

# Near-Linear Time Projection onto the $\ell_{1,\infty}$ Ball; Application to Sparse Autoencoders

Guillaume Perez<sup>a</sup>, Laurent Condat<sup>b</sup>, Michel Barlaud<sup>a</sup>

<sup>a</sup>Université Côte d'Azur, CNRS, Sophia Antipolis, 06900, France

<sup>b</sup>King Abdullah University of Science and Technology (KAUST) Thuwal, Kingdom of Saudi Arabia

---

## Abstract

Looking for sparsity is nowadays crucial to speed up the training of large-scale neural networks. Projections onto the  $\ell_{1,2}$  and  $\ell_{1,\infty}$  are among the most efficient techniques to sparsify and reduce the overall cost of neural networks. In this paper, we introduce a new projection algorithm for the  $\ell_{1,\infty}$  norm ball. The worst-case time complexity of this algorithm is  $\mathcal{O}(nm + J \log(nm))$  for a matrix in  $\mathbb{R}^{n \times m}$ .  $J$  is a term that tends to 0 when the sparsity is high, and to  $nm$  when the sparsity is low. Its implementation is easy and it is guaranteed to converge to the exact solution in a finite time. Moreover, we propose to incorporate the  $\ell_{1,\infty}$  ball projection while training an autoencoder to enforce feature selection and sparsity of the weights. Sparsification appears in the encoder to primarily do feature selection due to our application in biology, where only a very small part ( $< 2\%$ ) of the data is relevant. We show that both in the biological case and in the general case of sparsity that our method is the fastest.

*Keywords:* Projection, Optimization, Gradient-based methods, Green AI

---

## 1. Introduction

It is well known that the impressive performance of neural networks is achieved at the cost of a high-processing complexity and large memory usage. In fact, energy consumption and memory limits are the main bottleneck for training neural networks [1, 2]. This implies that most of the manpower energy is put into making the current hardware architectures able to work with such a high demand. Such methods range from parallelism to rematerialization [3, 4], the latter being NP-hard to solve. Recently, advances in sparse recovery and deep learning have shown that training neural networks with sparse weights not only improves the processing time and batch sizes, but most importantly improves the robustness and test accuracy of the learned models.

Looking for sparsity appears in many machine learning applications, such as the identification of biomarkers in biology [5, 6] or the recovery of sparse signals in compressed sensing [7, 8, 9]. For example, consider the problem of minimizing a reconstruction cost function  $F$  of a parameter

---

*Email addresses:* guillaume.perez06@gmail.com (Guillaume Perez), barlaud@i3s.unice.fr (Michel Barlaud)

vector  $x$ . In addition consider constraining the number of non-zero components ( $\ell_0$  seminorm) of the learned vector to at most a given sparsity value:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad F(x) \quad \text{subject to} \quad \|x\|_0 \leq \epsilon.$$

This problem is called *feature selection* and has been a large research area in machine learning. Unfortunately, this problem is generally strictly nonconvex, combinatorial, and very difficult to solve [10]. Nevertheless, many relaxed methods have been proposed, such as the *LASSO* method [11, 12], which considers the  $\ell_1$  norm instead of the  $\ell_0$  seminorm of  $x$ . One of the reasons why such regularization techniques are widely used is that Candès and Tao proved that using a  $\ell_1$  projection gives near-optimal guarantees on the reconstruction loss [13]. Since then, many methods have been defined using either the  $\ell_1$  or the reweighed  $\ell_1$  norm for sparse regularization [14]:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad F(x) \quad \text{subject to} \quad \|wx\|_1 \leq \epsilon.$$

Solving such optimization problems is usually done using projected gradient descent. Given the current point  $x_t$  and the objective function  $F$  to optimize, a gradient step is taken toward the objective:  $x_{t+1} = x_t - \gamma \nabla F(x)$ , for some step size, or learning rate,  $\gamma > 0$ . Gradient descent does not take into account the presence of constraints, hence constraints are usually inserted in the objective using Lagrange multipliers, or using projection or proximal methods. The projected gradient algorithm is then  $x_{t+1} = \alpha P_C(x_t - \gamma \nabla F(x)) + (1 - \alpha)x_t$ , with  $P_C$  the projection or proximal operator. Note that projecting onto the  $\ell_1$  or reweighed  $\ell_1$  norm ball is of linear-time complexity and is the common choice [15, 16].

In the context of deep learning, exploiting the sparsity of neural networks is not a new topic. Indeed, *dropout* for example is an early implementation of sparsity, whose goal is to increase the robustness of the learned representation [17, 18]. While dropout drastically improves the robustness of non-sparse neural networks, feature selection methods have proved more efficient to find robust and sparse models, leading to better accuracy. Indeed, in recent years, numerous methods have been proposed in order to *sparsify* the weights during the training phase [19, 20]. For example, sparse iso-flops or similar methods aim at replacing dense layers with transformation to improve the representation capacity [21, 22]. Other methods generally do produce sparse weight matrices, but this sparsity, while helping the accuracy, was not memory or processing efficient. To address this issue, the group-LASSO was proposed [23], in order to directly sparsify neurons without loss of performance [24, 25, 26]. For every  $p, q \in \mathbb{R}$ , the  $\ell_{p,q}$  norm of a real matrix  $X = [x_1 \cdots x_m] \in \mathbb{R}^{n \times m}$  with columns  $x_j$  and elements  $X_{i,j}$  is given by

$$\|X\|_{p,q} := \left( \sum_{j=1}^m \|x_j\|_q^p \right)^{\frac{1}{p}}, \quad (1)$$

where the  $\ell_q$  norm of the vector  $x_j \in \mathbb{R}^n$  is

$$\|x_j\|_q := \left( \sum_{i=1}^n |X_{i,j}|^q \right)^{\frac{1}{q}}. \quad (2)$$

By extension, the  $\ell_\infty$  norm of  $x_j$  is

$$\|x_j\|_\infty := \max_{i=1,\dots,n} |X_{i,j}|. \quad (3)$$

The group-LASSO aims at minimizing the  $\ell_{1,2}$  norm. It has been shown to outperform traditional stepwise backward elimination and provides interesting results.

Finally, the  $\ell_1$  ball projection and its derivatives have been used to enforce sparsity everywhere in deep neural networks, including with fully-connected layers, as we consider in this paper, with self-attention layers [27], and even as a replacement for the softmax activation [28]. Thus, more efficient projection algorithms have the potential to impact a large part of the deep-learning community.

The  $\ell_{1,\infty}$  norm is of particular interest because, compared to other norms, it is able to set a whole set of columns to zero, instead of spreading zeros as done by the  $\ell_1$  or  $\ell_{1,2}$  norms. This makes it particularly interesting for machine learning applications, and this is why many projection algorithms have been proposed [29, 30, 31, 32].

In this paper, we introduce a new projection algorithm for the  $\ell_{1,\infty}$  norm ball. The worst-case time complexity of this algorithm is  $\mathcal{O}(nm + J \log(nm))$  for a matrix in  $\mathbb{R}^{n \times m}$ .  $J$  is a term that tends to 0 when the sparsity is high, and to  $nm$  when the sparsity is low. While recent algorithms are either approximate or based on complex reformulations, like semismooth Newton-type methods, the proposed algorithm is simple yet efficient. As shown in the experimental section, it is faster than all existing algorithms in the presence of sparsity.

Moreover, we propose to incorporate the  $\ell_{1,\infty}$  ball projection while training an autoencoder to enforce feature selection and sparsity of the weights. Sparsification appears in the encoder to primarily do feature selection due to our application in biology, where only a very small part ( $< 2\%$ ) of the data is relevant. As shown in our experimental section, this setting allows us to accurately extract a tiny set (around 50) of relevant features from around three thousand biomarkers.

Our experimental section is split in two parts. First, we provide an empirical analysis of the projection algorithms onto the  $\ell_{1,\infty}$  ball. This part shows the advantage of the proposed method, especially in the context of sparsity. Second, we apply our framework on two biological datasets. In biology, the number of features (RNA or proteins) is very large. To make a diagnosis, only a reduced number of features is required. The problem is to select informative features. We show the advantage of using the  $\ell_{1,\infty}$  norm as a regularizer instead of other projection methods.

## 2. $\ell_{1,\infty}$ ball, simplex, and Projection

The projection onto the  $\ell_{1,\infty}$  ball has gain interest in the last years [29, 30, 31, 32]. The main reasons being its efficiency to enforce sparsity and most importantly to often increase accuracy. In this section, we formulate the problem and derive a near-linear algorithm for efficient sparse projection.

## 2.1. Definitions

Let  $Y \in \mathbb{R}^{n \times m}$  be a real matrix of dimensions  $m \geq 1, n \geq 1$ , with elements  $Y_{i,j}, i = 1, \dots, n, j = 1, \dots, m$ . The  $\ell_{1,\infty}$  norm of  $Y$  is

$$\|Y\|_{1,\infty} := \sum_{j=1}^m \max_{i=1,\dots,n} |Y_{i,j}|. \quad (4)$$

Given a radius  $C \geq 0$ , the goal is to project  $Y$  onto the  $\ell_{1,\infty}$  norm ball of radius  $C$ , denoted by

$$\mathcal{B}_{1,\infty}^C := \left\{ X \in \mathbb{R}^{n \times m} : \|X\|_{1,\infty} \leq C \right\}. \quad (5)$$

The projection  $P_{\mathcal{B}_{1,\infty}^C}$  onto  $\mathcal{B}_{1,\infty}^C$  is given by:

$$P_{\mathcal{B}_{1,\infty}^C} : Y \mapsto \arg \min_{X \in \mathcal{B}_{1,\infty}^C} \frac{1}{2} \|X - Y\|_{\text{F}}^2, \quad (6)$$

where  $\|\cdot\|_{\text{F}} = \|\cdot\|_{2,2}$  is the Frobenius norm. This projection can be derived from the projection onto the solid simplex  $\Delta_{1,\infty}^C$ :

$$\Delta_{1,\infty}^C := \left\{ X \in \mathbb{R}_+^{n \times m} : \|X\|_{1,\infty} \leq C \right\}, \quad (7)$$

where  $\mathbb{R}_+$  is the set of nonnegative reals. Indeed, let the sign function be defined by  $\text{sign}(x) := \{-1 \text{ if } x < 0; 0 \text{ if } x = 0; 1 \text{ if } x > 0\}$ . The projection of  $Y \in \mathbb{R}^{n \times m}$  onto  $\mathcal{B}_{1,\infty}^C$  is given by

$$P_{\mathcal{B}_{1,\infty}^C}(Y) = \text{sign}(Y) \odot P_{\Delta_{1,\infty}^C}(|Y|), \quad (8)$$

with  $\odot$  the Hadamard, or elementwise, product and  $|Y|$  the elementwise absolute value of  $Y$ . Moreover, if  $\|Y\|_{1,\infty} \leq C$ ,  $P_{\mathcal{B}_{1,\infty}^C}(Y) = Y$ . Thus, in the following, we focus on the projection onto  $\Delta_{1,\infty}^C$  of a matrix  $Y$  with  $\|Y\|_{1,\infty} > C$  and nonnegative elements. This projection can be characterized using auxiliary variables  $\mu_j, j = 1, \dots, m$ , as:

$$P_{\Delta_{1,\infty}^C} : Y \mapsto \arg \min_{X, \mu} \frac{1}{2} \sum_{i,j} (X_{i,j} - Y_{i,j})^2 \quad (9)$$

$$\text{subject to } \forall i, j, \quad X_{i,j} \leq \mu_j \quad (10)$$

$$\sum_{j=1}^m \mu_j = C \quad (11)$$

$$\forall i, j, \quad X_{i,j} \geq 0. \quad (12)$$

## 2.2. Properties

In the above reformulation, the objective is a direct expression of the squared distance. The constraint (10) enforces an upper bound on the values of the  $j$ -th column of  $X$ . The constraint (11)

enforces that the sum of the maximum values is equal to the radius  $C$ . The last constraint ensures non-negativity. The Lagrangian of this problem is:

$$\begin{aligned} \mathcal{L}_{1,\infty} &:= \frac{1}{2} \sum_{i,j} (X_{i,j} - Y_{i,j})^2 \\ &+ \sum_{i,j} \alpha_{i,j} (X_{i,j} - \mu_i) + \theta \left( \sum_i \mu_i - C \right) \\ &- \sum_{i,j} \beta_{i,j} X_{i,j}. \end{aligned}$$

**Lemma 1.** *At the optimal solution of problem (9)–(12), there exists a constant  $\theta \geq 0$  such that for every  $j = 1, \dots, m$ : either  $\mu_j > 0$  and  $\sum_i (Y_{i,j} - X_{i,j}) = \theta$ ; or  $\mu_j = 0$ ,  $\sum_i Y_{i,j} \leq \theta$ , and  $\forall i = 1, \dots, n$ ,  $X_{i,j} = 0$ .*

The proof is given in [29] and is a direct application of the Kuhn–Tucker theorem [33]. This lemma shows that a quantity  $\theta$  is removed from all the columns of the matrix whose sum is greater than  $\theta$ , otherwise the whole column is set to zero.

Let  $P_{\Delta_1^\theta}$  be the projection onto  $\Delta_1^\theta := \{x \in \mathbb{R}_+^n : \sum_{i=1}^n x_i \leq \theta\}$ , the solid simplex of radius  $\theta$ .

**Proposition 1.** *Let  $Y = [y_1 \cdots y_m] \in \mathbb{R}_+^{n,m}$  be a matrix such that  $\|Y\|_{1,\infty} > C$ . Then*

$$P_{\Delta_{1,\infty}^C}(Y) = [y_1 - P_{\Delta_1^\theta}(y_1) \cdots y_m - P_{\Delta_1^\theta}(y_m)], \quad (13)$$

with  $\theta$  defined in Lemma 1.

*Proof.* Consider a column  $y_j$  whose sum of elements is less than or equal to  $\theta$ . Then  $y_j = P_{\Delta_1^\theta}(y_j)$  so that  $y_j - P_{\Delta_1^\theta}(y_j)$  is the zero vector. Now consider a column  $y_j$  whose sum of elements is greater than  $\theta$ . We have, for every  $i = 1, \dots, n$ ,  $X_{i,j} = \min(Y_{i,j}, \mu_j)$ . Also, by properties of the projection onto  $\Delta_1^\theta$ ,  $z_j := P_{\Delta_1^\theta}(y_j)$  satisfies [34, 15], for every  $i = 1, \dots, n$ ,  $Z_{i,j} = \max(Y_{i,j} - \mu_j, 0)$ , so that  $X_{i,j} = \min(Y_{i,j}, \mu_j) = Y_{i,j} - Z_{i,j}$ . Hence,  $x_j = y_j - z_j$ . Also,  $\sum_i (Y_{i,j} - X_{i,j}) = \sum_i (Y_{i,j} - \max(Y_{i,j} - \mu_j, 0)) = \sum_i (\max(Y_{i,j} - \mu_j, 0)) = \sum_i Z_{i,j} = \theta$ .  $\square$

Thus, if  $\theta$  were known, the projection onto  $\Delta_{1,\infty}^C$  would be easily done using  $m$  projections onto  $\Delta_1^\theta$ . Thus, the difficulty essentially lies in finding  $\theta$ .

### 2.3. Relation between the $\ell_{1,\infty}$ and $\ell_{\infty,1}$ norms

As detailed in Section 2 of [32], the projection onto the  $\ell_{1,\infty}$  norm ball can be used to compute the proximity operator of the dual norm, which is the  $\ell_{\infty,1}$  norm:

$$\|Y\|_{\infty,1} := \max_{j=1,\dots,m} \sum_{i=1}^n |Y_{i,j}|. \quad (14)$$

Given a matrix  $Y \in \mathbb{R}^{n \times m}$  and a regularization parameter  $C > 0$ , the proximity operator of  $C\|\cdot\|_{\infty,1}$  is the mapping [35]

$$\text{prox}_{C\|\cdot\|_{\infty,1}} : Y \mapsto \arg \min_{X \in \mathbb{R}^{n \times m}} \frac{1}{2} \|X - Y\|_F^2 + C \|X\|_{\infty,1}. \quad (15)$$

Thus, computing this proximity operator amounts to solving the optimization problem in (15). This operator can be used as a subroutine in proximal splitting algorithms [36] to solve more complicated problems involving the  $\ell_{\infty,1}$  norm.

Then, by virtue of the Moreau identity [37], computing this proximity operator is equivalent to projecting onto the  $\ell_{1,\infty}$  norm ball:

$$\text{prox}_{C\|\cdot\|_{\infty,1}}(Y) = Y - P_{B_{\ell_{1,\infty}}^C}(Y). \quad (16)$$

Hence, our projection algorithm can also be used in problems involving the  $\ell_{\infty,1}$  norm.

### 3. Projection algorithms

#### 3.1. Algorithmic mechanisms

Let  $Y_j^{\mu_j} = \{i : Y_{i,j} \geq \mu_j\}$  the set of positions from column  $j$  where the values are greater than  $\mu_j$ . From the definition of the  $\ell_1$  simplex we can extract:

$$\mu_j = \frac{\sum_{i \in Y_j^{\mu_j}} Y_{i,j} - \theta}{|Y_j^{\mu_j}|}, \quad (17)$$

with  $|Y_j^{\mu_j}|$  the cardinality of the set. Let  $A$  denote the set of active columns ( $a_j = 1 \implies \mu_j > 0$ ). Let  $A = \{i, \dots, j\}$  the set of positions of ones in  $\mathbf{a}$ . Using Equation (17) and Equation (11) we have

$$C = \frac{\sum_{j \in A} \sum_{i \in Y_j^{\mu_j}} Y_{i,j} - \theta}{|Y_j^{\mu_j}|}. \quad (18)$$

Finally, from equation (17) and equation (18), we have

$$\theta = \frac{\sum_{j \in A} \sum_{i \in Y_j^{\mu_j}} \frac{Y_{i,j}}{|Y_j^{\mu_j}|} - C}{\sum_{j \in A} \frac{1}{|Y_j^{\mu_j}|}} \quad (19)$$

Let  $Z$  be the matrix where  $Z_{i,j}$  is the  $i$ th greatest value of column  $j$  of  $Y$ . Let  $S$  be the matrix where  $S_{i,j}$  is the cumulative sum of the  $i$  largest values of column  $j$  for  $Y$ ,  $S_{i,j} = \sum_{k=1}^i Z_{k,j}$ . Let  $\theta_t$  be the current approximation of  $\theta$ . Consider the addition of an element to  $\theta_t$  and its evolution with respect to its previous value. Let  $\theta_{t+1}$  be the new value after another element of  $Y$  is added to  $\theta_t$ .

**Proposition 2.** *Adding element  $Z_{i+1,j}$  to  $\theta_t$  such that  $\theta_t > iZ_{i+1,j} - S_{i,j}$  implies  $\theta_{t+1} \geq \theta_t$ .*

*Proof.* The proof is given in appendix. □

**Proposition 3.** Removing column  $j$  from  $\theta_t$  if  $\sum_i Y_{i,j} \leq \theta_t$  implies  $\theta_{t+1} \geq \theta_t$

*Proof.* The proof is given in appendix. □

Using these two proposition allows to define a first *Naive* algorithm. Algorithm 1 directly uses  $\ell_1$  projection to perform the  $\ell_{1,\infty}$  projection. This algorithms updates  $\theta_t$  until no further modifications are possible. At line 5 it removes columns with respect to proposition 3. At line 10 it gathers all the elements of a column that satisfy proposition 2. This algorithm, despite its simplicity, has been only recently proposed [32]. The authors proposed two *efficient* implementation preventing the recomputation the  $\ell_1$  projection from scratch each time. Nevertheless, its worst-case complexity is  $O(n^2mP)$  with  $P$  the complexity of projection onto the  $\ell_1$  simplex.

---

**Algorithm 1:** Projection naive [32]

---

**Data:**  $Y \in \mathbb{R}_+^{n,m}, C > 0$   
**Result:**  $X = P_{\ell_{1,\infty}}(Y)$

- 1  $\mathbf{a} \leftarrow \text{set}(\{1, \dots, m\})$
- 2  $\theta \leftarrow \frac{\sum_j \max y_j - c}{m}$
- 3 **while**  $\theta$  changed **do**
- 4     **for**  $j \in \mathbf{a}$  **do**
- 5         **if**  $\|y_j\|_1 < \theta$  **then**
- 6              $\mathbf{a} \leftarrow \mathbf{a} \setminus \{j\}$
- 7             **continue**
- 8         **end**
- 9          $x_j \leftarrow P_1^\theta(y_j)$
- 10          $S_j \leftarrow \text{set}(\{i | x_{i,j} > 0\})$
- 11     **end**
- 12      $\theta \leftarrow \frac{\sum_{j \in \mathbf{a}} \frac{\sum_{i \in S_j} Y_{i,j} - C}{|S_j|}}{\sum_{j \in \mathbf{a}} \frac{1}{|S_j|}}$
- 13 **end**
- 14  $\forall j, \mu_j \leftarrow \max(0, \frac{\sum_{i \in S_j} Y_{i,j} - \theta}{|S_j|})$
- 15  $\forall i, j, X_{i,j} \leftarrow \min(Y_{i,j}, \mu_j)$

---

*Total order.* Proposition 2 can be used to define a total order of the values of matrix  $Y$ . Let  $R = \{iZ_{i+1,j} - S_{i,j} | \forall i, \forall j\}$  be the residual matrix of  $Y$ . Let  $P$  be a non-increasing permutation of  $R$ .

**Lemma 2.** For all  $i, j \in [1, nm]$  such that  $i < j$ , if  $R_{P_i}$  cannot be added to  $\theta_t$  with respect to proposition 2, then  $R_{P_j}$  cannot be added too.

This implies that once  $P$  is known, iterating over  $P$  until proposition 2 can no longer be satisfied is enough to find all the elements that satisfy it. Here, proposition 3 is ignored, but it can be incorporated into  $P$ . Let matrix  $R' \in \mathbb{R}^{n+1,m}$  equal to  $R$  for the  $n$  first rows. The additional row filled with  $S_{n,j}$  for all  $j$ . Let  $P'$  be a non-increasing permutation of  $R'$ . Lemma 2 can be directly extended to  $P'$ .

*Build  $P'$  then find  $\theta$  [29].* One of the first projection algorithms starts by computing  $P'$  and then iterates over the elements of  $P'$  until  $R' < \theta_t$  [29]. Despite a different presentation, the processing of the residual matrix and its sorting is the same. This algorithm was one of the first algorithms able to project a matrix onto the  $\ell_{1,\infty}$  ball. Its complexity is  $O(nm + nm \log(nm))$ , a large part of the complexity being in the preprocess of  $P'$ . The performances of this algorithm are given in our experimental section.

### 3.2. Proposed Projection Algorithm

We propose here to follow a logical path to decrease the time complexity of the total order algorithm [29]. The complexity of computing  $Z$  is  $O(nm \log(n))$  as each of the columns have to be sorted. The complexity of computing  $P'$  is  $O(nm \log(nm))$  as the complete matrix  $R'$  has to be sorted. Then, the final step of finding the first element such that none of the proposition allows to add an element to the computation is linear  $O(nm)$ . More precisely, let  $K$  be the index in  $P'$  where the algorithm stops. It corresponds roughly to the number of modified values by the projection, either set to zero, or bounded. The final step of [29] is in fact of complexity  $O(K)$ , which implies that the global complexity can be seen as  $O(nm + nm \log(n) + nm \log(nm) + K)$ . In the next paragraphs, we will decrease the complexity step by step, using algorithmic improvements. The complete algorithm is then given.

- **From**  $O(nm + nm \log(n) + nm \log(nm) + K)$  **to**  $O(nm + nm \log(n) + K \log(nm))$ . Projecting vectors onto the  $\ell_{1,1}$  ball is a well-studied topic [38, 34, 15]. One of the first fast algorithms proposed to use a *heap* instead of sorting the complete vector [39]. We propose to reuse the same idea. Given a vector in  $\mathbb{R}^n$ , the creation of the heap (i.e. *Heapify*) time complexity is  $O(n)$ , the *Top* operation complexity is  $O(1)$ , the *Pop* operation and *Insert* operation complexity is  $O(\log(n))$ . Processing  $P'$  requires sorting a vector of size  $nm$ . We propose to use a heap to store  $P'$  and to extract elements one by one until  $\theta$  is found. As only  $K$  iterations over  $P'$  are required, the total complexity of this part of the algorithm is  $O(nm + K \log(nm))$  instead of  $O(nm \log(nm) + k)$ . Using a heap for the processing of  $P'$  leads to a global worst-case time complexity of  $O(nm + nm \log(n) + K \log(nm))$ .
- **From**  $O(nm + nm \log(n) + K \log(nm))$  **to**  $O(nm + K \log(nm))$ . At any moment of the algorithm, only the next largest value of a given column might be picked up by  $P'$ . This implies that the heap  $P'$  can contain only  $m$  elements at worst, instead of  $nm$  elements. The counterpart is that each time an element of  $P'$  is *popped*, the next greatest value of the column that just got popped must be inserted into the heap. If  $Z$  has been processed, then it is easy to get the next greatest element, but processing  $Z$  is costly. We propose to have one heap per column of  $Y$ , and each time the next greatest value of the column is required, then the column's heap is *popped*. Using a heap for the processing of  $P'$  and one heap per



column instead of sorting leads to a global worst-case complexity of  $O(nm + K \log(n) + K \log(m)) = O(nm + K \log(nm))$ .

- **From  $O(nm + K \log(nm))$  to  $O(nm + J \log(nm))$ .** The last and most important remark comes from the following point: Usually, the projection onto the  $\ell_{1,\infty}$  ball is applied to enforce sparsity, as in our experimental section where the best accuracy was around 99 percent of sparsity. In such case, most columns will be zeroed, and many values will be bounded in the remaining columns. Such a remark implies that  $K \approx nm$ , which implies that there is almost no gain in complexity from using all the proposed improvements. Let  $J = nm - K$  be roughly the number of non-modified values of the projected matrix. As  $K$  tends to  $nm$ ,  $J$  tends to 0 and vice-versa.

We propose to reverse the iteration over  $P'$ . Instead of starting from the beginning and looking for the first value smaller than  $\theta$ , We start from the end of  $P'$  and look for the first value greater than  $\theta$ . This value is the last value added by proposition 2 or the last column that need to be removed with respect to proposition 3. The worst-case time complexity of this algorithm is  $O(nm + J \log(nm))$ .

*Implementation.* A possible implementation is given in Algorithm 2. Function UpdateTheta() is  $\theta \leftarrow \frac{\sum_j \frac{a_j s_j - C}{k_j} - a_j}{\sum_j \frac{a_j}{k_j}}$ . First, at line 2, the global heap is created. This heap contains  $m$  elements, one for each column. For each element, two values are given, the first one is the column index, the second one is the sorting key. The initial sorting key is given by the sum of the elements of a column, this is because we are reversing the total order  $P'$ . At line 9, if it is the first time that the column is encountered, it is heapified as it will start being used by the global heap. Putting the heapify here and not at the beginning is done to spare the time used to heapify the zeroed columns. Then, the total sum of the elements of the columns is added to the current value of  $\theta$ . If the current value of  $\theta$  is already dominating the column, then the threshold has been found. Otherwise, at line 16, the current element is tested to check if it can be added to the current approximation of  $\theta$ . As shown in our experimental section, this new algorithm is faster compared to all other methods for sparse projections, and is the first near-linear method for high sparsity.

*columns eliminations.* Performances of [32] are strongly dependent on a  $O(nm + m \log(m))$  pre-process that tries to remove rows that provably will be set to zero. In the proposed algorithm, there is no need to apply this algorithm as our algorithm ignores such rows by design. Indeed, as the algorithm works backward, it never reaches rows that are dominated by  $\theta$ . In the worst case, it ends on a dominated row, and will directly discard it.

### 3.3. Masked projection

PyTorch is one of the most famous deep learning frameworks and is used in many industrial project [40]. Recently, a direct API to sparsify existing neural networks has been implemented into PyTorch <sup>1</sup>. Using this API, a sub-network can be extracted using a Boolean mask. The goal

---

<sup>1</sup>[https://pytorch.org/tutorials/intermediate/pruning\\_tutorial.html](https://pytorch.org/tutorials/intermediate/pruning_tutorial.html)

---

**Algorithm 2:** Projection Inverse Total Order

---

**Data:**  $Y \in \mathbb{R}_+^{n,m}$ ,  $C > 0$ **Result:**  $X = P_{\ell_{1,\infty}}(Y)$ 

```
1  $S \leftarrow (\sum_i y_{i,1}, \dots, \sum_i y_{i,m})$ 
2  $P \leftarrow \text{Heapify}((1 : -S_1, \dots, m : -S_m), \text{global, increasing})$ 
3  $\mathbf{k} \leftarrow \text{ones}(m, 1) \odot (n + 1)$ ;  $\mathbf{a} \leftarrow \text{zeros}(m, 1)$ ;
4  $\theta \leftarrow 0$ 
5 while  $\theta$  changed do
6   while  $\text{NotEmpty}(P)$  do
7      $j \leftarrow \text{Top}(P)$ ;  $i \leftarrow k_j$ 
8      $\mathbf{k}_j \leftarrow \mathbf{k}_j - 1$ 
9     if  $i = n + 1$  then
10       $\mathbf{a}_j \leftarrow 1$ ;  $\text{UpdateTheta}()$ 
11      if  $\|y_j\|_1 < \theta$  then
12         $\mathbf{a}_j \leftarrow 0$ ;  $\text{UpdateTheta}()$ 
13        Break
14      end
15       $X_j \leftarrow \text{Heapify}(Y_j, \text{increasing})$ 
16    else
17       $S_j \leftarrow S_j - \text{Top}(X_j)$ 
18       $\text{UpdateTheta}()$ 
19      if  $\frac{S_j - \theta}{k_j} < Y_{i,j}$  then
20         $\mathbf{k}_j \leftarrow \mathbf{k}_j + 1$ 
21         $S_j \leftarrow S_j + \text{Top}(X_j)$ 
22         $\text{UpdateTheta}()$ 
23        Break
24      end
25    end
26     $\text{UpdateTop}(P, k_j \text{Top}(X_j) - S_j)$ ;  $\text{Pop}(X_j)$ 
27  end
28 end
29  $\forall i, j, X_{i,j} \leftarrow \min(Y_{i,j}, \max(0, \frac{S_j - \theta}{k_j}))$ 
```

---

might be, for example, to extract interesting sub-networks satisfying the lottery ticket hypothesis [41]. We propose in this section to define the *masked projection*. The masked projection is defined by:

$$P_{\mathcal{B}_{1,\infty}^C}^M(Y) = \begin{cases} Y & \text{if } Y \in \mathcal{B}_{1,\infty}^C, \\ Y \odot \text{sign}(P_{\mathcal{B}_{1,\infty}^C}(|Y|)) & \text{otherwise,} \end{cases} \quad (20)$$

It corresponds to the restriction of the matrix to the remaining non-zero columns after the projection. This implies that the maximum value of the columns is not bounded. This masked projection

is compared in the experimental section against the projection onto the  $\ell_{1,\infty}$  ball. The results suggest that we can directly embed the masked projection onto PyTorch and keep high accuracy, while enforcing sparsity.

#### 4. Projection Experiments

This section presents experimental results of the projection operation alone. The goal of such experiments is to highlight the advantages and drawbacks of the proposed and known algorithms. We compared the proposed method against *Chu et. al.*[31] which uses a semi-smooth Newton algorithm for the projection. Then *Quattoni et. al.* [29], whose algorithm corresponds to the total order defined in section 15. Finally, *Bejar et. al.* [32] whose algorithm starts by removing columns that we know will be set to zero, and then applies Algorithm 1. All the code used in this experiment is the code generously provided by the authors of the respective algorithms. Unfortunately, only *Chu et. al.* and *Bejar et. al.* compete in term of performances against the proposed method. All other methods usually takes order of magnitude more times, hence are not present in most of our figures and tables. Note that such a result is coherent with already published papers [31, 32]. The complete code of these experiments can be found online<sup>2</sup>. The code used to implement the proposed method is directly using the standard library of C++ for heaps and vectors. The experiments were run on an *AMD Ryzen 9 5900X 12-Core Processor 3.70 GHz* desktop machine having 32 GB of memory. No parallelism was allowed.

The goal of the projection onto the  $\ell_{1,\infty}$  ball is usually to enhance sparsity. Our first experiment investigates the correlation between the radius  $C$  and the induced sparsity, and most importantly the running time of the algorithms. The size of the matrices is 1000x1000, values between 0 and 1 uniformly sampled and the radius are in  $[10^{-3}, 8]$ .

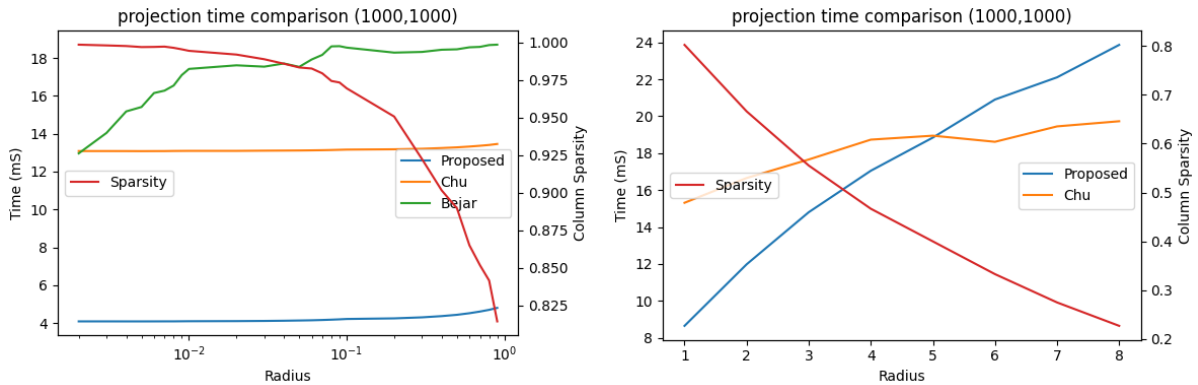


Figure 1: Impact of the radius on the sparsity of the matrix. Comparison of the projection times.

Figure 1 shows that the sparsity decreases exponentially as the radius is increasing. Moreover, we can see that the proposed algorithm is faster than the best existing methods when the sparsity is at least 40 percents. It is not surprising since the complexity of our method tends to linear when

<sup>2</sup><https://github.com/memo-p/projection>

the sparsity is high. *Bejar* method seems to be worst than *Chu et. al.*, that is why it is omitted from the second plot. As we can see, when less sparsity is present, the cost of using multiple heaps starts to slow down the algorithm. The same kind of results appears when the size of the matrix varies, as shown in Figure 2.

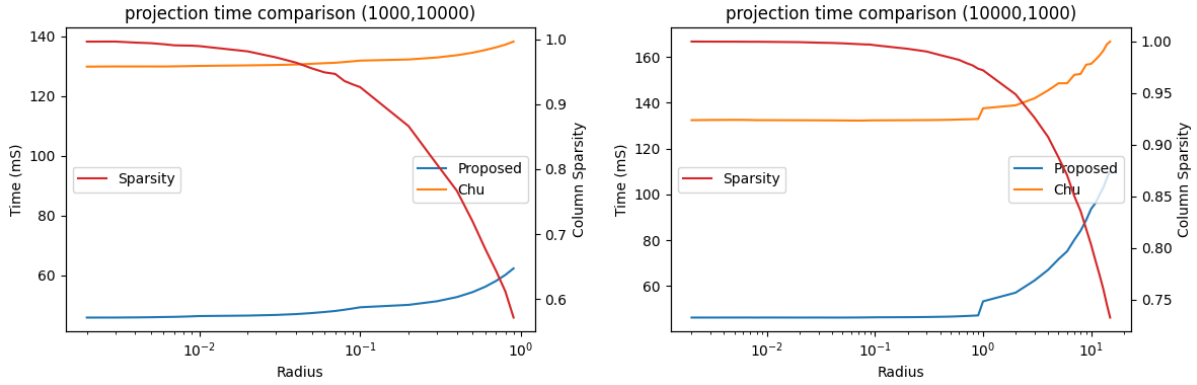


Figure 2: Projection time for matrix sizes (left) 1000x10000, (right) 10000x1000.

For the second experiment, we propose to vary the size of the matrix instead of the radius. Figure 3 gives a global view of the methods as the matrix size is increasing. we can see that as the matrix size growth, even for the radius of 1, the proposed method is significantly faster; Indeed, we can see that in both cases, the impact of the increase in the size has less impact on the proposed method. Note that the figure showing increase of size with fixed  $n$  is the best scenario for the proposed algorithm as the sparsity is increasing up with the size. We can see that overall, the proposed method is in average faster than the other methods. In the CAE experiment of the next section the proposed method was in average 2.18 faster than *Chu et. al.* given the configuration of the network.

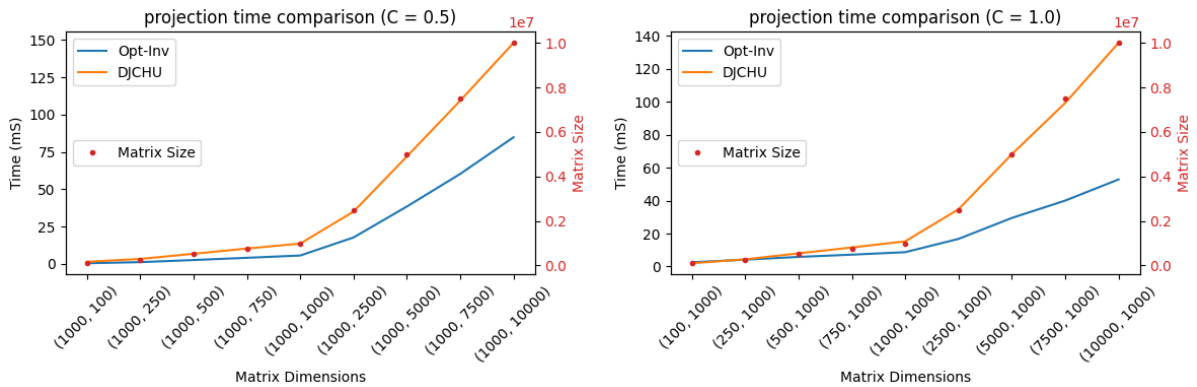


Figure 3: (left) Projection time for a fixed  $n$ . (right) Projection time for a fixed  $m$

## 5. Supervised Autoencoder (SAE) framework

Autoencoders were introduced within the field of neural networks decades ago, their most efficient application at the time being dimensionality reduction [42, 43]. Autoencoders have also been used for denoising different types of data to extract relevant features. One of the main advantages of the autoencoder is the projection of the data in the low dimensional latent space.

Autoencoders were used in application ranging from unsupervised deep-clustering [44] to supervised learning to improve classification performance [45, 46, 47]. In this paper, we use the supervised autoencoder (SAE) neural network, analogously to [48], where no constraints on the parametric distribution are present.

Figure 4 depicts the main constituent blocks of the approach. Let  $X$  be the dataset in  $\mathbb{R}^d$ , and  $Y$  the labels in  $\{0, \dots, k\}$ , with  $k$  the number of classes. Let  $Z \in \mathbb{R}^k$  be the encoded latent vectors,  $\hat{X} \in \mathbb{R}^d$  the reconstructed data and  $W$  the weights of the neural network. Note that the dimension of the latent space  $k$  corresponds to the number of classes. Let  $E(X) = Z$  be the encoder function of the autoencoder, and let  $D(Z) = \hat{X}$  be the decoder function of the autoencoder. It is an implementation of multitask learning [49] where both the reconstruction loss  $\psi(X, D(E(X)))$  and the classification loss  $\mathcal{H}(Y, E(X))$  are minimized. We use the Cross Entropy Loss for the classification loss  $\mathcal{H}$  and the robust Smooth  $\ell_1$  (Huber) Loss [50] as the reconstruction loss  $\psi$ . Parameter  $\lambda$  is a linear combination factor used to define the final loss  $\phi(X, Y) = \lambda\psi(X, \hat{X}) + \mathcal{H}(Y, Z)$ .

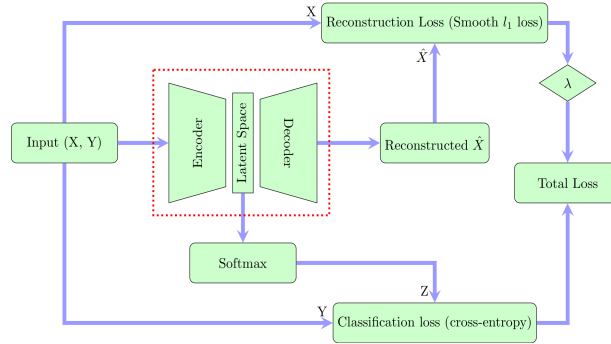


Figure 4: Supervised autoencoder framework

The goal is to learn the network weights  $W$  minimizing the total loss. In order to perform feature selection, as biomedical datasets often present a relatively small number of informative features, we also want to sparsify the network, following the work proposed in [48], [41] and [51]. We propose to use the  $\ell_{1,\infty}$  ball as a constraint to enforce sparsity in our model. The global problem to minimize is

$$\underset{W}{\text{minimize}} \quad \phi(X, Y) \quad \text{subject to} \quad \|W\|_{1,\infty} \leq C$$

Following the work by Frankle and Carbin [41] further developed by [51], we follow a double descent algorithm, originally proposed as follows: after training a network, set all weights smaller than a given threshold to zero, rewind the rest of the weights to their initial configuration, and

then retrain the network from this starting configuration while keeping the zero weights frozen (untrained). We train the network using the classical Adam optimizer [52].

To achieve structured sparsity, we replace the thresholding by our  $\ell_{1,\infty}$  projection. A possible implementation of this method is given in Algorithm 3. Note that low values of  $C$  imply high sparsity of the network. The impact and selection of such a value is discussed in the next section.

---

**Algorithm 3:** Projection algorithm.  $\phi$  is the total loss,  $\nabla\phi(W, M_0)$  is the gradient masked by the binary mask  $M_0$ ,  $A$  is the Adam optimizer,  $N$  is the total number of epochs and  $\gamma$  is the learning rate.

---

**Data:**  $W_{init}, \gamma, \eta$   
**Result:**  $W$

```

1  $t \leftarrow P_1^\eta(\|Y_1\|_1, \dots, \|Y_n\|_1);$  /* First descent */
2 for  $i \in 1, \dots, N$  do
3    $W \leftarrow A(W, \gamma, \nabla\phi(W))$ 
4    $W \leftarrow proj_{\ell_{1,\infty}}(W)$  /* Projection */
5    $(M_0)_{ij} \leftarrow \mathbb{1}_{x \neq 0}(w_{ij})$  /* Binary mask */
6 for  $i \in 1, \dots, N$  do
7    $W \leftarrow A(W, \gamma, \nabla\phi(W, M_0))$  /* Second descent */
```

---

## 6. SAE experimental results

We implemented our SAE method using the PyTorch framework for the model, optimizer, schedulers and loss functions. We chose the ADAM optimizer [52], as the standard optimizer in PyTorch. We used a symmetric linear fully connected network [48], with the encoder comprised of an input layer of  $d$  neurons, one hidden layer followed by a ReLU activation function and a latent layer of dimension  $k$ .

We compare 3 projections  $\ell_1, \ell_{2,1}, \ell_{1,\infty}$ . Note that our SAE provides a two-dimensional latent space where the samples can be visualized, and their respective classifications interpreted. Moreover we provide results using `torch.nn.utils.prune` to sparsify our neural networks, implementing our own custom projection pruning technique<sup>3</sup>

Finally, our supervised autoencoder specifically provides informative features [53] which are especially insightful for biologists. We provide for each experiment the accuracy and the column sparsity (number of columns set to zero).

### 6.1. Synthetic data

To generate artificial biological data to benchmark our  $\ell_{1,\infty}$  projection in the SAE framework, we use the `make_classification` utility from `scikit-learn`. This generator creates clusters of points that are normally distributed along vertices of a  $k$ -dimensional hypercube. This generator control

---

<sup>3</sup>[https://pytorch.org/tutorials/intermediate/pruning\\_tutorial.html](https://pytorch.org/tutorials/intermediate/pruning_tutorial.html)

the length of those vertices and thus the separability (we chose separability= 0.8) of the synthetic dataset. We generate  $n = 1,000$  samples (a number related to the number of samples in large biological datasets) with a number  $d$  of features. We chose  $d = 10,000$  as the dimension to test because this is the typical range for biological data. We chose a low number of informative features (64 ) realistically with single cell or metabolomic biological databases.

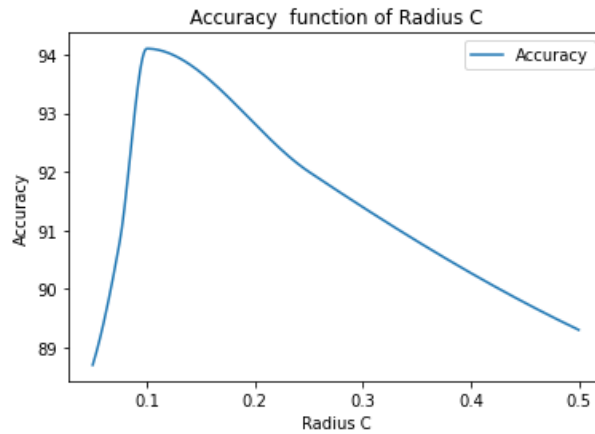


Figure 5: Synthetic data: Accuracy as a function of the radius  $C$ .

Figure 5 shows the impact of the radius ( $C$ ) used by the projection ( $\|W\|_{1,\infty} \leq C$ ). It can be seen that tuning the projection radius, and thus the sparsity, is necessary to improve the accuracy for synthetic data. From this figure, it can be seen that the best accuracy is around 0.1.

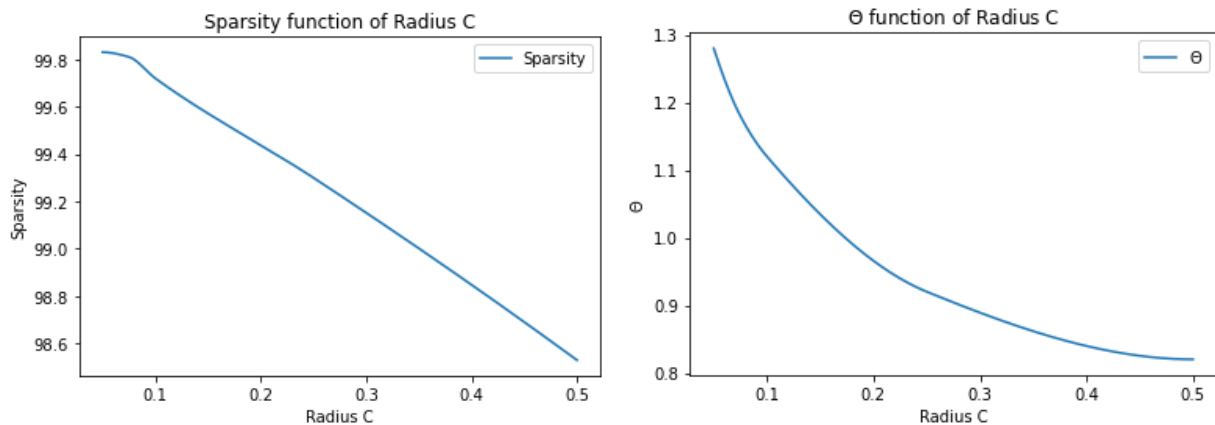


Figure 6: Synthetic data. Left: sparsity and parameter  $\theta$  as a function of the radius  $C$ . Right: Parameter  $\theta$  as a function of the radius  $C$ .

Then, Figure 6 (left) shows the impact of the radius on the obtained sparsity. Unsurprisingly, the larger is the radius, the smaller is the sparsity. Yet, by considering that the best accuracy is around 0.1, the column sparsity is around 99.6, hence the number of selected features is around 40. Figure 6 (right) shows the impact of the radius on the obtained parameter  $\theta$ .  $\theta$  is the threshold

used by the projection. As we can see, the  $\theta$  value does not decrease linearly with respect to the radius

Synthetic data	Baseline	$\ell_1$	$\ell_{2,1}$	$\ell_{1,\infty}$	$\ell_{1,\infty}$ masked
Accuracy %	86.60 $\pm$ 2.0	90.78 $\pm$ 1.51	89.1 $\pm$ 1.8	92.77 $\pm$ 1.8	92.73 $\pm$ 1.21
Colsp	0	81.94	93.97	99.6	99.6

Table 1: **Synthetic** dataset: Metrics over multiple seeds: comparison of no projection and 4 projections methods  $\ell_1$  ( $\eta = 10$ ),  $\ell_{2,1}$  ( $\eta = 10$ ),  $\ell_{1,\infty}$  (C=0.1), Proj  $\ell_{1,\infty}$  (C=0.1) masked.

Table 1 presents the results of the different possible implementation of the framework. The baseline is an implementation that does not contains any projection. It is the usual implementation of neural networks. Then,  $\ell_1$ ,  $\ell_{2,1}$ ,  $\ell_{1,\infty}$ , and  $\ell_{1,\infty}$  masked are the projection on their respective norms. Compared to the baseline the SAE using the  $\ell_{1,\infty}$  projection improves the accuracy by 6.12%, while using only 0.4% of the features. Moreover, the  $\ell_{1,\infty}$  projection improved the accuracy obtained with the  $\ell_1$ . Such a result is not surprising as the  $\ell_1$  does not consider the relationship inside columns, and only see the matrix from a global point of view. It is even more interesting that the  $\ell_{1,\infty}$  projection outperforms the  $\ell_{2,1}$ , whose results are even lower than the  $\ell_1$ . Results between the  $\ell_{1,\infty}$  and  $\ell_{1,\infty}$  masked are almost similar, hence this experiments cannot help decide which one is best. Finally, considering now the sparsity the  $\ell_{1,\infty}$  (and masked) projection outperformed the  $\ell_1$  and the  $\ell_{2,1}$  by 15% and 12% respectively.

## 6.2. Biological data

The biological **LUNG** dataset was provided by Mathe et al. [54]. The goal of this experiment is to propose a diagnosis of the Lung cancer from urine samples. This dataset includes metabolomic data concerning urine samples from 469 Non-Small Cell Lung Cancer (NSCLC) patients prior to treatment and 536 control patients. Each sample is described by 2944 metabolomic features. We apply to this metabolic dataset the classical log-transform for reducing heteroscedasticity and transforming multiplicative noise into additive noise.

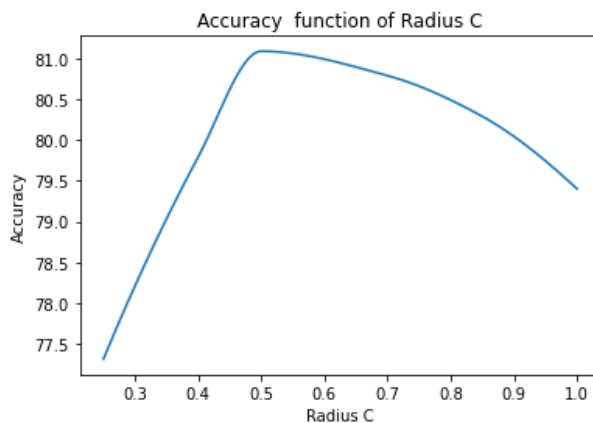


Figure 7: Lung dataset: Accuracy as a function of the radius C.



Analogously to the synthetic data, Figure 7 shows the impact of the radius on the accuracy. In this plot, it can be seen that the best accuracy is obtain with a radius of 0.5. It is also interesting to note that the slope of the decrease of the accuracy for the lung biological dataset, is smoother and less abrupt than the synthetic data.

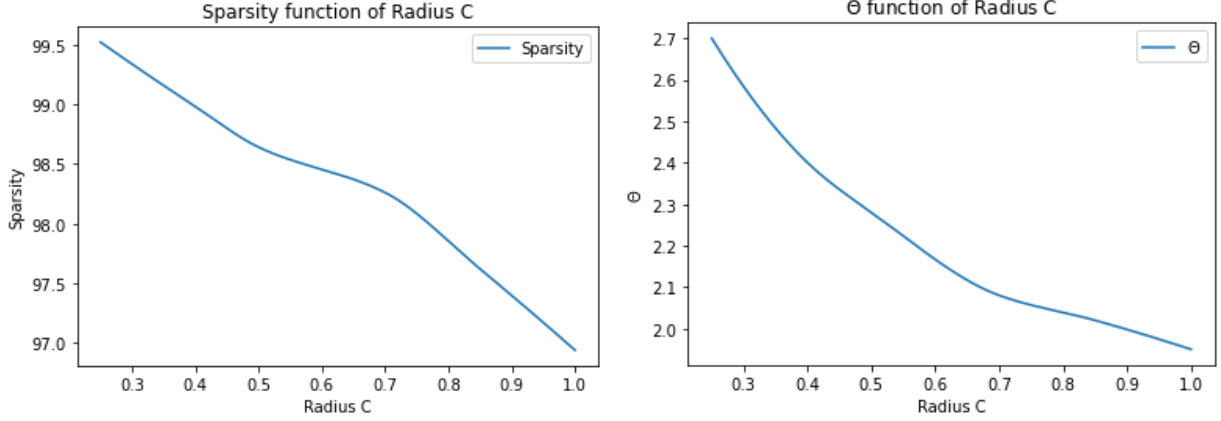


Figure 8: Left: sparsity and parameter  $\theta$  as a function of the radius C. Right: Parameter  $\theta$  as a function of the radius C.

Figure 8 (left) shows the impact of the radius on the sparsity. In this data set, at 0.5, where the accuracy is best, the sparsity is around 98.6. This implies that the best accuracy is achieved by using only around 40 metabolomic features, of the three thousand ones available. It is also interesting to remark that the curves look like a piece-wise linear with a cut at 0.5. Then, Figure 8 (right) shows the impact of the radius on the parameter  $\theta$ . The curve is almost similar to the synthetic data one.

Lung	Baseline	$\ell_1$	$\ell_{2,1}$	$\ell_{1,\infty}$	$\ell_{1,\infty}$ masked
Accuracy %	$77.12 \pm 3.45$	$79.8 \pm 2.20$	$78.5 \pm 2.24$	$81.09 \pm 2.14$	$80.84 \pm 1.72$
Colsp	0	45.72	73.53	98.6	98.6
Sum of W	-	49.99	405	45.44	241

Table 2: **Lung** dataset: Metrics over multiple seeds: comparison of no projection and 4 projections methods  $\ell_1$  ( $\eta = 50$ ),  $\ell_{2,1}$  ( $\eta = 50$ ),  $\ell_{1,\infty}$  ( $C=0.5$ ),  $\ell_{1,\infty}$  ( $C=0.5$ ) masked.

Table 2 presents the results of the different possible implementation of the framework for the Lung data set. Analogously to the previous experiment, the baseline is an implementation that does not contain any projection. The SAE with the  $\ell_{1,\infty}$  projection improves the baseline accuracy by almost 4%, while using 1.4% of the available features. Note that the  $\ell_{1,\infty}$  projection improved the accuracy obtained with the  $\ell_1$  and the  $\ell_{2,1}$  by 1.29% and 2.59% respectively. Considering now the sparsity the  $\ell_{1,\infty}$  projection outperformed the  $\ell_1$  and the  $\ell_{2,1}$  by 43% and 25% respectively.

*Overall: Masked VS Projected*. It can be seen in both experiments (synthetic and biologic) that the accuracy of the masked  $\ell_{1,\infty}$  is almost as good as the projection itself. Indeed, using the masked method, the accuracy drop is only of 0.04% in synthetic data, and of 0.75% in biological data. The only difference between the masked projection and the projection itself is the non-zeros

values that are upper bounded in each columns. Such results may implies that the upper bounding does have a beneficial effect on the result as it regularizes the data during the training. This can be seen in Table 2 where the sum of the weights of the projection is 5 times smaller that the masked one.

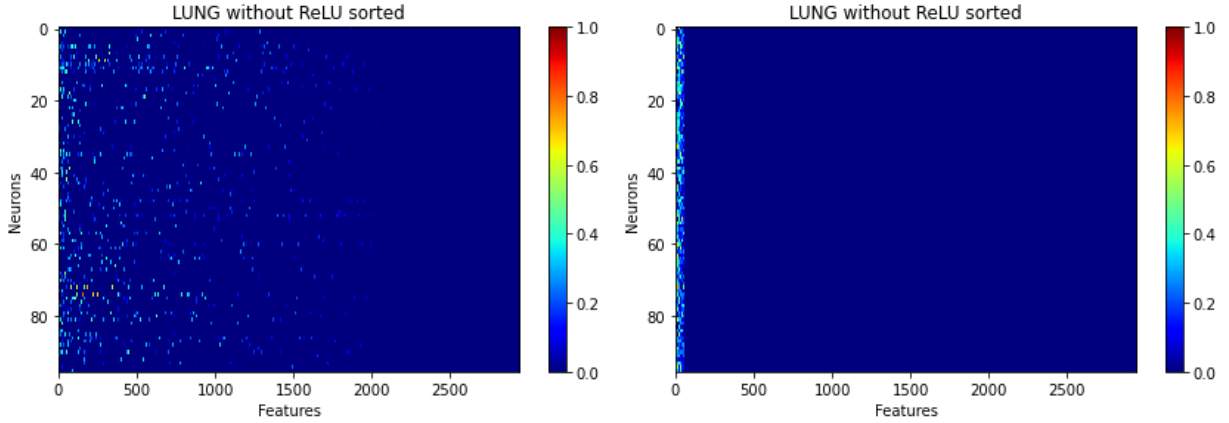


Figure 9: Sparsity of the first layer: Left: using  $\ell_1$  , Right: using  $\ell_{1,\infty}$

*Selected Features.* Figure 9 shows a heat map of the selected features. On the left, the selected features of the  $\ell_1$  method are displayed, they are 54.82% of the features. It is interesting to remark that selected features are randomly selected. On contrary, the  $\ell_{1,\infty}$  number of selected features is smaller, only 1.4% of the features. A closer look at the parameters shows that the  $\ell_1$  method radius was  $\eta = 50$ . Such a parameter implies that the maximum total sum of the weights is bounded by 50. The radius parameter of the  $\ell_{1,\infty}$  method is  $C = 0.5$ , this implies that the maximum total sum of the weights is also bound by  $0.5 * n = 48$  since  $n=96$ .

## 7. Conclusion and Perspectives

In this paper we introduced a fast projection algorithm onto the  $\ell_{1,\infty}$  ball. This projection algorithm is exact and of near-linear time complexity when the sparsity is high. As shown in our experiments, the proposed algorithm is faster than existing methods. In addition, the main goal of such a norm is to enforce structured sparsity for neural networks. As shown in the second part of our experiments, the use of the  $\ell_{1,\infty}$  ball to enforce sparsity is efficient in terms of feature selection, in terms of accuracy, and in terms of computational complexity. Such a result confirms that sparsity efficient projections should become mainstream for neural network training. Our future works involve to sparsify different types of neural networks such as autoencoders convolutional networks for image coding [55, 56].

## Acknowledgments

The authors thank Thierry Pourcher (TIRO Laboratory) for providing the Lung dataset.

## 8. Appendix

Consider the addition of an element to  $\theta_t$  and its evolution with respect to its previous value. Let  $\theta_{t+1}$  be the new value after the element  $Y_{k,l}$  is added to  $\theta_t$ . First, let's consider the impact on its local sum. Let  $v = \mu'_k$  be the new set of selected values and  $w = \mu_k$  be the value before the addition of the element.

$$\begin{aligned}\sum_{j \in Y_k^{\mu'_k}} \frac{Y_{k,j}}{|Y_k^{\mu'_k}|} &= \sum_{j \in Y_k^w} \frac{Y_{k,j}}{|Y_k^{\mu'_k}|} + \frac{Y_{k,l}}{|Y_k^{\mu'_k}|} \\ \sum_{j \in Y_k^{\mu'_k}} \frac{Y_{k,j}}{|Y_k^{\mu'_k}|} &= \sum_{j \in Y_k^w} \frac{Y_{k,j}}{|Y_k^w|} + \frac{Y_{k,l} - \bar{Y}_i^w}{|Y_k^{\mu'_k}|}\end{aligned}$$

Then we have:

$$\begin{aligned}\theta_{t+1} &= \frac{\sum_{i \in A} \sum_{j \in Y_i^{\mu_i}} \frac{Y_{i,j}}{|Y_i^{\mu_i}|} + \frac{Y_{k,l} - \bar{Y}_i^w}{|Y_k^{\mu'_k}|} - C}{\sum_{i \in A} \frac{1}{|Y_i^{\mu'_i}|}} \\ \theta_{t+1} &= \theta_t \frac{\sum_{i \in A} \frac{1}{|Y_i^{\mu_i}|}}{\sum_{i \in A} \frac{1}{|Y_i^{\mu'_i}|}} + \frac{\frac{Y_{k,l} - \bar{Y}_i^w}{|Y_k^{\mu'_k}|}}{\sum_{i \in A} \frac{1}{|Y_i^{\mu'_i}|}} \\ \theta_{t+1} &= \theta_t + \frac{\frac{\theta_t}{|Y_i^{\mu_i}|}}{\sum_{i \in A} \frac{1}{|Y_i^{\mu'_i}|}} + \frac{\frac{Y_{k,l} - \bar{Y}_i^w - \theta_t}{|Y_k^{\mu'_k}|}}{\sum_{i \in A} \frac{1}{|Y_i^{\mu'_i}|}} \\ \theta_{t+1} &= \theta_t + \frac{\frac{\theta_t |Y_k^{\mu'_k}| + |Y_k^{\mu_k}| (Y_{k,l} - \bar{Y}_i^w - \theta_t)}{|Y_k^{\mu'_k}| |Y_k^{\mu_k}|}}{\sum_{i \in A} \frac{1}{|Y_i^{\mu'_i}|}} \\ \theta_{t+1} &= \theta_t + \frac{\frac{\theta_t + |Y_k^{\mu_k}| (Y_{k,l} - \bar{Y}_i^w)}{|Y_k^{\mu'_k}| |Y_k^{\mu_k}|}}{\sum_{i \in A} \frac{1}{|Y_i^{\mu'_i}|}}\end{aligned}$$

$$\theta > jX_{i,j+1} - S_{i,j} \quad (21)$$

This condition is sufficient to ensure an increasing  $\theta$ .

When a row  $k$ , previously used until its  $l$ th element is removed: Then we have:

$$\begin{aligned}\theta_{t+1} &= \frac{\sum_{i \in A'} \sum_{j \in Y_i^{\mu_i}} \frac{Y_{i,j}}{|Y_i^{\mu_i}|} + \bar{Y}_k^l - \bar{Y}_k^l - C}{\sum_{i \in A'} \frac{1}{|Y_i^{\mu_i}'|}} \\ \theta_{t+1} &= \theta_t \frac{\sum_{i \in A} \frac{1}{|Y_i^{\mu_i}|}}{\sum_{i \in A'} \frac{1}{|Y_i^{\mu_i}'|}} - \frac{\bar{Y}_k^l}{\sum_{i \in A'} \frac{1}{|Y_i^{\mu_i}'|}} \\ \theta_{t+1} &= \theta_t + \frac{\frac{\theta_t}{|\bar{Y}_k^l|} - \bar{Y}_k^l}{\sum_{i \in A'} \frac{1}{|Y_i^{\mu_i}'|}}\end{aligned}$$

This time, it is clear that if the sum of the values of the removed row is below  $\theta$ , then the row can be safely removed and the  $\theta$  is increased.

## References

- [1] W. Zeng, X. Ren, T. Su, H. Wang, Y. Liao, Z. Wang, X. Jiang, Z. Yang, K. Wang, X. Zhang, et al., Pangu- $\alpha$ : Large-scale autoregressive pretrained Chinese language models with auto-parallel computation, preprint arXiv:2104.12369 (2021).
- [2] R. Schwartz, J. Dodge, N. A. Smith, O. Etzioni, Green AI, preprint arXiv:1907.10597 (2019).
- [3] R. Kumar, M. Purohit, Z. Svitkina, E. Vee, J. Wang, Efficient rematerialization for deep networks, Advances in Neural Information Processing Systems 32.
- [4] P. Jain, A. Jain, A. Nrusimha, A. Gholami, P. Abbeel, J. Gonzalez, K. Keutzer, I. Stoica, Checkmate: Breaking the memory wall with optimal tensor rematerialization, Proceedings of Machine Learning and Systems 2 (2020) 497–511.
- [5] T. Abeel, T. Helleputte, Y. Van de Peer, P. Dupont, Y. Saeys, Robust biomarker identification for cancer diagnosis with ensemble feature selection methods, Bioinformatics 26 (3) (2009) 392–398.
- [6] Z. He, W. Yu, Stable feature selection for biomarker discovery, Computational biology and chemistry 34 (4) (2010) 215–225.
- [7] D. L. Donoho, et al., Compressed sensing, IEEE Transactions on information theory 52 (4) (2006) 1289–1306.
- [8] S. J. Wright, R. D. Nowak, M. A. Figueiredo, Sparse reconstruction by separable approximation, IEEE Transactions on signal processing 57 (7) (2009) 2479–2493.
- [9] M. A. Figueiredo, R. D. Nowak, S. J. Wright, Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems, IEEE Journal of selected topics in signal processing 1 (4) (2007) 586–597.
- [10] B. K. Natarajan, Sparse approximate solutions to linear systems, SIAM journal on computing 24 (2) (1995) 227–234.
- [11] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society. Series B (Methodological) (1996) 267–288.
- [12] T. Hastie, R. Tibshirani, M. Wainwright, Statistical learning with sparsity: The lasso and generalizations, CRC Press.
- [13] E. J. Candès, T. Tao, Near-optimal signal recovery from random projections: Universal encoding strategies?, IEEE transactions on information theory 52 (12) (2006) 5406–5425.
- [14] E. J. Candès, M. B. Wakin, S. P. Boyd, Enhancing sparsity by reweighted  $\ell_1$  minimization, Journal of Fourier analysis and applications 14 (5-6) (2008) 877–905.
- [15] G. Perez, M. Barlaud, L. Fillatre, J.-C. Régim, A filtered bucket-clustering method for projection onto the simplex and the  $\ell_1$ -ball, Mathematical Programming.

- [16] G. Perez, S. Ament, C. Gomes, M. Barlaud, Efficient projection algorithms onto the weighted  $\ell_1$  ball, *Artificial Intelligence* 306 (2022) 103683.
- [17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, *The journal of machine learning research* 15 (1) (2014) 1929–1958.
- [18] J. Cavazza, P. Morerio, B. Haeffele, C. Lane, V. Murino, R. Vidal, Dropout as a low-rank regularizer for matrix factorization, in: *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018, pp. 435–444.
- [19] E. Tartaglione, S. Lepsøy, A. Fiandrotti, G. Francini, Learning sparse neural networks via sensitivity-driven regularization, in: *Advances in Neural Information Processing Systems*, 2018, pp. 3878–3888.
- [20] H. Zhou, J. M. Alvarez, F. Porikli, Less is more: Towards compact cnns, in: *European Conference on Computer Vision*, Springer, 2016, pp. 662–677.
- [21] S. Saxena, V. Thangarasa, A. Gupta, S. Lie, Sift: Sparse iso-flop transformations for maximizing training efficiency, preprint arXiv:2303.11525 (2023).
- [22] X. Ma, M. Qin, F. Sun, Z. Hou, K. Yuan, Y. Xu, Y. Wang, Y.-K. Chen, R. Jin, Y. Xie, Effective model sparsification by scheduled grow-and-prune methods, preprint arXiv:2106.09857 (2021).
- [23] M. Yuan, Y. Lin, Model selection and estimation in regression with grouped variables, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68 (1) (2006) 49–67.
- [24] J. M. Alvarez, M. Salzmann, Learning the number of neurons in deep networks, in: *Advances in Neural Information Processing Systems*, 2016, pp. 2270–2278.
- [25] Z. Huang, N. Wang, Data-driven sparse structure selection for deep neural networks, in: *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 304–320.
- [26] U. Oswal, C. Cox, M. Lambon-Ralph, T. Rogers, R. Nowak, Representational similarity learning with application to brain networks, in: *International Conference on Machine Learning*, 2016, pp. 1041–1049.
- [27] B. Cui, Y. Li, M. Chen, Z. Zhang, Fine-tune bert with sparse self-attention mechanism, in: *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, 2019, pp. 3548–3553.
- [28] A. Laha, S. A. Chemmengath, P. Agrawal, M. Khapra, K. Sankaranarayanan, H. G. Ramaswamy, On controllable sparse alternatives to softmax, *Advances in neural information processing systems* 31.
- [29] A. Quattoni, X. Carreras, M. Collins, T. Darrell, An efficient projection for  $\ell_{1,\infty}$  regularization, in: *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 857–864.
- [30] G. Chau, B. Wohlberg, P. Rodriguez, Efficient projection onto the  $\ell_{1,\infty}$  mixed-norm ball using a newton root search method, *SIAM Journal on Imaging Sciences* 12 (1) (2019) 604–623.
- [31] D. Chu, C. Zhang, S. Sun, Q. Tao, Semismooth newton algorithm for efficient projections onto  $\ell_{1,\infty}$ -norm ball, in: *International Conference on Machine Learning*, PMLR, 2020, pp. 1974–1983.
- [32] B. Bejar, I. Dokmanić, R. Vidal, The fastest  $\ell_{1,\infty}$  prox in the West, *IEEE transactions on pattern analysis and machine intelligence* 44 (7) (2021) 3858–3869.
- [33] M. A. Hanson, On sufficiency of the kuhn-tucker conditions, *Journal of Mathematical Analysis and Applications* 80 (2) (1981) 545–550.
- [34] L. Condat, Fast projection onto the simplex and the  $\ell_1$  ball, *Mathematical Programming Series A* 158 (1) (2016) 575–585.
- [35] J. J. Moreau, Fonctions convexes duales et points proximaux dans un espace hilbertien, *Comptes Rendus de l’Académie des Sciences de Paris* A255 (22) (1962) 2897–2899.
- [36] L. Condat, D. Kitahara, A. Contreras, A. Hirabayashi, Proximal splitting algorithms for convex optimization: A tour of recent advances, with new twists, *SIAM Review* 65 (2) (2023) 375–435.
- [37] H. H. Bauschke, P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, 2nd Edition, Springer, New York, 2017.
- [38] J. Duchi, S. Shalev-Shwartz, Y. Singer, T. Chandra, Efficient projections onto the  $\ell_1$ -ball for learning in high dimensions, in: *Proceedings of the 25th international conference on Machine learning*, ACM, 2008, pp. 272–279.
- [39] E. Van Den Berg, M. P. Friedlander, Probing the pareto frontier for basis pursuit solutions, *Siam journal on scientific computing* 31 (2) (2009) 890–912.

- [40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimselshin, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.
- [41] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, arXiv preprint arXiv:1803.03635.
- [42] G. E. Hinton, R. Zemel, Autoencoders, minimum description length and helmholtz free energy, *Advances in neural information processing systems* 6.
- [43] I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, Vol. 1, MIT press, 2016.
- [44] X. Guo, X. Liu, E. Zhu, J. Yin, Deep clustering with convolutional autoencoders, in: *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24*, Springer, 2017, pp. 373–382.
- [45] D. Kingma, M. Welling, Auto-encoding variational bayes, *International Conference on Learning Representation*.
- [46] D. P. Kingma, S. Mohamed, D. Jimenez Rezende, M. Welling, Semi-supervised learning with deep generative models, *Advances in neural information processing systems* 27.
- [47] J. Snoek, R. Adams, H. Larochelle, On nonparametric guidance for learning autoencoder representations, in: *Artificial Intelligence and Statistics*, PMLR, 2012, pp. 1073–1080.
- [48] M. Barlaud, F. Guyard, Learning a sparse generative non-parametric supervised autoencoder, *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Toronto, Canada.
- [49] R. Caruana, Multitask learning, *Machine learning* 28 (1997) 41–75.
- [50] P. J. Huber, *Robust statistics*, Wiley, New York, 1981.
- [51] H. Zhou, J. Lan, R. Liu, J. Yosinski, Deconstructing lottery tickets: Zeros, signs, and the supermask, in: *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 3597–3607.
- [52] D. Kingma, J. Ba, a method for stochastic optimization., *International Conference on Learning Representations* (2015) 1–13.
- [53] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, *Neural Information Processing Systems*, Barcelona, Spain 30.
- [54] E. Mathé *et al.*, Noninvasive urinary metabolomic profiling identifies diagnostic and prognostic markers in lung cancer, *Cancer research* 74 (12) (2014) 3259—3270.
- [55] L. Theis, W. Shi, A. Cunningham, F. Huszár, Lossy image compression with compressive autoencoders, *ICLR Conference Toulon*.
- [56] G. Cyprien, F. Guyard, M. Antonini, M. Barlaud, Learning sparse autoencoders for green ai image coding, *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Rhodes, Greece.