Journal of Artificial Intelligence Research 68 (2020) 777-816 Submitted 02/2020; published 08/2020

Gradient-based Learning Methods Extended to Smooth Manifolds Applied to Automated Clustering

Alkis Koudounas S1078994@studenti.univpm.it
Graduate School of Computer Science,
Polytechnic of Turin,
Turin, Italy

Simone Fiori s.fiori@univpm.it
Department of Information Engineering,
Marches Polytechnic University,
Ancona, Italy

Abstract

Grassmann manifold based sparse spectral clustering is a classification technique that consists in learning a latent representation of data, formed by a subspace basis, which is sparse. In order to learn a latent representation, spectral clustering is formulated in terms of a loss minimization problem over a smooth manifold known as Grassmannian. Such minimization problem cannot be tackled by one of traditional gradient-based learning algorithms, which are only suitable to perform optimization in absence of constraints among parameters. It is, therefore, necessary to develop specific optimization/learning algorithms that are able to look for a local minimum of a loss function under smooth constraints in an efficient way. Such need calls for manifold optimization methods. In this paper, we extend classical gradient-based learning algorithms on flat parameter spaces (from classical gradient descent to adaptive momentum) to curved spaces (smooth manifolds) by means of tools from manifold calculus. We compare clustering performances of these methods and known methods from the scientific literature. The obtained results confirm that the proposed learning algorithms prove lighter in computational complexity than existing ones without detriment in clustering efficacy.

1. Introduction

Gradient-based optimization methods stay at the very core of machine learning algorithms and are invoked whenever the performance of an adaptive system is evaluated through a smooth criterion function. A criterion function affords the evaluation of any given configuration of the parameters of an artificial learning system and the purpose of the optimization method is to seek the best configuration of parameters that minimizes the discrepancy between the current system's performances and its expected performances.

The behavior of a gradient-based optimization method depends on the shape of the criterion surface as, for instance, how 'deep' is a local minimum or how far local minima lie from one another. Starting from the basic gradient steepest descent method, which seeks local extrema of a criterion function by pursuing the direction indicated by the function's gradient, a number of gradient-based methods were derived. Each method in this category was developed to fix a specific issue arising in a specific situation. In the Section 2 of this paper, we are going to revise a number of classical and modern gradient-based

learning algorithms, such as basic gradient descent algorithms in Subsection 2.1, stochastic gradient descent in Subsection 2.2, mini-batch stochastic gradient descent algorithm in Subsection 2.3, gradient descent with momentum in Subsection 2.4, Nesterov accelerated gradient algorithm in Subsection 2.5, adaptive gradient method in Subsection 2.6, AdaDelta algorithm in Subsection 2.7 and adaptive moment estimation method in Subsection 2.8.

In several cases of interest, the parameters of a learning system are independent from one another, therefore the search space is $\mathbb{R}^n$, where the dimension n of the search space might be large (this is the case, for example, of a multilayer perceptron endowed with several layers and several neurons per layer). Over recent years, it occurred to researchers in this area that the parameters of a learning system may be subjected to mutual, non-linear (even very involved) constraints. If the constraints are smooth and holonomic, the constraints themselves might be represented by a smooth manifold $\mathbb{M} \subset \mathbb{R}^n$. In this event, the optimization methods at the core of systems' learning procedures need to be reformulated in terms of manifold language, namely, manifold calculus and numerical analysis on manifolds.

The basic gradient steepest descent learning method on manifold is already available in the scientific literature and found widespread application in machine learning as testified, for example, by Fiori (2010) and Bonnabel (2013). Since basic gradient descent suffers of known drawbacks, we endeavored to extend a number of classical and modern gradient-based learning methods to a general smooth manifold, as illustrated in Section 3. As a special case of particular interest in the present work, we recalled some definitions and details about the Grassmann manifold in Subsection 3.8.

In several applications – such as machine learning, image processing and computer vision – high dimensional data are widespread (Samet, 2005). Grassmann manifolds are abstract manifolds whose elements are subspaces. As such, Grassmann manifolds are natural candidates for data-size reduction and sparse representation, a necessary step in classification by high-dimensional data clustering.

Clustering is one of the most widely used data exploration tools. Its goal is to partition data points into several groups such that points in the same group are similar to one another, according to a pre-defined similarity measure, and points in different groups are dissimilar from each other. To this aim, the main steps to take are (a) creating a similarity/affinity matrix for a given dataset, and (b) performing clustering to categorize data samples. These two major steps determine the performance of spectral clustering methods. The goal of *Spectral (or Subspace) Clustering* (SC) (Lu, Yan, & Lin, 2016; Ng, Jordan, & Weiss, 2002), which is a simple extension of traditional clustering, is to cluster data points that lie in a union of low-dimensional subspaces. The key idea behind *Sparse Spectral Clustering* (SSC) is that, among infinitely many possible representations of a data point in terms of a dictionary, a sparse representation corresponds to selecting a few points from the same subspace, which form a small-size dictionary (Elhamifar & Vidal, 2013). This motivates solving a sparse optimization program whose solution is used in a spectral clustering framework. The Grassmannian manifold optimization assisted sparse spectral clustering, or GSC (Wang, Gao, & Li, 2017) provides a straightforward way to optimize the sparse clustering objective introduced by Lu et al. (2016) by adopting a Grassmann manifold optimization strategy, in order to learn a better and efficient latent feature representation.

The purpose of the present research endeavor is to extend classical and modern gradient-based machine-learning algorithms to smooth manifolds. The motivation of the present

endeavor is to improve the performances of the GSC sparse clustering technique based on Grassmann manifold representation. Such technique is based on extracting a latent representation of data based on rectangular (tall-skinny) orthogonal matrices, whose elements (the parameters of the representation) need to satisfy multiple quadratic constraints. An optimal latent representation is one that minimizes a cleverly defined loss function and arises as a trade-off between data representation ability and sparseness (i.e., a parsimonious representation is promoted). The resulting loss function is inherently non-linear in the representation parameters. Solving a non-linear optimization problem over a curved feasible set is not a straightforward task, hence specific optimization algorithms are developed. Such algorithms arise from the extension of classical gradient-based learning paradigms, such as gradient descent, momentum, and adaptive momentum. In summary, the primary contributions of this paper are:

1. To revise the GSC algorithm proposed by Wang et al. (2017) and propose a more efficient and faster way to compute the gradient of the criterion function that expresses an SSC;

2. To extend a number of classical and modern gradient-based learning methods to a general smooth manifold;

3. To evaluate the performance of the GSC algorithm learnt by these gradient-based learning methods on clustering both toy datasets and real-world (pictorial) databases.

The Section 4 of this paper contains a review of clustering in machine learning, with particular emphasis on the basic aspects of the SSC and the GSC algorithms. In this section, the main steps of the *NCut* clustering method are also recalled. In Section 5, the performance of the GSC algorithm learnt by all the gradient-based learning methods is assessed via clustering on synthetic and pictorial real-world datasets. Section 6 concludes this paper.

2. Summary of Gradient-Based Learning Methods in $\mathbb{R}^n$

The present section summarizes a number of classical as well as modern learning schemes based on parameter optimization known from the machine learning literature.

2.1 Gradient Descent (GD)

Gradient descent is an optimization algorithm used to minimize a given convex function, the so-called *loss function* $J(\theta)$, where θ is a parameter vector in $\mathbb{R}^n$. Developed in the 1970s and 1980s, GD iteratively moves parameters values in the direction of steepest descent as defined by the opposite direction of the gradient (Theodoridis, 2015).

Upon defining initial parameters values θ_0, *Gradient descent* iteratively adjusts the parameters values in order to seek a (local) minimum of the given loss function. The size of the steps taken by *GD* is determined by the *learning rate* η . The learning rate determines how fast or slow the movement towards the optimal parameters will be. In formulas we have:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t), \tag{1}$$

Algorithm 1 Stochastic Gradient Descent (SGD)

1: Input η, $\{(x^{(i)}, y^{(i)})_{i=1}^{N}\}$ and θ_0.
2: Set $t = 0$.
3: Randomly sample a point $i \in \{1, 2, \ldots, N\}$.
4: Update the parameters by

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t; x^{(i)}; y^{(i)}). \tag{2}$$

5: Increase t and return to 3 until stopping condition is fulfilled.

where $\nabla J(\theta)$ denotes the gradient of the criterion function J evaluated at a point θ and $t = 0,\ 1,\ 2,\ \ldots$ denotes an iteration index.

In order for *GD* to reach a minimum of the criterion function, the learning rate has to be set to an appropriate value, which is neither too low nor too high. This is because if the steps are too large, the GD algorithm might not reach the local minimum because it just bounces back and forth within the convex criterion. If the *learning rate* is set instead to a very small value, *GD* will eventually reach a local minimum but it will take too long time.

2.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is an optimization algorithm which improves the efficiency of the *Gradient Descent* algorithm . In fact, *GD* is computationally expensive to run on large data sets since gradient computation is based on the complete training set. Instead, *SGD* is able to take smaller steps corresponding to single input-output data pairs to be more efficient while achieving the same result (Bottou, 2010).

Let us denote by $\{(x^{(i)}, y^{(i)})_{i=1}^{N}\}$ a N-size training set. The Algorithm 1 shows how SGD optimization method works.

2.3 Mini-Batch Stochastic Gradient Descent

Mini-batch Stochastic Gradient Descent represents a trade-off between *Stochastic Gradient Descent* and *Gradient Descent*. In *Mini-batch Stochastic Gradient Descent*, the loss function (and therefore its gradient) is averaged over a small number $m \ll N$ of samples, which are usually selected upon a random permutation of the elements in the training set (Peng, Li, & Wang, 2019).

Such learning strategy appears as a generalization of the *SGD*, equivalent to a batch size of 1 sample, and as a specialization of the *GD*, equivalent to a batch size of N samples. In this way, the algorithm is much faster than GD, although there exist still some unsolved problems. First of all, the choice of the *learning rate* η is still a sensitive issue, because it should be neither too high nor too low, in order to avoid the same problems encountered in the *GD* optimization. Another notable limitation is that *Mini-batch Stochastic Gradient Descent* algorithm tends to get stuck in local minima.

2.4 Gradient Descent with Momentum

SGD has troubles navigating areas where the surface of the loss function bends more steeply in one dimension than in another. Such areas are common around local optima. The *Momentum* is a method that helps accelerate *SGD* along the relevant directions and softens the oscillations in the irrelevant directions (Qian, 1999). Such learning strategy consists in adding a fraction γ (usually $\gamma \in (0, 1]$) of the direction of the previous step to a current step. This entails amplification of speed in the correct direction and mellows oscillations along wrong directions, so that the *Gradient Descent* step could be larger, compared to *SGD*'s constant step.

The momentum term increases along dimensions whose gradients point in the same directions and reduces updates along dimensions whose gradients change directions. As a result, faster convergence and reduced oscillation are gained. In formulas, the parameter updating rule reads:

$$\begin{cases} v_t = \gamma\, v_{t-1} + \eta\, \nabla J(\theta_t), \\ \theta_{t+1} = \theta_t - v_t, \end{cases} \tag{3}$$

where the variable v_t denotes a sort of learning speed.

One of the most troubling problems with *Momentum* is that it tends to miss (or to oscillate around) the minima of the loss function.

2.5 Nesterov Accelerated Gradient (NAG)

Nesterov Accelerated Gradient is a simple change to normal *momentum* and it overcomes its main problem by starting to slow down early (Nesterov, 1983). For a recent review see, e.g., Botev, Lever, and Barber (2017).

In this algorithm indeed the gradient term is not computed from the current position θ_t in parameter space, but instead from an approximated new "look-ahead" position. This helps because while the gradient term always points in the right direction, the momentum term may not. If the momentum term points in the wrong direction or overshoots, the gradient can still "go back" and correct it in the same update step. The *NAG* learning strategy uses the next approximated position's gradient instead of using the current position's gradient. The NAG parameter updating rule reads:

$$\begin{cases} \hat{\theta}_t := \theta_t - \gamma\, v_{t-1} \\ v_t = \gamma\, v_{t-1} + \eta\, \nabla J(\hat{\theta}_t), \\ \theta_{t+1} = \theta_t - v_t. \end{cases} \tag{4}$$

The term $\hat{\theta}_t = \theta_t - \gamma v_{t-1}$, is somewhat reminiscent of the *Momentum* learning strategy.

2.6 Adaptive Gradient (AdaGrad)

AdaGrad was invented trying to improve the notion of learning rate in *GD*, which is constant and affects all the parameters at the same rate (Duchi, Hazan, & Singer, 2011). In fact, *AdaGrad* allows the *learning rate* to adapt based on parameters. Another advantage of *AdaGrad* is that it basically eliminates the need to tune the learning rate. Each parameter has its own learning rate and, due to the peculiarities of the *AdaGrad* strategy, the learning

rate is monotonically decreasing. This entails however the most important disadvantage of *AdaGrad*: at some point of time the learning rate is so small that the system stops learning.

What *AdaGrad* actually does is to accumulate the squares of all the gradient components with respect to all parameters, and to use such sum to normalize the nominal learning rate η. As a result, the effective learning rate could be smaller or larger depending on how the past gradients behaved. In fact, the adaptation of parameters that were updated largely will be slowed down, while parameters that received little updates will be subjected to bigger learning rates to accelerate their learning process.

The parameter update rule corresponding to the *AdaGrad* algorithm reads:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla J(\theta_t)_i. \tag{5}$$

In the above formula, $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each element in the diagonal (i, i) is the sum of squared gradient estimate over the course of training, up to the *time-step* t, and the subscript i denotes the i-th entry of each vector. In formulas, the component $G_{t,ii}$ is defined as:

$$G_{t,ii} = \sum_{\tau=1}^{t} (\nabla J(\theta_\tau)_i)^2. \tag{6}$$

Notice that parameters update is a component-wise operation, hence the learning rate is adaptive per-parameter. Furthermore, the constant ϵ is useful to avoid division by zero, so that optimization becomes numerically stable. For this reason, the constant ϵ is usually set to a considerably small value, like 10^{-8}.

2.7 AdaDelta

AdaDelta resolves the problem of continually decaying learning rate in *AdaGrad* by using a limited sliding window (which allows the sum to decrease), instead of summing all past squares. The actual accumulation process is implemented using the same concept as in the *Momentum* (Zeiler, 2012). Also see Qu, Yuan, Chi, Chang, and Zhao (2019). The highlights of *AdaDelta* algorithm are summarized in the Algorithm 2.

The *AdaDelta* learning algorithm attempts to alleviate the task of choosing a *learning rate* by introducing a new dynamic learning rate. Such learning rate is computed on a per-dimension basis using only first order information.

2.8 Adaptive Moment Estimation (AdaM)

Similar to *AdaDelta*, *AdaM* is an algorithm computationally efficient, little-memory demanding and appropriate for non-stationary objectives (Kingma & Ba, 2015; Zhong, Chen, Qin, Huang, Zheng, Xu, & Chen, 2020). The *AdaDelta* learning strategy computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The algorithm updates moving averages m_t of the gradient and moving averages v_t of the squared gradients' components. Two hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the first moment (the mean) and the second raw moment (the uncentered variance) of the gradient. However, these moving averages are

Algorithm 2 AdaDelta

1: Input values ϵ, γ, θ_0.
2: Set $t = 0$ and $E[\nabla J(\theta)_i^2]_{-1} = 0$.
3: Compute gradient $\nabla J(\theta)$ at the current time t
4: Accumulate gradient

$$E[\nabla J(\theta)_i^2]_t = \gamma E[\nabla J(\theta)_i^2]_{t-1} + (1-\gamma)\nabla J(\theta_t)_i^2. \tag{7}$$

5: Compute update

$$(\Delta\theta_t)_i = -\frac{\sqrt{E[(\Delta\theta)_i^2]_{t-1} + \epsilon}}{\sqrt{E[\nabla J(\theta)_i^2]_t + \epsilon}} \cdot \nabla J(\theta_t)_i. \tag{8}$$

6: Accumulate updates

$$E[(\Delta\theta)_i^2]_t = \gamma E[(\Delta\theta)_i^2]_{t-1} + (1-\gamma)(\Delta\theta_t)_i^2, \tag{9}$$

where γ is a decay constant similar to that used in the *Momentum* method.
7: Apply update

$$(\theta_{t+1})_i = (\theta_t)_i + (\Delta\theta_t)_i. \tag{10}$$

8: Increase t and return to 3 until stopping condition is fulfilled.

initialized as (vectors of) 0's, leading to moment estimates that are biased towards zero, especially during the initial timesteps, and especially when the decay rates are small. This initialization bias can be easily counteracted, resulting in bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$. The main steps of this method are shown in the Algorithm 3.

The authors of the algorithm proposed default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. They showed empirically that *AdaM* works well in practice and compares favorably to other adaptive learning algorithms.

3. Extension of Gradient-Based Learning Algorithms to Smooth Manifolds

The present section aims at extending the optimization algorithms recalled in the previous section to Riemannian manifolds. The reader should keep in mind that extending an optimization algorithm from a flat space like $\mathbb{R}^n$ to a Riemannian manifold $\mathbb{M}$ is neither straightforward *nor univocal*.

3.1 Manifold Notation

A d-dimensional manifold can be informally defined as a set $\mathbb{M}$ covered with a suitable collection of coordinate patches, or charts, that identify certain subsets of $\mathbb{M}$ with open subsets of $\mathbb{R}^d$. Such a collection of coordinate charts can be thought of as the basic structure required to perform differential calculus on $\mathbb{M}$.

Algorithm 3 Adaptive Moment Estimation (AdaM)

1: Input values β_1, β_2, ϵ, θ_0.
2: Set $t = 0$, $m_{t-1} = 0$, $v_{-1,i} = 0$.
3: Compute gradient $\nabla J(\theta)$ at the current time t
4: Update biased first moment estimate

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla J(\theta_t). \tag{11}$$

5: Update biased second raw moment estimate

$$v_{t,i} = \beta_2 v_{t-1,i} + (1 - \beta_2)(\nabla J(\theta_t)_i)^2. \tag{12}$$

6: Compute bias-corrected first moment estimate

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}. \tag{13}$$

7: Compute bias-corrected second raw moment estimate

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{14}$$

8: Update parameters

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\,\hat{m}_t. \tag{15}$$

9: Increase t and return to 3 until stopping condition is fulfilled.

At a point $U \in \mathbb{M}$, the tangent space to the manifold $\mathbb{M}$ is denoted $T_U\mathbb{M}$. The symbol $T\mathbb{M}$ denotes the tangent bundle defined as $T\mathbb{M} = \{(U, V) \mid U \in \mathbb{M}, V \in T_U\mathbb{M}\}$.

The *exponential map* is a map from a subset of a tangent space $T_U\mathbb{M}$ of a manifold $\mathbb{M}$ to $\mathbb{M}$ itself. Given a point $U \in \mathbb{M}$ and a vector $V \in T_U\mathbb{M}$, there is a unique geodesic[1] γ_V satisfying $\gamma_V(0) = U$ with initial tangent vector $\gamma_V'(0) = V$. The corresponding *exponential map* is defined by $\exp_U(V) = \gamma_V(1)$.

The *parallel transport* takes a point $U, V \in T\mathbb{M}$ and a vector $W \in T_U\mathbb{M}$ as input and transports the vector W along a geodesic arc departing from U along the direction V for a unit time. We will use the notation $\mathrm{P}_{U,V}(W)$. In the design of a numerical algorithm in Subsection 3.3, we shall invoke a version of parallel transport denoted by $\mathrm{P}_{U,V}(V)$, namely, the transport of a tangent vector along the geodesic line directed along itself. This concept exploits the well-known, defining property of a geodesic line to self-transport its own tangents. Parallel transport is also an isometry, which means that parallel transport changes the direction of a transported vector to make it conform to the geometry of the underlying manifold, without altering the length of the transported tangent vector.

In addition, we shall make use of the operator $\Pi_U[\cdot]$, which denotes an orthogonal projection over the tangent space $T_U\mathbb{M}$.

1. A geodesic is a curve representing the shortest path between two points in a manifold.

A detailed treatment of specific notions from manifold calculus exceeds the scope of the present paper. We point interested readers to available material, such as the survey by Fiori (2005) or applied papers by Fiori (2008a, 2008b). Specific formulas related to the Grassmann manifold, of interest in spectral clustering, will be given in Subsection 3.8.

3.2 Gradient Descent on $\mathbb{M}$

As mentioned in Subsection 2.1, this method imposes to add to the variable that should be optimized a small fraction of the anti-gradient, so as to follow the right path.

The parameters updating operation follows this formula:

$$U_{t+1} = \exp_{U_t}(-\eta \nabla_{U_t} J), \tag{16}$$

where $U_t \in \mathbb{M}$, $\eta \in \mathbb{R}^+$ and $\nabla_{U_t} J \in T_{U_t}\mathbb{M}$, since it represents the Riemannian gradient of the loss function J calculated in the point U_t. To start iteration, it is necessary to choose an initial guess U_0. The iteration step runs over $t = 0,\ 1,\ 2,\ \ldots,\ I$, where I denotes a pre-defined number of iterations.

This learning algorithm, as well as all the following ones, has been implemented in such a way that the iteration loop ends once a predetermined number of iterations is reached.

3.3 Gradient Descent with Momentum on $\mathbb{M}$

Faithful to the original idea recalled in Subsection 2.4, instead of using only the gradient in the current step, the *Momentum* method also accumulates the gradient of the past steps to determine the direction to move towards. Since gradients are calculated at different points, they belong to different tangent spaces: $\nabla_{U_{t-1}} J \in T_{U_{t-1}}\mathbb{M}$ whereas $\nabla_{U_t} J \in T_{U_t}\mathbb{M}$, therefore it is not possible to add these terms directly. Parallel transport has to be used to transport $\nabla_{U_{t-1}} J$ from $T_{U_{t-1}}\mathbb{M}$ to $T_{U_t}\mathbb{M}$. Upon being transported to $T_{U_t}\mathbb{M}$, it can be added to $\nabla_{U_t} J$, since they now both belong to the same tangent space. The parameters are then updated through the exponential map.

The formulas below show the essential steps of this extended learning algorithm:

$$\begin{cases} V_t = \gamma\, \mathrm{P}_{U_{t-1}, V_{t-1}}(V_{t-1}) + \eta\, \nabla_{U_t} J, \\ U_{t+1} = \exp_{U_t}(-V_t) \end{cases} \tag{17}$$

where $U_t \in \mathbb{M}$, $\eta > 0$ denotes a learning stepsize, $\gamma > 0$ is a momentum coefficient, and $\nabla_{U_t} J, V_t \in T_{U_t}\mathbb{M}$. The iteration step runs over $t = 0,\ 1,\ 2,\ \ldots,\ I$, where I denotes a pre-defined number of iterations. The initial point U_0 is chosen in $\mathbb{M}$ and the initial velocity V_0 may be either randomly picked in $T_{U_0}\mathbb{M}$, or set to $\nabla_{U_0} J$, or set to zero, while $V_{-1} = 0$.

3.4 Nesterov Accelerated Gradient on $\mathbb{M}$

The idea behind *NAG*, as recalled in Subsection 2.5, is that instead of calculating the gradient at the current position, it calculates the gradient at the position where the momentum is about to arrive, called "look-ahead" position, and by that time a fraction of this gradient is added to the previous ones. Since gradients are calculated at different points, they belong to different tangent spaces, therefore they must be taken back to the same tangent

space through the projection operator before updating the parameters. In formulas, this is written as:

$$\begin{cases} \hat{U}_t = \exp_{U_t}\left(-\gamma\, \Pi_{U_t}(V_{t-1})\right), \\ V_t = \Pi_{U_t}\left(\gamma\, V_{t-1} + \eta\, \nabla_{\hat{U}_t} J\right), \\ U_{t+1} = \exp_{U_t}(-V_t), \end{cases} \tag{18}$$

where $U_t \in \mathbb{M}$, $\eta > 0$ denotes a learning stepsize, $\gamma > 0$ denotes a forgetting factor and $\nabla_{\hat{U}_t} J, V_t \in T_{U_t}\mathbb{M}$. The iteration step runs over $t = 0,\ 1,\ 2,\ \ldots,\ I$, where I denotes a pre-defined number of iterations. The initial point U_0 is chosen in $\mathbb{M}$ and the initial velocity V_0 may be either randomly picked in $T_{U_0}\mathbb{M}$, or set to $\nabla_{U_0} J$, or set to zero, while $V_{-1} = 0$.

3.5 Adaptive Gradient on $\mathbb{M}$

A direct extension of the original AdaGrad, as recalled in Subsection 2.6, would need a decomposition of the Riemannian gradient of the criterion function into components in order to weight any component according to the square root of the accumulated component squares. In formulas, if we denoted by $\{\partial_1, \partial_2, \ldots, \partial_d\}$ the canonical basis of the tangent space $T_{U_t}\mathbb{M}$, the Riemannian gradient $\nabla_{U_t} J$ would be decomposed as $\nabla_{U_t} J = \sum_{i=1}^{d} (\nabla_{U_t} J)_i \partial_i$, where each $(\nabla_{U_t} J)_i \in \mathbb{R}$ denotes one of the components of the gradient with respect to the canonical basis and d denotes the dimension of the base manifold $\mathbb{M}$. The accumulated squared component may be updated as

$$G_{t+1,ii} = G_{t,ii} + (\nabla_{U_t} J)_i^2, \tag{19}$$

and a *normalized gradient* may be defined as follows:

$$\tilde{\nabla}_{U_t} J := \sum_{i=1}^{d} \frac{(\nabla_{U_t} J)_i}{\sqrt{G_{t,ii} + \epsilon}} \partial_i. \tag{20}$$

Although $\tilde{\nabla}_{U_t} J$ does no longer represent a Riemannian gradient of the criterion function J, it is still a tangent vector in $T_{U_t}\mathbb{M}$, therefore it is mathematically sound to update the current point U_t to the next point by

$$U_{t+1} = \exp_{U_t}(-\eta \tilde{\nabla}_{U_t} J). \tag{21}$$

The set of equations (19) and (20) are faithful to the original concept but are quite impractical due to the need of getting back and forth to the component representation and due to the need of calculating the canonical basis of each tangent space encountered during the optimization process.

Assuming that the manifold $\mathbb{M}$ is a matrix manifold (or that its elements may be represented as matrices), a possible workaround consists in weighting every single entry of the gradient by a weight that is inversely proportional to the square root of the accumulated square of the same entry across time. In formulas, a possible workaround may be expressed as follows:

$$\begin{cases} G_t = G_{t-1} + \nabla_{U_t} J \odot \nabla_{U_t} J, \\ \hat{G}_t = \Pi_{U_t}\left[\frac{\eta}{\sqrt[\circ]{G_t + \epsilon}} \odot \nabla_{U_t} J\right], \\ U_{t+1} = \exp_{U_t}(-\hat{G}_t), \end{cases} \tag{22}$$

where $\odot$ denotes the Hadamard (component-wise) matrix product and $\sqrt[\circ]{\ }$ denotes a component-wise square root. Here, $\eta > 0$ denotes a learning stepsize and $\epsilon > 0$ is a small-valued constant that prevents division by zero. Notice that both summation of a matrix by a constant and division between two matrices are intended component-wise. The iteration step runs over $t = 0,\ 1,\ 2,\ \ldots,\ I$, where I denotes a pre-defined number of iterations. The necessity of the projection operator is quite apparent since the result of the Hadamard product $\frac{\eta}{\sqrt[\circ]{G_t+\epsilon}} \odot \nabla_{U_t} J$ between a weighting matrix and a Riemannian gradient apparently is not a tangent vector any longer. To start iteration, it is necessary to set $G_{-1} = 0$.

3.6 AdaDelta on $\mathbb{M}$

Similarly to the AdaGrad method, the original AdaDelta algorithm, as recalled in Subsection 3.6, may be extended to a Riemannian manifold (and, indeed, even to non-Riemannian smooth manifolds) in a number of ways. In the following, a mathematically sound version is proposed:

$$\begin{cases} S_t = \gamma\, S_{t-1} + (1-\gamma) \cdot (\nabla_{U_t} J \odot \nabla_{U_t} J), \\ \hat{G}_t = \Pi_{U_t} \left[\sqrt[\circ]{\frac{\Delta_{t-1}+\epsilon}{S_t+\epsilon}} \odot (\gamma\, \nabla_{U_t} J) \right], \\ \Delta_t = \gamma\, \Delta_{t-1} + (1-\gamma) \cdot (\hat{G}_t \odot \hat{G}_t), \\ U_{t+1} = \exp_{U_t}(-\hat{G}_t). \end{cases} \tag{23}$$

The first two equations are meant to provide the accumulated gradients and the accumulated updates as expected in the original AdaDelta method.

Notice that the two matrix-sequences Δ_t and S_t do not show any particular structure. Also, it is worth underlining that, in the equation that defines values of $\hat{G}$, the summation by a scalar as well as the division between matrices are meant to be effected component wise and that the Riemannian gradient was further scaled by γ to improve the numerical stability of this learning algorithm. The iteration step runs over $t = 0,\ 1,\ 2,\ \ldots,\ I$, where I denotes a pre-defined number of iterations. To start iteration, it is necessary to choose an initial guess U_0. Moreover, we set $S_{-1} = 0$ and $\Delta_{-1} = 0$.

3.7 Adaptive Gradient with Momentum on $\mathbb{M}$

The AdaM algorithm outlined in Subsection 2.8 may be extended to a smooth manifold by an appropriate reformulation of its constituting equations. The proposed extension goes as follows:

$$\begin{cases} M_t = \beta_1\, \Pi_{U_t}(M_{t-1}) + (1-\beta_1) \cdot \nabla_{U_t} J, \\ V_t = \beta_2\, V_{t-1} + (1-\beta_2) \cdot (\nabla_{U_t} J \odot \nabla_{U_t} J), \\ \hat{M}_t = \frac{M_t}{1-\beta_1^t}, \\ \hat{V}_t = \frac{V_t}{1-\beta_2^t}, \\ \hat{G}_t = \Pi_{U_t} \left[\frac{\eta}{\sqrt[\circ]{\hat{V}_t+\epsilon}} \odot \hat{M}_t \right], \\ U_{t+1} = \exp_{U_t}(-\hat{G}_t), \end{cases} \tag{24}$$

where the notations β_1^t and β_2^t denote t^{th}-order powers.

Notice that the two matrix-sequences M_t and V_t do not show any particular structure. Also, it is worth underlining that, in the equation that defines values of $\hat{G}$, the summation by a scalar as well as the division between matrices are meant to be effected component wise. The iteration step runs over $t = 0,\ 1,\ 2,\ \ldots,\ I$, where I denotes a pre-defined number of iterations. To start the iteration, it is necessary to choose an initial guess U_0. Moreover, we set $M_{-1} = 0$ and $V_{-1} = 0$.

3.8 Grassmann Manifold

The aim of this subsection is to recall some basic concepts about Grassmann manifolds. As a further reference, readers might consult Edelman, Arias, and Smith (1998) and Fiori, Kaneko, and Tanaka (2015).

Let N be a positive integer and let K be a positive integer, not greater than N. The set of K-dimensional linear subspaces of $\mathbb{R}^N$ is called *Grassmann manifold* and is denoted by $\mathrm{Gr}(N, K)$. An element on a Grassmann manifold is generally represented by an arbitrarily chosen $N \times K$ full-rank matrix U, whose columns span the corresponding subspace. Given the large arbitrariness in the choice of a basis to represent a subspace, and since $K \ll N$ in applications, a sensible choice is to restrict this selection to a 'tall skinny' orthonormal (Stiefel) matrix. A Stiefel manifold $\mathrm{St}(N, K)$ is the set of matrices of size $N \times K$ with orthonormal columns.

A Grassmann manifold can be represented by a collection of such generator matrices. Mathematically, this may be written as:

$$\mathrm{Gr}(N, K) = \{\mathrm{span}(U) \mid U \in \mathbb{R}^{N\times K}, U^\top U = I_K\}.$$

This allows to represent a generic element of Grassmann manifold as $\mathcal{U} := \{UR \mid R \in \mathrm{O}(K)\}$, where $\mathrm{O}(K)$ denotes the orthogonal group (the set of $p \times p$ orthogonal matrices). An implication of this observation is that each element of $\mathrm{Gr}(N, K)$ is an equivalence set. This allows a Grassmann manifold to be treated as a quotient space of the larger Stiefel manifold $\mathrm{St}(N, K)$. A Stiefel manifold is defined as:

$$\mathrm{St}(N, K) := \{U \in \mathbb{R}^{N\times K} \mid U^\top U = I_K\}. \tag{25}$$

Specifically, a Grassmann manifold has the quotient manifold structure

$$\mathrm{Gr}(N, K) := \mathrm{St}(N, K)/\mathrm{O}(K). \tag{26}$$

Hence, while optimization is conceptually taken on the Grassmann manifold $\mathrm{Gr}(N, K)$, the quotient-space structure numerically allows to implement operations with concrete matrices – that are elements of $\mathrm{St}(N, K)$.

Given two points on Grassmann manifold $U, \hat{U}$ and a tangent vector $V \in T_U\mathrm{Gr}(N, K)$, a Grassmann geodesic can be written as

$$\gamma_{U,V}(t) = [UB \quad A] \begin{bmatrix} \cos(\Sigma t) \\ \sin(\Sigma t) \end{bmatrix} B^\top, \tag{27}$$

where $A\Sigma B^\top$ denotes the compact singular-value decomposition of V. Then the *exponential map*, denoted as $\exp_U(V) : T_U\mathrm{Gr}(N, K) \to \mathrm{Gr}(N, K)$, can be defined as the computation of $\hat{U} = \gamma_{U,V}(1)$.

Given V and W tangent vectors to the Grassmann manifold at U, a formula for parallel translation of W along the unique geodesic in the direction V can instead be described as follows:

$$\mathrm{P}_{U,V}(W) = \left[[UB \quad A] \begin{bmatrix} -\sin(\Sigma) \\ \cos(\Sigma) \end{bmatrix} A^\top + (I - AA^\top) \right] W. \tag{28}$$

Moreover, when the elements of a Grassmann manifolds are represented through Stiefel matrices, the projection operator over tangent spaces takes the expression $\Pi_U[A] = (I - UU^\top)A$, with A being any matrix of consistent size. As it is immediate to verify, computation-wise the projection over a tangent space is much more economical than parallel transport.

With that sorted, it is now possible to extend the above gradient-based learning methods to the manifold $\mathrm{Gr}(N, K)$ as needed.

4. Sparse Spectral Clustering by Grassmann Manifold Optimization

The aim of this section is to summarize the main concepts about Grassmann manifold optimization assisted spectral clustering algorithm, which is a clustering method based on sparse spectral clustering, used to cluster a collection of multi-subspace data using sparse representation techniques. This review is based mostly on the recent paper by Wang et al. (2017).

4.1 Review of Clustering in Machine Learning

Clustering is one of the most widely used data exploration tools. Its goal is to divide the data points into several groups such that data-points that are similar fall in the same group, while data-points that are dissimilar fall in different groups. In order to achieve this result, the main steps to take are: (a) to create a similarity/affinity matrix for the given data sample set, and (b) to apply a general clustering method to categorize these data samples, such as k-means, fuzzy c-means, expectation-maximisation, hierarchical clustering, graph clustering, and NCut (Jain, Murty, & Flynn, 1999; Kang, Xu, Wang, Zhu, & Xu, 2019; Papachristou, Miaskowski, Barnaghi, Maguire, Farajidavar, Cooper, & Hu, 2016).

There exist many recently developed clustering approaches that encounter challenging optimization problems, e.g. multi-view clustering. Recently, proximity-based methods have achieved great success in multiview clustering. A paper by Liu, Huang, Wang, Fan, and Yu (2019a) suggest to consider both the intraview relation and the inter-view correlation. Furthermore, through an adaptively weighted scheme, the information of the learned view-specific proximity matrices is integrated into a view-common cluster indicator matrix. A paper by Huang, Chao, and Wang (2019a) proposes a novel multi-view clustering method termed 'multi-view intact space clustering', which is able to simultaneously recover the latent intact space from multiple insufficient views and discover the cluster structure from the intact space. The main idea of the contribution by Huang, Wang, Chao, and Yu (2019b) is to design a multiview support vector domain description model, by which the information from multiple insufficient views can be integrated, and the outputting support vectors are utilized to abstract summary statistics of historical multiview data objects.

In several applications – such as image processing and computer vision – high dimensional data are widespread. This has unpleasant affects on the computation time and

memory requirements of algorithms used to extract information. However, it has been shown that high dimensional data often lie close to low-dimensional structures corresponding to several classes or categories. The goal of spectral/subspace clustering, which is a simple extension of traditional clustering, is to cluster data points that lie in a union of low-dimensional subspaces. Spectral clustering found a number of applications, as they are parts of more complex algorithms, like community detection algorithms (Makris, Pispirigos, & Rizos, 2020), film recommender systems for users (Cintia Ganesha Putri, Leu, & Seda, 2020), and algorithms to detect open-source software ecosystems (Liao, Wang, Liu, Zhang, Liu, & Zhang, 2019).

The notion of compressive robust subspace clustering, which is to perform robust subspace clustering with compressed data, has recently been proposed by Liu, Zhang, Liu, and Xiong (2019b). Compressive robust subspace clustering is generated by projecting the original high-dimensional data onto a lower-dimensional subspace chosen at random. A noteworthy contribution in this field was presented by Zhang, Ren, Li, Hong, Zha, and Wang (2019). This paper introduced an unsupervised representation learning model, termed rBDLR, that is able to recover multi-subspace structures and extract adaptive locality-preserving salient features jointly. rBDLR jointly learns the coding coefficients and salient features, and improves the results by enhancing the robustness to outliers and errors in given data, preserving local information of salient features adaptively and ensuring the block-diagonal structures of the coefficients. A paper by Li, Zhang, Wang, Liu, Yan, and Wang (2020) explores the deep multi-subspace recovery problem by designing a multilayer architecture for latent low-rank representations. Such paper proposes a new multilayer collaborative low-rank representation network model termed to discover deep features and deep subspaces.

Sparse spectral clustering improved spectral clustering. The underlying idea behind sparse spectral clustering is what its authors call *self-expressiveness* property of the data. According to this notion, each data point in a union of subspaces can be efficiently represented as a linear or affine combination of a few key points, which form a dictionary.

4.2 Formulation of Spectral Clustering as a Constrained Optimization Problem

Assume we are given

$$X = [x_1, \ldots, x_N] \in \mathbb{R}^{P \times N}, \tag{29}$$

where X is a set of N data-points to be clustered and P is the dimension of data. The purpose of clustering is to partition the dataset X into k clusters according to certain similarity criteria. In particular, spectral clustering partitions these N points into K clusters as specified in the Algorithm 4. It is important to underline two aspects of the Algorithm 4:

- This algorithm uses the data to compute the elements of an affinity matrix W. Through the affinity matrix, a normalized graph Laplacian L is evaluated. On the basis of the graph Laplacian, a subspace basis matrix U is defined. The columns of the matrix U are taken as inputs of a clustering algorithm (upon normalization). In summary, the original data are not clustered directly, but their information content is subjected to a series of transformation steps before being given as inputs to a clustering algorithm.

Algorithm 4 Spectral Clustering (SC)

1: Compute the $N \times N$ affinity matrix W defined by

$$W_{ij} = \begin{cases} \frac{e^{-\|x_i - x_j\|^2}}{2\sigma^2} & \text{if } i \neq j, \\ 0 & \text{otherwise,} \end{cases} \tag{30}$$

where $\sigma > 0$ controls the size of each neighborhood.

2: Compute the normalized graph Laplacian

$$L = I - D^{-1/2} W D^{-1/2}, \tag{31}$$

where D is the $N \times N$ diagonal matrix whose diagonal elements are given by $d_{ii} = \sum_{j=1}^{N} w_{ij}$.

3: Compute matrix $U \in \mathbb{R}^{N \times K}$ by solving the following *constrained* problem:

$$\min_{U \in \mathbb{R}^{N \times K}} \left\langle UU^\top, L \right\rangle \text{ s.t. } U^\top U = I. \tag{32}$$

4: Form matrix $\hat{U} \in \mathbb{R}^{N \times k}$ by normalizing each row of U to have unit Euclidean length.

5: Treat each row of $\hat{U}$ as a point in $\mathbb{R}^K$, and cluster them into K groups by any clustering algorithm.

- The optimization problem (32) that defines an optimal subspace basis matrix U is constrained to seek an orthonormal rectangular matrix. This optimization problem is therefore a constrained one and, in particular, a Stiefel optimization problem, which is better tackled by manifold calculus.

The rows of matrix U are regarded as the low-dimensional representation of the original data. The elements of the matrix $UU^\top$ represent similarity (or affinity) between the latent representation (i.e., the rows) of the original data. In an ideal scenario, the matrix $UU^\top$ can be permuted to a block diagonal structure, which is privileged as it improves clustering performance.

The idea of inducing or enforcing sparsity is the basis of sparse spectral clustering. SSC tries to obtain a better representation U by solving the following sparsity-induced optimization:

$$\min_{U \in \mathbb{R}^{N \times K}} \left(\left\langle UU^\top, L \right\rangle + \beta \|UU^\top\|_1 \right) \text{ s.t. } U^\top U = I, \tag{33}$$

where β represents a weight that promotes or demotes the sparsity of the solution U. The elements of UU^T corresponding to weak inter-cluster connections tend to be zero, while the ones corresponding to strong intra-cluster connections will be kept. But the solution U of the problem (33) may not be the best one. That is why GSC algorithm has been proposed by Wang et al. (2017). We could sum it up as follows. Consider the optimization problem (33). Denote the objective function by

$$f(U) := \left\langle UU^\top, L \right\rangle + \beta \|UU^\top\|_1, \tag{34}$$

where L is the normal Laplacian graph. The constraint condition in problem (33) defines the Stiefel manifold which consists of all the orthogonal column matrices. As a consequence, problem (33) is read as a manifold optimization problem on the Stiefel manifold $\mathrm{St}(N, K)$. Due to the definition of the Grassmann manifold as a quotient space of the Stiefel manifold $\mathrm{St}(N, K)$, it is possible to restate the problem on the Grassmann manifold as follows:

$$\min_{U \in \mathrm{St}(N,K)} \left(\left\langle UU^\top, L \right\rangle + \beta \|UU^\top\|_1 \right), \tag{35}$$

that is a Grassmann manifold optimization problem. The objective function (34) of the new optimization problem (35) is not differentiable at the location where the elements of $UU^\top$ are zero. In this case, Wang et al. (2017) suggest using a sub-differential instead of gradient-based optimization.

4.3 Role of Gradient-Based Optimization in Spectral Clustering

As it was explained in Subsection 4.2, spectral clustering may be formulated in terms of a loss function $f(U)$ (34) whose minimum is sought over a Grassmann manifold. Since the solution $U = 0$ does not belong to $\mathrm{St}(N, K)$, the loss function f is smooth over the feasible set $\mathrm{St}(N, K)$. In addition, the loss function is non-linear in its argument (namely, an expansion of the function f in terms of the entry-variables in U reveals a non-linear structure of the loss function). The feasible space itself is nonlinear, since the entry-variables in U need to satisfy multiple quadratic constraints. It is therefore clear that a closed-form solution to the problem (35) is out of reach.

Since a closed-form solution to the learning problem (35) is out of question, approximate or numerical solutions may be envisaged. A widely accessed solution is based on gradient-steepest descent method and on its extensions. Since the space of feasible solutions is a large-dimensional, smooth, matrix-type, quadratic manifold, it is natural to access numerical optimization methods developed in Section 3. These optimization methods, in fact, were developed to look for a minimum of a smooth function over a constrained search space. Their structure requires the knowledge of basic operators on manifolds (such as exponential map and parallel transport) and the computation of the Riemannian gradient of the loss function. The Grassmann manifold is a well-known and well-studied smooth manifold whose principal operators are known from the scientific literature.

We are going to present some details about the computation of the Riemannian gradient of the loss function f over a Grassmann manifold. We shall recall calculations developed in (Wang et al., 2017) and then propose a more efficient way to perform the same calculation. It is worth to introduce some notation and to revise the evaluation of the gradient of the objective function (34) as pursued by Wang et al. (2017). For a matrix A of size $m \times n$, $\mathrm{vec}(A)$ is a mn-dimensional vector constructed by stacking columns of A one by one, and $\mathrm{ivec}(\mathrm{vec}(A)) = A$ the inverse operation of vec. $A \otimes B$ is the Kronecker product of matrices A and B. The transform $T_{m,n}$ is a matrix of size $mn \times mn$ such that $\mathrm{vec}(A) = T_{m,n}\mathrm{vec}(A^\top)$.

For the first term in the objective function (34), it is possible to write that:

$$\left\langle UU^\top, L \right\rangle = \mathrm{trace}(UU^\top L) = \mathrm{trace}(U^\top LU), \tag{36}$$

therefore, for the Euclidean derivative, we have that

$$\frac{\partial}{\partial U}\left\langle UU^\top, L\right\rangle = LU + L^\top U = 2LU \tag{37}$$

because L is symmetric. Consider the second term of the objective function:

$$\operatorname{vec}\left(\frac{\partial\|UU^\top\|_1}{\partial U}\right)^\top = \operatorname{vec}(\operatorname{sign}(UU^\top))^\top \frac{\partial UU^\top}{\partial U} \tag{38}$$

where

$$\frac{\partial UU^\top}{\partial U} = (I_{N^2} + T_{N^2\times N^2})(U\otimes I_N). \tag{39}$$

Define the column vector M as

$$M = \left(\frac{\partial UU^\top}{\partial U}\right)^\top \operatorname{vec}(\operatorname{sign}(UU^\top)), \tag{40}$$

hence, the Euclidean derivative of the objective function (34) is:

$$\frac{\partial f(U)}{\partial U} = 2LU + \beta\,\operatorname{ivec}(M). \tag{41}$$

We believe that the matrix $\operatorname{ivec}(M)$ could be computed in a easier and efficient way, especially in those cases where the involved matrices are large. Knowing that

$$\|UU^\top\|_1 := \sum_i \sum_j |(UU^\top)_{ij}| \tag{42}$$

and that

$$UU^\top = \sum_k U_{ik}(U^\top)_{kj} = \sum_k U_{ik}U_{jk}, \tag{43}$$

we are able to write

$$\left[\frac{\partial\|UU^\top\|_1}{\partial U}\right]_{ab} = \frac{\partial}{\partial U_{ab}} \sum_i \sum_j \left|\sum_k U_{ik}U_{jk}\right|. \tag{44}$$

We can now distinguish three cases:

$$\begin{cases} i = a,\ k = b, & \text{or} \\ j = a,\ k = b, & \text{or} \\ i = j = a,\ k = b. \end{cases}$$

Hence

$$\begin{aligned} \left[\frac{\partial\|UU^\top\|_1}{\partial U}\right]_{ab} &= \frac{\partial}{\partial U_{ab}}\left[\sum_j |U_{ab}|\,|U_{jb}| + \sum_i |U_{ib}|\,|U_{ab}|\right] \\ &= 2\sum_i |U_{ib}|\operatorname{sign}\left[\sum_k |U_{ik}|\,|U_{ak}|\right]. \end{aligned} \tag{45}$$

Algorithm 5 Grassmann Manifold Optimization Assisted Spectral Clustering (GSC)

1: Construct a $N \times N$ affinity matrix W where each element w_{ij} measures the similarity between x_i and x_j as shown in Algorithm 4.
2: Defining K as the number of clusters, compute the initial latent representation $U^{(0)}$ of size $N \times K$ by taking the first K eigenvectors corresponding to the largest K eigenvalues of the matrix W.
3: Compute the Laplacian normalized matrix L as shown in Algorithm 4.
4: Taking as initial guess $U^{(0)}$, call any appropriate optimization algorithm on the Grassmann manifold to minimize the criterion function (34).
5: With the obtained sparse latent representation U, form the new affinity matrix $\hat{W} = UU^\top$.
6: Using the affinity matrix $\hat{W}$, compute the pair-wise Euclidean distance $\Delta_{ij} = \sqrt{P_{ij}}$ where, upon defining $\mathbb{1}$ as a $1 \times K$ all-one vector, the matrix P is computed as follows:

$$\begin{cases} H = \hat{W}^\top \hat{W}, \text{ (size } K \times K) \\ Q = \operatorname{diag}(H), \text{ (size } K \times 1) \\ P = Q\,\mathbb{1} + \mathbb{1}^\top Q^\top - 2H, \text{ (size } K \times K) \\ \text{Each negative entry of the matrix } P \text{ is set to zero.} \end{cases}$$

7: Take the new pair-wise data affinity matrix W^* as $W^*_{ij} := \exp(-\Delta_{ij}/\sigma)$, where $\sigma = 0.1$, as input of any clustering algorithm to separate data into clusters.

On the basis of the above formula to calculate ivec(M), at the representative U of a Grassmann-manifold point, the Riemannian gradient of the criterion function f can be computed as:

$$\nabla_U f = (I - UU^\top)\frac{\partial f}{\partial U}. \tag{46}$$

At this stage, it is possible to use any suitable optimization algorithm on the Grassmann manifold to solve the optimization problem (35) to get a solution U.

Given a data matrix $X = [x_1, \ldots, x_N]$ and the trade-off parameter β, GSC consists of the steps described in Algorithm 5 (Wang et al., 2017). Three observations on the Algorithm 5 are in order:

- The point 4 of the algorithm calls for a numerical optimization method that is able to find the minimum of a cost function over a smooth manifold (the Grassmannian). Apparently, the subject of optimization is not a data manifold, but rather a *parameter* (or a *latent representation*) manifold.

- As optimization algorithms, we are going to employ the gradient-based optimization methods developed in Section 3. In the computation of the gradient of the criterion function f, we are going to make use of the formula (45) to evaluate the array M.

- The point 7 of the algorithm calls for a clustering method to categorize data on the basis of the new affinity matrix W^*. Wang et al. (2017) chose to use Normalized Cut.

We shall follow this line in the experimental section, hence we are going to recall the Normalized Cut method in the following subsection.

4.4 Normalized Cut (NCut)

We will now briefly summarize the basic concepts about NCut (Shi & Malik, 2000). NCut refers indeed to an objective function used in spectral clustering. We shall refer to spectral clustering based on Normalized Cut or simply to NCut clustering, for short.

The set of points in an arbitrary feature space is represented as a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the nodes of the graph are the points in the characteristic space, and an edge is built between every pair of nodes. The weight on each edge $w(i, j)$ is a function of the similarity between nodes i and j. In grouping, the set of vertexes is partitioned into disjoint sets $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m$, where the similarity among the vertexes in a set $\mathcal{V}_i$ is high and across different sets $\mathcal{V}_i, \mathcal{V}_j$ is low.

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be partitioned into two disjoint sets $\mathcal{A}, \mathcal{B}$ with $\mathcal{A} \cup \mathcal{B} = \mathcal{V}$ and $\mathcal{A} \cap \mathcal{B} = \emptyset$, by simply removing edges connecting the two parts. The degree of dissimilarity between these two partitions can be computed as the total weight of the edges that have been removed. In a graph-theoretic language, it is called *cut*:

$$\mathrm{cut}(\mathcal{A}, \mathcal{B}) := \sum_{\substack{u \in \mathcal{A} \\ v \in \mathcal{B}}} w(u, v).$$

The optimal bi-partitioning of a graph is the one that minimizes such cut value. Even though there exist a large number of such partitions, finding the minimum cut of a graph is a well-studied problem and there exist efficient algorithms for solving it (Rendl & Sotirov, 2018).

The NCut objective function measures both the total dissimilarity between different groups as well as the total similarity within each group. The NCut objective function is defined as the cut cost as a fraction of the total edge connections to all the nodes in the graph, instead of looking at the value of total edge weight connecting the two partitions:

$$\mathrm{NCut}(\mathcal{A}, \mathcal{B}) := \frac{\mathrm{cut}(\mathcal{A}, \mathcal{B})}{\mathrm{assoc}(\mathcal{A}, \mathcal{V})} + \frac{\mathrm{cut}(\mathcal{A}, \mathcal{B})}{\mathrm{assoc}(\mathcal{B}, \mathcal{V})},$$

where $\mathrm{assoc}(\mathcal{A}, \mathcal{V}) := \sum_{u \in \mathcal{A}, t \in \mathcal{V}} w(u, t)$ amounts to the total connection from nodes in $\mathcal{A}$ to all nodes in the graph and $\mathrm{assoc}(\mathcal{B}, \mathcal{V})$ is similarly defined. With this measure of disassociation between groups, the cut that partitions out small isolated points will no longer amount to a small NCut value, since the cut value will almost certainly be a large percentage of the total connection from that small set to all other nodes. NCut clustering consists of the steps specified in the Algorithm 6.

It is important to underline that, unlike the popular *K-means* clustering method, NCut does not cluster data points directly in their native data space but instead forms a similarity matrix whose (i, j)-th entry denotes a similarity distance between the i-th and j-th data points in the training set. Therefore, the NCut algorithm is more general (and powerful) because whenever *K-means* is appropriate for use then so too is NCut (just use a simple Euclidean distance as the similarity measure). The converse is not true, though.

Algorithm 6 Normalized Cut (NCut)

1: Given a training set, build a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and set the weight on the edge connecting two nodes to be a measure of the similarity between these two nodes.
2: Summarize the information into matrices W and D as in Algorithm 4.
3: Solve for eigenvectors z with the smallest eigenvalues λ of $(D - W)z = \lambda Dz$.
4: Use the eigenvectors with the second smallest eigenvalue to bi-partition the graph by finding the splitting point such that the NCut value is minimized.
5: Decide if the current partition should be subdivided by checking the stability of the cut, and make sure that the NCut value lies below a pre-specified threshold.
6: Recursively re-partition the segmented parts if necessary.

5. Experimental Results

The present section illustrates results of numerical experiments performed on two categories of datasets, namely, synthetic 2-dimensional datasets used for testing purposes (Subsection 5.1, and real-world datasets used to validate the discussed gradient-based learning algorithms (Subsection 5.2) and to compare their performances with those exhibited by closely-related clustering algorithms known from the scientific literature (Subsection 5.3).

5.1 Clustering Results on Synthetic Datasets

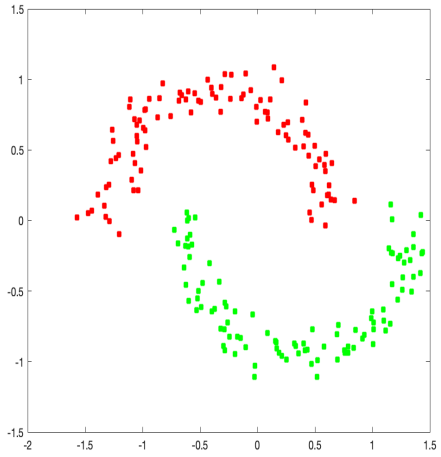
The synthetic datasets used in this section are drawn from (Wang et al., 2017), namely:

- **Two-moon data**: These data are randomly generated from two sine-shape curves with the noise percentage set to 0.09, and each cluster – in this specific case – contains 100 samples.
- **Three-Gaussian data**: Each cluster follows a Gaussian distribution with a variance of 0.05. Again, each cluster has 100 samples.
- **Three-ring data**: These data are distributed on circles, with the noise percentage set to 0.15. There are respectively 100, 100 and 150 samples in each cluster.
- **Two disjoint quadratic para-curves data**: These data are spread throughout two disjoint parabolic-shape curves without overlapping, altered with Gaussian noise of 0 mean and variance 0.05. In our experiment each cluster contains 200 samples.

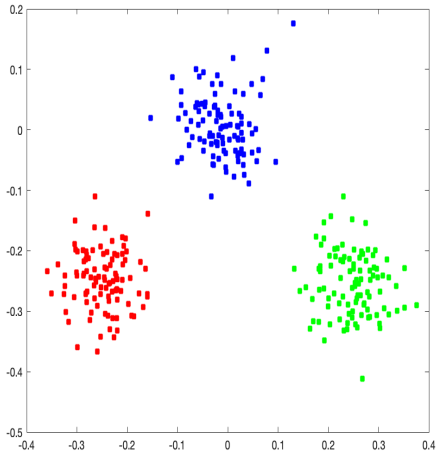
These datasets are shown in Figure 1, with the clusters colored. These numerical experiments were performed on a personal computer endowed with a dual-core Intel Core i5 processor, a clock frequency of 2.7GHz and a 8GB RAM by the help of MATLAB R2017b scripts.

5.1.1 Preliminary Illustrative Tests

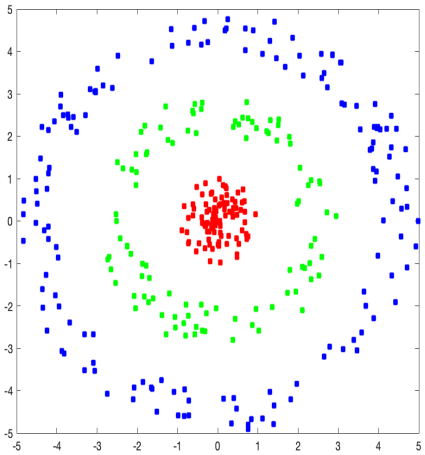
As a preliminary test, before discussing and comparing the efficiency of the clustering algorithms on each dataset, it is interesting to try directly the *NCut* method without extracting any latent representation in advance. The results are shown in Figure 2. The accuracy



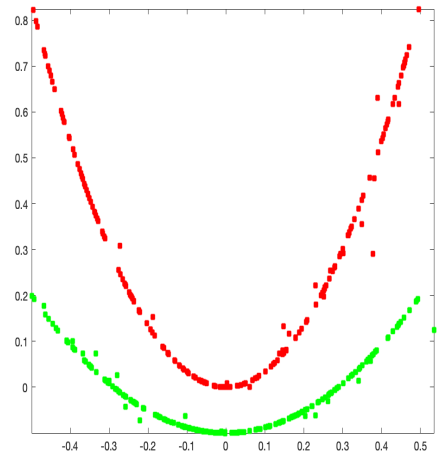
(a) *Two-Moon dataset.*



(b) *Three-Gaussian dataset.*



(c) *Three-Ring dataset*

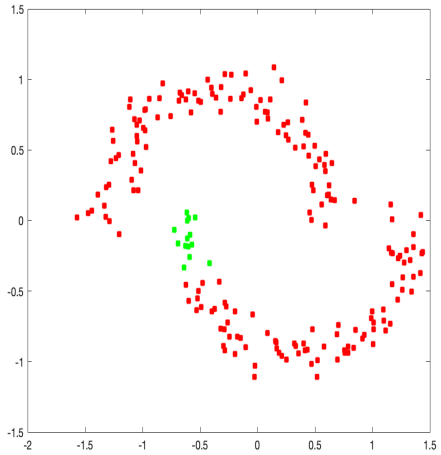


(d) *Two disjoint quadratic para-curves dataset*

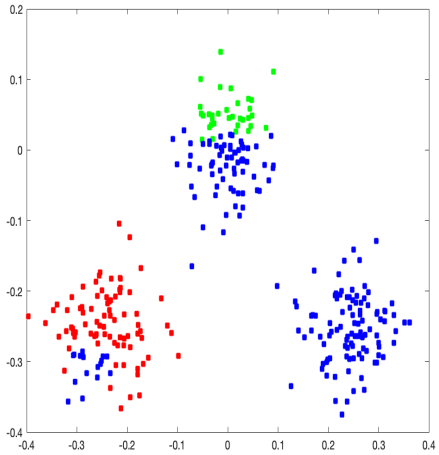
Figure 1: Synthetic datasets used to test clustering algorithms.

of NCut when the affinity matrix is not learned appears to be very low, which justifies summoning a learning algorithm to discover a useful latent representation of the original data.

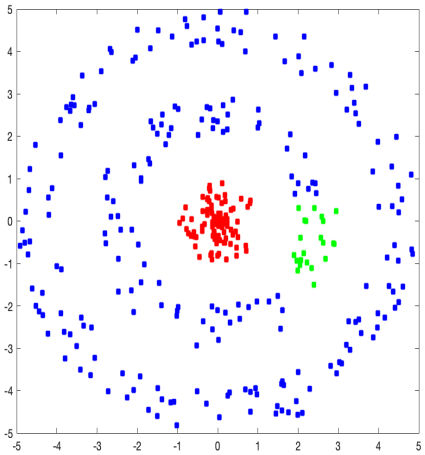
As a further preliminary test, we assessed the efficiency in the computation of the vector M using the expression (40) given in Wang et al. (2017) versus our improved expression (45). In order to get a fair comparison, we used as learning algorithm the *Trust Regions* method from the ManOpt tool (Boumal, Mishra, Absil, & Sepulchre, 2014). Results obtained by using (40) are summarized in Table 1, while results obtained by using expression (45) are summarized in Table 2. These tables show the number of iterations and the running time



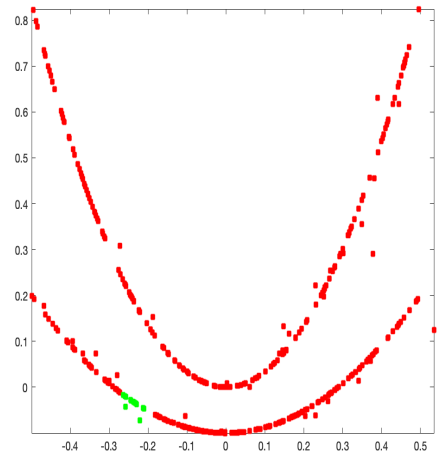
(a) *NCut with Two-moon dataset.*



(b) *NCut with Three-Gaussian dataset.*



(c) *NCut with Three-ring dataset.*



(d) *NCut with Two disjoint para-curves dataset.*

Figure 2: Clustering results obtained by the *NCut* algorithm without learning any affinity matrix.

that an algorithm takes to get a 100% clustering accuracy for each dataset. As we can see, our improvement in the way to compute M (45) makes the algorithm much faster.

Notice that the number of iterations is less meaningful than the running time. In fact, the more complex the toy dataset is, the longer an algorithm takes to converge properly, regardless of whether the number of iterations is slightly or significantly higher than the number of iterations taken by other datasets.

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	17	60.75
Three-Gaussian	8	162.87
Three-Ring	11	304.10
Two Disjoint Para-Curves	33	1047.79

Table 1: Comparison of the performance indexes of the TrustRegions (ManOpt tool) using (40).

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	12	4.59
Three-Gaussian	5	2.59
Three-Ring	10	3.42
Two Disjoint Para-Curves	11	4.39

Table 2: Comparison of the performance indexes of the TrustRegions with our improvement in computation of M (45).

5.1.2 Comparison of Six Gradient-Based Learning Methods on Synthetic Training Sets

We shall now move on to gradient-based learning methods. The following results were obtained by choosing, as calculation method for the vector M, the expression (45).

The performances of the *Gradient descent* learning algorithm on four synthetic datasets are summarized in Table 3. In this table, as well as in the following ones, the # iterations refers to a pre-fixed number of learning cycles that guarantees complete convergence. It is clear that the convergence of this algorithm is very slow.

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	200,000	567.05
Three-Gaussian	500,000	1,463.19
Three-Ring	750,000	9,617.48
Two Disjoint Para-Curves	500,000	5,277.69

Table 3: Comparison of the performance indexes of the GD-based learning algorithm on four synthetic datasets.

The performances of the *Momentum* learning algorithm on four toy datasets are summarized in Table 4. *Momentum* is much faster than *Gradient descent*, but it appears quite slow in converging to an appropriate latent representation, in absolute terms.

The performances of the *Nesterov accelerated gradient* learning algorithm on four toy datasets are summarized in Table 5. As expected, *Nesterov accelerated gradient* is similar to *Momentum* in terms of performance, although it appears to be slightly faster as the complexity of the input data increases.

The performances of the *Adaptive gradient* learning algorithm on four synthetic datasets are summarized in the Table 6. Adaptive methods turn out to be much faster, as it is possible

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	3,500	12.42
Three-Gaussian	1,500	18.28
Three-Ring	250,000	3,023.19
Two Disjoint Para-Curves	300,000	3,675.03

Table 4: Comparison of the performance indexes of the Momentum-based learning algorithm on four synthetic datasets.

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	6,500	21.86
Three-Gaussian	1,000	13.04
Three-Ring	10,000	156.30
Two Disjoint Para-Curves	10,000	173.70

Table 5: Comparison of the performance indexes of the NAG-based learning algorithm on four synthetic datasets.

to notice by observing the performance of the *AdaGrad*, which is however the slowest of them.

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	2,000	19.58
Three-Gaussian	200	8.92
Three-Ring	300	15.34
Two Disjoint Para-Curves	5,000	77.59

Table 6: Comparison of the performance indexes of the AdaGrad-based learning algorithm on four synthetic datasets.

The performances of the *AdaDelta* learning algorithm on four toy datasets are summarized in Table 7. *AdaDelta* is able to further improve over *AdaGrad* performances as it turns out to converge faster.

The performances of the *AdaM* learning algorithm on four toy datasets are summarized in Table 8. *AdaM* clearly appears to be the fastest learning method in these experiments, whatever the complexity of the input data is.

Apparently, the adaptive methods are much faster than the non-adaptive gradient-based methods. In particular, the *AdaM* learning algorithm converges the fastest, whereas the *GD* – as well as *Momentum* and *NAG* in the case of the Three-ring dataset – appear to be inappropriate because of their slowness. To better illustrate this observation, the Figure 3 shows a comparison of learning curves of six learning algorithms on a Two-moon dataset. The number of iterations is limited to 350, which corresponds to the number of iterations needed by the *AdaM* algorithm to converge to a sparse matrix U that guarantees 100% clustering accuracy.

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	1,000	5.91
Three-Gaussian	100	2.41
Three-Ring	400	7.08
Two Disjoint Para-Curves	3,500	32.83

Table 7: Comparison of the performance indexes of the Adadelta-based learning algorithm on four synthetic datasets.

Dataset	**# iterations**	**Running time** (sec)
Two-Moon	350	3.74
Three-Gaussian	100	4.34
Three-Ring	100	6.06
Two Disjoint Para-Curves	500	14.55

Table 8: Comparison of the performance indexes of the AdaM-based learning algorithm on four synthetic datasets.

5.1.3 Influence of the Trade-Off Parameter β on Clustering Ability of GSC

The clustering accuracy of the classical *NCut* algorithm on two-moon, three-Gaussian, three-ring datasets is 56.50%, 60.10%, and 86.00%, respectively. We evaluated the influence of the trade-off parameter β on the performance of *GSC* learnt by different adaptive methods, namely *AdaGrad*, *AdaDelta* and *AdaM*. We used for each optimization method a fixed number of iterations corresponding to the iterations they take to achieve 100% classification accuracy with $\beta = 0.00001$. Tables 9, 10, 11 and 12 show clustering performance versus parameter β on the toy datasets.

Trade-off parameter	**AdaGrad**	**AdaDelta**	**AdaM**
$\beta = 0.00001$	**100%**	**100%**	**100%**
$\beta = 0.00005$	92.75%	**96.50%**	**96.50%**
$\beta = 0.0001$	75.75%	**96.50%**	86.41%
$\beta = 0.0005$	62.41%	66.71%	**78.50%**
$\beta = 0.001$	53.50%	54.25%	**61.41%**
$\beta = 0.005$	53.50%	52.75%	**57.25%**
$\beta = 0.01$	48.41%	49.71%	**51.73%**

Table 9: Clustering accuracy of GSC – learnt, respectively, by AdaGrad, AdaDelta and AdaM – against different β on Two-Moon dataset.

For the two disjoint para-curve dataset, the clustering accuracy of *NCut* method is 54.50%. The best result for GSC method is 100% when $\beta = 0.000001$. Results reported in these tables show that *GSC* when a latent representation is learned by *AdaM* is much more robust, compared to the other considered learning methods against variations of the parameter β.

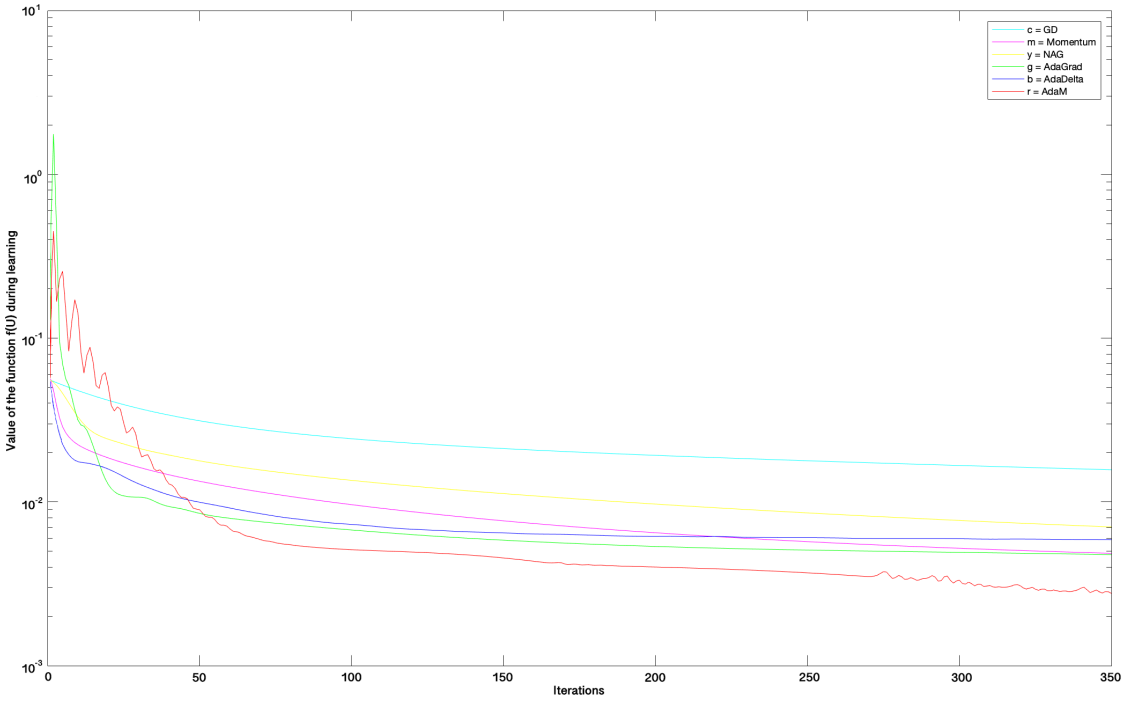


Figure 3: Complete view of learning curves of six algorithms on a Two-moon dataset.

Trade-off parameter	AdaGrad	AdaDelta	AdaM
$\beta = 0.00001$	**100%**	**100%**	**100%**
$\beta = 0.00005$	**100%**	**100%**	**100%**
$\beta = 0.0001$	**100%**	99.75%	**100%**
$\beta = 0.0005$	84.33%	89.50%	**95.70%**
$\beta = 0.001$	60.33%	74.41%	**89.71%**
$\beta = 0.005$	47.25%	74.41%	**88.50%**
$\beta = 0.01$	42.75%	61.71%	**84.33%**

Table 10: Clustering accuracy of GSC – learnt, respectively, by AdaGrad, AdaDelta and AdaM – against different β on Three-Gaussian dataset.

5.1.4 Visual Representation of Learned Affinity Matrix

In order to examine the underlying low-dimensional structure within data, we supply here a visual comparison of affinity matrix for every synthetic training set used in this section.

Figures 4 and 5 show the affinity matrix W as computed by the *NCut* algorithm and by the *GSC* clustering algorithm (learnt by an *AdaM* algorithm), respectively. Notice that the affinity matrix obtained using *AdaM* has the same structure we could get by employing any of the other optimization methods proposed in this paper. The affinity matrix obtained by *GSC* effectively reveals the cluster structure of data.

Trade-off parameter	AdaGrad	AdaDelta	AdaM
$\beta = 0.00001$	**100%**	**100%**	**100%**
$\beta = 0.00005$	**100%**	**100%**	**100%**
$\beta = 0.0001$	**100%**	**100%**	**100%**
$\beta = 0.0005$	79.71%	77.14%	**88.50%**
$\beta = 0.001$	69.71%	64.41%	**88.50%**
$\beta = 0.005$	58.43%	55.33%	**83.41%**
$\beta = 0.01$	58.43%	53.65%	**83.41%**

Table 11: Clustering accuracy of GSC – learnt, respectively, by AdaGrad, AdaDelta and AdaM – against different β on Three-Ring dataset.

Trade-off parameter	AdaGrad	AdaDelta	AdaM
$\beta = 0.000001$	**100%**	**100%**	**100%**
$\beta = 0.000005$	**100%**	**100%**	**100%**
$\beta = 0.00001$	95.75%	96.25%	**97.75%**
$\beta = 0.00005$	65.41%	68.71%	**79.50%**
$\beta = 0.0001$	56.75%	56.75%	**71.40%**
$\beta = 0.0005$	54.50%	55.33%	**61.71%**
$\beta = 0.001$	31.43%	34.66%	**46.75%**

Table 12: Clustering accuracy of GSC - learnt, respectively, by AdaGrad, AdaDelta and AdaM - against different β on Two Disjoint Para-Curves dataset.

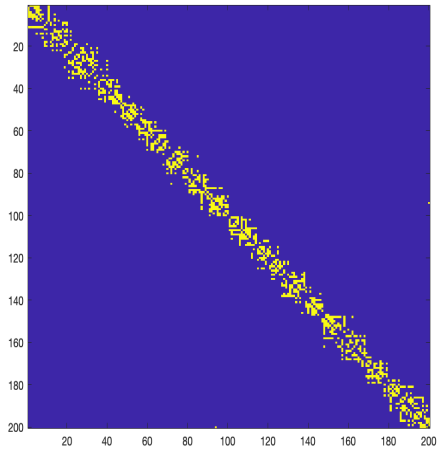
5.1.5 Summary of Results on Clustering Synthetic Datasets

From the results illustrated in the present subsection it is possible to draw some interesting observations:

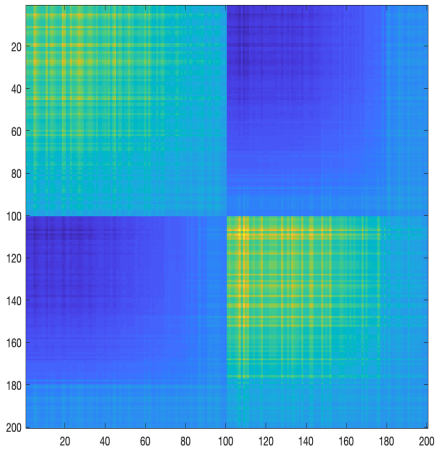
- Thanks to our adjustments in the calculation of the gradient of the loss function (equation (45)), the GSC algorithm appears to be clearly faster: looking at Tables 1 and 2, it is clear that the running time of the GSC clustering algorithm is considerably lower.
- The clustering results obtained by the GSC algorithm are way better than those obtained by the *NCut* algorithm alone, for each synthetic dataset and each optimization algorithm considered.
- *AdaM* appears to be the fastest optimization method among those considered. Especially for the complicated three-Gaussian, three-ring and two disjoint para-curves datasets, *AdaM* outperforms the other learning algorithms and is also much more robust versus variations of the sparse regularization parameter β.

5.2 Clustering Results on Three Pictorial Datasets

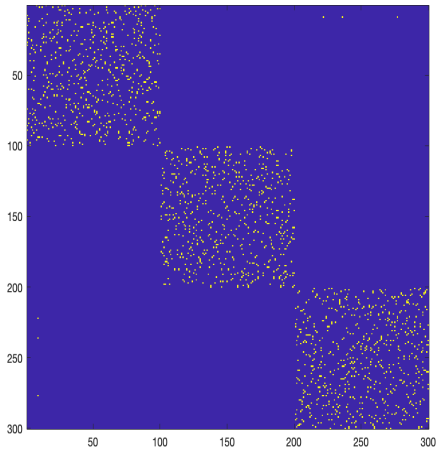
In this subsection, we perform some experiments on public databases to evaluate the performances of the proposed optimization methods on real-world dataset. All experiments are conducted on the following three public available datasets:



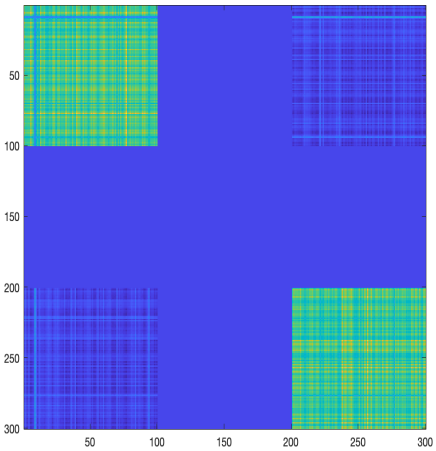
(a) *NCut with Two-Moon dataset.*



(b) *GSC with Two-Moon dataset.*



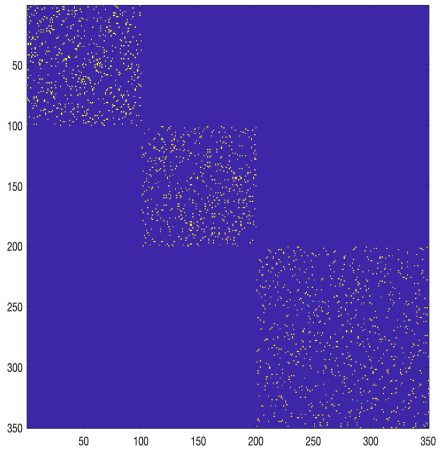
(c) *NCut with Three-Gaussian dataset.*



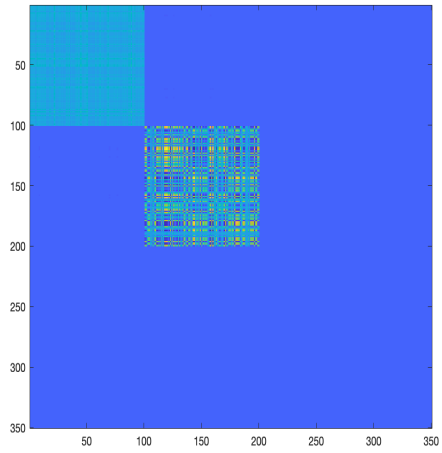
(d) *GSC with Three-Gaussian dataset.*

Figure 4: Comparison of the affinity matrix W computed on Two-Moon and Three-Gaussian Dataset: (a), (c) were obtained using *NCut*; (b), (d) were obtained using *GSC* learnt by *AdaM*.

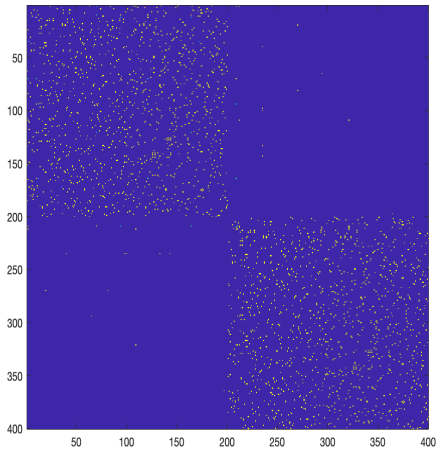
- **The YaleB face database**. The *YaleB dataset* consists of 192 × 168 pixel cropped face images of 38 individuals, where there are 64 frontal face images for each subject acquired under various lighting conditions. Some sample face images are shown in Figure 6(a). To reduce the computational cost and the memory requirements of all algorithms, we down-sampled the images to 32 × 32 pixels and treat each 1,032-dimensional vectorized image as a data point (`http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html`).



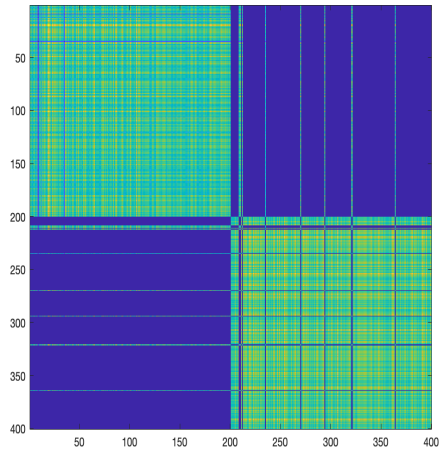
(a) *NCut with Three-Ring dataset.*



(b) *GSC with Three-Ring dataset.*



(c) *NCut with Two-ParaCurves dataset.*



(d) *GSC with Two-ParaCurves dataset.*

Figure 5: Comparison of the affinity matrix W computed on the Three-Ring and Two Disjoint Para-Curves datasets: (a), (c) were obtained using *NCut*; (b), (d) were obtained using *GSC* learned by *AdaM*.

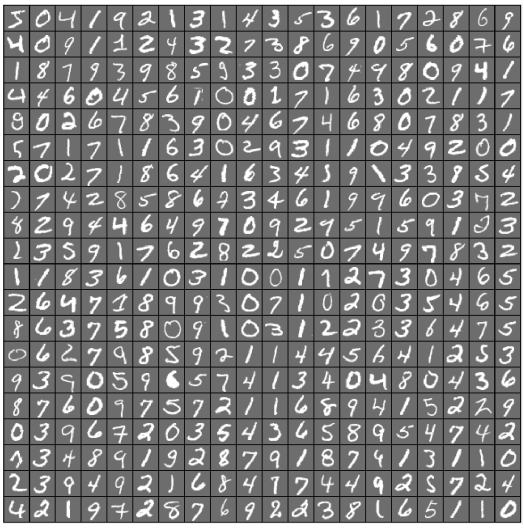
- **The ORL face database**. The *ORL dataset* is composed of 400 images of size 112 $\times$ 92, and some samples are shown in Figure 6(b). There are 10 different images of 40 distinct subjects. For some of the subjects, the images were taken at different times, varying lighting slightly, facial expressions (open/closed eyes, smiling/non-smiling) and facial details (glasses/no-glasses). All the images are taken against a dark homogeneous background and the subjects are in up-right, frontal position (with tolerance for some side movement). All the data is collected in a matrix of shape 10,304

(pixels) × 400 (faces). To avoid large values, the data matrix is divided by 100 (`https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`).

- **The MNIST database**. The *MNIST dataset* of handwritten digits has a training set of 60,000 examples and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image. This dataset contains 600 images of each digit. All images are normalized to fit into a 28 × 28 pixel bounding box and anti-aliased (`http://yann.lecun.com/exdb/mnist/`).

(a) *Examples of Extended Yale B Dataset.*

(b) *Examples of ORL faces Dataset.*



(c) *Examples of MNIST handwritten digits.*

Figure 6: Examples of the YaleB, ORL and MNIST datasets, respectively.

5.2.1 Comparison of TrustRegion and AdaM on Facial Datasets

We applied the method by Elhamifar and Vidal (2013) to construct the affinity matrix W by l_1-graph technique and subsequently applied GSC learnt by ManOpt-Trustregions and AdaM on the constructed affinity matrix for the two face (YaleB and ORL) datasets. Six subsets were constructed which consist of images of randomly selected subjects classified in K clusters for $K \in \{5, 8, 10, 12, 15, 18\}$. We set the same value $\beta = 0.00001$ and let gradient-based learning algorithms run over $1,500$ iterations.

The obtained results are summarized in Tables 13 and 14. They show how close is the accuracy reached by GSC learnt both by Trustregions and AdaM, although AdaM seems to reach a better accuracy with relatively low K whereas Trustregions is more reliable on a larger number of clusters.

Clusters	**TrustRegions**	**AdaM**
$K = 5$	96.50%	**97.15%**
$K = 8$	**91.65%**	**91.65%**
$K = 10$	85.24%	**86.33%**
$K = 12$	81.31%	**81.71%**
$K = 15$	**77.82%**	76.91%
$K = 18$	**74.61%**	74.51%

Table 13: Clustering results in terms of accuracy of GSC learned by TrustRegions and by AdaM on YaleB dataset.

Clusters	**TrustRegions**	**AdaM**
$K = 5$	**97.60%**	**97.60%**
$K = 8$	93.50%	**94.25%**
$K = 10$	**82.77%**	82.67%
$K = 12$	81.80%	**81.95%**
$K = 15$	**79.67%**	**79.67%**
$K = 18$	**78.86%**	77.95%

Table 14: Clustering results in terms of accuracy of GSC learned by TrustRegions and by AdaM on ORL dataset.

5.2.2 Comparison of TrustRegions and AdaGrad, AdaDelta and AdaM on Digits Datasets

Further, we compared the performance of GSC learned by ManOpt-TrustRegions and adaptive methods. We set the sparsity-promotion β to 0.00001 and the number of clusters K to 5, 8 and 10. Each training set was build up so as to contain a total of 400 images randomly selected from the same cluster.

As Table 15 demonstrates, GSC algorithm learned by AdaM outperforms GSC algorithm learned by all the other methods, even the TrustRegions.

Clusters	TrustRegions	AdaGrad	AdaDelta	AdaM
$K = 5$	96.57%	93.71%	94.15%	**97.50%**
$K = 8$	89.41%	85.69%	87.61%	**89.61%**
$K = 10$	**79.24%**	73.21%	75.86%	**79.24%**

Table 15: Clustering results in terms of accuracy of GSC learned by TrustRegions and by adaptive-methods on MNIST dataset.

5.2.3 Summary of Results on Clustering Pictorial Datasets

From the results illustrated in the present subsection it is possible to draw some interesting observations:

- The AdaM optimization algorithm outperforms the other adaptive gradient algorithms developed in Section 2. In addition, the AdaM algorithm outperforms the TrustRegions up to 10 classes on the YaleB dataset, up to 15 classes on the ORL dataset (Table 14 shows an exception for $K = 10$ which breaks the trend: we believe that a difference in performance of 0.1% is a meaningless statistical fluctuation) and up 10 classes on the MNIST dataset. It is worth underlining that, on the YaleB dataset, the performance discrepancy for $K = 18$ is also of 0.1%.

- The value chosen for the sparsity-promotion parameter β, which was validated on synthetic datasets, proved satisfactory even for real-world pictorial datasets.

5.3 Comparative Results

Comparative experiments have been performed on the following 10 publicly available pictorial datasets:

- **Recursion Cellular Image Classification** (in the experiments: 'Recursion'). Data from the Recursion 2019 challenge. The goal of the competition was to use biological microscopy data to develop a model that identifies replicates (`https://www.kaggle.com/xhlulu/recursion-cellular-image-classification-224-jpg`).

- **TensorFlow Patch Camelyon Medical Images** (in the experiments: 'Camelyon'). Medical image classification dataset from the TensorFlow website. This medical image classification dataset comes from the TensorFlow website. It contains just over 327,000 color images. The images are histopathological lymph node scans which contain metastatic tissue (`https://www.tensorflow.org/datasets/catalog/patch_camelyon`).

- **CoastSat Image Classification Dataset** (in the experiments: 'CoastSat'). Used for an open-source shoreline mapping tool, this dataset includes aerial images taken from satellites. The dataset also includes meta data pertaining to labels (`https://figshare.com/articles/CoastSat_image_classification_training_data/8868665/1`).

- **Images for Weather Recognition** (Ajayi, 2018) (in the experiments: 'Weather'). Used for multi-class weather recognition, this dataset is a collection of 1125 images divided into four categories: sunrise, shine, rain, and cloudy (`https://data.mendeley.com/datasets/4drtyfjtfy/1`).

- **Indoor Scenes Images** (in the experiments: 'Indoor'). From MIT, this dataset contains over 15,000 images of indoor locations. This dataset was originally built to tackle the problem of indoor scene recognition. All images have been divided into 67 categories. The number of images per category vary, however, there are at least 100 images for each category (`https://www.kaggle.com/itsahmad/indoor-scenes-cvpr-2019`).

- **Intel Image Classification** (in the experiments: 'Intel'). Created by Intel for an image classification contest, this image dataset contains approximately 25,000 images divided into the following categories: buildings, forest, glacier, mountain, sea, and street. The training set includes around 14,000 images and the testing folder has around 3,000 images (`https://www.kaggle.com/puneet6060/intel-image-classification/version/2`).

- **TensorFlow Sun397 Image Classification Dataset** (in the experiments: 'Sun397'). A dataset from Tensorflow that contains over 108,000 images used in Scene Understanding (SUN) benchmark. These images have been divided into 397 categories. The exact amount of images in each category varies. However, there are at least 100 images in each of the various scene and object categories (`https://www.tensorflow.org/datasets/catalog/sun397`). Most of these images were collected from *Flickr* and *Wikimedia Commons* (all of them under creative commons license). For further details on this dataset, see Xiao, Hays, Ehinger, Oliva, and Torralba (2010).

- **Architectural Heritage Elements** (in the experiments: 'Heritage'). This dataset was created to train models that could classify architectural images, based on cultural heritage. It contains over 10,000 images divided into 10 categories, namely altar, apse, bell tower, column, dome (inner), dome (outer), flying buttress, gargoyle, stained glass, and vault (`https://old.datahub.io/dataset/architectural-heritage-elements-image-dataset`).

- **Images of People Eating Food** (in the experiments: 'Eating'). This dataset consists of images of people eating fruits, cakes, and other foods. Human annotators classified the images by gender and age (`https://data.world/crowdflower/image-classification-people-an`).

- **Images of Cracks in Concrete for Classification** (in the experiments: 'Concrete'). From Mendeley, this dataset was collected from various METU Campus Buildings and includes 40,000 images of concrete slabs, where half of the images include concrete with cracks and half without. High-resolution images exhibit variance in terms of surface finishing and illumination conditions. (`https://data.mendeley.com/datasets/5y9wdsg2zt/2`). For further details on this dataset, see Özgenel (2019) and Zhang, Yang, Daniel Zhang, and Zhu (2016).

The comparison was effected on the basis of the following low-rank subspace clustering methods:

- **LRR**: A low-rank representation and sparse coding-based subspace clustering method that simultaneously considers feature information and spatial structures. LRR seeks the lowest rank representation over original spatial structures. Sparse coding learns a dictionary along feature spaces, so that each sample can be represented by a few atoms of the learned dictionary. The affinity matrix used for spectral clustering is built from the joint similarities in both spatial and feature spaces (Fu, Gao, Tien, Lin, & Hong, 2016).

- **LRSC**: Solves the problem of fitting a union of subspaces to a collection of data points drawn from one or more subspaces and corrupted by noise and/or gross errors. For one subspace, a particular case of the LRSC framework leads to classical PCA. For multiple subspaces, the low-rank coefficients obtained by the LRSC framework can be used to construct a data affinity matrix from which clustering of data can be obtained by spectral clustering (Vidal & Favaro, 2014).

- **LatLRR**: Latent Low-Rank Representation (LatLRR) delivers robust and promising results for subspace recovery and feature extraction through mining hidden effects (Fang, Han, Wu, Xu, Yang, Wong, & Li, 2018).

- **rLRR**: LatLRR is unable to preserve the locality of both similar principal and salient features. To solve this issue, a boosted version of LatLRR, referred to as Regularized Low-Rank Representation (rLRR), was proposed through explicitly including an appropriate Laplacian regularization that can maximally preserve the similarity among local features (Zhang, Yan, & Zhao, 2014).

- **AdaM-based GSC**: The clustering method based on GSC with an AdaM latent representation extraction engine.

The classification results were evaluated on the basis of the following indexes:

- **Jaccard index**: The Jaccard index is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard index measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets.

- **Adjusted Rand index** (ARI): The Rand index in statistics, and in particular in data clustering, is a measure of similarity between two data clusters. A form of the Rand index may be defined that is adjusted for the chance grouping of elements, this is the adjusted Rand index.

- **Fowlkes-Mallows index**: The Fowlkes–Mallows index is an external evaluation method used to determine the similarity between two clusters. This measure of similarity could be either between two hierarchical clusterings or a clustering and a benchmark classification. A higher value for the Fowlkes–Mallows index indicates a greater similarity between the clusters and the benchmark classifications.

- **Normalized mutual information** (NMI): We also consider the NMI as a quality metric because it quantifies the mutual dependence between two random variables based on well-established concepts of information theory.

We refer readers to, e.g., Rodriguez, Comin, Casanova, Bruno, Amancio, Costa, and Rodrigues (2019) for detailed definitions. As a baseline, when the two sets of labels have a perfect one-to-one correspondence, the quality measures are all equal to 1.

Results of comparative experiments between five clustering methods on ten datasets, evaluated in terms of Jaccard index, are summarized in Table 16. The result of this evaluation confirms that the AdaM-GSC outperforms the other clustering algorithms considered in this comparison on every dataset, except for the 'Concrete' dataset where the LRSC algorithm slightly outperforms AdaM-GSC.

Dataset	**LRR**	**LRSC**	**LatLRR**	**rLRR**	**AdaM-GSC**
Recursion	0.7496	0.7106	0.6822	0.7534	**0.8198**
Camelyon	0.7364	0.6711	0.7159	0.7745	**0.7782**
CoastSat	0.7503	0.7501	0.6807	0.7263	**0.7622**
Weather	0.7549	0.7234	0.7438	0.7012	**0.8258**
Indoor	0.6555	0.6535	0.7591	0.6928	**0.8054**
Intel	0.6604	0.7417	0.7478	0.7106	**0.7596**
Sun397	0.6946	0.7071	0.6838	0.7037	**0.8270**
Heritage	0.6961	0.6958	0.6568	0.661	**0.8457**
Eating	0.7161	0.7600	0.6823	0.6555	**0.7631**
Concrete	0.6652	**0.7669**	0.6562	0.6683	0.7661

Table 16: Results of comparative experiments between five clustering methods on ten datasets, evaluated in terms of Jaccard index.

Clustering results evaluated in terms of ARI are summarized in Table 17. The result of this evaluation confirms that the AdaM-GSC outperforms the other clustering algorithms on every dataset, except for the 'Camelyon' dataset, where the LatLRR algorithm slightly outperforms AdaM-GSC, and for the 'Weather' dataset, where rLRR slightly outperforms AdaM-GSC.

Clustering results evaluated according to the Fowlkes-Mallows index are summarized in Table 18. This evaluation confirms that the AdaM-GSC outperforms other clustering algorithms on every dataset.

Clustering results evaluated in terms of the NMI index are summarized in Table 19. This evaluation confirms that the AdaM-GSC outperforms other clustering algorithms on almost every dataset, except for the 'Indoor' experiment, where 'LatLRR' outperforms AdaM-GSC.

Results across the above four tables look coherent to one another. Also, this evaluation confirms that the AdaM-GSC clustering method is able to classify samples in the considered datasets with a higher degree of accuracy in almost every experiment.

Dataset	LRR	LRSC	LatLRR	rLRR	AdaM-GSC
Recursion	0.6889	0.6780	0.6515	0.7105	**0.7930**
Camelyon	0.6812	0.6836	**0.7768**	0.7642	0.7576
CoastSat	0.7563	0.6518	0.6622	0.7596	**0.7634**
Weather	0.6546	0.6799	0.7172	**0.7686**	0.7633
Indoor	0.6956	0.7440	0.7733	0.7069	**0.8062**
Intel	0.7180	0.6682	0.6635	0.6588	**0.8004**
Sun397	0.7016	0.7482	0.7092	0.7156	**0.7629**
Heritage	0.7743	0.6794	0.7490	0.6806	**0.8232**
Eating	0.7208	0.6942	0.6728	0.7684	**0.8310**
Concrete	0.7466	0.6724	0.7095	0.7490	**0.8443**

Table 17: Results of comparative experiments between five clustering methods on ten datasets, evaluated in terms of ARI.

Dataset	LRR	LRSC	LatLRR	rLRR	AdaM-GSC
Recursion	0.6958	0.6854	0.7448	0.7005	**0.8175**
Camelyon	0.7068	0.6961	0.7092	0.7708	**0.8450**
CoastSat	0.6504	0.7174	0.6819	0.7389	**0.8434**
Weather	0.6782	0.6563	0.7665	0.7126	**0.8066**
Indoor	0.7739	0.7413	0.7220	0.6977	**0.8325**
Intel	0.7626	0.6517	0.7201	0.7603	**0.8409**
Sun397	0.7638	0.7620	0.7317	0.7398	**0.8167**
Heritage	0.7316	0.7270	0.6605	0.7146	**0.7564**
Eating	0.7337	0.7390	0.6908	0.7302	**0.8364**
Concrete	0.6822	0.7512	0.7089	0.6955	**0.8055**

Table 18: Results of comparative experiments between five clustering methods on ten datasets, evaluated in terms of Fowlkes-Mallows index.

6. Conclusion

This paper studied and extended a number of classical and modern gradient-based learning methods to a general smooth manifold. After a quick overview of the spectral clustering world, we also examined the GSC model which adopts Grassmann manifold optimization strategy to optimize the sparse spectral clustering objective in a straight-forward way, and found out a way to make it converge faster. Extensive experiments conducted on both toy datasets and several real-world databases demonstrated the effectiveness of adaptive methods, in particular of AdaM.

Acknowledgments

The present research work was completed when the author AK was taking an internship at the Tokyo University of Agriculture and Technology (TUAT, Koganei campus) thanks to a scholarship of the Università Politecnica delle Marche (CampusWorld Program 2019)

Dataset	LRR	LRSC	LatLRR	rLRR	AdaM-GSC
Recursion	0.7585	0.6921	0.7535	0.7314	**0.7783**
Camelyon	0.7485	0.7324	0.7122	0.6728	**0.8301**
CoastSat	0.7498	0.7679	0.6528	0.7507	**0.8424**
Weather	0.6871	0.7133	0.7206	0.7396	**0.8388**
Indoor	0.6764	0.6512	**0.7711**	0.6959	0.7563
Intel	0.7198	0.7016	0.7747	0.6522	**0.7665**
Sun397	0.6614	0.7044	0.7726	0.7421	**0.7877**
Heritage	0.7365	0.712	0.7592	0.6947	**0.8045**
Eating	0.7307	0.6753	0.748	0.6658	**0.8149**
Concrete	0.7537	0.7236	0.7024	0.7784	**0.8343**

Table 19: Results of comparative experiments between five clustering methods on ten datasets, evaluated in terms of NMI.

during March-May 2019. The authors wish to gratefully thank Prof. Toshihisa Tanaka who made this internship possible. The present research was advanced while the author SF was visiting the TUAT, during April 2019, thanks to a TUAT visiting professor scholarship. The authors wish to thank Prof. Junbin Gao (The University of Sydney) for sharing part of the computer codes to implement the GSC algorithm, as well as Dr. Ehsan Elhamifar (University of California at Berkeley) and Prof. René Vidal (The Johns Hopkins University) for sharing part of the computer codes to implement sparse clustering on pictorial data.

References

Ajayi, G. (2018). Multi-class weather dataset for image classification. `http://dx.doi.org/10.17632/4drtyfjtfy.1#file-b3b8a956-4bcb-431d-bda0-d466af180d2e`.

Bonnabel, S. (2013). Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, *58*(9), 2217–2229.

Botev, A., Lever, G., & Barber, D. (2017). Nesterov's accelerated gradient and momentum as approximations to regularised update descent. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1899–1903.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Lechevallier, Y., & Saporta, G. (Eds.), *Proceedings of COMPSTAT'2010*, pp. 177–186, Heidelberg. Physica-Verlag HD.

Boumal, N., Mishra, B., Absil, P.-A., & Sepulchre, R. (2014). Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, *15*, 1455–1459.

Cintia Ganesha Putri, D., Leu, J.-S., & Seda, P. (2020). Design of an unsupervised machine learning-based movie recommender system. *Symmetry*, *12*(2).

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

Edelman, A., Arias, T., & Smith, S. (1998). The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, *20*(2), 303–353.

Elhamifar, E., & Vidal, R. (2013). Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(11), 2765–2781.

Fang, X., Han, N., Wu, J., Xu, Y., Yang, J., Wong, W. K., & Li, X. (2018). Approximate low-rank projection learning for feature extraction. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5228–5241.

Fiori, S. (2005). Quasi-geodesic neural learning algorithms over the orthogonal group: A tutorial. *J. Mach. Learn. Res.*, *6*, 743–781.

Fiori, S. (2008a). Leap-frog-type learning algorithms over the lie group of unitary matrices. *Neurocomputing*, *71*(10), 2224–2244.

Fiori, S. (2008b). A study on neural learning on manifold foliations: the case of the Lie group SU(3). *Neural Compututation*, *20*(4), 1091–1117.

Fiori, S. (2010). Learning by natural gradient on noncompact matrix-type pseudo-Riemannian manifolds. *IEEE Transactions on Neural Networks*, *21*(5), 841–852.

Fiori, S., Kaneko, T., & Tanaka, T. (2015). Tangent-bundle maps on the Grassmann manifold: Application to empirical arithmetic averaging. *IEEE Transactions on Signal Processing*, *63*(1), 155–168.

Fu, Y., Gao, J., Tien, D., Lin, Z., & Hong, X. (2016). Tensor LRR and sparse coding-based subspace clustering. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(10), 2120–2133.

Huang, L., Chao, H.-Y., & Wang, C.-D. (2019a). Multi-view intact space clustering. *Pattern Recognition*, *86*, 344–353.

Huang, L., Wang, C., Chao, H., & Yu, P. (2019b). MVStream: Multiview data stream clustering. *IEEE Transactions on Neural Networks and Learning Systems*.

Jain, A., Murty, M., & Flynn, P. (1999). Data clustering: A review. *ACM Comput. Surv.*, *31*(3), 264–323.

Kang, Z., Xu, H., Wang, B., Zhu, H., & Xu, Z. (2019). Clustering with similarity preserving. *Neurocomputing*, *365*, 211–218.

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. In *2015 International Conference on Learning Representations (ICLR)*.

Li, X., Zhang, Z., Wang, Y., Liu, G., Yan, S., & Wang, M. (2020). Multilayer collaborative low-rank coding network for robust deep subspace discovery. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*.

Liao, Z., Wang, N., Liu, S., Zhang, Y., Liu, H., & Zhang, Q. (2019). Identification-method research for open-source software ecosystems. *Symmetry*, *11*(2).

Liu, B., Huang, L., Wang, C., Fan, S., & Yu, P. (2019a). Adaptively weighted multiview proximity learning for clustering. *IEEE Transactions on Cybernetics*.

Liu, G., Zhang, Z., Liu, Q., & Xiong, H. (2019b). Robust subspace clustering with compressed data. *IEEE Transactions on Image Processing*, *28*(10), 5161–5170.

Lu, C., Yan, S., & Lin, Z. (2016). Convex sparse spectral clustering: Single-view to multi-view. *IEEE Transactions on Image Processing*, *25*(6), 2833–2843.

Makris, C., Pispirigos, G., & Rizos, I. O. (2020). A distributed bagging ensemble methodology for community prediction in social networks. *Information*, *11*(4).

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Dokl. Akad. Nauk SSSR*, *269*, 543–547.

Ng, A., Jordan, M., & Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In Dietterich, T., Becker, S., & Ghahramani, Z. (Eds.), *Advances in Neural Information Processing Systems 14*, pp. 849–856. MIT Press.

Özgenel, Ç.F. (2019). Concrete crack images for classification. `http://dx.doi.org/10.17632/5y9wdsg2zt.2`.

Papachristou, N., Miaskowski, C., Barnaghi, P., Maguire, R., Farajidavar, N., Cooper, B., & Hu, X. (2016). Comparing machine learning clustering with latent class analysis on cancer symptoms' data. In *2016 IEEE Healthcare Innovation Point-Of-Care Technologies Conference (HI-POCT)*, pp. 162–166.

Peng, X., Li, L., & Wang, F. (2019). Accelerating minibatch stochastic gradient descent using typicality sampling. *IEEE Transactions on Neural Networks and Learning Systems*.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12*(1), 145–151.

Qu, Z., Yuan, S., Chi, R., Chang, L., & Zhao, L. (2019). Genetic optimization method of pantograph and catenary comprehensive monitor status prediction model based on Adadelta deep neural network. *IEEE Access*, *7*, 23210–23221.

Rendl, F., & Sotirov, R. (2018). The min-cut and vertex separator problem. *Computational Optimization and Applications*, *69*, 159–187.

Rodriguez, M., Comin, C., Casanova, D., Bruno, O., Amancio, D., Costa, L., & Rodrigues, F. (2019). Clustering algorithms: A comparative approach. *PLoS ONE*, *14*(1).

Samet, H. (2005). *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 888–905.

Theodoridis, S. (2015). Chapter 8 - Parameter learning: A convex analytic path. In Theodoridis, S. (Ed.), *Machine Learning*, pp. 327–402. Academic Press, Oxford.

Vidal, R., & Favaro, P. (2014). Low rank subspace clustering (LRSC). *Pattern Recognition Letters*, *43*, 47–61.

Wang, Q., Gao, J., & Li, H. (2017). Grassmannian manifold optimization assisted sparse spectral clustering. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3145–3153.

Xiao, J., Hays, J., Ehinger, K. A., Oliva, A., & Torralba, A. (2010). Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3485–3492.

Zeiler, M. (2012). ADADELTA: An adaptive learning rate method. `https://arxiv.org/abs/1212.5701`.

Zhang, L., Yang, F., Daniel Zhang, Y., & Zhu, Y. J. (2016). Road crack detection using deep convolutional neural network. In *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3708–3712.

Zhang, Z., Ren, J., Li, S., Hong, R., Zha, Z., & Wang, M. (2019). Robust subspace discovery by block-diagonal adaptive locality-constrained representation. In *Proceedings of the 27th ACM International Conference on Multimedia (ACM MM)*, pp. 1569–1577.

Zhang, Z., Yan, S., & Zhao, M. (2014). Similarity preserving low-rank representation for enhanced data representation and effective subspace learning. *Neural Networks*, *53*, 81–94.

Zhong, H., Chen, Z., Qin, C., Huang, Z., Zheng, V., Xu, T., & Chen, E. (2020). Adam revisited: A weighted past gradients perspective. *Frontiers of Computer Science*, *14*.