MEM scales | Memory technologies with multi-scale time constants for neuromorphic architectures

Work Package WP1

Theory, algorithms and architectures

Deliverable D1.2

Report on options for the computational extension of physically obtained timescales

Funding Instrument:	Research and Innovation Action
Call:	H2020-ICT-2019-2
Call Topic:	ICT-06-2019 Unconventional Nanoelectronics
Project Start:	1 January 2020
Project Duration:	42 months
Beneficiary in Charge:	UOG

Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the Consortium (including the Commission Services)	
CO	Confidential, only for members of the Consortium (including the Commission Services)	

Deliverable Information

Document Administrative Information	
Project Acronym:	MeM-Scales
Project Number:	871371
Deliverable Number:	D1.2
Deliverable Full Title:	Report on options for the computational extension of physically obtained timescales
Deliverable Short Title:	Computational Timescales Extension
Document Identifier:	MeM-Scales-D12-ComputationalTimescalesExtension-draft-V1.0-V1.0
Beneficiary in Charge:	UOG
Report Version:	V1.0
Contractual Date:	31/12/2021
Report Submission Date:	
Dissemination Level:	PU
Nature:	Report
Lead Author(s):	Herbert Jaeger, Francky Catthoor
Co-author(s):	
Reviewer(s):	Francky Catthoor, Giacomo Indiveri
Keywords:	Timescales, memory hierarchy, slow-fast systems
Status:	_x_ draft, __ final, __ submitted

Change Log

Date	Version	Author/Editor	Summary of Changes Made
04/09/2021	V0.1	UOG	start of writing
02/01/2022	V1.0	UOG	final draft ready for review

Table of Contents

Executive Summary

In this report we inspect the challenge of “computational extension of physically obtained timescales” in detail. To place the discussion on a solid ground, we first identify three main theoretical layers of describing general computing systems: the physical hardware, the abstract computational model which formalizes the “algorithms”, and in between an analytical model which formalizes the physics of the hardware. This framework greatly differentiates an existing theoretical account of what “computing” means for general (digital or unconventional) physical hardware, and our detailed picture allows us to formulate the problem of computational timescale extension in a differentiated and meaningful way. We explain several ways how temporal progression of computations can be formalized, and identify three phenomenal aspects of “timescales” which need to be considered separately when one tries to find algorithmic solutions to temporal processing tasks. On the background of these theoretical considerations we survey twenty formal, algorithmical, and heuristic-practical techniques for analysing, transforming, and exploiting timescale phenomena on the two layers of the analytical and/or the abstract computational models.

1 Introduction

To the degree that neuromorphic hardware systems are truly brain-like — and thus to the degree that they can capitalize on the brain’s operating principles with regards to its astounding energy efficiency, robustness and data bandwidth — they are not *digitally simulating* brains but *physically instantiating* neural mechanisms. This line of reasoning has led to the worldwide surge of academic and industry research in memristive devices for physical in-memory computing in particular, and more generally of research in spiking analog neurochips, and even beyond that, of research in *physical computing* in general — also known under names like unconventional computing, in-materio computing etc. — a field where materials science connects with computing science to explore how any kind of nanoscale nonlinear physical effect can become harnessed for computation [Adamatzky, 2017, Stepney et al., 2018].

Numerous hurdles still have to be overcome to develop neuromorphic or physical computing to a mature and widely useful engineering discipline, for instance

- the fact that device mismatch, process variation and aging make it impossible to fabricate functionally identical clones of unconventional microchips,
- the fact that unconventional computing systems cannot be programmed in the classical sense but will have to be configured, trained, or evolved, which requires new views on how to use such systems for what tasks,
- the fact that the on-board processing on unconventional systems can only be physically measured in a small finite number of observables whereas such systems will typically be mathematically infinited-dimensional, a circumstance which hinders systematic testing and debugging of algorithmic procedures (as opposed to the digital domain where all bit operations can be monitored),
- the fact that thermal or quantum noise and temperature sensitivity will lead to low, variable, or even ill-defined numerical precision, which will require the development of new mathematical formats for representing or quantifying information,
- or the fact that unconventional systems will age or become physically transformed during

their lifetime by learning and adaptation, which implies that they cannot be “re-started” or “re-set” to some well-defined initial state.

Overcoming such hurdles will require from us to develop new intuitions about, and formal models of, “computing”. This endeavour can only be mastered through an interdisciplinary effort between the materials sciences, microsystems engineering, physics, theoretical computer science, complex systems research, mathematics and neuroscience. A comprehensively embracing new engineering science of unconventional computing still is a distant goal [Jaeger, 2021]. But, having this distant goal in the back of our minds, we can address limited segments of this challenge in productive ways.

In this deliverable we address one segment of one of the big hurdles that stand in our way, a hurdle that we did not yet mention in the listing above. It is the hurdle to which MemScales is centrally committed: coming to terms with multiple timescales. We have described in our project proposal and our deliverable D1.1 why neuromorphic or more general unconventional computing systems must become enabled to operate on several timescales. We will not redraw this “big picture” here, which includes aspects of materials and devices, circuit design, algorithms and mathematical modeling. Here we focus on the algorithmical aspect, which we will call the *computational timescale extension (CTE)* challenge:

Given that a computing system can physically realize only a limited range of physical timescales but many tasks need a wider range, how can the physically available timescales be extended by purely computational mechanisms?

This report has two main sections, followed by a summary and discussion. The first main Section 2 investigates CTE from a methodological angle, clarifying the concept in the first place. The second main Section 3 surveys CTE mechanisms that we are aware of at this time.

2 Theoretical considerations

Computational timescale extension is a not very well-defined umbrella term for a number of phenomena and techniques. Before we can discuss CTE in a meaningful way we must work out our understanding of the underlying conceptions of “computing” and “timescales” and also what we mean by “extension”. In this section we sharpen and differentiate our understanding of the CTE challenge.

2.1 “Computational extension of timescales” can mean many things

The first step is to become aware that “computational extension of timescales” is not a clear-cut single sort of task. There are numerous scenarios and reasons why or how one would want to manipulate timescales in some way or other. Some examples:

- One may wish to use an analog electronics processor, whose physical time constants are very large, in a “slow” online signal processing task which requires long memorizing and integration times of incoming information. This is one of the starting motivations for the MemScales project which we inherited from the predecessor project NeuRAM3.
- In online signal processing tasks, one may need to adapt the “speed” of the processing system to faster or slower versions of input signals.

- In a speech recognition task, one needs a hierarchy of short-term and working memory timescales that range from the millisecond range of phonemes to syllables to words to phrases to sentences to textual contexts that in turn range from a few sentences to lines of argument to an entire narrative. Furthermore, the processing of most of these intermediate timescales one needs to access information that is stored in a (more or less) permanent long-term memory. Similar multi time-scale support is required for most signal processing tasks, including imaging, audio, video, graphics, sonar, radar etc.
- While memory timescales reach into the past, many cognitive processing tasks likewise require to reach out into the future across several timescales. For example, in control architectures for autonomous mobile robots one needs anticipations of the effect of current actions on the short timescale of avoiding obstacle collision in arm motion generation, as well as on several long timescales of purposeful action planning. This can be extended to a much wider range of industrial control tasks.
- Incoming sensor information or user input can arrive at varying levels of "information density per time unit". In a realtime computer game, the human gamer may at some times do nothing for seconds or minutes, while at other times the gamer may fusillate the game engine with volleys of joystick commands. In those periods of high input density the engine must in some way process the incoming information faster than when the user is idling. Similar high variations in computational load occur in many other applications, for instance in fault monitoring or online decision support systems, and for a broad range of body-area-network oriented sensors which are vital for the future healthcare paradigm shift.
- So-called *anytime* algorithms in classical digital computing are designed to output possibly inaccurate results fast when immediately needed, but would continue processing the task and generate better results when given more time. "Cognitive" processing procedures in non-digital computing systems might perform similarly by running several processes in parallel and in different subsystems, fast ones for approximate results and slow ones for more deeply "thought-out" results. This may be related to the common idea in cognitive architecture research that fast responses are obtained from a first "feedforward" bottom-up pass through a neural processing hierarchy, while on a longer timescale top-down pathways become activated which lead to recurrent "deliberations" about an appropriate system response (as for instance in adaptive resonance theory [Grossberg, 1976a,9]).

This ad hoc list of scenarios illustrates that there are many sorts of demands for timescale management — there is not a single generic problem of "timescale extension", but a whole spectrum of such problems. The specific requirements are moreover highly application- and context-dependent which makes it even harder to derive a solid theoretical basis spanning this broad domain.

2.2 When does a physical system compute?

In order to get a clean and practically useful understanding of what "computational extensions of timescales" can actually mean, we first have to understand what we mean when we say that a physical system "computes". In this subsection we thus discuss the question "when does a physical system compute?"

This question is in fact the title of the foundational paper by Horsman et al. [2014]. The authors analyze the conditions when and in what sense one can claim for a physical system that

it “computes”. In short, their main conclusion is that “computation” is defined in an abstract formal-mathematical sphere (for instance framing computing as sequence of Boolean operations or as the update rules of a Turing machine). Inputs and outputs for computations are defined in this abstract sphere (for instance as a sequence of 0’s and 1’s in digital computing, or as real-valued timeseries in analog signal processing). In order to link this abstract casting of “computing” to an underlying physical machine, the abstract, formal inputs u and outputs y must be mapped to the physical signals that enter and leave the machine.

Horsman et al. call this mapping the *representation* relation between the physical and the abstract domains. Importantly, the representation relation is neither established by the physical machine, nor is it part of the abstract model of computing. Instead, the representation is effected by the external user of the system. This user calls upon practical judgement based on a shared community consensus to warrant that the physical I/O machinery (like keyboards, screens, oscilloscopes) realize the representation mapping in an acceptable way. The external user in which the representation becomes incorporated is almost always a human, but it could in principle also be another intelligent agent (animal or even a future machine). Horsman et al. call such agents *computational entities* and define them to be *the physical entities that locate the representation relation*. We will use the more common term “user”.

It happens within the user that, when a computer prints “the solution is x = 2” on its screen, this physical signal becomes *decoded* to the abstract output format (here the integer 2). This decoding involves epistemological conditions which philosophers find hard to understand, including

- the neural instantiation of the abstract model (here: the integers) in the user’s brain (or is it the cognitive instantiation in the user’s “mind”? don’t start asking philosophers!),
- and social processes of communication among all potential users of the system to reach a consensus that this visual reading of screen outputs is an acceptable and universally shareable decoding procedure.

While we all have become used to the various ways that inputs and outputs are given to and taken from digital machines, to a degree that none of us will question these procedures, the situation gets less clear when it comes to unconventional new kinds of computing machines. For example, when a slime mold in a petri dish grows branches that extend into higher-concentration areas of a nutrient (a popular model of unconventional computing with a substantial literature [Adamatzky, 2018]), how can one read out an abstract output like a Boolean “True” from this physical system?

Horsman et al. [2014] complete this account by assuming that the abstract domain includes some model (not further specified by the authors) of an abstract dynamical process which transforms the input into the output, which together with the physical dynamics leads to a diagram with four arrows which should commute (Figure 1**A**). That is, when the abstract dynamical model transforms input u to output y (green arrow in the figure), and when the abstract input u is first *encoded* in a physical input signal u^{Ψ}, then physically transformed to a physical output signal y^{Ψ}, which is finally *decoded* to an abstract output $\tilde{y}$ (grey arrows in Figure 1**A**), the abstractly determined output y should be (approximately) equal to the output $\tilde{y}$ obtained via the route through the physical machine.

We emphasize that according to this view of what makes a system “compute”, a computing system necessarily must have some provision to accept input and produce some output. This commitment seems natural enough, but lies in opposition to certain views in theoretical physics where quantum-gravitational theories are developed that attempt to reduce all physical laws at

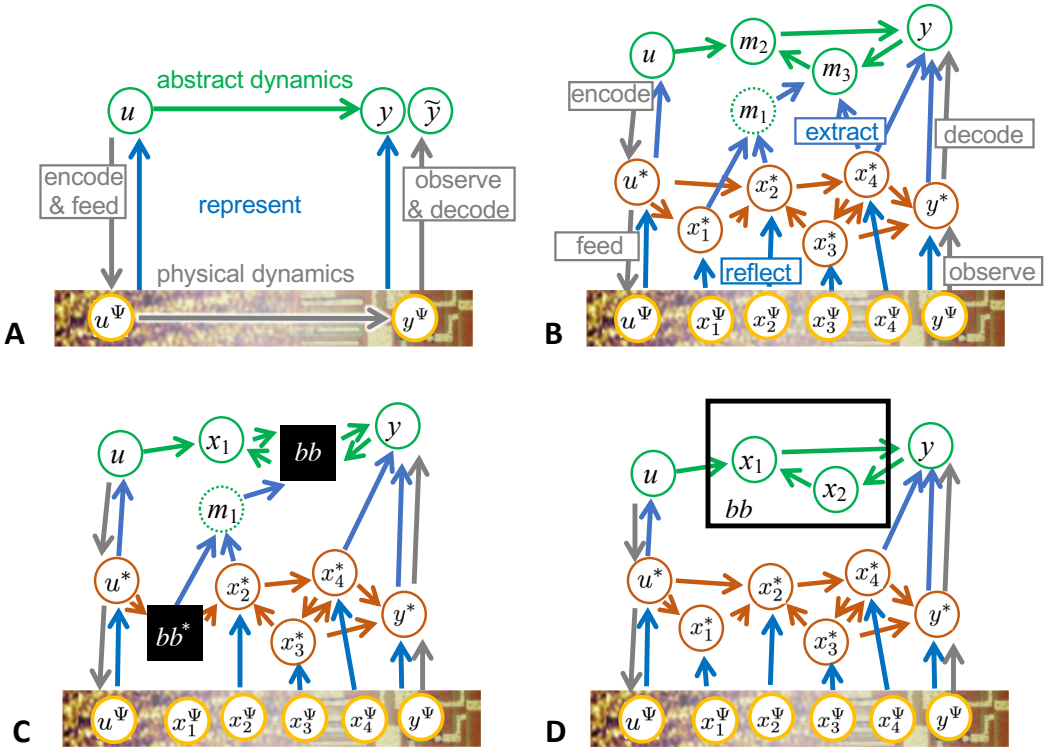


Figure 1: Increasingly complete accounts of a computing system. ***A:*** *The elementary view of Horsman et al. [2014].* ***B:*** *An idealized schema of a more complete modeling structure, comprising an analytical system model which is a reflection of the physical system, on top of which an abstract model of the input-to-output transformation is defined by meta variables that represent possibly complex subprocesses in the analytical model.* ***C:*** *Gaps in the analytical and abstract models can be filled with experimentally obtained blackbox models.* ***D:*** *The entire abstract input-to-output transformation can be modeled with a hand-crafted blackbox model. For explanation see text.*

the most elementary fabric of spacetime as information-processing — the "universe as computer" view [Zuse, 1982,9, Wheeler, 1989, Lloyd, 2013, Zenil, 2013, Deutsch and Marletto, 2015]. We follow Horsman et al. [2014] and other theorists of computing that a closed physical system cannot be said to "compute".

2.2.1 Analytical system model

We find that the account given by Horsman et al. [2014] is too much simplified for our purposes. In engineering and modeling practice, the picture becomes enriched with further elements (Figure 1**B**). The most important addition is an intermediate *analytical system model* inserted between the abstract model and the physical machine. This intermediate model formally reflects the physical processes which transform the physical input u^Ψ into the physical output y^Ψ. The input, intermediate state, and output variables u^*, x_i^*, y^* of the analytical physical model correspond to measurable physical quantities u^Ψ, x_i^Ψ, y^Ψ. The representation relation between the physical quantities v^Ψ (where v means u, x or y) and their analytical correlates v^* is again effected by the human engineer and involves physical measurements, knowledge of physical laws, and community consensus about adequacy criteria (for instance which measurement apparatuses and procedures are admissible according to which accuracy demands).

An analytical system model usually reflects its physical target system bidirectionally in the following sense: each model variable v^* corresponds to a single physical state variable v^Ψ; the model lets its variables evolve with regards to a continuous time variable $v^*(t)$ which is measured in standard physical units like seconds; model variable trajectories $(v^*(t))_{t\in T}$ can be matched against physical state trajectories $(v^\Psi(t))_{t\in T}$ timepoint by timepoint by comparing $v^*(t)$ with physical measurements $\mathcal{M}(v^\Psi(t))$ (and where time itself is measured in the physical world by a physical clock). Model trajectories $(v^*(t))_{t\in T}$ can *predict* physical trajectories $(v^\Psi(t))_{t\in T}$. Such predictions about measurable reality are a critical property of system models in the natural sciences.

The most common mathematical format for analytical system models are ordinary differential equations (ODEs). This is the mathematical language which is typically used in electronic circuit engineering. It is also a formalism of first choice in many models in computational and theoretical neuroscience. At a finer level of model granularity — needed often in materials research, device engineering, or microchip layout — one can also find partial differential equations or finite elements formalisms. Furthermore, stochastic versions of all of these may also be employed. However, in this report we will generally base our discussion on ODE based analytical models.

A crucial property of analytical system models is that the model variables v^* correspond 1-1 to physical quantities (like voltages or currents) which are, at least in principle, measurable. These model variables are quantified in terms of standard physical units of measurement, like Volt or Ampere.

The analytical system model serves a number of helpful functions:

- It is the interface to the physical machine which is used by the developer of abstract computational procedures. Abstract computing procedures are rarely (if ever) matched directly against the hardware — programmers do not normally use oscilloscopes to test or debug their programs.
- It is used by the hardware engineer to design and analyse physical systems.
- It can be used to simulate the physical system on a digital computer. This has become an indispensable routine for hardware developers, and in unconventional computing it is also needed by designers of abstract “algorithms”.

2.2.2 Abstract computational model

The abstract computational model sits on top of the analytical system model. It explains an abstract computation $u \to y$ through a process between intermediate abstract state variables m_j. These abstract state variables are meta variables which are defined by formal derivation from analytical model variables v^*, possibly in cascades with intermediate meta variables, like m_1 in Figure 1**B**. In various contexts, such meta variables are called by names like “features”, “properties”, “characteristics”, “descriptors”, “measures”, “records” or the like.

This picture (as in Figure 1**B**) is still a simplification. In real-world practice one finds significant variations of this idealized scheme, which will become relevant when we later discuss computational timescale extensions:

- The abstract modeling layer often is further sub-structured in layers of increasingly abstract models. A familiar case are compilation hierarchies in digital programming, where at the lowest sub-layers one finds microprocessor instruction sets, then assembler code

at the next higher level of abstraction, followed by further layers that are expressed in increasingly “higher” programming languages, until at the top one may find graphical user interfaces. For computing with (possibly analog) spiking neuromorphic microprocessors, such abstraction hierarchies are beginning to be developed [Zhang et al., 2020].

- In reservoir computing and other fields of unconventional computing, an analytical physical model is often not available. The abstract model of reservoir computing often reduces to a representation of the input signal, followed by an un-modeled “reservoir”, followed by a linear combination which defines the output signal from a number of reservoir-internal observables. The abstract-level “programmer” who wants to solve a computational task must at the same time operate like a physicist and provide and operate signal processing and measurement apparatuses for feeding input signals to the physical reservoir system and reading out the observables that become linearly combined into the abstract-level output signal. The modeling gap between the abstract input and output must be filled by a blackbox model, which in the case of reservoir computing is obtained by machine learning methods (a simple regularized linear regression).
- Generalizing from the reservoir computing example, one can think of abstract models of an input-to-output transformation process whose meta variables are only partly derived from an analytical model, and in which unmodeled process gaps are filled by blackbox models obtained with machine learning methods in experiments with the hardware system. Also the analytical system model may have gaps that are filled with experimentally determined blackbox models. Figure 1**C** gives an impression. A variation of this theme are analytical and abstract models which comprise a complete input-to-output transformation procedure but in which certain parameters have to be determined experimentally.

  Blackbox models are those parts of an abstract model whose variables are not derived as meta descriptors from analytical model variables. Blackbox models can be obtained experimentally by machine learning methods, or they can be hand-designed under the guidance of insight how local input variables that enter blackbox model become transformed to variables that leave the blackbox model, as illustrated in Figure 1**D**.
- While Horsman et al. consider a single user who embodies the representation relation, in real life this embodiment will often be split over several human users each of whom is taking care of realizing only one step in an abstraction hierarchy as in Figure 1**B** — for instance, an electronics engineer takes care of representing a physical machine by an analytical model; a microchip architecture designer creates the next abstraction level with a machine instruction set; and so forth up the abstraction ladder to a web designer creating a webstore interface. All these agents must communicate with at least the next expert below and the next expert above. The further such mutual understanding reaches out across levels, the more seamless become the operations of the overall multi-agent “computational entity”. In the digital computing world such distributed but mutually informed division of expertise has become well established over the decades that this field could evolve. Progress in unconventional computing technologies is still hampered by disciplinary boundaries between level specialists, like between materials scientists and machine learning experts.
- The distinctions between the physical machine, the analytical model and abstract meta models are not as clear-cut as we suggested so far. Consider, for example, an FPGA processor. When it is programmed (better word: configured) in two different ways for use in two different applications, its Boolean circuitry is in a sense “hardwired” in two different ways. Would we say that these are two different physical machines, needing two

different analytical models? Or more generally, if any digital machine includes non-volatile memory devices, like a harddrive or a BIOS memory, and these devices are re-set, one has permanently (though usually reversibly) changed the physics of the machine. Is a new analytical model needed? At this moment we see two principled options to reconcile such scenarios with our accounts from Figure 1:

1. Option 1: Observing that in digital machines the physically changeable memory elements together with their local control circuits are essentially bistable, a single analytical model which captures the bistable attractor dynamics of the memory elements will cover all the possible non-volatile configurations.
2. Option 2: Non-digital machines may be subject to persisting physical changes which cannot be cast as attractor dynamics. For instance, the graded resistance states of filamentary or phase-change memristors would rather be mathematically described as transients which are so slow that for practical purposes they become constant. Such and other non-attractor, very slow physical changes could either become modeled in a highly resolving analytical model, as in option 1. A quite different option would be to capitalize on the reversibility of such physical changes and mathematically cast a physical system as an equivalence class of reversible configurations. This may be easier and more mathematically insightful than option 1.

The correspondence between physical state variables $v^{\Psi}(t)$ and their formal reflections $v^*(t)$ are bidirectional in the sense mentioned in the previous subsection. The correspondence between the abstract input, meta and output variables u, m, y and variables v^* in the analytical model is of a different kind. For one, a single abstract variable can be formally derived from several analytical model variables (like m_1 and y in Figure 1), and a single analytical variable can enter the derivation of several abstract variables (like x_4^* in the figure).

Furthermore, abstract model variables may temporally evolve in other ways than by following trajectories measured by physical time t — this will be discussed in the next subsection. As we will see later, this possible separation in "kinds of time" is a key for computational extensions of timescales.

An abstract computational model need not and usually cannot predict the analytical system model from which it is derived. The abstract model will often only exploit only a fraction of the physical phenomena that are captured by the analytical model, and the abstract model may contain subprocesses that have no apparent counterpart in the analytical model. The word "model" in "abstract computational model" does not primarily refer to modeling the physical system or its analytical reflection. When we speak of a formal Turing machine as a model, we suggest that it is one possible way (one "model" among many) to formalize a specific input-to-output transformation. Confusion is added by the fact that sometimes however we *do* speak of an abstract computational as a model of a physical system, as in the case of the random access machine model, which is tailored to match halfway realistically a physical von-Neumann machine. But in general, the relation between an analytical physical model and an abstract computational model is unidirectional and partial: some of the abstract meta variables are formally derived from some of the analytical variables.

In order to emphasize this difference between the bidirectional relation between the physical and analytical layers on the one hand, and the unidirectional relation between the analytical physical model and the abstract computational model on the other hand, we use the word "reflect" (= measure upwards and predict downwards) for the first relation and the word "extract" (= partially derive upwards) for the second.

There is one limit to the freedom of decoupling an abstract computational model from an analytical system model: the abstract input u must be encodable in analytical input variables and extractable from them, and the abstract output y must be decodable from analytical output variables.

From an analytical system model one can extract an unlimited multitude of different abstract computational models. They can be related to each other in abstraction hierarchies. But they can also be represent incomparably different approaches to do useful "computations" on a given hardware basis with its analytical system model. For instance, a piece of hardware whose analytical model can be understood as a recurrent spiking neural network with adaptable synaptic connections, could be used as a basis for abstract computational models of

- *reservoir computing:* adapt only the readout synaptic connections by some algorithmic procedure that leads to a linear regression [He et al., 2019],
- *unsupervised STDP learning* which combined with a maximum-activity detection operation on selected neurons yields a classifier [Yousefzadeh et al., 2018, Cove et al., 2018],
- *gradient-descent learning* using a spike-adapted version of backpropagation through time [Yin et al., 2021],

just to list some examples of work done in the MemScales consortium.

2.2.3 Transformation-completeness and model viability.

We call an abstract model *transformation-complete* when it mathematically fully specifies how abstract outputs y result from inputs u. Abstract computational models used in computer science and neuromorphic computing are typically transformation-complete (while computational models in the neurosciences are often not). Transformation-complete models can be simulated on a digital computer.

A transformation-complete model can include blackbox models when they are fully mathematically specified. The meta variables inside blackbox components are not derived by abstraction from analytical model variables. In an extreme case the entire $u \rightarrow y$ transformation is contained in a single blackbox model (Figure 1**D**). But even when the blackbox model is insightfully hand-designed, it must be experimentally validated to approximately match the input-to-output transformation of the physical target system.

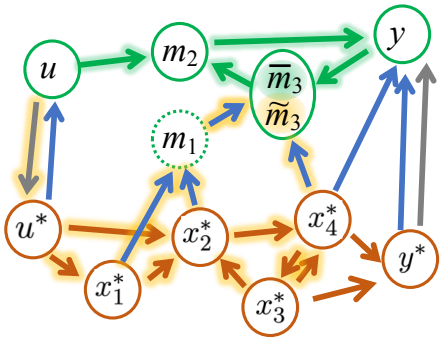


Figure 2: Twofold determination of meta-variables m through dependency pathways within the abstract computational model (green shadows) and via the enconding, analytical model transformations, and the extraction operations (orange shadows). Figure shows this double determination for m_3.

An transformation-complete abstract computational model must be *valid* in the following sense. First notice that each meta variable m in it is determined in two ways: firstly, by its ultimate dependence on the abstract input u, which is mediated through all paths within the abstract model that lead from u to m. Let us call the version of m which is determined in this way by $\bar{m}$. In Figure 2, which is an excerpt from Figure 1**B**, we pick m_3 to illustrate this. Here $\bar{m}_3$ depends on u via the paths overlaid with green shadows. Secondly, m is also determined by its formal derivation from variables in the analytical system model, which in turn are also dependent on the abstract input signal u via the input encoding operation from u to u^* followed by all pathways inside the analytical model which lead from u^* to variables x_i^* which then are used to derive m (these pathways have orange shadows in Figure 2). Let us call the version of m which is determined along these second paths as $\tilde{m}$.

The two versions of m must approximately be the same, $\bar{m} \approx \tilde{m}$. It is a delicate issue how "approximately" should be defined. It will depend on the many conditions that require a case by case study what tolerance is admissible such that the computations defined by a transformation-complete abstract computational model in terms of variables $\bar{m}$ are "well enough" served by the replay in variables $\tilde{m}$ which are derived from the analytical model. The situation becomes particularly difficult to analyse when there are recurrent cycles in the dependency paths (like the cycles $m_2 \to y \to m_3 \to m_2$ or $x_3^* \to x_4^* \to x_3^*$ in the figure). Such recurrent dependencies incur the danger that small mismatches between $\bar{m}$ and $\tilde{m}$ can become magnified over time.

In the world of digital computing, a "well enough" match between abstract bit variables $\bar{m}$ with values 0 or 1, which are extracted from analytically modeled gate voltages $x_i^*[\text{Volt}]$ as $\tilde{m}$, is ensured by three effects:

- The extraction operation $x_i^*[\text{Volt}] \to \tilde{m}$ is a binary thresholding operation which is insensitive to real-valued minor variations of $x_i^*[\text{Volt}] \to \tilde{m}$.
- The recurrent dynamics of an electronic Boolean circuit model is a bistable attractor dynamics which makes $x_i^*[\text{Volt}] \to \tilde{m}$ converge very fast toward two widely separated fixed points.
- The global clock within the analytical model gives temporal reference points (in the middle of a clock cycle, for instance) when the extraction $x_i^*[\text{Volt}] \to \tilde{m}$ is to be carried out, and the electronic bistable dynamics will be safely converged closely enough to either of the two Volt levels such that the thresholding operation will yield the correct 0 or 1 value.

We call the combination of an analytical physical model and a transformation-complete abstract computational model *valid* if the meta variables $\bar{m}$ as determined in the abstract model are "well enough" approximated by the extracted $\tilde{m}$. From an eagle's view this means that the diagrams in Figure 1 must commute. This is the essential message from Stepney et al. [2018]. The devil is in the detail: these diagrams will rarely commute in mathematical exactness, and which degrees (and which sorts) of mismatches between $\bar{m}$'s and $\tilde{m}$'s make the abstract model function as desired on the basis of the analytical model, versus making it disconnected from it, or something in between (sometimes appropriate and functioning as desired and sometimes not; or always functioning approximately in some sense) — needs to be studied anew for each case as long as we do not have a worked-out theory for this.

A further challenge is that the analytical model must capture the real physical dynamics "well enough", too. Similar considerations apply. The extraction operations here become experimental measuring operations. We do not pursue this issue in more depth here.

2.3 What time is it, and where?

So far we have developed a three-level picture of computing systems, with the physical system at the basis, abstract computational models at the top, and an analytical physical system model in between. For the theme of this report is it important to recognize that "time" may be conceptualized, quantified and formalized differently on these levels.

The physical system evolves in physical time – that continuous arrow of change that physicists and ordinary people (must) take for granted and that philosophers have not an easy time with [Callender, 2011].

Analytical system models use the symbol t to capture the continuous physical time in their ODEs, and the dot in $\dot{\mathbf{x}}$ means a rate of change that is quantified with respect to "real" seconds. The formal system trajectories $(\mathbf{x}(t))_{t_{\mathsf{min}} \leq t \leq t_{\mathsf{max}}}$ can be aligned to physical reality by human experts with physical measurements and stopwatches, and there seems little to debate.

However, when it comes to the abstract computational models, there are many ways for time to enter the game.

We start with a striking observation: in the textbooks of theoretical computer science one will not find the word "second". The update steps of a Turing machine, or the sequence of algorithmic transformations of symbolic data structures, are not connected to physical time. This has a deep reason: the theory of symbolic computing has historically emerged from the study of logical inference. Turing invented the Turing machine not as a model of a physical machine, but as a model of the logical reasoning steps that lead a mathematical reasoner from one argument in a proof to the next. One can say that the forward progression in the execution symbolic algorithms steps from Truth to Truth, — not from Time to Time. The decoupling from physical time is even explicitly stated by Turing [1936]: *"It is always possible for the computer to break off from his work, to go away and forget all about it, and later to come back and go on with it"* — provided that "the computer" (a human male in Turing's famous article) left a written note to remind him later at which stadium in the computation he took a break. In modern digital-technology parlance: after writing a bit into a memory cell, the bit stays there for arbitrary (physical-real) timespans until it is read again or overwritten. The availability of non-volatile discrete state changes in digital computers allows digital computations to decouple from physical time.

On the other hand, there are abstract computational models that *do* use the "t" of physicists. In particular we think of the signal transformation procedures which are designed in the fields of (analog) signal processing and control. These are abstract computational models, whose structure is often illustrated in box-and-arrow diagrams with the boxes being filters and the arrows signals $s(t)$.

We will use the term *mode of progression* to refer to the way how the successive transformations of meta variables in the course of a computational process can be seen as "some kind of time". We use the symbol t to denote any mode of progression.

In between the physically measurable time t on the one end, and the purely logical inference updates on the other end of a spectrum which ranges from physical-temporal to logical-atemporal, there are abstract computational models which use intermediate or hybrid modes of progression, for instance

- sampled timesteps Δt in digital signal processing theory (still synchronized with physical time t);

- dimensionless “unit” timesteps in iterated function models of computing, for example discrete-time recurrent neural networks;
- models of information-processing based on hybrid logical-physical modeling formalisms used in the formal verification of hardware-embedded digital computing systems [Geuvers et al., 2010], where logical argumentation steps are intertwined with analytical physical models of components of the complex hardware system that is being modeled;
- models of real-time operating systems for digital hardware where certain computational processes (expressed algorithmically as a sequence of logical update steps) must not exceed a limited number of update steps, such that this number times the (physical) clock period does not exceed a physical time limit,
- Petri net models of concurrent information processing systems which come in many variations with regards to time models, from the classical purely discrete algorithmic version through timed variants to variants that involve continuous flows of tokens, defined with respect to continuous but arbitrary (not physical) time [Alla and David, 1998].

In summary: while the physical computing system evolves in the real-world physical time, and the analytical physical system model must describe state evolutions with respect to the measurable t (measured in seconds), “anytime goes” for the meta variables’ modes of progression in abstract computational models.

This view leads to an interesting issue which is of importance for the theme of this report. Given (i) that the analytical system model uses physically measurable time t for monitoring the evolution of its analytical variables v^*, and (ii) that in the abstract computational model other modes of progression may be used for the meta variables u, m, y, the question arises how one can formally change the mode of progression when one extracts abstract meta variables from analytical variables or other meta variables. There are many ways. For illustration here is an ad hoc list of mathematical operations to create new meta variables from existing analytical or other meta variables. The first two extraction methods in this list are the two most widely used methods to obtain meta variables $m(t)$ that inherit the progression modus of physical time t and thus are *synchronized* to the variables they are derived from:

- A meta variable $m(t)$ with the physical mode of progression t can be obtained through a *function* $m(t) = G(v_1^*(t), \ldots, v_n^*(t)$ of analytical variables.
- A meta variable $m(t)$ can be obtained through a *filter* $(m(t))_{t\in\mathbb{R}} = H((\mathbf{v}^*(t))_{t\in\mathbb{R}})$ which transforms vector signals $(\mathbf{v}^*(t))_{t\in\mathbb{R}}$ to meta signals $(m(t))_{t\in\mathbb{R}}$.
- Analytical model variables $v(t)$ or other meta variables $m(t)$, where t is the physical time $t[\text{sec}] \in \mathbb{R}$, can be abstracted by dropping the commitment that the time parameter means the physical time that can be measured with physical clocks. To reveal this dramatic conceptual change in formal notation, we use $t^\emptyset$ as symbol for dimensionless time. The abstraction then is formally effected by the simple replacement of t by $t^\emptyset$. Abstract computational models that use dimensionless time can represent physical systems that run at arbitrary physical speeds.
- By discretizing physical time t through sampling with physical period length $\Delta[\text{sec}]$ one gets discrete time $n\Delta[\text{sec}]$ where $n \in \mathbb{Z}$.
- From discrete-time abstract models with progression modes $n\Delta[\text{sec}]$ or n one can always extract further discrete-time abstract models by subsampling, and sometimes by supersampling (interpolation).

- Discrete physical time $n\Delta[\text{sec}]$ can be abstracted to dimensionless discrete time n. – Note that every analytical or abstract model which uses continuous time t can be discretized by sampling, but the converse is not true: there are discrete-time system models (for instance, expressed in an iterated map formalism) for which no continuous-time corresponding system exists from which the discrete-time model is obtained through sampling.
- From analytical variable vectors $\mathbf{v}$ or meta variable vectors $\mathbf{m}$ which have modes of progression $\mathfrak{t} \in \{t[\text{sec}], t^{\emptyset}, n\Delta[\text{sec}], n\}$, one can extract a new meta variable m' by defining (i) a binary 0-1 *trigger signal* $s(\mathfrak{t})$ which is zero most of the time and jumps to one whenever $\mathbf{v}(\mathfrak{t})$ or $\mathbf{m}(\mathfrak{t})$ in their evolution meet some trigger criterion, and (ii) a filter M operating on the trajectories of $\mathbf{v}(\mathfrak{t})$ resp. $\mathbf{m}(\mathfrak{t})$. The new variable m' has mode of progression n and consists of the sequence of values returned by M at times $\mathfrak{t}$ where the trigger signal is one. Alternatively, one may also keep a record of the inter-trigger intervals and include this information in the progression mode of m', which would then take the format of a sequence of inter-trigger intervals of variable length. A familiar example is the abstraction of a continuous-time, continuous-valued neural membrane potential signal (as it may come out of the Hodgkin-Huxley equations for example) to a binary spike train signal. In digital real-time operating systems, the trigger signal can indicate the completion of a subprocess and M copies the result thereof. Event-based abstract computational models are generally extracted from analytical models by such wait-trigger-read mechanisms.
- By various methods of memorizing/buffering, predicting, and time-warping one can locally stretch or compress a mode of progression $\mathfrak{t}$ such that a source progression $v(\mathfrak{t})$ or $m(\mathfrak{t})$ becomes bidirectionally mapped on a target progression $m'(\mathfrak{t})$ in the same mode. There is a multitude of options which would need to be discussed in detail, depending on the specific formats of variables and their modes of progression. This is outside the scope of this report.
- One can define complex or compositional modes of progression in at least two ways. Firstly, one can let a variable m evolve in different modes at different periods in its evolution, for instance alternating between discrete and continuous update windows. The structure of such composition-in-time would be characterized by meta-modes. Secondly, one can consider sets of variables whose members evolve according to different modes of progression, with some sort of coordination between them. Such composition-in-space would again be characterized by meta-modes. Both ways could also be combined.

This indicative list must suffice here. A more complete catalogue of modes of progression, and an in-depth study of which modes can be obtained by formal operations from which other ones and for what formal types of variables and laws of evolution, remain to be worked out. Such a theory would be needed for a full understanding of our theme, “computational extensions of timescales”. Two interrelated key questions which await a systematic investigation are

1. How can the loss of information be characterized when abstracting variables to new modes of progression? Can modes of progression be ordered in an abstraction hierarchy, such that variables with modes higher in the hierarchy can be extracted from variables with modes lower in the hierarchy, but not vice versa? Presumably the physical time mode $t[\text{sec}]$ would lie at the bottom, and the atemporal logical inference steps of progression mode n in symbolic proof engines or the Turing machine would be found very high in the hierarchy.
2. If an abstract computational model is formalized with modes of progressions that lie high in the abstraction hierarchy of modes of progression, are there systematic ways to successively design less abstract models, ultimately arriving at an analytical physical model

with mode t[sec], such that the more abstract models in this sequence can be extracted from the "lower" ones? This is the question of systematic "compilation" mechanisms which can bridge different levels of modes of progression. An example of a specific compilation hierarchy of this kind is the neuromorphic system engineering framework proposed by Zhang et al. [2020].

2.4 Timing of input and output

Input can be delivered to an abstract computing model in various ways, for instance

1. as (part of) the *initial state* as in Turing machines, the Hopfield network, or the abstract model of a computing system proposed by Horsman et al. [2014],
2. as *clamped* input like in the Boltzmann machine [Ackley et al., 1985], or
3. as *interactive* input which is given to the computing system intermittently during its operation by a user, as in models of interactive Turing machines [van Leeuwen and Wiedermann, 2001], or
4. as an online *signal* which is continually fed to the computing system during its operation.

It is possible to formally represent the first three of these options as special cases of the last one, for instance as follows:

1. The input signal gets an extra channel with only two possible values 1 and 0, which is set at start time $t = 0$ to 1, and in addition a constant signal that at every time is equal to (an encoding of) the desired system start state $\mathbf{x}(0)$, and add a special mechanism to the system equations which set the system state to $\mathbf{x}(0)$ when the extra input channel reads "1".
2. The input signal has the constant "clamped" input value at all times.
3. Like in 1., add an extra 0-1 indicator input channel to an input signal which in its other channels has a "payload" input sequence, with the system equations configured such that the payload input is read whenever the indicator channel shows a "1".

These methods to cast inputs of sorts 1. – 3. as signals have been used, for instance, in the extensive studies of long-term memory capabilities in recurrent neural networks in Martens and Sutskever [2011] and Jaeger [2012].

In the remainder of this report we will thus consider "inputs" always as a temporal signal that is fed to the computing system throughout its operation.

By analogous arguments we will consider "outputs" likewise as temporal signals that are issued by the computing system throughout its operation.

2.5 Concepts of timescales

So far we worked out what we understand by a computing system, at a level of differentiation that will be suitable for discussing "computational extensions" of timescales. But we have not yet clarified what we mean by "timescales". In the following subsections we take a closer look and will find that there are quite different sorts of them, and angles to look at them.

2.5.1 Next to the physical basis: timescales as time constants

Computing systems are physical systems. Arguably the most popular formalism in physical system modeling is ordinary differential equations (ODEs). Physicists, neuroscientists or electronic circuit engineers will almost by reflex describe their target systems through state vectors $\mathbf{x}(t) \in \mathbb{R}^n$, where the individual system variables $x_1, \ldots, x_n$ correspond to physical quantities measured in standard physical units like Volt or Ohm, and the dynamics of the system is expressed by coupled differential equations

$$\tau_i \, \dot{x}_i = f_i(\mathbf{x}, \mathbf{u}, \mathbf{a}), \tag{1}$$

where $i = 1, \ldots, n$ and $\mathbf{u}(t)$ is an optional input signal and $\mathbf{a}$ is an optional vector of control parameters, and τ_i is the *time constant* of the dynamics of the physical quantity x_i. When τ_i is large, physicists and dynamical systems mathematicians call x_i a slow variable, and when it is small, a fast one. When all these equations are set up appropriately, the model will describe the temporal evolution of all the physical quantities modeled by the variables x_i in numerically correct rates of change with respect to the real-world physical time t, standardly expressed in seconds. One sometimes calls the time constants τ_i suggestively the *native* or *intrinsic* timescales of the respective physical quantities, with a background intuition that these time constants reflect the "real", or "causal", or "physical" timescale of the concerned quantity. We will use the term "physical time constant" in a quite narrow sense, namely for time constants associated with the physically measurable variables in ODE models whose variables are quantified in standard units of measurements.

We note that the absolute values of such time constants depend on the choice of units of measurement, both for system state variables and for time itself. If one measures time in hours instead of seconds, all time constants will have to be scaled by 1/3600; if one measures some voltage x_i in mV instead of V, one has to scale its time constant by a factor of 1000. The absolute values of time constants, and the ratios of time constants of two variables that correspond to different physical sorts (like voltage vs. current), are thus in essence arbitrary, and it makes little sense to speak of a fast variable only because its formal time constant is small.

There is however a crucial aspect of physical time constants that is not arbitrary. It comes to the surface when one compares two models A and B of a physical system, both expressed in the same units of measurement, where between the two versions there is a 1-1 mapping of a subset of system variables. For instance, in an electronic circuit model A one may change one resistance to obtain version B, which would give a global 1-1 variable mapping between A and B. Or in A one might add a little extra subcircuit to get B; the 1-1 mapping would be between the variables of A and all the variables in B that do not belong to the newly inserted subcircuit. When one compares A with B, the ratios between the time constants of the 1-1 mapped system variables is independent of the choice of units of measurement, and it is these ratios which guide a system engineer to let the final design fulfil its temporal specifications.

There are of course other modeling formalisms for physical systems besides ODEs, for instance partial or stochastic differential equations or more general sorts of stochastic process formalisms. In some of these one can find canonical correspondents to ODE time constants, for instance for the drift component in stochastic differential equations (chapter 15 in Kuehn [2015]), or the (inverse of the) variance of a Gaussian transition kernel in continuous Markov processes. We are however not aware of a unified definition of time constants across several classes of modeling formalisms, and will restrict our discussion to ODE models since these are so widely used, and since the mathematical theory of what mathematicians call *multi-timescale*

systems is rooted in ODE models [Kuehn, 2015].

A hallmark of physical time constants is that if one wants a system in whose model they are changed, one has to change the physical make-up of the system. An electronic circuit engineer who wants a certain time constant become faster or slower will have to create a new physical variant of the system, for instance by changing capacities or resistances or adding new circuitry.

Physical system models made by physicists or engineers are almost always intended to be *analytical* models, as opposed to the *blackbox* models that are typical in machine learning. In an analytical modeling spirit, the model's variables and formal dynamical laws should correspond to the "real" physical quantities and physical mechanisms in the target system. Thus, the model variables x_i should correspond to classical physical quantities which are measured with standard physical units. As a consequence, an analytically minded physical system modeler cannot do what mathematicians often like to do, namely investigate the same system in a transformed coordinate system. For instance, in a two-dimensional system model with x_1 capturing a voltage in Volts and x_2 a resistance in Ohms, applying a linear coordinate transformation A to state vectors $(x_1, x_2)'$ would give new system variables $(z_1, z_2)' = A\ (x_1, x_2)'$ which would formally correspond to weighted mixtures of voltages with resistances, for which there is no meaningful physical unit of measurement, and it would not be possible to build a measurement apparatus that could directly measure z_1 or z_2. Analytical ODE system models and their time constants are in a deep sense "mathematically locked" to a specific coordinate system.

An intuitive interpretation of physical time constants is not straightforward. Making a time constant τ_i smaller or larger in a system model does not imply that the concerned system variable x_i always changes faster or slower. For example,

- when a system whose time constants are all very small (hence called "fast") is close to a fixed point attractor, and its input signal is constant (or this system has no input), the system variables will be asymptotically coming to a standstill despite their fast time constants;
- a system whose model has all very large ("slow") time constants can exhibit fast oscillations even with absent or constant input if its internal feedback gains are strong enough,
- a coupled oscillator system model with a "fast" variable x_i that is associated with a small time constant τ_i may display slow changes of x_i first, which become fast oscillations when τ_i is *increased* — due to shifting the overall dynamics to a resonance frequency which was suppressed when τ_i was small.

There is thus not universal connection between making time constants smaller (in model simulations and/or by modifying the physical system) and observing that the concerned system variables "move faster".

It requires a case-by-case discussion to understand how a specific physical time constant impacts on dynamical "speed" properties of the modeled system. The observed "speed of change" of a variable x_i with time constant τ_i will depend, among other factors, on fast/slow properties of the driving input, strength of system-internal feedback, attractors present in the system, or whether the system is observed in transients or close to attractors. Intuitive or mathematical timescale-relevant interpretations of τ_i will differ. Examples for such case-by-case interpretations of time constants:

- When a system is driven by slow input, such that its (single) fixed point attractor is adiabatically following the input, the time constant τ_i characterizes an exponential *rate of*

convergence to the fixed point, which for changing input translates to *more precise* or *shorter delay* input tracking when τ_i gets smaller.

- (Only) in system models without input, scaling all time constants by the same scaling factor a will make the system trajectory $\mathbf{x}(t)$ move slower or faster through the state space $\mathbb{R}^n$ in proportion to the scaling factor a. This may be the original motivation to call the τ_i by the name "time constants".
- In singular perturbation methods in mathematical studies of so-called slow-fast systems, the ODE models have two groups of equations, one group with small and the other with large time constants (the "fast and slow subsystems"). When the ratio of the small over the large time constant approaches zero, specific operating and interaction conditions for the two subsystems can be isolated:
  - The fast subsystem approximately behaves as if the slow subsystem is standing still, yielding a control parameter vector to the fast subsystem. This admits bifurcation analyses of the fast subsystem.
  - The dynamics of the slow subsystem can be studied using only the equations of the slow subsystem on the *critical manifold* $C_0 \subset \mathbb{R}^n$ embedded in the total state space, where C_0 is the set of zeros of the fast subsystem. Close to C_0, system trajectories will be "pulled along" these slow trajectories within C_0 if C_0 is an attracting surface.

It becomes clear from such considerations that time constants in physical system models capture "real" physical dynamical characteristics of the concerned state variables on the one hand, but that on the other hand these physical characteristics do not 1-1 translate to phenomenal "fastness" or "slowness". The time constants which one finds in analytical ODE models of physical systems (or rather, their reciprocals τ_i^{-1}) would maybe better be called "coupling strength constants" than "time constants": the larger τ_i^{-1}, the stronger is x_i impacted by the other system variables on the r.h.s. in $\dot{x}_i = \tau_i^{-1}\, f_i(\mathbf{x}, \mathbf{u}, \mathbf{a})$. Since an analytical system is a stand-in for the real physical system, and its dynamics reflect physical causality, one could also say that these ODE time constants model *causal* effects.

2.5.2 Timescales of change

In contrast to the causal effects that are reflected in the analytical system model, "timescales" in abstract computational models are *phenomenal*. When we speak of timescales in the context of abstract computational processes, we describe *how* things change, not *why*.

In this subsection we consider the phenomenal aspect of "speed of change". This is maybe the aspect that is most immediately associated with the word "timescales". The core intuition here is that we call a temporally evolving variable "fast" if it changes strongly within short time intervals, like high-frequency oscillations; and we call it "slow" if it changes only a little or not at all as time goes on.

This basic aspect of "changing fast" vs. "changing slowly" is an idea of *motion*, "fast" meaning that much *distance* is covered per time unit. We can thus discuss timescales of change only for mathematical objects that lie in spaces where some kind of distance measure is available, like a mathematical metric or the Kullback-Leibler divergence. This includes objects like

- numerical (scalar) variables,
- vectors of numerical variables,

- suitably restricted classes of functions into metric spaces like $f : D \to \mathbb{R}^n$ which yield metrizable function spaces; this includes many kinds of vector fields,
- probability distributions over measurable spaces,
- nodes in an undirected graph where distance is graph distance,

and more. We leave it at that, and will refer to mathematical objects that come with some notion of distance as *metric descriptors* with the understanding that we admit other distance-like measures besides the standard mathematical definition of a metric.

In many theoretically important and practically useful formal models of computing systems (including the Turing machine), the three sorts of formal objects $\mathbf{u}(t), \mathbf{x}(t), \mathbf{y}(t)$ are however not metric descriptors but discrete objects like truth values, symbolic expressions, graphs or neural spikes. In order to start discussing "speed of change" in such situations, one can attempt to create meta descriptors which are metric in some sense and thus admit the definition of rates of temporal change. A common example is to define spatial or temporal averages of spike counts to get continuous-valued spike rates $m(t)$ which are metric descriptors. In a run of a Turing machine with tape alphabet $\{A, B\}$ one could (an entirely arbitrary example) define as a metric descriptor $m(t)$ the the number of A's written on the tape during the last 100 update cycles up to time t. This would make $m(t)$ a number between 0 and 100 which would change by at most ± 1 in a step from time t to $t + 1$, a measurable increment upon which timescale characteristics can be defined.

Thus, if we have a time series $v(t)$ or $v(n)$ in continuous or discrete time, and v is a metric descriptor — how can we define and measure "speed of change"? Here are some important options.

- A signal processing engineer will transform $v(t)$ or $v(n)$ to the frequency domain and call it slow when the Fourier spectrum is dominated by low frequency components (which in turn leads to the question how, precisely, one will declare frequency spectra to be dominated by what frequency components).
- In wavelet transforms, slow components of $v(t)$ or $v(n)$ could be associated with the projections on wide-support wavelets.
- A mathematician may want to compute norms like $1/T\ (\int_{t=0}^{T} |\dot{v}(t)|^k\, dt)^{1/k}$ and call v fast if this norm is large, on the grounds that a large norm implies "a lot of change" during a reference time interval T.
- Such norms however would also be large for a constant ramp signal $\dot{v}(t) = \text{const}$ when the slope is large. This may often not be what one wants, and one might prefer higher-order measures of change, like $1/T\ (\int_{t=0}^{T} |\ddot{v}(t)|^k\, dt)^{1/k}$ which quantify *variations* in speed of change instead of only speed of change itself.
- All such norm-based measures will scale linearly or nonlinearly with scalings $\alpha\, v$ of the concerned signal. This may or may not be desired, and if not, the signal $v(t)$ would first have to be normalized in some way.
- A complex systems researcher might consider $v(t)$ as slow when its autocorrelation plot has a "fat tail", that is when there are long-term temporal correlations which decay slower than exponentially. More generally, autocorrelation spectra may be said to contain slow components when autocorrelation values for large timelags are large.
- A dynamical systems mathematician will identify regions in an ODE state space $\mathbb{R}^n$

where $\mathbf{v}(t)$ moves slowly vs. regions where it moves fast, measured by $|\dot{\mathbf{v}}|$. Here slowness/fastness becomes a local property that changes with time t.

This list illustrates that there are many natural options to define quantitative measures of rates or speeds or variations of change.

In all these measures there is another source of freedom of definition, arising from the question how locally vs. globally one defines these measures. Consider a signal $v(t)$ which jumps between the values 0 and 1 with steep flanks, staying at the two extreme values for extended periods. To the degree that the flanks are steep, most of the measures listed above will give large values in small measurement intervals around the flanks, and zero "speed of change" when measured within one of the two plateaus. One might opt for defining the speed of change for such a signal (if it is stationary in the sense of stochastic processes) as the measure value in the limit of the measuring interval T going to infinity. But this will hide the possibly interesting and relevant fact that there are clearly discernible fast and slow subperiods. Measures of speed of change should thus be qualified by the observation interval. Instead of a single speed-of-change measurement value for a signal $v(t)$ one gets a hierarchy of such values, with levels ordered by durations of T; and within each level T one gets a distribution over measurement values.

Summarizing, a full account of analysing timescales of change for a stationary signal $v(t)$ or $v(n)$ would include

1. the choice of a numerical measure μ that yields a "speed of change" value for any given observation window $[t, t+T]$,
2. for each $T > 0$, a probability distribution over the outcomes $(\mu([t, t+T]))_{0 \leq t \leq \infty}$.

Further complications occur when the signal $v(t)$ is nonstationary. We will not attempt in this report to unravel the issues that arise in such cases.

2.5.3 Timescales of reactivity

For computational systems it is relevant to know *how soon will the system react to inputs?* This question in turn comes a number of interesting variants:

- In the classical model of symbolic computing, based on formalisations of algorithms like the Turing machine or computer programs, the question of reaction time to an initial input has become thoroughly studied in the theory of *computational complexity*, in particular *time complexity*. The time complexity of an algorithm is defined via the number of discrete state update steps that the algorithm needs to return the result, defined as the asymptotic growth function by which this stepping-time increases when the size of the input argument grows.
- Tasks that require online reactions to event-like input from the user often come with an objective to react to the inputs with short latency.
- In neuroscience modeling and analog signal processing one considers the real-time *delays* after which the hardware system shows a response to information in the continuous input signal. Different characteristics of the response (e.g. accuracy or statistical reliability) can reveal themselves after different delays, which more accurate or more reliable response components arriving later. Similarly, classical symbolic *anytime algorithms* can be queried for output after different processing durations, earning more precise or reliable results when one waits longer.

- In feedback tracking control systems, higher feedback gains lead to faster and more precise tracking, at the risk of coming closer to instability of the control loop.
- Time-to-failure prediction systems are often used in predictive maintenance of engines, generators or other industrial systems. These algorithms are typically machine learning models. Here a fast reactivity would mean that *early* warnings are generated, which in turn means that the monitoring system reacts very *sensitively* to changes in the monitored diagnostic signals. While high sensitivity of an online computing process is not the same as *fast* reactivity, the desired *early* reactions are a closely related aspect of reactivity timescales.

These examples appear to be rather different in their nature, and collecting them under the headline of “reactivity” is somewhat ad hoc. A more systematical analysis is beyond the scope of this report.

2.5.4 Timescales of memory

The MemScales project is concerned with a specific sort of physical systems, namely those that “compute” — brains, digital computers, analog electronic microchips and beyond those, other unconventional substrates and systems. For a dynamical system to “compute” it is quintessential that it can preserve and transform information over time. How to memorize what on what timescale is the name-giving theme of our project.

The most natural (if not naive) concept of “memorizing” is to cast it as “storing”: putting things on a shelf where they can be fetched whenever needed. This is in fact the concept of memory that dominates digital computing practice and theory: “writing” bits into non-volatile memory devices and “reading” or “deleting” them when needed.

But in many non-digital hardware systems including brains, non-volatile memory devices are not available or are plagued by numerical imprecision, process variation (at fabrication time), temperature sensitivity, stochastic fluctuations and device mismatch. The spectacular, classical long-term studies by psychologist Bartlett [1932] highlight that human memory systems do not “store” information but always keep on transforming any memorized information, continually reshaping and re-coding memory traces in order to preserve an overall consistent cognitive world representation. “Memorizing” thus is a complex game which includes

- *encoding* information in dynamical sub-processes at the time when this information appears in the input or when it was internally generated,
- *propagating* this encoding during some time lag by keeping traces or transformations of it “alive” in the ongoing system dynamics,
- *decoding* it back to a useful format when later needed.

Each of these three stages can become more concretely spelled out and realized in a multitude of ways which differ with regards to, for example, how the elusive concept of “information” is understood in the first place; whether the input is given only once at the beginning of a run of an algorithm or streams in continuously (and similarly whether the decoding is done only once at the end of a run or a continuous stream of readings from memory is needed); whether a gradual degradation of the (encoded) information over time is acceptable or not; what kind of dynamical mechanism is exploited for each of the three stages; or whether the memorizing dynamics are interwoven with learning processes that change the processing system itself (arguably always the case in biological brains), etc. Further scintillating facets are added to the

picture by the overlay of different memory mechanisms in the same processing system to cope with multiscale temporal tasks; meta-mechanisms of *attention* to decide which parts of currently available information needs to be memorized; other meta-mechanisms of *addressing* to decide where information can be found to be identified and decoded; and not to forget: *forgetting* or overwrite mechanisms to free "memory capacity".

There are many options for mathematical definitions and numerical measures of "how much information of what sort" is propagated through a time lag Δ. In the reservoir computing literature, a popular measure is the *memory capacity* introduced in Jaeger [2002]. This is a simple measure based on the correlation between input and output signals which is easily measured in computer experiments. However, a more fundamental and insightful definition/measure would be to define and quantify "information transfer" by the *mutual information* between system states $\mathbf{x}(t)$ and $\mathbf{x}(t+\Delta)$ separated by a lag Δ. This measure is independent of (bijective) encoding and decoding operations; it is a characteristic of the processing system with states $\mathbf{x}$ itself.

This information-theoretic line of thinking about "memory" implies that one cannot determine whether a system is capable of memorizing by looking only at a single example pair $(\mathbf{x}(t), \mathbf{x}(t+\Delta))$: even when one observes that a system is exactly in the same state $\mathbf{x}(t) = \mathbf{x}(t+\Delta)$ before and after the lag, this does not mean that the system has memorized $\mathbf{x}(t)$. In an information-theoretic view, memory mechanisms must be defined and identified by considering *distributions* of earlier-later system state pairs, in order to assess whether or how much information has been transferred through the lag interval.

Or, alternatively to considering distributions over states, one can define mutual-information-based memory for states that encode distributions. If $\mathbf{x}(t)$ and $\mathbf{x}(t+\Delta)$ each represent probability distributions, the mutual information between even a single such pair is defined. Such states-as-distributions occur, for instance, when $\mathbf{x}(n+\Delta)$ is the probability vector obtained from in a discrete Markov chain with transition matrix M by $\mathbf{x}(n+1) = (M')^{\Delta}\,\mathbf{x}(n)$, or when $\mathbf{x}(t)$ describes the evolution of a Fokker-Planck equation.

Hybrids between these two options can also be defined. Consider a stochastic system with states $\mathbf{x}(n)$ or $\mathbf{x}(t)$ whose states are "just states", not representing probability distributions, for example the symbol state sequences of a discrete Markov chain $X(n)$ over a state set S. Every state $s \in S$ can be associated with the conditional distribution $Y_{s,k}$ over k-step continuations of the process, that is the distribution over S^k given by

$$Y_{s,k}(s_1, \ldots, s_k) = P(X(n+1) = s_1, \ldots, X(n+k) = s_k \mid X(n) = s).$$

Then, when one has a pair of states $\mathbf{x}(n) = s, \mathbf{x}(n+\Delta) = s'$, one can define and measure the information carried from $\mathbf{x}(n)$ to $\mathbf{x}(n+\Delta)$ by the mutual information $I(Y_{s,k}; Y_{s',k})$. This way (which can be very much generalized) of associating or even identifying physical system states with the conditional distribution of the process' future after that state has a venerable history in physics [Zadeh, 1969] and has become central in certain representations of stochastic processes, such as epsilon machines in complex systems studies [Shalizi and Crutchfield, 2001], multiplicity automata in theoretical computer science [Schützenberger, 1961], observable operator models in machine learning and mathematical theory of stochastic processes [Jaeger, 2000], and predictive state representations in reinforcement learning and agent modeling [Littman et al., 2002]. A unifying review is given by Thon and Jaeger [2015]. This view is also constitutive for the *predictive brain* hypothesis in cognitive science [Clark, 2013] which posits that neural brain states encode expectations about what is going to happen next.

Unfortunately, numerically estimating the mutual information from samples of two distributions is both computationally expensive and even not well-defined unless one makes additional as-

sumptions about the distributions. In practice one will often take resort to correlational measures which sometimes provide qualitatively similar insights as mutual information [Metzner and Krauss, 2021].

Defining and measuring from samples the information transfer between two systems or between two times in the same system has been extensively studied. Many definitions and estimation algorithms have been proposed besides basic correlation and mutual information, for instance *transfer entropy* [Schreiber, 2000] which is an information theoretic measure, or *Granger causality* which in its original formulation is correlation-based [Granger, 1969], or *correntropy* [Xu et al., 2008] which is a hybrid. The last reference also gives a brief survey of other measures. Some of these measures are symmetric (correlation, mutual information and correntropy) which is maybe counter-intuitive: one may wish that information be carried forward through time is not the same as the information "reflected backward". Transfer entropy and Granger are non-symmetric and have been designed to capture a direction of *causation*. We cannot attempt a comprehensive account here.

When one has settled for a measure $M(\Delta)$ of information transfer across a time lag Δ, in order to define timescales of memory it is not enough to consider the information transfer across a single such lag. Instead one should base timescale discussions on *memory curves* $f : [0, \Delta_{\text{max}}] \to \mathbb{R}^{\geq 0}, \quad \Delta \mapsto M(\Delta)$. These *forgetting curves* may take qualitatively quite different shapes, indicative of different kinds of forgetting/memorizing. They can range from almost rectangular curves indicating perfect recollection up to a critical lag after which there is total forgetting (Figure 3A), to gradual long-time decay of memory traces (Figure 3C), with many interesting special cases like exponential decay or fat tails. It becomes clear that speaking about a "memory timescale" needs a careful definition of which measure of information transfer is used and which are the geometric properties of the memory curve.

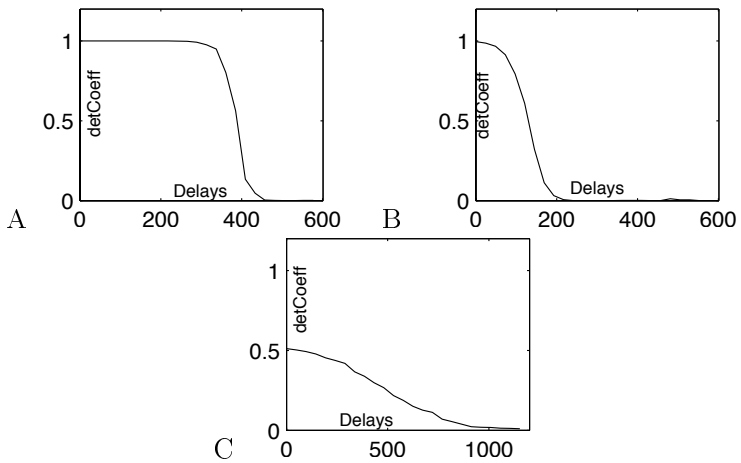


Figure 3: Illustrating the diversity of memory curves: Three memory curves, showing a correlational measure (determination coefficient) between inputs and outputs of the same recurrent neural network under the influence of three different hyperparmetrizations. Figure taken from Jaeger [2002].

In the light of all of these ramifications and complications, speaking about "timescales of memory" ceases to be a straightforward affair. This is true for artificial computing systems as well as for brains. Upon closer inspection, the customary coarse distinction between short-term, working, and long-term memory spreads out into a tangle of interwoven phenomena and mechanisms which in the human brain are served by a multitude of physiological mechanisms and anatomical structures (discussions in [Fusi and Wang, 2016, Jaeger, 2017]). If we neuromor-

phic computing researchers take our mission to “learn from the brain” seriously, we should brace ourselves to meet with extreme complexity.

2.6 Absolute and relative timescales

In a number of *online* computational task types, in particular in

- signal processing and control (using analog or digital processors) where the processing system is *entrained* to the driving input signal, and
- “real-time” information processing based on cascaded classical symbolic algorithms which in fast succession carry out sub-algorithms which have to terminate after admissible delays,

the task definition has input and output specifications that are based on physical modes of progression like $t[\text{sec}]$ or $n\Delta[\text{sec}]$. Designing computing systems for such tasks can be challenging for a number of reasons:

- The formalism used for the abstract computational model must be expressed in physical modes of progression for the input and output variables, which leads to formalization and design challenges when intermediate processing is most naturally expressed in terms of atemporal logical modes of progression. An example is control and action selection architectures for cognitive mobile robots, where continuous sensor-motor control loops must be integrated with naturally symbolic planning routines.
- If the underlying hardware is non-digital and only commands on volatile memory mechanisms, its physical time constants must be such that in concert with the abstract computational model all physical timescale constraints for speed of change, reactivity, and memory must be met. If all one has available is a physically fast system but one wishes to use it for a slow task, one has to extract slow enough meta descriptors for the abstract computational model to exploit them for the task — which we did in the precursor project NeurRAM3 in the heartbeat monitoring task where we had to “slow down” the too fast Dynap-SE neuromorphic microprocessor with purely computational methods [He et al., 2019]. If conversely the available hardware is too slow for the task, one would have to find “faster than hardware physics” meta descriptors. We lack an example. We suggest in passing that one way to create fast meta descriptors from slow oscillatory variables in the analytical system model may be to transform the latter to steeper-flanked versions first (for instance by taking higher powers of the slow signal) and then superimpose such steepened signals with delays, earning fast oscillations with steep flanks as “raw material” for sped-up abstract computations.

In digital real-time systems the design of chip and computer architectures, operating systems, and programming paradigms can become challenging. The higher the complexity of computations that need to be carried out in a given (real) short timespan, the shorter the clock cycle time must be. But there are limits to speeding up clock cycle times. Unfortunately, the trend in sub-10 nm technologies goes rather the opposite direction: on-chip delays become ever longer compared to the fast clock cycles used in modern processors and systems-on-chips [Saraswat, 2006].

In continuous online or real-time tasks we say that the computing system must meet *absolute* timescale requirements.

A different situation occurs in *offline* processing tasks, where a static data structure is inputted

to a computing system at the start of a computation run, and the result is outputted at the end of the run. This is the classical view of executing an algorithm in the digital-symbolic paradigm. Here it is usually desired (for reasons of economy of use) that the computing machine runs as fast as possible in its physical processing time. However, when

- the input data structure is itself a timeseries (for instance a pre-recorded signal or a text) where relevant information of different sorts develops at different timescales,
- and this timeseries is processed incrementally "from left to right",

the processing system must command on memory timescales that can preserve input information across different delays. This is the case in many text, speech, music, video or gesture processing tasks.

In digital machines this poses no principled problem because input or interim information structures can be digitally stored for as long as needed. Any memory duration demands can be satisfied by storing intermediate information in a RAM, and retrieve that information at any later time when needed.

In contrast, for neuromorphic or other non-digital hardware systems with volatile memory physics only, staggered memory timescales mandate that several timescales are effectively available, either directly in the physical (causal) timescales of the machine or by creating suitable meta descriptors and interpreting the task solution in terms of those. We then speak of the necessity to have a choice of *relative timescales* available.

2.7 Section summary and clarified task specification

We started out the MemScales project with a general awareness that on the one hand,

- analog, spiking neuromorphic hardware with volatile memory devices will usually provide only a small number of physical timescales,

but on the other hand,

- many computational tasks pose timescale demands that do not directly match the physically available ones,

which led to a core objective of MemScales, "computational extension of timescales". We are now in a much better position than at the time of writing the proposal to understand the nature of this objective (as explained in this theory section), and to assess and try out solutions (next section).

Summarizing the findings of our theoretical explorations, the task of "computational extension of timescales" can now be re-stated in more detail and with better strategic guidance:

- Our original intuitions were quite natural and straightforward: solve the mismatch between (i) a few available physical timescales and (ii) task-demanded other timescales by finding computational "tricks" to "extend" the physical timescales to the needed ones. This intuition was based on our experience and computational solutions in the precursor project NeuRAM3.
- But the task is not as simple or easily stated as that:
  - We find it insightful to study "timescales" on the background of a conception of a computational system which presents itself in three layers: the physical hardware

system; its reflection in an analytical system model which attempts to model the physical system in a veridical manner (that is, one can specify degrees of how accurate or even "true" this system model is); and abstract computational models which can be extracted from the analytical model in many different ways depending on which sort of computational tasks one wishes to get done with what abstract procedure.

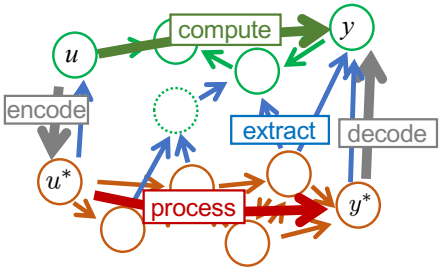
- There is not a uniform single concept of what is a "timescale". We find an assortment of conceptually, physically, and mathematically different ideas all of which relate to "timescales". This assortment includes the effects and phenomena which we highlighted in Sections 2.5.1 — 2.5.4: time constants reflecting strengths of causal couplings, and phenomenal timescales of change, reactivity, and memory. Future investigations will most likely reveal more kinds of phenomenal timescales. Further differentiations arise from considering how input signals relate to time (Section 2.4) and from distinguishing absolute from relative timescales (Section 2.6).

- On this background, the (too) generally stated task of "computational extension of timescales" splits into a spectrum of different sorts of tasks that need to be considered separately:

  Bottom-up abstract model design tasks

  1. *Given:* A physical hardware system and an analytical system model.
  2. *Given:* An abstract computational task with specific application requirements on phenomenal timescale characteristics, specified in terms of temporal input and output characteristics (in particular: only initial input and final output versus continuous stream of both; speed and memory scales in input; speed and memory and reactivity of output response); online or offline task; plus further conditions e.g. with regards to accuracy or robustness.
  3. *Wanted:* An abstract computational model for doing the task, extracted from the analytical model, which satisfies the given requirements, including a scheme for encoding the abstract input specification u in signals u^* which can be physically inputted, and for decoding observable signals y^* into the abstract output format y that is part of the task specification.

  Solving this abstract model design problem includes the definition of suitable meta variables x_i which can be (hierarchically) extracted from the analytical model and whose abstract transformation flowgraph (green arrows in Figure 1) leads to an overall abstract dynamical system such that the transformation diagram



  commutes (approximately) in the outward encode-process-decode and compute signal transformation paths, as well as in all internal subpaths which include extraction links (blue). The design of a suitable abstract model may involve the introduction of auxiliary meta variables (dotted green circle). The computational model variables

may have a variety of different modes of progression. This freedom of design may be crucial for finding a successful computational model.

Top-down physical realization tasks:

1. *Given:* An abstract computational model (suitable for some I/O transformation task with specific phenomenal timescale characteristics).
2. *Wanted:* an analytical system model from which the meta variables in the abstract model can be extracted, such that the analytical model can be actually fabricated, and such that the diagram above again commutes.

Solving this physical realization task is the reverse of the bottom-up task and includes the design of suitable analytical system variables x_i^* and their dynamical couplings, as well as possibly the introduction of auxiliary meta variables, such that the physical modes of progression $t[\text{sec}]$ or $n\Delta[\text{sec}]$ in the causal physics expressed in the analytical model can be transformed into the modes of progression used in the abstract model.

If only an abstract computational task with phenomenal timescale constraints is given, both an abstract-computational and an analytical-physical model must be found. This will involve a design process where the bottom-up and top-down modeling tasks are iterated. A good example is recent collaborative work done in MemScales [Payvand et al., 2021], where the Self-Organizing Recurrent Network (SORN) model of Lazar et al. [2009], which previously has only been digitally simulated, was realized on analog spiking hardware with memristive synapses. This resulted in a hardware-software co-design development process, where the abstract computational model was optimized for two physically available timescales of synaptic plasticity (represented through time constants in the analytical model). It was also optimized for the device variability and nonstationarity of the physical system. In the words of the researchers, "with the *technologically plausible algorithm design*, we aim to optimize the hardware implementation of algorithms by taking the hardware physics into account while developing the algorithm" [Payvand et al., 2021].

3 A zoo of computational mechanisms

In this second part of our report we give a condensed survey of computational mechanisms and phenomena which either directly relate to some sort of timescale manipulation, or could be used to that end in a natural manner, or provide useful background information to diagnose unexpected temporal behavior. We cover almost all mechanisms that we are currently aware of, but we are also aware that this listing is incomplete. We roughly order our reports according to the kind of timescale (of change, reactivity, memory, mixed and other).

3.1 Mechanisms impacting timescales of change

3.1.1 Explicit training of velocity changes in temporal pattern generation

In a variety of signal generation tasks one wishes to change the overall speed of change of the abstract computational model when it becomes executed by a physical system. For instance, in robot motor control one may wish to have motor pattern generation modules that have a velocity control input, such that for instance a walking gait or a pointing gesture can be generated

in slower or faster versions. In digital programming one can achieve this by letting the program numerically solve ODEs and globally scale all their time constants in proportion to the desired speedup / slowdown. But this convenience will not be available in abstract computational models extracted from analog neuromorphic or other unconventional hardware.

Motor control areas in biological brains or spinal neural *central pattern generators* obviously can change the speed of generated motor patterns. These systems are preferably modeled by RNNs in computational neuroscience and neuro-robotics, and RNNs in various degrees of abstraction from biology are a popular abstract model class in analog neuromorphic computing. This raises the question how the timescale of change (the "velocity") of a pattern-generating RNN can be controlled when the "trick" of global scaling of all time constants in ODEs is not possible. This question is relevant both for understanding biological RNNs and for non-digital neurocontrollers in robotics. It is also of theoretical interest for the general mathematical theory of dynamical systems.

An obvious method to obtain speed-controllable RNN pattern generators is to explicitly train them for this task in a supervised way, where the training input consists of a slowly varying (ramp) target speed indicator variable and the teacher output is the desired generated pattern in different velocity versions. It is not difficult to obtain robust frequency-controllable periodic pattern generators with this approach (e.g. in Jaeger [2007]).

A potential drawback of this method is that such training data may not be available in real-life situations, and that the speed modulation range is defined by a well-chosen training set of input-output combinations during the training phase, where the training examples should preferably be sampled rather densely across the desired velocity range. Any untrained input might lead to an undesired output shape modulation when the trained system is later used. Furthermore we find it not very plausible that biological motor speed control is learnt in this way.

3.1.2 Controlling the geometry of effective state space

Given the potential drawbacks of the direct training approach mentioned in the previous subsection, one would like to afford of other methods for RNN pattern generation velocity control, which would require training examples sampled for smaller number of teaching velocities, and/or which would include active feedback control for improved stability and generalization. This would also be interesting for computational neuroscience. However, we are aware of surprisingly little research in this direction, with existing studies apparently being mostly confined to the analysis of low-dimensional oscillatory dynamical systems (an indicative brief survey is in wyffels et al. [2014]).

In own previous research we tackled this problem with another approach that aimed for generic and robust solutions and was neither based on specific properties of specific low-dimensional ODE models, nor on direct training.

Our work was based on the observation that, when an RNN is externally driven with a temporal pattern, the region in RNN state space which is visited by the entrained RNN states systematically varies in its location and geometry when the presentation speed of the driving signal is varied (Figure 4).

The guiding idea to exploit this phenomenon of velocity-dependent changes in the geometry of the visited state space is to revert it: by controlling the admissible state space region, induce velocity changes in the generated patterns.

We pursued this line in two different ways:

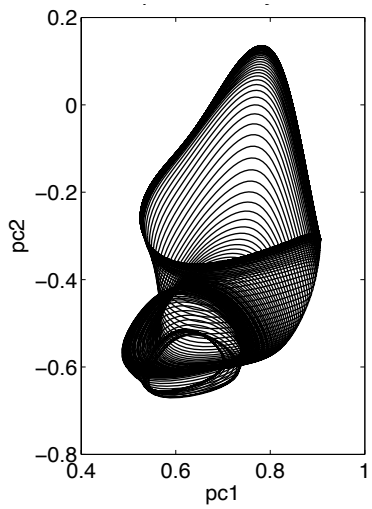


Figure 4: Change of 1000-unit RNN state space region when the network is driven with a periodic pattern of increasing frequency (here from slowest to fastest with a factor of 3). The plotting axes are the two first principal components of the RNN states (Image taken from wyffels et al. [2014]).

Proportional control of RNN bias: In wyffels et al. [2014], the velocity-dependent state space region was characterized simply by the mean of the states visited at a given frequency. This was translated to a bias vector $\mathbf{b}$ in a leaky integrator RNN upadate equation whose gain γ was controlled by a proportional feedback controller that compared the frequency of the generated output pattern y with a target signal:

$$\mathbf{x}(n+1) = (1-\lambda)\,\mathbf{x}(n) + \lambda\,\tanh(\mathbf{W}_{\text{rec}}\,\mathbf{x}(n) + \mathbf{W}_{\text{feedback}}\,y(n) + \gamma\,\mathbf{b}).$$

A velocity range of a factor 4 for sinewave patterns could be obtained with this method.

Conceptor-based velocity control: In Jaeger [2014], the velocity-dependent state space region was characterized by the principal axes of the state correlation matrix. This information was turned into a neural filter per task version, called *conceptors*, which are matrices that can be inserted into the recurrent network state update loop and then act as a projection operation which nudged states outside the task-specific region back into it. With regards to velocity control, Jaeger [2014] documents a simulation study where the training data consisted of only two sinewave samples whose frequency difference was 1 (in arbitrary units), from which two conceptor matrices were computed. By linear inter- and extrapolation between and beyond these two matrices, a range of new conceptors was synthesized after training which made the RNN oscillate in a frequency range of 5 units.

Conceptors have so far been mainly explored as a mechanism to train an RNN to generate a large number of different selectable temporal patterns (among them periodic patterns, chaotic patterns, and high-dimensional human motion patterns), yielding a lifelong trainable long-term memory mechanism [Jaeger, 2017]. As a bonus, having a conceptor in the loop has a strong stabilizing effect on the RNN dynamics. This effect may be particularly useful in computing systems based on noisy, low-precision, or modestly nonstationary hardware dynamics.

3.1.3 Input-induced timescales of change in adiabatic computing processes

Abstract and analytical models of computing systems that have internal feedback loops (which is typically the case) can be analyzed with formal tools from dynamical systems theory. In

many cases the internal timescales of the processing systems (often referred to as *intrinsic* or *native* system timescales) will be fast compared to the timescales of the incoming input signals. When there is such a *separation of timescales*, and when the internal dynamics satisfies certain stability conditions, the internal processing will *adiabatically* follow the slow input. In such a scenario, the input can be regarded as a control parameter that defines an attractor in the internal system dynamics, and due to the timescale separation this dynamics always "has enough time" to converge close to the attractor.

In the simplest case, the input defines a point attractor which moves as slow as the input changes, and the system dynamics tracks this stable fixed point with a small lag. This sort of fixed-point tracking dynamics has been postulated as a working principle for certain motor control tasks in neuroscience [Bizzi et al., 1992], and it can be regarded as a dynamical systems view on tracking feedback control.

The slowly modulated attractor need not be a point attractor. If it is a periodic attractor, the slow input may, for instance, change the geometry, offset, or frequency of the attractor. From a dynamical systems perspective this would be the natural modeling and design approach for *central pattern generators* in neural and robotic motor control.

The idea of adiabatic tracking is natural and has been explored and exploited many times. However, upon closer inspection, a number of difficulties appear.

A conceptual difficulty concerns what should be understood as the "native timescale" as compared to the input timescale. In the mathematical theory of multiple timescale dynamics [Kuehn, 2015], the main approach to formalization is to analyse ODE-based models whose equations split into two groups, giving a "fast" subsystem with small time constants and a "slow" subsystem with large time constants. Timescales are here identified with time constants. The mathematical analysis of such *slow-fast systems* has been carried out to great depths, giving rise to *singular perturbation theory*. The fast dynamics can often be approximately understood as transient convergence toward attracting *invariant manifolds*, and the slow dynamics as state evolution inside these, with the current state acting as point attractor in directions orthogonal to the manifold. However, in computing systems the abstract computational model may not be based on ODEs or other formalisms that have (analogs of) time constants, and the input signal can remain entirely un-modeled. This makes it difficult if not inappropriate to apply the theory of slow-fast systems. In our terminology, it would be appropriate to consider the input timescales and the induced slow changes of the internal attractors as timescales of change, and the transient tracking dynamics as timescales of reactivity.

A further difficulty is that the slow input, regarded as control parameter in the sense of dynamical systems theory, may induce bifurcations in the computing system. This will render it difficult to purposely and predictably design computing systems as natively fast systems controlled by slow input.

Another difficulty is that inputs may not be guaranteed to be slow at all times. When there are fast changes in the input, the tracking may lose contact with the currently followed attractor and be pushed into another attractor's basin.

3.1.4 Slow feature analysis

Real-life signals $(\mathbf{p}(n))_{n\in\mathbb{N}}$, where each $\mathbf{p}(n) \in \mathbb{R}^d$ is by itself a high-dimensional *pattern* like a video frame, a sensory input, or a neural network state, typically contain different *features* that change on different timescales. By definition, a feature is a function $f : \mathbb{R}^d/to\mathbb{R}$. Features can

be arbitrarily complex and nonlinear. Consider, for instance, a 5-minute photosafari videoclip taken from a car driving through a savannah. For one minute, the video catches and holds the sight of a leopard. Then a feature f which recognizes the presence of the leopard in a video frame — i.e. this feature returns a value of 1 if this animal is present and 0 else — is a *slow* feature because most of the time it is constant 0 or 1, and changes this value only twice.

Slow feature analysis (SFA, Wiskott [26-27], Wiskott and Sejnowski [2002]) is a neural learning algorithm to automatically identify slow features in such pattern signals. Its mathematical core is a combination of a nonlinear expansion of the pattern signal (optional; this step is needed if one wishes to obtain nonlinear filters) followed by signal whitening and a final principal component analysis to detect the principal direction in the expanded pattern correlation space that has the slowest average temporal variation. By repeating the last step, increasingly less slow orthogonal signals can be identified. The method has sometimes been used in machine learning, but its largest impact is in computational neuroscience. Biologically plausible adaptations of the basic learning algorithm have been developed, and SFA has been proposed as a mechanism that trains hippocampal place cells (these cells encode specific locations in an animal's habitat; such locations do not move — they are maximally slow, which is the key for their identification with SFA). An compact summary of SFA is Wiskott et al. [2011].

3.1.5 Slow transients

Dynamical systems can exhibit periods of very slow transient state progression alternating with periods of fast change. This can be due to a wide variety of more or less unrelated mathematical phenomena, for instance slow passages near saddle points in homo- or heteroclinic channels, critical slowdown near bifurcations, relaxation oscillators, gradient descent dynamics in potential landscapes with locally different curvatures in different directions, dynamics on center manifolds, or drift along line attractors. We are not aware of dedicated computational exploits of such phenomena and hint at this assortment of phenomena only to highlight that a rigorous mathematical analysis of slowness phenomena that one encounters in practice (maybe unexpectedly) needs a case-by-case investigation. We point out that in computer simulations of complex dynamical systems one sometimes finds that the observed state evolution seems to be coming to a standstill, and may erroneously conclude that it is approaching a fixed point attractor — while in fact the trajectory would have picked up speed again if one would only have continued the simulation for a *very* long time. Specifically, in machine learning one should not prematurely stop an iterative model optimization process when one judges that the model accuracy has reached a plateau.

Interestingly, a similarly rich compendium of scenarios for *fast* transient behavior seems not to have been explored.

3.1.6 Time un-warping

In machine learning for temporal tasks, input signals may be *time-warped*, that is they exhibit local velocity variations. For example, a human speaker will sometimes speak faster (when he/she is excited, for instance) than at other times; or sometimes linger on a vocal or make a pause while thinking about how to continue the sentence. Similar warpings occur in almost any real-life observation stream. This is a nontrivial challenge for machine learning algorithms both in training and exploitation because these algorithms are "warping-unaware" and can become seriously de-railed. This would happen, for instance, in a speech recognition task where input time windows of a feedforward neural network model are scaled to cover an entire spoken word,

but the speaker is lingering and the entire input window is filled with repetitions of the phoneme "aaaa...".

A number of machine learning methods have been developed to cope with this problem, especially in speech recognition. A classical signal processing technique is to unwarp the observed signal (by super- and subsampling) by optimizing its match against standardized reference patterns. An obvious difficulty is to determine which standardized reference pattern has to be used at any given time; this can be done by dynamic programming methods, as in [Itakura, 1975]. Hidden Markov models, which for two decades have been the primary workhorse for speech recognition before they became superseded by deep learning methods, account for repetitive speech recording frames by self-transitions of Markov states Rabiner [1990]. Sun et al. [1993] (also containing a short overview of previous neural-network techniques for unwarping) propose to make RNNs warping-invariant by making the stepsize Δt proportional to the difference in norm between two incoming speech vectors. This method has been adapted to reservoir computing in Lukoševičius et al. [2006].

In today's deep learning, the unwarping problem is typically solved by RNN models which include an *attention* mechanism that learns to identify task-relevant features in the incoming signal and temporarily memorizes them until needed. This solution of unwarping by a working memory mechanism has proven a breakthrough innovation for deep learning applications in speech and text processing [Bahdanau et al., 2015].

3.1.7 High-frequency oscillations from coupling low-frequency oscillators

It it not difficult to couple two nonlinear oscillators, each having a period Δ, such that from the coupled system one can extract a signal with period $\Delta/2$. All one has to do is to establish a coupling that lets the two oscillators evolve with a maximal relative phase difference, and extract a combined signal from both oscillators. This can be generalized to obtain period shortenings for higher than double frequencies n/Δ. Higher-frequency oscillator systems obtained from coupling lower-frequency ones could, for instance, be used to create faster clock signals than would be obtainable from "slow" oscillators available in the physical computing system.

The phenomenology of coupled oscillators has been studied in breadth and depth in theoretical physics. We can imagine numerous exploitations of purposely designed coupling schemes, but are not aware that this option has been pursued in the literature.

3.2 Mechanisms impacting timescales of reactivity

3.2.1 Increasing reactivity through arousal

In cognitive neuroscience, the term "arousal" refers to a spectrum of physiological conditions and cognitive effects that indicate a degree of wakefulness, alertness, expectancy, vigilance, acuity of sensory impressions, attention, speed of decision making, and the like. It is a not universally defined, multi-causal and multi-functional phenomenon. The term is used quite differently in different theories of cognitive psychology and neuroscience. Giving an overview is beyond our expertise.

In digital processing systems, a faint reflection of arousal could be recognized in adaptive clock frequency regulation, where the clock rate is increased at the expense of higher energy consumption when high computational loads have to be dealt with. The increase in energy con-

sumption has a parallel in neurobiology, where likewise higher arousal is connected with increased metabolic rates.

With regards to computing systems based on analog neuromorphic or other unconventional physical systems, a not all too far-fetched analogue of arousal may be seen in options to run the same physical system at different levels of energy throughput, for instance by increasing the overall voltage in electronic systems. Unless the physical system and its analytical model variables respond entirely linearly to changes in energy throughput, the physical dynamics will undergo nonlinear changes (in the sense that phase portraits at different energizing levels cannot be linearly transformed into each other) and possibly bifurcate (phase portraits cannot be continuously mapped onto each other). Abstract computational models for such "arousable" physical substrates would change their information processing characteristics in many ways, from subtle timing changes to qualitatively different input responses. In order to exploit "physical arousal" to help solving computational tasks, much care would be needed to balance desired effects (like faster reaction times) against undesired ones (like higher error rates or increased energy consumption). We are not aware that arousal-related mechanisms have been systematically explored for non-digital computing tasks. A study of the respective literature in psychology and cognitive neuroscience could become a well of inspiration for identifying and engineering innovative information processing models that make use of energy-dependent nonlinear physical effects.

3.2.2 Sensitivity control by positive and negative feedback

While systematic investigations of arousal-related processing modulations at the level of comprehensive computational tasks are missing to our (limited) knowledge, excitability modulation has been studied by theoretical neuroscientists at local levels of neural dynamics. Relating to this tradition, [Sepulchre et al., 2019] begin to develop an analysis of how global arousal signals sent to a network of nonlinear processing units lead to qualitative changes of the network dynamics, while the arousal signal are causally effective only at the microlevel of changing efficiencies of excitatory and inhibitory synaptic connections. The authors formulate their approach in electrodynamical terms for abstract neuron models and give examples from neuroscience, but also point out how their insights would transfer to other kinds of multiscale collective systems, whether they are computing systems (like analog circuits made from a collective of microcircuits) or non-computing systems (like city traffic flows).

Starting from specific assumptions about the dynamics of single neurons, they exemplify their approach of obtaining global and fast control over the entire network's collective behavior by local excitability modulation with three case studies. These case studies range in network size from 2-neuron motifs to a 5-neuron model of the crab somatogastric ganglion (a deeply investigated model system in neuroscience), and further to synthetic recurrent neural networks with 40 or 160 neurons. In the first two demonstrations, the global system behavior can be analytically predicted from local excitability modulation, while the larger network systems are explored by simulation.

A highly didactic workout of this approach, with suggestions for applications in neuromorphic computing, is given in Ribar and Sepulchre [2021].

A main objective for this work was to explain (and possibly use for engineering) how complex, collective computing systems can adapt their information processing *fast* to changing demands. They contrast their approach with traditional engineering solutions to guide the behavior of multiscale systems with hierarchical control architectures. Such classical control

architectures (paradigmatic, even an industry standard: Albus [1993]) achieve a global control objective through a cascade of control levels, where the global task is formulated and controlled at the highest level, to become hierarchically broken down into sub-control loops that address increasingly local subsystems. According to the authors, a principal drawback of such architectures is that each step through the hierarchy mandates a separation of timescales, such that the global control is necessarily slow — which cannot explain why animal brains can *quickly* respond to changing task demands. The approach of Sepulchre et al. is not hierarchical and the global control objective becomes immediately translated into local control. While we find this idea compelling in principle, task-specific or complex control objectives still remain beyond the reach of this approach.

3.3 Mechanisms impacting timescales of memory

3.3.1 Effects of numerical precision for dynamical memory

In recurrent neural network (RNN) models, information contained in inputs $\mathbf{u}(\mathrm{t})$ is transferred into neural states $\mathbf{x}(\mathrm{t})$ through their input laws, for example via $\mathbf{x}(n) = \sigma(W_{\text{recurrent}}\,\mathbf{x}(n-1) + W_{\text{input}}\,\mathbf{u}(n))$ (we discuss this here for dimensionless discrete time n, but our discussion transfers to other modes of progression (t[sec], $t^{\emptyset}$, $n\Delta$[sec] in an obvious way). Information that has been "infused" at time n in this way becomes amalgamated with the previous state $\mathbf{x}(n-1)$, and this mixture is transported forward to the next timestep where it is mixed with the next input via

$$\begin{aligned}\mathbf{x}(n+1) &= \sigma(W_{\text{rec}}\,\mathbf{x}(n) + W_{\text{in}}\,\mathbf{u}(n+1)) \\ &= \sigma(W_{\text{rec}}\,\sigma(W_{\text{rec}}\,\mathbf{x}(n-1) + W_{\text{in}}\,\mathbf{u}(n)) + W_{\text{in}}\,\mathbf{u}(n+1)),\end{aligned}$$

etc. Thus a network state $\mathbf{x}(n)$ will contain traces of all previous inputs. This is the principle of *dynamical memory*. However, input traces "thin out" as time progresses because they are superimposed by inputs which arrive later, and furthermore they become nonlinearly transformed at each new timestep. To make use of these memory traces and obtain a memory signal $y(n)$ for some subsequent processing operations, a *decoding* filter $y(n) = D(\mathbf{x}(n), y(n-1))$ must be designed. This decoding filter may have an internal state $\mathbf{d}(n)$:

$$\begin{aligned}\mathbf{d}(n) &= F(\mathbf{d}(n-1), \mathbf{x}(n), y(n-1)) \\ y(n) &= D(\mathbf{d}(n), \mathbf{x}(n), y(n-1)).\end{aligned}$$

Since dynamical neural memory is important both in neuroscience and machine learning, a substantial literature is available where this effect has been studied for different sorts of RNN models, tasks, and decoding operations (a selection: Jaeger [2002], Maass et al. [2002], Ganguli et al. [2008], Hermans and Schrauwen [2010], Charles et al. [2017], Voelker et al. [2019], Gonon et al. [2020]).

Due to the thinning-out and iterated nonlinear transformations, such decoding filters become increasingly complex, nonlinear, and vulnerable to noise when longer memory spans are needed. Whether a successful decoding is possible furthermore depends on *which aspect* of previously inserted information has to be retrieved.

An obviously favorable condition for achieving long memory spans is a high numerical precision in the network model. The more precisely (greater bitlength, less noise) network states are computed, the more information inserted into them at earlier time will be recoverable. In Turing-equivalent digital models of computing, arbitrarily high bit precision and zero noise are

possible. With regards to non-digital computing hardware, this convenience is unavailable: abstract computing models that can be extracted from the corresponding analytical system models will necessarily have limited precision.

While research on dynamical memory is mostly carried out in a context of RNN models, the basic insight that numerical precision correlates with achievable memory spans likewise applies to other dynamical systems which may be formalized in other ways than as models of RNNs.

High numerical accuracy with its benefits for preservation of memory traces over extended timespans is a convenience that is only available in digital computing models. When the given hardware system is non-digital and quantitative values of meta-variables are not *simulated* in the abstract computational model, but directly represent physical quantities, these meta-variables in the abstract computational model inherit the noisiness and other numerical imprecisions from the physical system. A method to compensate to some extent for this lack of numerical precision and realize abstract computations that still have reliable dynamical memory characteristics is a simple and generic technique which has been independently (re-)discovered several times for different purposes: as *self-predicting networks* for augmenting the performance of reservoir computing techniques [Mayer and Browne, 2004], as *equilibration* for enabling external controllability of RNN dynamics [Jaeger, 2010], as *reservoir regularization* for improved stability of neural motor controllers [Reinhart and Steil, 2011], as *self-sensing networks* for making RNN training methods more flexible [Sussillo and Abbott, 2012], and as *innate training* for reliable, noise-resistant reproducible chaotic patterns in short-term memory RNNs [Laje and Buonomano, 2013]. This method was a key enabler for the conceptor-based training of RNNs to generate a large number of different temporal patterns in Jaeger [2014], an for the *reservoir transfer* method proposed by He et al. [2019], where some of the beneficial numerical qualities of a RNN simulated on a digital computer with high precision were transferred to an RNN implemented on an analog spiking neurochip.

This method simply re-computes the internal weights $\mathbf{W}_{\text{rec}}$ of an RNN by driving this network with some input signal $\mathbf{u}(n)$, collecting the resulting RNN states $\mathbf{x}(n)$ and then calculating a new weight matrix $\mathbf{W}^*_{\text{rec}}$ as

$$\mathbf{W}^*_{\text{rec}} = \text{argmin}_{\tilde{\mathbf{W}}} \sum_n \| \tilde{\mathbf{W}}\mathbf{x}(n) - \mathbf{W}_{\text{rec}}\,\mathbf{x}(n) \|^2 + \alpha^2 \| \tilde{\mathbf{W}} \|^2_{\text{Fro}} .$$

This gives a Tychonov-regularized (also known as ridge regression) version of the original weight. In plain wording: the RNN with the new weights should approximately replay the original state sequence with minimized weights.

3.3.2 Effects of system size for dynamical memory

This subsection can be seen as a postscriptum to the previous one, where we pointed out that the effectively usable information encoding capacity of a state $\mathbf{x}(n)$ grows with numerical precision and reduction of noise. Another way to increase the capacity of states $\mathbf{x}(n)$ to transport input information forward through time — the same information for longer times, or more information for the same time — is to increase the dimension of state vectors.

In unconventional computing materials with a spatiotemporal dynamics that is continuous in a continuous space $\mathcal{S}$, states are not vectors but functions $\mathbf{x}(n) : \mathcal{S} \to \mathbb{R}$, which are infinite-dimensional. In mathematical principle, this might allow for unbounded information encoding

capacity if the spatial patterns could become arbitrarily finely structured. In real materials however the local spatial pattern variation is ultimately limited by the discreteness of materials at the atomic level, and well before this limit is reached by correlations and statistical dependencies between local signals at separate points in the substrate. Yet, it seems promising to engineer continuously extended nonlinearly excitable material substrates and study how abstract computational models can be extracted from them.

3.3.3 Effects of system heterogeneity for dynamical memory

We continue the thread from the previous two subsections. Regardless of whether states $\mathbf{x}(n)$ of a dynamical memory subsystem are finite vectors or functions, the amount of information that can be encoded and transported forward in time is limited by statistical dependencies between vector components or different locations in a continuous medium. For higher capacities of information encodings it is thus favorable to have states whose component dynamics are only weakly coupled in the sense of small mutual information. One way to aim for this goal is to design memory subsystems whose within-state coupling laws are heterogeneous in some way.

In the field of reservoir computing, a diversity of proposals have been made to design RNN matrices W_{rec} and feedback gain regimes which lead to "rich" dynamics. Many of these proposals recommend specific topological RNN connection structures (e.g. modular, small-world, hierarchical, or even a ring topology), others tailor algebraic characteristics W_{rec} (in particular the eigenvalue spectrum) and yet others invoke unsupervised learning algorithms which aim at increasing the diversity of within-reservoir signals, sometimes in a way that is optimized for specific input statistics [Lukosevicius, 2012]. In our perception, reported performance gains in memory-demanding tasks have not been spectacular. This may have to do with the fact that most reservoir computing solutions use linear decoding functions which essentially can take advantage only of linear decorrelational effects of within-reservoir signals induced by such reservoir optimization methods. More decisive benefits might be obtained from increased statistical independence of reservoir signals, which however would require nonlinear decoding methods to become exploited. We are not aware of research in this direction.

3.3.4 Effects of (non)linearity for dynamical memory

If an abstract computational model includes a submodel which is a dynamical system with numerical state vectors (for instance, a RNN), one can study the effects of (non)linearity of this dynamical subsystem on its dynamical memory properties. To this end one first would have to define a measure for the degree of nonlinearity. One straightforward approach would be to consider the Taylor expansion of the state update function and define the strength of nonlinearity as the ratio of the sum of all higher-order polynomial coefficients over the linear coefficient. Then one could analyse the effects of nonlinearity on dynamical memory characteristics.

This line of investigation has not been explored yet in mathematical depth. In the reservoir computing (RC) literature we find a mostly implicit, somewhat vague and general consensus that linear reservoirs work best for dynamical memory. But this view may be premature, and caused (i) by the general reliance in RC on linear decoding functions (the trainable "readouts" in the RC terminology) in conjunction with (ii) the fact that linear reservoirs are easier to analyse than nonlinear ones. If one admits nonlinear readouts, one might get very long dynamical memory with very nonlinear reservoirs. To see this, consider a reservoir RNN of the kind $\mathbf{x}(n+1) = \tanh(W_{\text{rec}}\,\mathbf{x}(n) + W_{\text{in}} + \mathbf{u}(n+1))$. When W_{rec} and W_{in} are scaled to have large absolute

matrix elements, such a reservoir will develop an almost binary state sequence with almost all state components $x_i(n)$ being close to $+1$ or -1. If will not be difficult to (hand-)design W_{rec} and W_{in} such that for input signals from a specific task, certain key events of kind A in the input will switch one of the reservoir states to (say) $+1$ and leave it there until another kind of key event B switches this back to -1. A threshold function readout could decode that this "first A then B" sequence in the input and thus implement a specific dynamical memory that has very long or even unbounded memory spans. Reservoir models based on FPGAs would directly realize dynamics of this kind. Research on discrete switching RNNs can look back on a venerable history where such systems have been studied as Boolean networks [McCulloch and Pitts, 1943], but their study within the RC paradigm have only begun [Bertschinger and Natschläger, 2004, Verstraeten et al., 2005, Legenstein and Maass, 2007, Tanaka et al., 2019]).

Linearity has also been found / declared as conducive for long dynamical memory in the deep learning field, namely through long short-term memory (LSTM) units [Hochreiter and Schmidhuber, 1997] and their variants like the gated recurrent unit (GRU, Cho et al. [2014]). These are small neural circuits embedded in RNNs which can be trained by backpropagation through time to store a real number (typically between 0 and 1) with an exponential decay rate that can be adapted through a trained control mechanism. The functional core of such a neural building block is a linear neuron. Further neurons in these circuits provide trainable "write and read" functionality. These units can thus be seen as elementary working memory devices, though this view was not the reason why LSTM units were originally proposed (the motivation was to mitigate the vanishing gradient problem of gradient descent optimization in neural network training). LSTM and GRU units are a pivotal enabler for deep learning solutions in temporal processing tasks. However, their mechanism and their trainability hinges on the high-precision, noiseless numerics of digital simulations of RNNs with rate neuron models, which renders them not directly applicable in analog spiking neuromorphic hardware systems. We mention them here for completeness.

3.3.5 Tuning networks close to criticality / chaos

A popular theme in the RNN literature — both in theoretical neuroscience and machine learning, especially but not exclusively in the RC field — is that some sorts of computational performance is optimal if the internal feedback gains in an RNN are tuned almost, but not quite high enough to render the resulting self-excited dynamics chaotic. This is called "edge of chaos" or, more rarely, "edge of criticality". This dynamical regime is characterized by particularly long dynamical memory spans and has been claimed to be optimal for computational performance.

Some cautionary remarks:

- Chaos and criticality, two terms that are surprisingly often used in an undifferentiating manner within the same article, are two fundamentally different phenomena. Chaos refers to a specific kind of attractors in dynamical systems, which can appear already in 1-dimensional discrete-time systems (iterated maps) and 3-dimensional continuous-time systems (described by ODEs). Chaos is typically defined and discussed in the context of deterministic dynamics in continuous state spaces, and generalizations to stochastic or discrete-state systems need some care. Criticality, in contrast, is a concept of statistical physics and is properly defined only in the infinite-size limit of stochastic systems. The fact that both concepts are often confounded may be due to a superficial similarity: complex systems (deterministic for chaos, stochastic for criticality) dramatically change their qualitative behavior when control parameters pass a critical value, which leads to a *bifurcation* (in the case of entering a chaotic regime) or to a *phase transition* (for criticality). That

bifurcations and phase transitions are entirely different concepts in two entirely different mathematical frameworks is apparently not apparent to a sizable number of authors and reviewers.

- In discussions of "edge of chaos" the context of reservoir computing it is often tacitly assumed that a reservoir transits from the dynamical mode where it has the echo state property Jaeger [2001] directly into a chaotic regime when a specific RC control parameter (the spectral radius of the reservoir weight matrix) exceeds a critical value. This is not generally true. Often the reservoir passes from the ESP regime first into other attractor regimes (fixed point, periodic) before it further transits to chaos; and some reservoirs do not transit into chaos at all regardless of how large that control parameter is scaled. Furthermore, standard RNN models in RC are deterministic, and using the word "edge of criticality" to describe behavioral changes is misguided.
- The concept of chaos is primarily defined for autonomous dynamical systems without input. But computing and signal-processing systems (brains, reservoirs and others) are eminently input-driven systems. Extensions of the mathematical concept of chaos to input-driven systems are nontrivial [Manjunath et al., 2012, Manjunath and Jaeger, 2014], and a generally accepted general definition of chaos for such systems is not available. Authors writing about the "edge of chaos" are mostly unaware that the "chaos" they discuss is not mathematically well-defined.
- Numerous papers with RNN simulation studies seem to re-confirm that "computing at the edge of chaos", or "at the edge of criticality" lead to optimal computational performance. However, these works almost invariably repeat variants of always the same few basic computational tasks — tasks which happen to be just of the kind that support this claim. This has led to a perception that "it is a well accepted fact that the computational potential of recurrent neural networks (RNNs) is optimized at what is called the edge of criticality" (first sentence in a rather recent article in a top-tier journal, reworded here a little to make it unsearchable on Google). As antidote let us cite verbatim from another, highly cited, much earlier article written by more clear-sighted authors: "Our experiment produced very different results, and we suggest that the interpretation of the original results [*finding that complex computational tasks are best served at the "edge of chaos"*] is not correct" [Mitchell et al., 1993], and "... best computational power does not necessarily correspond to the edge of chaos" [Legenstein and Maass, 2007]. In fact, practicians of reservoir computing who in their lives have carefully optimize reservoirs for a variety of tasks (such as the first author of this report) will regularly have witnessed that their optimized reservoirs land in a stable regime far away from any edge. It would be worth a socio-psychological study in scientific opinion-spreading to reveal why the "edge of chaos" myth continues to flourish.

Having deflated this myth, we nonetheless want to give a cautious summary appraisal of a certain class of effects which have a bearing on the study of dynamical memory:

- When feedback gains (in RC: the spectral radius) or related control parameters in recurrent, parallel, high-dimensional information processing systems (like reservoir RNNs or cellular automata) are scaled up toward, but not into a dynamical regime where self-excitation would override the entraining effects of input signals,
- the state trajectories of the processing system begin to develop autocorrelations (or other nonlinearly or statistically defined) dependency measures across time lags with increasingly "fat tails", indicating an impending transition from the usually observed exponential decay of these memory curves to only power-law decay profiles (so-called "fat tails" or

"1/f spectra"),

- which in turn may be beneficial in temporal tasks that require the integration of input information over several timescales, since memory curves with fat tails indicate that input information decays slower than exponentially.

A conceptually clean, mathematically correct, and unifying analysis of such effects remains to be done.

3.3.6 Oscillation-based dynamical memory

A possibility to encode information in a dynamical memory subsystem for long memory spans is to

- design the memory subsystem as a collective of uncoupled oscillators with different frequencies,
- encode input information at initial time by letting it modulate the relative phases of the oscillators,
- decode a desired output by a detector which is sensitive to these relative phases.

We are not aware of studies of such memory subsystems in the literature, but can report from own simulation experiments in a reservoir computing setting [Jaeger, 2012]. The task mimics an experimental design which is often used in cognitive psychology to study working memory in animals and humans. We used a version of this type of task that had before been used for learnability demonstrations of long-term memory spans in the deep learning field [Martens and Sutskever, 2011].

This version of the task was specified as follows. At the beginning of each experiment, a binary 2-dimensional, 5-timestep pattern (of which there are 32 different ones) is presented to the network. Then, for a period of duration $T_0 1$, a constant distractor input is given. After this, i.e. at time $T_0 + 5$, a cue signal is given as a spike input, after which in the final 5 time steps, the memory pattern has to be reproduced in the output units.

Here we report one of several variations of how this task was solved with RC methods [Jaeger, 2012]. For a memory span of $T_0 = 1000$ (measured in discrete network updates n) we designed a special reservoir weight matrix which realized a reservoir dynamics of 30 isolated oscillators with randomly chosen frequencies, and a "bulk" part of the reservoir which was randomly connected in the spirit of reservoir computing, and driven by input and the oscillators within the reservoir. All 32 input patterns could be retrieved without error with a linear decoder trained by linear regression, using a reservoir size of 500 neurons (including the oscillator subreservoirs).

The presence of the oscillators within the reservoir, as well as the random "bulk" part of the reservoir, were crucial to solve this task. Without oscillator sub-reservoirs, a reservoir size of 2000 neurons was needed to retrieve the 32 patterns after the much shorter waiting time $T_0 = 200$. The random "bulk" part of the reservoir presumably was needed to aid the decoder by supplying a high-dimensional nonlinear expansion of the oscillator signals which carried the input information through time.

We also report that a (much) more difficult version of this task with input pattern of length 10 and $T_0 = 200$ could not be solved with the backpropagation-based RNN training methods available in 2011, as reported by Martens and Sutskever [2011]. Using the oscillator-based

reservoir approach, it could be perfectly solved for $T_0 = 300$ with a 2000-neuron reservoir, requiring merely 121 seconds of training time on a 2.9 GHz, 8 GB RAM PC without GPU extension.

Oscillator-based approaches for dynamical memory subsystems may hold promises in neuromorphic / physical computing, because oscillators can be realized in hardware from fast devices with small time constants, yet the oscillations persist stably for long times.

3.3.7 Attractor-based working memory

While there is no universally defined and shared terminology, we recall that the concept of working memory usually implies that

- working memory systems can store only a limited, small number of separate memory items;
- memory items are "stored" and "recalled" by explicit control mechanisms;
- once stored, memory items can persist in memory for extended and potentially unbounded timespans, which however may require special mechanisms of attention or rehearsal;
- after retrieval, the item is deleted from the working memory, freeing capacity to store another item. In neural systems it is typically implied that the storage of an item is a dynamical phenomenon which does not lead to persistent structural (synaptic long-term memory) changes.

In digital computing systems, designing working memory mechanisms poses no principled difficulties with respect to hardware or control architectures — except that the notorious bottleneck in von-Neumann architectures is a major obstacle for computational speed and energy efficiency — and we will not further discuss it. In contrast, it is not easy to obtain working memory functionality in neural processing models, neither in neuroscience modeling nor in artificial neural network engineering. The difficulties are twofold: firstly, neural processing systems typically do not afford of non-volatile memory elements in which the information of a memory item can be safely encoded and preserved until recall and deletion, and secondly, the requisite mechanisms of evaluation (what should be put into working memory when), encoding, rehearsal, decoding and resetting are complex and need involved control mechanisms outside the item-storing subsystem proper. Artificial neural networks in machine learning are *trained*, and it is hard to train these control mechanisms. They need to be pre-designed at least partly, as in the spectacular study of designing a "neural Turing machine" [Graves et al., 2014, Graves et al, 2016] and in our RC-based model of a hierarchically structured working memory [Pascanu and Jaeger, 2011].

The literature in cognitive neuroscience on working memory is extremely extensive and we can only point out three surveys of Durstewitz et al. [2000], Baddeley [2003] and Fusi and Wang [2016].

The most natural candidate for representing and stable storing of discrete memory items in dynamical (neural) systems are *attractors*. This connects the study of working memory to a deep, ancient, and unresolved challenge in neuroscience and neural-network based machine learning, namely the problem of how discrete, symbolic "knowledge" can be neurally represented and processed. This theme can be traced back at least to the notion of cell assemblies formulated by Hebb [1949], reached a first culmination in the formal analysis of associative memories [Palm, 1980, Hopfield, 1982, Amit et al., 1985], and has since then diversified into a

range of increasingly complex models of interacting (partial) neural attractors and associated "attractor-like" phenomena [Yao and Freeman, 1990, Tsuda, 2001, Rabinovich et al., 2008, Sussillo and Barak, 2013, Schöner, 2019]. But other geometrical objects have also been selected as representatives of concepts, for instance solitons and waves [Lins and Schöner, 2014] or algebraically defined regions in state space [Tino et al., 1998, Jaeger, 2017], and many other cognitive entities besides concepts have been modeled by qualitative phenomena in dynamical systems [van Gelder and Port, 1995, Besold et al., 2017].

With regards to the topic of this report, attractors are of fundamental interest because they embody two timescales of change — namely the fast timescale of the ongoing state evolution in periodic and chaotic attractors, and the unlimited timescale of stably being "locked" in the attractor — and a timescale of reactivity, namely the (typically exponential) transient convergence toward the fully developed attractor dynamics. For purposes of supporting working memory functionality, one has to design or train memory subsystems which host many attractors, one for each possible discrete memory item. The "storing" task translates to control mechanisms which push the memory subsystem into the basin of attraction of the attractor that represents the targeted memory item.

There is a virtually unlimited freedom of designing dynamical (neural) systems which host many attractors and have control mechanisms to steer the overall system trajectory into specific attractors. The classical model of human working memory, the *phonological loop* model of Baddeley and Hitch [1974], Baddeley [1983], employs (in our terminology) a cyclic attractor. We cannot attempt a comprehensive overview on attractor-based working memory and instead point out to own research in this domain. In Jaeger and Eck [2008] a rather small reservoir system of 100 neurons is trained in a way that it hosts in the order of 10^7 (!) different cyclic attractors, each representing a short musical melody motif, where the system is steered into the desired attractor by entrainment to a short cue sequence. In Pascanu and Jaeger [2011] a neural reservoir is steered into different processing modes, each of them carrying out a text prediction task with different underlying grammars for each mode, where the switches between modes is triggered by opening and closing brackets that appear in the text. The processing is locked in the respective mode by switching external control neurons in a hierarchy of on-off states. In Jaeger [2017] *conceptors* are used as external control mechanism to switch an RNN into different trained attractors, which in a range of simulation experiments generated simple periodic signals, chaotic attractor dynamics, or human motion patterns. — These three studies are very different from each other, indicating the richness of design options for attractor-based working memory.

3.3.8 Tapped delay lines

In mathematical abstraction, a (discrete-time) tapped delay line is a memory element which at time n stores a moving window $(\mathbf{u}(n-L+1), \ldots, \mathbf{u}(n))$ from an input signal $(\mathbf{u}(n))_{n\in\mathbb{N}}$, together with a readout mechanism which can address and retrieve each of the $(\mathbf{u}(n-L+1), \ldots, \mathbf{u}(n))$ at time n. Continuous-time tapped delay lines are defined in an analog fashion. Tapped delay lines can thus be regarded as an elementary and at the same time powerful type of working memory which can serve a wide range of functions in temporal processing tasks.

Discrete-time tapped delay lines can obviously be programmed in digital general-purpose computers, though with a considerable computational overhead. For higher efficiency and faster access, another, hardware-based approach in digital systems is the use of clocked synchronous registers where the clock period is varied. Many technologies have been proposed for such digital register-based delays. However, this hinges on the availability of a clock signal which is

easily configured in periods that vary over a wide range of scales. This is an unsolved problem in practical large circuits which are required for a full system-on-chip. The trend is rather to go for a few well-defined clock periods that are hard-coded in the clock generation hardware.

Tapped delay line and related functionalities are necessary for a wide range of cognitive tasks that have been experimentally studied in cognitive neuroscience. Mauk and Buonomano [2004] give a survey, and Buonomano and Merzenich [1995] and Buonomano [2000] propose specific (spiking) neural circuits that may support these functionalities in the human brain. These circuits have a reservoir computing flavor in that they exploit the activation propagation in essentially random RNNs. Neural delay lines have also been modeled through traveling solitons in neural field models of cortical processing [Fard et al., 2015].

In the field of reservoir computing, a (by now) classical demonstration of echo state networks is to train them as tapped delay lines [Jaeger, 2002]. This has led to a rather extensive literature on the *memory capacity* of RNNs, which is essentially defined as the maximal window length L achievable with a given RNN trained as tapped delay line for i.i.d. input signals and using a linear readout operation (for example in Ganguli et al. [2008], Hermans and Schrauwen [2010], Charles et al. [2017], Voelker et al. [2019], Gonon et al. [2020]).

Physical realizations of delay lines in a reservoir computing setting have been variously studied, for instance in optical computing [Appeltant et al., 2011] or using traveling spin waves [Watt et al., 2021].

3.4 Mechanisms with mixed impacts

3.4.1 Frequency filtering and smoothing

In linear signal processing, numerous design principles for frequency filtering filters are known. Such filters dampen (“attenuate”) specific frequency components in their input signals and let the other components pass. One speaks of low-pass / high-pass filters if the high / low frequency components are attenuated, and of band-pass filters if frequencies lower or higher than a specified range of passed frequencies are attenuated.

When a signal becomes low-pass filtered, its timescales of change become longer: the filtered signal changes more slowly. This also changes reactivity: the filtered signal does not react to high-frequency components in the input anymore, and may react to slow input components only with a lag (if the filter is *causal*, i.e. the current output is determined on the basis only of past inputs, not of future ones). Finally, memory characteristics change too: filtered signals retain slower input components for longer and forget faster components more quickly or immediately. This account of altering memory characteristics however is limited to “information” that is encoded in frequency components.

The effects and uses of high-pass filtering are not directly complementary to the effects of low-pass filtering. While low-pass filtering effectively makes the signal change more slowly, high-pass filtering does not make it change faster. A common exploit is the removal of baseline drift from empirical measurement signals.

The mathematical theory of linear filters offers a host of designs to obtain frequency filters that optimize specific quality criteria that are mostly formulated in the frequency domain. Such mathematically defined filters can be perfectly realized (up to sampling effects) in digital signal processing systems by numerical computations. Analog processing hardware however can only practically realize a limited choice of filter designs and may need capacitors or inductances

whose area footprint in neuromorphic microchips may be prohibitive. Specifically, applications of on-chip low-pass filters for "slowing down" computations with highly miniaturized neuromorphic microchips are limited.

One of the simplest low-pass filters is the *exponential smoother*, which in its discrete-time version computes the output signal $y(t)$ iteratively from an input signal $s(t)$ with *leaking rate* λ by

$$y(t+\Delta) = (1-\Delta\,\lambda)\,y(t) + \Delta\,\lambda\,s(t),$$

where $0 \leq \Delta\,\lambda \leq 1$. In RNN theory, this filter is related to *leaky integrator neurons* in that the continuous-time RNN equation

$$\tau\,\dot{\mathbf{x}} = -\mathbf{x} + f(\mathbf{x},\mathbf{u}),$$

when integrated with the Euler approximation and stepsize Δ, becomes

$$\mathbf{x}(t+\Delta) = (1-\Delta/\tau)\,\mathbf{x}(t) + \Delta/\tau\,f(\mathbf{x}(t),\mathbf{u}(t+\Delta)),$$

hence the network states $\mathbf{x}$ can be regarded as exponentially smoothed versions of the "input" signal $f(\mathbf{x}(t),\mathbf{u}(t+\Delta))$ with leaking rate $1/\tau$.

Leaky integrator RNNs are often used to design layered RNN architectures where neurons in the bottom layer have a large leaking rate (i.e. it is "fast") and higher layers have increasingly smaller ones (i.e. they are increasingly slower). Input signals are typically fed to the bottom layer only, output signals are extracted from the top layer, or the bottom layer, or all layers, depending on the task. The guiding idea for such architectures is to mimic cognitive processing hierarchy of (sensor) signals, where the bottom layer provides the peripheral sensory interface (in some architectures also the motor command output interface) and increasingly higher layers process increasingly more abstract / comprehensive / complex "conceptual" representations. Letting higher layers operate more slowly allows their states to develop "conceptual" representations of information in the input stream that is integrated over longer memory timespans, while lower layers are devoted to extracting faster and more short-lived (and in spatially organized input, also smaller-scale) input detail. This basic idea unfolds into a bewildering spectrum of quite different designs for "intelligent" information processing tasks. A major design decision is whether these layers are only coupled unidirectionally bottom-up, or whether also top-down couplings are included; and another is whether these systems are trained in a supervised way (for example for classification, prediction, or motor control), or in an unsupervised or reinforcement learning paradigm (for concept discovery or behavior optimization in autonomous robotic agents). We cannot attempt a survey here and only arbitrarily pinpoint the lifetime works of Jun Tani who studied such layered, timescale-spreading RNN architectures for many purposes (e.g. robot control architectures [Nishimoto and Tani, 2009] or unsupervised gesture recognition [Jung et al., 2015]).

3.4.2 Event-based computing

In event-based computing models, local states remain unchanged until an incoming event triggers an update. Such events can range in complexity from a neural spike to calls for executing a complex subprogram in classical digital computing. The former is the type of implementations which we favor in MemScales. Since between events local states remain unchanged, event-based computing offers a direct opportunity for realizing slow or fast timescales of all our three sorts.

In a computing scheme with a mathematically pure event-based character (that is, the computational flowchart would contain no real-time timing conditions; state updates occur instantaneously; event signals are transmitted with zero delay; and whether a local processing subsystem updates its state only depends on whether logical enabling events have arrived), such slowdowns or speed-ups would be controlled exclusively by the frequency of input events. However, real computing systems do have nonzero physical times for state updates, signal transmission delays, and analog systems will have volatile states that do not remain unchanged between incoming events. Here the achievable timescale changes are limited by a variety of temporal interactions between and decay processes within local states and their updates. This mandates careful designs. To this end, modern (digital) microchip and computer architectures and real-time operating systems include Reliability, Availability, and Serviceability (RAS) modules which monitor ongoing operations and control the current processing speed. Consortium member Catthoor is active in RAS research [Rodopoulos et al., 2015, Noltsis et al., 2017].

4 Take-home messages and outlook

In this report we inspected the challenge of "computational extension of physically obtained timescales" in some detail. Let us summarize our main findings:

- The original problem statement in the title of this report is very much underspecified. In order to relate the two keywords "computational" and "physical" to each other in a productive and insightful way, we worked out the existing **theoretical framework for physics-based computing** of Stepney et al. [2018] in much greater detail. In our view, an analytical model of the physical computing system is a necessary interface model between the material hardware and abstract computational, "algorithmic" models:
  - "Computational" manipulations of timescales are defined and developed for the abstract computational model,
  - and the are connected to the physical system by formal extraction operations from the analytical model.
- One hardware basis can support many different abstract computational models. However, when one engineers a combination of a physical hardware system and an abstract computational model **for a given task**, one has to ensure that the abstract meta variables can be formally extracted from the analytical system model. In view of the enormous freedom in formulating abstract computational models — a blessing and a challenge — finding a solution to the combined hardware/computational model problem will likely result in iterative optimization cycles. A further difficulty is that analytical models are always approximate, which may mandate expensive prototype fabrication and experimental characterization.
- **"Timescales" is not a uniformly defined concept.** In order to carry out meaningful studies and do insightful engineering,
  - one must have a clear view of **how one formalizes temporal progression** in the first place, with an important distinction between modes of progression that are tied to external physical time and can be measured in seconds, versus abstracted mathematical concepts of serial order that are dimensionless — abstract computational models can use both sorts but the analytical system model is tied to physical time;
  - and one must be clearsighted about the **phenomenal aspects of "timescales"** that

one is speaking about; we distinguished the aspects of (metric) change, reactivity, and memory.

We furthermore pointed out that "timescales" *phenomena* are only indirectly connected to the *causal* influences that become expressed in the time constants of the analytical physical model.

- Decades of research in computational neuroscience, dynamical systems, theoretical physics, machine learning / neural networks and other fields have generated **a wealth of formal, algorithmical, and heuristic-practical techniques** for analysing, transforming, and exploiting timescale phenomena. We collected a "zoo" of 20 species, but are aware that we missed many more; that each of them would need a more in-depth scrutiny (formalization and survey) than what we could deliver here; and that for each of them, practical "tricks of the trade" would be welcome.

The deeper we delved into our subject matter, the more clearly did we perceive that we are only at the beginning of a long journey. Before we state what we see as important future work, let us share **an encouraging thought**:

- Perceiving the enormous degrees of freedom in designing computational models for a given hardware system, and the wealth of already well-studied computational mechanisms that relate to timescales, we find that one can (almost) always find a solution for a given task which frees the algorithmic model from the causal time constants of the physical infrastructure.

At present we still lack recognition, routine and recipes, and solving computational tasks that involve timescale challenges on non-digital hardware still requires heuristic trial-and-error experimentation. Only **future research** will bring us closer to a point where we can swiftly produce hardware engineering + computational model solutions to given tasks. In particular, we think that further work on the following subjects will bring advances:

- **Extend the repertoire of phenomenally defined timescales** beyond the three sorts that we registered in this report. For instance, we could imagine to introduce timescales of duration (how long does a process take), timescales of prediction, or timescales of information collection / sampling / integration.
- **More formal modes of progression** beyond the small assortment that we mentioned in our report would add more options for the design of abstract computational models. For instance, one could take a closer look at the classical temporal relations proposed by Allen [1991] in AI knowledge representation, or investigate how formalisms of temporal modal logic [Garson, 2014] can be used for algorithm design.
- We need a systematic study of how and when meta variables with a specific mode of progression can be extracted from other meta or analytical variables that have other modes of progression. It would be desirable to derive **a formal hierarchy of modes of progression**, such that modes higher in the hierarchy can be extracted with specified transformations from modes lower in the hierarchy.
- Finally, **time and space** are connected both in physical and abstract systems. Our report focused entirely on timescales. But physical systems are also organized on several spatial scales (metric or more generally topological), and computing tasks often have some kind of spatial organization in their data or task specification. There are many connections between time and space — starting from the elementary fact that motion needs both to be defined — which we cannot expand here. A complete study of timescales must

ultimately become connected to a study of spatial scales.

References

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

A. Adamatzky, editor. *Advances in Unconventional Computing Vol. 1: Theory, and Vol. 2: Prototypes, Models and Algorithms*. Springer International Publishing AG Switzerland, 2017.

A. Adamatzky. Towards fungal computer. *Interface Focus*, 8:20180029, 2018.

J. S. Albus. A reference model architecture for intelligent systems design. In P. J. Antsaklis and K. M. Passino, editors, *An Introduction to Intelligent and Autonomous Control*, chapter 2, pages 27–56. Kluwer Academic Publishers, 1993.

H. Alla and R. David. A modelling and analysis tool for discrete events systems: Continuous Petri net. *Performance Evaluation*, 33(3):175–199, 1998.

J. F. Allen. Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6(4):341–355, 1991.

D. J. Amit, H. Gutfreund, and H. Sompolinsky. Spin-glass models of neural networks. *Phys. Rev. A*, 32:1007–1018, 1985.

L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer. Information processing using a single dynamical node as complex system. *Nature Communications*, 2(468), 2011. DOI: 10.1038/ncomms1476.

A. Baddeley. Working memory: looking back and looking forward. *Nature Reviews: Neuroscience*, 4(10):829–839, 2003.

A. D. Baddeley. Working memory. *Philosophical Transactions of the Royal Society of London, Series B*, 302(1110):311–324, 1983.

A. D. Baddeley and G. Hitch. Working memory. *Psychology of Learning and Motivation*, 8: 47–89, 1974.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015. URL http://arxiv.org/abs/1409.0473v6.

F. C. Bartlett. *Remembering: a study in experimental and social psychology*. Cambridge University Press, 1932. Free legal online copies.

N. Bertschinger and T. Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004.

T. Besold, A. d'Avila Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kühnberger, L. C. Lamb, D. Lowd, P. Machado Vieira Lima, L. de Penning, G. Pinkas, H. Poon, and G. Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. arxiv manuscript, 2017. URL https://arxiv.org/pdf/1711.03902.

E. Bizzi, N. Hogan, F.A. Mussa-Ivaldi, and S. Giszter. Does the nervous system use equilibrium-point control to guide single and multiple joint movements? *Behavioral and Brain Sciences*, 15:603–613, 1992.

D. V. Buonomano. Decoding temporal information: A model based on short-term synaptic plasticity. *J. Neuroscience*, 20(3):1129–1141, 2000.

D. V. Buonomano and M. M. Merzenich. Temporal information transformed into a spatial code by a neural network with realistic properties. *Science*, 267:1028–1030, 1995.

C. Callender. Introduction. In C. Callender, editor, *The Oxford Handbook of Philosophy of Time*. Oxford University Press, 2011.

A. S. Charles, D. Yin, and C. J. Rozell. Distributed sequence memory of multidimensional inputs in recurrent networks. *JMRL*, 18:1–37, 2017.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arxiv manuscript https://arxiv.org/abs/1406.1078, 2014.

A. Clark. Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioural and Brain Sciences*, 36(3):1–86, 2013.

E. Cove, R. George, J. Frascaroli, S. Brivio, C. Mayr, H. Mostafa, G. Indiveri, and S. Spiga. Spike-driven threshold-based learning with memristive synapses and neuromorphic silicon neurons. *Journal of Physics D: Applied Physics*, 51:344003, 2018.

D. Deutsch and C. Marletto. Constructor theory of information. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2174):20140540, 2015.

D. Durstewitz, J. K. Seamans, and T. J. Sejnowski. Neurocomputational models of working memory. *Nature Neuroscience*, 3:1184–91, 2000.

F. S. Fard, P. Hollensen, D. Heinke, and T. P. Trappenberg. Modeling human target reaching with an adaptive observer implemented with dynamic neural fields. *Neural Networks*, 72(12): 13–30, 2015.

S. Fusi and X.-J. Wang. Short-term, long-term, and working memory. In M. Arbib and J. Bonaiuto, editors, *From Neuron to Cognition via Computational Neuroscience*, pages 319–344. MIT Press, 2016.

S. Ganguli, D. Huh, and H. Sompolinsky. Memory traces in dynamical systems. *PNAS*, 105 (48):18970 – 18975, 2008.

James Garson. Modal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2014 edition, 2014.

H. Geuvers, A. Koprowski, D. Synek, and E. van der Weegen. Automated machine-checked hybrid system safety proofs. In *Proc. Int. Conf. on Interactive Theorem Proving*, pages 259–274. Springer, 2010.

L. Gonon, L. Grigoryeva, and J.-P. Ortega. Memory and forecasting capacities of nonlinear recurrent networks. *Physica D*, 414(December 15):132721, 2020.

C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969.

A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines. Technical report, 2014. arXiv report http://arxiv.org/pdf/1410.5401v1.pdf.

A. Graves et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 7626:471–476, 2016.

S. Grossberg. Adaptive pattern classification and universal recoding: 1. parallel development and coding of neural feature detectors. *Biol. Cybernetics*, 23:121–134, 1976a.

S. Grossberg. Adaptive pattern classification and universal recoding: 2. feedback, expectation, olfaction, illusions. *Biol. Cybernetics*, 23:187–202, 1976b.

X. He, T. Liu, F. Hadaeghi, and H. Jaeger. Reservoir transfer on analog neuromorphic hardware. In *Proc. 9th Int. IEEE/EMBS Conf. on Neural Engineering*, pages 1234–1238, 2019.

D. O. Hebb. *The Organization of Behavior*. New York: Wiley & Sons, 1949.

M. Hermans and B. Schrauwen. Memory in linear recurrent neural networks in continuous time. *Neural Networks*, 23(3):341–355, 2010.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, 79:2554–2558, 1982.

C. Horsman, S. Stepney, R. C. Wagner, and V. Kendon. When does a physical system compute? *Proc. of the Royal Society A*, 470:20140182, 2014.

Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(1):67–72, 1975.

H. Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398, 2000.

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. URL http://https://www.ai.rug.nl/minds/pubs.

H. Jaeger. Short term memory in echo state networks. GMD-Report 152, GMD - German National Research Institute for Computer Science, 2002. URL https://www.ai.rug.nl/minds/uploads/STMEchoStatesTechRep.pdf.

H. Jaeger. Echo state network. In *Scholarpedia*, volume 2, page 2330. 2007. URL http://www.scholarpedia.org/article/Echo_State_Network.

H. Jaeger. Reservoir self-control for achieving invariance against slow input distortions. Technical Report 23, Jacobs University Bremen, 2010. https://www.ai.rug.nl/minds/uploads/ReservoirSelfControl_Techrep.pdf.

H. Jaeger. Long short-term memory in echo state networks: Details of a simulation study. Technical Report 27, Jacobs University Bremen, 2012. URL https://www.ai.rug.nl/minds/uploads/2478_Jaeger12.pdf.

H. Jaeger. Controlling recurrent neural networks by conceptors. Technical Report 31, Jacobs University Bremen, 2014. arXiv:1403.3369.

H. Jaeger. Using conceptors to manage neural long-term memories for temporal patterns. *JMRL*, 18:1–43, 2017. URL http://jmlr.org/papers/v18/15-449.html.

H. Jaeger. Toward a generalized theory comprising digital, neuromorphic, and unconventional computing. *Neuromorphic Computing and Engineering*, 1(1):012002, 2021. https://iopscience.iop.org/article/10.1088/2634-4386/abf151.

H. Jaeger and D. Eck. Can't get you out of my head: A connectionist model of cyclic rehearsal. In I. Wachsmuth and G. Knoblich, editors, *Modeling Communication for Robots and Virtual Humans*, volume 4930 of *Lecture Notes in Artificial Intelligence*, pages 310–335. Springer Verlag, 2008.

M. Jung, J. Hwang, and J. Tani. Self-organization of spatio-temporal hierarchy via learning of dynamic visual image patterns on action sequences. *PLOS-One*, (July 6), 2015. DOI:10.1371/journal.pone.013121.

C. Kuehn. *Multiple Time Scale Dynamics*. Springer Verlag, 2015.

R. Laje and D. V. Buonomano. Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nature Neuroscience*, 16(7):925–933, 2013.

A. Lazar, G. Pipa, and J. Triesch. SORN: a self-organizing recurrent neural network. *Frontiers in Computational Neuroscience*, 3:article 23, 2009.

R. Legenstein and W. Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007.

J. Lins and G. Schöner. Neural fields. In S. Coombes, P. beim Graben, R. Potthast, and J. Wright, editors, *Neural fields: theory and applications*, pages 319–339. Springer Verlag, 2014.

M. L. Littman, R. S. Sutton, and S. P. Singh. Predictive representations of state. In *(Proc. NIPS 2001)*, number 14 in Advances in Neural Information Processing Systems. MIT Press, 2002.

S. Lloyd. The universe as quantum computer. In H. Zenil, editor, *A Computable Universe: Understanding and exploring Nature as computation*, pages 567–581. World Scientific, 2013. https://arxiv.org/abs/1312.4455v1.

M. Lukosevicius. *Reservoir Computing and Self-Organized Neural Hierarchies*. Phd thesis, School of Engineering and Science, Jacobs University Bremen, 2012.

M. Lukoševičius, D. Popovici, H. Jaeger, and U. Siewert. Time warping invariant echo state networks. Technical Report 2, International University Bremen, 2006. URL https://www.ai.rug.nl/minds/uploads/techreport2.pdf.

W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11): 2531–2560, 2002. http://www.cis.tugraz.at/igi/maass/psfiles/LSM-v106.pdf.

G. Manjunath and H. Jaeger. The dynamics of random difference equations is remodeled by closed relations. *SIAM Journal on Mathematical Analysis*, 46(1):459–483, 2014.

G. Manjunath, P. Tino, and H. Jaeger. Theory of input driven dynamical systems. In *Proc. ESANN 2012*, pages ES2012–6, 2012. URL https://www.elen.ucl.ac.be/esann/proceedings/papers.php?ann=2012.

J. Martens and I. Sutskever. Learning recurrent neural networks with Hessian-free optimization. In *28th Int. Conf. on Machine Learning*, 2011. URL www.icml-2011.org/papers/532_icmlpaper.pdf.

M.D. Mauk and D. V. Buonomano. The neural basis of temporal processing. *Annu. Rev. Neurosci.*, 27:307–340, 2004.

N. M. Mayer and M. Browne. Echo state networks and self-prediction. In *Biologically Inspired Approaches to Advanced Information Technology*, volume 3141 of *LNCS*, pages 40–48. Springer Verlag Berlin / Heidelberg, 2004.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. of Mathematical Biophysics*, 5:115–133, 1943.

C. Metzner and P Krauss. Dynamical phases and resonance phenomena in information-processing recurrent neural networks. Technical report, 2021. arxiv manuscript https://arxiv.org/abs/2108.02545.

M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

R. Nishimoto and J. Tani. Development of hierarchical structures for actions and motor imagery: a constructivist view from synthetic neuro-robotics study. *Psychological Research*, 73:545–558, 2009.

M. Noltsis, N. Rodopoulos, D.and Zompakis, F. Catthoor, and D. Soudris. Runtime slack creation for processor performance variability using system scenarios. *ACM Transactions on Design Automation of Electronic Systems*, 23(2):1–23, 2017.

G. Palm. On associative memory. *Biol. Cybernetics*, 36(1):19–31, 1980.

R. Pascanu and H. Jaeger. A neurodynamical model for working memory. *Neural Networks*, 24(2):199–207, 2011. DOI: 10.1016/j.neunet.2010.10.003.

M. Payvand, F. Moro, K. Nomura, T. Dalgaty, E. Vianello, Y. Nishi, and G. Indiveri. MEMSORN: Self-organization of an inhomogeneous memristive hardware for sequence learning. submitted, 2021.

L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, pages 267–296. Morgan Kaufmann, San Mateo, 1990. Reprinted from Proceedings of the IEEE 77 (2), 257-286 (1989).

M. I. Rabinovich, R. Huerta, P. Varona, and V. S. Afraimovich. Transient cognitive dynamics, metastability, and decision making. *PLOS Computational Biology*, 4(5):e1000072, 2008.

R. F. Reinhart and J. J. Steil. A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, 19(1–2):27–46, 2011. DOI 10.1007/s12591-010-0067-x is an 2010 online pre-publication.

L. Ribar and R. Sepulchre. Neuromorphic control: Designing multiscale mixed-feedback systems. https://arxiv.org/abs/2011.04441, 2021. arXiv preprint.

D. Rodopoulos, S. Corbetta, G. Massari, S. Libutti, F. Catthoor, Y. Sazeides, C. Nicopoulos, A. Portero, E. Cappe, R. Vavrik, V. Vondrak, D. Soudris, F. Sassi, A. Fritsch, and W. Fornaciari. HARPA: Solutions for dependable performance under physically induced performance

variability. In *Proc. 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 270–277. IEEE, 2015.

K. C. Saraswat. Scaling of interconnections, 2006. Handout 6 of Course Advanced Integrated Circuit Fabrication Processes. Accessed: October 2021 from https://web.stanford.edu/class/ee311/.

G. Schöner. The dynamics of neural populations capture the laws of the mind. *Topics in Cognitive Science*, pages 1–15, 2019.

t. Schreiber. Measuring information transfer. *Physical Review Letters*, 85(2):461–464, 2000.

M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4 (2-3):245–270, 1961.

R. Sepulchre, G. Drion, and A. Franci. Control across scales by positive and negative feedback. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:89–113, 2019.

C. R. Shalizi and J. P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *J. Statistical Mechanics*, 104(314):817–879, 2001.

S. Stepney, S. Rasmussen, and M. Amos, editors. *Computational Matter*. Springer Verlag, 2018.

Guo-Zheng Sun, Hsing-Hen Chen, and Yee-Chun Lee. Time warping invariant neural networks. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 180–187, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-274-7.

D. Sussillo and L. Abbott. Transferring learning from external to internal weights in echo-state networks with sparse connectivity. *PLoS ONE*, 7(5):e37372, 2012.

D. Sussillo and O Barak. Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation*, 25(3):626–649, 2013.

T. Tanaka, G. abnd Yamane, J.B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, 2019. preprint in https://arxiv.org/abs/1808.04962.

M. Thon and H. Jaeger. Links between multiplicity automata, observable operator models and predictive state representations – a unified learning framework. *Journal of Machine Learning Research*, 16:103–147, 2015.

P. Tino, B.G. Horne, and C.L. Giles. Finite state machines and recurrent neural networks – automata and dynamical systems approaches. In O. Omidvar and J. Dayhoff, editors, *Neural Networks and Pattern Recognition*, chapter 6, pages 171 – 219. Academic Press, 1998.

I. Tsuda. Towards an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *Behavioural and Brain Sciences*, 24(5):793–810, 2001.

A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc.*, 42(2):230–265, 1936.

T. van Gelder and R. Port, editors. *Mind as Motion: Explorations in the Dynamics of Cognition*. Bradford/MIT Press, 1995.

J. van Leeuwen and J. Wiedermann. Beyond the turing limit: Evolving interactive systems. In *International Conference on Current Trends in Theory and Practice of Computer Science*, number 2234 in LNCS, pages 90–109. Springer, 2001.

D. Verstraeten, B. Schrauwen, and D. Stroobandt. Reservoir computing with stochastic bitstream neurons. In *Proceedings of the 16th annual Prorisc workshop*, pages 454–459, 2005.

A. R. Voelker, I. Kajic, and C. Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Proc. 33rd Conf. on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

S. Watt, M. Kostylev, A. B. Ustinov, and B. A. Kalinikos. Implementing a magnonic time-delay reservoir computer model. Technical report, 2021. arXiv manuscript https://arxiv.org/abs/2104.07879.

J. A. Wheeler. Information, physics, quantum: the search for links. In S. Kobayashi (et al), editor, *Proc. III Int. Symp. on Foundations of Quantum Mechanics, Tokyo, Japan*, pages 354–368. Physical Society of Japan, 1989.

L. Wiskott. Learning invariance manifolds. *Neurocomputing*, pages 925–932, 26-27.

L. Wiskott and T. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

L. Wiskott, P. Berkes, M. Franzius, H. Sprekeler, and N. Wilbert. Slow feature analysis. *Scholarpedia*, 6(4):5282, 2011.

F. wyffels, J. Li, T. Waegeman, B. Schrauwen, and H. Jaeger. Frequency modulation of large oscillatory neural networks. *Biological Cybernetics*, 108:145–157, 2014.

J.-W. Xu, H. Bakardjian, A. Cichocki, and J.C. Principe. A new nonlinear similarity measure for multichannel signals. *Neural Networks*, 21:222–231, 2008.

Y. Yao and W.J. Freeman. A model of biological pattern recognition with spatially chaotic dynamics. *Neural Networks*, 3(2):153–170, 1990.

B. Yin, F. Corradi, and S. Bohté. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. biorxiv preprint, 2021. https://doi.org/10.1101/2021.03.22.436372.

A. Yousefzadeh, E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco. On practical issues for stochastic STDP hardware with 1-bit synaptic weights. *Frontiers in Neuroscience*, 12(October):Article Nr 665, 2018.

L.A. Zadeh. The concept of system, aggregate, and state in system theory. In L.A. Zadeh and E. Polak, editors, *System Theory*, volume 8 of *Inter-University Electronics Series*, pages 3–42. McGraw-Hill, New York, 1969.

H. Zenil. Introducing the computable universe. In H. Zenil, editor, *A Computable Universe: Understanding and exploring Nature as computation*, chapter 1, pages 1–20. World Scientific, 2013.

Y. Zhang, P. Qu, Y. Ji, W. Zhang, G. Gao, G. Wang, S. Song, G. Li, W. Chen, W. Zheng, F. Chen, J. Pei, R. Zhao, M. Zhao, and L. Shi. A system hierarchy for brain-inspired computing. *Nature*, 586(15 Oct):378–384, 2020.

K. Zuse. The computing universe. *International Journal of Theoretical Physics*, 21(6/7), 1982.

K. Zuse. Cellular structured space (rechnender raum) and physical phenomena. Technical Report TR 93-12, Konrad-Zuse Zentrum für Informationstechnik Berlin, 1993.

Consortium

CEA	Commissariat à l'energie atomique et aux énergies alternatives (**coordinator**)	France
IMEC	Interuniversitair Micro-Electronica Centrum	Belgium
IMEC-NL	Stichting IMEC Nederland	Netherlands
IBM	IBM Research Gmbh	Switzerland
UZH	University of Zurich	Switzerland
CSIS	Consejo superior de Investigaciones Científicas	Spain
CNR	Consiglio Nazionale delle Ricerche	Italy
aiCTX	aiCTX AG	Switzerland
UOG	Rijksuniversiteit Groningen	Netherlands

Disclaimer

All information provided reflects the status of the MeM-Scales project at the time of writing and may be subject to change.

Neither the MeM-Scales Consortium as a whole, nor any single party within the MeM-Scales Consortium warrant that the information contained in this document is capable of use, nor that the use of such information is free from risk. Neither the MeM-Scales Consortium as a whole, nor any single party within the MeM-Scales Consortium accepts any liability for loss or damage suffered by any person using the information.

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

Copyright Notice