Differential Machine Learning – Appendix 2
Taking the First Step : Differential PCA

Brian Huge
brian.huge@danskebank.dk

Antoine Savine
antoine.savine@danskebank.dk

May 4, 2020

Introduction

We review traditional data preparation in deep learning (DL) including principal component analysis (PCA), which effectively performs orthonormal transformation of input data, filtering constant and redundant inputs, and enabling more effective training of neural networks (NN). Of course, PCA is also useful in its own right, providing a lower dimensional latent representation of data variation along orthogonal axes.

In the context of *differential* DL, training data also contains differential labels (differentials of training labels wrt training inputs, computed e.g. with automatic adjoint differentiation -AAD- as explained in the working paper), and thus requires additional preprocessing.

We will see that differential labels also enable remarkably effective data preparation, which we call *differential PCA*. Like classic PCA, differential PCA provides a hierarchical, orthogonal representation of data. Unlike classic PCA, differential PCA represents input data in terms how it affects the target measured by training labels, a notion we call *relevance*. For this reason, differential PCA may be safely applied to aggressively remove irrelevant factors and considerably reduce dimension.

In the context of data generated by financial Monte-Carlo paths, differential PCA exhibits the principal risk factors of the target transaction or trading book from data alone. It is therefore a very useful algorithm on its own right, besides its effectiveness preparing data for training NN.

The first section describes and justifies elementary data preparation, as implemented in the demonstration notebook *DifferentialML.ipynb* on `https://github.com/differential-machine-learning`. Section 2 discusses the mechanism, benefits and limits of classic PCA. Section 3 introduces and derives differential PCA and discusses the details of its implementation and benefits. Section 4 brings it all together in pseudocode.

1 Elementary data preparation

Dataset normalization is known as a crucial, if somewhat mundane preparation step in deep learning (DL), highlighted in all DL textbooks and manuals. Recall from the working paper that we are working with augmented datasets:

$$X^{(i)} \text{ : inputs, } Y^{(i)} \text{ : labels, and } Z^{(i)} = \frac{\partial Y^{(i)}}{\partial X^{(i)}} \text{ : differential labels}$$

with m labels in dimension 1 and m inputs and m differentials in dimension n, stacked in m rows in the matrices X, Y and Z. In the context of financial Monte-Carlo simulations *a la* Longstaff-Schwartz, inputs are Markov states on a *horizon date* $T_1 \geq 0$, labels are payoffs sampled on a later date T_2 and differentials are pathwise derivatives, produced with AAD.

The normalization of augmented datasets must take additional steps compared to conventional preparation of classic datasets consisting of only inputs and labels.

1.1 Taking the first (and last) step

A first, trivial observation is that the scale of labels $Y^{(i)}$ carries over to the gradients of the cost functions and the size of gradient descent optimization steps. To avoid manual scaling of learning rate, gradient descent and variants are best implemented with labels normalized by subtraction of mean and division by standard deviation. This is the case for all models trained with gradient descent, including classic regression in high dimension where the closed form solution is intractable.

Contrarily to classic regression, training a neural network is a nonconvex problem, hence, its result is sensitive to the starting point. Correctly seeding connection weights is therefore a crucial step for successful training. The best practice *Xavier-Glorot* initialization provides a powerful seeding heuristic, implemented in modern frameworks like TensorFlow. It is based on the implicit assumption that the units in the network, including inputs, are centred and orthonormal. It therefore performs best when the inputs are at the very least normalized by mean and standard deviation, and ideally orthogonal. This is specific to neural networks. Training classic regression models, analytically or numerically, is a convex problem, so there is no need to normalize inputs or seed weights in a particular manner.

Training deep learning models therefore always starts with a normalization step and ends with a 'un-normalization step' where predictions are scaled back to original units. Those first and last step may be seen, and implemented, as additional layers in the network with fixed (non learnable) weights. They may even be merged in the input and output layer of the network for maximum efficiency. In this document, we present normalization as a preprocessing step in the interest of simplicity.

1.1.1 First step

We implemented basic preprocessing in the demonstration notebook:

$$\tilde{Y}^{(i)} = \frac{Y^{(i)} - \mu_Y}{\sigma_Y} \text{ and } \tilde{X}_j^{(i)} = \frac{X_j^{(i)} - \mu_j}{\sigma_j}$$

where

$$\mu_Y = \frac{1}{m}\sum_{i=1}^{m} Y^{(i)} \text{ and } \mu_j = \frac{1}{m}\sum_{i=1}^{m} X_j^{(i)}$$

and similarly for standard deviations of labels σ_Y and inputs σ_j.

The differentials computed by the prediction model (e.g. the twin network of the working paper) are:

$$\frac{\partial \tilde{Y}}{\partial \tilde{X}_j} = \frac{\sigma_j}{\sigma_Y}\frac{\partial Y}{\partial X_j}$$

hence, we adjust differential labels accordingly:

$$\tilde{Z}_j^{(i)} = \frac{\sigma_j}{\sigma_Y} Z_j^{(i)}$$

1.1.2 Training step

The value labels $\tilde{Y}$ are centred and unit scaled but the differentials labels $\tilde{Z}$ are not, they are merely re-expressed in units of 'standard deviations of Y per standard deviation of X_j'. To avoid summing apples and oranges in the combined cost function as commented in the working paper, we scale cost as follows:

$$C = \frac{1}{m}\sum_{i=1}^{m}\left[\hat{Y}^{(i)}(w) - \tilde{Y}^{(i)}\right]^2 + \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{1}{||\tilde{Z}_j||^2}\left[\hat{Z}_j^{(i)}(w) - \tilde{Z}^{(i)}\right]^2 \tag{C}$$

and proceed to find the optimal biases and connection weights by minimization of C in w.

1.1.3 Last step

The trained model $\tilde{f}$ expects normalized inputs and predicts a normalized value, along with its gradient to the normalized inputs. Those results must be scaled back to original units:

$$f(x) = \mu_Y + \sigma_Y \tilde{f}(\tilde{x}) = \mu_Y + \sigma_Y \tilde{f}\left(\frac{x - \mu_X}{\sigma_X}\right)$$

where we divided two row vectors to mean elementwise division, and:

$$\frac{\partial f(x)}{\partial x_j} = \frac{\sigma_Y}{\sigma_j}\frac{\partial \tilde{f}(\tilde{x})}{\partial \tilde{x}_j}$$

1.2 Limitations

Basic data normalization is sufficient for textbook examples but more thorough processing is necessary in production, where datasets generated by arbitrary schedules of cashflows simulated in arbitrary models may contain a mass of constant, redundant of irrelevant inputs. Although neural networks are supposed to correctly sort data and identify relevant features during training[1], in practice, nonconvex optimization is much more reliable when at least *linear* redundancies and irrelevances are filtered in a preprocessing step, lifting those concerns from the training algorithm and letting it focus on the extraction of *nonlinear* features.

In addition, it is best, although not strictly necessary, to train on orthogonal inputs. As it is well known, normalization and orthogonalization of input data, along with filtering of constant and linearly redundant inputs, is all jointly performed in a principled manner by eigenvalue decomposition of the input covariance matrix, in a classic procedure called principle component analysis or PCA.

2 Principal Component Analysis

2.1 Mechanism

We briefly recall the mechanism of data preparation with classic PCA. First, normalize labels and center inputs:

$$\tilde{Y}^{(i)} = \frac{Y^{(i)} - \mu_Y}{\sigma_Y} \text{ and } X_j^{(i)} \equiv X_j^{(i)} - \mu_j$$

i.e. what we now call X is the matrix of *centred* inputs. Perform its eigenvalue decomposition:

[1]SVD regression performs a similar task in the context of classic regression, see note on differential regression.

$$\frac{1}{m}X^TX = PDP^T$$

where P is the orthonormal $n \times n$ matrix of eigenvectors (in columns) and D is the diagonal matrix of eigenvalues D_{jj}.

Filter numerically constant or redundant inputs identified by eigenvalues D_{jj} lower than a threshold ϵ. The filter matrix F has n rows and $\tilde{n} \leq n$ columns and is obtained from the identity matrix I_n by removal of columns corresponding to insignificant eigenvalues D_{jj}. Denote:

$$\tilde{D} = F^TDF \text{ and } \tilde{P} = PF$$

the reduced eigenvalue and eigenvector matrices of respective shapes $\tilde{n} \times \tilde{n}$ and $n \times \tilde{n}$, and apply the following linear transformation to centred input data:

$$\tilde{X} = X\tilde{P}\tilde{D}^{-\frac{1}{2}}$$

The transformed data has shape $m \times \tilde{n}$, with constant and linearly redundant columns filtered out. It is evidently centred, and easily proved orthonormal:

$$\begin{aligned}
\frac{1}{m}\tilde{X}^T\tilde{X} &= \tilde{D}^{-\frac{1}{2}}\tilde{P}^T\left(\frac{1}{m}X^TX\right)\tilde{P}\tilde{D}^{-\frac{1}{2}} \\
&= \left(F^TDF\right)^{-\frac{1}{2}}(PF)^T\left(\frac{1}{m}X^TX\right)(PF)\left(F^TDF\right)^{-\frac{1}{2}} \\
&= \left(F^TDF\right)^{-\frac{1}{2}}F^TP^TPDP^TPF\left(F^TDF\right)^{-\frac{1}{2}} \\
&= \left(F^TDF\right)^{-\frac{1}{2}}\left(F^TDF\right)\left(F^TDF\right)^{-\frac{1}{2}} \\
&= I_{\tilde{n}}
\end{aligned}$$

Note for what follows that orthonormal property is preserved by rotation, i.e. right product by any orthonormal matrix Q:

$$\frac{1}{m}\left(\tilde{X}Q\right)^T\left(\tilde{X}Q\right) = Q^T\left(\frac{1}{m}\tilde{X}^T\tilde{X}\right)Q = Q^TI_{\tilde{n}}Q = Q^TQ = I_{\tilde{n}}$$

To update differential labels, we apply a result from elementary multivariate calculus:

> Given two row vectors A and B in dimension p and a square non singular matrix M of shape $p \times p$ such that $B = AM$, and $y = g(A) = h(B)$ a scalar, then:
>
> $$\frac{\partial y}{\partial B} = \frac{\partial y}{\partial A}M^{-T}$$
>
> The proof is left as an exercise.

It follows that:

$$\begin{aligned}
\tilde{Z}^{(i)} &= \frac{\partial \tilde{Y}^{(i)}}{\partial \tilde{X}^{(i)}} \\
&= \frac{\partial \left(Y^{(i)} / \sigma_Y \right)}{\partial \left(X^{(i)} \tilde{P} \tilde{D}^{-\frac{1}{2}} \right)} \\
&= \frac{1}{\sigma_Y} \frac{\partial Y^{(i)}}{\partial X^{(i)}} \left(\tilde{P} \tilde{D}^{-\frac{1}{2}} \right)^{-T} \\
&= \frac{1}{\sigma_Y} Z^{(i)} \tilde{P} \tilde{D}^{\frac{1}{2}}
\end{aligned}$$

Or the other way around:

$$\frac{\partial y}{\partial x} = \sigma_Y \frac{\partial \tilde{y}}{\partial \tilde{x}} \tilde{D}^{-\frac{1}{2}} \tilde{P}^T$$

We therefore train the ML model $\tilde{f}$ more effectively on transformed data:

$$\tilde{X} = (X - \mu_X) \tilde{P} \tilde{D}^{-\frac{1}{2}} \ , \tilde{Y} = \frac{Y - \mu_Y}{\sigma_Y} \text{ and } \tilde{Z} = \frac{1}{\sigma_Y} Z \tilde{P} \tilde{D}^{\frac{1}{2}}$$

by minimization of the the cost function (C) in the learnable weights. The trained model $\tilde{f}$ takes inputs $\tilde{x}$ in the tilde basis and predicts normalized values $\tilde{y}$ and differentials $\partial \tilde{y} / \partial \tilde{x}$. Finally, we translate predictions back in the original units:

$$f(x) = \mu_Y + \sigma_Y \tilde{f}(\tilde{x}) \text{ and } \frac{\partial f(x)}{\partial x} = \sigma_Y \frac{\partial \tilde{f}(\tilde{x})}{\partial (\tilde{x})} \tilde{D}^{-\frac{1}{2}} \tilde{P}^T \text{ where } \tilde{x} = (x - \mu_x) \tilde{P} \tilde{D}^{-\frac{1}{2}}$$

PCA performs an orthonormal transformation of input data, removing constant and linearly redundant columns, effectively cleaning data to facilitate training of NN. PCA is also useful in its own right. It identifies the main axes of variation of a data matrix and may result in a lower dimensional latent representation, with many applications in finance and elsewhere, covered in vast amounts of classic literature.

PCA is limited to *linear* transformation and filtering of *linearly* redundant inputs. A nonlinear extension is given by autoencoders (AE), a special breed of neural networks with bottleneck layers. AE are to PCA what neural networks are to regression, a powerful extension able to identify lower dimensional *nonlinear* latent representations, at the cost of nonconvex numerical optimization. Therefore, AE themselves require careful data preparation and are not well suited to prepare data for training other DL models.

2.2 Limitations

2.2.1 Further processing required

In the context of a differential dataset, we cannot stop preprocessing with PCA. Recall, we train by minimization of the cost function (C), where derivative errors are scaled by the size of differential labels. We will experience numerical instabilities when some differential columns are identically zero or numerically insignificant. This means the corresponding inputs are *irrelevant* in the sense that they don't affect labels in any of the training examples. They really should not be part the training set, all they do is unnecessarily increase dimension, confuse optimizers and cause numerical errors. But PCA cannot eliminate them because it operates on inputs alone and disregards labels and how inputs affect them. *PCA ignores relevance.*

Irrelevances may even appear in the orthogonal basis, even when inputs looked all relevant in the original basis. To see that clearly, consider a simple example in dimension 2, where X_1 and X_2 are sampled from 2 standard Gaussian distributions with correlation $1/2$ and $Y = X_2 - X_1 + \text{noise}$. Differential labels are constant across examples with $Z_1 = -1$ and $Z_2 = 1$. Both differentials are clearly nonzero and both inputs appear to be relevant. PCA projects data on orthonormal axes $\tilde{X}_1 = (X_1 + X_2)/\sqrt{2}$ and $\tilde{X}_2 = (-X_1 + X_2)/\sqrt{2}$ with eigenvalues $3/2$ and $1/2$, and:

$$\tilde{Z}_1 = \frac{\partial Y}{\partial \tilde{X}_1} = \sqrt{2}\frac{\partial (X_2 - X_1)}{\partial (X_2 + X_1)} = 0$$

so after PCA transformation, one of the columns clearly appears irrelevant. Note that this is a coincidence, we would not see that if correlation between X_1 and X_2 were different from $1/2$. PCA is not able to identify axes of relevance, it only identifies axes of variation. By doing so, it *may* accidentally land on axes with zero or insignificant relevance.

It appears from this example that, not only further processing is necessary, but also, desirable to eliminate irrelevant inputs and combinations of inputs in the same way that PCA eliminated constant and redundant inputs. Note that we don't want to *replace* PCA. We want to train on orthonormal inputs and filter constants and redundancies. What we want is *combine* PCA with a similar treatment of relevance.

2.2.2 Limited dimension reduction

The eventual amount of dimension reduction PCA can provide is limited, precisely because it ignores relevance. Consider the problem of a basket option in a correlated Bachelier model, as in the section 3.1 of the working paper. The states X are realizations of the n stock prices at T_1 and the labels Y are option payoffs, conditionally sampled at T_2. Recall that the price at T_1 of a basket option expiring at T_2 is a nonlinear scalar function (given by Bachelier's formula) *of a linear combination* $X \cdot a$ of the stock prices X at T_1, where a is the vector of weights in the basket. The basket option, which payoff is measured by Y, is only affected (in a nonlinear manner) by *one* linear risk factor $X \cdot a$ of X. Although the input space is in dimension n, the subspace of relevant risk factors is in dimension 1. Yet, when the covariance matrix of X is of full rank, PCA identifies n axes of orthogonal variation. It only reduces dimension when the covariance matrix is singular, to eliminates trivially constant or redundant inputs, even in situations where dimension could be reduced by significantly larger amounts with relevance analysis.

2.2.3 Unsafe dimension reduction

In addition, it is not desirable to attempt aggressively reducing dimension with PCA because it could eliminate relevant information. To see why, consider another, somewhat contrived example with 500 stocks, one of them (call it XXX) uncorrelated with the rest with little volatility. An aggressive application of PCA would remove that stock from the orthogonal representation, even in the context of a trading book dominated by a large trade on XXX. PCA *should not* be relied upon to reduce dimension, because it ignores relevance and hence, might accidentally remove important features. PCA must be applied *conservatively*, with filtering threshold ϵ set to numerically insignificant eigenvalues, to only eliminate definitely constant or linearly redundant inputs.

2.2.4 Principal components are not risk factors

The Bachelier basket example makes it clear that the orthogonal axes of variation identified by PCA are *not* risk factors. In general, PCA provides a meaningful orthonormal representation of the state vector X, but it doesn't say anything about the factors affecting the transaction or its cashflows measured by the labels Y.

3 Differential PCA

3.1 Introduction

The question is then whether we can design an additional, similar procedure to effectively extract from simulated data the risk factors *of a given transaction*, and *safely* reduce dimension by removal of *irrelevant* axes? We expect the algorithm to identify the basket weights as the only risk factor in the Bachelier example, and XXX alone as a major risk factor in the 500 stocks example. In the general case, we want to reliably extract a hierarchy of orthogonal risk factors and safely eliminate irrelevant directions.

To achieve this, we turn to differential labels $Z^{(i)} = \partial Y^{(i)} / \partial X^{(i)}$, which, in the context of simulated financial data, are either risk sensitivities or *pathwise differentials*[2]. The main proposition of differential machine learning is to leverage differential labels computed e.g. with AAD, and we have seen their effectiveness for approximation by neural networks (main article) or classic regression (appendices). Here, we will see that they also apply in the context of PCA, not to improve it, but to combine it with an additional procedure, which we call 'differential PCA', capable of exhibiting orthogonal risk factors and safely removing irrelevant combinations of inputs.

As a data preparation step, differential PCA may significantly reduce dimension, enabling faster, more reliable training of neural networks, and a reduced sensitivity to seeding and hyperparameters. In particular, We will see that differential PCA reduces dimension while *preserving* orthonormality of inputs from a prior PCA step.

In its own right, differential PCA reliably identifies risk factors from simulated data. Like traditional PCA, it only extracts *linear* factors, but unlike PCA, it analyses and transforms on data through the lens of *relevance*.

3.2 Derivation

Start with a dataset:

$$X : \text{inputs } X^{(i)} \ , \ Y : \text{labels } Y^{(i)}, \text{ and } Z : \text{differentials } Z^{(i)} = \frac{\partial Y^{(i)}}{\partial X^{(i)}} \text{ stacked in rows in matrices } X, Y, Z$$

possibly orthonormal by prior PCA, with differentials appropriately adjusted and constant and redundant inputs filtered out.

We want to apply a *rotation* by right multiplication of the input matrix X by an orthonormal matrix Q so as to preserve orthonormality:

$$\tilde{X} = XQ$$

so that the directional differentials $\tilde{Z}_j^{(i)} = \partial Y^{(i)} / \partial \tilde{X}_j^{(i)}$ are mutually orthogonal. Recall from the lemma page 4:

$$\tilde{Z} = ZQ^{-T} = ZQ$$

and we want $\tilde{Z}$ to be orthogonal, i.e.:

$$\frac{1}{m} \tilde{Z}^T \tilde{Z} = E$$

with E a diagonal matrix whose entries E_{jj} are the mean norms of the columns $\tilde{Z}_j$ of $\tilde{Z}$, in other terms, the *size* of differentials, also called *relevance*, in the tilde basis.

Since $\tilde{Z} = ZQ$,

[2]Depending on whether labels are ground truth or sampled, see working paper.

$$\frac{1}{m}\tilde{Z}^T\tilde{Z} = \frac{1}{m}Q^T Z^T Z Q = E$$

or inverting:

$$\frac{1}{m}Z^T Z = QEQ^T$$

and we have the remarkable solution that Q and E are the eigenvectors and eigenvalues of the empirical covariance matrix of derivatives labels $(1/m)\, Z^T Z$.

We can proceed to eliminate irrelevant directions by right multiplication by a filter matrix on a criterion $E_{jj} > \epsilon'$.

Hence, differential PCA is simply PCA on differential labels.

Unlike with PCA, it is safe to filter irrelevance aggressively. Assuming that a prior PCA step was performed, differentials are expressed in 'standard deviation of labels per standard deviation of inputs'. Eigenvalues E_{jj} less than 10^{-4} reflect a sensitivity less than 10^{-2}, where it takes more than 100 standard deviations in the input to produce *one* standard deviation in the label. The corresponding input can safely be discarded as irrelevant. It is therefore reasonable to set the filtering threshold ϵ' on differentials to 10^{-4} or even higher without fear of losing relevant information, whereas the PCA threshold ϵ should be near numerical zero to avoid information loss.

Note that we performed differential PCA on the *noncentral* covariance matrix of derivatives. Constant derivatives correspond to linear factors, which we must consider relevant, at least for training. In order to extract nonlinear risk factors only, we could apply the same procedure with eigenvalue decomposition of the centred covariance matrix $\frac{1}{m}\left(Z - \mu_Z\right)^T\left(Z - \mu_Z\right) = QEQ^T$ instead.

3.3 Example

In the context of the simple example of a basket option with weights a in a correlated Bachelier model, we can perform differential PCA explicitly. The input matrix X of shape $m \times n$ stacks m rows of examples $X^{(i)}$, each one a row vector of the n stock prices on a horizon date T_1. The label vector Y collects corresponding payoffs for the basket option of strike K, sampled on the same path on a later date T_2:

$$Y^{(i)} = \left(S_{T_2}^{(i)} \cdot a - K\right)^+$$

For simplicity, we skip the classic PCA step. The differential labels, in this simple example, are money indicators:

$$Z^{(i)} = \frac{\partial Y^{(i)}}{\partial X^{(i)}} = 1_{\left\{S_{T_2}^{(i)} \cdot a > K\right\}} a^T$$

with the usual notations. Differential labels are 0_n on paths finishing out of the money and a on paths finishing in the money. Denote q the empirical proportion of paths finishing in the money. Then:

$$\frac{1}{m}Z^T Z = qaa^T$$

and its eigenvalue decomposition $1/mZ^T Z = QEQ^T$ is:

$$Q = \left[\left[\frac{a}{\|a\|}\right]\begin{bmatrix}0\\ \dots\\ 0\end{bmatrix} \dots \begin{bmatrix}0\\ \dots\\ 0\end{bmatrix}\right], E = \begin{bmatrix} q\|a\|^2 & & & \\ & 0 & & \\ & & \dots & \\ & & & 0 \end{bmatrix}$$

Hence, differential PCA gives us a single relevant risk factor, exactly corresponding to the (normalized) weights in the basket.

4 A complete data preparation algorithm

We conclude with a complete data processing algorithm in pseudocode, switching to adjoint notations, i.e. we denote $\bar{X}$ what we previously denoted Z. We also use subscripts to denote processing stages.

0. Start with raw data $\underbrace{X_0}_{m\times n_0}, \underbrace{Y_0}_{m\times 1}, \underbrace{\bar{X}_0}_{m\times n_0}$.

1. Basic processing

   (a) Center inputs (but do not normalize them quite yet) with means the row vector μ_x of dimension n_0, computed across training examples:
   $$(X_1)_i = (X_0)_i - \mu_x$$

   (b) Compute standard deviation σ_y of labels across examplesand normalize labels: $Y_1 = \frac{Y_0-\mu_y}{\sigma_y}$

   (c) Update derivatives: $\bar{X}_1 = \frac{\bar{X}_0}{\sigma_y}$

   (d) Reverse for translating predictions: inputs must be normalized first, consistently with training inputs. The model returns a normalized prediction $\hat{y}_1$ and its derivatives $\hat{\bar{x}}_1$. Translate predictions back into original units with the reverse transformations:
   $$\hat{y}_0 = \mu_y + \sigma_y \hat{y}_1 \text{ and } \hat{\bar{x}}_0 + \sigma_y \hat{\bar{x}}_1$$

2. PCA

   (a) Perform eigenvalue decomposition of $\frac{X_1^T X_1}{m} = P_2 D_2 P_2^{-1}$.

   (b) Shrink the diagonal matrix D_2 to dimension $n_2 \le n_1$ by removing rows and columns corresponding to numerically zero eigenvalues. Denote F_2 the filter matrix of shape (n_1, n_2), obtained by removal of columns in the identity matrix I_{n_1} corresponding to numerically null diagonal entries of D_2. The reduced diagonal matrix is $\tilde{D}_2 = F_2^T D_2 F_2$ of size n_2, and the reduced eigenvector matrix is $\tilde{P}_2 = P_2 F_2$ of shape (n_1, n_2). Note that the eigenvalue matrix remains diagonal and the columns of the eigenvector matrix remain orthogonal and normalized after filtering.

   (c) Apply the orthonormal transformation:
   $$X_2 = X_1 \tilde{P}_2 \tilde{D}_2^{-\frac{1}{2}}$$

   (d) Update differentials
   $$\bar{X}_2 \quad = \quad \bar{X}_1 \tilde{P}_2 \tilde{D}_2^{\frac{1}{2}}$$

   (e) Note the reverse formula for prediction of derivatives: $\bar{X}_1 = \bar{X}_2 \tilde{D}_2^{-\frac{1}{2}} \tilde{P}_2^T$

3. Differential PCA

   (a) Perform eigenvalue decomposition of:
   $$\frac{\bar{X}_2^T \bar{X}_2}{m} = P_3 D_3 P_3^{-1}$$

(b) Shrink the columns of the eigenvector matrix P_3 to dimension $n_3 \leq n_2$ by removing columns corresponding to small eigenvalues. Denote F_3 the corresponding filter matrix of shape $(n_2,\, n_3)$. The reduced inverse eigenvector matrix is $\tilde{P}_3 = P_3 F_3$ of shape $(n_2,\, n_3)$.

(c) Apply the linear transformation:

$$X_3 = X_2 \tilde{P}_3$$

(d) Update differentials:

$$\bar{X}_3 = \bar{X}_2 \tilde{P}_3$$

(e) Note reverse formula for prediction:

$$\bar{X}_2 = \bar{X}_3 \tilde{P}_3^T$$

4. Train model on the dataset X_3, Y_3, $\bar{X}_3$, which is both orthonormal is terms of inputs X_3 and orthogonal in terms of directional differentials $\bar{X}_2$, with constant, redundant and irrelevant inputs and combinations filtered out.

5. Predict values and derivatives with the trained model $\hat{y} = \hat{f}\,(x)$ from raw inputs $x = x_0$:

   (a) Transform inputs:

      i. $x_1 = x_0 - \mu_x$

      ii. $x_2 = x_1 \tilde{P}_2 \tilde{D}_2^{-\frac{1}{2}}$

      iii. $x_3 = x_2 \tilde{P}_3$

   (b) predict values:

      i. $\hat{y}_3 = \hat{f}\,(x_3)$

      ii. $\hat{y}_0 = \mu_Y + \sigma_Y \hat{y}_3$

   (c) predict derivatives:

      i. $\hat{\bar{x}}_3 = \frac{\partial \hat{f}(x_3)}{\partial x_3}$

      ii. $\hat{\bar{x}}_2 = \hat{\bar{x}}_3 \tilde{P}_3^T$

      iii. $\hat{\bar{x}}_1 = \hat{\bar{x}}_2 \tilde{D}_2^{-\frac{1}{2}} \tilde{P}_2^T$

      iv. $\hat{\bar{x}}_0 = \sigma_y \hat{\bar{x}}_1$

Conclusion

We derived a complete data preparation algorithm for differential deep learning, including a standard PCA step and a differential PCA step. Standard PCA performs an orthonormal transformation of inputs and eliminates constant and redundant ones, facilitating subsequent training of neural networks. Differential PCA further rotates data to an orthogonal relevance representation and may considerably reduce dimension, in a completely safe manner, by elimination of irrelevant directions.

Like standard PCA, differential PCA is a useful algorithm on its own right, providing a low dimensional latent representation of data on orthogonal axes of relevance. In the context of financial simulations, it computes an orthogonal hierarchy of risk factors for a given transaction. For example, we proved that differential PCA identifies basket weights as the only relevant risk factor for a basket option, from simulated data alone.

We achieved this by leveraging information contained in the differential labels, which, in the context of simulated financial data, are *pathwise differentials* and contain a wealth of useful information. Recall that traditional risk reports are averages of pathwise differentials. Averaging, however, collapses information. For example, the risk report of a delta-hedged European call (obviously) returns zero delta, although the underlying stock price most definitely remains a relevant risk factor, affecting the trading book in a nonlinear manner, an information embedded in pathwise differentials but eliminated by averaging. Pathwise differentials are sensitivities of payoffs to state in a multitude of scenarios. They have a broader story to tell than aggregated risk reports.

The main proposition of the article is to leverage differentials in all sort of machine learning tasks, and we have seen their effectiveness for approximation by neural networks (main article) or classic regression (appendices). Here, we have seen that they also apply in the context of PCA, not to improve it, but to combine it with an additional procedure, which we call 'differential PCA', capable of exhibiting risk factors and safely removing irrelevant combinations of inputs. As a preprocessing step, differential PCA makes a major difference for training function approximations, reducing dimension, stabilizing nonconvex numerical optimization and reducing sensitivity to initial seed and hyperparameters like neural architecture or learning rate schedule.