POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master Degree Thesis

Generalized Principal Component Analysis Theory

Advisor :
prof. Elena Maria Baralis

Joint Advisor :
(UIC) prof. Erdem Koyuncu

Samuele Battaglino
matricola: 246414

December 2019

Summary

In the large field of Principal Component Analysis (PCA), a fairly old technique, there has been lately some advancement that left open a lot of room for improvements.

We propose a way to generalize completely the PCA standard and Kernel version (KPCA) from using only the norm in their objective function.
Instead a generic one can be used to better fit different datasets and applications and to ultimately outperform the usual norm based PCA. In the literature several attempts with Lp-norms have resulted in pretty good advancement thus using other functions seems to be the logical following.
We provide here the mathematical models behind our working algorithms and testing results with different functions on a really large variety of datasets. In most of the cases our method proved to be more robust to both outliers and noise than the state-of-the art L1-norm PCA and L1-norm KPCA and more malleable than any other adaptation yet.
Lastly, we proved the local optimality of the majority of our work even with a neat comparison between KPCA and a Recurrent Neural Network (RNN).

The research behind this work is shared between two thesis, this one and its American counterpart [1] and a scientific article [2].

Acknowledgements

From the very beginning, maybe even primary school, I always chose the most challenging option. I still have to find out why I do this to myself but with this I completed my hardest achievement yet and I was not alone in this journey.

Firstly, my family has always been there for me, starting with the not-so-obvious financial help, then with the much appreciated emotional support. When I say family I intend everyone: my parents not without disagreements but at the end always on my side, my brother great companion of life and laughs throughout every step of the way, my grandparents that not always understood everything I did but they were super enthusiastically all over that, my uncle, my aunt, my cousins and every other relatives that were still a part of the squad.

Moving on from there, surely Beatrice had a great part in this, she has been my touchstone in moments in which everyone would have quit and she is still here, by my side, and for that I could not be more thankful.

Also all my dearest friends, my roommates and classmate that during all this long years tried to make me smile also in harsh situations and everyone in their own ways helped me out.

Last but not least, I would like thank everyone who contributed from the academic standpoint. In this group surely someone stands out like my advisor prof. Erdem Koyuncu for having me introduced to the wonderful world of research and guided in my first real journey through it, as well as my Italian one prof. Elena Maria Baralis for her overseas contribution.

Contents

List of Tables

List of Figures

Computer Science is no more about computers than astronomy is about telescopes

[Edsger W. Dijkstra]

Chapter 1

Introduction

The algorithms that fall under the category of Principal Component Analysis are used to extract from a set of observations a set of variables called principal components or loading vectors.
The first component maximize the variance of the observations, then the second one does the same thing but under the constraint to be orthogonal to the previous ones and so on. The result will be a set of orthogonal basis ([3, 4, 5]).
This method was invented in 1901 by Karl Pearson [6] and refined later by Hotelling [7]; today is a well known technique used in a very large variety of fields, each one with even more applications. In Computer Science is used to reduce dimensionality [8], novelty detection [9], cluster data [10], outliers detection [11] etc.

The standard approach (L2-PCA) implies simply the use of the mere definition, so maximizing the variance of each observation $x_i \in \mathbb{R}^l$ projected onto our unitary vector w:

$$\begin{aligned} &\arg\max_{w} \quad \sum_{i=1}^{N} ||w^T x_i||_2^2 = ||w^T X||_2^2 \\ &\text{subject to} \quad ||w||_2 = 1 \end{aligned} \tag{1.1}$$

Where X is the matrix that collects every N train sample x_i.

For this to work X should have at least zero mean, if it is not the case, every sample can be brought back to it by a simple subtraction:

$$x_i \leftarrow x_i - \mu \qquad \text{where } \mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1.2}$$

The mean vector μ will have the same dimension as the sample. So from here on out every equation will be given as if the samples had zero mean (if not specified differently).

Back to the 1.1, it can be neatly and optimally solved through a eigendecomposition, where the eigenvector with the greater eigenvalue is our PC (Principal

Component), then the other ones can be extracted by doing the same thing but on the X obtained with a greedy subtraction of the previously extracted component (Algorithm 2) or in other words under the constraint to be orthogonal in respect to the previously extracted Principal Components (PCs).

1.1 Standard (Lp) PCA

The theory behind L2-PCA seems great but it is not the best approach in every situation, indeed it has been observed that is very susceptible to outliers or highly noisy datasets.
The problem is that the maximization (1.1) can be seen also as a minimization of the mean-squared reconstruction error, so outliers will have a larger impact than the other observations thus affecting greatly the loading vector.
Over the years many techniques have been developed but the most widely accepted is the L1-PCA that relies on the absolute values (L1-norm) instead:

$$\begin{aligned} \arg\max_{w} \quad & ||w^T X||_1 = |w^T X| \\ \text{subject to} \quad & ||w||_2 = 1 \end{aligned} \tag{1.3}$$

This lowers effectively the impact of the outliers (actually of every sample) but loses the elegant solution of the eigendecomposition resulting also in to an higher computational cost becoming indeed a NP-hard problem [12].

In the paper [13] the author pushed the boundaries a little further by generalizing the concept to a Lp-norm. Theoretically, everything should work, but again there is not a closed form solution so, as he suggests, a gradient descent can be used and still get good results.

$$\begin{aligned} \arg\max_{w} \quad & ||w^T X||_p^p \\ \text{subject to} \quad & ||w||_2 = 1 \end{aligned} \tag{1.4}$$

The gradient was then furtherly improved by using the Lagrangian trick leading to a fast converging algorithm (Algorithm 1 is an adaptation of it).
Not only that, but also other norms, especially L0.5 and L1.5 got good results in both noisy/outliers-filled cases and more vanilla, comparable robustness results were only showed in the PCA theory by rotational invariant methods like [14].
These findings paved the way to a more complete generalization, moving away the PCA from the norm itself to more different functions.

Lastly, we know that for finding more than one PC the greedy approach is only optimal while using L2-norm but we just want to have a proof of concept that using

other functions instead of norms yielded better results anyway.
So for further reading, in both papers [13] and [15] different heuristics are proposed and those could be used to replace the greedy method also in our scenario.

1.2 Kernel PCA

In the same field as before, the Kernel Principal Component Analysis is an extension of the classic PCA using the well know Kernel technique. They were merged for the first time by Schölkopf in 1997 [16] and since then it has been used and expanded a lot.
Once again, this method is used more or less with the same aim as the standard version but being also a non-linear analysis, it has some few more uses like face recognition [17], image modeling [18], outliers detection [11] etc.
Let's take a step back: with this whole theory the aim is to find an orthogonal base that maximize the variance of the initial points if projected on it. This procedure works well enough but usually it is easier and more effective to find the PC in an higher dimensional space, by mapping the data samples with an arbitrary function like this:

$$x \in \mathbb{R}^l \to \Phi(x) \in \mathbb{R}^d \qquad \text{where } d >> l \tag{1.5}$$

For example thinking about clustering application, it is easier to find hyperplanes that separate points purposely mapped instead of working in their original dimensions. Another example is with support vector machines that extract only linear correlations but with a good mapping they can also extract non-linear ones.

The higher dimensional space to which $\Phi(\cdot)$ leads is called feature space and usually d is so big that already makes computing the new mapped samples unbearably intense. So in the Kernel theory it has been developed a way of avoiding altogether that task by instead constructing the very Kernel matrix:

$$K_{i,j} = K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j) \quad \forall i, j \in [1, N] \tag{1.6}$$

Where clearly it can also be interpreted as a two variable function and over the years a lot of them have been tested and perfected (i.e.: polynomial, Gaussian, Radial Basis Functions...).

At last, the KPCA aims to do find the PC in the feature (or Kernel) space and being already K a kind of covariance matrix, it is sufficient to proceed in a similar way as L2-PCA: by doing its eigendecomposition.
This method is referred indeed as L2-KPCA because it solves:

$$\begin{aligned} \arg\max_{w} \quad & \sum_{i=1}^{N} ||w^T \Phi(x_i)||_2^2 = ||w^T \Phi(X)||_2^2 \\ \text{subject to} \quad & ||w||_2 = 1 \end{aligned} \tag{1.7}$$

As in the normal PCA, also here the samples must have zero mean; in the feature space it means that $\Phi(X)$ should be centered. To do that it is possible to subtract the mean in the feature space and again working only on the Kernel Matrix:

$$\begin{gathered}\Phi(x_i) \leftarrow \Phi(x_i) - \mu \qquad \text{where } \mu = \frac{1}{N}\sum_{i=1}^{N}\Phi(x_i) \\ K_{centered} = K - 1_N K - K 1_N + 1_N K 1_N\end{gathered} \tag{1.8}$$

Where 1_N is a matrix of the same dimensions of K and composed entirely by the value $1/N$. For now on, it is assumed that K is indeed always centered.

The full derivation is present on the original Schölkopf work, for this thesis purpose it is sufficient to know that the resulting PCs in the feature space are composed by the eigenvectors $\alpha_j \in A$ multiplied by the samples in that space:

$$w_j = \sum_{i=1}^{N}\alpha_j \Phi(x_i) \quad \forall j \in [1, p] \tag{1.9}$$

Where p is the number of PCs extracted and the eigenvectors are ordered from the one with the greatest eigenvalue to the smallest (or stopping at the p^{th}).

The problem of having to compute, in the expression of the loading vector, explicitly $\Phi(\cdot)$ can be simply avoided because the PCs are almost always used to compute the projection on them of some other data y (obviously mapped as well) so only the α_j are kept and used when needed:

$$w_j^T \Phi(y) = \sum_{i=1}^{N} K(x_i, y)\alpha_j^T \quad \forall j \in [1, p] \tag{1.10}$$

The KPCA does not suffers from the same issues of searching for more PCs but scales badly with the dimension of the problem. This issue is originated by the construction of the Kernel Matrix, this is explained and tackled using an incremental approach successfully in [19].
We simply decided to avoid huge datasets just to apply only the vanilla approach.

1.2.1 L1-KPCA

The L2-KPCA is a fine procedure and with all that flexibility given by the Kernel function it seemed less prone to the same problems that affected the L2-PCA but still it would be interesting to find out what will happen if we pushed the boundaries as we did before.
The authors in [20] have done a great job in figuring out a way of resolving the L1-norm corresponding Kernel problem:

$$\begin{aligned}&\arg\max_{w} && |w^T \Phi(X)| \\ &\text{subject to} && ||w||_2 = 1\end{aligned} \tag{1.11}$$

Still the loading vectors will conserve the previous form (1.10), the α's will no longer be the eigenvectors of K but they are still present. Actually they will be computed again with a kind of Gradient Ascent and Lagrangian trick that can be easily understood by drawing a comparison with the RNN theory [21].

Their findings and algorithm opened the possibility for a more complete generalization both in terms of the norm and again away from that completely.
This is also the reason why their algorithm is not explicitly reported here; later on a more general one will be proposed altogether and its L1-norm form will be equal to theirs.

Chapter 2

Generalized PCA

At the very beginning, the main aim was finding different functions to outperform standard Lp-norm PCA and solve still the PCA:

$$\begin{aligned} \arg\max_{w} \quad & f(w^T X) \\ \text{subject to} \quad & ||w||_2 = 1 \end{aligned} \tag{2.1}$$

The focus was not on the functions themselves but onto their first derivatives, because finding the principal components not through an L2-norm requires some sort of iterative approach. In our case it was a kind Gradient Ascent based off the one in [13] and the algorithm is summarized below.
Where X is our train set and it is composed by N vectors (x_i) of dimension M, so our principal component (w) will be a vector of dimension M and it is indeed initialized as the normalized train sample with the greatest norm. We will loop until w converges under the arbitrary defined precision *eps* (usually kept at a safe 10^{-10}).

The function $f(x)$ can be whatever is wanted and for the standard PCA it is placed as the desired Lp-norm, so its derivative would be $f'(|w^T x_i|) = \frac{1}{p}|w^T x_i|^{p-1}$. Naturally, GPCA stands for Generalized Principal Component Analysis.

Algorithm 1: GPCA

Input: the current train set matrix X
Result: the extracted principal component w_*

initialize $w(0) = maxnorm(X)/||maxnorm(X)||_2$, $t = 0$

repeat
  $\nabla_w = \sum_{i=1}^{N} f'(|w(t)^T x_i|) sign(w(t)^T x_i) x_i$
  $w(t+1) = \nabla_w / ||\nabla_w||_2$
  t = t+1
until $||w(t) - w(t-1)|| > eps$;

$w_* = w(t)$

The full derivation is, above all, present on the original paper and from the same authors [22], but since it has been adapted and used a lot, in the Appendix A is present a summarized derivation.

The paper than proposed also an "Heuristic" method for extracting more features, but for ease of programming and because they do not differ in performance that much the decision was made to follow the usual greedy approach, listed again here below.
Where I_N is an identity matrix $N \times N$, p is the number of features to be extracted from X so W will be $M \times p$.

Algorithm 2: Greedy GPCA

Input: the train set matrix X
Result: the extracted principal components matrix W

for $i \leftarrow 1$ **to** p **do**
  Do **Algorithm 1** on X and get as result w_*
  $W_i = w_*$
  $X = (I_N - w_* w_*^T) X$
end

2.1 Optimality Check

We want to check the local optimality of our PCA Algorithm 1 and from its objective function 2.1 we can rephrase a more straight forward update rule without all those ∇:

$$w \leftarrow \frac{\sum_{i=1}^{N} f'(w^T x_i) x_i}{|| \sum_{i=1}^{N} f'(w^T x_i) x_i ||_2} \tag{2.2}$$

So the Theorem 1 and the finding of the paper [13] prove that if $f(\cdot)$ is non-decreasing (or if $f''(\cdot) \geq 0$) and if $\tilde{f}(w) = \sum_{i=1}^{N} f(w^T x_i)$ is convex then the sequence will converge to a local maximum.

In our GPCA, the first requirement on $f(\cdot)$ can simply be achieved by restricting the range of function you can actually choose.
Indeed only choosing non-decreasing ones will already satisfy the first constraint, for the second one a little more work is needed.

We have to check the convexity of that sum of function, so in other word we have to prove that its Hessian is positive semi-definite:

$$H(\tilde{f}(w)) = \sum_{i=1}^{N} f''(w^T x_i) x_i x_i^T \tag{2.3}$$

If you take a closer look at that you can see that H is actually always weakly positive is $f''(\cdot) \geq 0$ because the latter term is actually a squared term.
This newly found constraint is actually the same as the previous one so we do not need to restrict anymore the choice of the generic function and do not change anything else.

The theorem referred to above is reported and proven here for completeness, even if everything can be found in the original paper:

Theorem 1 ([13, Theorem 1]) *Let $F(w)$ be a convex function. Let $\|w'\| = 1$ and $w'' = \frac{\nabla F}{\|\nabla F\|}|_{w=w'}$. Then, $F(w') \geq F(w)$. In particular, the gradient ascent $w \leftarrow \frac{\nabla F}{\|\nabla F\|}$ provides a locally-maximum solution to the problem $\max_{w:\|w\|=1} F(w)$.*

We arrived at the theorem update rule by studying the optimal form given by the Lagrangian multiplier technique on the only constraint of the initial maximization problem in the Appendix A.
The found form of the Equation A.3 is true for both minimum and maximum points, but applying it in gradient ascent fashion, it is guaranteed that near a minimum the gradient will diverge and instead converge on a maximum.
In a more formal way we can say that proven the things we proved beforehand on the convexity we can easily prove:

$$F(w') - F(w) \geq \left(\frac{\nabla F}{\|\nabla F\|} \right)^T (w' - w) \geq 0 \tag{2.4}$$

Specifically, the second part hols because w' is parallel to $\frac{\nabla F}{\|\nabla F\|}$ and both w' and w have unit norm for definition.

For as far as the GPCA is concerned the method is proven to reach a local optimum under the previously mentioned constrain.

2.2 Results

2.2.1 Generic Functions

These algorithm allowed us to swap the usual norm function and instead place whatever $f(\cdot)$ is desired, naturally for good results they need to behave like norms, working with first derivatives the issue shift to those.
Eventually we could fit the function to the specific application and dataset but for now, just as a proof of concept we stayed close to Lp-norms.

Here is a visual representation of what they look like:

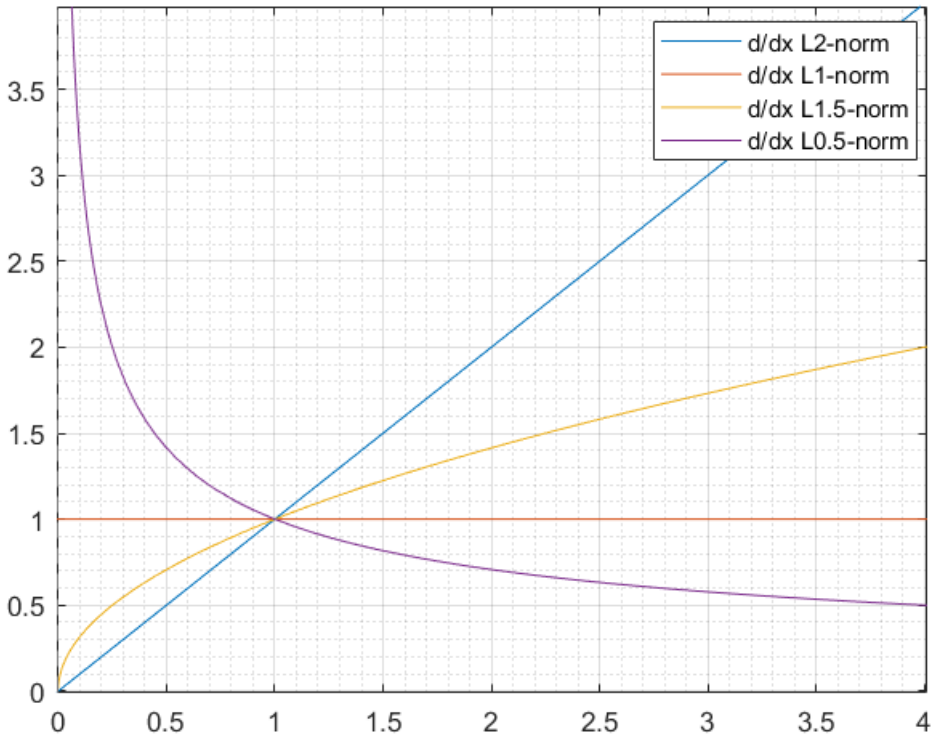


Figure 2.1: First derivatives of some Lp-norms

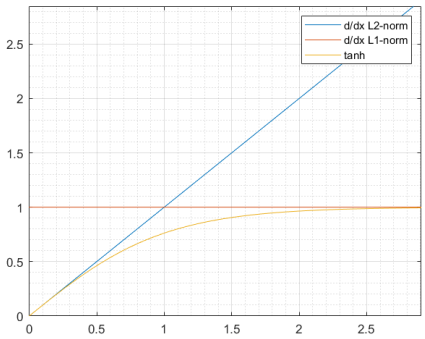
A fair share of functions were tried and below the more promising ones are displayed together to the first derivative of L1 and L2 norms just for a frame of reference.
Everyone of them has some thought behind it:

- "$tanh(x)$", "$3 - 3sech(x)$" and "$2tanh^2(x)$" are aimed to act as L2 when the data have small/normal projection magnitude and then saturated to a certain value with greater norms, much like L1, indeed fusing those two approaches.
  In a noisy environment the data that is not affected has a rightfully impact on the PC while others contribution is dampened.
  The only significant change among those three functions is the value at which they saturate.

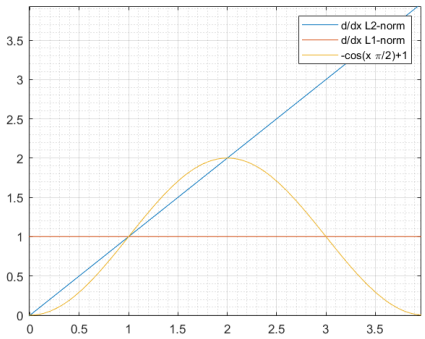
- "$4sigmoid(x) - 2$" acts like L2 on normal projection and then starts slowly to saturate, more like a L1.5 instead of a L1 as the first group.
  This is maybe more beneficial in dataset in which the noise is not excessive by never completely cutting off their contribution.

- "$2tanh(x)sech(x)$" works like the previous functions but instead of saturating it actually decreases.
  This approach could be more useful is the outliers are really in the majority, but it could also generate the opposite result by burdening to much the possible outliers.

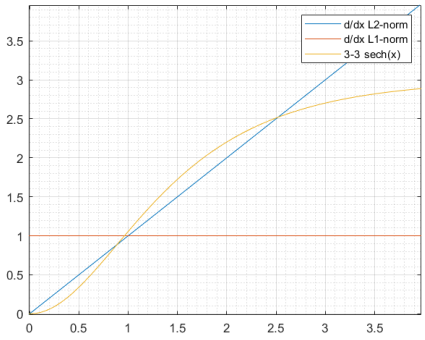
- "$-cos(x\pi/2)$" was a little bit of a gamble: it is a bit like "$2tanh(x)sech(x)$" but being a cosine it will increase and decrease again over and over.
  Indeed it apparently does not even satisfy the optimality constraint, but if the norm is bounded to a single period of the function then it behaves like the previous one with even a steeper slope.



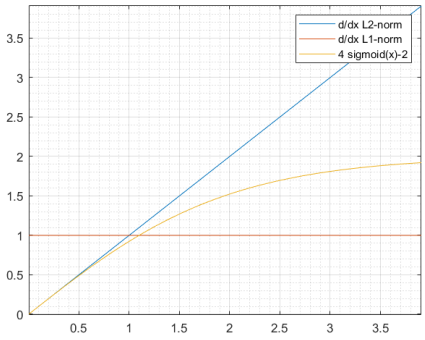
(a) Plot of tanh(x)



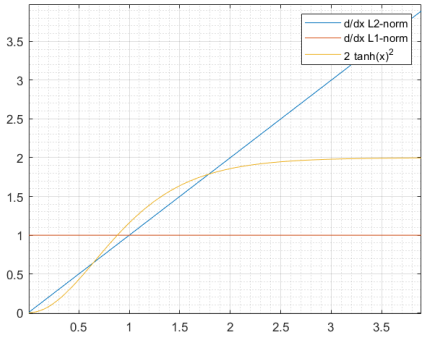
(b) Plot of -cos(xπ/2)



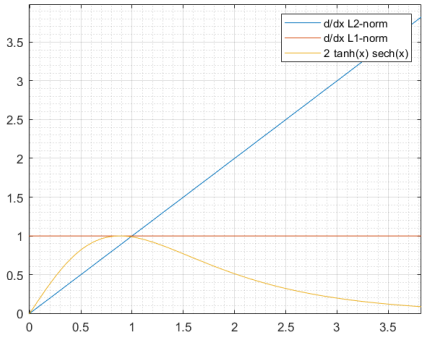
(c) Plot of 3-3sech(x)



(d) Plot of 4sigmoid(x)-2



(e) Plot of 2tanh^2(x)



(f) Plot of 2tanh(x)sech(x)

Figure 2.1: Tried Generic functions

Despite all of that, only the first group of three functions actually yielded interesting results the other ones failed to compare sometimes even with the standard L2-norm. So from here on those will be studied more in depth.

For short we named those function for example $tanh$ but actually, true to the algorithm 1, they will be applied like this: $\nabla_w = \sum_{i=1}^{N} tanh(|w(t)^T x_i|)sign(w(t)^T x_i)x_i$.

Finally, below we present the results for our test cases that for consistency sake and good comparison were against the same datasets and with the same procedures each time.
How exactly those tests were carried on is explained only once and for all in the Appendix B.

Furthermore, every table row will have in bold the cell(s) that have the best result, higher or lower depending on what has been computed.

USPS: This experiment helped us show how the selected functions start low but as the noise grows bigger they catch up, especially the $3-3sech$ and $tanh$ get on average better result than any other method.
This of course is the wanted result because by using functions similar to both L1 and L2 should result in the best of both world, resilient to noise and outliers but with a greater overall quality of loading vectors.

Table 2.1: GPCA: USPS WITH GAUSSIAN NOISE

Noise level (σ)	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)	$tanh$	$3 - 3sech$	$2tanh^2$
0	95.08	**95.52**	95.51	95.46	95.36	95.35	95.33
10	95.02	95.42	**95.45**	**95.45**	95.32	95.34	95.37
20	94.89	95.31	**95.34**	95.31	95.21	95.31	95.30
30	94.71	95.14	95.17	95.17	95.05	**95.19**	95.16
40	94.47	94.86	94.93	94.94	94.84	**95.00**	94.94
50	94.11	94.52	**94.60**	**94.60**	94.58	94.58	**94.60**
60	93.64	94.01	94.13	94.13	94.15	**94.29**	94.15
70	92.99	93.42	93.46	93.39	93.56	**93.68**	93.51
80	92.00	92.49	92.53	92.31	92.66	**92.82**	92.60
90	90.67	91.20	91.15	90.82	91.37	**91.56**	91.26
100	88.76	89.47	89.26	88.75	89.58	**89.83**	89.48
Average	93.30	93.76	93.78	93.67	93.79	**93.91**	93.78

Table 2.2: GPCA: USPS WITH S&P NOISE

Noise level (n)	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)	$tanh$	$3 - 3sech$	$2tanh^2$
0	95.08	**95.52**	95.51	95.46	95.36	95.35	95.33
0.05	94.36	94.79	94.92	94.90	94.78	**94.95**	94.94
0.1	93.36	93.76	93.84	93.87	93.81	93.95	**93.96**
0.15	91.78	92.17	92.23	92.24	92.32	92.30	**92.33**
0.2	89.45	89.78	89.75	89.77	**90.06**	89.83	90.05
0.25	86.08	86.37	86.25	86.19	**86.79**	86.37	86.60
0.3	81.74	82.17	81.98	81.24	82.25	**82.69**	82.09
0.35	76.08	76.43	75.89	74.95	76.44	**76.95**	76.34
0.4	69.28	69.60	68.76	67.54	69.51	**69.95**	69.36
0.45	61.71	61.98	61.00	59.52	61.84	**62.33**	61.65
0.5	53.96	53.94	52.91	51.22	53.83	**54.22**	53.74
Average	81.17	81.50	81.18	80.63	81.54	**81.72**	81.49

Yale: This is kind of a more hard scenario, in fact the new functions struggle a little more. The dataset is still filled with an increasing number of outliers but this time it is not simply classification but reconstruction.
At the end the wanted pattern from before is still confirmed: with the functions getting an overall better performance than the Lp-norms with emphasis on the more noisy scenarios.

Table 2.3: GPCA: YALE FACES

#noisy images	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)	$tanh$	$3 - 3sech$	$2tanh^2$
15 (9%)	2.10	2.07	2.13	2.24	2.06	2.05	**2.04**
30 (18%)	2.20	**2.18**	2.31	2.64	**2.18**	**2.18**	**2.18**
45 (27%)	2.28	2.29	2.43	2.79	2.28	2.28	**2.27**
60 (36%)	**2.35**	2.37	2.48	2.87	2.37	2.36	2.36
75 (45%)	**2.43**	2.44	2.53	2.90	2.44	2.44	**2.43**
90 (54%)	2.51	2.52	2.57	2.92	**2.50**	**2.50**	2.51
103 (63%)	2.58	2.58	2.60	2.94	**2.56**	**2.56**	2.57
Average	2.35	2.35	2.44	2.76	**2.34**	**2.34**	**2.34**

MNIST: This test suite completes a bit the kind of noises we could imagine: additive noise with the Gaussian, transformative with S&P, pure outliers injection with Yale Faces and now a multiplicative one.
The pattern is confirmed even if not with the same clear cut as beforehand. Still with greater variance than 2 our functions beat the competition, especially the $3 - 3sech$.

Table 2.4: GPCA: MNIST WITH SPECKLE NOISE

variance	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)	$tanh$	$3 - 3sech$	$2tanh^2$
0	4.22	**4.17**	4.28	4.28	4.21	**4.17**	4.25
0.04 (default)	4.63	4.27	4.29	4.45	4.33	**4.26**	4.36
0.2	4.98	**4.63**	4.70	4.69	4.74	4.69	4.72
0.4	5.67	**5.38**	5.49	5.50	5.47	5.44	5.45
0.5	6.04	**5.72**	5.84	5.85	5.80	5.77	5.79
0.8	6.79	**6.49**	6.58	6.68	5.58	6.52	6.57
1	7.12	**6.86**	7.02	8.65	6.94	6.90	6.93
2	8.32	**8.07**	8.19	8.33	8.13	**8.07**	**8.07**
4	9.45	9.22	9.39	9.52	9.25	**9.17**	9.21
5	9.82	9.56	9.72	9.86	9.53	**9.52**	**9.52**
8	10.41	10.25	10.43	10.59	10.25	**10.23**	10.24
10	10.79	10.56	10.73	10.91	10.55	**10.51**	10.54
Average	7.35	**7.10**	7.22	7.44	7.15	**7.10**	7.14

UCI: Contrary to what has been predicted earlier, the functions should get the best of both worlds but instead by trying them on a large variety of heterogeneous datasets as the UCI ones, without any noise, they do not shine as intended.
They also appear to be consistently worse and it is confirmed by both the average and the 2.6 that confront directly how many times they beat the standard

Lp-norms. Even if *tanh* loses only by one datasets against L1.
Using different function maybe more suited to the noiseless cases we are confident it would have beaten Lp-norms also here, but for now we did not pushed forward.

Table 2.5: GPCA: UCI CLASSIFICATION RATES

Datasets	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)	$tanh$	$3 - 3sech$	$2tanh^2$
Iris	92.03	95.32	95.34	95.35	95.34	95.99	**96.00**
Balance	74.49	74.40	74.45	66.90	74.50	**74.51**	74.43
Ionosphere	**83.59**	83.23	82.85	82.96	82.56	82.47	82.28
Yeast	**44.43**	41.52	41.17	39.91	39.98	38.19	38.08
Phishing	**88.71**	88.46	87.92	85.82	87.67	85.87	86.99
Liver	62.10	63.43	62.23	61.39	63.51	**63.70**	63.67
Waveform	83.50	**84.04**	83.89	83.90	83.95	83.87	83.91
Letter	59.74	61.40	61.75	**61.80**	61.48	61.65	61.54
Car	82.50	82.98	**82.70**	82.11	82.92	82.10	82.40
Average	74.57	**74.98**	74.70	73.35	74.67	74.26	74.37

Table 2.6: GPCA: UCI WINNING COUNT

	$tanh$	$3 - 3sech$	$2tanh^2$
$p = 0.5$	3	1	-1
L1 ($p = 1$)	-1	-1	-1
$p = 1.5$	0	-3	-3
L2 ($p = 2$)	3	-1	3

Finally just to wrap up what we discovered: our function works best with substantial noise and outliers level even outperforming the state-of-the-art for that application that is the L1-norm.
Same goes for the average of results in those cases, so we can confidently say that they could be use in all those fields that kind of robustness is needed: de-noising, reconstruction and dimensionality reduction in noisy cases, outlier detection etc...
We cannot instead say the same thing in the "vanilla" or noiseless cases, our functions do not win consistently over the standard Lp-norm approach.

2.2.2 Mixing L1-norm and L2-norm

Once completed the experiments with the generic functions, something popped out: the three working functions have all a very similar structure, summarizable with

this function that has been named "Skeleton":

$$g'(x,a) = \begin{cases} x & \text{if } |x| \leq a \\ a \cdot sign(x) & \text{otherwise} \end{cases} \tag{2.5}$$

Why shifting to such a function if the generic ones from before worked well?
First of all because it is more justifiable theoretically, the generic function were chosen to be kind like the norms but other than that they were cherry picked to suit our needs, instead this one can really be seen as a mix of L2-norm for all the observations that have a projection on the current PC with a value lower than a and L1-norm if higher. So it should get the best of both worlds and do not need any more justifications other than it is a mix of two of the most working theories in the PCA field. Again keeping in mind that last time only in noisy case we obtained some significant results.
Moreover if a tends to zero the function will behave as L1-PCA (actually a little bit different in a sense that has not a saturation value of 1 but something close to zero) and on the other side if it tends to infinity it will approximate L2-PCA, like it can be imagined by looking at its plot down here:

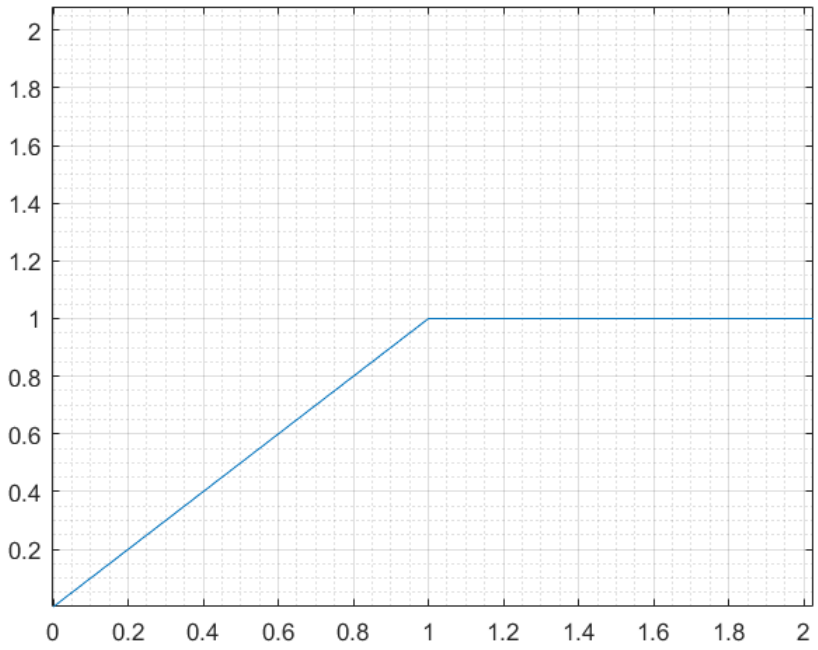


Figure 2.2: Skeleton with $a = 1$

The negative part is not printed because it is always used with a positive argument (see Algorithm 1), so actually could be simplified more by removing the $sign(x)$ in the second part and the absolute value in the case distinction.

"Skeleton"

The Skeleton function (2.5) is really the distillation of what has been discovered with the generic functions; if you place $a = 1$ the Skeleton behaves like a discrete "$tanh$", with 2 like "$2tanh^2$" and with 3 like "$3 - 3sech$".

For the first part just to test out if the findings of the Generic Function withheld the discretization and if this was the right path to follow, the decision was made to try to reproduce the results on those datasets using indeed $a \in \{1,2,3\}$.

For comparison sake, in each table is also reported the best function (in average) from the table in the previous subsection.
Instead, SPCA is short the GPCA that uses the Skeleton function.

USPS: By comparing the results with the one obtained before, we can clearly see that $a = 1$ is better than any other approach with Gaussian noise and only struggles against the *sech* that is the best previous function.
Anyhow also the other values of a get good results that stacks well against the standard methodologies.

Table 2.7: SPCA: USPS WITH GAUSSIAN NOISE

Noise level (σ)	$3 - 3sech$	$a = 1$	$a = 2$	$a = 3$
0	95.35	**95.65**	95.48	95.51
10	95.34	**95.55**	95.45	95.37
20	95.31	**95.41**	95.31	95.34
30	95.19	**95.25**	95.17	95.19
40	95.00	**95.02**	94.95	94.97
50	**94.58**	94.72	94.65	94.65
60	**94.29**	94.27	94.20	94.19
70	**93.68**	93.62	93.56	93.53
80	**92.82**	92.69	92.63	92.58
90	**91.56**	91.40	91.32	91.21
100	**89.83**	89.58	89.53	89.34
Average	93.91	**93.92**	93.83	93.82

Table 2.8: SPCA: USPS WITH S&P NOISE

Noise level (n)	$3-3sech$	$a=1$	$a=2$	$a=3$
0	95.35	**95.65**	95.48	95.51
0.05	**94.95**	94.91	94.86	94.90
0.1	**93.95**	93.91	93.87	93.87
0.15	**92.30**	92.38	92.31	92.30
0.2	89.83	**90.08**	90.00	89.93
0.25	86.37	**86.82**	86.71	86.58
0.3	**82.69**	82.25	82.20	82.10
0.35	**76.95**	76.45	76.39	76.27
0.4	**69.95**	69.51	69.38	69.10
0.45	**62.33**	61.73	61.65	61.36
0.5	**54.22**	53.70	53.64	53.20
Average	**81.72**	81.58	81.50	81.37

Yale: Once again with outliers the Skeleton stands out as the generic functions did, furthermore there is no clear winner: they all get really similar results and they get the best average as the generic functions did.
For comparison the $2tanh^2$ has been selected.

Table 2.9: SPCA: YALE FACES

#noisy images	$2tanh^2$	$a=1$	$a=2$	$a=3$
15 (9%)	**2.04**	**2.04**	2.05	**2.04**
30 (18%)	**2.18**	**2.18**	**2.18**	**2.18**
45 (27%)	**2.27**	2.28	2.28	**2.27**
60 (36%)	**2.36**	**2.36**	2.37	2.37
75 (45%)	**2.43**	2.44	2.44	2.44
90 (54%)	2.51	2.51	2.51	**2.50**
105 (63%)	2.57	**2.56**	**2.56**	2.57
Average	**2.34**	**2.34**	**2.34**	**2.34**

MNIST: Confirming what has been said in the previous two experiments: our skeleton achieves overall good results more clearly when the variance is high and the a is equal to 1.

Even against the best previous generic function $3-3sech$ that performed best as well in high noise scenarios.
This proves at last that it is a good path because we improved from the generic functions in what concerns the main focus of these methodologies.

Table 2.10: SPCA: MNIST WITH SPECKLE NOISE

variance	$3-3sech$	$a=1$	$a=2$	$a=3$
0	**4.17**	**4.17**	4.32	4.20
0.04 (default)	**4.26**	4.27	4.34	4.30
0.2	4.69	**4.67**	4.73	4.70
0.4	5.44	**5.41**	5.47	5.45
0.5	5.77	**5.75**	5.81	5.78
0.8	6.52	**6.50**	6.57	6.53
1	6.90	**6.87**	6.95	6.90
2	8.07	**8.04**	8.11	8.07
4	9.17	**9.14**	9.24	9.21
5	**9.52**	9.53	9.56	9.53
8	10.23	**10.19**	10.25	10.23
10	10.51	**10.48**	10.53	10.52
Average	7.10	**7.09**	7.16	7.12

UCI: This time the Skeleton seems to be slightly worse than the generic functions in terms of absolute performance but better in beating in more datasets the Lp-norms.
Skeleton with parameter 1 is finally able to outperform all the Lp-norms in the winning count.

The comparison this time is drawn with the L1-norm PCA that was the previous best in this repository.

Table 2.11: SPCA: UCI CLASSIFICATION RATES

Datasets	L1 ($p=1$)	$a=1$	$a=2$	$a=3$
Iris	95.32	**95.34**	**95.34**	**95.34**
Balance	74.40	74.49	74.48	**74.51**
Ionosphere	**83.23**	82.55	82.38	82.80
Yeast	**41.52**	39.84	39.69	39.80
Phishing	**88.46**	85.86	85.87	85.89
Liver	63.43	63.57	63.61	**63.64**
Waveform	84.04	**84.06**	83.91	83.83
Letter	61.40	61.44	61.55	**61.66**
Car	82.98	83.86	85.87	**85.89**
Average	**74.98**	74.48	74.34	74.41

Table 2.12: SPCA: UCI WINNING COUNT

	$a = 1$	$a = 2$	$a = 3$
$p = 0.5$	3	-1	1
L1 ($p = 1$)	3	-1	-1
$p = 1.5$	1	-3	-3
L2 ($p = 2$)	1	1	-1

Optimizing a

Now the only thing that remains to do is trying to optimize also a in the Skeleton function.
It seems easy at first but with a little more depth the problem is not trivial, let's start by finding a primitive of our skeleton function, that will look like something like this:

$$g(x, a) = \begin{cases} \frac{1}{2}x^2 + c_1 & \text{if } |x| \leq a \\ a|x| + c_2 & \text{otherwise} \end{cases} \tag{2.6}$$

Continuity should not be an issue in general but in order to stay on the safe side, the newly obtained function can be made as such by placing $c_1 = 0$ and $c_2 = -\frac{1}{2}a^2$. This will produce the following function, printed with $a = 2$ as an example and against L2 and L1 for reference:

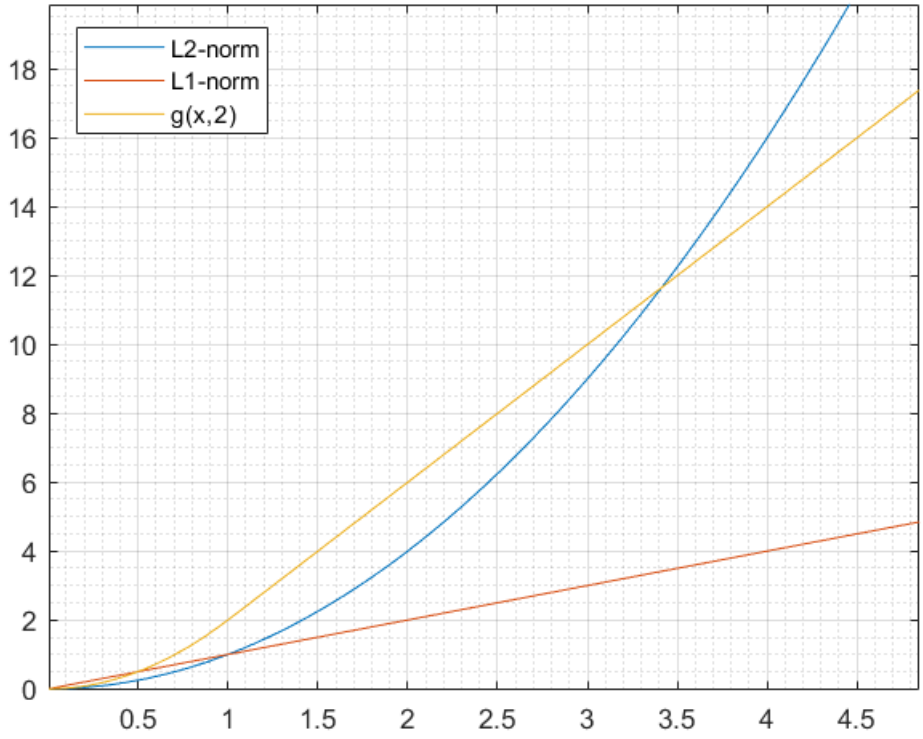


Figure 2.3: Plot of $f(x,2)$ with L2 and L1 norm

That being said the next step is to try to solve this optimization problem:

$$\begin{aligned} \arg\max_{w,a} \quad & g(|w^T X|, a) \\ \text{subject to} \quad & ||w||_2 = 1, a \geq 0 \end{aligned} \tag{2.7}$$

From this it seems again just a matter of computing its partial derivatives and do gradient ascent as before or maybe even try a Lagrangian trick, but the maximization formulated in that way has an intrinsic fault.
It can be seen more clearly once the Gradient Ascent approach is written out, so here is the partial derivatives:

$$\nabla g(|w^t X|, a) = \begin{bmatrix} \frac{\partial f}{\partial w} \\ \frac{\partial f}{\partial a} \end{bmatrix} = \begin{bmatrix} w_t X_1^2 + a \cdot sign(w^t X_2) X_2 \\ |w^t X_2| - a \end{bmatrix} \tag{2.8}$$

To make it easier to compute the function has been linearized by defining those new matrices in such a way that both have the same dimensions as the original X but X_1 has only the observations whose projections on the current PC are less than a and every other element to zero, dually X_2 has only the ones greater than a.

On the surface it seems to work and converge, the only thing is that a always goes to a value that exceeds the maximum projection value making $X_1 = X$ and $X_2 = \emptyset$, or in other words it makes the method equal to a L2-PCA.
This result should not be surprising, in fact the primitive of Skeleton is just a mix of the projection matrix and its square, most likely the projections will have a norm greater than 1 so it makes sense that by maximizing the function it will tend to the "square part" instead of the other one. Furthermore while optimizing, it is difficult to take into account the fact that the value of a makes the ranges of the piece-wise function itself change. So not only it impacts the objective function by being multiplied to the second piece but also make some observation shift between the two sets X_1 and X_2.
Lastly its update rule is balanced only by a term that was arbitrary inserted to make the primitive continuous, without that a would always increase until X_2 is empty.

The only thing left to do would be to estimate a externally or in a second time and still work with the previous maximization (2.1) but again is not an easy task: it is like asking where is the threshold from which the projections will be better off truncated, because they are mostly a product of outliers.
That question alone is worth a lot, so for now it has not been identified a decent way of doing that and by decent it means beating at least the $a = 1$ results that were clearly the best from the previous analysis.

So as far as this goes a good approach would be trying different
$a \in [0, ||maxnorm(X)||_2]$ on the train set, pick the best one and then test it.
The parameter a should still be a real number but it seems that training only integer values in a logarithmic fashion (as shown in the images below) could be a viable solution. Furthermore there is no need to test all the way to the maximum norm because the values saturates much earlier on the result of the L2-norm.
The last tweak that can be used to restrict the field is to divide the cases from being outliers-prone or more "normal", because according to that it is advised to work more with values that are < 10 for the first case and larger in the second one. It is not possible to actually select a range for this part because it depends on the $||maxnorm(X)||_2$, in our case it usually is from 10 to 500.
For example down here are reported the averages of different a on the USPS dataset with S&P noise and as expected the best results are on the left side of the chart.
The "real" best is confirmed to be the one we obtained beforehand: $a = 1$. Those spikes suggests that under 50 the chosen granularity maybe a little to big so it is possible to dig deeper around the more interesting values (i.e.: from 1 to 40).

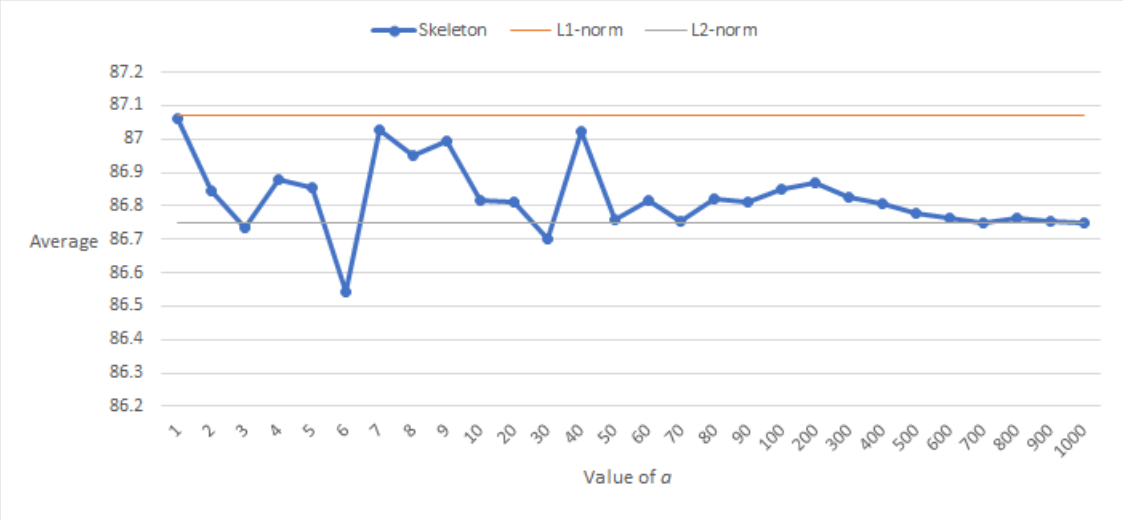


Figure 2.4: Different results with different a for USPS S&P

This makes sense because being the "best" part on the left side even little variation can make a sensible difference, but for what was intended to be proven here (that this approach was doable) the search has not been pushed on.

Dually in a dataset like the UCI Letter the best results are on the right side, more close to the L2-norm.

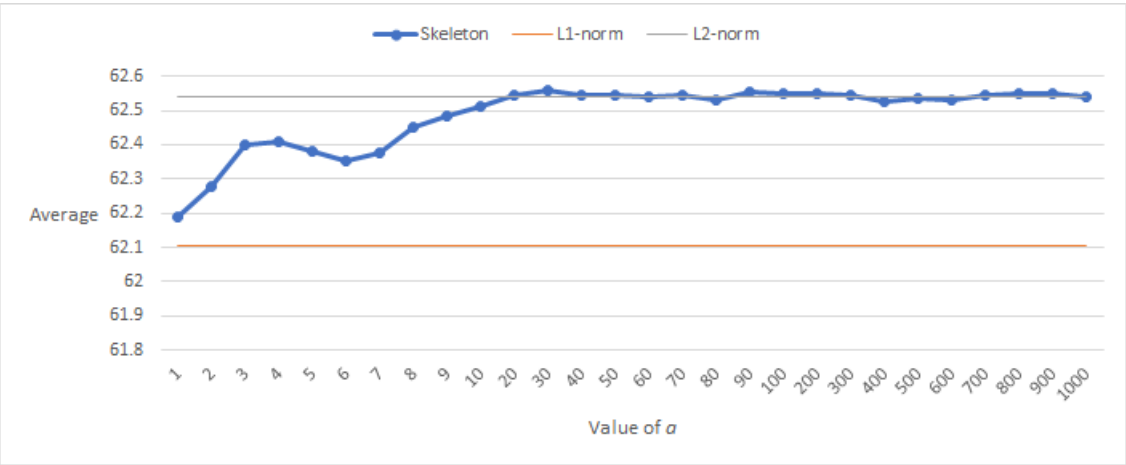


Figure 2.5: Different results with different a for UCI Letter

Already it is clear that results above 20 are preferable and the trend is much smoother suggesting that the scale is a little more appropriate than before. This makes again sense because the more interesting results are on the right side and

thus a is big so changes affect it less.
Anyway the best results seem to be around 30 or 90 so here is the test outcomes for both, highlighting that the first option was indeed the best one:

Table 2.13: OPTIMIZE A: UCI LETTER TEST RESULTS

a Value	Average Classification Rate
30	61.84
90	61.83
L1-norm	61.40
L2-norm	61.80

Again the search could be expanded around those values and maybe improve even further but here it is out of scope.

In conclusion, the choice problem can be expressed in other words: instead of guessing if the dataset has or not outliers, the decision can be made on the L1-norm and L2-norm PCA results: if the first wins then concentrate on the small a, otherwise on large ones.

Chapter 3

Generalized Kernel PCA

As previously mentioned, this new derivation takes a lot from the algorithm in [20] but since it is still different the full derivation is reported here.

The general maximization to be solved is the most generalized form possible of 1.7 similarly to 2.1:

$$\begin{aligned} &\arg\max_{w} \quad \sum_{i=1}^{N} f(w^T \Phi(x_i)) \\ &\text{subject to} \quad ||w||_2 = 1 \end{aligned} \tag{3.1}$$

The problem can be tackled with the Lagrangian method (as in the A), that having only a condition will result in:

$$\begin{aligned} &\arg\max_{w} \quad \min_{\lambda} \quad (\sum_{i=1}^{N} f(w^T \Phi(x_i)) - \lambda(||w||_2^2 - 1)) = L(w, \lambda) \\ &\text{subject to} \quad \lambda \geq 0 \end{aligned} \tag{3.2}$$

So now the Lagrangian function above $L(w, \lambda)$ has to be derived over w and made equal to zero to obtain the optimal formulation of the PC itself:

$$\frac{\partial L(w, \lambda)}{\partial w} = \sum_{i=1}^{N} f'(w^T \Phi(x_i)) \Phi(x_i) - 2\lambda w = 0 \tag{3.3}$$

Now the w expression can recovered, it just remains to apply the starting constraint $||w||_2 = 1$ to it and get:

$$w = \frac{\sum_{i=1}^{N} f'(w^T \Phi(x_i)) \Phi(x_i)}{|| \sum_{i=1}^{N} f'(w^T \Phi(x_i)) \Phi(x_i) ||_2} \tag{3.4}$$

This expression is the optimal form of our PC but it cannot be used directly in a iteration because it has $\Phi(\cdot)$ and not K so some adjustments must be done.

Let's start by unrolling the denominator and defining a new term $c_i = f'(w^T \Phi(x_i))$:

$$w = \frac{\sum_{i=1}^{N} c_i \Phi(x_i)}{\sqrt{\sum_{i,j=1}^{N} c_i c_j K_{ij}}} \tag{3.5}$$

Multiplying both sides by $\Phi(x_k)^T$ makes $\Phi(\cdot)$ on the right side disappear:

$$\Phi(x_k)^T w = \frac{\sum_{i=1}^{N} c_i K_{ki}}{\sqrt{\sum_{i,j=1}^{N} c_i c_j K_{ij}}} \quad \forall k \in [1, N] \tag{3.6}$$

Now the left side is almost c_k so by transposing and applying on both sides $f'(\cdot)$ a usable iterative form emerges:

$$c_k = f'\left(\frac{\sum_{i=1}^{N} c_i K_{ki}}{\sqrt{\sum_{i,j=1}^{N} c_i c_j K_{ij}}} \right) \quad \forall k \in [1, N] \tag{3.7}$$

The transposition do not effect anything if not changing only the result from a column vector to a row one.
Just for clarity sake here below are reported the vectorialized forms of c with the iteration counter, α and w:

$$c(t+1) = f'\left(\frac{Kc(t)}{\sqrt{c(t)^T K c(t)}} \right) \tag{3.8}$$

$$\alpha = \frac{c_*}{\sqrt{c_*^T K c_*}} \tag{3.9}$$

$$w = \frac{\Phi(X) c_*}{\sqrt{c_*^T K c_*}} = \Phi(X)\alpha \tag{3.10}$$

Where c_* is the optimal value of c after convergence and t keeps tracks of the iterations.

So now it is time to write down the algorithm to extract one principal component in the GKPCA scenario, where that abbreviation stands for Generalized Kernel PCA:

Algorithm 3: GKPCA

Input: the Kernel matrix K
Result: the extracted c_*

initialize $c(0)$ as instructed, $t = 0$

repeat
  $c(t+1) = f'\left(\frac{Kc(t)}{||Kc(t)||}\right)$
  t = t+1
until $||c(t) - c(t-1)|| > eps$;

$c_* = c(t)$

The optimality of this algorithm is discussed later in this chapter.

Anyhow the problem is a maximization most likely non-smooth, if not directly non-convex, so a good initialization must be used.
In [13] the authors found out that placing $w(0) = maxnorm(X)/||maxnorm(X)||_2$ was a really good approach and also here it has been proven a very strong option. From that expression in the feature space it is possible to extract the initialization of c:

$$c(0)_j = f'(w(0)^T x_j) = f'\left(\frac{K_{ij}}{\sqrt{Kii}}\right) \qquad \text{where } i \to maxnorm(\Phi(X)) = \Phi(x_i) \tag{3.11}$$

All that is left now is also to adapt the way of doing the greedy subtraction to the Kernel space, so from Algorithm 2:

$$\Phi(X) = (I_M - ww^t)\Phi(X) \tag{3.12}$$

Multiplying each side by its own transpose in order to eliminate $\Phi(\cdot)$ and have only K, plus using the definition of Kernel PC (1.9) the previous expression can be written as:

$$K_{ij} = K_{ij} - \sum_{q=1}^{N} K_{iq}\alpha_q \sum_{q=1}^{N} K_{qj}\alpha_q \qquad \forall i, j \in [1, N] \tag{3.13}$$

Finally it is possible to write down the full algorithm to extract p PCs using the GKPCA:

Algorithm 4: Greedy GKPCA

Input: the train set matrix X
Result: the extracted α

initialize $K = \Phi(X)^T \Phi(X)$

for $i \leftarrow 1$ **to** p **do**
  Do **Algorithm 3** on K and get as result c_*
  $\alpha_i = \frac{c_*}{\sqrt{c_*^T K c_*}}$
  $K = K - K\alpha_i \alpha_i^T K$
end

Given everything above, now there is only left to test out what has been discovered.
The test cases will remain the same as the normal PCA (as described in the B) but some arrangements have to be made in order to still being able to test in the feature space. These much needed changes and other testing stuff are reported in the Appendix C.

3.1 Recurrent Neural Network Comparison

As mentioned before, the equation 3.7 represent indeed a valid update rule for a neural network like the one in figure 3.1. So now the generic $f'(\cdot)$ let us do

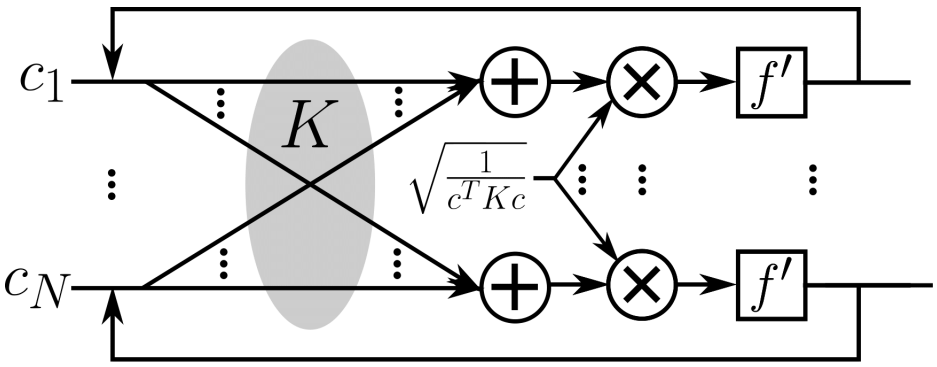


Figure 3.1: The Recurrent Neural Network for GKPCA.

GKPCA on this kind of recurrent neural network, so we may have for example, L1-norm KPCA with $sign(\cdot)$ or using the identity function for L2-norm and so on.

In particular, this RNN has two really important special cases and are the ones cited above.

With the $sign(\cdot)$ function we have a kind of almost canonical Hopfield Network describe better here [21]. This network could function in two ways: synchronous update in which all the weights are updated together and asynchronous update where they are update serially (and actually there could be a third in-between way but not too much interesting for our needs).
According to [23], the iterations then converge to a cycle of length 2 in general. Again, in practice, we have always observed convergence without any cyclic behavior. To guarantee convergence, one can implement $c \leftarrow \text{sign}(Kc)$ with serial, instead of parallel operation. In this case, the components of c are updated one at a time instead of all simultaneously [23]. With serial operation, the iteration is guaranteed to converge to a unique c. The serial implementation of the iterations $c \leftarrow \text{sign}(Kc)$ yields the exact same algorithm that is used in [20] to solve the L1-KPCA problem. Our work thus provides an alternative derivation and shows that Hopfield networks can also be understood as principal component analyzers.

Furthermore, in the GKPCA scenario is almost impossible to use the asynchronous update due to the impossibly long time to actually perform an update and converge, we did it anyway for small datasets like USPS and some in UCI but we did not discovered any significant difference with the synchronous methodology. Thus we proceeded that way and the literature agrees with us that is well set an Hopfield Network leads to the same results with both update rules [24].
So as far as L1-KPCA we do not need any other proofs.

Even more interesting to point out is the case with the identity activation function, because it actually does Von Mises (or Power) Iteration ([25] and [26]) that is a very famous numerical way to compute eigenvectors.
In fact, K is symmetric (based of its definition in the equation 1.6) thus always diagonalizable; this means that c will converge to a multiple of its dominant eigenvector with the iterative form of the equation 3.8 that is proportional to this form of the Power Method: $c^{(t+1)} = Kc^{(t)}$.
Again the L2-KPCA arrive at a local optimum with our algorithm.

Despite all that has been said above, with more generic functions this theory alone does not always guarantees to arrive at a local optimum so we need to go further.

3.2 Optimality Check

We want to check the local optimality of our two Algorithm 3 and because it is very similar to the GPCA case we can resume from the optimal proven form in Section 2.1.
The update equation 3.4 is equal to the one in the standard version here 2.2 except

from the input being in the feature space.
So the proof would be over if we used that form in our algorithm but we could not because it has the explicit computation of $\Phi(\cdot)$.

In the beginning of this chapter, we discussed how we passed from the equation 3.4 to 3.8 and the transformation is not at all linear so the optimality is not guaranteed to be conserved.

So part of the local optimality is already proved by the RNN parallel, enlisted in the section above.

At this stage, the full computation of the optimality was a little bit out of scope since it presented itself as a very hard process. Anyway the result showed pretty good convergence and pretty good results in almost every case so we leave the full check to a future project.
Interestingly enough, we noticed that also not non-decreasing functions converged and performed really well too, this could underline that the proof given to the GPCA could be additionally expanded even more.

3.3 Results

For the KPCA results there are little more things to adjust but let's go over them in order.
The type of Kernel Function used is the simple Gaussian $K(x_1, x_2) = e^{\frac{||x_1 - x_2||_2^2}{\sigma^2}}$ because once again we want just to bring the proof of concept that this generalization does really improve the normal Principal Component Analysis theory.
This poses another issue: the choice of σ. In our work we decided to optimize it by trial and error only once on the L2-KPCA and leave it untouched for all the other methods.
This inevitably plays in favor of L2 norm, but as seen in the results, it is surpassed anyway.

Since the samples are already altered by the Kernel function we loose that clean interpretation of large norms being most likely outliers/noisy sample.
In other words, deciding which function to use is not anymore an easy task. From the very beginning we would like to use the Lp-norms with $p = [0.5, 1.5]$ that proved to be valuable in the standard PCA by [13].
For the generic functions there will be a different approach enlisted in the Subsection 3.3.2.

The remaining problem is derived by the fact that we are testing in the feature space, so as the equations 3.11 and 3.13 had to be adjusted, also the testing methodologies need some work.
Since it is a lot of slightly tedious matrix and vector computations, the whole derivation is reported in the Appendix C.

The datasets have remained the same with roughly also the same test cases and the frame of reference is now only the L1 and L2 KPCA.

3.3.1 Lp-KPCA

Below in this subsection are reported all the different datasets results for the Lp-KPCA (GKPCA that uses Lp-norms) with $p = [0.5, 1.5]$ against the two "base" methodologies L2 and L1 norm based.

USPS: Being this a much rather simple dataset the two canonical L2-KPCA and L1-KPCA excel over our Lp. With the Gaussian noise it was easy to predict that L2 would win because being the same as the Kernel function and the optimized $\sigma = 2.5e3$ really boosted it.
With Sat&Pepper the L0.5 with higher noises proved to be valuable but not enough to be better on the average.

Table 3.1: Lp-KPCA: USPS WITH GAUSSIAN NOISE

Noise level (σ)	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)
0	95.73	96.26	95.41	**96.53**
10	95.69	96.28	96.34	**96.50**
20	95.71	96.27	96.32	**96.46**
30	95.66	96.24	96.26	**96.40**
40	95.57	96.18	96.17	**96.30**
50	95.43	96.08	96.01	**96.09**
60	94.20	**95.81**	95.74	95.79
70	94.79	**95.35**	95.30	**95.35**
80	94.16	**94.62**	94.57	94.61
90	93.21	**93.54**	93.47	93.47
100	91.81	**91.98**	91.86	91.80
Average	94.82	95.33	95.31	**95.39**

Table 3.2: Lp-KPCA: USPS WITH S&P NOISE

Noise level (n)	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)
0	95.73	96.26	95.41	**96.53**
0.05	95.56	96.07	96.11	**96.25**
0.1	95.15	95.65	95.68	**95.79**
0.15	94.36	94.83	94.85	**94.94**
0.2	93.00	93.36	94.37	**93.40**
0.25	90.79	**90.94**	90.85	90.84
0.3	87.08	**87.19**	86.90	86.83
0.35	**81.92**	81.89	81.36	81.21
0.4	**75.25**	75.22	74.42	74.20
0.45	**67.69**	67.69	66.66	66.44
0.5	**59.78**	59.74	58.65	58.41
Average	85.12	**85.35**	85.02	84.99

Yale: Since it is no more that easy a scenario L2 even with its optimized $\sigma = 7e3$ cannot keep up with the noise, thus L1 and L1.5 are clearly superior, but the latter is indeed better.

Table 3.3: Lp-KPCA: YALE FACES

#noisy images	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)
15 (9%)	1.06	1.02	**1.01**	1.18
30 (18%)	1.13	1.09	**1.08**	1.32
45 (27%)	1.19	1.14	**1.12**	1.35
60 (36%)	1.25	1.20	**1.16**	1.36
75 (45%)	1.30	1.25	**1.19**	1.36
90 (54%)	1.34	1.30	**1.21**	1.37
103 (63%)	1.39	1.34	**1.23**	1.37
Average	1.24	1.19	**1.14**	1.33

MNIST: As for the USPS with Salt&Pepper noise, here the kind the noise is no more in favor of the L2 but L1 is indeed the best option. The two other norms struggles to keep up here.
The selected σ is equal to $5e3$.

Table 3.4: Lp-KPCA: MNIST WITH SPECKLE NOISE

variance	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)
0	3.66	3.62	3.65	**3.61**
0.04 (default)	3.77	3.67	3.66	**3.63**
0.2	4.04	**3.95**	4.00	3.97
0.4	4.63	**4.43**	4.51	4.49
0.5	4.81	**4.72**	4.79	4.81
0.8	5.55	**5.28**	5.35	5.36
1	5.78	5.63	5.57	**5.55**
2	6.56	**6.41**	**6.41**	6.57
4	7.51	**7.21**	7.37	7.32
5	7.62	**7.46**	7.58	7.71
8	8.14	8.01	**7.94**	8.13
10	8.22	**8.14**	8.22	8.38
Average	6.06	**5.71**	5.94	5.99

UCI: Since there are a lot of different datasets here, the optimized σ are reported in the second column Being noiseless the L2 should win but the L0.5 proved to be better in more databases but still losing by a small amount against L1.

Table 3.5: Lp-KPCA: UCI CLASSIFICATION RATES

Datasets	σ	$p = 0.5$	L1 ($p = 1$)	$p = 1.5$	L2 ($p = 2$)
Iris	2	97.40	98.01	97.98	**98.03**
Balance	2.5	77.27	**77.78**	76.74	77.41
Ionosphere	5	**92.76**	92.66	92.58	92.31
Yeast	0.5	**50.40**	48.58	49.35	48.14
Phishing	1	91.52	91.66	90.79	93.01
Liver	15	66.67	66.89	**67.12**	67.01
Waveform	20	85.93	**86.41**	86.39	86.38
Letter	3	**87.77**	87.70	85.72	82.71
Car	2	91.01	**91.60**	90.58	90.01
Average		82.30	**82.37**	81.92	81.67

Just to wrap this subsection up, the Lp-norm KPCA proved to be valuable in some cases as it was for the PCA still again without optimizing on p itself.

For now these results are sufficiently satisfying, now it is time to explore even more this field with completely generic functions.

3.3.2 Purely GKPCA

We then moved to test the previous functions (*tanh*, *sech* and $g(w^T x, a)$) but they turned out not to be that great because again the intuition behind them holds no more in the feature space.
We resort to Gaussian like functions (in the first derivative stage) $h'(c, q) = e^{-|c|^q} sign(c)$ that aimed to amplify and support the effects of the Gaussian Kernel. The first argument is the iterative variable c and the other one is a free parameter as a in $g(w^T x, a)$, the absolute value and the $sign(\cdot)$ are placed in order not to lose the sign information of c with even q.

Anyhow, in the next tables the best result from the Lp-norms are reported against our h' with the top three values of q = [2,3,4] that with some trial and error performed the best.

Naturally the σ for the kernel are the same as before.

USPS: A little bit of the same issues with the Gaussian noise because L2 really is very boosted but still $q = 3$ managed to outperform it.

With Sat&Pepper our h' comes always on top.

Table 3.6: GKPCA: USPS WITH GAUSSIAN NOISE

Noise level (σ)	L2 ($p = 2$)	$q = 2$	$q = 3$	$q = 4$
0	**96.53**	96.41	96.43	96.49
10	**96.50**	96.35	96.45	96.47
20	**96.46**	96.34	96.40	96.38
30	**96.40**	96.28	96.34	96.35
40	**96.30**	96.20	96.24	96.19
50	**96.09**	95.95	96.08	95.99
60	95.79	95.70	**95.81**	95.79
70	95.35	95.26	**95.39**	95.27
80	94.61	94.64	**94.69**	94.56
90	93.47	93.48	**93.68**	93.45
100	91.80	91.94	**92.18**	91.88
Average	95.39	95.32	**95.42**	95.34

Table 3.7: GKPCA: USPS WITH S&P NOISE

Noise level (n)	L2 ($p = 2$)	$q = 2$	$q = 3$	$q = 4$
0	**96.53**	96.41	96.43	96.49
0.05	**96.25**	96.02	96.19	96.16
0.1	**95.79**	95.69	95.74	95.71
0.15	**94.94**	94.76	94.93	94.88
0.2	93.40	93.32	**93.51**	93.37
0.25	90.84	90.93	**91.14**	90.78
0.3	86.83	87.11	**87.45**	86.97
0.35	81.21	81.74	**82.14**	81.79
0.4	74.20	75.03	**75.46**	74.85
0.45	66.44	67.24	**67.92**	97.25
0.5	58.41	59.48	**60.05**	59.63
Average	84.99	85.25	**85.54**	85.26

Yale: Already L1.5 was better so it is an easy task for our new function to win as seen below.

Table 3.8: GKPCA: YALE FACES

#noisy images	$p = 1.5$	$q = 2$	$q = 3$	$q = 4$
15 (9%)	**1.01**	1.03	1.02	1.02
30 (18%)	**1.08**	1.09	1.09	1.09
45 (27%)	**1.12**	1.14	1.14	1.15
60 (36%)	**1.16**	1.21	1.18	1.21
75 (45%)	1.19	1.26	**1.17**	1.26
90 (54%)	1.21	1.28	**1.20**	1.30
103 (63%)	1.23	1.25	**1.13**	1.28
Average	1.14	1.18	**1.13**	1.19

MNIST: L1 was indeed the best option and even with h' it is still equal to the best value $q = 3$.
Probably this is due to the fact that the MNIST is such a huge dataset that Kernel approaches can not improve so drastically.

Table 3.9: GKPCA: MNIST WITH SPECKLE NOISE

variance	L1 ($p = 1$)	$q = 2$	$q = 3$	$q = 4$
0	**3.62**	3.76	3.74	3.63
0.04 (default)	**3.67**	3.76	**3.67**	**3.67**
0.2	3.95	4.00	**3.94**	3.99
0.4	**4.43**	4.45	4.46	4.50
0.5	4.72	4.70	**4.68**	4.78
0.8	**5.28**	5.42	5.40	5.29
1	5.63	5.59	**5.54**	5.56
2	6.41	6.52	6.41	**6.38**
4	7.21	7.22	**7.19**	7.31
5	7.46	7.48	**7.43**	7.53
8	8.01	8.00	**7.95**	7.97
10	8.14	**8.11**	**8.11**	8.15
Average	**5.71**	5.75	**5.71**	5.73

UCI: Being always the aim to be noise resilient, also for the new h', this test is the hardest.
Despite that, the standard norms do not come on top in the majority of the datasets and, for the first time, same goes for the average.

Table 3.10: GKPCA: UCI CLASSIFICATION RATES

Datasets	L1 ($p = 1$)	$q = 2$	$q = 3$	$q = 4$
Iris	98.01	**98.02**	97.99	97.96
Balance	**77.78**	77.55	77.71	77.65
Ionosphere	92.66	92.51	**92.69**	92.60
Yeast	48.58	48.89	**48.90**	48.75
Phishing	91.66	91.64	**91.71**	91.68
Liver	66.89	66.92	**66.98**	66.93
Waveform	86.41	86.44	**86.51**	86.39
Letter	87.70	**87.93**	87.79	87.79
Car	91.60	**91.64**	91.59	91.46
Average	82.37	82.40	**82.43**	82.36

Table 3.11: GKPCA: UCI WINNING COUNT

	$q = 2$	$q = 3$	$q = 4$
L1 ($p = 1$)	3	3	-1
L2 ($p = 2$)	3	3	3

Finally we managed also in the Kernel space to obtain pretty good result and outperform the literature standards. Even better than the normal PCA we also improved more greatly on noiseless cases.

Chapter 4

Conclusion

In this thesis we proposed a model that generalize completely both the standard PCA and the Kernel PCA, giving these methods newly found customization possibilities.
From the model itself we designed two distinct algorithms able to tackle efficiently those problems and proved them to be locally optimal, under certain function constraint and experimentally viable for even completely generic functions.
As a proof of concept we have finally tested them against the state-of-the-arts in Principal Component Analysis theory and found that even not overly accurately chosen function can outperform them all.
The tests were conducted on a large variety of very different datasets and noises just to be sure that our ideas were also adaptable.

Lastly, we think there is a huge margin of improvement just only by considering the two newly proposed functions $g(w^T x, a)$ and $h'(c, q)$, they performed almost always better than anything else, just by adjusting their free parameters without optimizing directly onto those. A proper study of these two could possibly increase even more the performances.
Same things about the choice of the function in general, we believe that really tailoring it down to the specific application and dataset could really make the difference.

Appendices

Appendix A

Derivation of Algorithm 1

The general optimization to solve is the 2.1, first thing first it has to be added the absolute value, this not only simplifies the choice of a function (by only requiring it to be defined for $\mathbb{R} \geq 0$) but also allows us to come closer to the canonical problem based on Lp-norms.
The obvious next step is to do a Gradient Ascent on this function, so the first thing is to find the derivative in respect to our maximization variable w:

$$\nabla_{w_i} = \frac{\partial f(|w^T x_i|)}{\partial w} = \frac{df(u)}{du} \cdot \frac{du}{dw} = f'(u) sign(w^t x_i) x_i = f'(|w^t x_i|) sign(w^t x_i) x_i \tag{A.1}$$

The change of variable $u = |w^t x_i|$ is done to compute the derivative of f more smoothly.
Next up is to compute the Lagrangian Optimization and see if something better is obtained:

$$L(w, \lambda) = \sum_{i=1}^{N} f(|w^T x_i|) + \lambda(||w||_2^2 - 1) \tag{A.2}$$

Then by computing the derivative of the following equation and make it equal to zero we check the form of the optimal solution:

$$\frac{\partial L(w, \lambda)}{\partial w} = \sum_{i=1}^{N} \nabla_{w_i} + \lambda w = 0 \tag{A.3}$$

This indicates that an optimal solution to the problem will have to make w equal (or better speaking parallel) to ∇_w.
The only thing that remains is to apply the constraint and get this new update rule ready to use:

$$w \leftarrow \frac{\sum_{i=1}^{N} \nabla_{w_i}}{|| \sum_{i=1}^{N} \nabla_{w_i} ||_2} \tag{A.4}$$

Lastly, extracting more than one PC is done Greedly and the initialization, that matters a lot since this method is basically a Gradient Ascent and thus it could get stuck in a local minimum, is the normalized sample with the greatest norm.
The main reason behind it is because it has been proven experimentally to be a really good option, again, in the paper [13].

Appendix B

How the testing has being carried on

Each datasets has its own tricks and test suit(s), so everything is explained below, divided for each set. Anyhow something is still shared among all: every time some random operation (like adding noise) is involved, the results are given with an approximate confidence of 95%, if not otherwise specified the number of extracted loading vector is 30 (the focus was not in obtaining the right number of PCs by trading of cost and performance but just compare methods) and the testing is usually done by classification.
Each test sample y is classified as the class j that minimize the reconstruction error based of our PCs for that class ($w_{ji} \in W_j \qquad \forall i \in [1, N]$), than it is computed the percentage of successful classifications.

$$\text{class\_classified}(y) = \arg\min_j \quad ||(y - \mu_j) - \sum_{i=1}^{N} w_{ji} w_{ji}^T (y - \mu_j)||_2^2 = ||(y - \mu_j) - W_j W_j^T (y - \mu_j)||_2^2 \quad \text{(B.1)}$$

Where μ_j is the mean vector of j^{th} class train set.

- USPS digits
  The USPS digit dataset [27] consists of 1100 examples of 16×16 grayscale images of handwritten digits.
  This dataset has been tested with both Gaussian and Salt&Pepper noise: the first is the standard additive Gaussian noise with standard deviation σ, the second one is given in respect of n that is the probability that a pixel flips black or white indistinctly.

The division training and test was the same as the paper [13] listed: train on the first 300 and test on the last 800 of every digit.

- MNIST
  The famous MNIST digit dataset [28] does not need much introduction and it is divided in train and test as suggested by their authors, then it is introduced some speckle noise with increasing variance. It is a kind of multiplicative noise usually due to capturing tools in the medical field.
  According to the standard notation on the site, the results are given in percentage of misclassifications instead of successful ones.

- Yale Faces
  The Yale Face dataset [29] is composed by 165 grayscale images of 15 individuals each having the same number of observations, the actual dataset used was taken in this adapted form from MIT [30].
  There are no classes and the loading vectors are extracted from all the faces. The test was devised as the [13] did in the second experiment, so some dummy images composed of just random black and white pixels were added to the set. With the extracted PCs the reconstruction error (like the one minimized in B.1) is computed for the original images and the result is a mean of all those: the lower the better.
  For representation sake the results are always scaled back with a factor of 10^3 in the standard version and up by 10 in the Kernel. Lastly, with KPCA only 5 PCs were extracted.

- UCI repository
  A lot of datasets were used from this repository but the test is the same: only extract one PC and proceed to classification right away; the division in train and test is done with a n-random-fold cross validation. The reconstruction error has been computed as well as how many times our method obtain an higher result than the norms.
  The observations and classes are usually numerical (or integers or real or Boolean) but every single datasets has its own formatting and number of samples (that goes from around 10 to around 10^4).
  For the whole description of each one of them, it is advised to go on the UCI website [31].

Appendix C

Feature Space Testing

The problem remains the same as the one in B.1 but it should be brought into the feature space.
Let's begin by using the previous equation with zero mean for simplicity and the first step is to expand the square:

$$\arg\min_j \quad ||y||^2 - 2\sum_{i=1}^{N} y^T w_{ji} w_{ji}^T y + ||\sum_{i=1}^{N} w_{ji} w_{ji}^T y||^2 = \\ ||y||^2 - 2y^T W_j W_j^T y + ||W_j W_j^t y||^2 \quad \text{(C.1)}$$

Focusing on the last term that can be expanded once more, resulting in this: $y^T W_j W_j^T W_j W_j^T y$.
Given that our PCs have unit norm that result can be collapsed in $y^T W_j W_j^T y$ and substituted back in C.1:

$$\arg\min_j \quad ||y||^2 - y^T W_j W_j^T y = ||y||^2 - ||W_j^T y||^2 \quad \text{(C.2)}$$

Now the Kernel space can be finally introduced:

$$\arg\min_j \quad ||\Phi(y)||^2 - ||W_j^T \Phi(y))||^2 \quad \text{(C.3)}$$

By then using the definition of Kernel matrix (1.6) and the one of the PC in the feature space (1.9), the previous equation can be rewritten as such:

$$\arg\min_j \quad K(y,y) - ||\sum_{i=1}^{N} \alpha_{ji}^T K(x_{ji}, y)||^2 = K(y,y) - ||\alpha_j^T K(X_j, y)||^2 \quad \text{(C.4)}$$

Now there are two problems that remains to be solved, the first is that in 1.9 the loading vectors have to be multiplied to the $\Phi(X_j)$ from which their α_{ji} are actually extracted so while with the L2-KPCA they are all extracted from the same original

matrix, in our iterative algorithm the matrix changes with each PC, so this should be taken into account.
This means that in the last term the matrix $K(X_j, y)$ should change at each principal component, or in other words every single α_{ji} should have its matrix.
One way to do that is reapplying the same greedy subtraction principle here (3.12) and after having computed the value of the first element of the summation subtract the just used PC to the matrix and repeat the process over and over again.
This is can be done by using this iterative form of the last matrix:

$$K(X_{ji}, y) \leftarrow K(X_{ji}, y) - (\sum_{k=1}^{n} K(y, X_{jk})\alpha_{jk})(\sum_{k=1}^{n} K(X_{ji}, X_{jk})\alpha_{jk}) \qquad \forall i \in [1, n] \tag{C.5}$$

Also the kernel matrix of only the train set should be updated but instead the whole multiplication can be already precomputed while extracting the PC and thus with the "right" values each time. Indeed they are called scores and they are projection of the train set onto the found loading vectors: $score_j = K(X_j, X_j)\alpha_j$.

The other problem to be solved is the fact that most likely the data will not have zero mean and for the classification, the mean to subtract to the test sample should be the one of the class j in consideration.
The pure mean is never computed from the train samples but it is always used through 1.8 so the same thing can be done to the two matrices that need to be centralized:

$$\begin{aligned} K(y, y) \leftarrow ||\Phi(y) - \frac{1}{N}\sum_{i=1}^{N}\Phi(X_{ji})||_2^2 = \\ K(y, y) - \frac{1}{N}\sum_{i=1}^{N}K(X_{ji}, y) - \frac{1}{N}\sum_{i=1}^{N}K(y, X_{ij}) + \frac{1}{N^2}\sum_{i=1}^{N}\sum_{k=1}^{N}K(X_{ji}, X_{jk}) = \\ K(y, y) - \frac{2}{N}\sum_{i=1}^{N}K(X_{ji}, y) + \frac{1}{N^2}\sum_{i,k=1}^{N}K(X_{ji}, X_{jk}) \end{aligned} \tag{C.6}$$

Same goes for the matrix between test and train samples:

$$\begin{aligned} K(y, X_{jq}) \leftarrow ||\Phi(y) - \frac{1}{N}\sum_{i=1}^{N}\Phi(X_{ji}))^T(\Phi(X_{jq}) - \frac{1}{N}\sum_{i=1}^{N}\Phi(X_{ji})||_2^2 = \\ K(y, X_{jq}) - \frac{1}{N}\sum_{i=1}^{N}K(y, X_{ji}) - \frac{1}{N}\sum_{i=1}^{N}K(X_{ji}, X_{jq}) + \frac{1}{N^2}\sum_{i,k=1}^{N}K(X_{ji}, X_{jk}) \end{aligned} \tag{C.7}$$

Bibliography

[1] S. Battaglino, E. Koyuncu, and E. M. Baralis. A generalization of principal component analysis. Master's thesis, University of Illinois at Chicago (UIC), Dec 2019.

[2] Samuele Battaglino and Erdem Koyuncu. A generalization of principal component analysis, 2019. arXiv:1910.13511.

[3] J. Shlens. A tutorial on principal component analysis. *ArXiv e-prints*, 2014. https://arxiv.org/abs/1404.1100 [Online; accessed 09/19/2019].

[4] H. Abdi and L. J. Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433 – 459, 2010.

[5] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, Berlin, Germany, 1986.

[6] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559 – 572, 1901.

[7] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417 – 441, 1933.

[8] J. B. Tenenbaum, V De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science (New York, N.Y.)*, 290(5500):2319 – 2323, 2000.

[9] H. Hoffmann. Kernel pca for novelty detection. *Pattern Recognition*, 40(3):863 – 874, 2007.

[10] C. Ding and X. He. K-means clustering via principal component analysis. In *ICML Proceedings of the twenty-first international conference on Machine learning*, 2004.

[11] G. Stefatos and A. B. Hamza. Cluster pca for outliers detection in high-dimensional data. In *IEEE International Conference on Systems, Man and Cybernetics*, Montreal, Que., Canada, 2007.

[12] P. P. Markopoulos, G. N. Karystinos, and D. A. Pados. Optimal algorithms for l1-subspace signal processing. *IEEE Transactions on Signal Processing*, 62(19):5046 – 5058, Jul. 2014.

[13] N. Kwak. Principal component analysis by lp-norm maximization. *IEEE Transactions on Cybernetics*, 44(5):594 – 609, May 2014.

[14] C. Ding, D. Zhou, X. He, and H. Zha. R 1-pca: rotational invariant l 1-norm

principal component analysis for robust subspace factorization. In *Proceedings of the 23rd international conference on Machine learning*, pages 281 – 288. ACM, 2006.

[15] F. Nie, H. Huang, C. H. Q. Ding, D. Luo, and H. Wang. Robust principal component analysis with non-greedy l1-norm maximization. In *IJCAI*, 2011.

[16] B. Schölkopf, A. Smola, and K. Müller. Kernel principal component analysis. In *Artificial Neural Networks — ICANN'97*, Lecture Notes in Computer Science, vol 1327, Heidelberg, Berlin, Germany, 1997.

[17] Q. Wang. Kernel principal component analysis and its applications in face recognition and active shape models. *ArXiv e-prints*, 2011. https://arxiv.org/abs/1207.3538 [Online; accessed 08/27/2019].

[18] K. In Kim, M. O. Franz, and B. Schölkopf. Iterative kernel principal component analysis for image modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9):1351 – 1366, Sep 2005.

[19] T. Chin and D. Suter. Incremental kernel principal component analysis. *IEEE Transactions on Image Processing*, 16(6):1662 – 1674, 2007.

[20] C. Kim and D. Klabjan. L1-norm kernel pca. *ArXiv e-prints*, Aug 2015. https://arxiv.org/abs/1709.10152 [Online; accessed 04/15/2019].

[21] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79:2554 – 2558, 1982.

[22] N. Kwak. Principal component analysis based on l1-norm maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9):1672 – 1680, 2008.

[23] J. Bruck. On the convergence properties of the hopfield model. *Proceedings of the IEEE*, 78(10):1579 – 1585, 1990.

[24] J. Mandziuk. Solving the n-queens problem with a binary hopfield-type network. synchronous and asynchronous model. *Biological Cybernetics*, 72(5):439 – 446, 1995.

[25] R. Von Mises and H. Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung. *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, 9:152 – 164, 1929.

[26] Å. Björck. *Numerical methods in matrix computations*, volume 59. Springer, 2015.

[27] S. Roweis. Data for matlab hackers, 2009. https://cs.nyu.edu/~roweis/data.html [Online; accessed 02/20/2019].

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 – 2324, Nov. 1998.

[29] P. Belhumeur, J. Hespanha, and D. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711 – 720, Jul. 1997.

[30] MIT-Media-Lab. The normalized yale face database, 2000. https://vismod.media.mit.edu/vismod/classes/mas622-00/datasets/ [Online; accessed 02/21/2019].

[31] D. Dua and C. Graff. UCI machine learning repository, university of california, irvine, school of information and computer sciences, 2017. http://archive.ics.uci.edu/ml [Online; accessed 03/08/2019].