Time series analysis with dynamic law exploration

A. Jakovac

Wigner Research Centre for Physics, 1121 Budapest, Hungary
April 23, 2021

Abstract

In this paper we examine, how the dynamic laws governing the time evolution of a time series can be identified. We give a finite difference equation as well as a differential equation representation for that. We also study, how the required symmetries, like time reversal can be imposed on the laws. We study the compression performance of linear laws on sound data.

1 Introduction

Intelligence is hard to define, and a lot of effort is made to bring up a sensible interpretation [1,2]. The significance of the definition itself is that it provides a framework of thinking, and as such it initiates the possible questions and thus the possible answers, too.

As an example, in the field of artificial intelligence (AI) the study of the visual cortex of the brain motivated the creation of feed forward (deep) neural networks, in particular the convolutional networks [3]. Complemented by the mathematical notions like Bayesian analysis lead to numerous brilliant achievements [4]. In all types of tasks, like classification, regression, data compression and generation the present day neural networks have achieved considerable success.

Nevertheless, AI systems face serious challenges. They are still vulnerable to unexpected errors, adversarial attacks [5], catastrophic forgetting [6]. Usually long range correlations are hard to maintain with current AI methods (see for example [7]). All these difficulties make it necessary to rethink the basic concepts of AI over and over again.

As classification is concerned, the basic view of the intelligent actor is that as output it provides the (conditional) probability \( p(y|x) \) that a certain class \( y \) can be observed in the input \( x \) [8]. One can give arguments why this approach is correct [8]. On the other hand human perception recognize classes not based on a single probability, but recognizing numerous features in the input. If all,
or at least most of the features characteristic to a given class can be found in
the input, then we confidently can single out that class as a result of classi-
fication. If characteristic features are missing, or, even worse, we find features
that contradict to the given class, then we exclude the input from that class. It
is not a linear process, but a hard veto.

In [9] we tried to lay the mathematical foundation of a theory that supports
the intelligent perception through features. The goal of the present paper is
to apply these ideas for the analysis of time series with linear features. Linear
mappings have a long history, from support vector machines [10] to Linear
Predictive Coding [11]. The presented method of identifying laws are close to
these techniques, but uses a different line of thought.

This paper is organized as follows. First we overview the relevant and irrel-
vant features of [9] in Section 2. Then we recap the treatment of time series
with embedding 3. In the next Section (Section 4) we examine the properties
of linear features. Then we apply our technique for musical data compression
eexample in Section 5. Finally we close the discussion with a Conclusions section
(Section 6).

2 Relevant and irrelevant features

Intelligence can be thought to be an ability to rephrase a complex input in
terms of new notions that fit best to the input set. For example the notion of
quadrupedality applies for most mammals, the carnassial dentition is character-
istic for carnivora, while the property to be rided applies mostly for horses (or,
eventually donkeys, ostriches, water buffalo etc.). But a useful property is also
that a given person has a name Betty; although there are a lot of Betties in the
world, but in our closer environment probably there are just few.

Thus a property is useful if either it characterizes a whole set, taking a
constant value on each of its elements (for example the property of having
carnassial dentition is “true” for carnivores and “false” for other animals); or if
it can be used to make distinction between elements within the set (she is the
only person among our friends called Betty). These useful properties can be
called features, and, following [9], they are called relevant, if they are constant
on a set, and irrelevant if they can tell elements of a set apart.

For classification the relevant features are the most useful, since each relevant
feature singles out a set, the common appearance of several relevant features
single out the intersection of the given sets, thus restricting the interpretation of
the input. On the other hand for compression (when we know the set) we need
to remember only the irrelevant features; for example we can tell our friends to
call Betty without needing to specify other properties of the person.

The mathematical basis to deal with relevant and irrelevant features are
published in Ref. [9], here we only recall the most important findings. In this
approach the probability space of the intelligent actor is \((X, F_X, P)\), where \(X\)
is the set of all possible inputs (considered to be a finite set), \(F_X\) is its power
set, and \(P\) is the uniform discrete probability measure (i.e. \(P(A \subset X) =\)
The intelligent actor (e.g. a neural network) is a set of random variables \( \{ \xi_i | i = 1, \ldots, N \} \) over this probability space with joint probability distribution over \( p^{(A)}(\xi_1, \ldots, \xi_N) \) over \( A \). To represent a subset \( A \subset X \) we choose random variables that are independent over \( A \), i.e. the joint probability distribution factorizes \( p^{(A)}(\xi_1, \ldots, \xi_N) = p^{(A)}(\xi_1) \cdots p^{(A)}(\xi_N) \). Moreover these random variables should be either constant \( \xi_i(x \in A) = \sigma_i \), or they have uniform distribution (i.e. the distribution is constant \( p^{(A)}(\xi_i = \sigma) = 1/|A| \)). The former are called relevant feature, the latter irrelevant feature. One can prove that any set can be completely described by giving appropriate number of relevant and irrelevant features \[9\].

In practice instead of determining the exact relevant features, we are usually satisfied with an approximation. This means that instead \( \xi_i(x \in A) = \sigma_i \) constant we can only ensure \( |\xi_i(x \in A) - \sigma_i| < \varepsilon \) for some small \( \varepsilon \).

A relevant feature restricts the possible input set to be in \( A \subset \xi_i^{-1}(\sigma_i) \), where \( \xi_i^{-1} \) is the inverse image. In the approximate case we have \( A \subset \xi_i^{-1}([\sigma_i - \varepsilon, \sigma_i + \varepsilon]) \). Several relevant features restrict more and more the possible elements of the input set, and so it is the basic building block of classification.

Irrelevant features, on the other hand, are useful to tell apart the elements of the studied subset of input. Indeed, if we denote the range \( \xi(A) \) by \( R \), then \( \xi_i^{-1}(r), r \in R \) yields a pairwise disjoint subdivision of our subset. Using independent irrelevant features, these subdivisions will differ, and so with enough number of irrelevant features all elements of \( A \) can be uniquely described. Thus the task of compression is equivalent to find all the irrelevant features.

Following this line of thoughts we can think about an intelligent actor as one that is capable to find the best features of the studied input set. Depending on the task one should seek the relevant or the irrelevant ones.

In the following we discuss the application of the above ideas for the treatment of time series. We will mostly concentrate on relevant features, or laws, because it leads to a recursive description of the series, from where one can restore the complete series.

### 3 Time series, laws and symmetries

Time series are mathematically \( y : \mathbb{R} \to \mathbb{V} \) functions, where \( \mathbb{V} \) is a finite dimensional Hilbert space. To be able to handle this function, we have to prepare a finite, representative set from it. We will choose a finite set \( \{ t_k | k \in \{1, \ldots, K\} \} \subset \mathbb{R} \) as initial points, and then we will sample the time series at finite past points starting from \( t_k \). In this way we generate the database

\[
\mathcal{Y} = \{ Y^{(k)} \in \mathbb{V}^{n+1} | Y^{(k)}_i = y(t_k - i\Delta t), \ i \in \{0, \ldots, n\}, \ k \in \{1, \ldots, K\} \}. \quad (1)
\]

We will need to find relevant features (laws), which satisfy

\[
\mathcal{F} : \mathbb{V}^{n+1} \to \mathbb{R}, \quad \mathcal{F}(Y^{(k)}) = 0, \ \forall k. \quad (2)
\]

\[\text{Thus the output is not the probability distribution, as it is usual in a lot of works.}\]
This construct can be thought of as a neural network (c.f. Fig. 1), if we choose $t_k = k \Delta t$. In this case we can build a network, where we have $n$ memory slots for the past $i = 1, \ldots n$ elements, and we always put the new element into the first (0th) place, shifting all the others down. The content of the $i$th slot after $k \geq n$ steps is $Y_i^{(k)}$, and the single output is simply

$$\text{output}^{(k)} = F(Y_0^{(k)}, \ldots, Y_n^{(k)}).$$

(3)

We may have restrictions on the actual form of $F$ from requirements (prior knowledge) about the time series. For example we may want that the law is time reversal symmetric: this may come from the knowledge that basic physical laws are time reversal invariant, and if we want to describe only a steady state then the state also does not single out an arrow of time. Time reversal symmetry requires that if we feed the function in a reverse direction, the same law applies:

$$F(x_0, \ldots, x_{n-1}) = F(x_{n-1}, \ldots, x_0).$$

(4)

In some cases, like acoustic signals, or market prices we also expect that the law is independent of the amplitude. Therefore if $x$ satisfies the law, then $\alpha x$ should also satisfy it:

$$\text{scaling symmetry: } F(x) = 0 \Rightarrow F(\alpha x) = 0.$$  

(5)

This requirement can be fulfilled if

$$F(\alpha x) = \alpha^z F(x),$$

i.e. $F$ is a scaling function with (an arbitrary) scaling dimension $z$. This requires that all activation functions are power functions.

### 4 Linear mapping

The simplest form of a law is a linear map, then the network is similar to the linear support vector machine, albeit we apply a different logics here.
If the maps of the feature function are linear, then it is worth to apply the linear algebra notations. In particular we denote the input set as a matrix

$$Y_{ki} = Y_i^{(k)} = y(t_k - i\Delta t), \quad k \in \{0, \ldots, K\}, \quad i \in \{0, \ldots, n\}. \tag{7}$$

A linear map is equivalent to giving a vector \(w \in V^{n+1}\) for which

$$F(z \in V^{n+1}) = \sum_{i=0}^{n} z_i \cdot w_i \equiv z^T w, \tag{8}$$

where the first dot denotes the \(V\) scalar product, while in the second form we use the usual matrix notation. Applying it to our sample set we get \(K\) results

$$F(Y^{(k)}) = \sum_{i=0}^{n} Y_{ki} w_i \equiv (Yw)_k \tag{9}$$

by denoting it as a matrix multiplication.

We remark here that we could use a more general Ansatz here. If we define a coefficient matrix \(R \in V^{n+1} \otimes V^{n+1}\), then the form of the law could be

$$F(z) = z^T R w. \tag{10}$$

If \(R\) is invertible, then this simply redefines the \(w\) weights. Practical use of this form comes when \(R\) is diagonal and have zero and one elements.

$$F(z) = \sum_{i=0}^{n} r_i z_i \cdot w_i, \quad r_i \in \{0, 1\}. \tag{11}$$

In this way we can leave out certain past points from the consideration, which can be a useful technique to reduce the number of effective weights. Technically this can be taken into account with \(Y_{ki} \rightarrow r_i Y_{ki}\), and in the resulting weights we put \(w_i \rightarrow r_i w_i\). Thus this general case can be completely embedded in the present discussion.

The above map is a relevant feature/law if \(F(Y^{(k)}) = 0\) for all \(k \in \{0, \ldots, K\}\), it is

$$Yw = 0. \tag{12}$$

This is equivalent to \(|Yw| = 0\) with some norm. With the usual quadratic norm we may define

$$\chi^2 = \frac{1}{K} (Yw)^T (Yw) = w^T C_{PCA} w \equiv 0, \tag{13}$$

where

$$C_{PCA} = \frac{1}{K} Y^T Y \tag{14}$$

is the correlation matrix of the given problem (the same matrix appears in the Principal Component Analysis, PCA).

This can be simply satisfied with \(w = 0\). To avoid this trivial solution we require \(|w| = 1\). To implement this constraint we realize that the minimum of
\(\chi^2\) is zero, so instead of an equation, we can think the problem as minimization. Using the Lagrange multiplicator method we should minimize

\[
\chi^2_{\lambda} = w^T C_{PCA} w - \lambda w^T w = \text{minimal.}
\]

The solution of this requirement is

\[
C_{PCA} w^{(\lambda)} = \lambda w^{(\lambda)}
\]

eigenvalue equation. Since \(Y^T Y\) is symmetric and positive definite, all eigenvalues are real and positive. Rewriting it to \(\chi^2\) we find

\[
\chi^2 = \lambda.
\]

This means that an exact law would be equivalent to find a zero eigenvalue in \(C_{PCA}\).

In practice it is usually not possible to satisfy, we always have a (small) breaking of the law. We can describe it as

\[
\mathcal{F}(Y^{(k)}) = \xi_k,
\]

where \(\xi_k\) can be thought of as a random process. It is remarkable that

\[
\langle \xi^2 \rangle = \frac{1}{K} \sum_{k=1}^{K} \xi_k^2 = \chi^2 = \lambda.
\]

This means that the variance of the accuracy of the law is given by the eigenvalue.

We remark that although in the PCA analysis we use the same matrix, but there we seek the largest eigenvalue. The PCA logic is to find the direction that aligns with the data the most, i.e. the direction where the data vary the most, which corresponds to the largest eigenvalue of the PCA matrix. We, however, look for a direction that characterizes all the data, so we need the direction where the data differ the least. This corresponds to the smallest eigenvalue of the PCA matrix. In the language of [9], the eigenvectors belonging to the largest eigenvalues are the irrelevant, the ones belonging to the smallest eigenvalues are the relevant features.

In realistic situations when we analyze the eigensystem of the PCA matrix we find a spectrum, similar to what can be seen in Fig. [4]. As we see, the PCA matrix has a few large(ish) eigenvalues and a lot of small ones. If we went on to determine longer laws, we would have determined more and more small, i.e. relevant eigenvalues. This is a very general observation for natural processes: we have much more relevant than irrelevant features [9]. Any of the relevant features can be used as a law.

### 4.1 Function class compatible with linear law

A generic matrix does not have a zero eigenvalue. Let us examine what properties a time dependent function must have to make it possible. Here we restrict ourselves to \(V = \mathbb{R}\) real valued time series.
A law over real valued time series can be organized as a recursion. First, using the definition of $Y$ \((7)\) we can rewrite the law \((12)\) as

$$y(t_k) = -\frac{1}{w_0} \sum_{i=1}^{n} y(t_k - i\Delta t)w_i.$$ \hspace{1cm} (20)

Now choosing $t_k = k\Delta t$ we can define a series

$$y_k = y(k\Delta t).$$ \hspace{1cm} (21)

It satisfies then

$$y_k = -\frac{1}{w_0} \sum_{i=1}^{n} y_{k-i}w_i, \hspace{1cm} \forall k \geq n.$$ \hspace{1cm} (22)

Being a linear recursion, sum of solutions is also a solution. For a particular solution we try an Ansatz

$$y_k = q^{-k}.$$ \hspace{1cm} (23)

Rewriting this Ansatz into \((22)\) we find

$$0 = \sum_{i=0}^{n} w_i q^{-k+i} = q^{-k} \sum_{i=0}^{n} w_i q^i.$$ \hspace{1cm} (24)

This is true, if

$$\sum_{i=0}^{n} w_i q^i = 0.$$ \hspace{1cm} (25)

That means that the solutions are roots of the

$$P(x) = \sum_{i=0}^{n} w_i x^i = 0$$ \hspace{1cm} (26)
nth order polynomial. This equation yields \( n \) solutions, that are either real, or complex, appearing in conjugated pairs (so altogether \( n \) real data). Let us denote the roots as \( \{ q_a | a = 1 \ldots n \} \). With all these roots we can fully fit the \( n \) initial conditions of the recursion, thus it yields the complete solution.

As a side remark we can establish a relation between \( w_s \) and \( q_s \) from the condition that
\[
\sum_{i=0}^{n} w_i x^i = w_0 \prod_{a=1}^{n} (x - q_a).
\]

(27)

Thus the time series with generic initial conditions that satisfies the linear law can be written as
\[
y(k\Delta t) = \sum_{a=1}^{n} c_a q_a^{-k},
\]

(28)

where the \( c_a \) coefficients are complex numbers. With the notation
\[
\alpha_a = \frac{1}{\Delta t} \ln q_a
\]

(29)

we can write
\[
y(t) = \sum_{a=1}^{n} c_a e^{-\alpha_a t}.
\]

(30)

Although originally it is valid for discrete times, but it is easy to verify that the above form is a solution for any starting time. This form gives the definition of that function class that is compatible with a linear law.

To give still another view, we observe that
\[
(\partial_t + \alpha) e^{-\alpha t} = 0.
\]

(31)

Therefore (30) satisfies
\[
\prod_{a=1}^{n} (\partial_t + \alpha_a) y(t) = 0.
\]

(32)

Thus \( y(t) \) satisfies an \( n \)th order linear differential equations with constant coefficients.

A corollary is that in the linear order a law can be given in multiple ways:

- give an \( w \in \mathbb{R}^{n+1} \) weight function, then the element satisfy the law lie in the orthogonal subspace: \( y^T w = 0 \)
- give a linear \( n \)th order recursion: \( y_0 = -\sum_{i=1}^{n} \bar{w}_i y_i \), where \( \bar{w}_i = w_i / w_0 \)
- give the \( q_a \) roots of the polynomial \( \mathcal{P}(x) = \sum_{i=1}^{n} w_i x^i \): the weights can be reconstructed from \( \sum_{i=1}^{n} w_i x^i = w_n \prod_{a=1}^{n} (x - q_a) \)
- give the corresponding differential equation.

The recursive form of the law (22) can be generalized to nonlinear laws, then the law should be formulated as a (nonlinear) recursion.
4.2 Stability of a recursion

Let us consider the asymptotic behavior of the recursion, i.e. for large $t$. If the real part of $\Re \alpha > 0$, then for large $t$ it yields an exponentially growing solution, that is to be excluded. If $\Re \alpha < 0$, then for long times it is vanishing, and we will not observe it. Therefore for long times only the imaginary $\alpha$ survive.

Since $y(t)$ must be real for all $t$, the roots $e^{i\omega t}$ and $e^{-i\omega t}$ both appear in the series (with complex conjugate coefficients). This means that in the differential equation form we find

$$(\partial_t + i\omega)(\partial_t - i\omega) = \partial_t^2 + \omega^2.$$ \hfill (33)

Therefore the differential equation is time reversal invariant.

What does it imply for the weights in the discretized law? We should consider the relation (27) which connects the roots with the coefficients. Now the roots appear in pairs:

$$(x - e^{i\omega \Delta t})(x - e^{-i\omega \Delta t}) = x^2 - 2x \cos(\omega \Delta t) + 1.$$ \hfill (34)

Here the coefficient of the $x^2$ and 1 terms are the same. This will be inherited to all of the weights. To see it, we observe that the polynomial coming from the above pairs satisfies (using $Z_a = -2 \cos(\omega \Delta t)$)

$$P\left(\frac{1}{x}\right) = \prod_{a=1}^{\ell} (x^{-2} + Z_a x^{-1} + 1) = x^{-2\ell} \prod_{a=1}^{\ell} (1 + Z_a x + x^2) = x^{-2\ell} P(x). \hfill (35)$$

where $\ell$ is the number of frequencies (i.e. $n = 2\ell$). If we consider the same polynomial in weight representation, we can write (note that $2\ell = n$)

$$x^n P\left(\frac{1}{x}\right) = x^n \sum_{i=0}^{n} w_i x^{-i} = \sum_{i=0}^{n} w_i x^{n-i} = \sum_{j=0}^{n} w_{n-j} x^j.$$ \hfill (36)

This is equal to $P(x)$, if

$$w_{n-i} = w_i, \quad \forall i = 0, \ldots, n.$$ \hfill (37)

This also means that the recursion forward and backward happens with the same law: this is the manifestation of time reversal symmetry.

This is the property that can be easily generalized to nonlinear recursions: we should require that the forward and backward iteration happens with the same law.

4.2.1 Example: a single sin

To see an example, consider a simple function:

$$y(t) = A \sin(\omega t + \varphi).$$ \hfill (38)
This satisfies \((\partial_t^2 + \omega^2)y(t) = 0\), so we know that it can be represented a law consisting of 3 terms (it can be longer, too, see later).

Indeed, from the harmonic function addition law

\[ y_c = y(t + c\Delta t) = A\sin(\omega t + \varphi)\cos(c\omega \Delta t) + A\cos(\omega t + \varphi)\sin(c\omega \Delta t) \quad (39) \]

we see that

\[ y_0 - 2\cos(\omega \Delta t)y_0 + y_1 = 0. \quad (40) \]

Thus the weights are \(w = (1, Z, 1)\), where \(Z = -2\cos(\omega \Delta t)\), as it was claimed before.

We could use the relation between the root and weight representation of a polynomial \((27)\) to find the same result.

### 4.2.2 Example: sum of two sin functions

Let us consider the function

\[ y(t) = A_1 \sin(\omega_1 t + \varphi_1) + A_2 \sin(\omega_2 t + \varphi_2). \quad (41) \]

This satisfies

\[ (\partial_t^2 + \omega_1^2)(\partial_t^2 + \omega_2^2)y = 0. \quad (42) \]

The corresponding polynomial reads

\[
\sum_{k=0}^{4} w_k x^k = (x^2 + Z_1 x + 1)(x^2 + Z_2 x + 1) =
\]

\[ = x^4 + (Z_1 + Z_2)x^3 + (2 + Z_1 Z_2)x^2 + (Z_1 + Z_2)x + 1. \quad (43) \]

We see that the composite law maintains time reversal symmetry, as it should.

### 5 Data compression using linear laws

Laws, as we have seen, represent a function class, and so they can be used to represent data with appropriate coefficients. Linear laws include polynomials, exponentials and harmonic functions (and also their mixture or product of them), and thus they can replace existing methods with broader possibilities.

If the data support our function class, then the law represents it for a long time, in extreme cases all points. We remark that these include also \(a x^ne^{-bx}\) like transient or saturation processes besides other examples.

For the description an actual data series which (approximately) satisfy a law, we need also the initial conditions for the recursion. Given the law, i.e. the function class, then each initial condition represents a different element of this class. These are useful to tell apart the elements of the class, and so they correspond to the irrelevant quantities in the language of \([9]\).
5.1 Coefficient fitting

We can determine the initial conditions in several ways. The first is to determine the $q_a$ roots of the corresponding polynomial. Then we know the the data can be described by the function $f(x)$, we just have to fit the coefficients. In order to do that we introduce the matrix

$$ Q_{ki} = q_i^k, \quad (44) $$

and use least square method

$$ \chi^2 = \sum_k (y_k - \sum_i c_i q_i^k)^2 = y^T y - 2y^T Qc + c^T Q^T Qc = \text{minimal}. \quad (45) $$

This means that the coefficient vector comes from the solution of

$$ Q^T Qc = Q^T y. \quad (46) $$

Once we have the coefficients, we can propose a continuous function that approximates data the best.

Another method is that we use the recursion directly to describe the data. The problem is that if our law is not exact, i.e. the data are noisy, then we can not just take simply the first $n$ terms to start the recursion. We should start from an optimal $\bar{y}_k$ starting values in order to achieve the best overall fit to the data.

The best way to find these optimal values is to define a series basis starting from initial condition $y_k = \delta_\ell k$ with given $\ell$, for $\ell = 0, \ldots, n - 1$. Using the recursion $f(x)$ we will have a series $s^{(\ell)}$. Then we require

$$ \chi^2 = \sum_k (y_k - \sum_{\ell=0}^{n-1} \bar{y}_\ell s^{(\ell)}_k)^2 = \text{minimal}. \quad (47) $$

Introducing a matrix again

$$ S_{kt} = s^{(\ell)}_k, \quad (48) $$

the initial condition vector $\bar{y}$ comes from the solution of

$$ S^T S \bar{y} = S^T y. \quad (49) $$

In case of nonlinear laws we have to use the recursive form of the law to determine the best description of the data. There the sum of solutions is not a solution and so the initial conditions should be fitted.

5.2 Results

Usually the accuracy of a single law will decrease with increasing data length. An $n+1$ long linear law means $n$ roots and $n$ amplitudes, together $2n$ parameters: it can therefore describe a $N_{data} = 2n$ long data vector. For longer vectors we unavoidably lose precision, the question is only, what is the rate of this loss.
To numerically characterize the loss, we can plot the accuracy $A$ versus the compression rate $R$, defined as

$$A = 1 - \frac{|X - X_{sym}|}{|X|}, \quad R = \frac{N_{data}}{2^n},$$

(50)

where $X$ is the data which is $N_{data}$ long, $X_{sym}$ is the simulated data using the $n + 1$ long law. We always get $A = 1$ for $R = 1$.

An extreme situation is when the data exactly support our function class: there $A = 1$ for any $R$. The other extreme is the random data series, where there is no law at all, see Fig. 5.2. There we can observe that $A = R^{-3/2}$ is the average trend, independently on the number of laws we use.

In all intermediate situation we lose precision, the rate depends on the actual data. As an example we study the sound of a flute, and determine the accuracy of the representation. It may remain close to a perfect description for a large compression rate, as we can see from Fig. 5.2. As it can be seen, even for very large compression rates of 40, the best laws may provide reasonable 90% accuracy. It is not the level of HiFi acoustic experience, but for recognizing a sound it is perfectly usable.

Usually a the best accuracy for a given compression rate comes from an appropriate choice of the number of terms in the law and $\Delta t$. In case of clear acoustic data, like the sound of the flute, one can reach a large compression, while for more noisy situation, like in percussion instruments, it is hard to increase the compression rate too much, c.f. Fig. 5.2. The problem with percussion instruments (even if the are imitated by human voice, like in Bobby McFerrin’s "Don’t worry" song) is that they consist mainly of “switch-on” effects, when the device starts to emit sound. These are sudden and short burst, which means that the laws change frequently. The linear law method we can only give a homogeneous law, so it will have difficulty to represent these musical instruments.
Figure 4: Accuracy at given compression rate for flute. In the left and right panel we used different compression rate range and different $N_{law}$. In both panel $\Delta t = 4$/sampling rate.

Figure 5: The available best accuracy for different music, coming from optimizing $N_{law}$ and $\Delta t$. 


Nevertheless, the nearly 97% accuracy in the randomly chosen part of the song, at compression rate about 4, is again not a bad description. It can also be noted that for small compression rate (up to about 1.5), the compression is practically lossless. This property can be used to lossless compression techniques.

5.3 Comparison with Linear Predictive Coding technique

Linear Predictive Coding (LPC), applied in many areas of sound file compression (e.g. FLAC [12]), uses a similar formalism. There are, however some important differences. LPC uses a recursive form with $w_0 = -1$ prescription for the weight, which also means that the $w_{n>0}$ coefficients must be determined by matrix inversion. In our case the weights are eigenvectors belonging to the smallest eigenvalue, which is more stable procedure. A further difference is that we only determine the dominant frequencies (relevant features), the actual coefficients of the recursion comes with an extra coefficient fitting step.

Another difference is that LPC wants to predict the next data point, while we give a continuous model that describes it. As a consequence we are not forced to use $\Delta t = 1 / \text{sampling rate}$, which extends our possibilities. Usually the best compression comes from choosing not the smallest $\Delta t$.

In addition, a law can be formed to satisfy symmetry requirements, like time translational symmetry.

6 Conclusions

The goal of this paper was to demonstrate how the idea of finding laws in input data sets is working in the linear case. We used time series to analyze, in particular sound tracks, embedded in finite $n+1$ dimensions.

Linear laws can be represented by an $n+1$ dimensional vector that is perpendicular to all embedded data. These vectors can be found as the eigenvectors belonging to the zero eigenvalue of the corresponding PCA problem. In realistic case the data do not support the law exactly. In this case we choose the smallest eigenvalue, then we know that the variance of the law breaking term is equal to the value of this eigenvalue.

As it was shown, the function class that satisfies a linear law consists of the solutions of $n$th order differential equations with constant coefficients. We determined the relation between the law and the coefficients, which allows us to give a continuous function that describes the data.

Laws can be constrained to satisfy certain requirements. Long time stability, for example, requires time reversal symmetry, which is manifested in the weights as a symmetry under reversing it.

These ideas are checked to describe realistic data, in this paper we considered music instruments. We plotted the accuracy of the description of the data vs. the compression ratio. Depending on the data the efficiency of linear laws are different. In the lack of laws, like in random data set, the accuracy decreases very fast. The more the data support a linear law, the longer a high precision
maintains. Among our examples the best compression rate could be achieved in case of piano music.

A lot of finding of the linear laws can be taken over for the nonlinear case. In particular in neural networks the last layer, which is usually a fully connected one, can be substituted by a linear law finder. Then in the result we will find several optimized features, corresponding to the corresponding laws. With more relevant features hopefully the classification tasks can be performed more reliably, since it is more and more difficult to find such adversarial directions that satisfy all the laws, but still takes us out of the given class.

Acknowledgment

The author acknowledges important discussions with T.S. Biro, D. Berenyi, P. Mati, P. Posfay and A. Telcs. The research was supported by the Ministry of Innovation and Technology NRDI Office within the framework of the MILAB Artificial Intelligence National Laboratory Program, and the Hungarian Research Fund NKFIH (OTKA) under contract No. K123815.

A Relation between root finding of a polynomial and diagonalization

Consider a recursion

\[ y_N = \sum_{k=1}^{n} c_k y_{N-k}. \]  

This recursion can be written two ways.

First we may assume that the solution of the recursion is

\[ y_N = q^{-N}, \]  

which yields

\[ \sum_{k=0}^{n} c_k y_k = 0 \]  

with \( c_0 = -1 \).

Second we can set up a matrix form with

\[
Y_N = \begin{pmatrix}
y_N \\
y_{N-1} \\
\vdots \\
y_2 \\
y_1
\end{pmatrix}, \quad M = \begin{pmatrix}
c_1 & c_2 & \cdots & c_{n-1} & c_n \\
1 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 0 & 0 \\
0 & 0 & \cdots & 1 & 0
\end{pmatrix}
\]  

as

\[ Y_N = M Y_{N-1}. \]
The $M$ matrix is diagonalizable $M = U^{-1}D U$, then with $Z = U Y$ we find

$$Z_N = D Z_{N-1} = D^N Z_0 = \begin{pmatrix} \lambda_1^N & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2^N & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_{N-1}^N & 0 \\ 0 & 0 & \cdots & 0 & \lambda_N^N \end{pmatrix} \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{N-2} \\ z_{N-1} \end{pmatrix}$$

Thus

$$z_{N+1-k} = \lambda_k^N z_{1-k}.$$ \hspace{1cm} (57)

The original $y_N$ is a linear combination of these expressions, any of this is a solution of the recursion. Therefore the roots are equal to the eigenvalues

$$q_a = \lambda_a.$$ \hspace{1cm} (58)

**References**

[1] F. Chollet, "On the Measure of Intelligence", arXiv:1911.01547 [cs.AI]

[2] H. Geffner, "Model-free, Model-based, and General Intelligence", arXiv:1806.02308 [cs.AI]

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning", Nature, 521(7553):436, 2015

[4] J. Schmidhuber, "Deep learning in neural networks: An overview", Neural networks, 61:85–117, 2015.

[5] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, J. Li; "Boosting Adversarial Attacks With Momentum" Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 9185-9193

[6] J. Kirkpatrick et.al., "Overcoming catastrophic forgetting in neural networks", PNAS 2017 114 (13) 3521-3526; https://doi.org/10.1073/pnas.1611835114

[7] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, W. Brendel "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness", ICLR 2019 Conference

[8] I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning", MIT Press, Cambridge, USA, 2017)

[9] A Jakovac, D Berenyi, P Posfay, "Understanding understanding: a renormalization group inspired model of (artificial) intelligence", arXiv:2010.13482 [cs.AI]

[10] Sh. Suthaharan, "Support Vector Machine", 2016, Springer, Boston, MA, https://doi.org/10.1007/978-1-4899-7641-3_9
[11] R. Sproat and J. Olive, "Text-to-SpeechSynthesis", Digital Signal Processing Handbook (1999) 9-11; also https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.710.1476&rep=rep1&type=pdf

[12] https://en.wikipedia.org/wiki/FLAC