Lecture Notes on
Slow Feature Analysis
- Applications -

Laurenz Wiskott
Institut für Neuroinformatik
Ruhr-Universität Bochum, Germany, EU

14 December 2016

Contents

LECTURE 1/1  1

1 Slow feature analysis (SFA) (→ slides)  1

2 Applications of SFA in machine learning  10

  2.1 Extracting driving forces (→ slides)  10

  2.2 Nonlinear blind source separation (→ slides)  12

LECTURE 1/1

1 Slow feature analysis (SFA) (→ slides)

Slow feature analysis was first published in 1998 (Wiskott, 1998). This section is based on (Wiskott and Sejnowski, 2002).

© 2008, 2009, 2016 Laurenz Wiskott (homepage https://www.ini.rub.de/PEOPLE/wiskott/). This work (except for all figures from other sources, if present) is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/. Figures from other sources have their own copyright, which is generally indicated. Do not distribute parts of these lecture notes showing figures with non-free copyrights (here usually figures I have the rights to publish but you don’t, like my own published figures). Figures I do not have the rights to publish are grayed out, but the word ’Figure’, ’Image’, or the like in the reference is often linked to a pdf. More teaching material is available at https://www.ini.rub.de/PEOPLE/wiskott/Teaching/Material/.
Slow feature analysis is an algorithm that has been developed in context of modeling the primate visual system, but it has also been applied successfully in technical contexts. It is based on the slowness principle, which is introduced here from the view of learning invariances in visual perception.
Slowness as a learning principle is based on the observation that different representations of the visual sensory input vary on different time scales. Our visual environment itself is rather stable. It varies on a time scale of seconds.

The primary sensory signal on the hand, e.g., responses of single receptors in our retina or the gray value of a single pixel of a CCD camera, vary on a faster time scale of milliseconds, simply as a consequence of the very small receptive field sizes combined with gaze changes or moving objects. As an example imagine you are looking at a quietly grazing zebra. As your eyes scan the zebra, single receptors rapidly change from black to white and back again because of the stripes of the zebra. But the scenery itself does not change much. Finally, your internal high-level representation of the environment changes on a similar time scale as the environment itself, namely on a slow time scale. The brain is somehow able to extract the slowly varying high-level representation from the quickly varying primary sensory input. The hypothesis of the slowness learning principle is that the time scale itself provides the cue for this extraction. The idea is that if the system manages to extract slowly varying features from the quickly varying sensory input, then there is a good chance that the features are a good representation of the visual environment.

A number of people have worked along these lines. Slow feature analysis is within this tradition but differs in some significant technical aspects from all previous approaches.

Figure: (Wiskott et al., 2011, Fig. 2, © CC BY 4.0, URL)
Optimization Problem

Given an input signal $x(t)$.

Find an input-output function $g(x)$ (e.g. polynomial of degree 2).

The function generates the output signal $y(t) = g(x(t))$.

This is done instantaneously.

The output signal should vary slowly, i.e. minimize $\langle \dot{y}_i^2 \rangle$.

The output signal should carry much information, i.e. $\langle y_i \rangle = 0$, $\langle y_i^2 \rangle = 1$, and $\langle y_j y_i \rangle = 0 \quad \forall j < i$.


Slow feature analysis is based on a clearcut optimization problem. The goal is to find input-output functions that extract most slowly varying features from a quickly varying input signal.

It is important that the functions are instantaneous, i.e. one time slice of the output signal is based on just one time slice of the input signal (marked in yellow). Otherwise low-pass filtering would be a valid but not particularly useful method of extracting slow output signals. Instantaneous functions also make the system fast after training, as is important in visual processing, for instance. It is also possible to take a few input time slices into account, e.g. to make the system sensitive to motion or to process scalar input signals with a fast dynamics on a short time scale. However, low-pass filtering should never be the main method by which slowness is achieved.

Without any constraints, the optimal but not very useful output signal would be constant. We thus impose the constraints of unit variance $\langle y_i^2 \rangle = 1$ and, for mathematical convenience, zero mean $\langle y_i \rangle = 0$. To make different output signal components represent different information, we impose the decorrelation constraint $\langle y_j y_i \rangle = 0$. Without this constraint, all output components would typically be the same. Notice that the constraint is asymmetric, later components have to be uncorrelated to earlier ones but not the other way around. This induces an order. The first component is the slowest possible one, the second component is the next slowest one under the constraint of being uncorrelated to the first, the third component is the next slowest one under the constraint of being uncorrelated to the first two, etc.

Figure: (Wiskott et al., 2011, Fig. 1, © CC BY 4.0, URL)
Consider a simple two-dimensional trajectory as an example. $x_1(t)$ and $x_2(t)$ are both quickly varying. Non-linearly hidden in this signal is $\sin(t)$, which is relatively slow. It can be extracted with a polynomial of degree two, since $x_1(t) - x_2(t)^2 = \sin(t)$. In the trajectory plot (right) one can see the fast back and forth rocking on a parabula, which itself is slowly moving up and down.

Figure: (Wiskott and Sejnowski, 2002, Fig. 2, © non-free, URL)

From this input signal we want to extract only a single output component, so that the optimization problem simplifies somewhat. As seen above, the slowest output signal is $y(t) = \sin(t)$ and the function required to extract it is $g(x) := x_1 - x_2^2$. Its level lines are parabulas and follow the principal curve of the input trajectory.

Figure: (Wiskott and Sejnowski, 2002, Fig. 2, © non-free, URL)
Slow Feature Analysis (SFA)

The SFA-algorithm is relatively straightforward. The upper left panel shows the trajectory $x_1(t) = \sin(t) + \cos(11t)^2$ and $x_2(t) = \cos(11t)$. It is quickly rocking back and forth on a parabola, which itself is slowly moving up and down. SFA finds this slow feature as follows:

Step 1: The input signal is nonlinearly expanded into a high-dimensional feature space $\tilde{z}$ (upper middle panel). We often use polynomials of degree two, which would result in $\tilde{z}_1 := x_1$, $\tilde{z}_2 := x_2$, $\tilde{z}_3 := x_1^2$, $\tilde{z}_4 := x_1 x_2$, and $\tilde{z}_5 := x_2^2$. Within this space the problem can be solved linearly, because any polynomial of degree two can be written as a linear combination of the $\tilde{z}_i$.

Step 2: The expanded signal is whitened or sphered (upper right panel). This operation first removes the mean and then stretches the signal along its principal axes such that it has unit variance in all directions. This has the great advantage that the constraints are easy to fulfill. If we project the whitened signal onto any unit vector, the projected signal has zero mean and unit variance; if we project the whitened signal onto any set of orthogonal unit vectors, the projected signal components are uncorrelated. Thus, we only have to find the orthogonal unit vectors that produce the slowest signal components. The constraints are then taken care of automatically. In our simple example, we only need one unit vector to project to.

Step 3: To find the direction in which the signal varies most slowly we calculate the derivative of the whitened signal (lower left panel). The variance of the derivative is small in directions of slow variation and large in directions of fast variation. The principal component with smallest eigenvalue therefore gives us the unit vector that yields the slowest possible output signal component. If we want to extract more slow features, we simply take the principal components with next larger eigenvalues. Thus, once we are here, it is easy to extract many slow output signal components.

If one concatenates the nonlinear expansion, the whitening, and the projection onto the unit vectors, one gets the nonlinear functions $g_i(x)$. The lower right panel shows the function found here for the simple example. Evaluating this function along the input trajectory yields the output signal $y(t)$ (lower middle panel).

Figure: (Wiskott and Sejnowski, 2002, Fig. 2, © non-free, URL)
**Nonlinear Expansion**

Assume we have an algorithm that solves a certain optimization problem for the class of linear functions $g_l(x)$, i.e. it finds the optimal coefficients $a_0$ to $a_2$ or $b_0$ to $b_5$ for the linear functions
g_{l0}(x) := a_0 + a_1 x_1 + a_2 x_2 \quad \text{or} \\
g_{l5}(z) := b_0 + b_1 z_1 + b_2 z_2 + b_3 z_3 + b_4 z_4 + b_5 z_5
to $g_{l5}(z)$, respectively, according to some optimization criterion.

Example: Linear regression finds the coefficients that minimize the mean squared distance between the output values $y^\mu := g_l(x^\mu)$ and some target values $s^\mu$.

Nonlinear expansion is a simple way of extending a linear method to nonlinear problems. It can generally be applied if the linear method is insensitive to any linear transformation of the input data.

If we set
$z_1 := x_1$, $z_2 := x_2$, $z_3 := x_1^2$, $z_4 := x_1 x_2$, $z_5 := x_2^2$,
and solve the linear optimization problem for
g_{l5}(z) := b_0 + b_1 z_1 + b_2 z_2 + b_3 z_3 + b_4 z_4 + b_5 z_5
then we get the optimal quadratic function for the input data $x^\mu$:
g_{q2}(x) := b_0 + b_1 x_1 + b_2 x_2 + b_4 x_1^2 + b_5 x_1 x_2 + b_6 x_2^2.

This is a simple and general technique to generalize linear algorithms to nonlinear functions.

Problem 1: The dimensionality of the expanded space can get very large.

Problem 2: Parameterized nonlinearities, such as $g(x) = a \sin(kx + \phi)$, cannot be optimized this way.

The idea of nonlinear expansion is to apply a large number of fixed nonlinear functions to the original data. This produces new input data of higher dimensionality to which the linear method can be applied. The linear combination of the expanded signal that solves the problem induces a nonlinear function that solves the problem on the original input data. One simply takes the same linear combination of the fixed nonlinear functions. Taking monomials like in the example shown here is just one possibility. Any set of nonlinear functions can be used. A problem of this technique is that the dimensionality of the expanded space can quickly grow beyond manageable size. If that happens, kernel methods might be a way to go. Another limitation is that parameterized nonlinearities can not be optimized that way.
This example illustrates the effect of nonlinear expansion. The blue and green dots on the x-axis are not linearly separable. However, if one expands the one-dimensional input space $x$ to the two-dimensional space of $x$ and $x^2$, the problem can be solved with a linear classifier. This linear function mapped back to the original input space becomes a quadratic function solving the problem.

Whitening data means to first remove the mean and then stretch the data along the principal axes such that it has unit variance in all directions. If the original data is an unisotropic Gaussian to begin with, e.g. having the shape of a flying sausage or a cigar, it has a spherical shape after whitening, which is the reason to call it also sphering.

Whitened data has the advantage that you can project it onto any unit vector and it has unit variance. Projected onto two orthogonal unit vectors, the two projected data sets are uncorrelated.
Derivative of a Trajectory

Trajectory: \( x(t) = (x_1(t), x_2(t), \ldots, x_N(t))^T \).

Derivative: \( \dot{x}(t) = (\dot{x}_1(t), \dot{x}_2(t), \ldots, \dot{x}_N(t))^T \).

Formally the derivative of a trajectory is simply the vector of the derivatives of the components of the trajectory. If the trajectory is discretized in time with step size 1, its derivative is simply the sequence of difference vectors of two successive time points.

© CC BY 4.0

An interesting property of SFA is that it can be applied in a cascade. The input signal of the example shown here has a slow feature in it that cannot be extracted with a polynomial of degree two. Thus SFA\(^2\), i.e. quadratic SFA, alone cannot solve the problem. But if one applies SFA\(^2\) again to the first three output components of the first SFA\(^2\) and then a third time, the slow feature gets extracted, compare the contour plot of \( g_{3,1}(x_1, x_2) \) with the trajectory plot of \( x_2(t) \) versus \( x_1(t) \).

Figure: (Wiskott and Sejnowski, 2002, Fig. 7, © non-free, URL)\(^7\)
2 Applications of SFA in machine learning

2.1 Extracting driving forces (→ slides)

This section is based on (Wiskott, 2003).
An iterative map is a discrete dynamical system. For the tent map \( f \) (shown in black) one starts with an arbitrary value \( w_0 \) between 0 and 1 and simply applies \( f \) to get the next value \( w_1 = f(w_0) \). Repeating this process leads to a time series \( w_t \) like the one shown below. The tent map is peculiar in that the resulting time series has no obvious structure and looks like white noise. A similar time series results if one shifts the mapping function \( f \) cyclicly within the interval \([0, 1]\) by some value \( \gamma' \) (shown in red). If \( \gamma' \) itself depends on time, it is called a driving force of the system and it changes the system dynamics over time. If \( \gamma'(t) \) changes on a slower time scale than the time series itself, SFA should be able to extract it.

The top graph shows a tent-map time series with a driving force \( \gamma(t) \) as shown in the bottom graph (solid line). In order to apply SFA, one has to consider several (in this case 10) successive values together as the input vector for SFA, a method called time embedding. In other words SFA sees a sliding window of ten points of the time series and tries to extract some slow feature from it. Applying SFA with polynomials of degree 3 results in the slow feature shown in the bottom graph as points overlayed over the solid curve of the true driving force. Mean and variance, which can principally not be extracted, are normalized to make the curves comparable. The correlation coefficient is \( r = 0.96 \).
SFA cannot only extract continuous features but also features that switch discretely between different values. This graph is the same as above except that the driving force changes in a step-like fashion. Figure: (Wiskott, 2003, Fig. 3, © CC BY 4.0, URL)

2.2 Nonlinear blind source separation (→ slides)

This section is based on (Sprekeler et al., 2014).
We consider here the problem of nonlinear blind source separation for two sources. The sources $s_1$ and $s_2$ are shown at the top left and are plotted together in the 2D scatter plot on the right. The mixing can be viewed as stretching the 2D space and winding it up in a spiral, see scatter plot below. The first source runs along the arms of the spiral; the second source runs perpendicular to it and has a very small amplitude. The resulting mixed signals are shown as $x_1$ and $x_2$ on the left. The task of nonlinear blind source separation is to extract the two sources without knowing anything about the mixture or the sources, except that they are statistically independent and smoothly varying, i.e. not white noise.

The theory of SFA formalizes two properties that can explain why SFA might be suitable to perform nonlinear blind source separation. Roughly speaking: (i) Any nonlinearly transformed version of a signal varies faster than the signal itself. (ii) Any mixture of two signals varies faster than the slower of the two signals. (These statements are true only modulo an invertible transformation of the single sources. But that is all you can hope for in any case, because the problem of nonlinear blind source separation is defined only up to an invertible transformation of the single sources.)

Figure: (Wiskott et al., 2011, Fig. 13, © CC BY 4.0, URL)
This graph illustrates the principle that a signal typically gets faster if you transform it. $s_1$ is the original signal with a $\Delta$-value of 0.063 (the $\Delta$-value measure the ‘fastness’ of a signal). $g_{11}(s)$ is an optimal invertible transformation that makes the signal slightly slower, see $h_{11} = g_{11}(s_1)$ with a $\Delta$-value of 0.060. The other noninvertible transformations shown all make the signal faster. The functions $g_{1m}$ have been found by applying SFA with high polynomials to $s_1$ directly without time embedding. $g_{11}$ is therefore optimal in yielding a slower version of $s_1$. The resulting signals $h_{1m}$ are called harmonics and play an important role in the theory of SFA.

Figure: (Wiskott & Escalante, 2010, unpubl., © CC BY 4.0)

This graph illustrates the principle that a mixture of two signals is typically faster than the slower of the two signals. Since any mixture can be expressed as a linear combination of products of harmonics of the two sources (similar to a 2D Taylor expansion), we only consider the first harmonics $h_{11}$ and $h_{21}$ of the two sources and various products of first and higher harmonics. All products are faster, i.e. have a higher $\Delta$-value, than $h_{11}$.

Figure: (Wiskott & Escalante, 2010, unpubl., © CC BY 4.0)
In xSFA (extended SFA) the mixture is first expanded into a very high-dimensional space. The slowest signal within that space is then extracted with SFA and declared the first source. Next, all harmonics of that first source are projected out of the expanded signal. Then, SFA is applied again and the slowest signal declared the second source. Next, all harmonics and all products of harmonics of first and second source are projected out of the expanded signal. Iterating this scheme can in principle extract an arbitrary number of sources (if they are in the mixture), but noise reduces performance for later sources. The graphs at the bottom show the extracted sources $y_1$ and $y_2$. As the scatter plots on the left between the extracted and the true sources show, the first source has a wrong sign (it can principally not be recovered), but otherwise the extraction is very good.

Figure: (Wiskott et al., 2011, Fig. 13, © CC BY 4.0, URL)
References


Notes


2 Wiskott et al., 2011, Scholarpedia 5(2):1362, Fig. 2, © CC BY 4.0, http://scholarpedia.org/article/Slow_feature_analysis

3 Wiskott et al., 2011, Scholarpedia 5(2):1362, Fig. 1, © CC BY 4.0, http://scholarpedia.org/article/Slow_feature_analysis


9 Wiskott et al., 2011, Scholarpedia 5(2):1362, Fig. 11, © CC BY 4.0, http://www.scholarpedia.org/article/Slow_feature_analysis

10 Wiskott, 2003, arXiv:0312317, Fig. 1, © CC BY 4.0, http://arxiv.org/abs/cond-mat/0312317/


14 Wiskott et al., 2011, Scholarpedia 5(2):1362, Fig. 13, © CC BY 4.0, http://www.scholarpedia.org/article/Slow_feature_analysis

15 Wiskott et al., 2011, Scholarpedia 5(2):1362, Fig. 13, © CC BY 4.0, http://www.scholarpedia.org/article/Slow_feature_analysis