Projections onto the ℓ_1 Ball

Thomas Paniagua

March 3, 2021

1 INTRODUCTION

Many problems in fields such as machine learning, data science, and signals processing can be formulated as optimization problems. Furthermore, if possible, these problems are formulated within the context of convex functions due to favorable properties such as unique solutions. To limit the complexity of solutions, Penalized and Constrained Optimization (PAC) is often used. In other words, optimization objectives are minimized while fitting some constraint or minimizing an additional weighted term in the objective function to limit the complexity of a solution.
Typical optimization problems are formulated as such

$$\underset{X}{\text{minimize}}\ F(X) \text{ for unconstrained optimization}$$

$$\underset{X \in C}{\text{minimize}}\ F(X) \text{ for constrained optimization}$$

$$\underset{X}{\text{minimize}}\ F(X) + \lambda G(x) \text{ for penalized optimization where G(x) is a penalty}$$

Projection onto the ℓ_1 ball of radius α is a form of constrained optimization problem where $C = \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x}||_1 \leq \alpha\}$, $F(x) = \frac{1}{2}||\mathbf{x}-\mathbf{y}||^2$, and $\mathbf{y}$ is the vector we are calculating a projection for. $||x||_1$ is convex and C is a level set of a convex function, making it a convex set.

ℓ_1 projection can also be cast as a penalized optimization problem. Consider the dual problem of a projection onto the unit ℓ_1 ball through its lagrangian

$$L(\mathbf{x}, \lambda) = \frac{1}{2}||\mathbf{x}-\mathbf{y}||_2^2 + \lambda(||\mathbf{x}||_1 - 1)$$

minimizing the lagrangian where $\lambda = 0$ if $||\mathbf{x}|| \leq 1$ casts the problem from a penalized optimization perspective [4].

2 Motivation

ℓ_1 ball projection is used in scenarios where sparse solutions to an optimization problem are desired. A sparse solution to an optimization problem may result in better generalization bounds or easier interpretability of a solution. [4]
To notice why ℓ_1 ball projections often yield sparse solutions we may look at the realization of an ℓ_1 ball of radius 1 in $\mathbb{R}^2$.

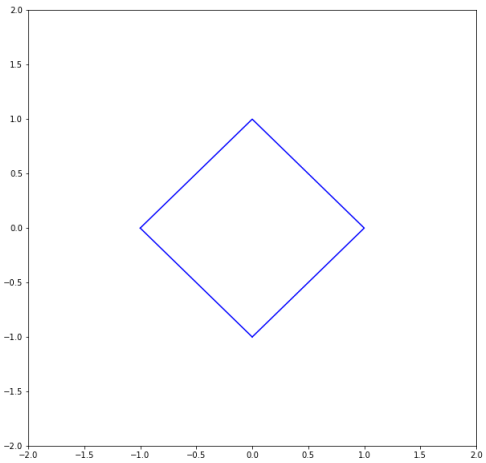
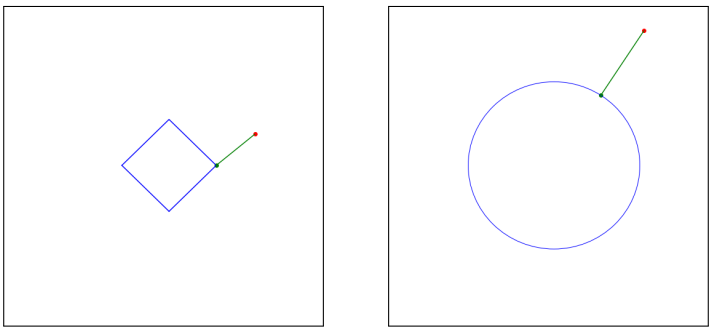


Figure 2.1: ℓ_1 ball in $\mathbb{R}^2$

The orthogonal projection of a vector onto the ℓ_1 ball will often result in a point in one of the "spikes" seen in 2.1. With other projections, such as the one onto the unit ℓ_2 ball, there is no induced sparsity 2.2.



(a) ℓ_1 (b) ℓ_2

Figure 2.2: ℓ_1 and ℓ_2 projections

While the ℓ_2 ball has a closed form of the form $\mathbf{x} = \frac{\mathbf{y}}{max\ \{1,\ ||\mathbf{y}||_2\}}$, using the same type of projector for the ℓ_1 ball where we substitute the ℓ_2 norm for the ℓ_1 norm yields a point on the ℓ_1 ball, but this point is not the **orthogonal** projection onto the ℓ_1 ball. Furthermore an optimization via a gradient-based method for calculating this projection will suffer from the fact that the ℓ_1 ball is non-differentiable at these "spikes".

3 Solution

By expanding the lagrangian formulation into

$$L(\mathbf{x}, \lambda) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||_2^2 + \lambda||\mathbf{x}||_1 - \lambda$$

we notice that this is minimized when $x = \hat{x}$ where

$$\hat{x} = \inf_{\mathbf{x}} \frac{1}{2}||\mathbf{x} - \mathbf{y}||_2^2 + \lambda||\mathbf{x}||_1$$

In this case, we can see $\hat{\mathbf{x}}$ is the proximal operator of the ℓ_1 norm, $prox_{||\mathbf{x}||_1}$. To derive an analytical solution to the proximal operator we can use the subgradient of the ℓ_1 norm and the fact that $\mathbf{x} - \mathbf{p} \in \partial||\mathbf{x}||_1$ [1]. Using the following subgradient,

$$\partial||\mathbf{x}||_i = \left\{ \begin{array}{ll} [-1, 1], & \text{for } x_i = 0 \\ 1, & \text{for } x_i < 0 \\ -1, & \text{for } x_i < 0 \end{array} \right\}$$

we can obtain the proximal operator of the ℓ_1 norm (the soft-thresholder) [1],

$$(prox_{\lambda||\mathbf{x}||_1})_i = \left\{ \begin{array}{ll} x_i - \lambda, & \text{for } x_i > \lambda \\ 0, & \text{for } |x_i| < \lambda \\ x_i + \lambda, & \text{for } x_i < -\lambda \end{array} \right\}$$

Since this operator yields the solution to the lagrangian formulation of our constrained optimization problem, all that is left to find an orthogonal projection is find an appropriate value λ for the lagrangian multiplier. The following algorithms focus on finding this value for λ. The following figure, 3.1, shows a line obtained by applying the soft-thresholding operator for a point in $\mathbb{R}^2$. The solution for the optimal projection is the intersection of the ℓ_1 ball and this line.

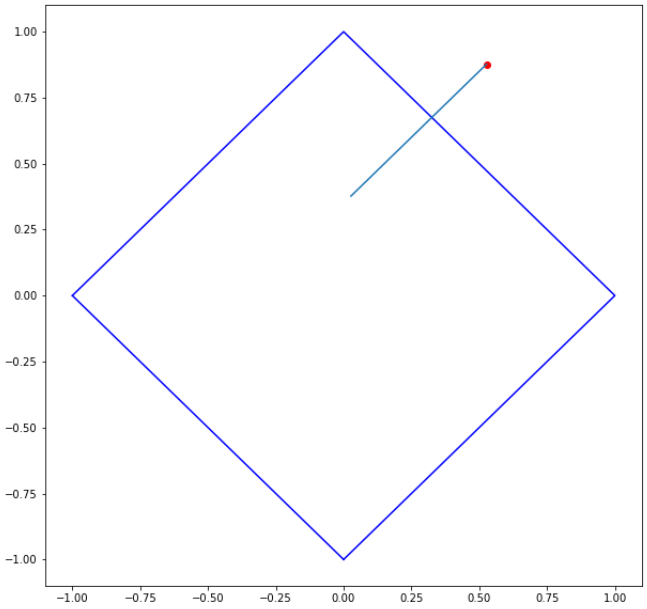


Figure 3.1: ℓ_1 ball along with line of different values of λ

4 Review of Algorithms

4.1 Naive Projected Descent

As a baseline to compare with the rest of the algorithms, we provide a steepest descent algorithm to solve for the optimal value of λ presented in the problem above.

Algorithm 1: Naive Projected Descent

Result: λ
$\lambda = max\{\mathbf{y}\}$ where $\mathbf{y}$ is the vector to be projected ;
$s = 1$;
while $\mid \|prox_{\lambda\|\mathbf{y}\|_1}\|_1 - 1 \mid > 1e-3$ **do**
  $\lambda = \lambda - (\|prox_{\lambda\|\mathbf{y}\|_1}\|_1) * \alpha$;
  $\lambda =$max $\{0, \lambda\}$
end

While this algorithm is not fast or rigorous, it provided a baseline to compare the following algorithms against.

4.2 DUCHI [4]

Along with some of the other algorithms in this review, this method first solves the problem of projection onto the probability simplex before solving the projection onto the ℓ_1 ball. This is a common method since it has been established [3] that if we have a projector onto the simplex, P_Δ, we can simply say

$$P_{\ell_1}(\mathbf{y}) = P_\Delta(abs(\mathbf{y})) * sign(\mathbf{y})$$

$$sign(\mathbf{y})_i = \left\{ \begin{array}{ll} -1, & \text{for } y_i < 0 \\ 0, & \text{for } y_i = 0 \\ 1, & \text{for } y_i > 0 \end{array} \right\}$$

The probability simplex is characterized by the intersection of the ℓ_1 ball of radius 1 and $\mathbb{R}_+$. [4] presents the solution to the lagrangian formulation of the projection problem by solving for a value of λ.
To solve this problem, they observe all the $x_i \leq \lambda$ will be 0 and only those $x_i > \lambda$ will contribute to $||\mathbf{x}||_1$. For this reason, they consider a sorted vertion of $\mathbf{x}$, $\mathbf{u}$, where $u_i > u_j$ if $j > i$. After this, the problem becomes finding an appropriate value for λ. For this they find the largest index j where $u_j - \frac{1}{j}(\sum_{i=0}^{j} u_i - 1) > 0$, then $\lambda = \frac{1}{\eta}(\sum_{i=0}^{j} u_i - 1)$. Due to the use of a sorting algorithm, the complexity of this method is at best $O(nlog(n))$.

Algorithm 2: Duchi

Result: x

$a = 1$;

1. Sort **y** into **u** in descending order (quicksort is often used) ;
2. Build a vector **v** such that $\mathbf{v}_i = \sum_{j=0}^{N} u_j$;
3. Find the max k such that $(v_k - a)/k < u_k$;
4. $\lambda = (v_k - a)/k$;
5. $\mathbf{x} = prox_{\lambda||\mathbf{y}||_1}$;

Further optimizations are proposed by the authors of [4] that propose not fully sorting vectors. These optimizations propose using a heap instead of sorting or repurposing the partioning operation of quicksort. While each of these offer performance optimizations in theory, in practice they prevent parallelism from being used in this algorithm.

This algorithm is considered to be of complexity $O(nlog(n))$ due to the complexity of the sorting algorithm used, in this case quicksort. While the complexity of this algorithm is theoretically $O(nlog(n))$, it is worth noticing that most of the steps in this algorithm (including sorting) are massively parallelizable with modern computing techniques, which would yield a much lower observed complexity if any of these optimizations are used. The following algorithms in this review achieve lower theoretical complexities, but their procedures do not lend themselves to parallelism as easily.

4.3 MICHELOT [7]

The active set method of Michelot uses the fact that for the vector $\mathbf{y}$ that we are trying to project onto the probability simplex, a lower bound for the value of λ can be calculated from any subset of $\mathbf{y}$. This lower bound is used as a pivot, since a lower bound for λ can be easily calculated, all elements $\mathbf{y}_i$ of $\mathbf{y}$ that are less than $\hat{\lambda}$ are removed from the active subset of $\mathbf{y}$ where $\hat{\lambda}$ is a lower bound for λ [3].

Algorithm 3: Michelot

Result: $\mathbf{x}$

$|\cdot|$ is the vector length. ;

$sum(\mathbf{y}) = \sum_{i=0}^{N} y_n$;

$\hat{\mathbf{y}} = y$;

$\rho = (sum(\hat{\mathbf{y}}) - 1)/|\hat{\mathbf{y}}|$;

$active\_elements = 0$;

while *active_elements* $\neq |\hat{\mathbf{y}}|$ **do**

  $active\_elements = |\hat{\mathbf{y}}|$;

  $\hat{\mathbf{y}} = \{ y_i \in \hat{\mathbf{y}} \mid y_i > \rho \}$;

  $\rho = (sum(\hat{\mathbf{y}}) - 1)/|\hat{\mathbf{y}}|$;

end

$\mathbf{x} = prox_{\rho||\mathbf{y}||_1}$;

Michelot's method achieves a better theoretical performance than Duchi's, but every iteration is dependent on the previous iteration, making it more difficult to parallelize.

4.4 CONDAT [3]

Condat's algorithm is a modification of Michelot's. Where Michelot's algorithm reads the entire active subsequence of $\mathbf{y}$ for every iteration, Condat's updates the pivot value of ρ while reading individual elements of $\mathbf{y}$.

Algorithm 4: Condat from [3]

Result: $\mathbf{x} = prox_{\rho||\mathbf{y}||_1}$;
$a = 1$;
$\mathbf{v} = (y_1)$, $\hat{\mathbf{v}} = (\,)$, $\rho = y_1 - a$;
for $i = 1;\ i \leq |\mathbf{y}|;\ i = i + 1$ **do**
  if $y_n > \rho$ **then**
    $\rho = \rho + (y_n + \rho)/(length(\mathbf{v}) + 1)$;
    if $\rho > y_n - a$ **then**
      $\mathbf{v} = \mathbf{v} + (y_n)$ (list concatenation);
    else
      $\hat{\mathbf{v}} = \hat{\mathbf{v}} + \mathbf{v}$ (list concatenation);
      $\mathbf{v} = (y_n)$;
      $\rho = y_n - a$;
    end
  end
end
if $length(\hat{\mathbf{v}}) > 0$ **then**
  for $i = 1; i <= length(\hat{\mathbf{v}}); i = i + 1$ **do**
    if $\hat{\mathbf{v}}_i > \rho$ **then**
      $\mathbf{v} = \mathbf{v} + \hat{\mathbf{v}}_i$, $\rho = \rho + (\hat{\mathbf{v}}_i - \rho)/length(\mathbf{v})$
    end
  end
end
$active\_elements = 0$;
while $active\_elements \neq length(\mathbf{v})$ **do**
  $active\_element = length(\mathbf{v})$;
  $zeroes = (\,)$;
  for $i = 1; i < length(\mathbf{v}); i++$ **do**
    if $\mathbf{v}_i \leq \rho$ **then**
      $zeroes = zeroes + (\mathbf{v}_i)$, $\rho = \rho + (\rho - \mathbf{v}_i)/length(\mathbf{v})$
    end
  end
  Remove elements from $zeroes$ in $\mathbf{v}$.
end

```python
import numpy as np
def condat_simplex(y, s=1):
    # 1.
    v = [y[0]]
    v_hat = []
    p = y[0] - s
    N = y.shape[0]
    # 2.
    for n in range(1, N):
        if y[n] > p:
            p = p + (y[n] - p) / (len(v)+1) # 2.1
            if p > y[n] - s: # 2.2
                v.append(y[n])
            else:
                # 2.3
                v_hat.extend(v)
                v = [y[n]]
                p = y[n] - 1
    # 3
    if len(v_hat) > 0:
        for y_ in v_hat:
            #3.1
            if y_ > p:
                v.append(y_)
                p = p + (y_ - p)/(len(v))
    first = False
    v_len = len(v)
    # 4
    while v_len != len(v) or not first:
        v_len = len(v)
        to_remove = []
        for i, y_ in enumerate(v):
            if y_ < p:
                to_remove.append(i)
                p = p + (p - y_)/(len(v)-len(to_remove))
        for index in sorted(to_remove, reverse=True):
            del v[index]
        first = True
    # 5
    tau = p
    K = len(v)
    return (y-tau).clamp(min=0)
```

Since the above algorithm is longer and more complicated than the rest reviewed, a *Python* implementation is also shown to resolve any ambiguity.

Condat's algorithm is theoretically the fastest of the reviewed algorithms and has an O(n) complexity. This comes at the cost of longer and lower-level code than the other algorithms. While this might be an advantage in some situations where language-specific features can be exploited to optimize code, this also makes Condat's algorithm less adaptable. Condat's algorithm exploits the fact that not every value of y_n must be known to form a lower bound and iteratively forms a lower bound for λ. For this reason, its implementation is not parallelizable in the current form of the algorithm.

5 Evaluation of Algorithms

5.1 Baseline against reviewed algorithms in $\mathbb{R}^N$ for $N \leq 15$

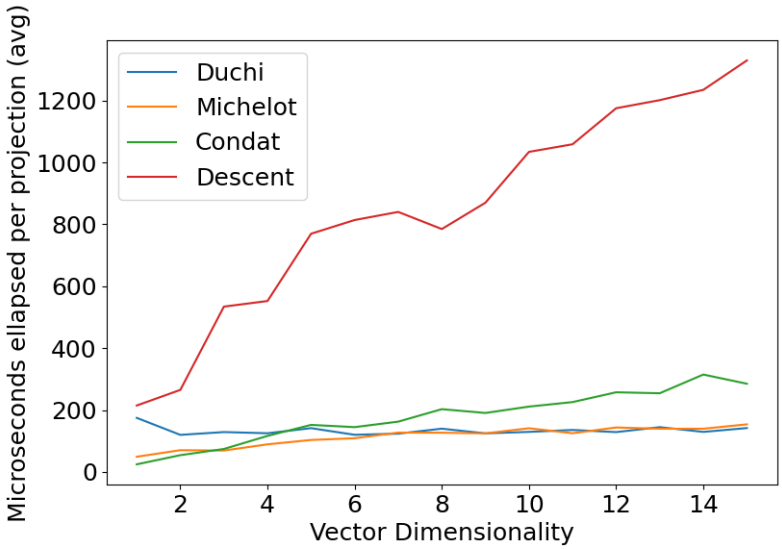
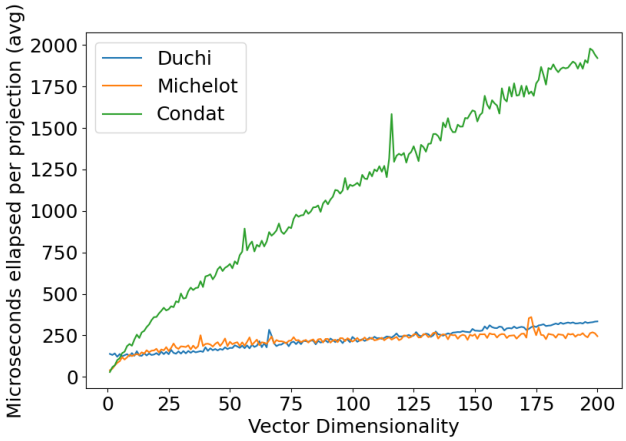


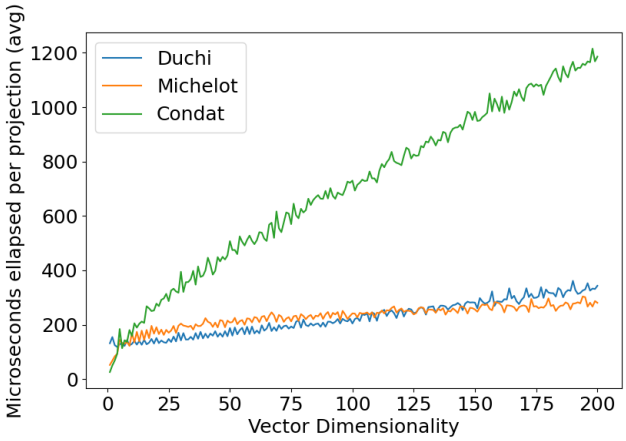
Figure 5.1: Vector Dimensionality against Ellapsed Time

The above figure shows a comparison of how the baseline algorithm based on steepest descent scales against the well established methods reviewed above. The range $N \leq 15$ was chosen since the descent based algorithm scales poorly compared to the others beyond this range.
For every dimensionality N, 50 vectors, $\mathbf{y}$ were sampled were each $y_i \sim U(-2,2)$ is a sample from a uniform distribution. Times shown in the plot are averaged between these 50 vectors.

6 Reviewed algorithms in $\mathbb{R}^N$ for $N < 200$



(a) $U \sim (-2, 2)$



(b) $N \sim (0, 2)$

Figure 6.1: Vector Dimensionality against Ellapsed Time

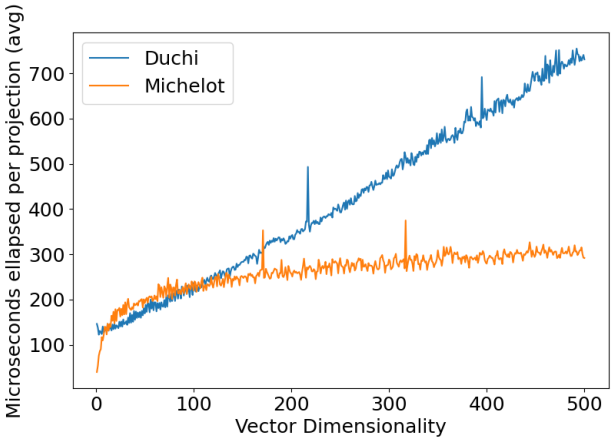
The steepest descent baseline was removed from the figure above to allow for a higher vector dimensionality. The above plots were generated by sampling 50 vectors $\mathbf{y}$ where each element $\mathbf{y}_i$ is either sampled from a uniform distribution in (a) or a normal distribution with 0 mean and variance 2 in (b).

For a high N, the observed complexity of these algorithms is $O(N)$ as the all seem to approximately scale linearly. It is notable that the above results contradict [3], since Condat's algorithm seems to scale very poorly with an increasing dimensionality. To understand why this could happen we must consider the above algorithms beyond their theoretical complexity and understand the implementation differences of these algorithms between this review and [3]. Implementation details regarding language of choice and exact code are not available in [3], so **the following arguments are speculations**.

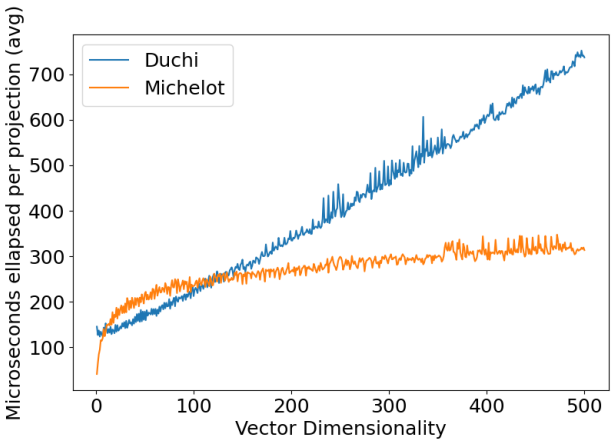
As previously discussed, Condat's algorithms gains speed from the fact that each element in a vector $\mathbf{y}$ is considered individually and the λ value for the soft-thresholder is adjusted iteratively after observing every element in $\mathbf{y}$ where elements below a lower bound are ignored. While this has the advantage of not requiring full consideration of every element in $\mathbf{y}$, it comes at the disadvantage of creating loop iterations that are dependent on previous iterations. In other words this algorithm can not be executed in parallel within one vector. Furthermore, due to the low-level nature of this algorithm, it can likely not exploit optimizations that can be exploited in Michelot's and Duchi's algorithm such as the sum operation being parallelized in a high-level call.

The reviewed algorithms were implemented in $Python$ using the $PyTorch$ library [8] for tensor computations such as sum. The code for the implementations is available publicly in the following repository `https://github.com/thomaspaniagua/L1Ball-Projections` .

6.1 Duchi's (Quicksorted) and Michelot's Algorithms in $\mathbb{R}^{\mathbb{N}}$ for $N \leq 500$



(a) $U \sim (-2, 2)$

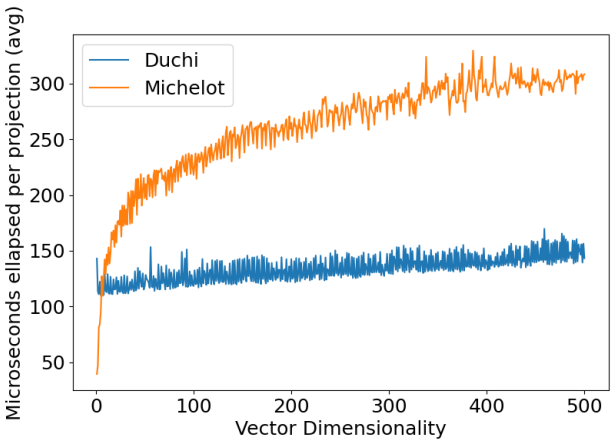


(b) $N \sim (0, 2)$

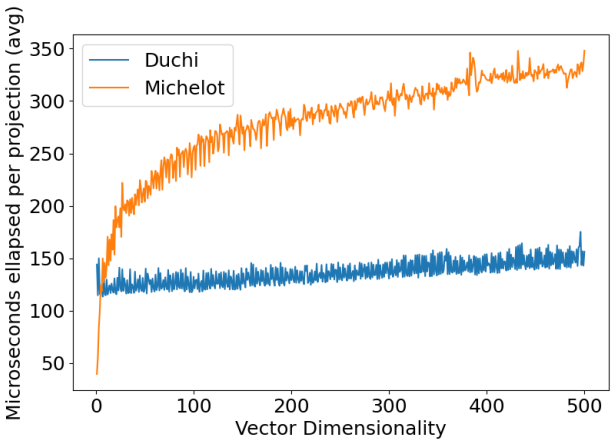
Figure 6.2: Vector Dimensionality against Ellapsed Time

The above plot was generated by using the same sampling and averaging strategy as the previous example. We have only kept Duchi's algorithm using Quicksort and Michelot's algorithm in these plots since they are closer to each other in performance where Condat's algorithm is not. A low-level implementation of quicksort was used for Duchi's algorithm since high-level languages have paralellized implementations of sorting by default. Duchi's plot here does not exploit parallel computation for sorting.

6.2 Duchi's (Parallel Sorted) and Michelot's Algorithms in $\mathbb{R}^{\mathbb{N}}$ for $N \leq 500$



(a) $U \sim (-2, 2)$



(b) $N \sim (0, 2)$

Figure 6.3: Vector Dimensionality against Ellapsed Time

The above plot was generated by using the same sampling and averaging strategy as the previous example. This example shows the ellapsed time for Duchi's algorithm and Michelot's algorithm where Duchi's now uses the parallel computed sorting of a vector provided by the $PyTorch$ [8] library.

7 Conclusion from Evaluations

Aside from the baseline, Condat's algorithm was clearly the weaker algorithm for a high N in the examples shown above, though it was the fastest for a very low N. This is likely due to the fact that tensor operation parallelism is not advantageous for $N < 5$ and thus, Condat's

algorithm proved to be the winner for a low N.
For a high N, Michelot's and Duchi's algorithm seem to scale better. Since Duchi's runtime is mostly spent in sorting on our implementation, the quicksorted version of Duchi's algorithm quickly becomes slower than Michelot's algorithm for a high N since Michelot's algorithm does not require sorting and makes use of tensor operation parallelism.
For a high N, the parallel sorted version of Duchi's algorithm seems to scale remain faster than Michelot's algorithm; making Duchi's algorithm with parallel sorting the fastest variation of sorting in this review (for high N).

8 Applications

8.1 Sparse-Max [6]

In classifiers models, the softmax function, $\text{softmax}(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_{i=0}^{N} \mathbf{x}_i}$, is often used to obtain an output that exists on the probability simplex since the outputs of a linear model would be unbounded otherwise. [6] proposes replacing softmax with the projection to the unit ℓ_1 ball to obtain an orthogonal projection onto the probability simplex for this problem.

8.2 Matrix Completion [2]

The matrix completion problem, or matrix recovery is commonly found in applications such as recommender systems. Systems can be partially filled matrices where the object of interest is the missing entires in these matrices. Matrix completion problems are often solved through low-rank matrix approximation, these problems take the form of

$$\underset{\mathbf{X}}{\text{minimize}}\ \frac{1}{2}||\mathbf{X}-\mathbf{Y}||_F^2 + \lambda||\mathbf{X}||_*$$

where $\mathbf{X}$ is a matrix we're approximating with $\mathbf{X}$ and $||\cdot||_*$ is the nuclear norm of a matrix, defined as the summation of its singular values. This problem can also be cast as a constrained optimization problem

$$\underset{\|\mathbf{X}\|_* \leq \lambda}{\text{minimize}}\ \frac{1}{2}||\mathbf{X}-\mathbf{Y}||_F^2$$

While initially, this might not seem like a problem we can apply ℓ_1 ball projections in, it is clear if we consider the Singular Value Decomposition of the matrix $\mathbf{X}$, $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\mathbf{T}$, $\Sigma = diag(\sigma_i)_{i<r}$ where r is the rank of matrix $\mathbf{X}$ [2]. We can rephrase the problem as

$$\underset{\|\Sigma_{diag}\|_1 \leq \lambda}{\text{minimize}}\ \frac{1}{2}||\mathbf{U}\Sigma\mathbf{V}^\mathbf{T}-\mathbf{Y}||_F^2$$

and solve using the projected gradient method and the ℓ_1 ball projection on the vector formed by the diagonal of Σ.

8.3 Sparse Signal Recovery [9]

The sparse signal recovery problem allows to recover a signal that we know is 0 almost everywhere, but it is observed under the presence of some disturbance. Consider, for example a signal $\mathbf{X}$ that consists of radar pulses at sparse intervals and 0 elsewhere. Depending on how the signal is measured, or where its observed, the resulting recorded signal $\mathbf{Y}$ might have been distorted with noise. If we assume this noise is additive, we can model the following process as $\mathbf{Y} = \mathbf{X} + \mathbf{N}$ where $\mathbf{N}$ is some noise signal such as gaussian white noise. To recover the signal $\mathbf{X}$ from its observed signal $\mathbf{Y}$ we can find the solution to the following problem.

$$\underset{\|\mathbf{X}\|_1 \leq \lambda}{\text{minimize}} \ \frac{1}{2}||\mathbf{X} - \mathbf{Y}||_2^2$$

which can be solved through the projected gradient method using the ℓ_1 ball projection.

8.4 Sparse Regression [10]

If we consider the linear regression problem where we have a set of datapoints $\mathbf{X} \in \mathbb{R}^{NxF}$, $\mathbf{Y} \in \mathbb{R}^{N\times 1}$. We are looking to find a mapping $\mathbf{w} \in \mathbb{R}^{Fx1}$ where $\frac{1}{2}||\mathbf{Y} - \mathbf{X}\mathbf{w}||_2^2$ is minimized. If we know some of the measurements in our datapoints are redundant or useless, in other words should not be considered in our mapping $\mathbf{w}$, we can phrase this as a constrained optimization problem.

$$\underset{\|\mathbf{w}\|_1 \leq \lambda}{\text{minimize}} \ \frac{1}{2}||\mathbf{X}\mathbf{w} - \mathbf{Y}||_2^2$$

8.5 Sparse Coding [5]

Sparse coding can be seen as reconstructing a vector $\mathbf{Y}$ from an overcomplete basis of vectors, where each vector only has a few non-zero elements. We can think of these basis vectors as rows in the matrix $\mathbf{X} \in \mathbb{R}^{B\times F}$, $\mathbf{w} \in \mathbb{R}^{1\times B}$ is a set of coefficients for each of these vectors. The problem seeks a sparse solution for $\mathbf{w}$. This problem is very similar to the sparse regression problem above. The problem can be stated as

$$\underset{\|\mathbf{w}\|_1 \leq \lambda}{\text{minimize}} \ \frac{1}{2}||\mathbf{w}\mathbf{X} - \mathbf{Y}||_2^2$$

REFERENCES

[1] *Chapter 6: The Proximal Operator*, pages 129–177.

[2] Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[3] Laurent Condat. Fast Projection onto the Simplex and the l1 Ball. *Mathematical Programming, Series A*, 158(1):575–585, July 2016.

[4] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 272–279, New York, NY, USA, 2008. Association for Computing Machinery.

[5] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 399–406, Madison, WI, USA, 2010. Omnipress.

[6] Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. volume 48 of *Proceedings of Machine Learning Research*, pages 1614–1623, New York, New York, USA, 20–22 Jun 2016. PMLR.

[7] C Michelot. A finite algorithm for finding the projection of a point onto the canonical simplex of rn. *J. Optim. Theory Appl.*, 50(1):195–200, July 1986.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[9] I. Selesnick. Sparse signal restoration. *Connexions*, 2009.

[10] Ryan Tibshirani. A closer look at sparse regression.