JACOBS UNIVERSITY

Herbert Jaeger

Discovering multiscale dynamical features with hierarchical Echo State Networks[1]

Technical Report No. 10
July 2007

School of Engineering and Science

[1]This is a corrected version of the original techreport from June 2007. Revision history: October 2007: Change: figure 5 has been correctly instantiated (was a duplicate of figure 4 in first version). April 2008: error in equation (22) fixed.

Discovering multiscale dynamical features with hierarchical Echo State Networks

Herbert Jaeger

Jacobs University Bremen
School of Engineering and Science
Campus Ring 6
28759 Bremen
Germany

E-Mail: h.jaeger@iu-bremen.de
http://www.faculty.iu-bremen.de/hjaeger/

Abstract

Many time series of practical relevance data have multi-scale characteristics. Prime examples are speech, texts, writing, or gestures. If one wishes to learn models of such systems, the models must be capable to represent dynamical features on different temporal and/or spatial scales. One natural approach to this end is hierarchical models, where higher processing layers are responsible for processing longer-range (slower, coarser) dynamical features of the input signal. This report introduces a hierarchical architecture where the core ingredient of each layer is an echo state network. In a bottom-up flow of information, throughout the architecture increasingly coarse features are extracted from the input signal. In a top-down flow of information, feature expectations are passed down. The architecture as a whole is trained on a one-step input prediction task by stochastic error gradient descent. The report presents a formal specification of these hierarchical systems and illustrates important aspects of its functioning in a case study with synthetic data.

1 Introduction

Many time series data of practical relevance display features on multiple temporal or spatial scales. For instance, in dictating systems, the incoming speech signal must be analysed on a variety of timescales that range from the millisecond domain of phonemes to the multiple-seconds scale of phrases. Similar challenges are posed by (hand-)writing or gesture recognition systems, by motion or action controllers in robots, or in video sequence analysis applications.

A natural strategy to deal with multiscale input is to design *hierarchical* information processing systems, where the modules on the various levels in the hierarchy specialize on features on various scales. Bengio and LeCun [2] provide an enlightening discussion from a statistical / machine learning point of view why hierarchical architectures are unavoidable when dealing with complex (in a wider sense than multiscale) data. Investigating hierarchical information processing is a pervasive and classical topic in the behaviour and brain sciences [6] [29] [16]. Numerous artificial systems in artificial intelligence, robotics, pattern recognition and machine learning follow this strategy, for instance (in rough chronological order) Brooks' subsumption architecture for robot control [3], hierachical ART/ARTMAP architectures [4], hierarchical mixtures of experts [15] [27], Shastri's SHRUTI models [25], hierarchical Markov decision processes [22], convolutional neural networks [19], hierarchical HMMs [23] – not to mention standard textbooks methods for hierarchical clustering and AI planning. I will not even attempt to give a systematic account of any sort here.

Despite the naturalness of the basic conception and the substantial and varied effort, it is fair to say that so far no artificial system comes close to the performance of humans or higher animals (or even eukaryotes...) with regard to coping with, and adapting to, multiscale input. Several reasons can be made out for this:

- Most of the artificial models have been designed for *static* input patterns; ecologically realistic signals are however time series (signals).
- Most models work well only for low-dimensional input data; ecologically relevant data streams are however often very high-dimensional.
- Many hierarchical models of information processing lack a *learning* mechanism; learning and adaptation are however mandatory for any system aspiring at coming close to biological systems.
- Many models are largely hard-wired, rely on human-designed data preprocessing and feature definitions. Getting such systems to run imposes on the human designer an enourmous workload of pre-designing the "ontology" (from perceptual features to abstract concepts) of the internal representations needed for efficient processing.
- Hierarchical learning algorithms are often very slow and computationally expensive, and (if gradient-based) suffer from vanishing gradients on the layers which are remote from the input / error definition level.
- The most widely used classes of learning algorithms for hierarchical systems are not biologically plausible (I would rate EM, backprop, and sampling-based methods as biologically implausible, although this is certainly a matter

of debate). If true, this would mean we haven't understood something basic about biological information processing yet.

Turned positively, what one (still) would like to see is a design of a hierarchical information processing system, which

1. feeds on high-dimensional time series data,
2. discovers its own (dynamical!) features and concepts with little human-given priors,
3. has a computationally cheap, statistically efficient, biologically not impossible, and online adaptive learning algorithm.

In the machine learning and artificial neural networks communities, the quest for such a design has recently been rekindled by the introduction of *Deep Belief Networks* (DBNs) [7] [8]. DBNs expand on the classical Boltzmann machines of the same authors [1] [9], obtaining a crucial increase in computational efficiency of sampling-based learning algorithms by reducing the sampling depth to two steps ("contrastive divergence") and by restricting the connectivity of multilayered networks to between-layer links ("reduced Boltzmann machines", RBM). On high-dimensional (image) benchmark data, DBNs outperformed previous methods in classification tasks, which is particularly remarkable because these models were trained on raw pixel data and had to discover relevant features by themselves. Layered RBMs have also been applied to time series data [28] [26]. The published work on temporal layered RBM modeling is however still scarce and hardly allows a substantial assessment.

In this report I introduce a hierarchical learning architecture which targets the three desirable properties listed above. I have been pondering various aspects of this architecture for some years, and the motivations had been different than the desiderata from the above list. But it was Hinton's et al deep belief networks that gave the final spur to condense earlier bits and pieces into a coherent hierarchical architecture, and to aspire to the bold goals stated above.

I will briefly comment on the original motivation behind earlier (unpublished) versions of the model presented in the rest of this paper.

The context in which I was working were echo state networks (ESNs). This is an elementary recurrent neural network (RNN) architecture where a randomly created "reservoir" RNN is passively driven be the input signal; the desired output is distilled from the multitude of nonlinear input transform signals provided by the reservoir, by a trainable linear combination. Thus only the reservoir-to-output connections are trained in an ESN. For an introduction to ESNs, the reader is referred to [10], [12], or [13].

The standard way to employ ESNs is in supervised training of temporal tasks. The trainable parameters (i.e. the reservoir-to-output weights) are linear regression weights. Many algorithms are known to compute linear regressions, batch and online adaptive ones. Among the latter, the simplest and most widely used (in linear signal processing) is stochastic gradient descent, which for linear regression weights is known in signal processing as the least means square (LMS) algorithm [5]. The convergence properties of LMS are well understood. Specifically, it is known that the attainable rate of convergence depends on the spectral spread (ratio of largest to smallest absolute eigenvalue) of the input signal correlation matrix. In a context of standard echo state networks, this implies that LMS can be used to compute output weights online only if the correlation matrix of the reservoir-internal signals has a low spectral spread. Unfortunately, with randomly created reservoirs the spread typically is 1e+12 and higher, which precludes the use of LMS. I tried to construct or pre-train reservoirs in many ways with the goal of achieving a small spread, but always in vain – which is why I and others have used recursive least squares (RLS) algorithms instead of LMS for online adaptation of echo state networks [11] [18]. RLS algorithms are not susceptible to spectral properties of their input signals and often converge very rapidly, but they have quadratic (here: in reservoir dimension) update complexity, are prone to become numerically instable, and are not biologically possible.

Thus I continued to desire an alternative approach where simple first-order stochastic gradient descent adaptation methods would work. I envisioned an unsupervised training scheme for reservoirs, which would lead to a decorrelation of the signals of the reservoir units; this in turn would allow me to use this pre-trained reservoir for stochastic gradient descent based, supervised training algorithms. In one of the attempts to orthogonalize reservoir signals, I tried to adopt the idea of neural "experts" which are trained competitively. This idea is familiar e.g. in the context of unsupervised time series segmentation [24] [17] or motor control [30]. In these methods, several neural networks are trained incrementally to predict the input signal; the adaptation rates are coupled to the current prediction performance; this leads to a specialization of the networks which ultimately take turn in predicting the input (or controlling a motor task) according to which sort of input is fed. Implementing such schemes needs a scoring method (measuring the local-in-time performance of each expert) and a voting/competing scheme (deciding what adaptation rate is allotted to an expert, given the performance score). If the training functions well and the experts take turn in a winner-take-all fashion, their voted output signals are trivially orthogonal (because only one among them is nonzero at each time).

Playing around with architectures of this sort, using echo state network output nodes (not entire echo state networks!) as experts, and stacking such systems on top of each other to realize hierarchies of experts in this way or another, and finally drawing insight and motivation from deep belief networks, I in the end arrived at the architecture which is presented in this report. It is the first one that works to my satisfaction: a hierarchy of dynamical features is discovered in the input data stream in an unsupervised fashion; these features develop pleasant orthogonality properties and thus could serve as input signals to LMS-type, cheap, supervised, adaptive learning algorithms for externally given tasks of all sorts; the adaptation mechanisms that lead to the feature discovery in the first place are computationally cheap and not biologically impossible.

The architecture is first explained informally in section 2, then formally specified in section 3. Section 4 presents the stochastic gradient weight update equations. A case study with a two-scale synthetic dataset follows in 5 – wherein it is also revealed why I would like to call the new architecture by the name of *Aha Machines.*

2 Elements of the Architecture: Informal Description

The purpose of the proposed architecture is to discover, in an unsupervised way, dynamical features in a (possibly high-dimensinal) stationary stochastic time series $\mathbf{u}(n)$, such that

- each feature is itself a time series (a signal), possibly high-dimensional – to emphasize this I will sometimes speak of *dynamical features*;
- the dynamical features are related to each other in a hierarchical fashion, with fast and/or local features at the bottom of the hierarchy and slow / global features at the top;
- the original time series can be approximately reconstituted from the extracted features, that is, the features are a structured representation (or transform) of the original time series.

Here are the global design characteristics of the proposed *dynamical feature discoverer* (DFD) architecture:

- The architecture is layered, where the main component in each layer layer is an echo state network. The bottom layer is the interface to the driving time series $\mathbf{u}(n)$ and operates at the same timescale as this time series. Higher layers operate at increasingly slower timescales. Each layer extracts features at its respective timescale.
- In tuning itself to predict $\mathbf{u}(n)$, the DFD develops a hierarchy of dynamical features, which can be seen as components (on several scales) of the conditional probability distribution of the future.
- In a bottom-up flow of information, higher levels discover/extract increasingly coarse/slow dynamical features, which can be combined (within a level) to increasingly coarse/slow representations of the input signal. The input to some level in the hierarchy is the feature-based reconstruction of the external signal available at the next lower level.

- In a top-down flow of information, each level generates expectations ("votes") about which features on the next lower level are present in the input signal to which degree. These votes are the combination weights by which the next-lower-level features are combined into the representation of the input signal at the timescale of that level.

- An DFD is equipped with an unsupervised adaptation mechanism which allows the system to "discover" features in an input signal. The basic rationale is that the DFD tunes itself to predict, at time n – discrete time is used throughout – the next input $\mathbf{u}(n)$ from the already received inputs $\ldots, \mathbf{u}(n-2), \mathbf{u}(n-1)$. Mathematically, any system that can perform a one-step prediction of a stochastic time series must implicitly or explicitly host a representation of the conditional distribution of the future given the past up to the present. The mathematical perspective for discussing DFDs is to see them as developing a hierarchical-feature-based representation of this conditional probability distribution of the future. This perspective is constitutive for observable operator models [14] and their derivatives, e.g. predictive state representations [20].

- The DFD architecture uses stochastic gradient descent (of the prediction error gradient for the predictions of $\mathbf{u}(n)$) for all adaptations.

The central concepts in dynamic feature discoverers are features and votes - both of which are dynamical in nature (i.e., signals) and are organized in a hierarchy. To avoid misunderstandings (the notion of a feature is used in many ways by different authors in different traditions), I briefly explain my usage of these terms.

Let $\mathbf{y}(n)$ be a vector-valued signal - for instance, a video sequence (vector components = pixels) or a spectrographic recording of a birdsong (vector components = frequency subband intensity). Furthermore let $\mathbf{f}_i(n)$ $(i = 1, \ldots, F)$ be a collection of signals of the same dimension as $\mathbf{y}(n)$, and let $v_i(n)$ (where again $i = 1, \ldots, F$) be a collection of one-dimensional signals. Then I will understand the signals $\mathbf{f}_i(n)$ as *features* of $\mathbf{y}(n)$, and $v_i(n)$ as *votes*, if $\mathbf{y}(n)$ can be rendered approximately by the vote-weighted combination of these features, that is, if

$$\mathbf{y}(n) \approx \sum_{i=1,\ldots,F} v_i(n)\mathbf{f}_i(n). \tag{1}$$

Note that according to this usage of terminology,

- the votes $v_i(n)$ need not fall in the interval $[0, 1]$ (although this may be desirable for intuitive interpretability in some conditions, and is indeed enforced in the architecture presented below),

- similarly, the votes $v_i(n)$ need not sum to unity – I have no probabilistic interpretation of features as mixture components in mind,

- a feature $\mathbf{f}_i(n)$ can be localised in time by having its vote equal to zero at the times when the feature is not "present",
- a feature $\mathbf{f}_i(n)$ can be localised in space by having most of its vector components equal to zero - the feature is then not "present" at those locations,
- a feature is not a predefined signal – nothing like a Fourier component or wavelet; it is context-dependent, variable/stochastic, and may require a complex, trained mechanism to be spotted and monitored,
- a signal is conceived as a time-varying *linear* combination of its features; thus my perspective here is restricted to numerical signals.

For example, a feature and its vote in a spectrographic recording of a Bach fugue $\mathbf{y}(n) = (y_1(n), \ldots, y_D(n))^\mathsf{T}$, where each y_j is the power signal of one frequency subband, could be any of the following:

- the power of the jth frequency subband, that is, $\mathbf{f}_i(n) = (0, \ldots, y_j(n), 0, \ldots, 0)^\mathsf{T}$; the vote $v_i(n)$ would then always be one;
- the spatial indicator of the jth frequency subband, that is, $\mathbf{f}_i(n) = (0, \ldots, 1, 0, \ldots, 0)^\mathsf{T}$ (the "1" at position j); the vote $v_i(n)$ then would be equal to $y_j(n)$;
- the first voice, that is, $\mathbf{f}_i(n)$ traces the frequency power components of the first voice; the vote $v_i(n)$ could then always be one – but could also assume a value of 1 only when the first voice is actually active and else be zero (or anything else);
- a temporal indicator of the part in the fugue when the second theme is introduced in the first voice: then $\mathbf{f}_i(n)$ could be just equal to $\mathbf{y}(n)$ with $v_i(n)$ going from zero to one exactly while the second theme is played in the first voice for the first time; alternatively, $\mathbf{f}_i(n)$ could be the zero signal at all times except during the introduction of the second theme, when $\mathbf{f}_i(n) = \mathbf{y}(n)$ – with $v_i(n)$ being one throughout or going to one while $\mathbf{f}_i(n)$ is nonzero.

These examples demonstrate that there is an inherent ambiguity in the feature-vote-representation: a nonzero contribution $v_i(n)\mathbf{f}_i(n)$ is identical to a contribution $(\alpha v_i(n))(\frac{1}{\alpha}\mathbf{f}_i(n))$ for any $\alpha \neq 0$, and a zero contribution $v_i(n)\mathbf{f}_i(n) = \mathbf{0}$ can be obtained by a zero vote, a zero feature, or by both simultaneously.

The "extraction" of a feature $\mathbf{f}_i(n)$ from a signal $\mathbf{y}(n)$ may involve a specialized, non-trivial nonlinear filter with memory. In this report these filters will be realized by echo state networks, but this is not one of the central characteristics of the proposed architecture. Other (maybe even linear) adaptive filters may be considered as well. Figure 1 gives a schematic of the reconstitution of a signal as a vote-weighted feature combination.

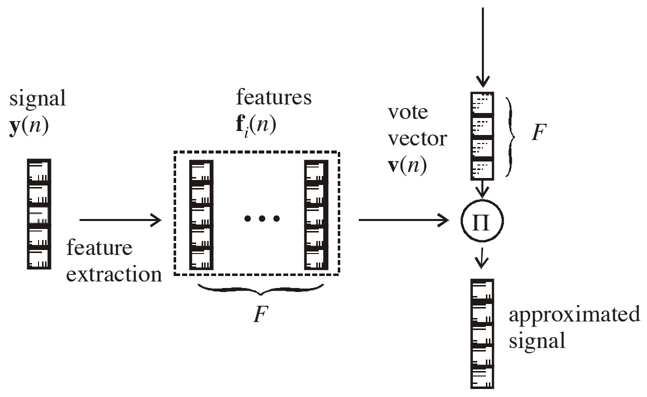

Figure 1: Schematic of approximating a signal by a voted feature combination. This schema represents the main building block of the proposed hiearchical architecture. Vectors with same texture have same dimension. For explantation see text.

While this account of representing a signal by voted features is very formal and liberal, I want to point out some intuitions which may help to constrain the design of features and their votes.

- In a feature-vote-pair, the feature can be understood as the "what" part of explaining an aspect of a signal, while the vote can be understood as the "when" or "how strong" part.
- Intuitively, the feature is finer-grained (spatially and temporally), more concrete, more informative than its vote. The vote signal $v_i(n)$ can be seen as an *abstraction* of the the feature signal $\mathbf{f}_i(n)$ in which the details of the "what" have been lost.
- The vote vector $\mathbf{v}(n) = (v_1(n), \ldots, v_F(n))^\mathsf{T}$ can thus be seen as an abstract description of $\mathbf{y}(n)$. Typically and intuitively, $\mathbf{v}(n)$ should be slower/smoother than $\mathbf{y}(n)$.
- In hierarchical representations, the vote vector $\mathbf{v}(n)$ of the features of a raw input signal $\mathbf{y}(n)$ can be subjected to a next-level feature-vote-decomposition, leading to a next-higher vote vector $\mathbf{v}'(n)$, etc.
- In such hierarchical representations, it is not necessarily the case that the number of features in higher levels of abstraction is smaller than the number of features in lower levels. While such a reduction of feature number can be motivated by the goal of compactness of representations (as for instance in [8]), it is not obvious that in human cognition the number of features decreases with the abstraction level. To the contrary, there are more words

than phonemes, more canned phrases than words, more concepts than elementary semantic roles (in many accounts of linguistic semantics), more muscial motifs than notes, etc. In the perspective adopted in this report, the purpose of feature-based representations is not dimension reduction, not *compression*, but rather the opposite: the unfolding, the *explication* of a stream of complex information into a rich display of relevant aspects.

3 Architecture: Formal Description

The central design idea concerns how to compose the features, which are extracted at each level, such that together a representation of a complex multiscale input signal is formed. See figure 2 for a graphical overview of the architecture which is now explained in more detail.

A remark on notation: the hierarchy levels of the proposed DFD architecture are structured in very similar ways. In order to refer to a quantity q at the first level, we use (like in Matlab structures) the notation R1.q; for the analog quantity at the second level we write R2.q; etc.

Assume that the input signal $\mathbf{u}$ has dimension d (may be one-dimensional, low-dimensional or high-dimensional, as in image signals).

At the lowest, first processing level of the DFD, a number of features R1.$\mathbf{f}_i(n)$, $i = 1, \ldots, \mathrm{R1}.F$ are extracted, each of which is also of dimension d. Notice again that we understand features as dynamical items, that is, as time series. The features extracted at time n should be suitable to reconstruct the external input $\mathbf{u}(n)$ in the sense that a suitable linear combination of them approximates $\mathbf{u}(n)$. Writing all features R1.$\mathbf{f}_i(n)$ as columns in a matrix R1.$\mathbf{f}(n)$ of size $d \times \mathrm{R1}.F$, we want an R1.F-dimensional vector R1.$\mathbf{v}(n)$ of *votes* to combine the features in R1.$\mathbf{f}_i(n)$ into an approximation of $\mathbf{u}(n+1)$:

$$\mathrm{R1}.\mathbf{f}(n)\ \mathrm{R1}.\mathbf{v}(n) = \hat{\mathbf{u}}(n) \approx \mathbf{u}(n). \tag{2}$$

The features R1.$\mathbf{f}_i(n)$ are computed as the outputs of an Echo State Network (ESN) with reservoir state R1.$\mathbf{x}(n)$. This ESN is driven by the external input $\mathbf{u}$ according to the leaky integrator state update equation

$$\mathrm{R1}.\mathbf{x}(n) = (1 - \mathrm{R1}.a)\ \mathrm{R1}.\mathbf{x}(n-1) + \sigma\left(\mathrm{R1}.\mathbf{W}\,\mathrm{R1}.\mathbf{x}(n-1) + \mathrm{R1}.\mathbf{W}^{\mathrm{in}}\mathbf{u}(n-1)\right), \tag{3}$$

where R1.a is the leaking rate of neurons in level 1, R1.$\mathbf{W}$ is the weight matrix for the reservoir-internal weights and $\mathbf{W}^{\mathrm{in}}$ is the input weight matrix. For the sigmoid σ, which is applied element-wise, we use the standard logistic sigmoid

$$\sigma(q) = \frac{1}{1 + \exp -q}. \tag{4}$$

For each feature R1.$\mathbf{f}_i(n)$ there is an output weight matrix R1.$\mathbf{W}_i^{\text{out}}$, which is used to obtain the feature from the reservoir state by

$$\text{R1}.\mathbf{f}_i(n) = \text{R1}.\mathbf{W}_i^{\text{out}}\ [\text{R1}.\mathbf{x}(n);\ \mathbf{u}(n-1)], \tag{5}$$

where we use the Matlab-inspired notation $[u;\ v]$ to denote vertical vector concatenation. In figure 1, all the output weight matrices R1.$\mathbf{W}_i^{\text{out}}$ are lumped together in a data structure R1.$\mathbf{W}^{\text{out}}$.

Generating the vector R1.$\mathbf{v}(n)$ of optimal votes is the task of the next higher level. At this level, the signal R1.$\mathbf{v}(n)$ is generated in the same way as the first level generates $\mathbf{u}(n)$, that is, by combining R1.$\mathbf{v}(n)$ from (second-level) features R2.$\mathbf{f}_i(n)$, $i = 1, \ldots,$ R2.F through multiplication with with another voting vector R2.$\mathbf{v}(n)$ from yet a level above. In the current implementation, creating the voting vector R1.$\mathbf{v}(n)$ involves an additional leaky integration and passing through a sigmoid. The leaky integration $\mathcal{L}$ with leaking rate λ of a signal $q(n)$ is carried out according to

$$\mathcal{L}_\lambda(q(n)) = (1-\lambda)\mathcal{L}_\lambda(q(n-1)) + \lambda q(n), \tag{6}$$

where the recursive leaky integration is initialized with a zero value. Writing again the second-level features R2.$\mathbf{f}_i(n)$ into a single R1.F $\times$ R2.F matrix R2.$\mathbf{f}(n)$, we obtain the votes R1.$\mathbf{v}(n)$ by

$$\text{R1}.\mathbf{v}(n) = \sigma(\mathcal{L}_{\text{R1}.\lambda}(\text{R2}.\mathbf{f}(n)\ \text{R2}.\mathbf{v}(n))). \tag{7}$$

The input to the ESN on the second level is made from the predicted external signal $\hat{\mathbf{u}}(n-1)$ obtained at the previous time step, which gives the following state update equation at the second level:

$$\text{R2}.\mathbf{x}(n) = (1 - \text{R2}.a)\ \text{R2}.\mathbf{x}(n-1) + \sigma\left(\text{R2}.\mathbf{W}\,\text{R2}.\mathbf{x}(n-1) + \text{R2}.\mathbf{W}^{\text{in}}\hat{\mathbf{u}}(n-1)\right). \tag{8}$$

This construction is repeated through the higher levels of the architecture. In each of the higher levels k ($k > 2$), the input to the respective ESN with state Rk.$\mathbf{x}(n)$ is the voting vector R$[k-2]$.$\mathbf{v}(n-1)$ produced on the next lower level.

At the highest level there are no votes available from above; thus the votes that are passed down from the highest to the second highest level are computed directly. If for instance the highest level is the third, as in figure 1, this means that the votes R2.$\mathbf{v}(n)$ passed from level 3 to level 2 are computed by

$$\text{R2}.\mathbf{v}(n) = \sigma(\mathcal{L}_{\text{R2}.\lambda}(\text{R3}.\mathbf{W}^{\text{out}}\ [\text{R3}.\mathbf{x}(n);\ \text{R1}.\mathbf{v}(n-1)])). \tag{9}$$

We conclude this section by an overview of all the computation steps introduced above. For ease of notation, we denote the inputs to the level k reservoirs by Rk.$\mathbf{i}(n)$. That is, R1.$\mathbf{i}(n) = \mathbf{u}(n-1)$, R2.$\mathbf{i}(n) = \hat{\mathbf{u}}(n-1)$, and R$k$.$\mathbf{i}(n) =$ R$[k-2]$.$\mathbf{v}(n-1)$ for $k > 2$. Furthermore, let $\bar{k}$ denote the maximal layer index.

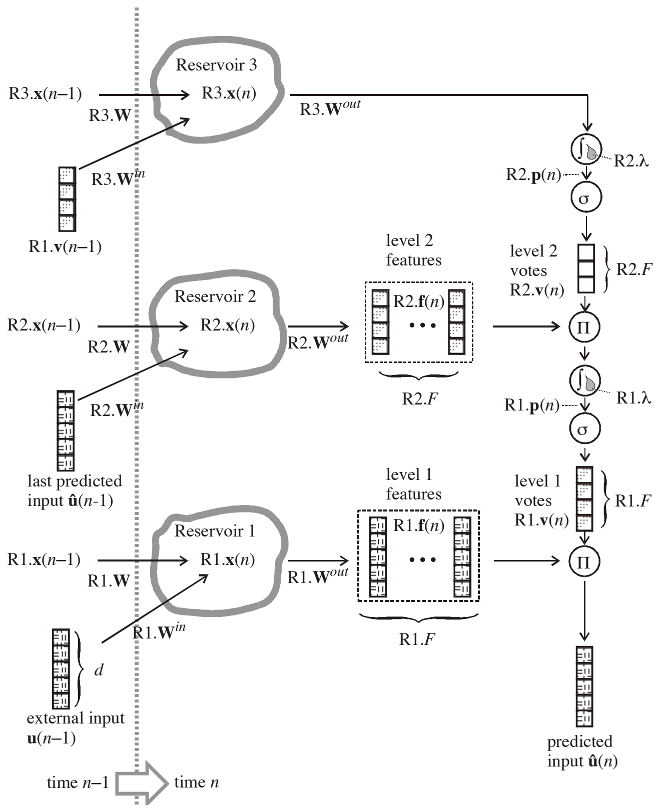

Figure 2: Overview of the DFD architecture, illustrated with a 3-level instantiation. The processing steps of one time increment are shown. Vectors of same dimension are filled with same texture for clarity. For an explantation see text.

Then, within the complete nth update cycle, first all reservoir states are updated (for $k = 1, \ldots, \bar{k}$) by

$$\mathrm{R}k.\mathbf{x}(n) = (1 - \mathrm{R}k.a)\,\mathrm{R}k.\mathbf{x}(n-1) + \sigma\left(\mathrm{R}k.\mathbf{W}\,\mathrm{R}k.\mathbf{x}(n-1) + \mathrm{R}k.\mathbf{W}^{\mathrm{in}}\mathbf{i}(n-1)\right), \tag{10}$$

and the feature blocks $\mathrm{R}k.\mathbf{f}(n)$ are obtained column-wise, for $k = 1, \ldots, \bar{k} - 1$ by

$$\mathrm{R}k.\mathbf{f}_i(n) = \mathrm{R}k.\mathbf{W}^{\mathrm{out}}_i\,[\mathrm{R}k.\mathbf{x}(n);\,\mathbf{i}(n)]. \tag{11}$$

Then the votes which are passed down from the hightest level $\bar{k}$ to the second highest level are computed:

$$\mathrm{R}[\bar{k}-1].\mathbf{v}(n) = \sigma(\mathcal{L}_{\mathrm{R}[\bar{k}-1].\lambda}(\mathrm{R}\bar{k}.\mathbf{W}^{\mathrm{out}}\,[\mathrm{R}\bar{k}.\mathbf{x}(n);\,\mathrm{R}\bar{k}.\mathbf{i}(n)])). \tag{12}$$

Next, descending to the second level, the votes $\mathrm{R}k.\mathbf{v}(n)$ are obtained, for $k = \bar{k}-1, \bar{k}-2, \ldots, 2$, by

$$\mathrm{R}k.\mathbf{v}(n) = \sigma(\mathcal{L}_{\mathrm{R}k.\lambda}(\mathrm{R}[k+1].\mathbf{f}(n)\ \mathrm{R}[k+1].\mathbf{v}(n))). \tag{13}$$

Finally, the external output $\hat{\mathbf{u}}(n)$ is obtained by

$$\hat{\mathbf{u}}(n) = \mathrm{R1}.\mathbf{f}(n)\ \mathrm{R1}.\mathbf{v}(n). \tag{14}$$

4 Learning

The only adaptive parameters in an DFD are the ESN output weights $\mathrm{R}k.\mathbf{W}^{\mathrm{out}}$. They are adapted at each time step and thus actually are time-varying quantities $\mathrm{R}k.\mathbf{W}^{\mathrm{out}}(n)$. On all levels they are adapted by stochastic gradient descent, where the gradient is taken with respect to the squared prediction error

$$\varepsilon(n) = \|\mathbf{u}(n) - \hat{\mathbf{u}}(n)\|^2. \tag{15}$$

For convenience of notation we introduce an error vector

$$E(n) = \mathbf{u}(n) - \hat{\mathbf{u}}(n). \tag{16}$$

Also, as indicated in figure 1, on each non-top level $k = 1, 2, \ldots$ we refer to a "vote potential" (the leaky integrated quantity before passing through the sigmoid)

$$\mathrm{R}k.\mathbf{p}(n) = \mathcal{L}_{\mathrm{R}k.\lambda}(\mathrm{R}[k+1].\mathbf{W}^{\mathrm{out}}\,[\mathrm{R}[k+1].\mathbf{x}(n);\ \mathrm{R}[k+1].\mathbf{i}(n)]). \tag{17}$$

Using these auxiliary quantities and denoting furthermore by $\sigma'(q) = \sigma(q)(1 - \sigma(q))$ the ordinary derivative of our logistic sigmoid; denoting the j-th component of a vector $\mathbf{q}$ by $\mathbf{q}_{[j]}$; denoting the component-wise multiplication of two equally sized matrices or vectors by $.*$[2]; and denoting by $\cdot^{\mathsf{T}}$ the vector transpose, elementary calculus gives us the following weight update equations (here given for the three-layer case from figure 1):

[2]binding order: $.*$ binds more weakly than ordinary vector/matrix multiplication.

$$\begin{aligned}
&\text{R1}.\mathbf{W}_i^{\text{out}}(n+1) = \text{R1}.\mathbf{W}_i^{\text{out}}(n) \\
&\quad + \quad \text{R1}.\gamma\ \text{R1}.\mathbf{v}_{[i]}(n)\ E(n)\ [\text{R1}.\mathbf{x}(n); \text{R1}.\mathbf{i}(n)]^\mathsf{T} \quad (i = 1, \ldots, \text{R1}.F) \qquad (18)
\end{aligned}$$

$$\begin{aligned}
&\text{R2}.\mathbf{W}_i^{\text{out}}(n+1) = \text{R2}.\mathbf{W}_i^{\text{out}}(n) \\
&\quad + \quad \text{R2}.\gamma\ \text{R2}.\mathbf{v}_{[i]}(n)\ \big(\text{R1}.\mathbf{f}^\mathsf{T}(n)\ E(n) \\
&\qquad .*\ \text{R1}.\lambda\ \sigma'(\text{R1}.\mathbf{p}(n)))\ \ [\text{R2}.\mathbf{x}(n); \text{R2}.\mathbf{i}(n)]^\mathsf{T} \qquad (19) \\
&\qquad\quad (i = 1, \ldots, \text{R2}.F)
\end{aligned}$$

$$\begin{aligned}
&\text{R3}.\mathbf{W}^{\text{out}}(n+1) = \text{R3}.\mathbf{W}^{\text{out}}(n) \\
&\quad + \quad \text{R3}.\gamma\ \text{R2}.\mathbf{f}^\mathsf{T}(n)\ \big(\big(\text{R1}.\mathbf{f}^\mathsf{T}(n)\ E(n)\ .*\ \text{R1}.\lambda\ \sigma'(\text{R1}.\mathbf{p}(n))\big) \\
&\qquad .*\ \text{R2}.\lambda\ \sigma'(\text{R2}.\mathbf{p}(n)))\ \ [\text{R3}.\mathbf{x}(n); \text{R3}.\mathbf{i}(n)]^\mathsf{T} \qquad (20)
\end{aligned}$$

where R$k.\gamma$ are learning rates.

The error terms in these expressions can be computed recursively. Basing the recursion on $E_1(n) := E(n)$, the update recursion for all non-base levels $k = 2, 3, \ldots$ reads

$$\begin{aligned}
E_k(n) &= \text{R}[k-1].\mathbf{f}^\mathsf{T}(n)\ E_{k-1}(n) \\
&\quad .*\ \text{R}[k-1].\lambda\ \sigma'(\text{R}[k-1].\mathbf{p}(n)) \qquad (21)
\end{aligned}$$

$$\begin{aligned}
\text{R}k.\mathbf{W}_i^{\text{out}}(n+1) &= \text{R}k.\mathbf{W}_i^{\text{out}}(n) \\
&\quad + \text{R}k.\gamma\ \text{R}k.\mathbf{v}_{[i]}(n)\ E_k(n)\ [\text{R}k.\mathbf{x}(n); \text{R}k.\mathbf{i}(n)]^\mathsf{T}. \qquad (22)
\end{aligned}$$

From a perspective of stochastic gradient descent adaptations, the quantities E_k can be formally interpreted as error vectors, although except for E_1 they have not been derived from taking a difference between some correct and some estimated value (one could interpret higher-level E_k as "backpropagated" versions of E_1 however).

5 A case study

The architecture has many free parameters which call for optimization: learning rates at each level, leaking rates for the votes at each level, dimensions of reservoirs and numbers of features, timescales of reservoirs, and others. Furthermore, there are a number of architectural alternatives which call for exploration. For instance, there are many plausible candidates for what to use as bottom-up input from lower levels to highers; or at the various places where in the architecture described above sigmoids are used to bound signals within a range of (01), these could be abolished or replaced by other nonlinearities.

All of these optimizations or explorations have not yet been tackled in a systematic fashion. Two preliminary observations may however be worth mentioning already at this early stage:

- Passing the votes through a sigmoid (cf. eqref5) seems to aid stability of the learning process. In some experiments with unbounded votes, the learning dynamics went out of bounds.

- In some experiments I used as input to the k-level reservoir the full block of features $\mathrm{R}[k-1]\mathbf{f}^\mathsf{T}(n)$ from the level below. This appeared to function equally well as the input suggested here. The reason for preferring the inputs $\mathrm{R}k.\mathbf{i}(n)$ as defined above is solely computational efficiency: the full feature blocks $\mathrm{R}[k-1]\mathbf{f}^\mathsf{T}(n)$ are of a much higher dimension than the inputs used here, which leads to much larger-sized matrix multiplications in the state and weight update computations.

I can at this point only report from a first case study using synthetic data. The input time series $\mathbf{u}(n)$ was prepared in the following way:

1. A one-dimensional signal $\tilde{u}(n)$ of length 50,000 was generated by a random switching between three different generators, where at each time step n with a probability of 0.05 a switching from the current to another generator (randomly chosen from among the three) occurred. One generator produced a constant-period sinewave ranging in $[0,1]$, the second generator the (chaotic) iterated tent map, and the last one a constant value which was randomly chosen in $[0,1]$ at each new call of this generator. Figure 3 gives an impression of $\tilde{u}(n)$.

2. The one-dimensional signal $\tilde{u}(n)$ was space-coded into a 5-dimensional signal $\mathbf{u}(n) = (u_1(n), \ldots, u_5(n))^\mathsf{T}$ by triangular membership functions:

$$\begin{aligned} u_i(n) &= \mathrm{Triang}(4\,\tilde{u}(n) - i + 1) \quad (i = 1, \ldots, 5) \\ \mathrm{Triang} &: \mathbb{R} \to \mathbb{R},\ x \mapsto \begin{cases} 0, & \text{if } \mathrm{abs}(x) > 1, \\ 1 - \mathrm{abs}(x), & \text{else.} \end{cases} \end{aligned}$$

This 5-dimensional signal $\mathbf{u}(n)$ was fed to a three-layer DFD as described in sections 3 and 4. The three reservoirs had 40 units each. The input weights $\mathrm{R}k.\mathbf{W}^{\mathrm{in}}$ were sampled from a uniform distribution over $[-1,1]$. The reservoir weight matrices $\mathrm{R}k.\mathbf{W}$ were generated as sparse matrices with 25% connectivity; nonzero weights were first sampled from a uniform distribution over $[-1,1]$, then rescaled such that the weight matrices attained spectral radii of 0.2, 0.5 and again

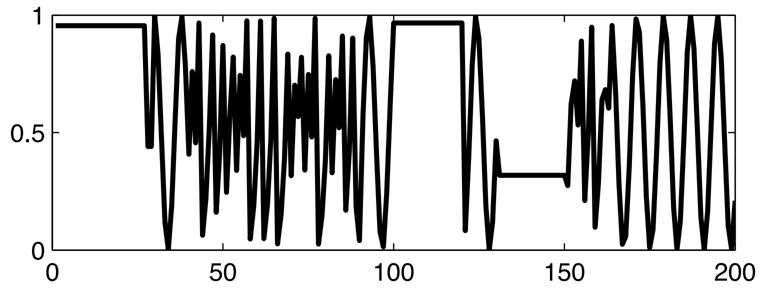

Figure 3: A 200 step sample of the input signal $\tilde{u}(n)$.

0.2 respectively for levels 1, 2, and 3. At level 1 R1.$F = 20$ and at level 2 R2.$F = 10$ features were used, all of dimension 5.

The output weight matrices Rk.$\mathbf{W}_i^{\text{out}}$ were designed to be sparse: from the possible 40 reservoir-to-output-unit connections, only 5 (randomly chosen ones) were allowed to be nonzero. This amounts to a total of 3,600 adjustable weights.

The other control parameters were set as follows: the leaking rates R1.a, R2.a, R3.a were put to 1, 0.5, 0.2 respectively (equipping higher level reservoirs with increasingly pronounced low-pass characteristics); the vote integration leaking rates R2.λ and R3.λ were set to 0.5 and 0.2 (similarly making higher levels slower in the sense of being more integrative). The learning rates Rk.γ were set to 0.01 on all levels (and not adapted during the learning process).

The reservoir states were initialized to all zeros, and the output weight matrices to small random values ranging in $[-0.05, 0.05$ for the first two levels and ranging in $[-0.01, 0.01$ for the third level.

This system was then driven by the 50,000 step training input $\mathbf{u}(n)$ for 20 Mio steps, re-cycling the training sequence, and a number of diagnostics were recorded during this run. The most interesting diagnostic concerns the prediction error. It was computed, separately for each of the five components of $\mathbf{u}(n)$, as a normalized root mean square error (NRMSE) for the last 1,000 step subsequence in every cycle through the 50,000 step training sequence:

$$\text{NRMSE}_j = \sqrt{\frac{\langle(\hat{\mathbf{u}}_{[j]}(n) - \mathbf{u}_{[j]}(n))^2\rangle}{\sigma^2_{[j]}}}, \tag{23}$$

where $\langle\cdot\rangle$ denotes the mean over the steps $49,001 - 50,000$ (modolo $50,000$), and $\sigma^2_{[j]}$ is the variance of the jth component of $\mathbf{u}(n)$. For plotting the five NRMSE$_j$ were averaged to a single NRMSE measure.

Figure 4 illustrates various aspects of the evolution of this error. The bold line in panel **a.** traces the NRMSE of the aha machine. Panel **b.** shows the second derivative, globally scaled by 50,000 and in addition multiplied with step index n in order to magnify values that come late in the series. The oscillations in curvature which are revealed in this plot indicate that progress in reducing the NRMSE is concentrated in intermittent episodes of sudden improvement. A more detailed account of this interesting phenomenon is given below. Finally, the log-log plot of the NRMSE (panel **c.**) suggests that on the average, across these episodes, the error decreases by a power law. I now proceed to discuss the observations which can be made in figure 4**a.-c.** in more detail.

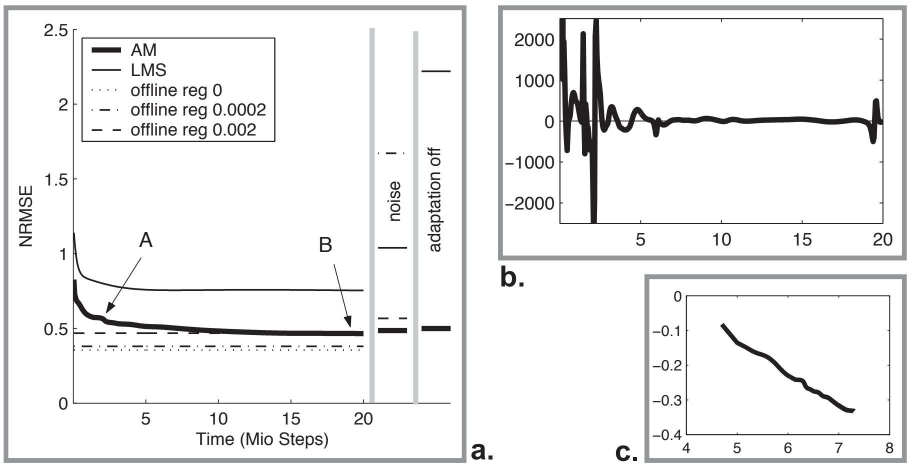

Figure 4: **a.** NRMSE errors of various learning conditions vs. update time. The first set of four short lines offset at the right end indicate performance degradation under noise conditions; the second set of two short lines the performance degradation when adaptation is switched off in LSM and DFD. **b.** Linearly magnified curvature $n \times d\mathrm{NRMSE}^2(n)/dn^2$ of the DFD error development. "A" and "B" mark two aha events. **c.** Loglog (base 10) plot of DFD NRMSE. Time axis spans 20 million update steps in all panels. For explanation see text.

5.1 Learning performance

The learning performance of this DFD was compared with standard, single-reservoir ESNs. The DFD has altogether $3,600$ trainable ESN output weights, 720 per output dimension. In order to compare the training performance with single-reservoir ESNs, the latter should have a similar number of adjustable weights, that is, 720 per output dimension, which would mean a reservoir with 720 neurons. Due to hardware limitations (256 MB RAM was insufficient for standard offline ESN training on 50,000 data points) I resorted to a single-reservoir ESN with only 500 neurons, which could just be hosted on my machinery. Thus the following comparison is slightly biased in disfavour of standard ESNs. The 500-unit ESN was set up with in analogy with the first-level reservoir in the DFD, that is, with a spectral radius of 0.2 and a leaking rate of 1.0 (which amounts to a non-integrating behaviour). Output weights were computed in four different fashions: (i), using the online adaptive LMS algorithm with a learning rate of 0.01 (same as in the DFD);

(ii) – (iv) computing the 5×505 output weight matrix $\mathbf{W}^{\mathrm{out}}$ by the regularized Wiener-Hopf solution of the linear regression task

$$\mathbf{W}^{\mathrm{out}} = \left((\mathbf{R} + \alpha^2 \mathbf{I})^{-1}\, \mathbf{d}\right)^{\mathsf{T}}, \tag{24}$$

where $\mathbf{R}$ is the 505×505 correlation matrix of the 48,000 reservoir-plus-input-states $[\mathbf{x}(n); \mathbf{u}(n)]$ obtained by driving the reservoir with the training sequence ($n = 2001, \ldots, 50,000$; states from the first 2000 time steps discarded to accomodate for initial reservoir transients); α is the Tikhonov regularizing coefficient; $\mathbf{d}$ is the 505×5 target matrix $\mathbf{d} = \mathbf{X}\,\mathbf{U}^{\mathsf{T}}$, where $\mathbf{X}$ contains the 48,000 reservoir-plus-input-states $[\mathbf{x}(n); \mathbf{u}(n)]$ as columns and $\mathbf{U}$ contains the 48,000 next-outputs $\mathbf{u}(n+1)$ as rows. The conditions (ii) – (iv) were distinguished by using regularizers $\alpha = 0.0; 0.0002; 0.002$ respectively.

The thin solid line in panel **a.** shows the NRMSE development obtained by LMS adaptation (like in the DFD condition, the NRMSE was estimated from the last 1000 steps in the 50,000 step training sequence; the adaptation was not switched off during the NRMSE assessment). After an initial period of rapid improvement, the learning rate becomes stuck in an extremely slow exponential mode which is due to the enormous eigenvalue spread of standard ESNs (see [5] for a textbook account of these LMS properties). The learning rate of 0.01 turned out to be close to the admissible maximum; with a learning rate of 0.02 the LMS algorithm became unstable. Thus the LMS adaptation of a standard ESN, in this case again like always before, turns out to be useless.

The dotted, dotted-dashed, and dashed lines mark the NRMSE of the 500-unit ESNs which were trained offline in conditions (ii) – (iv). The NRMSE was computed on the last 1000 steps of the 50,000 training sequence, the same evaluation subsequence used to assess the progress in the DFD and LMS conditions. The regularizing coefficient $\alpha = 0.002$ leads to an accuracy which is almost identical to that of DFD after 20 Mio steps. The $\alpha = 0.0$ accuracy of NRMSE ≈ 0.355 – which is arguably close to the optimal NRMSE achievable with any method – gives a baseline for optimality; the DFD accuracy after 20 Mio steps of approximately 0.465 misses this virtual optimum by about 25%.

An important quality criterium of system modelling is noise robustness – both with regard to signal noise (in the input signal which has to be predicted) and to system noise (internal to the model; important for analog hardware, low-precision arithmetics, or biological modelling). To assess the susceptibility to noise, the DFD and the four comparison models (i) – (iv) were tested in a condition where both the input signal and the various ESN state vectors were corrupted by additive noise sampled uniformly from $[-0.005, 0.005]$. For the two adaptive models (DFD and LMS), the adaptation was not switched off during this test. The short lines offset at the right of panel **a.** show the resulting NRMSE, again estimated from the last 1000 steps. The LMS-adapted model degrades to chance output (a NRMSE of about 1 means complete loss of correlation between teacher and model); the medium-regularized condition (iii) exhibits a degradation to a NRMSE of about

1.7 (which in addition to loss of correlation means a global scaling error); the sharp condition (iv) ($\alpha = 0.0$) jumped to a NRMSE of more than 120 and is not plotted; the cautiously regularized condition (ii) ($\alpha = 0.002$) degrades to about NRMSE ≈ 0.567. The winner in this robustness game is the DFD model which deteriorates only slightly from 0.465 to about 0.485.

Another interesting aspect in comparing DFD vs. LMS is the different behaviour when the adaptation is switched off. For both models this was done after step 20 Mio (that is, learn rates were set to zero). The second set of short lines in panel **a.** shows that the LMS-trained model is completely destroyed (NRMSE under zero adaptation jumps to 2.22), while the DFD model suffers only a mild drop in accuracy (from 0.465 to 0.499). This indicates the the LMS model relied heavily for its accuracy (of however only a NRMSE of approx. 0.75 after 20 Mio updates) on an ongoing adaptation: a closer inspection of the LMS-trained model (omitted here) would reveal that the model uses weight adaptation especially to accomodate to the constant-value-episodes in the input signal. In contrast, the DFD model has "really learnt" important properties of the input signal.

All in all, compared to other models, the online adaptation of this DFD model exhibits a useful and unique combination of properties:

- The computational cost per update (equations (21) and (22)) is linear in the number of adaptable parameters, as it is for LMS.
- The convergence is apparently not encumbered by the slow mode dominance which plagues the LMS adaptation of standard ESNs. A performance level which falls off from the assumed optimum by about 25% is reached after a feasible (if admittedly long) training time of 20 Mio steps.
- The DFD model is much more robust to state and signal noise than offline-trained ESN models or LMS-trained models.

It remains of course to be seen whether this favourable impression is repeated in other case studies, – and what can be ascertained by mathematical analysis. Furthermore, a comparison with a model obtained by RLS adaptation would be of interested (not done so far).

5.2 The Aha! effect

If the DFD learning curve from figure 4**a.** would be plotted larger and with a better resolution, one could see that the decrease of NRMSE proceeds in a sequence of rather sudden drops which are followed by longer periods where the error improvement is slower. Panel **b.** renders the curvature of the NRMSE; since the curvature becomes very small for larger n, it was scaled by n for the plot. Episodes of NRMSE drops become visible in this curvature plot as occasions where the curvature briefly dives into negative values.

There are altoghether six such events which are clearly discernible in the curvature plot. Each of them except the first one is connected to the "discovery" of new features in the top level of our DFD architecture. I will first describe what happens in the episode marked "A" in figure 4**a.** and **b.**.

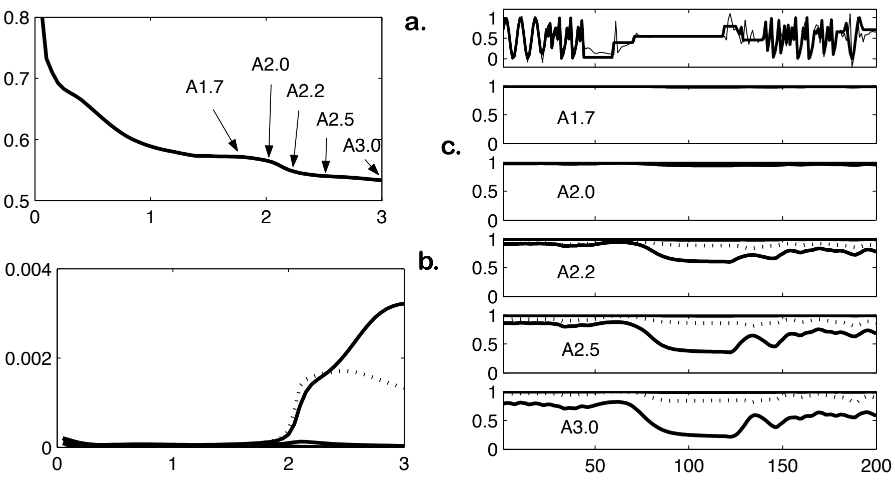

Figure 5: **a.** Closeup on the NRMSE of the DFD model for the first 3 Mio steps. Arrows indicate locations of the snapshots given in panel **c.**. **b.** Development of the 3rd-layer error signal $E_3(n)$ ((21)). **c.** Top graph, thick line: a short subsequence of the original one-dimensional signal $\tilde{u}(n)$. Top graph, thin line: the prediction $\hat{\mathbf{u}}(n)$ of the DFD version obtained after 1.7 Mio steps (retransformed to a one-dimensional $\hat{\tilde{u}}(n)$. Other graphs show the vote vector R3.$\mathbf{v}(n)$ plotted in for the same short subsequence as in the top graph. They were recorded after training times 1.7, 2.0, 2.1, 2.5, 3.0 Mio. For explanation see text.

Figure 5 collects diagnostic graphics from that episode. Panel 5**a.** gives a closeup of the NRMSE of the DFD model for the first 3 Mio steps, clearly exhibiting the relatively sudden "dip" in the NRMSE that is connected to feature discovery events. Here this dip occurs roughly between time steps 2.0 and 2.2 Mio. Panel 5**b.** plots the ten components of the third-level error $E_3(n)$ ((21)). Eight of these components remain too close to zero to become visible in this plot; two components take a sudden rise around time $n = 2$Mio. Panel 5**c.** presents a number of snapshots of the vote vector R3.$\mathbf{v}(n)$, recorded for a 200-step sequence after training times 1.7, 2.0, 2.1, 2.5, 3.0 Mio. The top graph shows the section of the original one-dimensional signal $\tilde{u}(n)$ which was used for this 200 steps (thick line), together with the prediction $\hat{\mathbf{u}}(n)$ of the DFD model obtained after 1.7 Mio steps (re-transformed to a one-dimensional $\hat{\tilde{u}}(n)$). The other graphs in panel 5**c.** show the 10-dimensional vote vector R3.$\mathbf{v}(n)$ plotted for the various training stages. Eight of the components of R3.$\mathbf{v}(n)$ are so close to 1 throughout that they are not visible at the given resolution. Two components however undergo a rapid differentiation away from a constant value of about 1 after training time 2.0 Mio.

These are the same two components whose corresponding errors in $E_3(n)$ exhibit the sudden increase.

More "aha" episodes similar to this one occur around times 6 Mio and 19.5 Mio. They are centered in time at single oscillations visible in figure 4**b.** In each of these events,

- one further among the 10 components of R3.$\mathbf{v}(n)$ becomes "recruited" and differentiates from an constant-value-1 signal to a time-varying signal;
- simultaneously, the corresponding error component in $E_3(n)$ takes a sudden rise, to fall off slowly thereafter;
- the overall prediction NRMSE plotted in figure 4 takes a drop (whose magnitude however decreases roughly in inverse proportion to n).

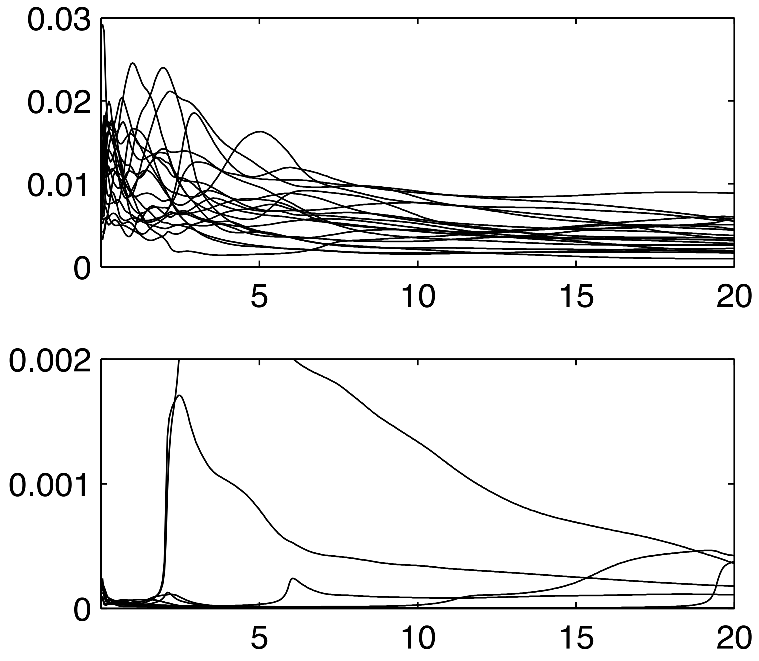

Figure 6: Development of the 20-dimensional error signal $E_2(n)$ (top) and the 10-dimensional $E_3(n)$ signal (bottom) during the 20 Mio training run. For explanation see text.

Figure 6 (bottom) shows the development of the third-level error signal $E_3(n)$ during the 20 Mio training run. Figure 7 depicts the votes R3.$\mathbf{v}(n)$ generated by the model obtained at the termination of the run. At this time, five from the ten components of R3.$\mathbf{v}(n)$ and $E_3(n)$ have developed a non-constant dynamics, differentiating away from their initial constant values of close to 1 (votes) or close

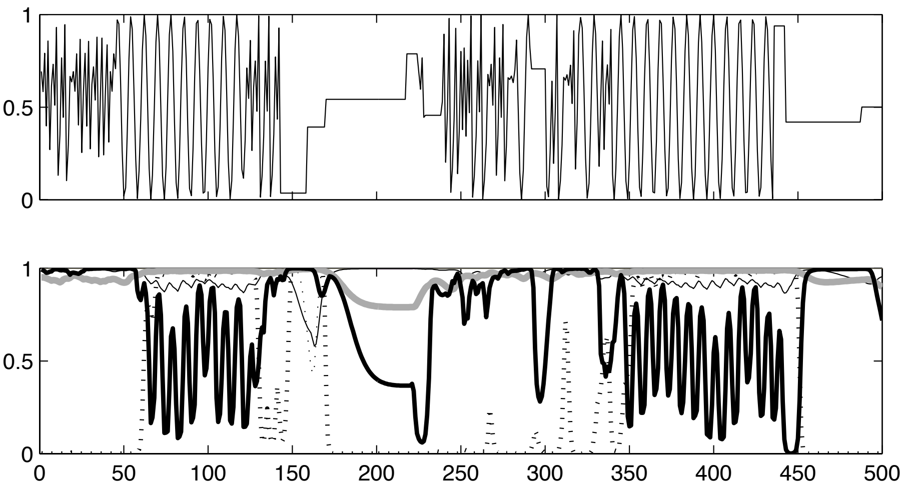

Figure 7: Top panel: a 500 step subsequence of the original one-dimensional signal $\tilde{u}(n)$. Bottom: the five differentiated third-level votes from R3.$\mathbf{v}(n)$, recorded from the 20 Mio step trained model on the input shown in the top panel. For explanation see text.

to 0 (errors). Most of these differentations-away-from-constant occurred in temporally sharp events; only one of the non-constant votes visible in figure 7 acquired its interesting behaviour relatively slowly (in an interval from about $n = 14$ Mio to about $n = 18$ Mio, compare the slowly rising curve in figure 6, (top)).

Furthermore, there is one oscillation visible in figure 4 **b.**, at about time 4 Mio, which is not accounted for by a differentiation of a layer-3 vote. This drop in NRMSE may be tentatively attributed to the differentiation of one layer-2 vote which is reflected in a rise of an isolated component of the error $E_2(n)$ around time 4 Mio (visible in figure 6 top). The majority of votes R2.$\mathbf{v}(n)$ in the second layer differentiate, with much temporal overlap, during the early training stages before $n = 2$ Mio (again visible in figure 6 top).

The differentiated level-3 voting signals are related to the succession of the three types of generators that make up the input $\tilde{u}(n)$. Inspecting figure 7, one can find, for instance, that the vote drawn by the bold dotted line behaves almost like an indicator for the slow sine generator; the vote drawn by the bold black line largely follows the slow sine input when it is the current generator, while it rises to close to maximum values when the iterated tent map generator is in command; the other votes are not so clearly to characterize. It may be worth pointing out that one does *not* find a clear "segmentation voting", that is, one doesn't observe that for each of the three generators an indicator vote develops – which one might have hoped.

Finally, I want to point out a remarkable decorrelation phenomenon. As pointed out in the Introduction, transforming an input signal into decorrelated nonlinear transform signals was a major original motivation for me to develop this architecture.

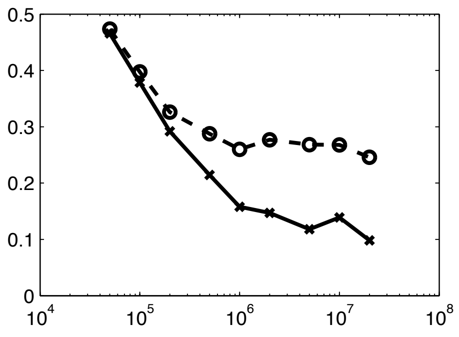

Figure 8: Development of the correlation measure c_{ij} (circles) and c'_{ij} (crosses). Time axis is logarithmic. Measurements were taken from models trained at times 50K, 100K, 200K, 500K, 1Mio, 2Mio, 5Mio, 10Mio and 20Mio. Each plotted point represents the mean absolute value of all off-diagonal entries in the 20×20 matrix (c_{ij}) or (c'_{ij}) (diagonal entries c_{ii}, c'_{ii} measure autocorrelation and have unit values).

The desired decorrelation effect is obtained between the features. Consider the first-level features R1.$\mathbf{f}_i(n)$ ($i = 1, \ldots, 20$). To compute a numerical estimate of the (de-)correlation between two such feature vectors (in our example each of length 5), each feature was recorded within the training run for the last 1,000 steps in each 50,000 cycle (like other diagnostics described earlier). For each feature R1.$\mathbf{f}_i(n)$ and every diagnostic recording of length 1,000 this yielded a feature value matrix $\mathbf{F}_i$ of size 1000×5. These matrices were first normalized to Frobenius norm 1 (by division of $\mathbf{F}_i$ with $\sqrt{\sum_{\nu=1\ldots1000,\,\mu=1\ldots5} \mathbf{F}_i^2(\nu,\mu)}$). Then, to obtain a correlation c_{ij} between the feature signals R1.$\mathbf{f}_i(n)$ and R1.$\mathbf{f}_j(n)$ ($n = 49001, \ldots, 50000 \mod 50000$), c_{ij} was calculated by the following inner matrix product between $\mathbf{F}_i$ and $\mathbf{F}_j$:

$$c_{ij} = \mathbf{1}_{1000}^{\mathsf{T}} \, (\mathbf{F}_i . * \mathbf{F}_j) \, \mathbf{1}_5, \tag{25}$$

where $.*$ again denotes element-wise multiplication and $\mathbf{1}_d$ the all-ones column vector of dimension d. A similar correlation measure c'_{ij} was also computed in a version where the computation started not from features R1.$\mathbf{f}_i(n)$ but from the vote-weighted features R1.$v_i(n)$R1.$\mathbf{f}_i(n)$. Figure 8 shows the evolution of these two correlation measures during the training run. It becomes apparent that the vote-weighted features decorrelate almost completely in the course of the training run, while the unweighted features decorrelate only to a lesser extent.

6 Discussion

6.1 Summary

The take-home points from this report are the following:

- DFDs build on a feature-vote concept where features have the same dimension as the signals they are features of, and vote vectors can be seen as abstract representations of the feature dynamics (which gives rise to hierarchical stacking).
- DFDs are trained by stochastic gradient descent on the next input prediction task. However, it is not the actual prediction output which is the learning target of interest, but the hierarchy of features and votes which evoloves in the process.
- These features and votes apparently (as far as one can say after one simple case study) exhibit striking orthogonality properties, which recommends them as inputs for cheap LMS-type learning algorithms.
- The appropriate background intuition for DFDs is not one of representational efficiency (as in most if not all other approaches) but one of unfolding, *explication* of a complex input signal into a rich spectrum of informative dynamical features.
- With all caution, the reported case study suggests that DFD learning leads to models with superiour noise robustness (compared to offline or LMS training of standard ESNs).
- Still with due caution, a functional role of higher-level features may be to facilitate (= speed up) the learning of the next input prediction task, i.e. the learning of a predictive state representation.

6.2 Related work

This technical report presented an early stage of development of a rather complex learning architecture, which was checked so far only on a rather simple synthetic dataset. It is therefore too early for any quantitative evaluation of its performance or a comparison with other approaches. Some qualitative comments however are in place to highlight essential points that distinguish the DFD from other hierarchical dynamical learning architectures.

Hierarchical hidden Markov models (HHMMs). HHMMs [23] are hierarchic in the sense that an input sequence is segmented into segments, which in turn are segmented into subsegments, etc., up to a predefined order. An illustrative example is the segmentation of a text into phrases into words into syllables into letters. In contrast, the features extracted by a trained DFD need not have a segment-subsegment temporal relationship as one descends one level; features within one level may temporally overlap; the votes can gradually blend features in and out without sharp segmentation boundaries. This point made, I add that one *may* find in higher-level votes indicators for segmentation (as in figure 7). Another obvious and fundamental difference between HHMMs and DFD is that the former yield a rigorous model of a process distribution, while the latter are not a statistical model at all.

Hierarchical mixtures of experts (HMEs). HMEs are originally and mostly described as models of static distribution [15], but lend themselves also to dynamical tasks when the expert and gating subsystems are implemented by HMMs or RNNs ([27]). A variety of HME variants has evolved, some more of a statistical and others more of a neuro-dynamical flavour. The all share however three characteristics which sets them apart from DFDs: (i) higher (gating) levels receive the same external input signal as the lowest (expert) level, while in DFDs higher processing levels receive higher-level input which is distilled in levels below; (ii) by design, the hierarchy in HMEs is tree-like, with higher levels having (exponentially) fewer gating modules than lower ones, while in DFDs there is no rule that ties the "width" of a layer to its height in the hierarchy, (iii) since HMEs are set up as mixture distribution models, the gating weights within one hierarchical level must sum to unity, while in DFDs the votes of one level don't have to satisfy a sum constraint.

Deep Belief Networks (DBNs). DBNs are hierarchically stacked, restricted Boltzmann machines and inherit from the latter the statistical physics background of associating a distribution with an energy function and of learning a distribution with energy minimization. DBNs are arguably the currently most powerful proven machine learning architecture for learning complex static distributions and may eventually claim the same for distributions of processes. A unique feature of DBNs which distinguishes them from HHMMs, HMEs and DFDs alike is the reliance on binary neurons (which have their origin in the spin lattice models of statistical physics). The most commonly (only?) used learning algorithm, *contrastive divergence* [7] also is unique to DBNs. DBNs share with DFDs that higher levels do not necessarily have to have fewer features than lower ones (although all work on DBNs accessible to me uses "thinning-out-upwards" structures). It is not immediately obvious how different timescales can be implemented in different DBN levels, but an approach may be to provide bias for not changing a neuron state in order to engender slower dynamics (Y. Bengio, personal communication).

Furthermore, DFDs are different from all the mentioned other models in their conception of a feature as a signal with the same dimensionality as the external input.

6.3 Further research

There are so many open questions and urgent next steps that it is even difficult to arrange them in an orderly overview. Here is a unordered list of natural next steps:

- More simulations on more complex datasets are needed to see whether the observations made here, especially the aha effect, persist.

- The architecture, as presented in this report, has many features which are accidental and result from ad hoc design decisions. Examples are the leaky integration and squashing of the votes, or the type of neurons employed in the reservoirs, or the type of input which is propagated bottom-up to higher-level reservoirs. The only design characteristic which I consider fixed (and defining for this approach) is the multiplication of trainable feature outputs with voting vectors to give the outputs of one layer, where the voting vectors are the outputs of the next higher level in turn. So there is a large space of design options to be explored.

- While in this report I have exclusively focussed on temporally multiscale properties, there are also spatially multiscale data which could be tackled with the presented architecture (e.g., videosequences), or even mixed temporal/spatial multiscale data. This will require significant adjustments of the architecture.

- Granularity, whether temporal or spatial, is not the only cause for hierarchicity. Another prime motif for hierarchic representations in AI is *abstraction.* Whether DFDs can be geared towards, or interpreted as, abstraction hierarchies remains to be investigated.

- The architecture has so many tuning parameters (learning rates, time constants, leaking rates, spectral radii of reservoirs, input scalings, dimensions...) that a manual optimization will be infeasible except for the coarsest adjustments. Automated adaptation rules for these tuning parameters are a necessity. As far as the core control parameters for the reservoirs are concerned, current research in reservoir computing has some approaches to offer which should be checked out (see [21] for a survey). As to the larger picture, my perception is that principles for adaptation/tuning of control parameters in complex learning architectures will become an interdisciplinary research area of its own standing, connecting neuroscience with machine learning, statistics, and dynamical systems theory.

- The original motivation to develop this architecture was to transform an input signal into a set of nonlinear transforms with nice decorrelation properties. The latter are important for using the LMS algorithm and for noise tolerant yet precise models. It remains to investigate whether the presented architecture comes close to this target. Concretely, for some complex supervised training task (like dynamic pattern classification) one would first train the presented architecture on the task's input signal. After training, when the voting signals in higher levels are well differentiated, one would use the vote-weighted features $\mathrm{R}k.v_i(n)\mathrm{R}k.\mathbf{f}_i(n)$ as input signals for an LMS (or any other linear regression) algorithm, just as in standard ESNs one uses the reservoir signals.

- The "aha" dynamics, the most intriguing phenomenon in this architecture, needs to be analysed theoretically.
- From figure 6 it appears that the error $E_3(n)$ which drives the adaptation in the third layer is smaller than the second-layer error signal $E_2(n)$ by an order of magnitude. This may foreshadow a serious problem of error gradients which vanish as one climbs up the layers of this architecture. A theoretical analysis is called for (in conjunction with analysing the aha effect).
- Humans, as social beings, do not learn everything they know in an unsupervised way. Arguably, *most* of the concepts hosted by a modern human are acquired in a supervised / imitation-driven way. Complex hierarchical learning architectures for multiscale data, like the one presented here or others [2], ultimately aim at the complexity levels of human intelligence. In my opinion this target can possibly only become reachable if the learning system is allowed to acquire "ultra-informative" priors by direct teach-in. With respect to the architecture sketched in this report, this may for instance mean that some higher-level features are trained in a supervised way with large learning rates, in relatively short training episodes interspersed in the "training lifetime" of a large and enduring learning system.

References

[1] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] Y. Bengio and Y. LeCun. Scaling learning algorithms towards ai. In Bottou L., Chapelle O., DeCoste D., and Weston J., editors, *Large-Scale Kernel Machines*. MIT Press, 2007.

[3] R.A. Brooks. New approaches to robotics. *Science*, 253:1227–1232, 1991.

[4] G.A. Carpenter and S. Grossberg. ART 3: Hierarchical search using chemical transmitters in self- organizing pattern recognition architectures. *Neural Networks*, 3(2):129–152, 1990.

[5] B. Farhang-Boroujeny. *Adaptive Filters: Theory and Applications.* Wiley, 1998.

[6] P. Greenfield. Language, tools and brain: The ontogeny and phylogeny of hierarchically organized sequential behavior. *Behavioral and Brain Sciences*, 14:531–595, 1991.

[7] G. Hinton, S. Osindero, and Teh. Y. W. A fast learning algorithm for deep belief networks. *Neural Computation*, 2006.

[8] G. E. Hinton and R. R. Salakuthdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(July 28):504–507, 2006.

[9] G.E. Hinton, P. Dayan, B.J. Frey, and R.M. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161, 1995.

[10] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001. http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf.

[11] H. Jaeger. Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 593–600. MIT Press, Cambridge, MA, 2003. http://www.faculty.iu-bremen.de/hjaeger/pubs/esn_NIPS02.

[12] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304:78–80, 2004. http://www.faculty.iu-bremen.de/hjaeger/pubs/ESNScience04.pdf.

[13] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20(3):335–352, 2007.

[14] H. Jaeger, M. Zhao, K. Kretzschmar, T. Oberstein, D. Popovici, and A. Kolling. Learning observable operator models via the es algorithm. In S. Haykin, J. Principe, T. Sejnowski, and J. McWhirter, editors, *New Directions in Statistical Signal Processing: from Systems to Brain*, chapter 20, pages 417–464. MIT Press, 2006.

[15] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[16] A.H. Klopf, J.S. Morgan, and S.E. Weaver. A hierarchical network of control systems that learn: Modeling nervous system function during classical and instrumental conditioning. *Adaptive Behavior*, 1(3):263–319, 1993.

[17] J. Kohlmorgen, K.-R. M/"uller, J. Rittweger, and K. Pawelzik. Identification of nonstationary dynamics in physiological recordings. *submitted to Biological Cybernetics*, 83(1):73–84, 2000.

[18] A. U. Kü cükemre. *Echo State Networks for Adaptive Filtering.* Master thesis, University of Applied Sciences Bonn-Rhein-Sieg, 2006.

[19] Y. LeCun, L. Bottou, J. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf.

[20] M. L. Littman, R. S. Sutton, and S. Singh. Predictive representation of state. In *Advances in Neural Information Processing Systems 14 (Proc. NIPS 01)*, pages 1555–1561, 2001. http://www.eecs.umich.edu/∼baveja/Papers/psr.pdf.

[21] M. Lukosevicius. Overview of reservoir recipes. Technical Report 8, School of Engineering and Science, Jacobs University Bremen, 2007. to appear.

[22] A.W. Moore, L.C. Baird, and L. Kaelbling. Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. In Th. Dean, editor, *Proc. IJCAI-99, Vol. 2*, pages 1316–1323. Morgan Kaufmann, 1999.

[23] K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2001. http://www.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/.

[24] K. Pawelzik, K.R. Müller, and J. Kohlmorgen. Divisive strategies for predicting non-autonomous and mixed systems. Arbeitspapiere der GMD 1069, GMD, Sankt Augustin, 1997.

[25] L. Shastri. Advances in shruti – a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Artificial Intelligence*, 11:79–108, 1999.

[26] I. Sutskever and G. Hinton. Learning multilevel distributed representations for high-dimensional sequences. Technical Report UTML TR 2006-003, Department of Computer Science, University of Toronto, 2006.

[27] J. Tani and S. Nolfi. Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks*, 12(7-8):1131–1142, 1999.

[28] G. W. Taylor, G. E. Hinton, and S. Roweis. Modeling human motion using binary latent variables. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19.* MIT Press, Cambridge, MA, 2006.

[29] T. Tyrrell. The use of hierarchies for action selection. *Adaptive Behavior*, 1(4):387–420, 1993.

[30] D.M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7-8):1317–1330, 1998.