0,1]^s} f_\theta(x)^2\varphi_X(x)dx - 2\int_{[0,1]^s\times\mathbb{R}} f_\theta(x)y\varphi(x,y)d(x,y) + \int_{\mathbb{R}} y^2\varphi_Y(y)dy.\end{aligned}$$

Now using low-discrepancy points as the compressed data points $P = \{\boldsymbol{z}_1,\dots,\boldsymbol{z}_L\} \subset [0,1)^s$ for the dx-integration and the data from $\mathcal{Y}$ for the dy-integration, one obtains an approximation of the form (1.3), i.e.

$$\mathcal{L}(\theta) \approx \operatorname{app}_L^{avg}(f_\theta) = \sum_{l=1}^{L} f_\theta^2\left(\boldsymbol{z}_l\right) W_{\mathcal{X},P,\nu,l} - 2\sum_{l=1}^{L} f_\theta\left(\boldsymbol{z}_l\right) W_{\mathcal{X},\mathcal{Y},P,\nu,l} + \frac{1}{N}\sum_{n=1}^{N} y_n^2.$$

Thus, if the empirical loss function (1.1) is close to the expected prediction error $\mathcal{L}(\theta)$, which is reasonable if N is large, then $\operatorname{app}_L^{avg}(f_\theta)$ should provide a good approximation of (1.1).

3 Data compression methods

In this section we present the different compression algorithms and provide the principal ideas of their implementation.

3.1 Quasi-Monte Carlo compression

Here we present the algorithm and results of [5], where digital nets, which are particular (t_α, m, s)-nets, are used as low-discrepancy sets. In the following, we assume that $\alpha \geq 1$ is an integer and that $b \geq 2$ is prime. For a vector $\boldsymbol{d}$, $|\boldsymbol{d}|$ denotes its ℓ_1-norm, while for a set A, the notation $|A|$ denotes its cardinality.

Definition 3.1 (e.g., p.5, [5]) For $\boldsymbol{d} \in \mathbb{N}_0^s$ we set $K_{\boldsymbol{d}} := \left\{\boldsymbol{a} = (a_1,\dots,a_s)^\intercal \in \mathbb{N}_0^s : a_j < b^{d_j}\right\}$. A point set $P = \{\boldsymbol{z}_1,\dots,\boldsymbol{z}_L\} \subset [0,1)^s$ consisting of $L = b^m$ points is called a (t_α, m, s)-net in base b of order α, if every elementary interval $I_{\boldsymbol{a},\boldsymbol{d}} = \prod_{j=1}^{s}\left[\frac{a_j}{b^{d_j}}, \frac{a_j+1}{b^{d_j}}\right)$ with $|\boldsymbol{d}| = m - t_\alpha$ and $\boldsymbol{a} \in K_{\boldsymbol{d}}$ contains exactly b^{t_α} points.

The idea behind such nets is to identify a set of points that is evenly distributed within the unit cube $[0,1)^s$. It is essential that each elementary interval of size $b^{-(m-t_\alpha)}$ contains

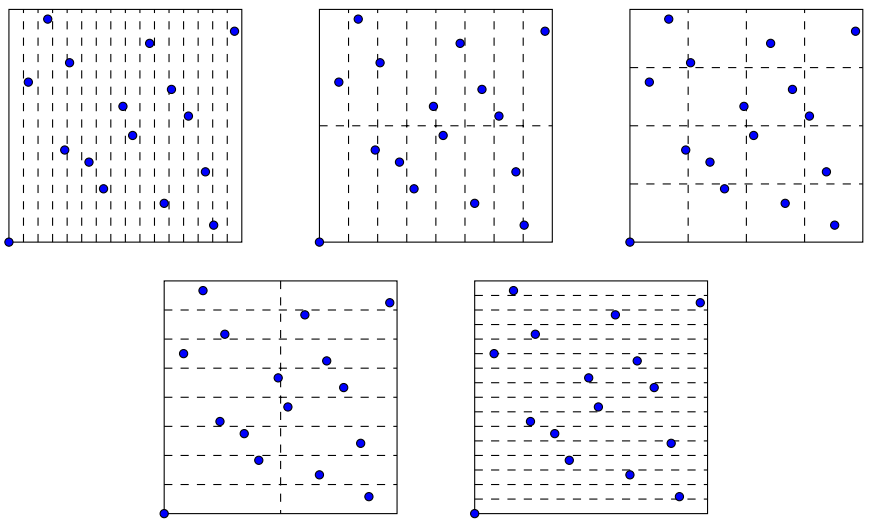

Figure 1 (0, 4, 2)-net in base 2 (blue points)

an identical number of points. Figure 1 shows an example of a first order $(0,4,2)$-net in base $b = 2$. The point set is well distributed for every partition of the unit square into elementary intervals of size $b^{-4} = \frac{1}{16}$. This means that every interval contains the same number of points.

For each (t_α, m, s)-net P in base b of order α and for each $\nu \leq m - t_\alpha$, $\boldsymbol{d} \in \mathbb{N}_0^s$ and $\boldsymbol{a} \in K_{\boldsymbol{d}}$, it can be shown that $\left|P_{\boldsymbol{a},\boldsymbol{d}}\right| = b^{m-\nu+q}$, where $P_{\boldsymbol{a},\boldsymbol{d}}$ denotes the set P intersected with $I_{\boldsymbol{a},\boldsymbol{d}}$ and $|\boldsymbol{d}| = \nu - q$ for $q \in \{0, \ldots, \min(s-1, \nu)\}$. As mentioned in Chap. 2 of [5], this is a consequence of Definition 3.1 and will be important for deriving the weights.

We need to introduce further notations. For $\boldsymbol{y} \in [0,1)^s$, $I_{\boldsymbol{d}}(\boldsymbol{y})$ describes the elementary interval $I_{\boldsymbol{a},\boldsymbol{d}}$, which contains $\boldsymbol{y}$. We define the set of points that lie within the given elementary interval as follows:

$$\mathcal{X}_{\boldsymbol{a},\boldsymbol{d}} := \mathcal{X} \cap I_{\boldsymbol{a},\boldsymbol{d}}, \quad \mathcal{X}_{\boldsymbol{d}}(\boldsymbol{y}) := \mathcal{X} \cap I_{\boldsymbol{d}}(\boldsymbol{y}) \quad \text{and} \quad P_{\boldsymbol{d}}(\boldsymbol{y}) := P \cap I_{\boldsymbol{d}}(\boldsymbol{y}).$$

Furthermore, we define the set K_ν as the union of all $K_{\boldsymbol{d}}$ for which $\boldsymbol{d} \in \mathbb{N}_0^s$ and $|\boldsymbol{d}| = \nu$. The volume of each elementary interval is determined by the value of ν, which is used to calculate the weights.

With these notations in hand, we state a Lemma, which will be used to calculate the weights by an averaging procedure, and allows us to work with K_ν. This is done in order to include every partition of the unit cube, for which the elementary intervals exhibit the same volume.

Lemma 3.2 (Lemma 1, [5]) *Let $\nu \geq 0$ be an integer. For all $\boldsymbol{a} \in \mathbb{N}_0^s$ the combination principle*

$$\mathbb{1}_{K_\nu}(\boldsymbol{a}) = \sum_{q=0}^{\min(s-1,\nu)} (-1)^q \binom{s-1}{q} \sum_{\substack{\boldsymbol{d} \in \mathbb{N}_0^s \\ |\boldsymbol{d}| = \nu - q}} \mathbb{1}_{K_{\boldsymbol{d}}}(\boldsymbol{a})$$

holds. Here, $\mathbb{1}_A$ denotes the indicator function for an arbitrary set A.

Lemma 3.2 can be exploited to find an expression for the weights $\left\{W_{\mathcal{X},P,\nu,l}\right\}_{l=1}^{L}$. In the following, we assume that our data set is given by $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, our responses by $\mathcal{Y} = \left\{y_1, \ldots, y_N\right\}$, our compressed data set by $P = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L\}$, and that ν is fixed.

Firstly, we consider a fixed $\boldsymbol{d} \in \mathbb{N}_0^s$, which represents a fixed partition of the unit cube. This allows us to derive the following approximation:

$$\frac{1}{N}\sum_{n=1}^{N} f_\theta^2(\boldsymbol{x}_n) \approx \sum_{l=1}^{L} f_\theta^2(\boldsymbol{z}_l)\,\frac{|\mathcal{X}_{\boldsymbol{d}}(\boldsymbol{z}_l)|}{N}\frac{1}{|P_{\boldsymbol{d}}(\boldsymbol{z}_l)|}.$$

If the function f_θ is constant on each of the elementary intervals provided by the fixed partition, equality is achieved, since $|\mathcal{X}_{\boldsymbol{d}}(\boldsymbol{z}_l)|$ is the number of data points from $\mathcal{X}$, which are in $I_d(\boldsymbol{z}_l)$, and $|P_{\boldsymbol{d}}(\boldsymbol{z}_l)|$ is the number of compressed data points from P, which are in $I_d(\boldsymbol{z}_l)$. Consequently, dividing by this number removes multiple counting. For regular functions, the approximation should perform well. Conversely, the greater the variation in function values on each interval, the worse the approximation will be.

Rather than focusing on a single partition, [5] considers all partitions with volume $b^{-\nu}$. Therefore, we must average over all partitions $\boldsymbol{d} \in \mathbb{N}_0^s$ with $|\boldsymbol{d}| = \nu$ and apply Lemma 3.2:

$$\frac{1}{N}\sum_{n=1}^{N} f_\theta^2(\boldsymbol{x}_n) \approx \sum_{l=1}^{L} f_\theta^2(\boldsymbol{z}_l) \underbrace{\sum_{q=0}^{\min(s-1,\nu)} (-1)^q \binom{s-1}{q} \sum_{\substack{\boldsymbol{d}\in\mathbb{N}_0^s \\ |\boldsymbol{d}|=\nu-q}} \frac{|\mathcal{X}_{\boldsymbol{d}}(\boldsymbol{z}_l)|}{N}\frac{1}{|P_{\boldsymbol{d}}(\boldsymbol{z}_l)|}}_{=W_{\mathcal{X},P,\nu,l}}.$$

The quality of this approximation depends strongly on P and ν, and its efficacy will be evaluated at a later stage. For the time being, we merely motivate the selection of weights. In the above equation the ratio $\frac{|\mathcal{X}_{\boldsymbol{d}}(\boldsymbol{z}_l)|}{|P_{\boldsymbol{d}}(\boldsymbol{z}_l)|}$ reflects the relevance of $\boldsymbol{z}_l$. The greater the number of points in the set $\mathcal{X}$ that fall within the same elementary interval as the point $\boldsymbol{z}_l$ for a multitude of partitions, the higher the weight assigned to $\boldsymbol{z}_l$. Consequently, the quantity $f_\theta^2(\boldsymbol{z}_l)$ will be of greater significance in the approximation.

Assuming that P is a (t_α, m, s)-net in base b of order α, we can exploit the (t_α, m, s)-net property, namely the fact that $\left|P_{\boldsymbol{a},\boldsymbol{d}}\right| = b^{m-\nu+q}$, where $|\boldsymbol{d}| = \nu - q$ for $q \in \{0, \ldots, \min(s-1, \nu)\}$. This simplifies the weights to the following expression given in [5]:

$$W_{\mathcal{X},P,\nu,l} = \frac{b^{\nu-m}}{N}\sum_{q=0}^{\min(s-1,\nu)} (-1)^q \binom{s-1}{q}\frac{1}{b^q}\sum_{\substack{\boldsymbol{d}\in\mathbb{N}_0^s \\ |\boldsymbol{d}|=\nu-q}} |\mathcal{X}_{\boldsymbol{d}}(\boldsymbol{z}_l)|. \tag{3.1}$$

Note that in order to use this representation of the weights, it is necessary to ensure that $m - t_\alpha \geq \nu$.

The derivation of the weights $\left\{W_{\mathcal{X},\mathcal{Y},P,\nu,l}\right\}_{l=1}^{L}$ is analogous. Once again, we consider a fixed partition $\boldsymbol{d} \in \mathbb{N}_0^s$ first and obtain an approximation:

$$\frac{1}{N}\sum_{n=1}^{N} y_n f_\theta(\boldsymbol{x}_n) \approx \sum_{l=1}^{L} f_\theta(\boldsymbol{z}_l)\,\frac{1}{N}\frac{1}{|P_{\boldsymbol{d}}(\boldsymbol{z}_l)|}\sum_{\substack{n=1 \\ \boldsymbol{x}_n\in I_{\boldsymbol{d}}(\boldsymbol{z}_l)}}^{N} y_n.$$

Considering all possible partitions with a volume of $b^{-\nu}$, the following result is obtained:

$$\frac{1}{N}\sum_{n=1}^{N} y_n f_\theta(\boldsymbol{x}_n) \approx \sum_{l=1}^{L} f_\theta(\boldsymbol{z}_l) \underbrace{\sum_{q=0}^{\min(s-1,\nu)} (-1)^q \binom{s-1}{q} \sum_{\substack{\boldsymbol{d}\in\mathbb{N}_0^s \\ |\boldsymbol{d}|=\nu-q}} \frac{1}{N}\frac{1}{|P_{\boldsymbol{d}}(\boldsymbol{z}_l)|} \sum_{\substack{n=1 \\ \boldsymbol{x}_n\in I_{\boldsymbol{d}}(\boldsymbol{z}_l)}}^{N} y_n}_{=W_{\mathcal{X},\mathcal{Y},P,\nu,l}}.$$

Once more, the quantity $\frac{1}{|P_{\boldsymbol{d}}(\boldsymbol{z}_l)|}\sum_{n=1,\boldsymbol{x}_n\in I_{\boldsymbol{d}}(\boldsymbol{z}_l)}^{N} y_n$ represents the relevance of $\boldsymbol{z}_l$ in the same manner as previously. Concurrently, it shall approximate y_n. The greater the number of points in the same elementary interval as $\boldsymbol{z}_l$ and the larger the response values y_n, the greater the influence of $f_\theta(\boldsymbol{z}_l)$ in the approximation.

Again, if P is a (t_α, m, s)-net in base b of order α and if $m - t_\alpha \geq \nu$, then the weights simplify to

$$W_{\mathcal{X},\mathcal{Y},P,\nu,l} = \frac{b^{\nu-m}}{N} \sum_{q=0}^{\min(s-1,\nu)} (-1)^q \binom{s-1}{q} \frac{1}{b^q} \sum_{\substack{\boldsymbol{d}\in\mathbb{N}_0^s \\ |\boldsymbol{d}|=\nu-q}} \sum_{\boldsymbol{x}_n\in I_{\boldsymbol{d}}(\boldsymbol{z}_l)}^{N} y_n. \tag{3.2}$$

Figure 2 is a modified version of Fig. 1. The red crosses represent $\mathcal{X}$, the blue points P. For a given blue point, the ratio of points in $\mathcal{X}$ to points in P for that elementary interval is calculated, which contains the blue point. It is necessary to consider all possible partitions, as the ratio in question can vary considerably depending on the partition. This is illustrated in Fig. 2.

3.1.1 Construction of digital nets

In this subsection, we will give a brief outline for the construction of the aforementioned (t_α, m, s)-nets. If constructed by the following methodology going back to Niederreiter's work [21], they are referred to as digital nets. We define the finite field with b elements,

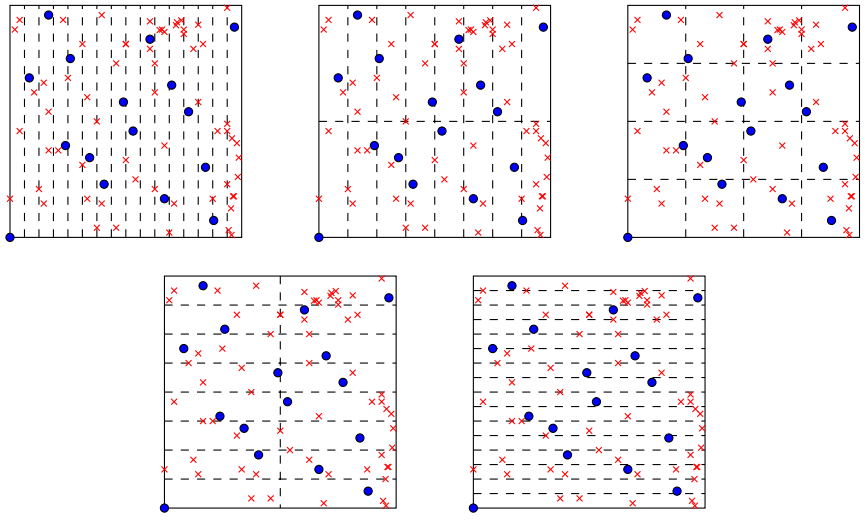

Figure 2 (0, 4, 2)-net in base 2 (blue points) with point set $\mathcal{X}$ (red crosses)

denoted by $\mathbb{Z}_b$, as the set $\{0,1,\dots,b-1\}$. The addition and multiplication operations are performed modulo b for which we use the notation $x \pmod b$.

We start with the construction of (t_1,m,s)-nets. Let $C_1,\dots,C_s \in \mathbb{Z}_b^{m\times m}$ be quadratic matrices of size m, which will determine the (t_1,m,s)-net. Then the j-th component $z_{l,j}$ of the l-th point of the net is given as follows: Assume that the b-adic expansion of $l-1$ is given by $\sum_{i=0}^{m-1}\lambda_i b^i$ and define the vector $\vec{\lambda} := (\lambda_0,\dots,\lambda_{m-1})^\intercal \in \mathbb{Z}_b^m$. Furthermore, let the vector $\vec{z}_{l,j} := \left(z_{l,j,1},\dots,z_{l,j,m}\right)^\intercal \in \mathbb{Z}_b^m$ be given by $\vec{z}_{l,j} \equiv C_j\vec{\lambda} \pmod b$. The final $z_{l,j}$ then is

$$z_{l,j} = \frac{z_{l,j,1}}{b} + \cdots + \frac{z_{l,j,m}}{b^m} = \sum_{i=1}^{m} \frac{z_{l,j,i}}{b^i}.$$

The determining matrices $C_1,\dots,C_s$ must have suitable properties in order to obtain (t_α,m,s)-nets. An illustrative example is the case of the Faure matrices, which are defined as follows:

$$C_j := \begin{pmatrix} \binom{1}{1}j^{1-1} & \binom{2}{1}j^{2-1} & \binom{3}{1}j^{3-1} & \cdots & \binom{m}{1}j^{m-1} \\ \binom{1}{2}j^{1-2} & \binom{2}{2}j^{2-2} & \binom{3}{2}j^{3-2} & \cdots & \binom{m}{2}j^{m-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \binom{1}{m}j^{1-m} & \binom{2}{m}j^{2-m} & \binom{3}{m}j^{3-m} & \cdots & \binom{m}{m}j^{m-m} \end{pmatrix} \pmod b.$$

Since $\binom{n}{k} = 0$ for $k > n$, these are upper triangular matrices with ones on the diagonal.

Based on the determining matrices of $(t_1,m,\alpha s)$-nets, we can construct (t_α,m,s)-nets for $\alpha \geq 2$ in the following way. Let $C_1,\dots,C_{\alpha s} \in \mathbb{Z}_b^{m\times m}$ be αs-many quadratic matrices of size m, which determine a $(t_1,m,\alpha s)$-net. For the ν-th row of C_u we will write $(C_u)_\nu$, where $u = 1,\dots,\alpha s$ and $\nu = 1,\dots,m$. We then construct the matrices D_j, that determine a (t_α,m,s)-net, as

$$D_j := \begin{pmatrix} (C_{(j-1)\alpha+1})_1 \\ \vdots \\ (C_{j\alpha})_1 \\ \vdots \\ (C_{(j-1)\alpha+1})_m \\ \vdots \\ (C_{j\alpha})_m \end{pmatrix}, \quad j = 1,\dots,s.$$

Similarly to the previous case, for each integer $l \in \{1,\dots,b^m\}$, we define the vector $\vec{z}_{l,j} := D_j\vec{\lambda} \pmod b$, with $\vec{\lambda} := (\lambda_0,\dots,\lambda_{m-1})^\intercal \in \mathbb{Z}_b^m$ consisting of the coefficients of the b-adic expansion $l-1 = \sum_{i=0}^{m-1}\lambda_i b^i$. Using $\vec{z}_{l,j} = \left(z_{l,j,1},\dots,z_{l,j,\alpha m}\right)^\intercal \in \mathbb{Z}_b^m$, we finally obtain

$$z_{l,j} = \frac{z_{l,j,1}}{b} + \cdots + \frac{z_{l,j,\alpha m}}{b^{\alpha m}} = \sum_{i=1}^{\alpha m} \frac{z_{l,j,i}}{b^i}.$$

In the following we will use (t_1,m,s)-nets generated by Sobol or Niederreiter-Xing matrices [15, 23].

3.1.2 Error bounds

While the formulas (3.1) and (3.2) for the weights look computationally heavy, an advantage of the QMC-averaging method is the availability of a rigorous (determinstic) error bound.

To state these bounds, we need to introduce the following norms:

Definition 3.3 (p.12, p.14, [5])

(i) For a continuous function $g : [0,1]^s \to \mathbb{R}$, which is λ_s-almost everywhere once partially differentiable with respect to each component, we define

$$\|g\| := \sum_{\boldsymbol{u}\subseteq\{1,\dots,s\}} \int_{[0,1]^{|\boldsymbol{u}|}} \left|\partial_{\boldsymbol{z}_{\boldsymbol{u}}} g\left(\boldsymbol{z}_{\boldsymbol{u}}, \mathbf{1}_{-u}\right)\right| \, \mathrm{d}\boldsymbol{z}_{\boldsymbol{u}}.$$

Here, $\partial_{\boldsymbol{z}_{\boldsymbol{u}}} g\left(\boldsymbol{z}_{\boldsymbol{u}}, \mathbf{1}_{-\boldsymbol{u}}\right)$ denotes the partial mixed derivative of order one for components in $\boldsymbol{u}$. The vector $\left(\boldsymbol{z}_{\boldsymbol{u}}, \mathbf{1}_{-\boldsymbol{u}}\right)$ for an arbitrary $\boldsymbol{u} \subseteq \{1,\dots,s\}$ is $\boldsymbol{z}_j$ in its j-th component, if $j \in \boldsymbol{u}$, and 1 otherwise.

(ii) Let $\alpha \geq 2$ and $1 \leq p < \infty$ be integer. The p, α-norm of a function $g : [0,1]^s \to \mathbb{R}$, which is α-times partially differentiable with respect to each component, is given by

$$\|g\|_{p,\alpha}^p := \sum_{\boldsymbol{u}\subseteq\{1,\dots,s\}} \sum_{\boldsymbol{v}\subseteq\boldsymbol{u}} \sum_{\boldsymbol{\tau}\in\{1,\dots,\alpha-1\}^{|\boldsymbol{u}\setminus\boldsymbol{v}|}} \int_{[0,1]^{|\boldsymbol{v}|}} \left| \int_{[0,1]^{s-|\boldsymbol{v}|}} \left(\prod_{j\in\boldsymbol{v}} \partial_{z_j}^{\alpha} \prod_{j\in\boldsymbol{u}\setminus\boldsymbol{v}} \partial_{z_j}^{\tau_j} \right) g\left(\boldsymbol{z}\right) d\boldsymbol{z}_{\{1,\dots,s\}\setminus\boldsymbol{v}} \right|^p \times d\boldsymbol{z}_{\boldsymbol{v}}.$$

Here, $\left(\prod_{j\in\boldsymbol{v}} \partial_{z_j}^{\alpha} \prod_{j\in\boldsymbol{u}\setminus\boldsymbol{v}} \partial_{z_j}^{\tau_j}\right) g(\boldsymbol{z})$ denotes the partial mixed derivative of order α or τ_j in the j-th coordinate.

Both norms appear naturally when dealing with Quasi-Monte Carlo integration, see e.g. [6, 22].

In [5] several error bounds are given. Their proofs are based on the Koksma-Hlawka inequality (2.1) and on a Walsh-series analysis (Appendix A in [6]). Optimizing the choice of ν with respect to m and t_α yields the following bounds (see Sect. 4.2 in [5]):

Theorem 3.4 (Corollary 12, [5]) *Let $P = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_L\}$ be a digital (t_1, m, s)-net in base b with $L = b^m$ and let $\nu = \frac{m}{2}$. Moreover, define $y_{\max} := \max_{n=1,\dots,N} |y_n|$. Then, there exists a constant $C_{s,b,t_1,y_{\max}} > 0$ such that*

$$\left|\mathrm{err}\left(f_\theta\right) - \mathrm{app}_L^{avg}\left(f_\theta\right)\right| \leq C_{s,b,t_1,y_{\max}} \left(\left\|f_\theta^2\right\| + \left\|f_\theta\right\| + \left\|f_\theta^2\right\|_{2,2} + \left\|f_\theta\right\|_{2,2} \right) \log_b(L)^{2s-1} L^{-\frac{1}{2}}. \tag{3.3}$$

We now turn our attention to the case in which the order α is at least 2.

Theorem 3.5 (Corollary 14, [5]) *Let $P = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_L\}$ be a digital (t_α, m, s)-net in base b of order $\alpha \geq 2$. Moreover, let $L = b^m$ and $\nu = \frac{\alpha}{\alpha+1} m$ and define $y_{\max} := \max_{n=1,\dots,N} |y_n|$. Then, there exists a constant $C_{\alpha,s,b,t_\alpha,y_{\max}} > 0$ such that*

$$\left|\mathrm{err}\left(f_\theta\right) - \mathrm{app}_L^{avg}\left(f_\theta\right)\right| \leq C_{\alpha,s,b,t_\alpha,y_{\max}} \left(\left\|f_\theta^2\right\|_{2,\alpha} + \left\|f_\theta\right\|_{2,\alpha} \right) \log_b(L)^{\alpha s} L^{-\frac{\alpha}{\alpha+1}}. \tag{3.4}$$

The convergence order in the case of t_1-nets is similar to Monte Carlo subsampling with order 1/2, but the error bound has the advantage of being non-random in comparison to Monte Carlo subsampling. For $\alpha \geq 2$ one even obtains the better convergence order $\alpha/(1+\alpha)$ (up to logarithmic terms).

3.1.3 Implementation

The most complex part of the implementation is clearly to calculate the weights (3.1) and (3.2). Here the quantities $|\mathcal{X}_{\boldsymbol{d}}(\boldsymbol{z}_l)|$ and $\sum_{n=1,\boldsymbol{x}_n\in I_{\boldsymbol{d}}(\boldsymbol{z}_l)}^{N} y_n$ have to be computed for each elementary interval with $|\boldsymbol{d}| = \nu - q$. In total, there are $\binom{\nu-q+s-1}{s-1}$ different vectors $\boldsymbol{d}$, that satisfy this property. Thus, for every value of l between 1 and b^m and for each value of $0 \leq q \leq \min(\nu, s-1)$, we need to determine the quantities $S_r(\boldsymbol{z}_l)$ and $T_r(\boldsymbol{z}_l)$ defined by

$$S_r(\boldsymbol{z}_l) := \sum_{\substack{\boldsymbol{d}\in\mathbb{N}_0^s \\ |\boldsymbol{d}|=r}} |\mathcal{X}_{\boldsymbol{d}}(\boldsymbol{z}_l)|, \quad T_r(\boldsymbol{z}_l) := \sum_{\substack{\boldsymbol{d}\in\mathbb{N}_0^s \\ |\boldsymbol{d}|=r}} \sum_{n=1}^{N} y_n \mathbb{1}_{I_{\boldsymbol{d}}(\boldsymbol{z}_l)}(\boldsymbol{x}_n),$$

where $r = \nu - q$. In fact, if $y_n = 1$ for $n = 1,\dots,N$, then the sum of indicator functions $\sum_{n=1}^{N} \mathbb{1}_{I_{\boldsymbol{d}}(\boldsymbol{z}_l)}$ counts the number of original data points, which are in the same elementary interval as $\boldsymbol{z}_l$ and we obtain $S_r(\boldsymbol{z})$.

The following algorithm computes $T_r(\boldsymbol{z}_l)$. In order to obtain $S_r(\boldsymbol{z}_l)$, one just sets $y_n = 1$ for $n = 1,\dots,N$.

The choice $i_j = 0$ in line 4 will occur if none of the coefficients of the b-adic expansions of z_j and $(x_n)_j$ are identical. The objective of Algorithm (W) is to identify the smallest s-dimensional elementary interval, which has a length of at least b^{-r} for each one-dimensional elementary interval contained within it, and which includes $\boldsymbol{z}$. This is accomplished for each data point $\boldsymbol{x}_n$. Given a specific $\boldsymbol{x}_n$, we count the number of vectors $\boldsymbol{d} \in \mathbb{N}_0^s$ with $|\boldsymbol{d}| = r$, such that $\boldsymbol{x}_n$ is part of $I_{\boldsymbol{d}}(\boldsymbol{z}_l)$. The numbers $N_{r,\boldsymbol{i}} := \#\left\{\boldsymbol{d} \in \mathbb{N}_0^s : |\boldsymbol{d}| = r, \boldsymbol{d} \leq \boldsymbol{i}\right\}$ are universal, i.e. independent of the data and regression function, and can be computed in advance. See [5] for a particular algorithm for this task, which leads to a total cost of $\mathcal{O}(rsN)$ for Algorithm (W) with a storage space of order $\mathcal{O}(\max(s,r))$.

In order to calculate the weights, the following equation must be used:

$$W_{\mathcal{X},\mathcal{Y},P,\nu,l} = \frac{b^{\nu-m}}{N} \sum_{q=0}^{\min(s-1,\nu)} (-1)^q \binom{s-1}{q} \frac{1}{b^q} T_{\nu-q}(\boldsymbol{z}_l).$$

Choosing $S_{\nu-q}(\boldsymbol{z}_l)$ instead of $T_{\nu-q}(\boldsymbol{z}_l)$ results in the weights $W_{\mathcal{X},P,\nu,l}$. The cost of computing all weights is of order $\mathcal{O}(b^m m s^2 N)$.

Algorithm (W) Calculation of $S_r(\boldsymbol{z})$ and $T_r(\boldsymbol{z})$ [5]

input: $\boldsymbol{z} \in [0,1)^s$, $\mathcal{X}$, $\mathcal{Y}$, b, $r \geq 0$
output: $T_r(\boldsymbol{z})$ (or $S_r(\boldsymbol{z})$ if $y_n = 1$ for $n = 1:N$)

```
1 set T_r(z) = 0
2 for n = 1:N
3     for j=1:s
4         find maximal i_j ∈ {0,...,r}, so the first i_j digits of the b-adic
5         expansion of z_j and (x_n)_j are equal
6     end
7     set i = (i_1,...,i_s) and T_r(z) = T_r(z) + y_n # {d ∈ N_0^s : |d| = r, d ≤ i}
8 end
```

As we mentioned in the beginning, the weights are independent of θ. Therefore, if $\mathcal{X}$, $\mathcal{Y}$ and P remain constant throughout the optimization process, it is sufficient to calculate the weights once at the outset. The procedure for fitting the neural network merely requires updating

$$\mathrm{app}_L^{avg}(f_\theta) = \sum_{l=0}^{L-1} f_\theta^2(\boldsymbol{z}_l)\, W_{\mathcal{X},P,v,l} - 2\sum_{l=0}^{L-1} f_\theta(\boldsymbol{z}_l)\, W_{\mathcal{X},\mathcal{Y},P,v,l} + \frac{1}{N}\sum_{n=1}^{N} y_n^2 \tag{3.5}$$

in each optimization step. Assuming that f_θ^2 and f_θ can be evaluated with cost $\mathcal{O}(s)$, we end up with cost of order $\mathcal{O}(Ls)$, rather than $\mathcal{O}(Ns)$, which is the cost if we use $\mathrm{err}\left(f_\theta\right)$. We require additional storage of $\mathcal{O}(L)$ for the weights and $\sum_{n=1}^{N} y_n^2$. This suggests that training the neural network with the compressed data set will be faster. However, it is not yet clear how much time will be saved in the whole optimization procedure, due to the additional effort required at the outset.

3.2 Supercompress method

A classical approach to deal with regression problems are K-nearest neighbors algorithms, which in their simplest form estimate the regression function at a point $\boldsymbol{z}$ by averaging the responses y_i of the K data points $\boldsymbol{x}_i$, which are the K nearest points to $\boldsymbol{z}$. See e.g. [3] Chaps. 2 and 14 in [10]. These algorithms are not data reduction methods themselves. However, data reduction can be achieved by incorporating clustering, which leads, e.g. to the supercompress method proposed by [14].

The aim of the supercompress method is again to find a compressed point set $P = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K\}$, but instead of weights, one obtains the corresponding (approximate) responses $\mathcal{W} = \{w_1, \ldots, w_K\}$. This is achieved through a specific K-means clustering, which does not employ a conventional clustering approach on the input space, but utilizes the responses $\mathcal{Y}$. More precisely, the supercompress method aims to find the points $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K\}$ such that the loss function

$$L = \sum_{j=1}^{K} L_j = \sum_{j=1}^{K} \sum_{i \in I_j} \left(y_i - w_j\right)^2 \tag{3.6}$$

with the approximate responses

$$w_j = \frac{1}{|I_j|} \sum_{i \in I_j} y_i, \quad j = 1, \ldots, K, \tag{3.7}$$

and the Voronoi-clusters

$$I_j = \left\{ i \in \{1, \ldots, N\} : \left\|\boldsymbol{z}_j - \boldsymbol{x}_i\right\|_2 \leq \left\|\boldsymbol{z}_{j'} - \boldsymbol{x}_i\right\|_2 \text{ for all } j' \neq j \right\}, \quad j = 1, \ldots, K, \tag{3.8}$$

is minimized. In contrast, the classical K-means algorithm on the input space would find the data points $\left\{\boldsymbol{z}'_1, \ldots, \boldsymbol{z}'_K\right\}$ and Voronoi-clusters $I'_1, \ldots, I'_K$, which minimize the loss

$$L' = \sum_{j=1}^{K} L'_j = \sum_{j=1}^{K} \sum_{i \in I'_j} \left\|\boldsymbol{z}'_i - \boldsymbol{x}_i\right\|^2.$$

Algorithm (S) Calculation of supercompress data compression [14]

input: $\mathcal{X}$, $\mathcal{Y}$, K
output: $P = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_K\}$, $\mathcal{W} = \{w_1, \dots, w_K\}$

set $P = \{\}$, $\mathcal{I} = \{\}$ and $\mathcal{W} = \{\}$
split $\mathcal{X}$ into 2 K–means clusters with centers $\{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ and partitions $\{I_1, I_2\}$
for $j = 1,2$ compute responses $w_j = \frac{1}{|I_j|}\sum_{i\in I_j} y_i$ and losses $L_j = \sum_{i\in I_j}(y_i - w_j)^2$
set $P = P \cup \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$, $\mathcal{I} = \mathcal{I} \cup \{I_1, I_2\}$ and $\mathcal{W} = \mathcal{W} \cup \{w_1, w_2\}$
for $k = 3{:}K$
    find index $j^\star = \max_{j=1,\dots,k-1} L_j$
    split cluster $j^\star$ into 2 K–means clusters with centers $\{\boldsymbol{z}_{j^\star}, \boldsymbol{z}_k\}$ and partitions $\{I_{j^\star}, I_k\}$
    for $j = j^\star, k$ update responses $w_j = \frac{1}{|I_j|}\sum_{i\in I_j} y_i$ and losses $L_j = \sum_{i\in I_j}(y_i - w_j)^2$
    set $P = P \cup \{\boldsymbol{z}_k\}$, $\mathcal{I} = \mathcal{I} \cup \{I_k\}$ and $\mathcal{W} = \mathcal{W} \cup \{w_k\}$ and update cluster $j^\star$
end

Although clustering on the input space is already known to be a NP-hard problem, see e.g. [1], many fast algorithm exist, which converge at least to a local minimum. See e.g. [2]. For practical implementation of the supercompress method, the authors of [14] propose the following iterative K-means algorithm (see Algorithm (S)).

The initial step involves partitioning the input space $\mathcal{X}$ into two clusters using a K-means approach. The loss for each cluster is then calculated according to the formula $L_j = \sum_{i\in I_j}(y_i - w_j)^2$. The cluster with the higher loss is divided into two clusters based on K-means on the input space once more. This results in two new clusters, each with a center and a corresponding response. These two new clusters replace the old cluster with the highest loss. The losses L_j for the two new clusters are updated. Then, the cluster with the greatest loss is divided into two clusters in the same way as before. This procedure is repeated until K clusters have been obtained. Each cluster has a center point, denoted by $\boldsymbol{z}_k$, which is part of the compressed data, and a corresponding response, denoted by w_k, which is contained in $\mathcal{W}$.

In [14] the authors propose another, more robust algorithm by taking the convex combination

$$\widetilde{L} = \lambda \sum_{j=1}^{K}\sum_{i\in I_j} \left\|\boldsymbol{z}_j - \boldsymbol{x}_i\right\|_2^2 + (1-\lambda)\sum_{j=1}^{K}\sum_{i\in I_j}\left(y_i - w_j\right)^2. \tag{3.9}$$

Here $\lambda \in [0,1]$ is a weight parameter quantifying the trade-off between the two terms. When $\lambda = 0$, we return to the case of the original supercompress algorithm. If $\lambda = 1$, the problem reduces to the traditional K-means problem. One can interpret this modified criterion as a trade-off between a fully supervised reduction strategy ($\lambda = 0$), which fully incorporates response information for data reduction, and a fully robust reduction strategy ($\lambda = 1$), which reduces the data using only the input feature information. In particular, a default choice of $\lambda = 1/(s+1)$ is suggested in [14]. This is motivated by the observation that the objective function $\widetilde{L}$ then becomes proportional to

$$\sum_{j=1}^{K}\sum_{i\in I_j}\frac{1}{s}\left\|\boldsymbol{z}_j - \boldsymbol{x}_i\right\|_2^2 + \sum_{j=1}^{K}\sum_{i\in I_j}\left(y_i - w_j\right)^2.$$

In order to optimize the new loss function $\widetilde{L}$, it is possible to use Algorithm (S) with a slight modification. This involves changing the y_i's to $\tilde{y}_i = y_i\sqrt{(1-\lambda)/\lambda}$, and exchanging

the loss functions L_j with

$$\widetilde{L}_j = \sum_{i \in I_j} \left(\left\| \boldsymbol{z}_j - \boldsymbol{x}_i \right\|_2^2 + \left(\tilde{y}_i - w_j \right)^2 \right).$$

We will refer to this more robust algorithm, which optimizes $\widetilde{L}$, as robust supercompress.

Error estimates for this supercompress method in terms of the number K of compressed points seem to be unknown. One can consider the standard K-means clustering as a sub-optimal solution of a discrete quantization problem. For the latter the error of the optimal quantization decays as $\mathfrak{o}(K^{-1/d})$. See e.g. Chapter II.6 in [9]. Thus, we can expect at best a similar behavior for the approximation of

$$\operatorname{err}\left(f_\theta\right) := \frac{1}{N} \sum_{n=1}^{N} \left(f_\theta\left(\boldsymbol{x}_n\right) - y_n \right)^2$$

by

$$\operatorname{app}_L^{clst}(f_\theta) := \frac{1}{K} \sum_{k=1}^{K} \left(f_\theta\left(\boldsymbol{z}_k\right) - w_k \right)^2.$$

3.3 QMC-Voronoi method

While testing the Quasi-Monte Carlo compression approach, we observed several phenomena. To understand their source, in particular whether they are caused by the weights or the digital nets, we tried a method which combines (unsupervised) clustering and the use of digital nets. We call this method QMC-Voronoi method.

To construct the compressed data set for the QMC-Voronoi method, we start with a digital net. Instead of calculating complex weights, we form clusters by using the Voronoi diagram, see e.g. Section I.1 in [9], on the digital net. This implies that for each data point $\boldsymbol{x}$, the closest point $\boldsymbol{z}$ of the digital net P is identified based on the Euclidean distance, and $\boldsymbol{x}$ is assigned to the Voronoi region of $\boldsymbol{z}$. Hence, P represents the compressed set. For each compressed point $\boldsymbol{z}$, the corresponding response w is found by taking the average of the corresponding responses of the data points $\boldsymbol{x}$, which are in the Voronoi region of $\boldsymbol{z}$. Consequently, as for the supercompress method the error $\operatorname{err}(f_\theta)$ can therefore be approximated by $\operatorname{app}_L^{clst}(f_\theta)$. The above procedure is summarized in Algorithm (V), where we take $K = b^m$ points for the compressed data set.

Algorithm (V) Calculation of QMC-Voronoi data compression

input: $\mathcal{X}$, $\mathcal{Y}$, b, m
output: $P \subseteq \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K\}$, $\mathcal{W} = \{w_1, \ldots, w_K\}$

```
create digital net P = {z_1, ..., z_K}
for i = 1:N
    find closest point z ∈ P to x_i and asign y_i to z
end
for k = 1:K
    if at least one y_i got asigned to z_k
        compute w_k as the average of each response y_i, which is asigned to z_k
    else
        delete z_k from P
    end
end
```

This method can be seen as a specific clustering on the input space $\mathcal{X}$ with prescribed centers of the clusters, so we expect a behavior of the error similar or worse than for the (robust) supercompression method.

4 Numerical results

Finally, we undertake a numerical comparison of the methods. We start by considering three fixed functions f and compare $\mathrm{err}(f)$ with $\mathrm{app}_L^{avg}(f_\theta)$ and $\mathrm{app}_L^{clst}(f_\theta)$, respectively. Subsequently, we train a neural network using the MNIST dataset [16] and compare the accuracy achieved with the different compression methods. All implementations can be found in our Git repository [11].

4.1 Test functions

The following functions are taken from the virtual library [24]. For an input point $\boldsymbol{x} = (x_1, \dots, x_s) \in [0,1)^s$ and vectors $\boldsymbol{a} = (a_1, \dots, a_s) \in \mathbb{R}^s$ and $\boldsymbol{u} = (u_1, \dots, u_s) \in [0,1)^s$ we define

$$f_1(\boldsymbol{x}) := \exp\left(-\sum_{i=1}^{s} a_i |x_i - u_i|\right). \tag{4.1}$$

This function satisfies $\|f_1\| < \infty$ as well as $V(f_1) < \infty$, but has no higher regularity w.r.t. $\|\cdot\|_{p,\alpha}$ for $\alpha \geq 2$, due to the involved absolute values. The second test function is given by

$$f_2(\boldsymbol{x}) := \begin{cases} 0 & \text{if } x_1 > u_1 \text{ or } x_2 > u_2, \\ \exp\left(\sum_{i=1}^{s} a_i x_i\right) & \text{otherwise.} \end{cases} \tag{4.2}$$

This function is discontinuous, so it has no regularity w.r.t. $\|\cdot\|_{p,\alpha}$ and $\|\cdot\|$, but still has finite Hardy-Krause variation, i.e. $V(f_2) < \infty$. Finally, the so-called Zhou function is given by

$$f_3(\boldsymbol{x}) := \frac{10^s}{2}\left(\phi\left(10\boldsymbol{x} - \frac{10}{3}\right) + \phi\left(10\boldsymbol{x} - \frac{20}{3}\right)\right), \tag{4.3}$$

where

$$\phi(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{s}{2}}} \exp\left(-\frac{\|\boldsymbol{x}\|^2}{2}\right).$$

This function satisfies $V(f_3) < \infty$, $\|f_3\| < \infty$ as well as $\|f_3\|_{p,\alpha} < \infty$ for any $p \geq 1$ and $\alpha \geq 2$.

Note that these functions have different scales. For example, for $s = 2$ the second function attains values between zero and $\max\{1, \exp(a_1 u_1)\} \max\{1, \exp(a_2 u_2)\}$, while the first one takes values between zero and one only. Finally, Zhou's function contains values between zero and at most 7.9579. In the following we will choose $u_1 = \cdots = u_s = 0.5$ so that f_1 is symmetric around the center of the unit cube and the discontinuity of f_2 appears at the same position. Furthermore we set $a_1 = \cdots = a_s = 5$. Thus the maximum value of f_2 for $s = 2$ is $(\exp(2.5))^2 \approx 148.41$, which is much larger than that of f_1 and f_3. For more information and illustrations of these functions, see [24].

For evaluating the different compression methods, a total of $N = 3000$ uniformly distributed points are sampled in the interval $[0,1)^s$. For each point, the function is evaluated and an independent perturbation from the $\mathcal{N}(0, 0.02)$-distribution is added. Subsequently, the compressed point set P is generated with $L = K$ points, where L is selected

Table 1 average error (top row supercompress, bottom row QMC-averaging)

Compression rate	N	s	K	f_1	f_2	f_3
1%	3000	2	2^5	$3,1054 \cdot 10^{-4}$ $2,24 \cdot 10^{-2}$	$2,2135 \cdot 10^{4}$ $4,4712 \cdot 10^{4}$	$4,86 \cdot 10^{-2}$ $3,5859 \cdot 10^{0}$
4%	3000	2	2^7	$3,3239 \cdot 10^{-4}$ $1,37 \cdot 10^{-2}$	$1,2331 \cdot 10^{3}$ $2,0453 \cdot 10^{4}$	$2,5 \cdot 10^{-3}$ $2,0974 \cdot 10^{0}$
9%	3000	2	2^8	$2,5366 \cdot 10^{-4}$ $1,9 \cdot 10^{-3}$	$2,9351 \cdot 10^{2}$ $6,9552 \cdot 10^{3}$	$5,98 \cdot 10^{-4}$ $1,1437 \cdot 10^{0}$
17%	3000	2	2^9	$1,7051 \cdot 10^{-4}$ $1,9 \cdot 10^{-3}$	$8,7618 \cdot 10^{1}$ $6,4545 \cdot 10^{3}$	$1,2786 \cdot 10^{-4}$ $1,1702 \cdot 10^{0}$
34%	3000	2	2^{10}	$1,0473 \cdot 10^{-4}$ $2 \cdot 10^{-3}$	$1,9970 \cdot 10^{1}$ $8,2950 \cdot 10^{3}$	$1,5783 \cdot 10^{-5}$ $4,878 \cdot 10^{-1}$

Table 2 average error for f_2 for differently scaled function values

Compression rate	N	s	K	No scale	Scale 1
9%	3000	2	2^8	$2,9351 \cdot 10^{2}$ $6,9552 \cdot 10^{3}$	$1,3326 \cdot 10^{-2}$ $3,1577 \cdot 10^{-1}$
17%	3000	2	2^9	$8,7618 \cdot 10^{1}$ $6,4545 \cdot 10^{3}$	$3,9779 \cdot 10^{-3}$ $2,9304 \cdot 10^{-1}$

from the set $\{2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$. Finally, in Table 1 we compare the approximation error $\left|\mathrm{err}(f) - \mathrm{app}_K^{clst}(f)\right|$ of the supercompress method to the one of the QMC-averaging method, i.e. $\left|\mathrm{err}(f) - \mathrm{app}_L^{avg}(f)\right|$, for different compression rates K/N. This is repeated 100 times and the observed values are averaged in order to enhance the robustness of the result. For the QMC-averaging method we use the "Magic Point Shop" [15, 23] to construct the set P. In fact, we utilize Niederreiter-Xing matrices of order 1, as presented in [15, 23].

Table 1 illustrates the results for the dimension $s = 2$. The top row expresses the average error of the supercompress algorithm, while the bottom row refers to that of the QMC-averaging method. It can be observed that the supercompress algorithm performs better for each function. Only for the discontinuous function f_2 and high compression rates, i.e. small K, both methods perform equally bad. Since $f_2([0,1]^2) = [0, \exp(5)]$, and $\exp(5) \approx 148,4132$, the observed behavior might be a consequence of the lack of scaling. To analyze this, we repeated the procedure for $\gamma f_2 / \|f_2\|_\infty$ with different scales γ. In Table 2 we see, that both methods yield a smaller error for $\gamma = 1$, which corresponds to standardized test functions. Thus, both methods behave similar for all test functions when the latter are properly scaled; the supercompress method seems to be always the superior choice. Interestingly, the regularity of the test functions does not seem to have a big influence on the performance of the QMC-averaging method. You can find a scaled version with $\gamma = 1$ of all tables regarding the error analysis in the Appendix (Table 8, Table 9, Table 10).

We also performed the same numerical experiment without adding noisy perturbations. The results are found to be consistent with those above, indicating that adding noise does not significantly influence the comparison. In fact, as long as the noise level of the data is reasonably low, the effects of the above analysis remain the same.

The sensitivity of both methods in terms of the dimension is presented in Table 3. For f_1, the size of the problem does not affect the accuracy of our methods. However, it can be observed that there is a deterioration in accuracy for the other functions. In particular, the error increases significantly in higher dimensions when considering f_2. Note that for the largest dimension $s = 10$, QMC-averaging leads to smaller errors for f_1 and f_3 than

Table 3 average error (top row supercompress, bottom row QMC-averaging)

Compression rate	N	s	K	f_1	f_2	f_3
9%	3000	2	2^8	$2,5366 \cdot 10^{-4}$	$2,9352 \cdot 10^{2}$	$5,98 \cdot 10^{-4}$
				$1,9 \cdot 10^{-3}$	$6,9552 \cdot 10^{3}$	$1,1437 \cdot 10^{0}$
9%	3000	3	2^8	$2,6782 \cdot 10^{-4}$	$3,8841 \cdot 10^{6}$	$5,03 \cdot 10^{-2}$
				$1,9 \cdot 10^{-3}$	$7,2364 \cdot 10^{7}$	$7,7694 \cdot 10^{0}$
9%	3000	5	2^8	$3,286 \cdot 10^{-4}$	$4,3730 \cdot 10^{13}$	$1,4568 \cdot 10^{1}$
				$2,3514 \cdot 10^{-4}$	$6,8260 \cdot 10^{14}$	$8,9822 \cdot 10^{1}$
9%	3000	10	2^8	$3,5688 \cdot 10^{-4}$	$1,0240 \cdot 10^{29}$	$2,9730 \cdot 10^{5}$
				$5,0307 \cdot 10^{-8}$	$8,2610 \cdot 10^{31}$	$7,4510 \cdot 10^{3}$

Table 4 average error (top row QMC-averaging, bottom row QMC-Voronoi)

Compression rate	N	s	K	f_1	f_2	f_3
4%	3000	2	2^7	$1,37 \cdot 10^{-2}$	$2,0453 \cdot 10^{4}$	$2,0974 \cdot 10^{0}$
				$4,4655 \cdot 10^{-4}$	$2,6420 \cdot 10^{3}$	$6,92 \cdot 10^{-2}$
9%	3000	2	2^8	$1,9 \cdot 10^{-3}$	$6,9552 \cdot 10^{3}$	$1,1437 \cdot 10^{0}$
				$7,0245 \cdot 10^{-5}$	$4,6737 \cdot 10^{2}$	$3,36 \cdot 10^{-2}$
17%	3000	2	2^9	$1,9 \cdot 10^{-3}$	$6,4545 \cdot 10^{3}$	$1,1702 \cdot 10^{0}$
				$1,7547 \cdot 10^{-4}$	$1,3446 \cdot 10^{3}$	$2,41 \cdot 10^{-2}$

supercompress. We do not have a mathematical explanation at hand and did not explore this phenomenon further due to the very high running time of QMC-averaging for larger s; see also Table 5.

In order to gain further insight into the behavior of QMC-averaging, we designed the QMC-Voronoi method and conducted a comparison between its performance and that of the QMC-averaging method of [5]. The results can be seen in Table 4. The upper row displays the results for the QMC-averaging method, while the bottom row presents the average error regarding the QMC-Voronoi method. The setup of this experiment is the same as the one for the comparison of QMC-averaging and supercompress.

The results of QMC-Voronoi consistently outperform those of the QMC-averaging method in all scenarios. In fact, its performance is not too far away from the one of the supercompress method. This indicates that clustering algorithms are more effective for the considered problem.

Additionally, a significant drawback of the QMC-averaging method is its complex calculation. In order to include all partitions of the unit cube into elementary intervals, we use alternating sums with binomial weights, which may result in significant numerical instability. At the same time, we perform an approximation for every partition, which leads to a possible accumulation of errors. The QMC-Voronoi approach considers only one partition of the unit cube, which is more flexible than the one based on elementary intervals.

To underline this observation, we evaluated the running time required to compute the compressed data sets. A total of $N = 10{,}000$ data points are sampled uniformly at random from the interval $[0,1)^s$. The corresponding responses are determined by the function f_2 given by (4.2). This procedure is repeated 20 times and the mean is taken. The results are presented in Table 5. The dimension s of the space is varied in the first column and the number of compressed points K is varied in the first row.

Although the supercompress algorithm performs the best, it does not take the most time. In fact, it is the fastest algorithm for low K. We observe that the QMC-averaging algorithm takes the most time. As previously indicated, this is caused by the calculation of

Table 5 average running time for data compression in seconds (top row supercompress, middle row QMC-averaging, bottom row QMC-Voronoi)

$s \backslash K$	2^6	2^8	2^{10}	2^{12}
2	$5{,}7 \cdot 10^{-3}$	$1{,}69 \cdot 10^{-2}$	$5{,}35 \cdot 10^{-2}$	$1{,}980 \cdot 10^{-1}$
	$1{,}813 \cdot 10^{-1}$	$8{,}122 \cdot 10^{-1}$	$3{,}2312 \cdot 10^{0}$	$1{,}3529 \cdot 10^{1}$
	$5{,}53 \cdot 10^{-2}$	$6{,}34 \cdot 10^{-2}$	$7{,}36 \cdot 10^{-2}$	$7{,}97 \cdot 10^{-2}$
3	$6{,}3 \cdot 10^{-3}$	$1{,}85 \cdot 10^{-2}$	$5{,}39 \cdot 10^{-2}$	$1{,}957 \cdot 10^{-1}$
	$3{,}257 \cdot 10^{-1}$	$1{,}5233 \cdot 10^{0}$	$6{,}1759 \cdot 10^{0}$	$2{,}5003 \cdot 10^{1}$
	$5{,}47 \cdot 10^{-2}$	$5{,}87 \cdot 10^{-2}$	$6{,}49 \cdot 10^{-2}$	$8{,}10 \cdot 10^{-2}$
5	$5{,}6 \cdot 10^{-3}$	$1{,}71 \cdot 10^{-2}$	$5{,}44 \cdot 10^{-2}$	$1{,}979 \cdot 10^{-1}$
	$5{,}735 \cdot 10^{-1}$	$3{,}0826 \cdot 10^{0}$	$1{,}2265 \cdot 10^{1}$	$5{,}3843 \cdot 10^{1}$
	$5{,}56 \cdot 10^{-2}$	$6{,}42 \cdot 10^{-2}$	$7{,}53 \cdot 10^{-2}$	$9{,}23 \cdot 10^{-2}$

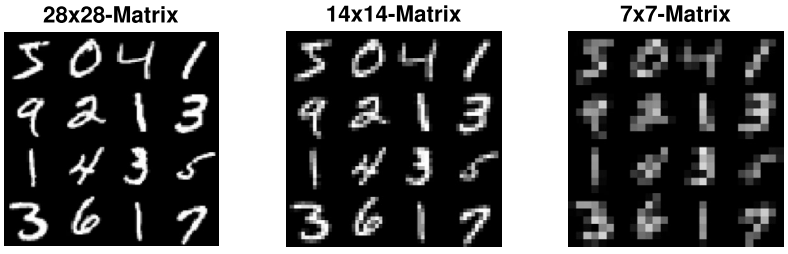

Figure 3 visualization of the precompressed data

the weights, which averages over many different partitions. Additionally, the supercompress and QMC-Voronoi algorithms are more robust regarding the dimension. Even for a slight increase in s, the QMC-averaging algorithm doubles its running time. For large values of K, the QMC-Voronoi method is computed at a faster rate than the supercompress method. Furthermore, the cost of the QMC-Voronoi method does not increase at the same rate as the supercompress method in relation to K.

4.2 Neural networks

Thus far, we have considered only uniformly random data and a rather simple connection between data points and responses. However, in general this is not the case. To illustrate this, we will now examine the MNIST data set, which can be found in [16]. This data set contains $6 \cdot 10^4$ grey-scale images of handwritten numbers between 0 and 9. Each image is represented by a matrix of dimensions (28×28), with entries from 0 to 1. The value of each entry indicates the darkness of the corresponding pixel. By concatenating all the pixels in a given image, we obtain the image of the handwritten number. Consequently, the set $\mathcal{X}$ contains the pixel sets,[1] while the set $\mathcal{Y}$ contains the corresponding handwritten numbers. Figure 3 illustrates 16 examples of such pixel sets. In the case of an entry of 0, the pixel is depicted as black.

Our aim is to train a neural network with the data in question. To compare the different methods, the accuracies of the trained neural networks will be evaluated on a test data set, which is part of the MNIST data set.

It should be noted that the dimension of a matrix with dimensions (28×28) is much larger than the dimensions of the points that have been considered thus far. In order to

[1] Here we identify 1 as $1 - \texttt{eps}$, where eps is the machine accuracy. Thus, we can still work with the assumption $\mathcal{X} \subset [0,1)^s$.

reduce the running time, particularly that required for calculating the weights, we implement a precompression strategy. Initially, we limit our consideration to the first $N = 10^4$ grey-scale images. Additionally, we reduce the dimension by transforming each (28×28)-matrix to a (14×14)-matrix. This results in a reduction of the dimension from $s = 784$ to $s = 196$. The idea is to take a submatrix of a pixel set with dimensions of two by two and represent it by its average value. Figure 3 illustrates the impact of this precompression on the data. On the left, we have examples of the original data. In the middle, the (2×2)-submatrices are exchanged by their average value, while the same process occurs for (4×4)-submatrices on the right.

It can be observed that, despite the increased pixelation, the (14×14)-matrices can still be read, at least to some extent. Further compression of the data to (7×7)-matrices renders it more challenging for the human eye to discern the individual values. This is the reason why we opted to train the network with (14×14)-matrices.

After these preparations, the neural network can be trained. We start by compressing the data in accordance with the compression algorithms established in Sect. 3. Afterwards, the neural network is trained with the compressed set. In order to have a broad overview, we will use the QMC-averaging method, the (robust) supercompress method, the QMC-Voronoi method and the traditional K-means clustering. In fact, K-means and (robust) supercompress appear to be similar in many respects. However, there is a key difference between them in the way they consider the image space. K-means clusters solely based on the $\boldsymbol{x}$-space, whereas robust supercompress attempts to combine clustering based on the input space and the y-space. The normal supercompress algorithm is focused on finding clusters based on the output space.

In this instance, the dimension of the input data is too large to utilise Niederreiter-Xing matrices. Consequently, Sobol matrices [15, 23] are employed as the construction matrices of the digital net, with ν set to 2. The design of the neural network and its training are based on [13] with 100 epochs. For clustering algorithms, this procedure can be employed without modification. The QMC-averaging method requires adjustments to the neural network, as it must be trained based on points and weights instead of points and corresponding responses.

At the end of each training, the final loss and the required time (including compression and training) are observed. Additionally, we test the neural network to obtain the accuracy (the relative number of times it predicts the correct number) and a confusion chart. In order to assess the efficacy of the compression method, we initially train the neural network with the uncompressed data. The resulting confusion chart is presented in Fig. 4.

If our prediction is accurate, the result will be on the diagonal. The number indicates the number of times the neural network predicts the class on the x-axis, while the class on the y-axis is the correct one. For example, 1 is predicted correctly 1101 times. The least accurate combination is 9 and 4. The neural network predicts 52 times a 9, despite the real written number being a 4. It is evident that the objective is to maximize the numbers on the diagonal.

A comparison of the performance of the compression algorithms with that of the uncompressed training is presented in Table 6. The table shows the accuracy, the required time (i.e. 12 : 34 corresponds to 12 minutes and 34 seconds), and the final loss, with the exception of the QMC-averaging method, for which the complex input structure (weights instead of corresponding responses) does not allow a comparable loss value. The num-

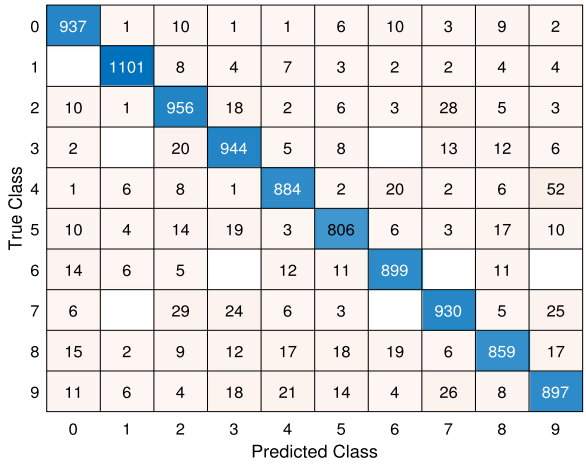

Figure 4 confusion chart of the neural network without compression

Table 6 comparison of the different compression methods for the training of a neural network

Method	Compression rate	Accuracy	Time [min]	Loss
no compression	–	92,13%	13 : 00	0,0215
supercompress	5%	55,15%	0 : 57	0,3533
	10%	72,68%	1 : 26	0,2799
	20%	88,16%	2 : 45	0,164
	40%	91,98%	6 : 47	0,1126
robust supercompress	5%	53,45%	0 : 44	0,2748
	10%	55,33%	1 : 24	0,2389
	20%	61,51%	2 : 46	0,1678
	40%	64,89%	5 : 23	0,1165
QMC	5%	8,92%	14 : 28	–
	10%	10,28%	29 : 08	–
	20%	10,09%	59 : 02	–
K-means	5%	11,01%	1 : 57	0,4367
	10%	9,2%	3 : 58	0,4457
	20%	11,22%	7 : 37	0,4513

ber of compressed points $L = K$ is varied for values in the set $\{2^9, 2^{10}, 2^{11}, 2^{12}\}$. We have dropped the results of QMC-averaging and K-means for $L = K = 2^{12}$, due to no significant improvement in accuracy.

In general, supercompress performs the best. Its normal version is more accurate than the robust one, though it takes a bit longer. QMC-averaging and K-means are very poor. It seems that they are as poor as guessing. Additionally, they take longer, especially QMC-averaging. This is caused by the high effort for precalculating the weights. Consequently, they are not suitable for this compression problem. Why is that so?

We observed a striking phenomenon about the weights. As previously stated in Sect. 3.1, the weights serve to indicate the relative importance of each point within the digital net. It is notable that for all compression rates, the weights associated with the first two points (of which one is always the zero vector) are considerably higher than those of the remaining points. Indeed, the latter are largely approximately zero. This implies that the majority of the points are located within the same elementary interval as the first two points for a significant number of partitions. Consequently, the QMC-averaging point set is not an ac-

curate representation of the data set $\mathcal{X}$. This is primarily due to particular structure of $\mathcal{X}$. Since a substantial number of pixels are completely black (have a 0-entry), it is understandable that a large proportion of points is close to the 0-vector. Therefore, a representative set that is well distributed on the unit cube, is not a good choice for our data. In fact, this behavior can be even more drastically observed when attempting to use the QMC-Voronoi algorithm to train the network. It is noteworthy that all points from the set $\mathcal{X}$ are assigned to only two points (in fact, the first two points) in the set P. Consequently, these points are the only points for which a corresponding response exists and, therefore, the only points to train the neural network with. Given that two points are insufficient for training a neural network, this method could not be included in Table 6.

The reason why K-means clustering is not performing well may be attributed to the clustering based on the $\boldsymbol{x}$-space. This causes many clusters to contain points with different corresponding responses. This is not a problem for a cluster like $\{2, 2, 2, 2, 2, 2, 2, 4, 4\}$, since the rounded cluster mean response is still 2. Though, clusters such as $\{2, 2, 2, 2, 4, 4, 4, 4\}$ correspond to a response of 3, which is not even a response within the cluster. Therefore incorporating the y-space into the clustering method appears to be a logical approach. This could also explain why the robust version performs less effectively in this instance. It incorporates the $\boldsymbol{x}$-space into the loss function instead of just the y-space. As a result, for this discrete y-space problem, a clustering approach based on the output space, performs the best. One potential solution to this issue is to employ a different rounding method. Instead of taking the average of the responses, we could select the response that appears the most frequently. However, this does not address the fundamental challenge that methods primarily using the $\boldsymbol{x}$-space to cluster can produce clusters with a multitude of corresponding responses.

To gain further insight, we examine the confusion matrix of each method for a compression rate of 20%. This is depicted in Fig. 5.

The supercompress is the only method that predicts every number (with a certain degree of accuracy). The robust version fails to predict zeros. Consequently, it overpredicts other numbers, particularly high numbers such as 7 and 8. This is the primary reason for the lower accuracy compared to the normal supercompress. The K-means clustering method drastically overestimates 2 and 3 to be the correct numbers. This systematic error highlights the ineffectiveness of this method in this context. The most unsatisfactory result is produced by QMC-averaging, which predicts only one number. This is not surprising, given that only two points have notable weights and therefore dominate the loss function. The reason for the absence of certain classes in the prediction can be explained by Table 7, which shows the distribution of the corresponding responses among the original and compressed data sets. It should be noted that due to rounding, the percentages may not sum to 100. Additionally, there is a greater concentration of mass around high numbers. This is a consequence of the fact that many clusters contain two points. If the average value has a 0.5 part, the response is rounded up, for example, the response corresponding to the set $\{8, 9\}$ is 9.

Clearly, a neural network trained with a non-representative data set will be not useful for prediction.

At the end of this section, we present a comparison of the results obtained for different compression rates for the most effective method, namely supercompress. Figure 6 shows the confusion charts.

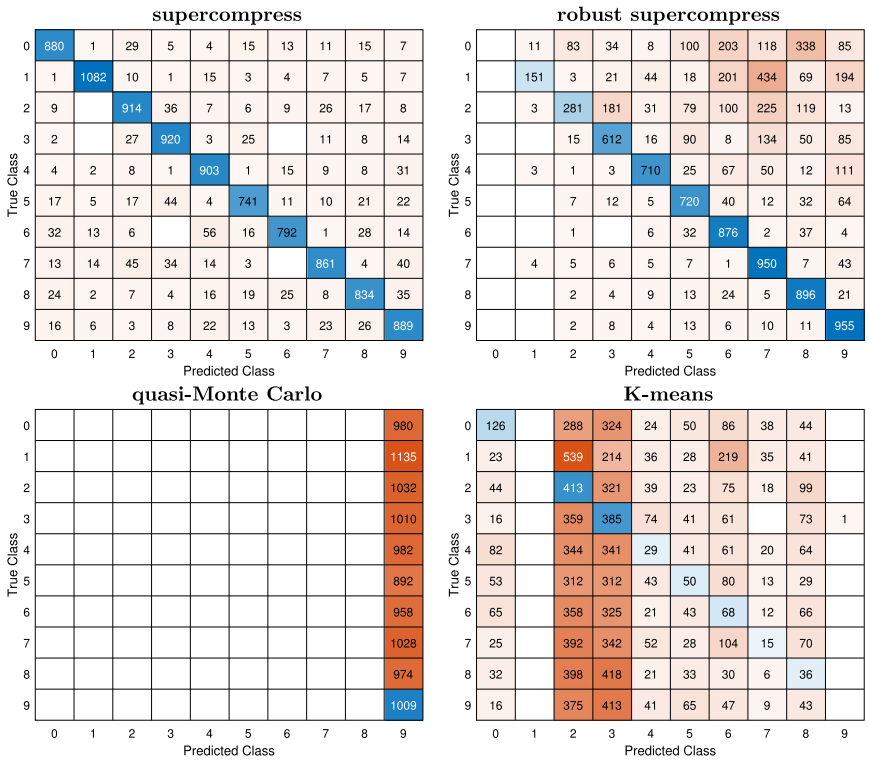

Figure 5 confusion charts for different methods with a compression rate of 20%

Table 7 distribution of the handwritten numbers for a compression rate of 20%

Method	0	1	2	3	4	5	6	7	8	9
no compression	10%	11%	10%	10%	10%	9%	10%	10%	9%	10%
supercompress	7%	6%	8%	11%	13%	8%	7%	9%	15%	15%
robust supercompress	1%	2%	3%	6%	7%	11%	15%	18%	19%	20%
K-means	10%	4%	14%	11%	11%	11%	12%	10%	12%	6%

The supercompress neural network encounters difficulties in predicting 9 when the compression rate is low. Instead, it assumes that the value is 8. However, this misclassification disappears as the number of compressed data points increases. In fact, the result for a compression rate of 20% is already close to the uncompressed prediction. The time required is reduced by approximately 80%, while the accuracy only decreases by 4%. Perhaps even more impressive is the result for 40%. Here, we require approximately half the time, yet the accuracy differs by less than 0.2%. This suggests that the supercompress method is a suitable alternative for training the neural network with a smaller set, while maintaining a high level of prediction accuracy.

5 Conclusion

In this experimental study we compared two QMC-based data reduction approaches with the supercompress methods from [14]. While supercompress performs well for the classical MNIST data set, both QMC-methods drastically fail for this data set. A reduced data set given by QMC-points is not able to recapture the original data adequately, neither in combination with an averaging procedure and appropriate weights as proposed in [5] nor

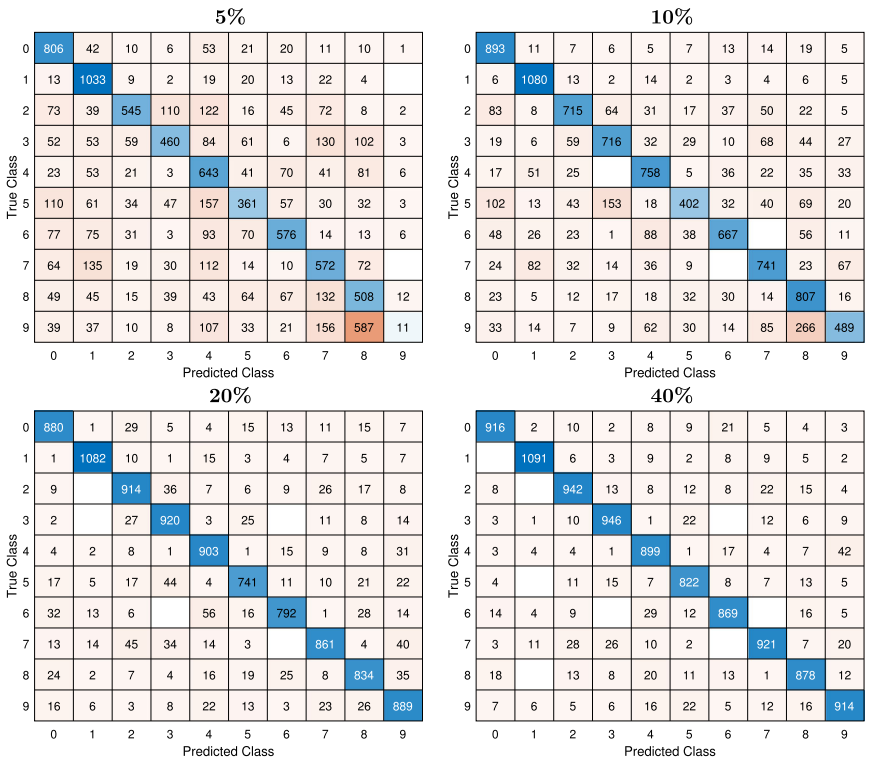

Figure 6 confusion charts for supercompress with different rates

in combination with clustering in the $\boldsymbol{x}$-space as in the QMC-Voronoi method. For simpler test functions, i.e. more regular problems, the QMC-Voronoi method performs comparably well to the supercompress method, while the QMC-averaging method also performs badly here, in view of both error and running time. The high running time in comparison to the other methods is a direct consequence of the averaging procedure, and this procedure might also explain the bad error behavior: alternating averaging procedures, which even involve binomial coefficients, are prone to possible numerical instabilities; see e.g. [8, 12].

The supercompress method performs best both for regular and irregular data. The reason for this seems to be its particular focus on the y-space for the clustering; in contrast to it robust version, which uses also the $\boldsymbol{x}$-space, or the QMC-Voronoi method, which only uses the $\boldsymbol{x}$-space.

Although a general judgment on the applicability of QMC points in date reduction might be premature, our observations underline the importance of two pieces of wisdom from applied mathematics' folklore: Regular problems can be solved with non-adaptive methods, while irregular problems require adaptive methods. Moreover, in practical applications, heuristic algorithms can be superior to theoretically well-understood methods.

Appendix

Table 8 scaled average error (top row supercompress, bottom row QMC-averaging)

Compression rate	N	s	K	f_1	f_2	f_3
1%	3000	2	2^5	$3,1054 \cdot 10^{-4}$ $2,24 \cdot 10^{-2}$	$1,0049 \cdot 10^{0}$ $2,0299 \cdot 10^{0}$	$4,6801 \cdot 10^{-2}$ $3,5859 \cdot 10^{0}$
4%	3000	2	2^7	$3,3239 \cdot 10^{-4}$ $1,37 \cdot 10^{-2}$	$5,5982 \cdot 10^{-2}$ $9,2858 \cdot 10^{-1}$	$3,993 \cdot 10^{-5}$ $3,3122 \cdot 10^{-2}$
9%	3000	2	2^8	$2,5366 \cdot 10^{-4}$ $1,9 \cdot 10^{-3}$	$1,3326 \cdot 10^{-2}$ $3,1577 \cdot 10^{-1}$	$9,4435 \cdot 10^{-6}$ $1,8061 \cdot 10^{-2}$
17%	3000	2	2^9	$1,7051 \cdot 10^{-4}$ $1,9 \cdot 10^{-3}$	$3,9779 \cdot 10^{-3}$ $2,9304 \cdot 10^{-1}$	$2,0192 \cdot 10^{-6}$ $1,848 \cdot 10^{-2}$
34%	3000	2	2^{10}	$1,0473 \cdot 10^{-4}$ $2 \cdot 10^{-3}$	$9,0664 \cdot 10^{-4}$ $3,7659 \cdot 10^{-1}$	$2,4926 \cdot 10^{-7}$ $7,7036 \cdot 10^{-3}$

Table 9 scaled average error (top row supercompress, bottom row QMC-averaging)

Compression rate	N	s	K	f_1	f_2	f_3
9%	3000	2	2^8	$2,5366 \cdot 10^{-4}$ $1,9 \cdot 10^{-3}$	$1,3326 \cdot 10^{-2}$ $3,1577 \cdot 10^{-1}$	$9,4435 \cdot 10^{-6}$ $1,8061 \cdot 10^{-2}$
9%	3000	5	2^8	$3,286 \cdot 10^{-4}$ $2,3514 \cdot 10^{-4}$	$6,0731 \cdot 10^{2}$ $9,4799 \cdot 10^{3}$	$5,7063 \cdot 10^{-5}$ $3,5184 \cdot 10^{-4}$
9%	3000	10	2^8	$3,5688 \cdot 10^{-4}$ $5,0307 \cdot 10^{-8}$	$1,975 \cdot 10^{7}$ $3,1688 \cdot 10^{20}$	$5,822 \cdot 10^{-1}$ $2,8581 \cdot 10^{-8}$

Table 10 scaled average error (top row QMC-averaging, bottom row QMC-Voronoi)

Compression rate	N	s	K	f_1	f_2	f_3
4%	3000	2	2^7	$1,37 \cdot 10^{-2}$ $4,4655 \cdot 10^{-4}$	$9,2858 \cdot 10^{-1}$ $1,1997 \cdot 10^{-1}$	$3,3122 \cdot 10^{-2}$ $1,0932 \cdot 10^{-3}$
9%	3000	2	2^8	$1,9 \cdot 10^{-3}$ $7,0245 \cdot 10^{-5}$	$3,1577 \cdot 10^{-1}$ $2,1219 \cdot 10^{-2}$	$1,8061 \cdot 10^{-2}$ $5,299 \cdot 10^{-4}$
17%	3000	2	2^9	$1,9 \cdot 10^{-3}$ $1,7547 \cdot 10^{-4}$	$2,9304 \cdot 10^{-1}$ $6,1046 \cdot 10^{-2}$	$1,848 \cdot 10^{-2}$ $3,7982 \cdot 10^{-4}$

Abbreviations
QMC, Quasi-Monte Carlo; NP, complexity class; MNIST, Modified National Institute of Standards and Technology.

Author contributions
AN and JH worked on the theoretical contribution. The numerical simulation is provided by JH while SG contributed to the interpretation of the resulting data. All authors read and approved the final manuscript.

Funding
Open Access funding enabled and organized by Projekt DEAL. The publication of this article was funded by the Ministry of Science, Research and the Arts Baden-Württemberg and the University of Mannheim. This work was financially supported by the DFG Projects GO1920/11-1 and 12-1.

Availability of data and materials
Relevant data is cited or drawn randomly.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 11 July 2024 Accepted: 13 December 2024 Published online: 03 January 2025

References

1. Aloise D, Deshpande A, Hansen P, et al. NP-hardness of Euclidean sum-of-squares clustering. Mach Learn. 2009;75:245–8.
2. Beygelzimer A, Kakadet S, Langford J, Arya S, Mount D, Li S. FNN: "Fast nearest neighbor search algorithms and applications". R 1.1.3. 2019. Available at https://CRAN.R-project.org/package=FNN.
3. Biau G, Devroye L. Lectures on the nearest neighbor method. Berlin: Springer; 2015.
4. Cervellera C, Macciò D, Rebora F. Deep learning and low-discrepancy sampling for surrogate modeling with an application to urban traffic simulation. In: 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China; 2021. p. 1–8. https://doi.org/10.1109/IJCNN52387.2021.9533357.
5. Dick J, Feischl M. A quasi-Monte Carlo data compression algorithm for machine learning. J Complex. 2021;67:101587.
6. Dick J, Pillichshammer F. Digital nets and sequences. Cambridge: Cambridge University Press.
7. Feldman D. Introduction to core-sets: an updated survey. WIREs Data Min Knowl Discov. 2020;10:e1335.
8. Goldberg D. What every computer scientist should know about floating-point arithmetic. ACM Comput Surv. 1991;23:5–48.
9. Graf S, Luschgy H. Foundations of quantization for probability distributions. Lecture notes in mathematics. Berlin: Springer; 2007.
10. Hastie T, Tibshirani R, Friedman JH. The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York: Springer; 2009.
11. Heieck J. Data compression for machine learning and applications. GitHub, 01.02.2024. https://github.com/jaheieck/data-compression-for-machine-learning-and-applications.
12. Higham NJ. The accuracy of floating point summation. SIAM J Sci Comput. 1993;14(4):783–99.
13. Inc M. Train Network Using Custom Training Loop. 2019.
14. Joseph RV, Mak S. Supervised compression of big data. Stat Anal Data Min ASA Data Sci J. 2021;14:217–29.
15. Kuo FY, Nuyens D. Application of quasi-Monte Carlo methods to elliptic PDEs with random diffusion coefficients - a survey of analysis and implementation. Found Comput Math. 2016;16(6):1631–96.
16. LeCun Y, Cortes C, Burges CJC. The MNIST Database of Handwritten Digits. http://yann.lecun.com/exdb/mnist/.
17. Longo M, et al. Higher-order quasi-Monte Carlo training of deep neural networks. SIAM J Sci Comput. 2021;43(6):3938–66.
18. Ma P, Mahoney MW, Yu B. A statistical perspective on algorithmic leveraging. J Mach Learn Res. 2015;16:861–911.
19. Mak S, Joseph RV. Support points. Ann Stat. 2018;46(6A):2562–92.
20. Mishra S, Rusch TK. Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. SIAM J Numer Anal. 2021;59(3):1811–34.
21. Niederreiter H. Point sets and sequences with small discrepancy. Monatshefte Math. 1987;104:273–337.
22. Niederreiter H. Random number generation and quasi-Monte Carlo methods. Philadelphia: SIAM; 1992.
23. Nuyens D. The "Magic Point Shop" of QMC point generators and generating vectors. 14.12.2017. https://people.cs.kuleuven.be/~dirk.nuyens/qmc-generators/.
24. Surjanovic S, Bingham D. Virtual Library of Simulation Experiments. 2013. https://www.sfu.ca/~ssurjano/zhou98.html.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com