UNIVERSITY OF TARTU

Institute of Computer Science

Computer Science Curriculum

Kaarel Kaasla

Evaluating Slow Feature Analysis on Time-Series Data

Bachelor's Thesis (9 ECTS)

Supervisor: Meelis Kull, PhD

Tartu 2021

Evaluating Slow Feature Analysis on Time-Series Data

Abstract:

In this thesis, we investigate Slow Feature Analysis (SFA) as a method of extracting slowly-varying signals from quickly-varying input data. The main aim of the thesis is two-fold. The first primary objective is to evaluate how the level of noise in input data affects the performance of SFA for different input feature combinations. The second objective of this thesis is to compare the performance of the classical formulation of SFA to a biologically plausible version of the algorithm. The first half of the thesis gives reader a theoretical overview of how the algorithm works and explores some of the previous applications. The second half conducts three experiments that explore the primary research questions of the thesis and discusses possible further research directions.

Keywords:

Machine learning, unsupervised learning, Slow Feature Analysis, neural networks

CERCS: P170 Computer science, numerical analysis, systems, control

Aeglaste tunnuste analüüsi hindamine aegrea andmetel

Lühikokkuvõte:

Käesolev bakalaureusetöö uurib aeglaste tunnuste analüüsi kui meetodit mis võimaldab eraldada kiiresti varieeruvatest sisendsignaalidest aeglaselt varieeruvaid signaale. Töö peaeesmärk on kahene. Esimeseks peamiseks eesmärgiks on hinnata kuidas müra tase sisendandmetes mõjutab aeglaste tunnuste analüüsi võimekust erinevate sisendtunnuste kombinatsioonide puhul. Töö teiseks eesmärgiks on võrrelda aeglaste tunnuste analüüsi klassikalise vormi ning bioloogiliselt inspireeritud versiooni võimekust. Töö esimene pool annab lugejale aeglaste tunnuste analüüsi algoritmi teoreetilise ülevaate ning toob välja mõningad varasemad rakendused. Töö teine pool viib läbi kolm eksperimenti mis käsitlevad töö peamiseid uurimisküsimusi ning lisaks toob välja mõned edasised võimalikud uurimissuunad.

Võtmesõnad:

Masinõpe, juhendamata õpe, aeglaste tunnuste analüüs, tehisnärvivõrgud

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1 Introduction

One of the tasks of the brain is processing primary sensory information transmitted from primary sensory organs to their corresponding primary cortex, such as the primary visual cortex for vision. However, the data that gets sent to the cortices is not entirely the same as how we perceive it; instead, our perception gets a compact representation of the raw input. For vision, the human retina has over a hundred million receptor cells, all of which contain information. This enormous amount of information contains all of the necessary information that the brain deems essential, but also a lot of unimportant noise, and the brain needs to be able to separate important from unimportant. Our human brain can quickly and reliably perform this operation of extracting meaningful information, but nevertheless, the operation's mechanisms and why it works are not yet entirely understood.

This concept can be extended from the human brain to computer vision and signal processing, where it is often necessary to extract important data from noisy input signal to make models more efficient. In the past, several dimensionality reduction models for learning and computing such representations have been proposed, probably most well-known of them being Principal Component Analysis. However, other possible approaches could be employed. One of them being Slow Feature Analysis (SFA), an algorithm that uses time-series data to learn latent features that contain important information about input [1].

Even though SFA has been around for almost two decades, the research on it is relatively scarce. The objective of this thesis is to investigate the applicability of SFA for learning invariant representations from noisy data using the classical formulation of the algorithm shown in [1], and additionally, evaluate a recently published biologically plausible version of the algorithm [2]. This thesis is divided into two broad sections. In the first section, we start by explaining the motivation of SFA through an illustrative and practical example and, after that, outline the mathematical formulation of the algorithm. We continue by describing a biologically plausible version of the same algorithm and how it differs from the classical formulation. The end of the first section also shows possible approaches to feature extraction that could be performed before applying SFA on data and also describes a few select studies where SFA has been successfully applied in the past. The second section of the thesis conducts three experiments – how adding noise to input signal affects SFA results, how feature engineering influences the algorithm's performance, and the relevancy of the biologically plausible version of the algorithm compared to the classical formulation. Lastly, the thesis offers suggestions for further research directions and related areas worth exploring.

2 Slow Feature Analysis

2.1 Motivation

2.1.1 Types of Learning

Computational learning theory distinguishes three broad forms of learning: supervised, reinforcement, and unsupervised learning. The premise is the same for all three where a model receives input signal $\mathbf{x}$ and produces output signal $\mathbf{y}$ which depends on not only the input data but also some parameters $\mathbf{w}$ that control the behavior of the system [3]. The model then learns by adapting the parameters $\mathbf{w}$ by some optimization criteria. For supervised learning, the training data consist not only of the input data $\mathbf{x}$ but also of so-called target labels which tell a learning model which output $\mathbf{y}$ it should produce for a given input. The objective of a supervised learning model is to adapt $\mathbf{w}$ so that the model reproduces the expected output as close as possible to the actual labels. Reinforcement learning differs from supervised learning in a way that the performance of a model is still evaluated, but with a difference that instead of providing the correct output, the feedback is limited to periodically telling whether the output was in the correct direction (reward) or incorrect direction (punishment) and the main objective is to maximize the overall reward gained through adaption of the $\mathbf{w}$ coefficients. In unsupervised learning, data contains no real labels or reward-punishment signal, but instead, a model possesses some internal guideline that governs adaption of $\mathbf{w}$ to the features of the input data.

2.1.2 A Slow Feature Analysis Example

As an illustrative example about unsupervised learning and more specifically Slow Feature Analysis (SFA) algorithm, let us create an artificial example where we want to know the estimated hourly air temperature measurements for the past 24 hours. However, we do not have access to the actual temperature measurements, but instead have two devices that output values that are somewhat able to capture the overall correct trend, meaning that these correlate with the actual signal but have errors in the measurements and the scales do not correspond to the actual scale. Drawing comparisons to real life, a similar situation might arise if we are weighting ourselves on an uncalibrated weighing scale. In that case, the measurements likely include a systematic error that leads to a wrong scale of measurements and also a random error caused by standing on a particular position on the scale. Visually, the measurements of this particular example about temperatures are shown in Figures 1 and 2.

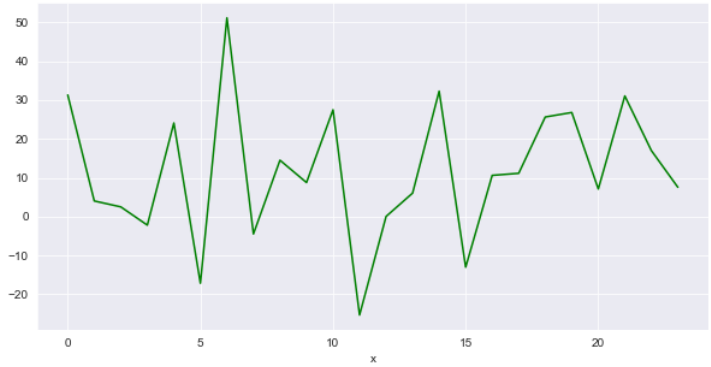

Figure 1: Temperature measurements of the first device.

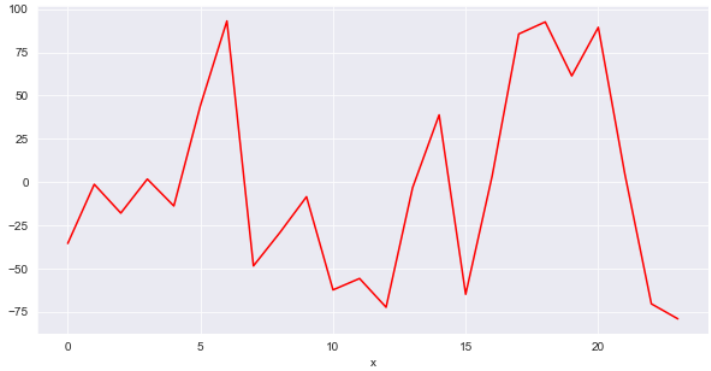

Figure 2: Temperature measurements of the second device.

In Figures 1 and 2, the data consists of measured values for both devices over the last 24 hours in hourly intervals. As this example is artificially generated, the input signals are created from the true signal, and a detailed explanation of the generation process is outlined in Chapter 3. We could ask a question, that given the device measurement data, can we approximate the shape of the real measurements from these? We obviously cannot know the true scale of the data, which makes it impossible to approximate the original measurements, and additionally, both of the input signals have differing scales, which makes approximating the shape of the real signal even more difficult. What we can do in such a situation is to transform both of the input signals so that they both have zero mean and unit variance, thus bringing them onto the same scale, and the approximation can be made on that. One of the most common approaches to scale data is standardization, also known as z-score transformation. This technique takes data,

subtracts its mean from it, and divides it by standard deviation. More formally,

$$\mathbf{z} = \frac{\mathbf{x} - \bar{x}}{\sigma} \tag{1}$$

where $\mathbf{x}$ is a sample as a vector, $\bar{x}$ is the mean value of that sample, σ is the standard deviation of the sample, and $\mathbf{z}$ is the z-score transformed vector of the initial vector $\mathbf{x}$. This transformation ensures that the resulting transformed data has zero mean and unit variance. We can then do the same for the second signal, and after z-score transforming both measurements, the resulting data is depicted in Figure 3.

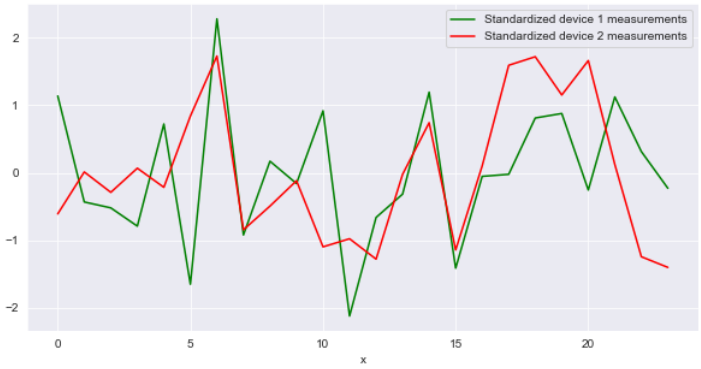

Figure 3: Measurements of the first and second device after applying the z-score transformation.

In Figure 3, both of the device measurements are now on the same scale, having zero mean and unit variance. What the y-axis on Figure 3 shows is how far is some value from the mean. For example, suppose a measurement has a value of 1. In that case, its value is 1 standard deviation above the signal's mean; similarly, if a value is -0.8, it is 0.8 standard deviations below the mean, and so on.

As can be seen from Figure 3, after z-score transforming the data, between the first and second time-step, the first device's measurement goes down and the second device's up, meaning that there is noise in the measurement data. There are other similar instances where both signals move in opposite directions due to the noise. In a situation like this, where two signals move in opposite directions, it seems reasonable to take an average of the signals as it would somewhat even out these opposite data points. To take an average, we can multiply both signals by 0.5 and combine the resulting signals by addition. This can be written $\mathbf{y} = 0.5 \cdot \mathbf{x}_1 + 0.5 \cdot \mathbf{x}_2$ where vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are first and second device's measurement respectively and $\mathbf{y}$ the resulting output vector. Visually, the result of averaging with both weights of 0.5 would look as shown in Figure 4.

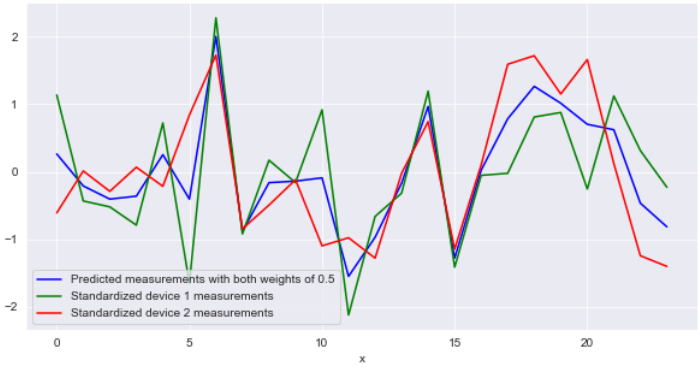

Figure 4: Z-score transformed input signals and an output signal created by weighting both input signals by 0.5.

This resulting signal is less noisy than the original measurements, evening the noise out at the time-steps where the measurements are going in the opposite directions. However, it seems from Figures 1 and 2 that both measurements do not have the same amount of noise; the first device seems to be noisier than the second as it moves more rapidly. In this case, it is not the best approach to add even weighting to the measurements but rather inversely proportional to the level of relative noise, meaning that higher the noise, the lower the weight should be. We could make an educated guess and say that it would be a better fit if the first device had a weight of 0.25 and the second 0.75, or more formally $\mathbf{y} = 0.25 \cdot \mathbf{x}_1 + 0.75 \cdot \mathbf{x}_2$. The result of doing so is shown in Figure 5.

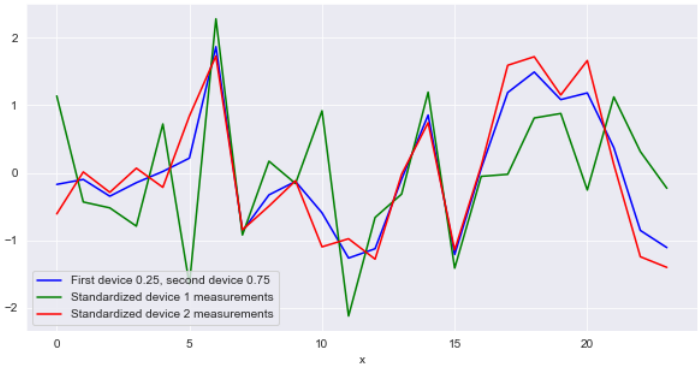

Figure 5: Z-score transformed input signals and an output signal created by weighting first device's signal by 0.25 and the second's 0.75.

The result seems to be even less noisy than before. It has become evident that if we want

to approximate the shape of the actual signal, we could keep trying different combinations of the weights. However, this raises a question: how do we measure whether one output signal is better than another, meaning for what criteria should we optimize? We cannot measure the difference between the true signal and the predicted output since we do not know the true signal. For this reason, the best we can do is make an assumption about the true measurements and optimize for that.

One such assumption we could make about the actual signal, and thus the signal we want to approximate using SFA, is that it moves slowly. By the term "slowly," we mean that the signal's measurements do not change much between adjacent time-steps. Assuming that is the case, we want to measure the differences between adjacent time-steps of an output signal and find such weights for calculating it that give the lowest average difference. More formally, we could write that for $\mathbf{y} = (y_0, y_1, ..., y_T)$ we want to find weights $\mathbf{w} = (w_1, w_2)$ that satisfy the following condition.

$$\underset{(w_1,w_2)}{\operatorname{argmin}} \quad \frac{1}{T}\sum_{t=1}^{T}(y_t - y_{t-1})^2 \tag{2}$$

where

$$y_t = w_1 x_{1,t} + w_2 x_{2,t} \tag{3}$$

with t being t-th time-step of input data.

Coming back to the previous two examples, with in the first case both inputs having the weights of 0.5 and then the first device having the weight of 0.25 and the second having 0.75, the slowness parameters for the output signals are 1.77 and 1.25 respectively. However, as we look for the weights of the slowest possible signal achievable if the weights are real numbers, it is a constant response where both of the weights are zero, and the difference between any two adjacent points is zero. It would not make any sense for an actual signal to be constant, and it would eliminate the need for methods that attempt to approximate the original signal. Therefore, we want the predicted output signal to have non-zero variance. Non-zero variance can be assured by fixing the variances of both input measurements to a specific number. There is no specific requirement what the variance needs to be as it can be any non-zero positive value, but as a general rule, variance is chosen to be 1, also known as unit variance. We can now add the variance constraint to the equation (2) so it is

$$\text{subject to } \frac{1}{T+1}\sum_{t=0}^{T}\left(y_t - \frac{1}{K+1}\sum_{k=0}^{K} y_k\right)^2 = 1 \quad \text{(unit variance)}. \tag{4}$$

As we are looking for the slowest possible non-zero variance signal, it also needs to be unique, meaning that there exists a slowest signal, not, for example, multiple slowest

signals with the same slowness value. To show that there can be multiple signals with the same slowness value, let us first take the data presented in Figures 1 and 2 and find the output signal by weighting both features by 0.5, but this time, not applying the z-score transformation. Now, let us take the same data and repeat the process, but this time, we add 100 to each of the first device's measurements. Plotting the results, we get Figure 6.

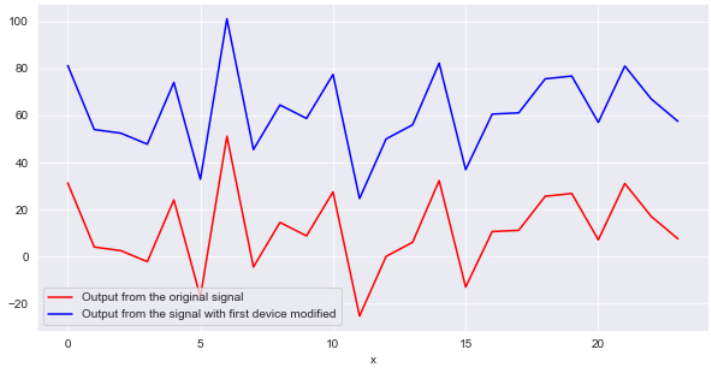

Figure 6: Output signals of even 0.5 weighting where the red signal is data from Figures 1 and 2 and the blue line has 100 added to each of the first device's measurements.

If we calculate the slowness of both of these signals, we get the same number in both cases: 29969.97. As can be seen, we can shift features, and the solution would be different, but slowness the same as without adding the constant, thus leading to an infinite number of equally slow outputs. The non-uniqueness of a solution occurs when a mean is not fixed. As with variance, the mean could be fixed to any positive real number, but as a general principle, the mean is fixed to zero. With zero mean, even if a constant is added to all features, after standardization, the values are the same as without adding a constant, leading to the problem having a unique solution. As with unit variance, we can define the constraint that the input must have zero mean and the equation (2) is

$$\text{subject to } \frac{1}{T+1}\sum_{t=0}^{T} y_t = 0 \quad \text{(zero mean)}. \tag{5}$$

Since mean must be zero, we can re-write the equation (4) to take that into account. With zero mean, the equation (2) is now

$$\text{subject to } \frac{1}{T+1}\sum_{t=0}^{T} y_t^2 = 1. \tag{6}$$

Additionally, it should be noted that the z-score transformation pre-processing step earlier did just what the constraints (5) and (6) require.

Previously, we tried two different versions, one where both weights were equal to 0.5 and one where the first device's weight was 0.25 and the second's 0.75. We also found the slowness for both versions, but we are looking for the slowest signal over all possibilities. One possibility would be to keep iterating over different values, measuring the average change between adjacent points, and trying to get as low as possible. However, this approach is inefficient as the weights are real numbers, so there exists an infinite number of possible combinations. Therefore, we can never be sure whether we have actually found values that yield the slowest signal or simply the slowest for some particular search space. Another, more efficient option would be to give the input data to some specific algorithm with a guideline that the average change between adjacent points needs to be as low as possible but non-constant and let it find the weights that satisfy the criteria. Viewing this task through the general framework of computational learning theory, it is an optimization problem where we have the input $\mathbf{x}$ which is the two noisy measurements and are looking for parameters $\mathbf{w}$ such that when multiplying $\mathbf{x}$ and $\mathbf{w}$ the resulting output $\mathbf{y}$ changes as slowly as possible.

After letting SFA solve the problem, we end up with the weight of -0.09174359 for the first device's signal, 1.00951226 for the second device's, and the slowness of 0.98. Though we likely do not have access to the actual measurements in real life, for the sake of the example, the resulting output signal with the true z-score-transformed values looks as shown in Figure 7.

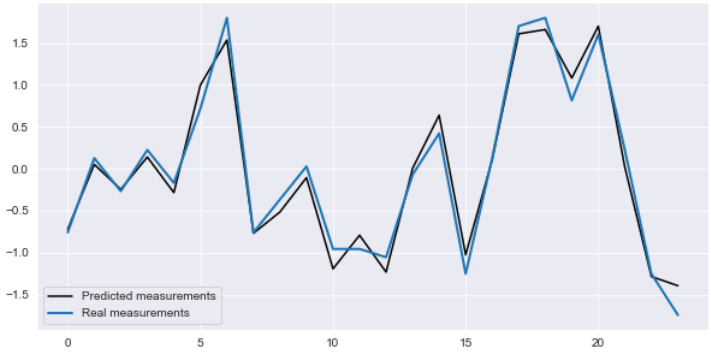

Figure 7: Comparison of the predicted and the true measurements.

As can be seen from Figure 7, even with 24 points of data, we can still recover a signal that approximates the true values relatively closely. Knowing the true measurements,

we can also calculate how far the predicted signal is from the actual. One possibility to measure that is to use Root Mean Square Error (RMSE) which finds the difference as follows where $\mathbf{S}$ denotes the true signal and $\mathbf{y}$ is the output signal predicted by SFA where S_t and y_t are t-th elements of the respective signals.

$$RMSE = \sqrt{\Big(\sum_{t=0}^{T} \frac{(y_t - S_t)^2}{T+1}\Big)} \tag{7}$$

Measuring the RMSE for the predicted signal, the result comes to be 0.17, which can be interpreted as that on average, the predicted signal differs from the true signal by 0.17 distance units. Similarly, we can calculate RMSE for the previous scenario with even weights and also the case where the first signal had 0.25 weight and the second signal had 0.75 weight. In the first case, the RMSE is 0.61, and for the second, 0.31.

2.1.3 The Slowness Principle

SFA employed in the previous example is an unsupervised learning method that uses noisy, high-dimensional time-series data to learn latent features, first proposed by Wiskott and Sejnowski [1]. We tried to recover the slowest signal in the previous example because the theoretical foundation of SFA is motivated by the slowness principle, which postulates that the essential features of data such as the identity of an object typically change on a much slower time-scale than the quickly-varying primary sensory signals. As exemplified by Goodfellow et al. [3] we can imagine observing a running zebra. The signal perceived by a receptor cell in the retina would quickly switch between black and white due to the zebra's stripes. However, from this rapidly changing signal, the human brain manages to extract the aspect that there exists a zebra in the scene. Once the zebra is present in a scene, the feature indicating that persists at least for a few seconds, so the representation of a zebra changes on a much slower time-scale than the signals received by the receptor cells. What this means is that the slowly varying features are likely to be more relevant than the ones that change more quickly, and by being able to extract slowly changing features from a noisy input signal, we can identify the underlying driving forces in the data as we did in the Chapter 2.1.2. Based on this hypothesis, we can formulate the slowness principle - given the input data, learn the model parameters such that the resulting output signals vary as slowly as possible.

2.2 The Optimization Problem

As previously mentioned, SFA aims to minimize the temporal slowness of the signal as shown in the equation (3) while being constrained by the equations (5) and (6). That, however, only works for cases where we approximate one output signal, as in Chapter 2.1.2. If we want to output multiple signals, we also need to add a constraint for the

learned features not to be linearly correlated. If the learned features were not linearly decorrelated from each other, all of them would capture the same signal - the slowest one. Adding this additional constraint to the optimization problem, we ensure that the algorithm not only outputs the slowest possible signal but also, if trying to recover more than one signal, the signals are separate from one another.

This leads to the SFA optimization problem that works for an arbitrary number of output signals that was formalized by Wiskott and Sejnowski as follows [1]:

> Given a finite-dimensional function space $\mathcal{F}$ and I-dimensional time-series input signal $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_I]^T$, where $\mathbf{X}$ is a matrix of input signals, find an input-output function $\mathbf{g}(\mathbf{X}) = [g_1(\mathbf{X}), ..., g_J(\mathbf{X})]^T$ generating the J-dimensional output signal $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_J]^T$ with $\mathbf{y}_j := g_j(\mathbf{X}), j \in \{1, ..., J\}$, such that
>
> $$\underset{\mathbf{g}\in\mathcal{F}}{\operatorname{argmin}} \quad \sum_{j=1}^{J} \frac{1}{T} \sum_{t=1}^{T} (y_{j,t} - y_{j,t-1})^2 \tag{8}$$
>
> under the constraints
>
> $$\frac{1}{T+1} \sum_{t=0}^{T} y_{j,t} = 0 \quad \text{(zero mean)}, \tag{9}$$
>
> $$\frac{1}{T+1} \sum_{t=0}^{T} y_{j,t}^2 = 1 \quad \text{(unit variance)}, \tag{10}$$
>
> $$\forall j' < j : \frac{1}{T+1} \sum_{t=0}^{T} (y_{j',t})(y_{j,t}) = 0 \quad \text{(decorrelation and order)}. \tag{11}$$

Since we are operating in a finite-dimensional function space $\mathcal{F}$, we can consider only linear combinations of the basis functions in the optimization step and thus reduce the optimization problem to a generalized eigenvalue problem [4] which simplifies it significantly and can be computed using a computationally efficient algorithm.

2.3 The Algorithm

As mentioned, if confining to a finite-dimensional function space, we can optimize over the coefficients of the basis of the function space. Lets assume that the function space $\mathcal{F}$ has dimension K and is finite-dimensional. It is spanned by a set of basis functions f_i

which can be arranged in a vector-valued function $\mathbf{z}(\mathbf{X}) = [f_1(\mathbf{X}), ..., f_K(\mathbf{X})]^T$. Using that, we can write any function $g \in \mathcal{F}$ as a weighted sum of these bases

$$\mathbf{y}_j = g_j(\mathbf{X}) = \mathbf{w}_j^T \mathbf{z}(\mathbf{X}) \tag{12}$$

where $\mathbf{w}_j = [w_{j,1}, ..., w_{j,K}]T$ is some transposed weight vector and $\mathbf{w}_j^T \mathbf{z}(\mathbf{X})$ is matrix multiplication of the weight vector and the signals. Lets assume that the signals $\mathbf{z}$ have zero mean. Now, we can write the equations (8) and (10) for each $\mathbf{y}_j$ as follows

$$\frac{1}{T}\sum_{t=1}^{T}(y_{j,t} - y_{j,t-1})^2 = \langle \dot{\mathbf{y}}_j^2 \rangle = \mathbf{w}_j^T \langle \dot{\mathbf{z}}\dot{\mathbf{z}}^T \rangle \mathbf{w}_j = \mathbf{w}_j^T \mathbf{A} \mathbf{w}_j \tag{13}$$

$$\frac{1}{T+1}\sum_{t=0}^{T} y_{j,t}^2 = \langle \mathbf{y}_j^2 \rangle = \mathbf{w}_j^T \langle \mathbf{z}\mathbf{z}^T \rangle \mathbf{w}_j = \mathbf{w}_j^T \mathbf{B} \mathbf{w}_j \tag{14}$$

Here the angle brackets $\langle \cdot \rangle$ indicate temporal averaging which is simply a more concise version of the summation shown in Chapter 2.2 and $\dot{y} = y_t - y_{t-1}$. Additionally, $\mathbf{A} = \langle \dot{\mathbf{z}}\dot{\mathbf{z}}^T \rangle$ is a matrix of seconds moments of signal's $\mathbf{z}$ time derivative and $\mathbf{B} = \langle \mathbf{z}\mathbf{z}^T \rangle$ is the covariance matrix of $\mathbf{z}$.

SFA aims to minimize its mean square of the temporal derivative under the constraint of unit variance [5]. One possible technique to solve such constrained optimization problem is the Lagrange multipliers. We can search for stationary points (where the derivative with the respect to the parameters vanishes) of the objective function

$$\mathcal{L}(\mathbf{w}) = \langle \dot{\mathbf{y}}^2 \rangle - \lambda \langle \mathbf{y}^2 \rangle = \mathbf{w}_j^T \mathbf{A} \mathbf{w}_j - \lambda \mathbf{w}_j^T \mathbf{B} \mathbf{w}_j \tag{15}$$

where λ is some value of the Lagrange multiplier such that the optima fulfills the necessary constraint. If we calculate the gradient of $\mathcal{L}$ and set it to 0, from above, we get the generalized eigenvalue problem

$$\mathbf{A}\mathbf{W} = \lambda \mathbf{B}\mathbf{W}. \tag{16}$$

The generalized eigenvalue problem has a matrix $\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_J]$ of eigenvectors and eigenvalues $\lambda_1, ..\lambda_J$. Sorting the eigenvectors ascendingly by their eigenvalues, the functions $g_j(\mathbf{x}) = \mathbf{w}_j^T \mathbf{z}(\mathbf{X})$ are solutions to the SFA optimization problem, the slowness value is given by the eigenvalue λ_j, and the constraints (9)-(11) are fulfilled.

2.4 Biological Slow Feature Analysis

As was shown in Chapter 2.1.3, the method is inspired by brain function. However, as formulated in [1], the way the algorithm works does not correspond to how the brain

operates. However, in [2] the authors published a SFA algorithm with what they call a biologically plausible neural network implementation which is likely to be more similar to the actual brain function than the previous approaches. The authors defined biologically plausible as "operates in the online setting and can be mapped onto a neural network with local synaptic updates" [2, p. 1]. In this case, online setting means that after receiving a single input, the model computes its output before receiving the next input.

In contrast, the standard SFA formulation of [1] does the calculations on the whole dataset at once. Local synaptic updates in terms of biological plausibility mean that the network weight updates depend only on immediately preceding and succeeding neurons. These two conditions lead to a model which is both more biologically plausible and has a lower computational overhead compared to the classic formulation. There have been previous attempts at creating a more biologically plausible version of the algorithm than [1], none have found a model that satisfies both of the requirements. For example, Kompella et al. [6] proposed a version called Incremental SFA, an online algorithm for SFA, but it relies on non-local learning rules that do not meet the above biological plausibility criteria.

Similarly, Malik et al. [7] created a model that operates in an online setting and can be implemented in a biologically plausible neural network. However, the model is limited because it cannot find multi-dimensional projections of output in the online mode, but it works only for one-dimensional output signals. It is necessary for an algorithm to be able to output arbitrary dimensionality as long as the output dimensionality is smaller than the input signal. The model, what [2] call Biological Slow Feature Analysis (bio-SFA), satisfies both the criteria of online setting and local learning rules while being able to output multiple signals and could therefore be viewed as the first proper biologically plausible implementation of the algorithm. The architecture of the bio-SFA network is depicted in the following Figure 8.

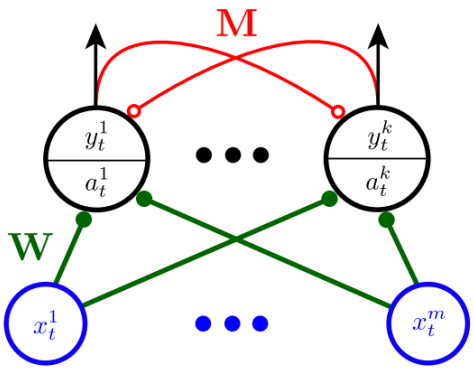

Figure 8: Architecture of the biologically plausible SFA network as proposed in [2].

In Figure 8, $\mathbf{x}_t$ is the input signal of a single time-step, both $\mathbf{W}$ and $\mathbf{M}$ are weights, $\mathbf{a}_t$ are the activation functions, and $\mathbf{y}_t$ the output signals. Implementation-wise, the algorithm takes the signal $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$, the output signal dimension k, and learning rates γ, τ, η. Additionally, before running the computations, the algorithm initializes matrices $\mathbf{W}$ and $\mathbf{M}$ where $\mathbf{W}$ has the dimensions $m \times k$ and $\mathbf{M}$ is positive definite such as an identity matrix with dimensions $k \times k$. The learning rate γ controls the activation function. The parameter $\eta > 0$ is the ratio of the learning rates of $\mathbf{W}$ and $\mathbf{M}$ and $\tau \in (0, \eta)$ is the learning rate for $\mathbf{W}$. The constraint that $\eta < \tau$ is necessary that the matrix $\mathbf{M}$ remains positive definite if it is initialized as a positive definite matrix [2].

After the initialization, the m-dimensional signal is streamed one sample at a time. A sample is multiplied by the weight matrix $\mathbf{W}$ and the resulting product $\mathbf{a}_t = \mathbf{W}\mathbf{x}_t$ is the projection of the input signal onto the k-dimensional subspace. This is then followed by running the fast recurrent neural dynamics using the Euler's method to convergence. The authors of the paper state that these dynamics equilibriate at $\mathbf{y}_t = \mathbf{M}^{-1}\mathbf{a}_t$. This results in $\mathbf{y}_t \leftarrow \mathbf{y}_t + \gamma(\mathbf{a}_t - \mathbf{M}\mathbf{y}_t)$. This operation is repeated until the convergence criteria is fulfilled. There is no set convergence criteria, but usually, the operation is repeated until $(\mathbf{a}_t - \mathbf{M}\mathbf{y}_t)$ is smaller than some predetermined threshold. After the output has converged, the weight update for $\mathbf{W}$ is $\mathbf{W} \leftarrow \mathbf{W} + 2\eta(\bar{\mathbf{y}}_t\bar{\mathbf{x}}_t^T - \mathbf{a}_t\mathbf{x}_t^T)$. Similarly for $\mathbf{M}$, the update is $\mathbf{M} \leftarrow \mathbf{M} + \frac{\eta}{\tau}(\bar{\mathbf{y}}_t\bar{\mathbf{y}}_t^T - \mathbf{M})$. For both updates, $\bar{\mathbf{x}}_t = \mathbf{x}_t + \mathbf{x}_{t-1}$ and $\bar{\mathbf{y}}_t = \mathbf{y}_t + \mathbf{y}_{t-1}$ which is the delayed sum of the inputs and outputs respectively. In the $\mathbf{W}$ update the second term $\mathbf{a}_t\mathbf{x}_t^T$ whitens the inputs in the network and in the $\mathbf{M}$ update $\mathbf{M}$ ensures that the projections are decorrelated. The whole algorithm could be written as shown in Figure 9.

Algorithm 1: Bio-SFA

input expanded signal $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T\}$; dimension k; parameters γ, η, τ
initialize matrix $\mathbf{W}$ and positive definite matrix $\mathbf{M}$
for $t = 1, 2, \ldots, T$ **do**
  $\mathbf{a}_t \leftarrow \mathbf{W}\mathbf{x}_t$ ▷ project inputs
  repeat
    $\mathbf{y}_t \leftarrow \mathbf{y}_t + \gamma(\mathbf{a}_t - \mathbf{M}\mathbf{y}_t)$ ▷ compute neural output
  until convergence
  $\bar{\mathbf{x}}_t \leftarrow \mathbf{x}_t + \mathbf{x}_{t-1}$
  $\bar{\mathbf{y}}_t \leftarrow \mathbf{y}_t + \mathbf{y}_{t-1}$
  $\mathbf{W} \leftarrow \mathbf{W} + 2\eta(\bar{\mathbf{y}}_t\bar{\mathbf{x}}_t^\top - \mathbf{a}_t\mathbf{x}_t^\top)$ ▷ synaptic updates
  $\mathbf{M} \leftarrow \mathbf{M} + \frac{\eta}{\tau}(\bar{\mathbf{y}}_t\bar{\mathbf{y}}_t^\top - \mathbf{M})$
end for

Figure 9: Algorithm of the biologically plausible SFA network as proposed in [2].

After the network has repeated the above algorithm for all time-steps in the data, the matrices $\mathbf{W}$ and the inverse of $\mathbf{M}$ can be used to calculate the output for new samples. Formally, this can be written $\mathbf{Y} = \mathbf{M}^{-1}\mathbf{W}\mathbf{X}$ which corresponds to the slowest possible projection. To evaluate the slowness of this projection, we can measure it against the SFA closed-form solution, which is guaranteed to give the slowest possible projection for a given input signal.

2.5 Feature Engineering

The earlier example of SFA had two noisy signals as its input. However, often in real applications, we do not have multiple input signals but rather a single noisy input. In that case, finding some weight would not be much of use because it would simply vertically flatten or stretch the input signal. Instead, what is usually done in such situations is increasing the dimensionality by creating new features from the original input signal. Although there are no set rules for increasing dimensionality, there are two main approaches used, often in conjunction.

The first is adding time-shifted copies of the original signal. For example, if adding three time-delayed copies, every datapoint would have its original value at time t that existed before and additionally the signal values at $t-1$, $t-2$, and $t-3$. Each data point now has four dimensions, making it easier to find the weights, which minimize the slowness. There are no concrete requirements about how many time-shifted dimensions should be added to a signal, but generally, the number is kept small, around $1-4$, not to run into possible curse of dimensionality problems.

The second approach is applying polynomial expansion to input data to create new di-

mensions. For example, if the input data is 4-dimensional with dimensions a, b, c, d, applying quadratic expansion to each 4-dimensional datapoint yields a new 14-dimensional datapoint with dimensions t, t^2, tu for distinct $t, u \in \{a, b, c, d\}$. Similarly, if applying a cubic expansion to the same signal, each new point would be 34-dimensional with $t, t^2, t^3, tu, tuv, t^2u$ for distinct $t, u, v \in \{a, b, c, d\}$. Generally, quadratic expansion is used as the cubic version can already run into the curse of dimensionality problems. The polynomials higher than cubic are virtually never used as these expansions would yield too high dimensionalities to be of practical use.

2.6 Applications

SFA was first presented in the 2002 article "Slow feature analysis: Unsupervised learning of invariances" by Laurenz Wiskott and Terrence Sejnowski [1] which has since become the seminal paper on the topic. In addition to the theoretical framework of why and how the method works, the article also provided examples of experiments conducted on synthetic data to showcase its capabilities. However, the real-world use-cases of the method remained unsubstantiated. In the following years, experiments involving SFA and real-world applications have been published. However, both the method and the slowness principle, in general, have remained relatively niche and have not been really used as a basis for any state-of-the-art applications [3]. Nevertheless, it is worth exploring some select problems SFA has been applied to in the past.

2.6.1 Blind Source Separation

One of the main applications for SFA has been blind source separation which is essentially a problem of extracting source signals from unknown time series where signals have been mixed together. A didactic example that is often given is the so-called "cocktail party problem," where several people are in a room talking simultaneously, and each of them is being recorded by the same number of microphones. As the result, each of the microphones has a different mixture of all the people in the room. From these signals, it is theoretically possible to isolate each person's speech, assuming that it is known how the signals are mixed together. Suppose the source mixtures are linear and the sources are independent of each other. In that case, statistical independence can be chosen as an optimization criterion, and independent component analysis (ICA) has been found to be an appropriate algorithm [8].

However, if the relationship between the mixtures and sources is not linear, ICA is not sufficient anymore [9]. Unless the source signals are independent and identically distributed random variables which also have a temporal structure, separation of source signals can still be achieved using SFA [10]. One of the reasons SFA tends to work in this case is that if a signal gets distorted non-linearly, the resulting signal generally

varies more quickly. Since the objective of SFA is to extract latent signals from quickly-varying data, it would prefer the original source signals over the distorted versions. Additionally, if mixing two different signals together, the mixture is quicker-varying than the slower of the original source signals, and this helps SFA separate the sources [9]. Assuming that a correct non-linear expansion is used and if the original source signals are somewhere in the expanded space, these two properties allow SFA to find the slowest signal of the mixture. This slowest signal can then be removed from the mixture, leaving a mixture of the remaining signals. From there, we can iteratively keep removing the slowest signals until all of them have been separated.

In the above application, SFA was used as an unsupervised algorithm. However, it can also be extended to supervised learning, namely for classification and regression problems.

2.6.2 MNIST Digit Prediction

The first known application for classification was on the MNIST hand-written digits dataset [11]. Though SFA was initially conceived as a method to learn latent features of time series, the study showed that even if the input data does not have a temporal structure, it is still possible to use SFA by reformulating the algorithm. The paper constructed a set of time series, each with two elements randomly chosen from the same class, and then SFA with polynomials of degree 3 was trained on the collection of these sequences. As a result, patterns that corresponded to the same class would cluster in the feature space, which, in turn, allowed for applying a Gaussian classifier. The method's error rate was 1.5%, which is comparable to the error rate of 0.95% achieved by LeNet-5, a special-purpose model designed for hand-written character recognition. In later studies, the same reformulated SFA algorithm has also been applied to human gesture recognition [12] and monocular road segmentation [13].

2.6.3 Age And Gender Estimation

For regression problems, one of the bigger but simultaneously more challenging SFA applications has been age and gender estimation from image data. One such study [14] used artificially and randomly generated frontal face images with different age, gender, race, and identity parameters for age estimation. It is worth noting that the gender variable was not binary, but instead a continuous feature on a scale from -3 (very feminine) to $+3$ (very masculine). The resulting 135×135-pixel images were ordered along both age and gender dimensions so that for both sequences, the relevant parameter changed as slowly as possible. After applying both linear and non-linear SFA to the sequences of the images, the three slowest resulting output signals were used to train a Gaussian classifier and closest center classifier for the age and gender estimation.

As for the results, a linear network with a Gaussian classifier had the best performance. For age, the algorithm gave the root-mean-square error of 3.8 years, which is relatively close to the ground truth, and for gender, 0.33 units.

3 Experiments

The experiments conducted in this Chapter attempt to gain insights into three main research questions. The first question tackles the situation if the input data has variable amount of noise, how does this level of relative noise in data affect the error in the recovered signal? The second question builds on the first and explores what effect applying feature engineering has on the output error if again noise varies? The last question aims to explore the applicability of the biological SFA. More specifically, could bio-SFA be considered a viable alternative to the analytical solution? Additionally, could bio-SFA be used for applications where the analytical approach cannot really be used, such as live systems where the data is streamed into a model?

The data used for the experiments of this thesis is from the United States National Oceanic and Atmospheric Administration's Climate Reference Network's climate monitoring database. This agency has made available the climate data for their weather stations, and the following experiments make use of both the hourly and sub-hourly 5-minute interval datasets from the year 2020 for the Yosemite station in California. The choice of year and the data was rather arbitrary without any particular analytical reason.

The way all of the experiments are set up is similar to how the example in Chapter 2.1.2 was created. More specifically, first, we scale the original measurements to two different new scales. How the scaling is done is as follows.

$$x_{normalized} = (b - a) \times \frac{x - min(x)}{max(x) - min(x)} + a. \tag{17}$$

Above, a, b are the lower and upper bound of the range we want to convert our data to. Similarly $min(x)$ and $max(x)$ are the current minimum and maximum values of the data. After scaling the true measurements, we add Gaussian noise to the newly generated features, and finally, we try to approximate the initial signal from these artificially generated noisy signals.

3.1 Effect of Relative Noise on Error

This experiment uses the dataset with hourly intervals, and after manually cleaning the data of obvious outliers such as the temperature of -9999 degrees, the length of the dataset comes out to be 8771 samples. In the present case, the first input is on the scale $0-15$, and the second is between -1 and 1. As for the experiment, it consists of running the analytical version of SFA for 100 iterations with different noise levels between 10% and 100% relative noise in 10% intervals for both signals. Relative noise means that the level of noise is dependent on the signal's variance. For example, for the first signal that

is scaled between 0 and 15, 10% of variance would be 1.5. However, for the other input signal, 10% would be 0.2. It is important to use relative levels and not absolute because, as can be seen, if scales are different, simply adding some value to both signals would likely have very different effects depending on the scale. After running the algorithm with simply two aforementioned inputs for different noise levels, the results look as shown in Figure 10.

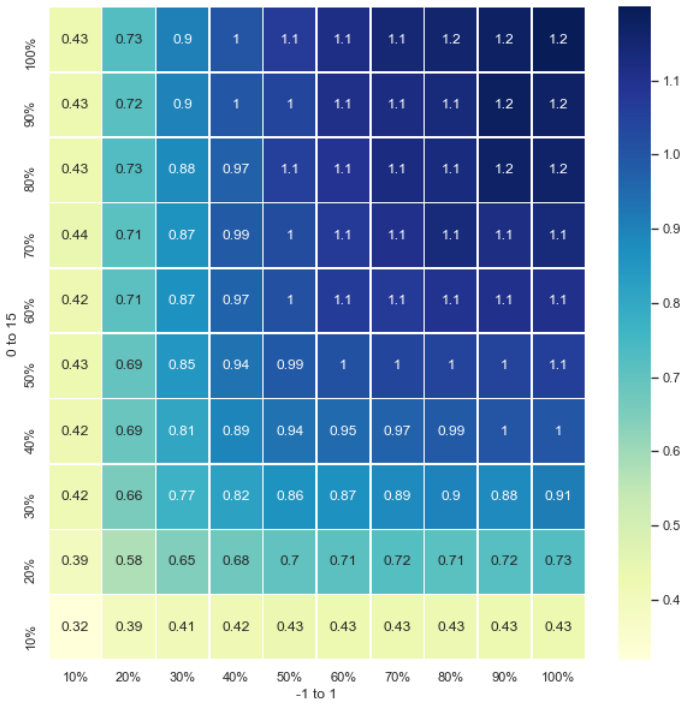

Figure 10: Effect of relative noise on RMSE over relative variance levels if the first signal is scaled to $0 - 15$ and the second is scaled -1 to 1. Numbers in the cells denote RMSE for that particular combination.

How to read these results is that, for example, if both scaled versions of the true signal have 10% of relative noise added to them, the root-mean-square error between the true signal and the predicted output is 0.32 units. Likewise, if the signal which is scaled $0 - 15$ has 10% added variance and the other signal has 20%, the RMSE is now 0.39. By plotting the instances where in the first case both signals have 10% of relative noise and in another both have 100%, we get following Figures 11 and 12.

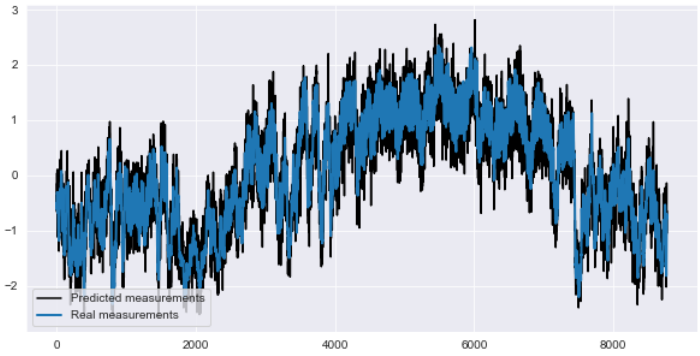

Figure 11: Effect of relative noise on RMSE over relative variance levels if the first signal is scaled to $0 - 15$ and the second is scaled -1 to 1 with both signals having 10% relative variance.

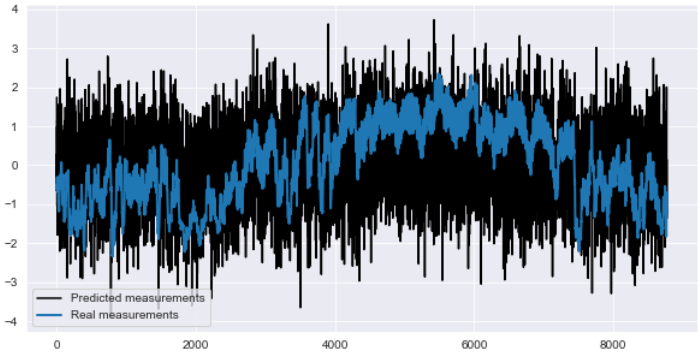

Figure 12: Effect of relative noise on RMSE over relative variance levels if the first signal is scaled to $0 - 15$ and the second is scaled -1 to 1 with both signals having 100% relative variance.

As can be seen from Figures 11 and 12 there is a clear difference between the results. In Figure 11, the predicted signal seems to be able to model the real signal to a degree. However, in Figure 12 the predicted signal seems to almost completely fail to model the true signal as there is no discernible trend in it.

There are a couple of other observations that can be made from the overall Figure 10 results. As was expected, there seems to be a direct correlation between the amount of added noise and the algorithm's performance with the higher the relative level of noise, the worse the result. Secondly, it can be seen from the results that the scales of the

signals do not matter for the error measure. That is evident because the results seem to be almost completely symmetric with minor differences. Lastly, and maybe most interestingly, the results seem to show that the factor that drives the change in error is not so much the total amount of noise added to the input but instead how much noise has the signal with the least relative noise. For example, it can be seen that if one signal has 100% relative noise added and the other only 10%, the result is better than adding both signals 20% of noise. This means that if one signal has lower relative variance than the other, the signal with lower relative variance is assigned a higher weight, and it influences the output signal to a greater degree than the other input with is rendered much less relevant. Similar to Figure 10, we can plot the ratios of weights for all of the iterations in Figure 13.

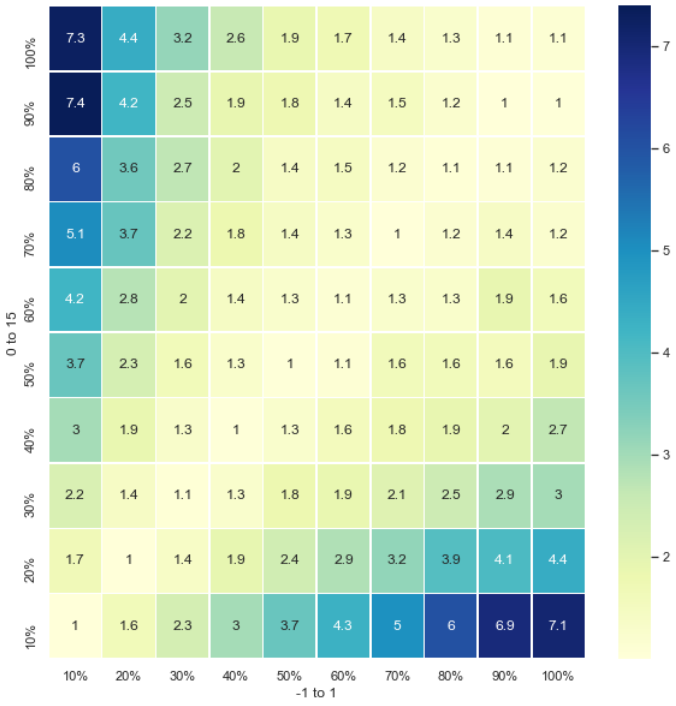

Figure 13: Ratios of weights for different relative noise levels if the first signal is scaled to $0 - 15$ and the second is scaled -1 to 1.

Figure 13, which shows the weight ratios for both input signals over different relative variance levels, can be interpreted that, for example, if the first signal scaled between $0 - 15$ has 10% of relative noise and the other signal from -1 to 1 has 100% noise

added, the first signal's weight is 7.1 times that of the second's. This is expected since, as mentioned in Chapter 2.1.2, a feature's weight changes more-or-less inversely proportional to its relative noise level compared to other features. As mentioned, one weight is always much higher than the other if one feature has less noise than the other, but similarly, if both have the same amount of relative noise, the weights are almost always equal. It also shows that the initial scaling of features does not matter if the data gets z-score transformed before running SFA on it.

3.2 Effect of Feature Engineering On Error

For this experiment, the same hourly-interval data as before and scaled inputs are used. The difference, however, is that if before we simply tried to approximate using only these two inputs, this experiment involves using feature engineering to increase dimensionality and inspect how the error changes. Additionally, higher the dimensionality, greater effect noise has on the ability to accurately model the output, so for this experiment, each sub-experiment is not run once but 25 times with different random seeds in generating the noise and for each relative noise combination, the mean RMSE is presented.

The first case of the experiment is adding time-delayed versions of the input. For this, one time-step is added not to increase dimensionality by too much and potentially suffer from the curse of dimensionality. By adding one time-step, if the initial was two-dimensional, then the new input is four-dimensional by having the original features and their values from one step earlier. Running the algorithm with this configuration, the results look as shown in Figure 14.

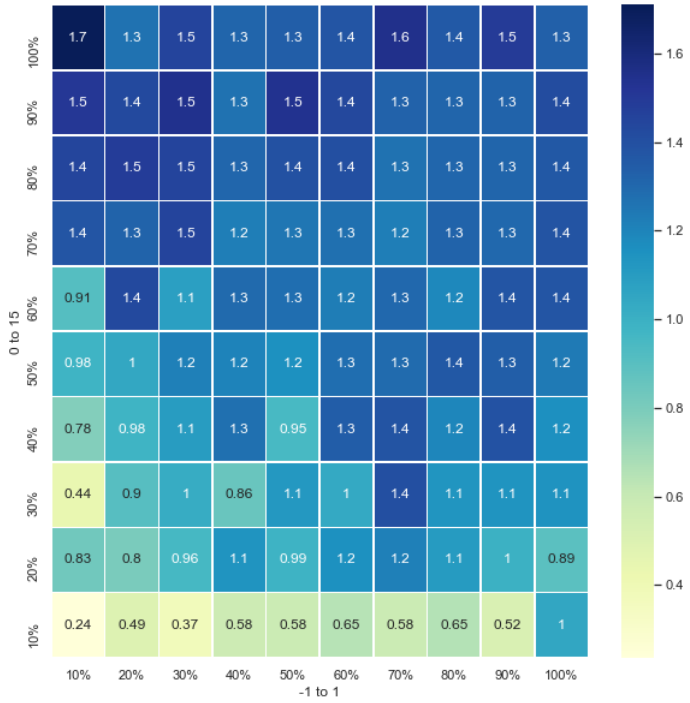

Figure 14: Effect of relative noise on the RMSE with one time-delayed step if the first signal is scaled to $0 - 15$ and the second is scaled -1 to 1. Numbers in the cells denote RMSE for that particular combination.

As can be seen, the results are worse for almost all combinations with the exception of the case where 10% of relative noise is added to both signals. Though intuitively, it would seem like having more features should yield better results, it is not always the case, as can be seen here. Additionally, if Figure showed gradual rise in error as relative noise increased, Figure does not follow the same trend. If one signal has 10% of noise, it seems to be able to model the data somewhat, but as soon as noise increases, the error rate stabilizes around $1.3 - 1.4$. It could be argued that the results are such because since SFA keeps all feature weights non-zero, having the weights of the new dimensions in the mix does not allow the model to find weights for all of the features simultaneously that would give better results than with just the two features.

Instead of adding delayed time-steps, we can also add transformed versions of features. For example, if using quadratic transformations on input features x_1 and x_2, we would add x_1^2, x_2^2, and $x_1 x_2$. Doing so and running the algorithm, we get Figure 15.

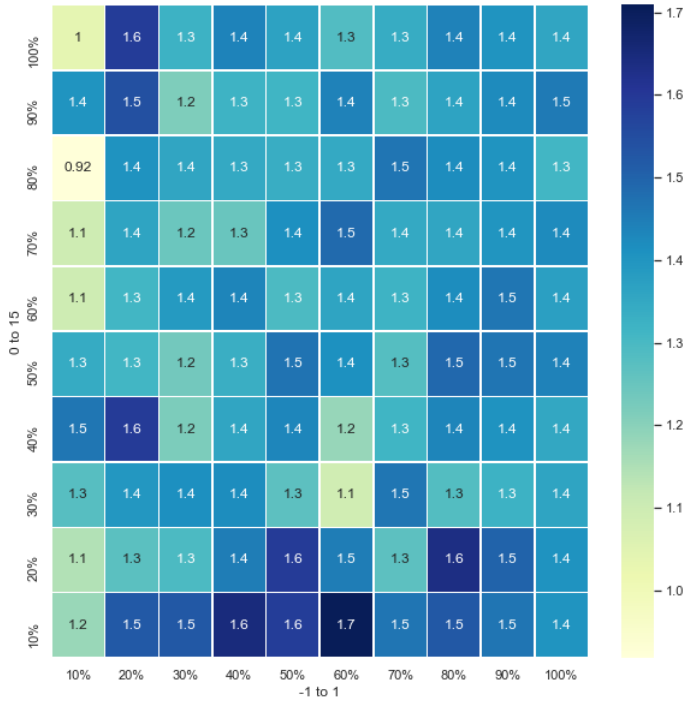

Figure 15: Effect of relative noise on the RMSE with squared features and pairwise products added if the first signal is scaled to $0 - 15$ and the second is scaled -1 to 1. Numbers in the cells denote RMSE for that particular combination.

As was the case with adding delayed time-steps, adding transformed features does not improve the error either and is even worse for smaller noise levels. The reason is likely the same as for the previous model that the increased number of features cannot find such a combination of weights that would yield a better result. However, an interesting observation could be made that the amount of noise does not seem to play a very relevant role in this case. It can be seen that no matter how much noise we add, the error stays around $1.3 - 1.5$.

Lastly, we can combine the two approaches and create a model that includes both delayed time-steps and transformed features in Figure 16.

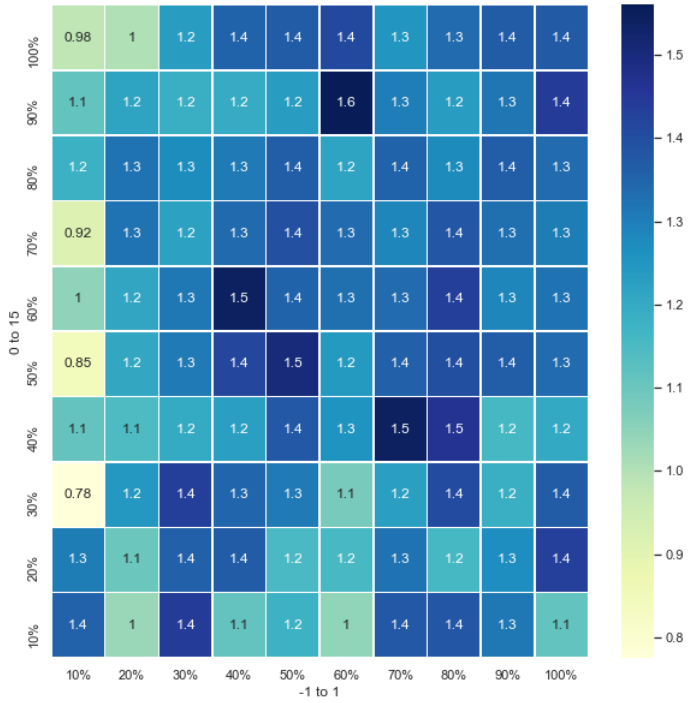

Figure 16: Effect of relative noise on the RMSE with one time-delayed step, squared features, and pairwise products added if the first signal is scaled to $0-15$ and the second is scaled -1 to 1. Numbers in the cells denote RMSE for that particular combination.

As was to be expected, the results mostly fail to improve by combining both feature engineering techniques and are in the same vein as Figures 14 and 15. While generally, it is advisable to perform feature engineering on data because applying SFA on it, in the present case, adding additional features does not offer any benefits for model performance, and it is best to simply run the model using the initial two input signals created from the actual data.

3.3 Comparison of Biological Slow Feature Analysis and the Closed-Form Solution

To compare the analytical SFA solution and the bio-SFA, the problem must be approached differently than we did for the previous experiments and relative noise levels. If the analytical solution simply calculates the coefficients which give a new series

when multiplied by the original features once, bio-SFA finds the coefficients incrementally by looking at each time step individually and modifying its weights accordingly. To compare these two algorithms, the best approach would be to calculate the RMSE values for a sample with one data-point, then on a sample with two data-points, and so on until running both algorithms on the whole dataset. However, bio-SFA does not offer a simple and computationally efficient way to calculate RMSE for each of its iterations. What we can directly compare instead is the slowness of both the analytical solution and bio-SFA. It is a suitable substitute for RMSE since the slower a signal output by SFA is, the closer it is to the actual signal, and thus by transitivity, we can use slowness for evaluation. For both, we can calculate slowness as shown in equation (2) and then simply plot the difference.

As the first and second experiments showed that we get the best results using only two scaled inputs, this experiment also uses the same features. Running this scenario on the same data as for the previous experiments and adding 10% of relative noise for both features, we get the results shown in Figure 17.

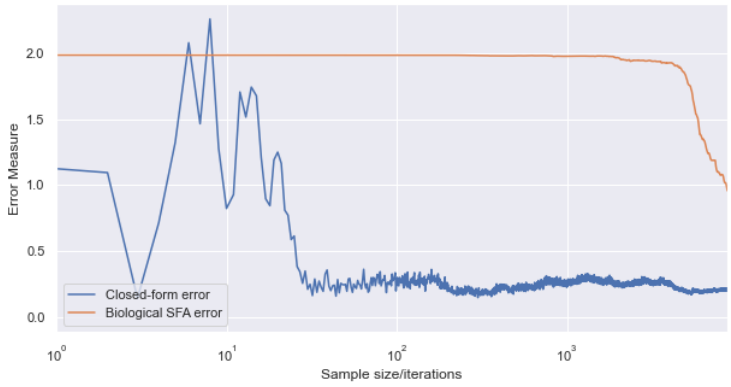

Figure 17: Comparison of the closed-form and bio-SFA slowness with 10% added noise.

As can be seen, the closed-form solution does not have a very good slowness error rate if there are less than around $25 - 30$ samples in the time series. However, for bigger datasets, the closed-form solution does not seem to be very dependent on the size of the dataset as the error rate changes relatively little over subsequent iterations. Additionally, it can be seen that the bio-SFA solution starts to convergence to the closed-form solution around 5000 time-steps but does not get near the closed-form solution error within the given data.

Knowing how the two algorithms behave in the given case, we can increase the noise and see how the results change. Additionally, since even with low levels of noise, the

bio-SFA solution did not manage to converge to the analytical solution completely, it seems sensible to increase the number of iterations the algorithm runs to account for the likely slower speed of convergence. As bio-SFA is a neural network, the easiest way to increase the number of iterations the algorithm goes through is to simply add more epochs. The previous sub-experiment ran for one epoch, but if we increase the number of epochs to, for example, 100, the new number of total iterations is $8771 \times 100 = 877100$. However, for the closed-form solution, we cannot really increase the size of data the model sees like that. Instead, we simply train the model as we did for Figure 17 and in addition plot the error of running the algorithm on the whole dataset separately for comparison with the bio-SFA solution. Doing so and adding 25% of relative noise to both signals results in Figure 18.

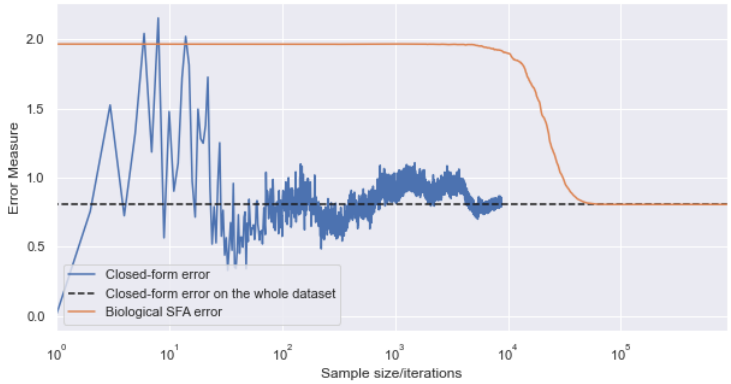

Figure 18: Comparison of the closed-form and bio-SFA slowness with 25% added noise.

From this, it can be seen that even though bio-SFA now converges to the closed-form solution, it now takes around 50000 iterations or little less than 6 epochs and is definitely more computationally expensive.

Two previous sub-experiments definitely raise the question that why we should even use bio-SFA when it takes such a long time to converge and uses more resources than the closed-form solution. As mentioned in Chapter 2.4, the main difference between the closed-form solution and the bio-SFA is that if the former simply calculates the results and these cannot be used for further training on data from the same distribution, bio-SFA fills this gap. This is possible because bio-SFA allows for saving the model weights and later initializing a new model using the saved weights. This paves the way for the possibility of applying bio-SFA to live systems where the input is continuously streamed to the model, and it would not be possible to use the classical version of the algorithm. To simulate such a live system, we use a model that runs only one epoch, but the input time series is long enough that the model should converge. For that, we can

use a dataset from the same year that does not have hourly temperature measurements but in 5-minute intervals, totaling 105255 data points. For noise levels, we can make an educated guess and add 40% for both features since, in the real world, data is expected to have some noise, but not as much as the more extreme examples presented in this thesis. Conducting such an experiment, we get the results shown in Figure 19.

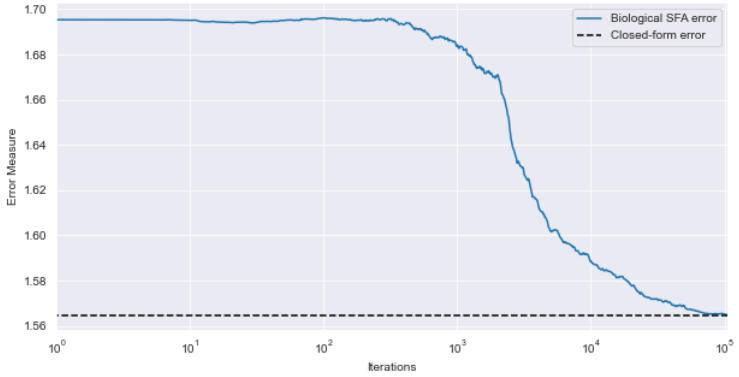

Figure 19: Comparison of the closed-form and bio-SFA slowness with a long time-series and one epoch with 40% added noise.

As can be seen, the results converge at around 80000 iterations. Whether results like presented allow for using the algorithm in live systems is not clear-cut. This mainly depends on the shape of input data and its availability. As shown in this thesis, the noisier the input data, the more difficult it is to apply the algorithm to. Additionally, if there is enough historical data that is from the same distribution as the data the algorithm will be applied to, it would be somewhat trivial to train the algorithm model accurately straight from the get-go, and it would be feasible to use bio-SFA for that system. However, imagining a system as above where we get a new data-point every 5 minutes, and there is no previous data to train on, it would take approximately $9 - 10$ months for the model to be useful, which is definitely a disadvantage. All in all, whether the biological algorithm can tackle the tasks that the classical version cannot seems very situation-dependent and should be analyzed before attempting to apply it.

4 Conclusions and Further Research

The main objective of this thesis was to investigate the applicability of SFA for learning invariant representations from noisy input data and evaluate a biologically plausible version of the algorithm. The body of the thesis was divided into two main sections: theoretical background and experiments. Theoretical background first outlined the different types of learning and went through an example of approximating the actual signal from noisy measurements. After the example, the thesis went more specific into SFA theory and outlined both the optimization problem and the actual SFA algorithm. After showing how the classical version of SFA works, the thesis also brought in a biologically plausible version of the model. The last part of the theoretical section of the thesis focused on outlining possible SFA feature engineering approaches and singled out some of the previous applications where SFA has been successfully applied.

The second part of the thesis conducted three experiments. First of these looked at the case where we have two input signals and examined how does performance of SFA change if we increase relative noise of inputs. That experiment showed a direct relationship between the amount of relative noise added and the increase in error. The same experiment also showed that scales of the initial data do not matter to SFA if z-score transforming input data before applying the algorithm to it. The second experiment built upon the first and analyzed the case where we add additional features to data and see how the performance of SFA changes. The experiment showed that none of the instances of applying feature engineering to that particular data improved upon the first experiment's results. It does not mean that feature engineering should be discounted, but rather it depends on particular data. The last experiment compared the classical formulation of SFA and its biologically plausible version if the latter would be a suitable alternative. The experiment showed that if we are dealing with a situation where we simply want to extract features from a static dataset, the classical formulation of SFA should be preferred as it is computationally cheaper and faster. However, suppose we have a live system where data is continuously streamed into a model and the analytical version cannot be used. In that case, bio-SFA is a viable option as it converges to the optimal solution within a reasonable number of iterations.

As previous literature on SFA is scarce, there is a wealth of different directions for future research. Perhaps the most interesting would be to further explore bio-SFA for possible applications as the method is relatively recent, and there are no studies on it other than the original paper [2]. As the primary purpose of SFA is to find latent features from noisy data, one possible application would be to see how does bio-SFA manage to extract relevant features from natural images. It would also be beneficial to study applications of bio-SFA that are not easily possible with the classical formulation of the algorithm, such as anomaly detection from video data.

Bibliography

[1] Wiskott, L. and Sejnowski, T.J., 2002. Slow feature analysis: Unsupervised learning of invariances. *Neural computation, 14*(4), pp.715-770.

[2] Lipshutz, D., Windolf, C., Golkar, S. and Chklovskii, D.B., 2020. A biologically plausible neural network for Slow Feature Analysis. *arXiv preprint arXiv:2010.12644*.

[3] Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press.

[4] Berkes, P. and Wiskott, L., 2005. Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of vision*, 5(6), pp.9-9.

[5] Sprekeler, H. and Wiskott, L., 2008. Understanding slow feature analysis: A mathematical framework. *Available at SSRN 3076122*.

[6] Kompella, V.R., Luciw, M. and Schmidhuber, J., 2011, June. Incremental slow feature analysis. In *Twenty-Second International Joint Conference on Artificial Intelligence.*

[7] Malik, Z.K., Hussain, A. and Wu, J., 2014. Novel biologically inspired approaches to extracting online information from temporal data. *Cognitive Computation, 6*(3), pp.595-607.

[8] Hyvärinen, A., 1999. Survey on independent component analysis. *Neural Computing Surveys, 1*(2), pp.94-128.

[9] Laurenz Wiskott et al. 2011, *Slow feature analysis*, Scholarpedia, viewed 7 May 2021, `http://www.scholarpedia.org/article/Slow_feature_analysis`.

[10] Sprekeler, H., Zito, T. and Wiskott, L., 2014. An extension of slow feature analysis for non-linear blind source separation. *The Journal of Machine Learning Research*, 15(1), pp.921-947.

[11] Berkes, P., 2005. Pattern recognition with slow feature analysis. *Cognitive Sciences EPrint Archive*, pp.1-14.

[12] Koch, P., Konen, W. and Hein, K., 2010, July. Gesture recognition on few training data using slow feature analysis and parametric bootstrap. In *The 2010 international joint conference on neural networks (IJCNN)* (pp. 1-8). IEEE.

[13] Kühnl, T., Kummert, F. and Fritsch, J., 2011, June. Monocular road segmentation using slow feature analysis. In *2011 IEEE Intelligent Vehicles Symposium (IV)* (pp. 800-806). IEEE.

[14] Wiskott, L., 2010, June. Gender and age estimation from synthetic face images. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems* (pp. 240-249). Springer, Berlin, Heidelberg.

Appendix

I. Experiment Source Code

All of the code for this thesis was written using Jupyter Notebooks and Python programming language. All of the notebooks and associated data are available at the following GitHub repository: `https://github.com/kaarelkaasla/sfa`

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Kaarel Kaasla**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Evaluating Slow Feature Analysis on Time-Series Data,
   (title of thesis)

   supervised by Meelis Kull.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kaarel Kaasla
06/05/2021