ABSTRACT

Title of Dissertation: DYNAMICAL MEMORY IN DEEP NEURAL NETWORKS

Matthew S Evanusa, Doctor of Philosophy, 2024

Dissertation directed by: Dr. Yiannis Aloimonos, Department of Computer Science

I have attempted to build up from neurodynamical and physical principles a foundation for reverse engineering the human brain towards the next state of artificial intelligence. I argue that events in the universe are not isolated, random events, but are rather highly temporally correlated. These temporal correlations, temporal sequences, must be a singular focus for the evolution of the living brain. I argue that the brain does in fact focus heavily on encoding temporal sequences, via mechanisms that were discovered by Donald Hebb in the middle of the 20th century. These resonating cell assemblies form the core building blocks of all sequential memory in the brain. This temporal, sequential activity of resonating cell assemblies combines with a readout mechanism in output cortices to produce the functions that we observe in living creatures. I propose a new paradigm, *Maelstrom* networks, where the maelstrom is the set of these resonating cell assemblies, removed from the error learning component of the network. The error learning, instead, is performed by a control network, and a readout network, parameterized by deep neural networks.

DYNAMICAL MEMORY IN DEEP NEURAL NETWORKS

by

Matthew S Evanusa

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Yiannis Aloimonos, *Chair*, Department of Computer Science
Professor Cornelia Fermüller. *Co-chair*, Department of Computer Science
Professor Michelle Girvan, Department, of Physics
Professor James Reggia, Department of Computer Science
Professor Daniel A. Butts, Department of Biology
Dr. George Stantchev, US NRL
Professor Joseph JaJa, Department of ECE, *Dean's Representative*

DISTRIBUTION STATEMENT A. APPROVED FOR PUBLIC RELEASE. DISTRIBUTION IS UNLIMITED

Contents

Intent

In this work, I will begin to lay out a roadmap or framework for which I believe will serve the scientific communities of artificial intelligence and cognitive neuroscience of interest, in future development and design of a thinking intelligent machine, based on the accumulated knowledge I have gathered across many sources: from my advisors, peers and colleagues, collaborators, talks, symposia and conferences, and long paper dives, for the almost decade that I have spent at my new home in College Park, Maryland. It is my hope and intent that this thesis serves in its stated goal to advance the science of memory integration in neural networks, but in addition, to further the distant dream of discovering the mystery of what it means to be alive. It is important to note that while this thesis is focused on the critical integration of memory mechanisms into artificial neural networks, the authors' larger goal is the creation of an overarching cognitive architecture that takes advantages of the right amount of advances from deep learning, with the right amount of insights from cognitive and neuroscience - a "Goldilocks" of sorts for AI. It is my hope that through understanding mechanisms of memory and how they interact with our stimuli, we move one step closer to understanding our place in the cosmos.

As such, this paper will broach into a variety of topics that I think are critical for developing of this thinking machine. It will begin with a venture into the major theories that I have condensed to be the most important for me, as well as what is missing from the current ventures into the brain. It will then move into the more specific topic of *memory*, specifically *dynamical, continuous memory* which is the stated goal of the thesis, and methods in which we can combine dynamical memory with artificial neural networks, which so far has remained elusive, and may serve to finally give artificial agents an embodiment.

Acknowledgements

What a long and strange journey it has been! There are too many people to thank, and omission from this list does not mean I don't see you! But with that said, I would first like to thank my mother Michel, without whose support and love I would not have been able to finish. I would like to thank my late father Stephen who passed just before my admission to UMD, for giving me at a young age the interest in looking to the stars and thinking beyond what is currently understood - I know you would be very proud. To my sister Julia, who supports and loves me through everything - dad would be so proud of you too. My family away from home Carl and Adriana and Stephen, who I lived with my first few weeks here in Maryland, and who have been a bedrock of support far away from New York. Steven at CUNY who gave me a chance and allowed me to propel up, and Jim who took a chance on me for UMD. My two advisors, Yiannis and Cornelia, for being so supportive of my wild ideas from the very beginning, and for offering the continuous philosophical and scientific guidance that lit my path ahead. You always encouraged me to keep going, and introduced me to so many new ideas, and for that I am deeply grateful. I would like to thank Greg and Felix for introducing me to the neuroscience angle and for our continued discussions to this day, discussions which influenced many parts of this work. I would like to thank Susan Epstein for introducing me to the art of science and AI, and Anoop for being my first lab partner. To Ron and Michael, who caught me when I fell and reminded me who I am. To Michelle for introducing the wonders of dynamical systems and reservoirs to my way of thinking, and to Dan for introducing me to neuroscience proper, which I continue to learn more of each day. To Carlos, for helping me understand what it means to be a free, independent thinker and helping mold me into the person I am today, as well as for grounding me in the former arts of cybernetics and giving me invaluable feedback and ideas towards memory. To Natasha, who kept the love of music in me alive, and who I will keep in my heart forever. I would like to thank all of "Demon's Bane's Bane" and Felix for putting in all the hard work in world building, for the constant DnD fun that helped keep me sane during covid. I would like to thank my co-conductor Sam and everyone at the UMD Gamer Symphony Orchestra for re-injecting music back into my life these many years and whose spirit and energy propelled my research forward, and to everyone in the GSO "Retirement Home" for being the amazing people you are. To Bryan, Joel, Marilyn, George, and everyone at NRL who has supported me, and who helped inspired me with the real-world needs for neural networks, and allowed me to continue my research and come up with these ideas I present in this work. To everyone at Telluride and PLENA who filled me with inspiration and energy about reverse engineering the brain, your inputs have been invaluable. I would like to thank everyone at UMD and in the PRG lab for their constant support and feedback and advice. And of course, I would like to thank Vaishnavi for being there with me every step of the way after my proposal, a singularly wonderful person, collaborator who became family, who encouraged my ideas and discussed with me over these many years; you were my bedrock here at UMD and without you this would not have been possibl

1 Reverse Engineering the Brain

A perceptron is first and foremost a brain model, not an invention for pattern recognition. As a brain model, its utility is in enabling us to determine the physical conditions for the emergence of various psychological properties.

Frank Rosenblatt
On the Perceptron, Principles of Neurodynamics, 1962

The Power of Brain-Derived Principles

Since the Dartmouth summit of 1955, researchers have strived to come up with a cohesive, overarching theory that can be boiled down into one, or a set, of equations, which could accurately describe the human brain's functionality as a *single, cohesive unit*. There do exist larger theories boiling into single equations that at a more philosophical level describe the goals of a thinking machine, which I will detail in the following section; these do not specify the implementational details of how to train such a machine, but rather give a larger goal that the brain strives for, which may be implemented in multiple sub-modules.

Proponents of the effort to boil down brain functionality to simple rules range from early work on Hebbian neural networks [Hebb, 2005], to the focus on classification tasks with single neural networks, theories of spike coding by Izhikevich alert that include Polychronous groups, and self-organizing maps using Hebbian principles [Kohonen, 2006] (SOMs). It is notable to point out that the work of Hebbian learning and SOMs is *associative* in nature and *unsupervised*: the data is learned not according to a *task* (which requires error learning) but rather organizes the data according to itself for no particular end goal. This will end up being a critical difference that I will return to later. In parallel, while working on a brain model for *pattern recognition*, Frank Rosenblatt set off the *error learning* revolution with the introduction of the *Perceptron*. The result of these networks were implicitly focusing on the stimulus-response aspect of brain processing: the functionality that takes input from the world, and produces some meaningful *response* to that input, with respect to some *goal*. Whether you have a goal or not turns out to be a critical difference.

Regardless of if you took the goal-oriented or map-oriented path, reverse-engineering the brain, *at the appropriately chosen level of magnification*, offers the most direct path to deriving intelligence in artificial agents. All of the major advancements that we see today in A.I. systems resulted from an initially biological focused origin. Of course, they matured and were later analyzed in more depth using mathematical tools at our disposal, but the initial inspiration for the models were of biological origin. This statement sounds noncontroversial, however, there are equally valid arguments to be made with respect to engineering an AI system from the ground up using the end-goal of the brain's function as a guide, rather than the mechanisms the brain might be using; we could do this using physical principles such as control theory, dynamical systems, and others. However, I will argue that some of these were either actually biologically inspired (as in the case of control), or are valid sub-tools that need to be stitched together to form a working brain, as in the case of my proposed *Maelstrom Paradigm* that forms the end

result of this thesis. However, we know from neuroscience and genetics that the brain did not just *suddenly appear*, as in the case of a watchmaker suddenly making a watch from pieces of material. The brain evolved slowly and smoothly through a continuum of evolutionary steps, from earlier creatures to our present humans, slowly picking up components and discarding others, and slowly re-using old molecules for new purposes (as in the case of serotonin, which is found ubiquitously throughout plants and animals, and has a myriad of uses) [Weiger, 1997]. The task of reverse engineering the brain *should*, on paper, be the same as that of neuroscience; once we discover how the brain works, we should in theory be able to replicate it in an artificial agent. The biggest question is, what is the major roadblock for us doing so, and why have the more mathematical networks taken off? I would say it is because unlike in the case of the *Perceptron*, we have missed the "sweet spot" of reverse-engineering.

This brings up, what I would argue, is the **most important question in artificial intelligence** (and something ng that I have heard from neuroscientists as well): *what degree of abstraction in modeling the brain is the most critical, when creating intelligent agents?* What is the "zoom" level that we need to fully capture brain function? **It needs to be in the "goldilocks" zone**: not too low level, as there we may be replicating functions that are unnecessary, and not too high level, where we may be missing critical functional rules that are the core of intelligence.

Learning Rules and the Case for the *Middle-Down* Approach

With this hierarchy, we can ask, is it better to go top-down, or bottom up? Each one has its own advantages and disadvantages. Bottom-up ensures that any resulting systems theory will be biologically plausible; this is the viewpoint of the Blue Brain Project [Markram, 2006] and similar endeavors, where we first simulate hyper-realistic neurons and look at the resulting effects. This was my view when I was in my early years, and I coded up my first deep STDP spiking network. The danger with this approach, however, is it somewhat makes us search blindly for possible systems architectures, which may be a large number of them resulting from the same biological mechanisms. Secondly, we may not be correctly listing the parameters for the neurons that we model. Even though these massive networks contain the "best we have" in terms of neurophysiology, there is a high chance that we missed a critically important parameter that may greatly affect the resulting system. Anyone that has worked with dynamical systems knows that the attractor space for even a low-dimensional system is highly complex, and for different parameters, you get wildly different spaces (imagine doing this for 80+ variables!). While I admire and appreciate all of this work, as it was something I was very interested in, I somewhat deviated from this view, as these downsides for scientific discovery became apparent. On the flip side, we cannot simply go completely top down; this I leads I believe to the AI of the 1970's, which were fueled primarily by logic and tree reasoning. This started "too zoomed out" and led us to develop systems that were theoretically what an agent should be doing, but were not able to successfully interact with the environment due to complexity and inability to learn features of the environment. As such, here I argue for what I am calling the *Middle-Down* approach, or as I mentioned the "goldilocks zone"; **we start with biologically reasonable ideas about the structure of the systems of the brain, and recreate them using principles that could be swapped at a later time for more biologically realistic mechanisms**. As I just mentioned, there is an ongoing debate as to the bio-plausibility of backpropagation [Stork, 1989], the workhorse that has enabled the deep neural network revolution. At one of the lower levels of the previously-described zoom levels, we have the debate to which neurons, in a network, learn. It is my view that the "magic", so to speak, in the neural network both artificial and biological, rests on the *network* of neurons, the *graph* of their interactions, not the individual brilliance of each individual neuron. The same view was espoused in [Sporns, 2014]. In fact, each individual neuron is rather stupid, it is a simple machine that takes in input and spits an output if that input passes a threshold. [1] Here, I will make the case that until we have a better systems level understanding, we should not be focusing on proving the exact biological mechanisms for neuron learning. These mechanisms are

[1] This was something emphasized by Professor Marja-Leena Linne at the PLENA summit; that potentially Glia are the smart ones, not the neurons, which might be seen as tools of the Glia. This is a relatively explosively disruptive view and not one I will endorse but, it is interesting to think about.

critical for any understanding of the brain, but must be derived after the top-down understanding is understood, rather than building up. The danger of building up is that the search space for solutions is much more difficult when we do it this way; we can simulate a spiking neural network, but the resulting systems architecture that we desire is only one possible outcome of this. If we begin our search a bit higher up the hierarchy, we stand a much better chance of narrowing the potential solutions at the lower levels of the hierarchy.

A beautiful example of a successful Middle-Down approach was the *Perceptron* by Frank Rosenblatt. This model was actually meant to serve as a brain model first, but abstracted away the specific learning rules (rules which were *known* to Rosenblatt at the time!) to focus on the core computational principles of integration of information within a network of neurons. It was this model that inspired me to follow the Middle-Down approach, and I will relate much of my thinking to Rosenblatt's original trailblazing thinking. Those are aware of the importance of the *Perceptron* will immediately realize how significant this is, given the fact that *Perceptrons* are the core building block of all modern artificial neural networks.

An Idea on the Right "Zoom" Level

With that being said, I will attempt to list some ingredients that may comprise a way to look at the "correct" zoom level to approach the brain. It is important to note that the brain faced many phylogenetically critical, yet computationally non-important, evolutionary traits, which make for unique solutions to problems, but also make some solutions less than optimal, or truly "necessary". The brain's neurons evolved to be good at one particular task: accumulating (integrating) voltage from its synaptic terminals, and using that integration to guide a firing to propagate this signal further, or to cease the firing if the integration does not match a learned pattern.

These debates about the structure, and organization, of the brain, sit at varying levels of specificity along a hierarchy of the organization of the brain. We can decide to focus on several different zoom levels of abstraction in the brain: Figure 1.1 depicts a visual schema of these levels of the hierarchy. Where we are aiming to go depends on the task: for cognitive neuroscience and psychology, we may be interested in stopping at an internal node such as the functional groups, if we are interested in simply discovering the mechanisms for how a certain area works. Generally, however, we are interested in finishing in the final box, Behavior, as we want to tie together the entire picture of how our mind integrates new information and produces behaviors. The diagram is not meant to be exhaustive, and is merely one possible organization of the layers (others may choose to subdivide the boxes into further divisions), but I believe few would disagree with the given chosen organization. For example, a similar diagram (which I discovered after I made the below diagram) can be found in, where the author notes, "Time will eventually tell the most favorable mix of physical realism and digital abstraction in the cognitive race along the task-complexity axis". [Cauwenberghs, 2013].

The top layer describes the lowest possible level: the molecular interactions. In this layer, we specify the precise *molecules* that comprise the neuron: we model the calcium, sodium, and potassium channels that control the Unfortunately, as I have discovered in my academic journey, there is a severe disconnect between these levels of abstraction, and a lack of consensus, or even a drive to, connect these independent layers. Even today at the gatherings of top computational neuroscientists, the question of "what can we glean from reverse engineering the brain" is still an open and hotly debated question.

Phylogenetic Origins of Intelligence

To understand how the brain works, it is critical we also look phylogenetically and historically at the brain's, as well as the neuron's, development from the origins of evolution. What hobbles this effort is we still do not fully understand why life exists in the first place - known as the question of *Abiogenesis*, or the question of how and why life arose from matter - and for what purpose does it continue to reproduce. If we had in our hands an answer for this question, we could easily devise a *loss function* in an optimization framework to optimize this quantity, that we perceive life to be optimizing. It has become clearer however that life is not simply a giant optimization loop,

to the dismay of researchers who want an easy task to optimize over. In contrast to the optimization problem, we can view the system as simply maintaining a homeostasis; this leads to other theories, such as those of *autopoiesis*, which attempts to argue that life is rather a homeostasis which self-organizes to maintain itself, but is not trying to optimize a specific quantity. Newer work such as the Energy Homeostasis Principle [Vergara et al., 2019] push this further and try to see how an energy homeostasis system would self-organize into useful structures.

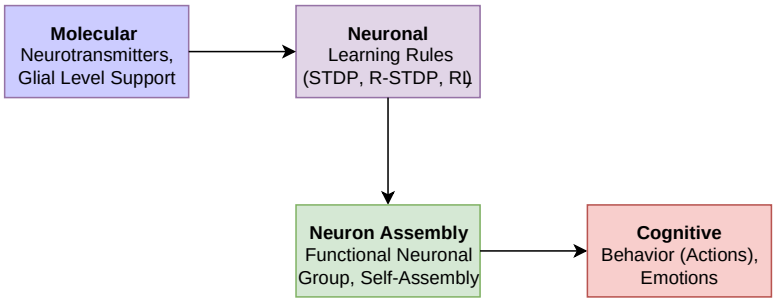


Figure 1.1: **The Reverse-Engineering Hierarchy** Efforts in Cognitive Modeling and reverse-engineering the brain have been hamstrung by the fact that there are multiple unique levels of abstraction that we can consider before we begin. The categories of the layers are listed in bold, and some example constituent research areas are given following. I give some examples in each category that are not meant to be exhaustive, but are rather a representative instantiation. There has not been a cohesive ”gluing” that brings together these various levels. If one desires, one can model the brain at increasingly lower and lower levels of sub-granularity. Where do we decide to ”start” the process? Importantly, one can *start* the ”modeling” process at any of these given boxes, and proceed forward.

Creatures Persist in the World, and Take Actions

What can we glean from looking at this hierarchy? Certainly, there are multiple possible ways to perform the molecular interactions (why are we carbon based?) There are also multiple ways to achieve the cognitive effects (look at the literature on symbolic A.I. during the “A.I. winter!”). What is it then, that is core to the experience of an artificial agent? From looking at the evolution of life, we can see that there is really only one purpose (once life evolved) in a creature: to survive, and to reproduce. And while the agent is surviving, it exists as a sense of itself in the world, persisting across time. Thus, there is a singular core computational function amongst creatures, which is to maintain a memory of itself, learn from its mistakes, and persist this memory of itself through time. This memory, according to Donald Hebb, sits at the level of cell assemblies; these assemblies are the core building block of all processing in the brain (neurons are, by themselves, relatively stupid after all). For this reason, I will focus primarily on the connectionist web of neurons at the assembly level as the core “reverse engineering” saddle point, and surmise on their role towards a persistent state of the artificial creature.

There is still a reasonable debate as to the level of *specificity* required to model the brain. This is arguably the most important question in theoretical neuroscience. And this is also the most critical question for those coming from the other direction, trying to understand what mechanisms are necessary to reproduce the brain. I will offer my account on what connectionist systems have offered us towards developing artificial systems, and where they have come up short. It is this notion of stateful memory that I will focus on for the remainder of the thesis as a key missing component of current artificial intelligence systems. In the next chapter, I will expand a bit on this idea of *memory*, in particular memory that relates to time and the cause-effect relation of the universe. I will delve into what memory means and what it doesn’t mean, and look into how memory is can be implemented in physical hardware. This notion of stateful, sequential memory is also our best shot, in my view, towards the next step, which is a functioning executive system that is able to perceive itself through time.

Sequence Memory, Physics, Graphs, and the Universe

2

> Yet, if I cease to recall them even for short intervals of time, they are again so submerged–and slide back, as it were, into the further reaches of the memory–that they must be drawn out again as if new from the same place
>
> *St. Augustine*
> *Confessions, X.XI.18*

Few things in our world are as important as memory. It may even be argued that memory is the single most important thing that we have - it is what sustains us, what reminds us who we are, is a key element of our personality, and is a lasting impression on the world that our brief time here affords. Our current self is irrevocably impacted by our experiences and the events that shaped us - for better or worse. Memory crosses multiple scales - not only for our own lives, but for the course of humanity. We build statues, write great works of literature and paint great art, and tell great stories by word of mouth, and through culture, to remember the past and preserve the experiences that we and our ancestors and forebearers experienced. In some sense, the memory of being human is quintessential to being human. And to not remember - forgetting our past - is considered the greatest of tragedies. Clearly, a memory of the past, and how to incorporate it into our lives, is critical to being human. Our culture and brains sustains themselves through this memory, but how does memory propagate itself through time? And how can we define this propagation? This is the question I will attempt to answer in this section.

The two major questions we have to ask are between the *how*, and the *why* of the brain's approach to memory. There is a long and storied history of research into memory from the neuroscientific as well as psychological angle [Squire, 2004]. It is clear that that study of memory is also intertwined with the study of epistemology, or the study of knowledge or what it means to know. We can ask, why do we know the things we know, and what gives the things we know meaning? Here I take the viewpoint that something needs to "ground" our knowledge in the real world; that we are not simply abstract proof engines that process arbitrary symbols. This I believe has a stronger link to evolutionary biology.

"Encoding" as Mutual Information

At the core, without thinking about agents or conscious creatures, we think of memory as representing something from the past. This implies that at least part of memory involves an encoding mechanism, a decoding mechanism, and two entities: the encode and the encoder. If we have Plato's example of a wax tablet, the encoder would be the wax, and the encodee would be the tablet that is being pressed into it. To give a more mathematical intuition, we have to objects A and B; we can consider A the encodee and B the encoder. We have the relationship between them such that the encoder contains some mutual information (MI) of the encodee, which we can say as $MI(A,B) > 0$; that is, there is some essence or imprint within this encoding entity B that gives a greater than random chance of

decoding the correct information about entity *A*. I will refer to this as a "recording" [2], because this definition is similar to a camera recording a scene; there is nothing grounding the recording to any task or *reason* for its recording, but simply that it exists. An example of a naturally occurring "recording" in the universe is shown in Fig. 2.5. What makes this different from a *memory*? The difference is that memories are grounded, grounded in our existing in the real world and surviving.

Memory vs. Recording

Following the ideas from [Glenberg, 1997] and others, we say that memory serves in the preparation of action and embodiment, and this is what gives it meaning. It becomes then an open question on how we can say that certain episodic memories are grounded in action, such as remembering going skiing with your family, but this is not an insurmountable objection; we can take the Piagetian approach and say that hierarchically this does, in a convoluted intertwining, wind its way back to action [Carlson, 1997]. And in some ways, this is the job of many psychologists and psychiatrists, to find the ways that our more abstract behaviors originate in more grounded beginnings. To this end, we will say the *why* of memory is *embodiment*, which I will touch on in Sec. 6. We could also define memory in terms of information theory, and say that memory is when we have information of the past encoded in some method, that can be decoded later. To shift into the *how*, we must ask, how is the brain performing these actions? From theoretical neuroscience, we know that the brain encodes its information as not only rates but as *temporal* codes [Theunissen and Miller, 1995]. Here, I combine the view that neurons encode temporal "traces" of memory, with a graph representation of those neurons, and argue for a definition of memory as a traversal across a specific kind of graph which relates to ideas from Markov Decision Processes and Discrete Finite Automata.

Sequence of Events and Time

What can we say about the structure of the information in the physical world? Is it sort of a jumbled mess of data points, or is there some inherent structure to it? We know that there is one key structure that underlies all of the matter in the universe: time. The "arrow of time" flows in one direction, and the matter flows along this time like waves in the ocean. Waves along the arrow of time are a direct causal result of the waves that came before them. Thus, the information in the universe is not a "jumbled mess", but rather organized quite neatly into what I will call *temporal sequences*. This is to distinguish this unique definition of sequence from the more general mathematical concept of a sequence. From this definition, we can see that having general-statistical memory is not enough, both to develop artificial agents as well as to describe the nature of the brain. A true memory theory needs to take into account these temporally locked sequences, which are causally linked in the arrow of time. Figure 2.1 displays visually this dichotomy. Importantly, the view of data as untemporally correlated is the view of "windowed deep learning", as I will describe in subsequent sections; it is the view that we can sample randomly from a set of the world and make a reasonable prediction without looking at the temporal structure.

Action as the Sequential Reference

An astute observer might comment, all elements in the universe are interconnected in the same space-time continuum, and thus it makes no sense to cluster them according to unique temporal sequences. What does it mean for elements along the sequence, as the black line in Figure 2.1, to be contained to that sequence? The answer to this question is similar in nature to the idea of *reference points* of quantum mechanics. These temporal sequences are temporally-causally related with respect to a specific reference point. This reference point, I will argue, is the same as my views and the view of [Aloimonos et al., 1988, Glenberg and Kaschak, 2002], that our sense of self is

[2] From conversations with Carlos Sluzki.

grounded in *action*. Here, the action of the apple falling from the tree is what temporally links the sequence of the apple falling; the action of traveling to a foreign country links those sequences.[3]

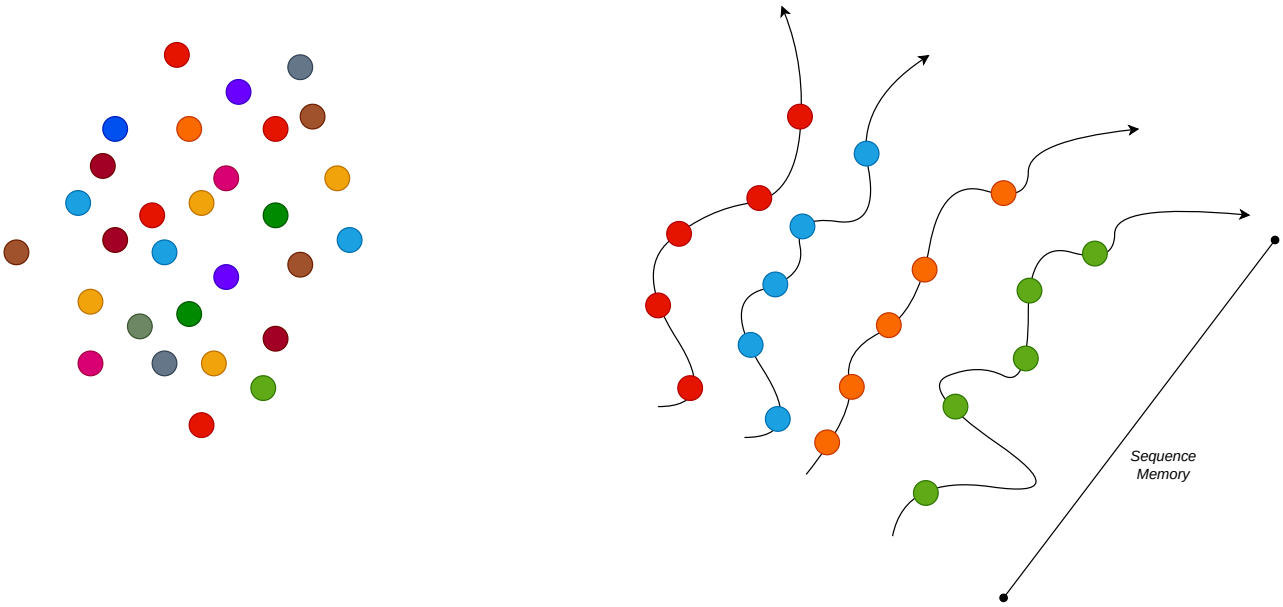


Figure 2.1: A visual description of the nature of temporal sequences. *Left:* The world if it were a jumbled set of uncorrelated data. This is the world of sequence-memory-less deep learning, where we only care about remembering data points from the population as a whole, and their correlations. This world assumes no underlying structure for the data in time. *Right:* The view of temporal sequences, where the data points are not seen as randomly uncorrelated data, but have a temporal-causal relationship along the stream of time, in line with the physics of the universe. The black line here represents points that are strongly temporally-causally related, for example, the apple's trajectory as it falls from the tree, or the joy one experiences when traveling to a new country. This relationship is with respect to a certain reference point, which is philosophically similar to the reference-oriented views of quantum mechanics. The reference point that I will argue is with respect to action; as action creates a temporal cause and effect relationship. The ability of a system to remember a point (of the same color in this visualization) at different points along the same temporal sequence is what I refer to as *Sequence Memory*.

Sequence Memory Gives Us Consciousness

This is a rather contentious subject to broach, but I would like to firmly stake a claim here that the idea of consciousness may arise naturally from this notion that the brain encodes sets of temporal sequences. If the brain is really running as sets of sequences, which through predictive coding are constantly being re-experienced, then this re-experiencing of predictions of sequences sounds like a reasonable description of the conscious state. I will attempt to broach the brain's view a bit later, but from a philosophical angle, this is a good starting point.

Memory as Temporal Trace in Neural Graph

Because the brain evolved in the natural world, it thus must have evolved a mechanism to deal with the highly temporally structured nature of temporal sequences. One way to encode a temporal sequence is via a *graph*: the

[3] I do not believe it is a coincidence that the *constant of action* comes into play in physics and quantum mechanics; although these are different useages of action, I believe there is something deeper here. Action in those cases is a measure of energy on a trajectory, and here we discuss temporal sequences as trajectories.

nodes of the graph can encode the various information of the world, and the relationships between the nodes can encode the temporal ordering. We know the brain, via epigenetic mechanisms responsible for neuron connection growth at development, is a much more random process than we would like to believe [Edelman, 1987]. While there is structure that is inductively coded into the genes, a theory for memory traces in the brain must account for the randomness that inevitably occurs in the brain, and cannot overfit to specific synaptic-level circuit types in more homogenous regions. While in some regions like the cerebellum, these generic circuits are certainly stereotyped, this is at a more zoomed-out level, and not at the level of designing specific circuits for specific memories. A memory theory must be robust to the randomness, possibly even take advantage of it as in [Edelman, 1987]. This idea that randomness allows for a more rich set of features is also present in reservoir computing [Jaeger, 2001], and it is this inspiration that I move forward with. First, I will describe the mathematical structure for the graphs, and move forward with examples of how this memory is then encoded in these graphs.

Sequential Memory as a Canal

Another way to link the dynamics of the temporal graph with those of sequential memory, is to think of the evolution of the activity of the network flowing as if it were water flowing through a canal (Fig. 2.2). At each timestep, the network enters a new state, which is akin to the water entering a new area of the canal. The canal is segmented according to the structure of the pipes, likewise, the network is segmented according to the structure of the network. The activity cannot suddenly jump to a different part of the network in the same way that the water cannot simply jump to a different part of the canal. The water is causally linked according to the flow of the water, each action of the water's movement causing subsequent action on the flow; likewise, the causal relationships on the graph's activity define the flow of the activity across time.

MDP Definition

A *Markov Decision Process* (MDP) is a mathematical graphical construct that lets use describe a system and its transitions in time in terms of a set of nodes and edges of a graph. It contains: A set of world states S, A set of Actions A that are possible to take at each state by the agent, $P_a(s,s')$, the probability of transitioning from state s to state s' for any two states s and s', and $R_a(s,s')$ given action a, the reward the agent gets from transitioning from s to s'. At each state, the a set of actions A lists the actions an agent can take, given the state, to transition to the next state.

Discrete Finite Automata in Action Space

A *Discrete Finite Automata* or DFA, is the mathematical framework from computer science and applied mathematics that will endow our graph a temporal processing capability, as it is commonly agreed that since the brain is living in the world, it must somehow process input stimuli, read them, and perform some action (although, this is not potentially it's only purpose). The main reason to dip into this definition is to give a mathematical and graphical understanding for the necessity of loops in the graph; if we consider node activity as "actions" in the graph, there must be some consequence in time for the absence of rerouting the activity back on itself; the activity simply dissipates. A DFA consists of a graph, like an MDP, contains a set of states $s \in S$, and it takes actions at states, and reads in the sequence of actions $A = (A_1,A_2,A_3..A_N)$ and processes them one at a time, before arriving at a terminal state. While generally defined in terms of languages, here I do not define them the same, but notably in terms of *actions*, that are taken, at each step, which brings us to a next state of the automata. Note that redefining the language change brings DFAs close to our definition of the MDP framework; the main addition that the DFA gives us is the notion of a *terminal state*, as well as loop transitions. In the DFA, if we want to maintain information across time, or if we want to stay in the same state (i.e., remain in the DFA), we must include a looped transition from a state back to itself, or form a cycle of states. This notion of requiring a cycle or loop to remain in the DFA is critical and will come back in Section 2 for the encoding of memory.

State

What is a state? The idea of a state is a convenient mathematical construct that lets us discretize time into arbitrary steps, and lets us reason about dynamical processes (for example, the neurons in the brain) over a temporally quantized graph. By state, what really is meant in the MDP literature is *the configuration and minimum variables required to describe and code the world and its components*. I borrow here the definition from Thermodynamics and quantum mechanics [Aharonov and Vaidman, 1991] because I aim here to bridge the distance between physical laws of the universe and the brain. This is also the view of a "state" when we discuss Markov Decision Processes and State later on. There, we can describe a system in terms of both a *state*, which gives us static information, as well as *state transition*, which gives us dynamic. At the most primitive level, we can think of a state as the vector of the principle particles in the Universe. However, this assumes we want to encode every detail about the universe; this is likely not the case, and for a living organism, we likely only will be encoding the world based on the information from the sensory organs. This state is also dependent on the level of "zoom" that we wish to focus on. For the state of a cylinder of gas, we likely consider the state to

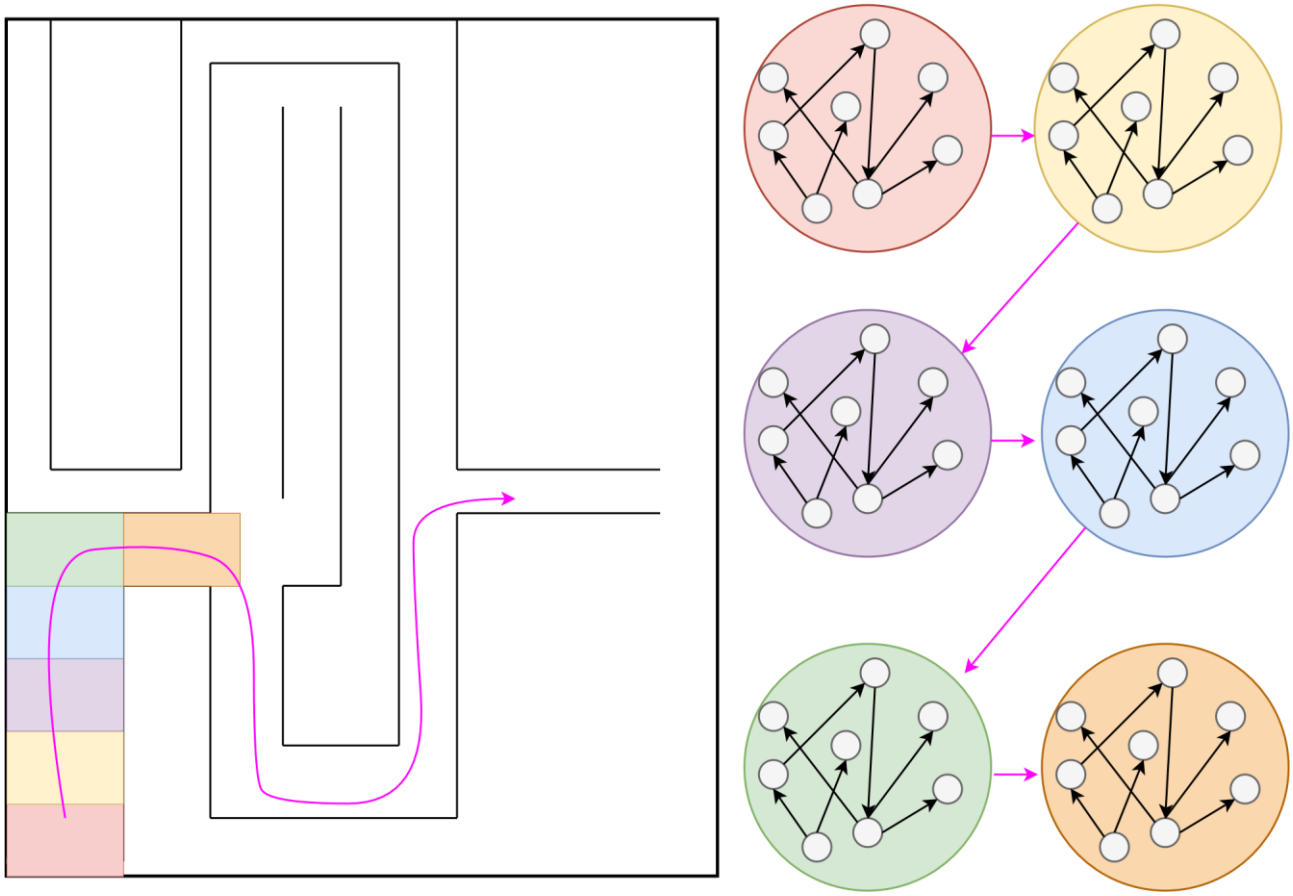


Figure 2.2: Visual depiction of the "memory as a canal" schema. This links the idea of sequential memory, temporal sequences, traversals on a graph, and physical causal relationships of the universe. This is also equivalent to the view of memory as a fading trace, although it incorporates the idea that the memory "dissipates" in the same way a physical system of particles does. The fact that the canal is in two dimensions is for illustrative purposes only; in reality it is in an N-dimensional space, where N is the number of neurons of the reservoir or memory. The walls represent the constraints imposed upon the flow of the activity by the weights and adjacency matrix (the topology of the memory). This can be seen also as a depiction of what happens in the maelstrom (Chapter 7). The activity, one can think of as the water in the canal, flows through the passageways as it progresses through the system; it is not "stuck" in a "stable" attractor as is the case for Hopfield or attractor networks, but does have a unique pathway for a given input. *Right:* The states of the activity of the reservoir or memory, represented with a different

color for each discrete timestep. At each step, the activity bounces around the network and dissipates, while following a "trajectory" in activity space. This space corresponds to one particular pathway in the canal. *Left:* The corresponding states represented as locations in the canal, with the memory activity trace of that reservoir consisting of one line through the canal. Different stimuli produce different pathways in the canal. They all start at the same place due to the fact that the initial configuration is the same. These activity states, as is clear from the graphical depiction on the right, obey a Markov property. As may be apparent with the Markov property, this begins to look similar to the reinforcement learning setup, although it has key differences.

be the quantum states of the gas atoms in the cylinder. We *could* go deeper, and also describe the states of the individual protons and electrons and neutrons, or we could go even further and describe the states of the quarks (or even Strings, should we subscribe to that explanation). More aligned with this work, for the brain, the base state I will consider for this section is a vector containing the voltages of each neuron's membrane potential in the brain. Alternatively, we can store a vector containing the concentrations of various ions, for each neuron for example, the main ions represented in the Hogdkin-Huxley model [Nelson and Rinzel, 1995]. A sufficiently rich vector, let's call it S, would allow us at any timestep t, given the vector (which we will denote S_t), to

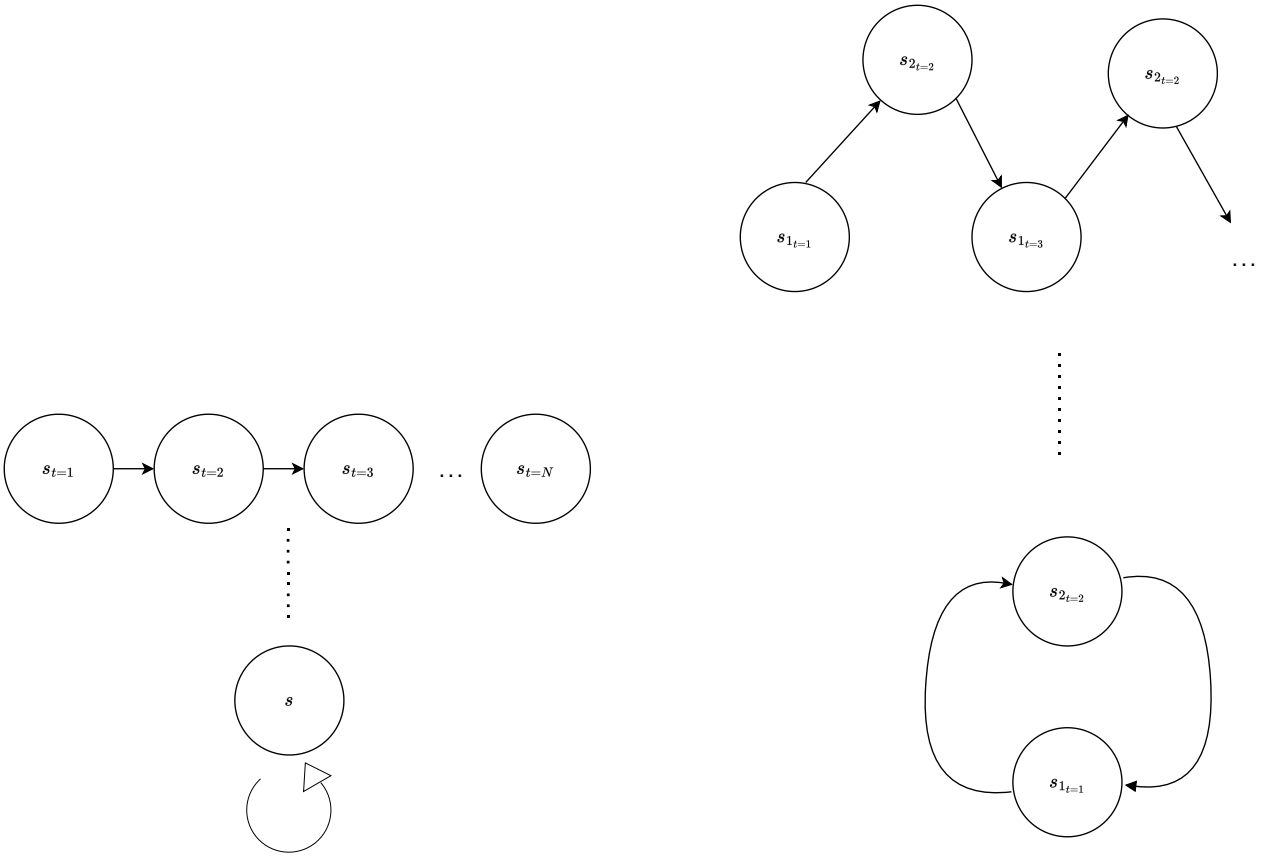


Figure 2.3: A Markov Decision Process graphical depiction of the time-varying graph of states, and how it relates to memory. Here, s denotes the state vector representing the state of a single element in the world, and the arrows represent temporal transitions to the next state at the following timestep. The subscript t denotes the next timestep. Importantly, here s is lower-case: it is not a single vector containing the entire system, but rather a sub-component (neuron) of the system (S). We can consider S to be the larger vector consisting of a list of all s. As a concrete example, s is the vector containing the state of each individual neuron in a brain, and S is the collective state of the entire brain. *Left:* The case where an object or system does not pass its information to other nodes, but retains the information itself (such as the case of Plato's wax imprinting memory schema from the *Theaetetus*

[Cooper, 2015]). The lowercase *t* denotes the timestep; here, in the case of memory, the notion of *state* is integrally connected with the notion of time. In this case, at each time, the state information flows back to the same object indefinitely. *Right:* the more general case where information traverses a graph, such as in a neural network. In both cases, rather than writing an infinite sequence in time of the information passing, we can reduce this to a simplified notation with the notion of a *Loop*: a dynamical connection that resonates and maintains activity within itself.

accurately predict the state at the next timestep S_{t+1}. This larger state can be composed of many smaller states (i.e., in the case of the brain - the larger state is the brain, while the smaller states are each neuron, see Fig. 2.3). This, however, does not give us any sense of the time-varying dynamics of the system; in the MDP framework, the states give us static information in the nodes, but the *transition graph*, the edges of the graph, gives us a description of how this information flows in time. As we are agents in time, and information can flow from one node to the next in time while retaining itself within the system, we need a way to describe how information can be contained, yet flow, at the same time. Next, we need to describe these flows, which I will refer to as Loops.

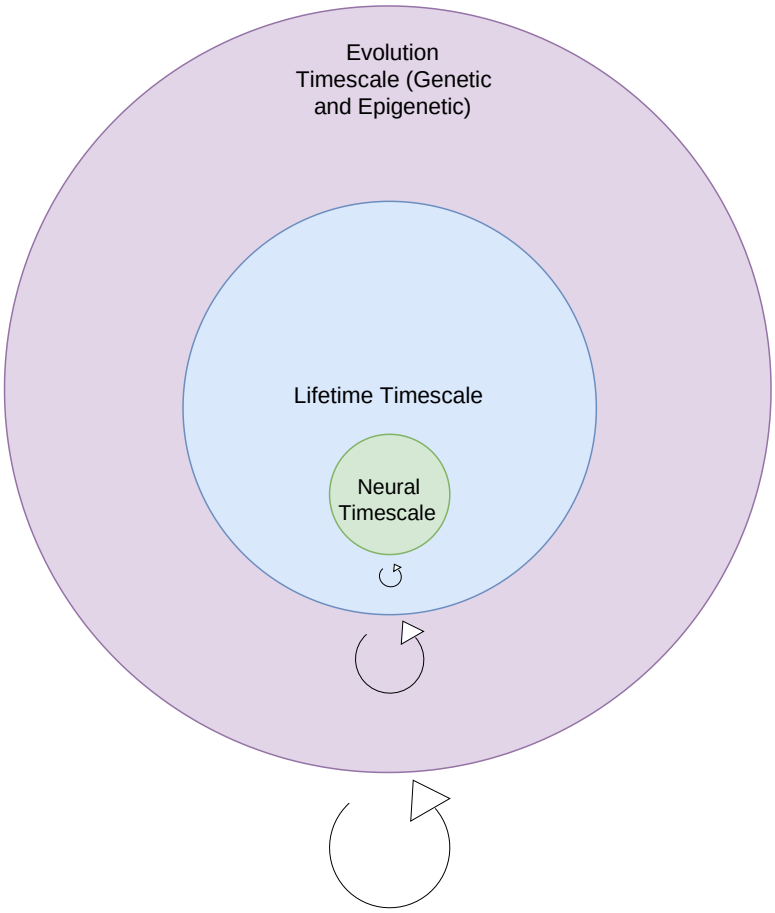


Figure 2.4: Loops occur in the brain on many levels (Adapted from [Wang, 2021]. This was also inspired from the talk by Robert Legenstein at PLENA, *"Porting few-shot learning principles to memristive hardware".* These can be seen as a form of memory, or state, as described earlier. On the outermost layer we have evolution, which guides through genetic (and epigenetic!) processes the inductive prior, such that learning can effectively take place. This

includes the architectural structure and topology of the brain and its regions [Edelman, 1987] as well as the reflexes and physics models that serve to "bootstrap" our learning processes [Lake et al., 2017].

Loops

Sequence Memory as Traversal in Temporal Graph

With just the state described, we can describe how information sits in one infinitesimally small time instant, but we have no way to reason about how this information is stored over time, which is an intuitively core requirement of memory. It is, after all, the interplay in time of the network of neurons that performs the magic of consciousness as I will argue, so describing these graph edges are really at the core of the connectionist framework.

This idea of matter, as a state, taking an "action", and resulting in a new state, is core to the concept of quantum mechanics as well. For the example of an electron that is bumped up to a higher orbit, this is an example of the state of the world changing from one where an electron was in a lower orbit, to one where it was in a higher orbit, now resting in a new state as a result of traversing the edges of the temporal graph. Matter itself moved to reflect the imprint of the action on the world. We can think of this electron moving to a new state as a subset of the larger graph S , s, representing the electron, following a particular path in the MDP memory graph.

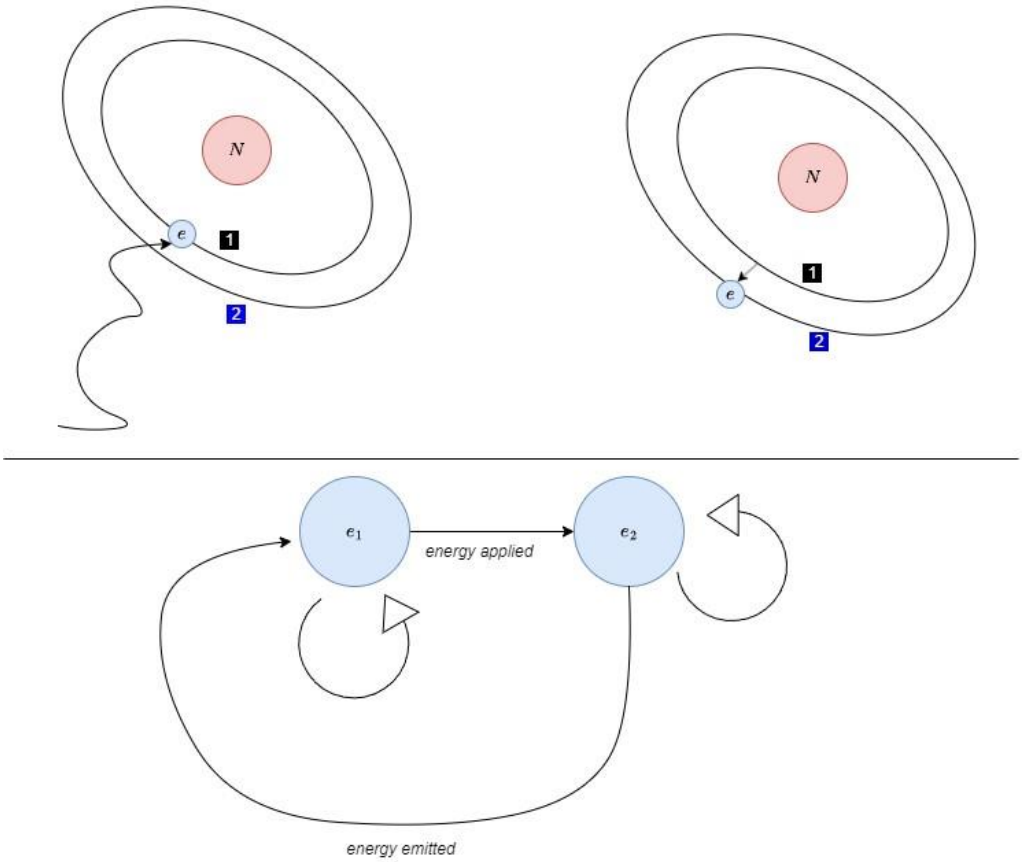


Figure 2.5: An electron, being struck by an errant electromagnetic wave, can be moved to a higher electron orbital; this is well known in classical and quantum physics. *Top:* The electron remains in an orbital until it is struck by

electromagnetic radiation, at which point it either falls back to a lower level and emits energy (emission), or it remains at the higher orbital .We can look at this as a form of encoding, the same way that RAM encodes memory in loops as long as current is applied. *Bottom:* I view this as a traversal through an MDP/DFA graph, and show the one-to-one correspondence between the encoding and traversals through these forms of graphs. Here, e_i represents the electron at the given orbital, and the edges are the transitions to new states, in this simplified version the two states are either the electron is at orbital 1, or orbital 2. As we can see, to remain in a particular state, requires the introduction of self-loops back to itself. As I mentioned in the beginning, however, an encoding without use for an agent is simply a recording; the fact that it has use gives it meaning and grounding. An electron that is bumped to a higher orbital but has no use for an agent is not a memory, however, electrons that are bumped up in service of action would be considered memories. This was also independently from me, noticed in [Von Foerster and von Foerster, 2003]. Note that the electron's movements are bound by the temporal causality of the universe: only this electron can move to the next orbit due to its past time history, and the temporal trace of this electrons movements are locked to this temporal sequence.

Resonant Memory

What can we call these traversals in the temporal graph that lead to memory? Clearly, from this view, these temporal traversals are the first phenomenon to occur as the stimulus floods into the brain, and the graph cycles "resonate" with the activity of the sequence memories. To this end, I am proposing an earliest stage of memory formation, which I am calling "resonant memory", which serves as the building blocks of later memory formation. These resonances eventually move into the hippocampus, and then back to the cortex for long term consolidation, keeping their core essence as temporal sequences but potentially being abstracted to shorter

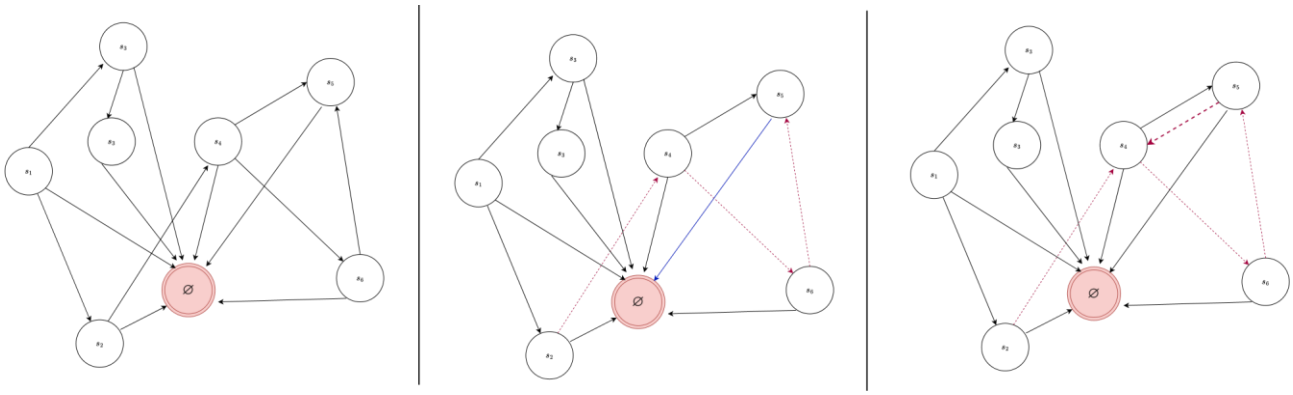


Figure 2.6: A depiction of sequence memory as temporal traversals in a MDP/DFA graph. The states *s* are numbered 1 through N, the maximum number of states, and represent the sub-states present; the collection of all states is *S*. The edges between the nodes represent the actions or activity that travels between states, and takes us from one state to the next. *Left*: Borrowing the idea of a terminal state in time from the DFA framework, the activity or actions *must go somewhere*, it is moving in time, and thus must take some action. Taking no action amounts to following the path to the null state, denoted by $\emptyset$ *Middle*: With this framework, we can define a dynamical memory, or a *memory engram* as it's referred to in the literature [Josselyn and Tonegawa, 2020], as a path in this DFA/MDP space. Here, I demonstrate an example potential engram in red (color best for viewing). However, there is a big issue here: without any sort of cycles, the activity naturally flows into the terminal state, represented by the blue arrow. *Right*: By adding one loop represented by the magenta arrow between s_5 and s_4, I have now created a path such that the memory can flow indefinitely without going into the terminal state and dissipating. This is related to the idea of temporally locked cell assembly activity as in
[Dragoi and Buzs´aki, 2006]

lengths. These have a tight relationship with Hebb's notion of cell assemblies, as I will expand upon at the end of the thesis.

The Memory Engram

Owing to this fact, we can describe a system which contains memory as an MDP, where *time* is represented by the connections between states, and the states are represented as nodes. As time progresses, information flows from the node-states through the connections to the next state. With this, we are able to abstractly represent both dynamical state as well as matter state using the same unified notation. But from the DFA definition, we need some kind of loop to remain in the graph. Remember, implicitly in the graph, without any form of loop, the activity or action goes to a terminal state, which we can think of as a dead state or no activity.

Thus, this formulation allows us to define for the first time memory as a traversal in a defined MDP of the space of an encoding model, however, it is incomplete as it does not handle how the system can perpetuate the memory forward in time, which is a core requirement of memory. Memory, as we describe, *must persist* across time. To be able to persist, it must retain the same information, forward in time, such that a system can decode its information into the same representation on the decoding end. In a dynamical system of nodes, a system that persists activity must contain recurrent connections, or *Loops*, or "cycles", in the graph, such that the activity can resonate on itself and persist across time. Without cyclical activations, the activity simply decays rapidly. In a more "permanent" memory storage, say as in Plato's wax tablet example, the state transition can be seen as simply looping back to itself indefinitely (it is in the same state at the subsequent timesteps) (Fig. 2.3, *left*). In a more complex system, where information is passed between multiple nodes before coming back to itself, for example as in a network of neurons, or people in a society, this loop may be more complex; however the information still flows back in a cycle such that the activity persists. We can assume, in this framework, that nodes with no possible connections or edges, flow into a *null* state, which is a terminating state where the activity ceases. We see this cyclical activity that resonates across many facets of the physical universe, from electrons that encode spin of an atom orbiting around a nucleus, to planets orbiting around a star, to waves rotating around each other propagating through space.

Figure 2.6 demonstrates the necessity of loops in this framework, and how this leads to the definition of a *memory engram*: A specific path in dynamical MDP/DFA space that contains a cycle, allowing for its indefinite resonance, assuming that some activity can perpetuate this. There is a strong connection here between this, and the idea that the brain encodes information in clusters of neurons called cell assemblies, which originated in Hebb's original work but has shown continued support in combination with a "readout mechanism",[Buzs´aki, 2010]; specifically, what is critical is that the temporal ordering of the cell assembly firing is critical, which has experimental support in the hippocampus where short-term memories are believed to be stored [Dragoi and Buzs´aki, 2006]. Importantly, the role of this readout mechanism, with respect to the temporal assemblies, is what I will argue for in the maelstrom paradigm in Chapter 7.

Assemblies and Readout

Lastly, how are these engrams decoded? Following on Hebb's initial proposal of the assemblies as the underlying currency of the stateful sequential memory, further theories developed what are called "readout" mechanisms, which are tasked with being "eyes" on the assemblies, and learning to map specific sequences of assembly activity to particular output functions [Buzs´aki, 2010]. This combination is what I would call the "assembly and readout"' paradigm of brain function. In bridging the gap between deep learning and brain function, we can notice that since the readout is a feed-forward function, it can be approximated by a deep learning network.

Memory

To summarize, I am proposing a solution to the discussion of memory in the brain, by proposing a new definition of memory, *sequence memory*. Sequence memory refers to memory within a specific temporal sequence, which encodes a strand of temporal-causal relations relative to a state of the physical world. The brain, acting as a dynamical system, can encode the sequence memories by specific activations along a temporal graph, the graph being the dynamical system of neurons. A memory engram is then smallest temporal graph traversal that

represents a specific concept. A memory engram is a specific traversal in the memory graph, which can be decoded at a later time to represent a specific aspect of the world. The decoding is done by a separate readout mechanism, which is tasked with learning the patterns of the temporal assemblies [Buzs´aki, 2010]. This temporal graph is implemented by temporally-activated Hebbian assemblies in the brain. Following the predictive coding view of memory, the brain is constantly using its memory, sequence memory, to make predictions, and thus is constantly re-experiencing the past, which can be one foundation for consciousness. This view of sequence memory also lines up with the brain's need to chain together events to remember past experiences, the inability of the brain to function as random-access memory, and the view of the brain as a dynamical system. The brain collects these dynamical traces of the graph, which I am calling *resonant memories*, and recombines them in a compounded fashion through the various memory networks, including consolidating back into the cortex for long term memory. It is worth noting that the long-term memories also share this chained feature. In the next section, I will delve into the current theories for general purpose memory in the brain, and make connections. I will then propose a novel deep learning paradigm, *Maelstrom Networks*, that account for learning sequential memories in the deep learning framework. The Maelstrom in effect learns to map resonant memories to actions.

3 Memory in the Brain

> The memory doubtless is, so to say, the belly of the mind: and joy and sadness are like sweet and bitter food, which when they are committed to the memory are, so to say, passed into the belly where they can be stored but no longer tasted. It is ridiculous to consider this an analogy; yet they are not utterly unlike.
>
> *St. Augustine*
> *Confessions X.XIV.21*

Now that I have given a theoretical and mathematical intuition for how I view memory systems *in general*, let's take a closer look to our current understanding of how this memory is implemented in the human brain. I will then offer a new idea for a new form of memory that I believe may serve as the building blocks for other, longer forms of memory. Before I delve into this new idea, it is important for me to clear up some issues regarding naming and taxonomy that have confused me in the past, and likely cloud an effort to create a unifying theory for memory both in artificial agents as well as in the human brain.

Defining Memory in People vs. Memory in Machines

Notably, as I have tried to impress, the brain seems to be more focused on sequences. This is in stark contrast to the way we encode memory in computers, which focuses on information as a jumbled set of uncorrelated data. Previously, I discussed a theory for the *how* the brain is performing its memory-related tasks. A discussion of the *why* will come later in the sections devoted to embodiment in Sec. 6. Now that I have given a more formal graph

definition of the implementation of memory in the brain, let's step back and give a more intuitive one. Loosely, we can think of memory as "having information about actions or objects that occurred in the past". The brain offers a "simulator" that allows us to relive experiences that already occurred, even though we have already moved passed them.[4] This is exactly the same role that other, more "solidified", forms of encoding also take, such as writing a note on a piece of paper; that writing enshrines information about the past that can be later decoded. This led early thinkers of memory to posit that memory is like "a wax tablet", taking an imprinting and holding it forever [5]. This simplified notion of a wax tablet or tape has shown to be too simplistic to describe the more complex mechanisms for retrieval of information. With that being said, current *electronic* memory systems, which also perform the same task of storing information for later use, *do* in fact use the simplified "wax tablet" idea for memory storage; the memory is exactly and perfectly stored in a way that mirrors its encoding, in neat rows, in an encoding that segments the memory into neat packages that can be decoded at a later time in order. The memory is (energy-permitting) non-decaying, and perfectly decodable (assuming the system is working) 10 years later, versus 100 years later. The brain, on the other hand, *does* decay, but only for *some* kinds of memories. For example, remembering where you went three weeks ago for food is likely not going to remain in your brain. However, remembering your first baseball game, likely *is*. It is apparent that there is something more complex occurring in the brain that stores memories not only by their content, but also by their emotional valence, as well as their type. For example, the kinds of memories we call "episodic" memories, where we can consciously recall a sequence of events, is a different kind of memory from a *motor* memory, for example riding a bike; however, according to our definition, these two kinds of memories *should* be identical. Intuitively, however, we do not necessarily see riding a bike at the same "level" as remembering an event from your past, even though philosophically these both boil down to remembering data from the past! Philosophically they are different, yet phenomenologically they are distinct. How are we to make anything from this mountain of at times contradictory information about memory? It is this mess of memory that I will make an attempt to disentangle and present in a coherent framework in the subsequent sections.

Taxonomy of Memory

Moving back a bit to the general concept of memory, not of specifically sequence memory, we can look at the current understanding of how the brain stores *any* information at all. [6]This topic of trying to create a unifying taxonomy for memory is itself worthy of its own dissertation, and I will not be making too many hard claims one way or another, but it surprised me looking through the literature that there was such a broad spectrum of ways to classify the phenomenological experience we refer to as "memory". For example, looking at [Squire, 2004], we can see an entire tree structure of memory emerge. The tree is asymmetric, and it's unclear how many of these tasks are actually the same "general" neurodynamical principles with just different outputs. Rather than view this a tree structure, however, I argue here that the different 'flavors' of memory are simply different factors of variation that the memory can take, and sit along orthogonal axes.[7] In this telling, there are three main axes of variation: the axis along which the duration of memory is measured, the axis that explains to what degree the central executive is involved (i.e., is it an 'automatic', reflex-like behavior, or does it involve contemplation), and the axis that dictates the mechanism by which memory is learned, is it unsupervised or self-organized, or is it error-driven as in the case of motor skills. In the next sub-sections I will describe these axes, and display a diagrammatic explanation in Fig. 3.1.

[4] This fact that it is simulator, rather than a "tape", is also why we can have false memories.

[5] From the *Theaetetus*

[6] I would argue, however, that it all comes from sequence memory, a topic I will broach with *resonant memory*.

[7] This is not an incredibly radical view as these kinds of memory have been described separately along these axes; the difference here is that I combine them all together into a single multi-axis system, which can make predictive power about potential kinds of memory (points in the axis space) that we may not have thought about. For example, a kind of error-learned, long-term, executive memory.

Short-Term vs. Long Term

Most of us intuitively already understand, from experience in learning, that we have a distinction between our ability to remember *short term*, or events that occurred recently in time, and *long term*, for those in the distant past. Experiments on memory that led to revelations such as those for Patient H.M. [Squire, 2009], showed us that there are specific regions in the brain responsible for short and long term memory distinction, and they have unique mechanisms associated with it. It is also being revealed that sleep plays a critical role in memory consolidation; this would help explain why sleep is so critical to us that we completely shut down without getting it, beyond just replenishing neurotransmitter stores. This theory suggests that the two main regions involved in declarative (long-term) memory are the *hippocampus* and the *neocortex*. New memories (short-term) are immediately associated in the hippocampus; the ones that are deemed novel enough for encoding are consolidated or "hardened" in the neocortex during sleep [Feld and Born, 2017, Marshall and Born, 2007]. This is a very interesting suggestive model that can give us insight into designing mechanisms in artificial systems for short term memory encoding.

Explicit vs. Implicit

The next split in literature comes between memories that we must activate consciously, such as memories where we follow a chain of episodic events to recall, versus memories that spontaneously occur without or conscious effort. These are split in the literature according to the labels *explicit* and *implicit*. Here, for our purposes, the most important distinction is that for explicit memories, there is a clear involvement of a central executive function, whereas for implicit memories, the memory goes forward "as is", so to speak, and occurs as a result of deterministic flows of activity from stimuli, or current brain processes that trigger these engrams. An example of an implicit memory would be a learned eye-blink behavior; this would be a motor memory, located within the cerebellum [Krupa et al., 1993]; it is a memory because it is a learned behavior from past experience, but it has no conscious component. The two major classes of explicit memories are "semantic" memories, those constituting facts, and "episodic memories", or those constituting sequences of events. From an embodied memory viewpoint as in [Glenberg, 1997],I would personally argue this distinction is fabricated, as both are eventually grounded in action about the world. This also has important connections to deep learning; we can think of the weights of a neural network as "implicit memories", according to this taxonomy, because they are encodings of the past (training data that has been seen before, mapped to labels), that output a specific response *automatically* (i.e., without the role of a contemplating executive, resulting from the dynamical flow of the activity up from the lower levels to the higher levels). From this perspective, all of AI currently is being dominated by implicit memories; the space of potential work where an executive dynamically manipulates working memory is completely lacking. The viewpoint that I will be proposing is one that delineates explicit and implicit along the same lines as the executive system and the involuntary system, one that delineates whether the system runs automatically responding to stimulus, or whether the system runs responding to its own internal model (state) running. In this viewpoint, state is essential for an executive system as well as consciousness.

Error-Learned vs. Associative

Lastly, we can describe memory according to its mechanism of creation, which splits naturally and interestingly along the main differentiation for machine learning approaches: whether the memory was created according to some supervisory signal (supervised learning), or whether there was no supervisory signal and the memory self organized according to similarities among the data itself. This also splits biologically into the difference between operant conditioning, where a stimulus is associated with some other stimulus regardless of reward, and classical conditioning, where a stimulus is associated with a reward or punishment [McSweeney and Murphy, 2014].[8] An example of an error-learning memory would be riding a bike, when we fail, we fall off the bike (punishment), and we continue to practice until we perfect the task. It appears that separate regions of the brain are responsible for

[8] It seems likely that operant and classical conditioning relate strongly to associative learning [Diamond and Rose, 1994], and my view is that everything in the brain is likely associative, and that we simply associate with rewards rather than with neurons in either case.

these more error-learned memories, such as the cerebellum's postulated role in error-corrective motor learning [Popa and Ebner, 2019], whereas the hippocampus is more associated with associative learning. This brings up an important point, that the topological structures of the brain segment functional roles much more clearly than we attempt to do in deep learning.

The Importance of State, Implicit, and Explicit Memory

Thus, when we delineate the kinds of processing in the brain, we can see a strong relationship between the split of implicit processing versus explicit processing, and conscious versus unconscious behavior. We can see that what defines implicit processing is its complete reliance on the stimuli, and its unwavering devotion to following through on the stimuli as if it were a function. The implicit memory is thus akin to a rock rolling down a hill, a complete servant to the laws of the universe. In terms of how the brain has impacted deep learning, all of the amazing results in connectionist AI thus far have been from completely implicit processing.

Explicit, Implicit, and Sequence Memory

We can even go further, and try to see the connections between implit memory, explicit memory, and sequence versus non-sequence memory. As implicit memory is a completely feed-forward process, based on the immediate stimuli, we can argue that implicit memory is that memory which operates on non-sequence memory. Explicit memory, as it requires an executive, and is responsible for unrolling chains of memories, can then be seen as that

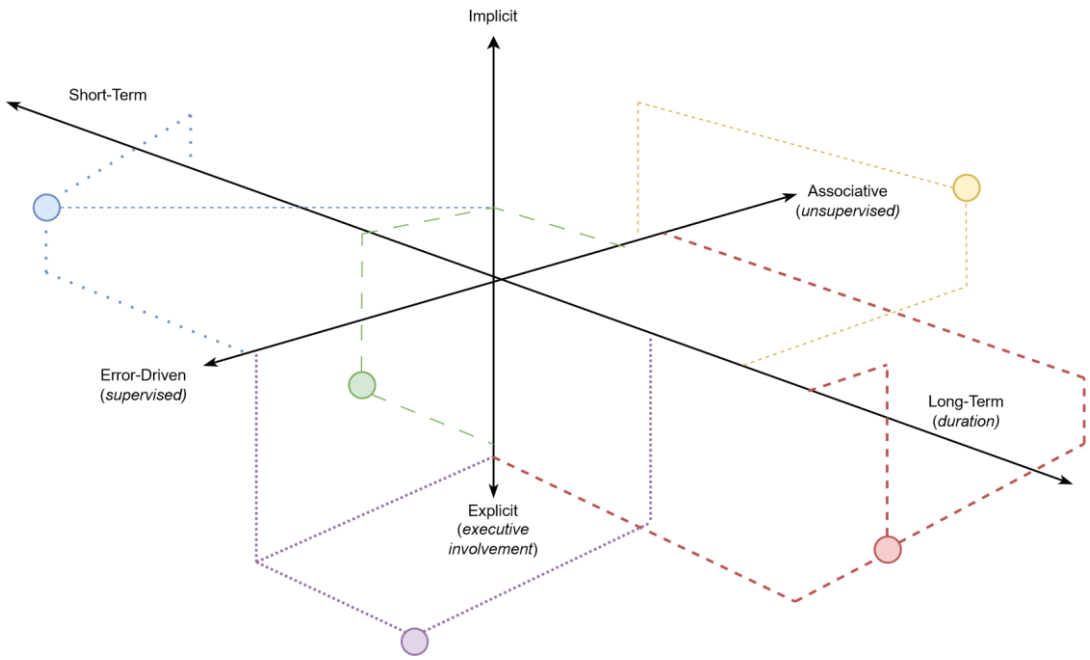


Figure 3.1: I propose that memory lies along three axes of separation. One axis dictates the *duration* of memory, ranging from long-term to short-term. The second axis dictates the *executive involvement* in extracting the memory. And the third axes represents to what degree is the learning of the memory *unsupervised*, which corresponds to associative learning, versus *supervised*, which corresponds to error-driven learning, also referred to as procedural learning. This also measures the degree to which we are consciously choosing to remember a certain engram. I believe the disparate kinds of memory can be separated on this axis. For example, the red dot represents what would be called *episodic memory*, which is a sequence of events that occurred to the agent with no real regard to reward or punishment except in terms of emotional valence. The purple dot, which is trained using error learning, would represent motor skills such as riding a bike, the green dot represents elements in working memory. Yellow is an example of a long-term associative memory that is unconscious, for example a long-term learned fear aversion to visiting a specific place because of a bad event that occurred there. This also somewhat makes

predictions for new forms of memory, such as the light blue dot, which would represent a form of priming in short-term memory tasks [Cowan, 2008], but for procedural (error-learning) tasks. The projections (dotted lines) are meant to help guide where the points lie in the 3-d plane projected onto the 2-d image.

memory that operates on sequence memory. This is not so clear-cut, however, as you can have an implicit-like memory mechanism that runs on a running sequence memory, which itself operates in an implicit fashion, but contains information about the sequence. What is clear, however, is the reverse: that executive, explicit memory proper is the purest form of sequence memory, and implicit memory (proper) is the purest form of non-sequence memory.

Memory Leads to High Intelligence

Here I will make the argument that memory regions play an important role in the boost in intelligence that humans hold versus other animals. We can ask, why is it that humans, and mainly humans alone (save some exceptions in dolphins and elephants) can perform such incredible self-aware tasks, community feats, and leaps of science, that completely eclipses our nearest neighbors? Surely it should not be so drastic as it is now. My main point of evidence will be a topological one, borrowing from phylogenetic evidence. It has been argued that the hippocampus, the ability to remember episodes of past experiences, has been a hallmark in higher intelligence [Murray et al., 2018]. It can be shown that almost 500 million years ago, an analogue to the hippocampus evolved in a more primitive reptiles and vertebrates, showing that the seeds of this region were planted long before hominids evolved. While it has been the dominant view of evolutionary neuroscience throughout the 1900s, newer analyses of phylogenetic linked growth of brain regions has showed that the growth of the neocortex is not, in fact, significant enough to explain the jump of hominids from earlier primates [Murray et al., 2018, Schilder et al., 2020]. Rather, newer theories suggest that it is a complex combination of divergent tracts of evolution that have recombined to create our intelligence, as well as our mediating networks that integrate these disparate regions and topologies [Hampshire et al., 2012] - another point of evidence that the segmented topology of the networks plays a crucial role. Perhaps, then, this mediating network can be seen as the “seat of consciousness”, or responsible for executive action - but this is speculation. Figure 3.2 displays the table taken from [Rapoport, 1990] that shows relative size increase of various topological regions of the human brain. This appears to be an open area of investigation, so I believe a hypothesis that memory processing plays a key (potentially *the* key) role may in fact be a novel contribution looking forward.

Brain regions having accelerated progression during primate evolution

Brain region	*PI[a] humans*	*Division with maximal progression during primate evolution*
System I: telencephalic regions, including amygdala and hippocampal formation		
Neocortex	156	association areas, including: — prefrontal cortex — visual association — Broca's speech area (44 and 45, Brodmann) — inferior parietal lobule (39 and 40, Brodmann) — Brodmann area 37 (part of Wernicke's speech area)
Corpus callosum	↑	
Amygdaloid complex	3.9	cortico-basolateral group
Hippocampal formation	4.2	subiculum, CA1, dentate gyrus
Entorhinal cortex	5.5	
Septum	4.0	
Nucleus basalis of Meynert	↑	
System II: segregated circuits involving basal ganglia and thalamus		
Basal ganglia	16.4	internal segment, globus pallidus
Thalamus	↑	pulvinar, dorsomedial nucleus
substantia nigra	16	pars compacta
Cerebellar and mesencephalic-midbrain catecholaminergic nuclei		
Cerebellar nuclei	?	dentate nucleus
Red nucleus	↑	parvocellular division
Raphe nucleus	?	
Locus coeruleus	?	
Ventral tegmental area	?	
Retrorubral nucleus	?	

[a] Progression index in *Homo sapiens* (ratio of weight of human brain to weight of brain in primitive notional insectivore of equivalent body weight).
↑, PI > 1 but value not known in humans; ?, PI not reported.

Figure 3.2: Taken from [Rapoport, 1990], the topological areas that saw the most growth among primates. While it is clear that the neocortex did see growth, newer arguments [Murray et al., 2018, Schilder et al., 2020] are being presented that the neocortex alone cannot explain the remarkable boost in intelligence that hominids achieved relatively quickly. Interestingly, the areas that show the most improvement are the basal ganglia (error-based memory learning), the cerebellum (error-based memory learning), and the hippocampus (self organized based memory learning), the amygdala (emotional memory) [Sah et al., 2003], and critically the entorhinal cortex [Garcia and Buffalo, 2020], which is the main gateway and processing conduit and organizer for information coming from the cortex, and the septum [Rizzi-Wise and Wang, 2021] which is another 'hub' region responsible for integrating emotional and other memory information, including to the basal ganglia .

4 Neural Networks

At the time that the first perceptron model was proposed, [I] was primarily concerned with the problem of memory storage in biological systems, and particularly with finding a mechanism...It soon became clear that the problem of memory

> mechanisms could not be divorced from a consideration of what it is that is remembered, and as a consequence the perceptron became a model of a more general cognitive system, concerned with both memory and perception..
>
> *Frank Rosenblatt*
> *Principles of Neurodynamics*

The concept of memory is intertwined with the study of connectionist models for artificial intelligence. The parameters that we learn in the neural network are a form of memory, that remember "on demand", immediate memories (implicit) that are mapped to an action [Rosenblatt, 1961]. As this work introduces a new framework built on neural networks as the core 'mapping' enabler, we now turn to the basic outline of the current state of the art of neural networks from its foundation upward. As this is a huge field with numerous courses dedicated to this one area, I will do my best to condense all of this into a coherent section but can make no promises.

Historical Notes

What inspired us to begin to look at neurons in the brain? Likely it was the innate desire for discovery in all of us, to learn what makes us tick, and the core of consciousness and free will. After initially thinking the core of humans was in the heart [Figueredo, 2021], in more recent centuries it became clear that the brain was actually the "seat" of consciousness. The origins of neural networks for artificial intelligence truly began with neuroscientists, not engineers, trying to discover how the brain works, although now reverse-engineers, or neuromorphic engineers, can use this knowledge to impart brain-like behavior into constructs.

Cajal and Hebb

The beginning of artificial neural network research truly begins with the revelatory work of Raymon Cajal in the early 1900s [Ram´on y Cajal, 1906], who discovered that neurons, and transitively the brain, is not born with all of the knowledge it will possess (a fact that would have been common knowledge to anyone since antiquity), but that the neuron itself undergoes chemo-physical changes that alter its structure and operation, after repeated exposure to stimuli that provoke this change. This idea carried into the work of Donald Hebb, who in 1949 wrote *The Organization of Behavior* [Hebb, 2005], which further developed Cajal's ideas from the notion of individual neurons to populations of neurons, gifting us a notable phrase *neurons that fire together wire together*. This elevated the role of learning above the individual neuron to *groups* of neurons. This important and profound idea, that it is not the individual neuron but rather the community of neurons that perform computation, still guides us today; an idea that also has deep insight for our operations as humans within a society. It is thus not the individual synapses within a neuron that provoke behavior, but rather the *graph* structure of these neurons within connected components.

McCulloch, Pitts, Frank Rosenblatt and the ***Perceptron***

The idea that we can model the nervous system as networks of simple units began in earnest with McCulloch and Pitts, who showed that binary neurons with step functions could serve as logical gates [McCulloch and Pitts, 1943]. However, current understanding is not that the neurons are small logic operators, but rather recognizers of signals from the environment. The idea of neurons as signal processing units comes from Frank Rosenblatt and the origins of *artificial neural networks* as a machine learning tool, which was the beginning of the connectionist deep learning revolution. In 1958, A working for the US government, Rosenblatt condensed the idea of individual neuron units

and groups of neuron units into a simplified model which he termed the *Perceptron*. The perceptron is a beautiful melding of two completely disparate fields: neurobiology and statistics, into one unified graph structure. The perceptron, like the biological neural network, aims to "learn", or adjust strengths of the neuron's connections (as a result of Cajal's seminal work [Ram´on y Cajal, 1906]) to better match performance needed for some task. Although today the Perceptron is taught and thought of as a single-layer network, in the original manuscript Rosenblatt was actually fully aware of the multi-layered *hierarchical* nature of the network structure, but did not have a learning rule to update the weights of this structure. While the current zeitgeist is that this learning rule would come about in the 1980s with *Backpropagation* [Rumelhart et al., 1986], some have tried to correct an unjust record, and show it arose as early as 1970 [Linnainmaa, 1970].

Neurons

What is a single neuron? A neuron (arguably) is the most basic computational unit of the brain. Figure 4.3 shows a crude rendition drawn by the author of the stereotypical "neuron". One result that neural networks has us, however, is that the individual neuron itself is not quite that "smart". Although it is a well-accepted fact currently that neurons are not simply "point" computations, and that the dendritic arbors (the multiple terminals that accept voltage from other neurons) can actually perform simple computation themselves [London and H¨ausser, 2005], I will make the case here that these are still relatively simple computations, and that higher-order advanced computation is still the result of the *network* of interconnected neurons.[9]

Fully Connected Network - Multi-Layered Perceptron

The perceptron is a great tool for pattern recognition due to its ability to learn the correct weights of the input, in a sense learning the kernel it is to apply to the data to map each input to a correct output. As was proven in an ill-sighted work, Minsky proved that the perceptron could not learn non-linear patterns [Minsky and Papert, 2017], while also implying that this meant that they would be of no use to the A.I. community. Now, given that connectionist deep networks have completely taken over the field, this was clearly an incorrect prediction, and while Rosenblatt's multi-layered networks ended up losing the debates of the day, their enduring legacy shows that Rosenblatt has the last laugh. Regardless, the backpropagation algorithm eventually led to the resurgence of connectionist, perceptron-driven networks in the 1980s forward, and the subsequent deep learning revolution. Figure 4.4 shows the canonical MLP, with the input being fed into the lowest level, and a target to be mapped to outputting from the last layers. The network, through training,

[9] This is not too dissimilar from the impressive feats that humans can do in a society, versus what one human can do on their own.

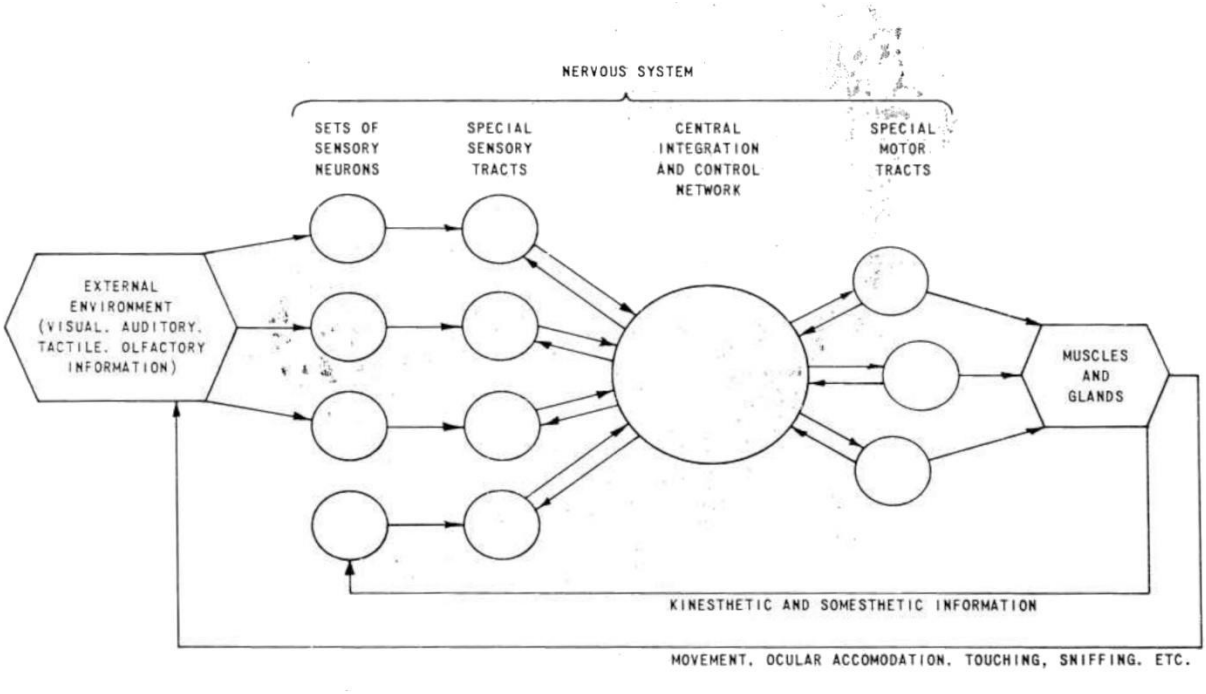


Figure I BASIC TOPOLOGICAL STRUCTURE OF THE NERVOUS SYSTEM AND ITS SOURCES OF INFORMATION

Figure 4.1: A diagram of Rosenblatt's system level architectural understanding of the nervous system, taken not from his original work on the *Perceptron*, but on subsequent elaborative work that also included the bigger picture of recreating the brain [Rosenblatt, 1961]. The dreams and goals of Frank Rosenblatt are a guiding principle for my work here - bridging the gap of statistical learning and neuroscience, and trying to fuse the two together into a system more powerful than the sum of their parts . This schematic highly influenced myself and my work here can be seen as a dynamical expansion of his original work. As is clear, Rosenblatt did not have a simple statistical learning dream, or pattern recognition, which is the main focus of current AI, and is the reason d'etre of machine learning. Rather, he viewed the system as a control loop, one that takes actions and explores the world. In this diagram, the "central integration and control network" is akin to the executive controller I will discuss in later sections. We see that the loop backwards creates the recurrent loop that I would argue is the critical defining motif of consciousness and awareness, and ultimately life. I delve more into the role of control systems in Chapter 6.

learns the parameters for the synaptic weights (the black connections in Fig. 4.4), which is analogous to our current understanding of the brain's learning of synaptic strengths during learning, which in both cases forms the core of learning. From this perspective, Rosenblatt's "zoom level" of learning synaptic connections in a graph of simple units has held the test of time.

Convolutional Neural Network

The next big evolution in (feed-forward) neural networks came with the introduction of the Convolutional Neural Network [Fukushima et al., 1983, LeCun et al., 1989]. The idea of this was also inspired by the brain, particularly the visual cortex, as in [Fukushima et al., 1983]. The human cortex has a highly structured and patterned topology of many neurons connecting in a pyramid-like structure upward to one neuron, called *convergent* connections

[Bullier, 2003]. Naturally, for a given patch of connections, this is a fewer number of synapses than if all of the neurons were connected in an all-to-all fashion as in an MLP. Thus, the introduction of the CNN actually *reduced* the number of learnable parameters for the network drastically, but increased the performance equally drastically. One can argue that the introduction of backpropagation-trained CNNs in [LeCun et al., 1989], com-

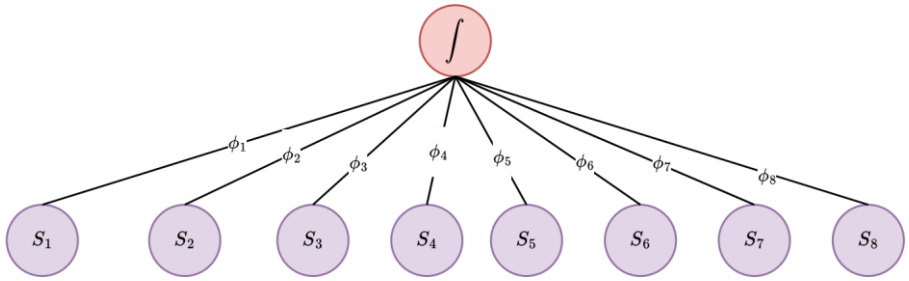


Figure 4.2: A single-layer variant of Frank Rosenblatt's original *Perceptron* which would eventually become the canonical perceptron, although Rosenblatt actually envisioned the perceptron as a multi-layered network, which was later skewed by Minski in his famous ill-advised retort that killed off neural network research for 30 years. The input stimulus features each flow through their own neuron; for 8 input features, we would have for instance, 8 input neurons (numbered here $S_{1...8}$). In a single layered perceptron, these are not actually neurons, but rather each "neuron" is the value of the stimulus at that feature index. It is closer to think of them as the rods and cones in the retina, which directly fire from the stimulus, rather than integrating information from other sources as true "neurons" do. This also holds true for the multi-layered perceptron case. Each of these "neurons" (stimulus inputs) has a weight, labeled $\phi_{1..8}$, accordingly. The task of perceptron *learning* is to adjust the values of the ϕ to maximize the performance of the network. This equates to minimizing the negative performance, or *error* of the network, on a given task. The fact that this strongly resembles statistical regression demonstrates the tight coupling between statistics and neural networks, even though the fields began in mathematics and biology, respectively.

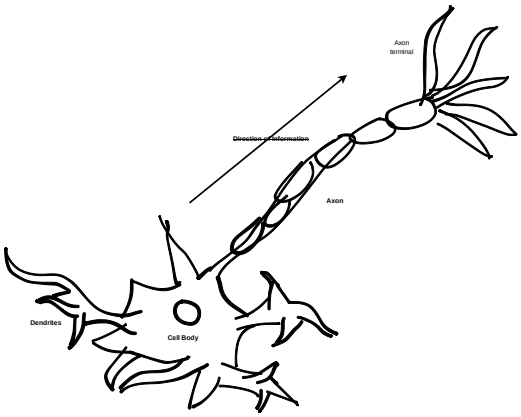


Figure 4.3: The author's rendition of a stereotypical Neuron, the building block of the biological neural network, and inspiration for artificial ones. The most important elements are the Cell Body, which is the *state* of the neuron, the Dendrites, which *receive* information from other neurons, and the Axon Terminal, which *sends* information to other neurons. In terms of parallels to artificial neural networks, the cell body is only present in models that contain a state such as recurrent networks, whereas the "weights" refer to the dendrites and/or axon terminal, depending on if you assign them to the sender or receiver. The information flows from the dendrites to the axon, although it is possible to send information backwards [Buzs´aki and Kandel, 1998]. This means that for all intents and purposes, each individual neuron is functionally *feed forward*. To achieve recurrence, the neuron must connect back to itself in a loop through other neurons.

bined with the increased computational power granted by graphics cards, enabled the so-called "deep learning revolution", which began in earnest with the very deep variant in AlexNet [Alom et al., 2018].

Recurrent Networks

Concurrently during the revolution that moved towards the inductive bias, was the revolution that this thesis will more closely focus towards: the idea that instead of connecting the graph of the neurons and the subsequent

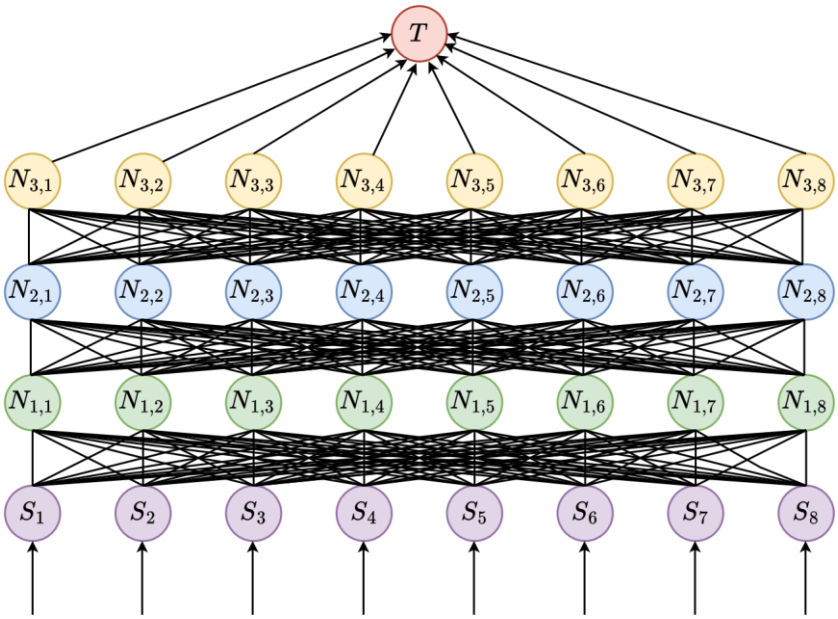


Figure 4.4: The real revolution of deep learning has come from the hierarchical nature of the multilayered perceptron (MLP). MLPs are as their name suggests, created by taking a single-layer perceptron as described here, and stacking them into feed-forward layers. The input is fed into the lowest layer, with each neuron unit getting a single feature from the feature vector. The information then passes through the network, and the subsequent layer 'sees' the next layer's output as if it were the stimuli coming from the bottom; however, unlike the bottom layer, each neuron actually receives information from each of the lower level neurons' outputs. At the last layer, the output from each neuron is passed through one last integration represented as T, where it is trained to output a target value, either a regression (real value) or a class label (integer). This kind of structure has been found to be somewhat plausible biologically, as most neuroscientists believe that early layers of the sensory cortex are "functionally" feed-forward. The mathematical form of this network was hypothesized by Rosenblatt in 1962 [Rosenblatt, 1961], long before the deep learning revolution. However, as he stated, at the time we did not have a sufficient learning rule that best assigned credit to each synapse for training. This learning rule would eventually come in the form of backpropagation [Linnainmaa, 1970, Rumelhart et al., 1986], which allows for mathematically exact credit assignment, albeit at a hit for biological plausibility. In terms of pattern recognition as a simple mapping from features to labels, it is hard to beat the performance of backpropagation. However, I will argue, this feature mapping is only one part of the brain's function, as it misses any form of state. All subsequent advancements in deep learning have simply been better ways of organizing these MLPs into various new structures. It is this missing state that is the primary focus of this thesis. information flow only *feed-forward*, such that the graph is directed and has no cycles, we can instead loop a connection back onto itself and create a cycle within the graph. This leads to a structure referred to as a *recurrent* neural network, or RNN. The fact that feed-forward networks have no cycles implicitly means there need be no *state* that saves the past input. When we have a feed-forward structure, all of the computation can be thought of as taking place in a "time-less" space where the computations occur simultaneously and output some value - we can think of this as timeless in the same way that computing a function $f(x)$ is timeless, we simply place in our input x and receive an output mapping. Having a loop or self-loop in the graph means we must *save* the previous value of computation, to

compute the next value in the sequence. This saving of the value, or state, can be seen corresponding to the *memory* of the system. If our function, now rather than $f(x)$, is $f(x,x_{t-1})$, we must store in some kind of state the value of the previous timestep x_{t-1}. It is interesting to think about the causal ordering of this evolution: if the state evolved as a means to save the previous timestep, or if the previous timestep evolved in service of fulfilling some other function. It is this saving the previous timestep that makes RNNs unique from MLPs and other feed-forward networks [Boden, 2002], and it is this aspect of a trace of the past that I argue is critical and fundamental to executive function as well as consciousness.

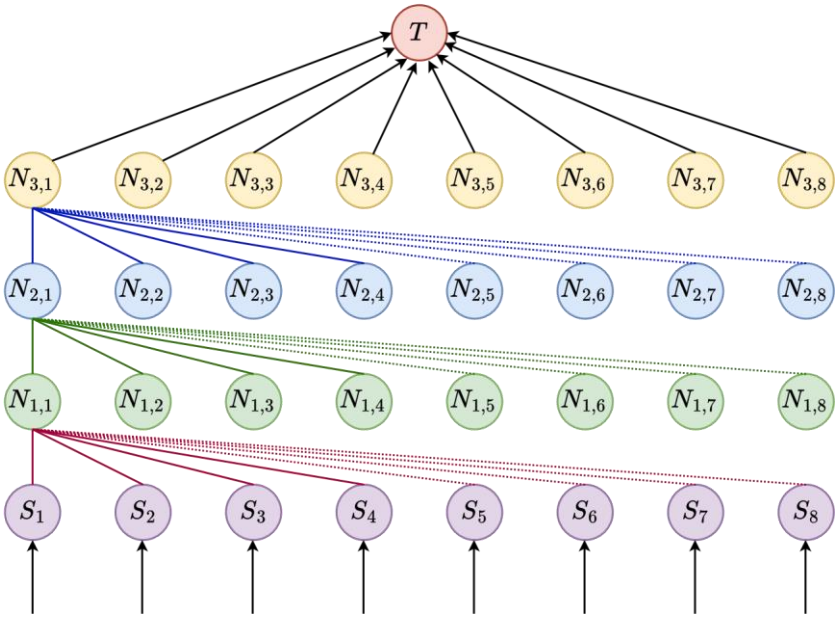


Figure 4.5: The CNN [Fukushima et al., 1983, LeCun et al., 1989] imposed specific architectural 'constraints', which we can also think of as inductive biases, into the fully-connected MLP. In effect, this removed the number of connections in the network, and only has truncated connections from a receiving neuron to a "patch" of neurons (solid lines). The patch is then shifted across the signal (1-d or 2-d), using the same weights. These small patches of weights are referred to as "kernels" in the literature. Different colors denote different layers. In a CNN, there are actually multiple parallel kernels for each layer; this taps into the idea that random initializations of the kernels at the start of training can lead to different representations learned. This architecture is roughly inspired by the convergent connections in the mammalian V1 cortical neurons [Bullier, 2003]. The important aspect to glean from this is that imposing some prior on the architecture, even through removing connections and constraining the model, drastically improved the training and accuracy of the network. This shows that topologies and organization of the network is key to understanding how these networks lead to functional intelligence.

Attention Mechanism

Last but not least, the final biological inspiration for neural networks comes in the idea of *attention*; attention is the notion that the brain, or an intelligent system, must selectively focus on certain aspects of the stimuli for maximum processing capability. While the brain likely does not have a tape-processing paradigm as in Von Neumann architectures (which naturally creates an idea of attention already), it is suggested that the brain focuses on particular stimuli to aid in processing [Petersen and Posner, 2012]. This idea of an attention mechanism was loosely integrated into deep networks in the LSTM framework for language translation [Bahdanau et al., 2016], although I would argue this is less "attention" and more a simple gating mechanism, which assigns different weights to different parts of a sequence. A true attention mechanism would also account

for an executive that is making decisions *potentially against* the implicit stimuli, but these attention networks are still themselves completely implicit and stimulus driven. One interesting viewpoint for executive systems then, is to think about to what degree the system is stimulus driven, and what degree the system runs on its own internal model.

The Role of Inductive Bias in Architecture Performance

Perhaps one of the most compelling stories of the evolution of neural network design, and the most likely explanatory variable, is the growing acceptance of the paramount importance of *inductive bias* in the design of the network. Inductive bias means that we have some prior knowledge of the world's likely distribution, and we pre-design *certain* parts of the network to better match the learning environment. In nature, this "pre design" comes from millions of years of evolution and trial and error (for artificial neural networks, it also comes from trial and error to a large degree!). As an instructive example, take for instance the multi-layered

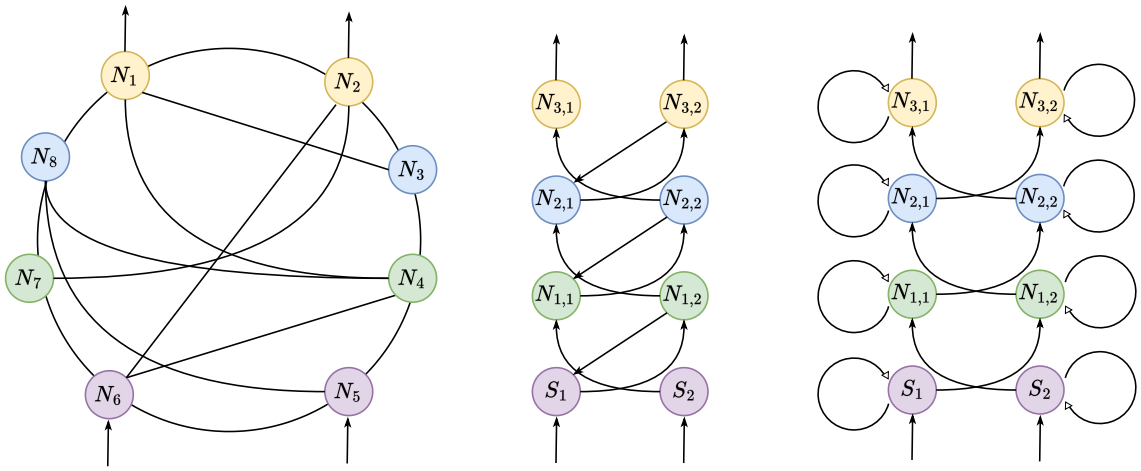


Figure 4.6: The recurrent neural network creates a state by adding self-connections, or cycles, either to a single neuron or to a group of neurons. Displayed are there example topologies of recurrent networks - the introduction of cycles drastically increases the number of possible topologies you can achieve. *Left*: A Hebbian style network, where the neurons are more closely all-to-all self-connected (connections are bi-directional). This is more typical of the *reservoir* topologies, where a percentage of connections (from a fully-connected RNN) are removed randomly. While it is shown here that the input goes to two neurons and is read out from two other neurons, this need not be the case; generally for reservoirs all neurons are fed in, and all are read out. However, I will argue that for a larger cognitive system, it is imperative that certain neurons be separated from input and certain from output, to preserve topological variation. Note that in this highly recurrent network, we cannot label neurons according to "layers" anymore. *Middle:* a more layered network that creates a 2-cycle between neurons (For example, S_1 and $N_{1,2}$.) *Right:* An RNN showcasing self-loops.

perceptron. It actually contains the maximum number of learnable parameters or weights that are possible in a feed-forward network with all-to-all connections and no skip connections. From this fact, we might assume that more weights equals better learning, or more powerful computation. However, by *reducing* the number of weights, Convolutional Neural Networks somehow *outperform* multilayered perceptrons on image classification tasks. How can this be, when MLPs contain more learnable parameters on average? The answer is hypothetically that the inductive bias in creating the learnable *filters* better matches how images are structured in the real world. This prior knowledge of a better fit for learnable convolutional filters, over all-to-all connections, is an example of an inductive bias creating better network design. In our work, example for this is in DOT-VAE, where our inductive bias of the bottleneck structure better extracts latent, disentangled features.

Reservoir Computing: State and Embodiment

An interesting line of development started when researchers began to think about what it means for a neural network to have a state, and a fading trace of its past, and what would be required for a network to maintain a persistent activity. We can think of the minimal requirements as thus: the network must, as I have made the case for, have cycles within its graph that propagates and resonates its activity back onto itself, and this activity must be stable such that the network does not go into a state where its activity is completely saturated, which is analogous to an epileptic seizure in the brain. In this simple assumption, we do not assume that we can actually train the recurrent weights with much certainty, and in fact, we start to ask if there is any wisdom in hand-tuning the weights for a sequence which will be fixed regardless, a point I will come back to in Chapter 5. We can assume that evolution gives us a specific topology, a sort of basis set, that projects the activity randomly into a higher dimensional "pool" of recurrently connected neurons, which reverberates the activity onto itself via cyclical (recurrent) connections. We need not focus too heavily on fine-tuning each individual connection in the recurrent weight matrix, as overfitting to a specific sequence is not the job of the recurrence, but rather the job of a *readout* mechanism, which is some mapping function that takes the current state of the pool of neurons, and maps them to some target action or label. This leads us to the idea of *Reservoir Computing* [Jaeger, 2001, Maass et al., 2002]. Because the recurrent pool merely projects the activity from an input and serves as a memory, we can think of these neurons as either spiking or rate-encoding, but their function does not depend on this fact. With this, we have effectively separated memory (the reservoir) as an independent entity from the readout (the gradient-trained network). Thus, with reservoir computing as the core recurrent component, we can see the beginnings of how to design a more complex system that maintains a temporal trace of its past in a state vector (the activity of the reservoir neurons), which separates itself from the gradient learning (versus LSTMs), and which also speaks to the biological plausibility (the recurrent weights need not train via backpropagation). This idea will serve as the backbone for my later proposed Maelstrom paradigm, and will be core in the connections between this and state space models. However, "reservoir computing" is a loaded term, and implies that the system also has a linear readout, and the other requirements that were laid out in the original paper. For this reason, while I borrow the ideas of the reservoir from reservoir computing, ideas henceforth are not exactly reservoir computing as such, but rather something that has grown out of reservoir computing.

The components of a reservoir are thus: an input matrix, which projects the input signal at each time, the reservoir, which serves as the state of the system and is its core sense of embodiment, and a readout mechanism, which maps the current state of the system to a target label. Here, I use the leaky update equations from [Tong and Tanaka, 2018]. At each timestep t, input is fed into the reservoir through a weight matrix W_{in}, and the reservoir and output vectors are updated according to the following equations with activity from time step $t-1$:

$$x_t = (1-\alpha)x_{t-1} + \alpha f(W_{in}u_t + W_{rec}x_{t-1}) \tag{4.1}$$

$$y_t = W_{out}x_t, \tag{4.2}$$

In Chapter 6 I will show that this bears a striking resemblance to the formulations of state space models. We can already see that these equations endow the network with a sense of its *state* (in the form of the x matrix), a fact that will come to be the most important aspect of resevoirs, and why I choose to include this as a feature of the proposed *Maelstrom* paradigm.

5 Neural Networks and Memory

(i) Omit perception: the system is incapable of representing internally environmental regularities.
(ii) Omit memory: the system has only throughput. (iii) Omit prediction, i.e., the faculty of drawing inferences: perception degenerates to sensation, and memory to recording.

Heinz von Foerster
Understanding Understanding: Essays on Cybernetics and Cognition

Now that we have gone over the fundamentals of neural networks as machine learning pattern recognition models, we can delve into their relationship with sequence memory. The origins of modern neural networks, starting with the Perceptron [Rosenblatt, 1958], were built on a feed-forward architecture, although Rosenblatt himself is famously misquoted in this regard, and was actually keenly interested in recurrent connections (he claims just lacked the computing power). And, critically as I have described earlier, Rosenblatt was actually interested in deriving the functions for memory in the brain, from which the learnable parameters of the network arose. Importantly, this memory is of the uncorrelated data, and not specifically sequence memory. From the viewpoint of integrating some form of non-sequence memory into a neural network, Rosenblatt was entirely successful: the parameters of the network were imbued with the memory of the training data it had seen in the past, and the memory is not a mere "recording" as these parameters also served in the pursuit of *action*, namely that the network outputs a target mapping, conditioned on the learned memory from the past. From this perspective, can we claim this a resounding success and resign ourselves from further pursuit of memory? Absolutely not, I will argue, and hopefully I have made my reasons clear by now. These networks are missing the sense of sequence memory; these networks are critically missing a form of *state*, or a sense of self, of the network, that keeps some value from previous *passes* of the network, in order to preserve sequence memory. Without state, the network simply continues to be a mathematical function with no notion of the sequence it is presented. The parameters of this function can be learned, certainly, and this can be seen as a form of long term memory; in fact, I will argue that this is one form of long term memory, or *implicit* memory.[10] From the viewpoint of the different structures of the brain, all of the modern feed-forward networks are forms of implicit memory, they require no rumination or contemplation nor conscious awareness, no association with other information or sequential processing, and no ephemeral feeling of time's passing, they simply (in a complex fashion) map an input *instantly* to an output. I believe this is what von Foerster referred to in [Von Foerster and von Foerster, 2003] as the system

[10] As I argue, implicit memory can be seen as a form of non-sequence memory, while sequence memory is explicit.

having *"only throughput"*; he must have been aware that the system had some form of memory in the parameters, but without a state it is just "passing along" its information forward.

Sequence Memory

From this context, it may be useful to delineate then this form of memory, without falling into the trap of the definition of memory. We can say that the feed-forward network has memory (it has the parameters it learned from exposure to static data), but it has no, what I am calling, *sequence memory*. This idea of sequence memory implies two constraints, (1) that the network is running in time continuously, and (2) that the ordering of the data in the sequence matters. These are not terribly harsh constraints, however, as the physical laws of the universe ensure that any data in the real world will naturally oblige. In Figure 5.1 I display the demon plaguing modern feed-forward systems: the idea that the system is *only throughput*, as von Foerster would remark. The system can compute, for a given window of a sequence, a correct labeling. However, the knowledge of the signal at any subsequent step is thrown out, and the system throughputs the next window, outputting a separate label. In his terminology, then, networks must have sequence memory to avoid the trap of throughput. To have sequence memory, the network must have a state.

The Trap of Gradient Learning RNNs - Lack of Dynamical Modulation

Feed-forward networks completely lack sequence memory. The only widely-adopted deep network that can handle sequence memory, in that it has a state, are recurrent neural networks. It may be tempting, given the huge success of deep learning, to simply create a recurrent neural network, and train the weights for the recurrent component, and even the state, via an error-driven mechanism. Indeed this is the method that was adopted for the vast majority of recurrent neural networks, leading to the development of the pinnacle of RNNs in the LSTM [Hochreiter and Schmidhuber, 1997]. Notwithstanding that the LSTM is a remarkable machine in its own right, the fact that it must learn the states via the same gradient learning process as the labels is, in my opinion, its Achilles heel. The fact that the brain has such a rich access to heterogenous learning, different learning for the memories versus tasks (as in the basal ganglia), offers it tremendous flexibility and modularity in training. Given that I am trying to both appeal to the neuroscientists and the artificial ingelligencia, while it is important to note that the training of an RNN with gradients is highly biologically implausible (as it involves not only error on the backpropagation, but now unrolled in time), this actually will not be my main point of contention with it. My main contention will be the limiting factor in overfitting memories, in addition to labels, to the same error signal.

While numerous works have, in recent years starting in the late 1980s to alternatives to using the backpropagation rule [Marschall et al., 2020, Hochreiter and Schmidhuber, 1997, Bellec et al., 2020], the standard and most tried-and-true method is to use a network-unrolling-in-time algorithm, which converts the RNN into a deep network of depth equivalent to the number of timesteps. Error is propagated from a training set, across time, to a set of fixed weights, which are then frozen during evaluation. However, this set of "learned weights" is an averaged approximation of the optimal values for all of the time series, across all time steps. Naturally, one may ask: what happens if different weights are needed for different time steps? The nature of time series, after all, is that the data itself changes! Unlike in static images, where the data is fixed. And this data is highly correlated across timesteps, with the sequence having internal temporal patterns that define itself. Transformers [Vaswani et al., 2017], however, do something a bit differently: they learn a special attention weight for each element of the sequence. This was in fact originally done with a stateful RNN in [Bahdanau et al., 2016], which led to the development of the Transformer with the attention matrix. However, that network still learned features in time using the error from the task - it's fatal trap and flaw, I believe. This explanation fits the observations such far that feed-forward networks that flatten the time dimension, such as transformers, have been far outperforming their recurrent counterparts: I believe this is completely due to the fact that the weights overfit to an average of the sequence, and do *not* allow for dynamic re-altering of the weights during run-time. As such, this learned set of weights is *as good* as a random matrix that projects the data. The task of learning the features of this "fixed" set of weights falls onto the readout - the neural network that is tasked with observing the features of the recurrent component, not the actual computations of the recurrent component itself.

This would be akin, in the case of 2-D images, to learning a single 1-d vector of weights, and passing this across the image. Even though the image changes throughout, this single vector must (impossibly) learn an

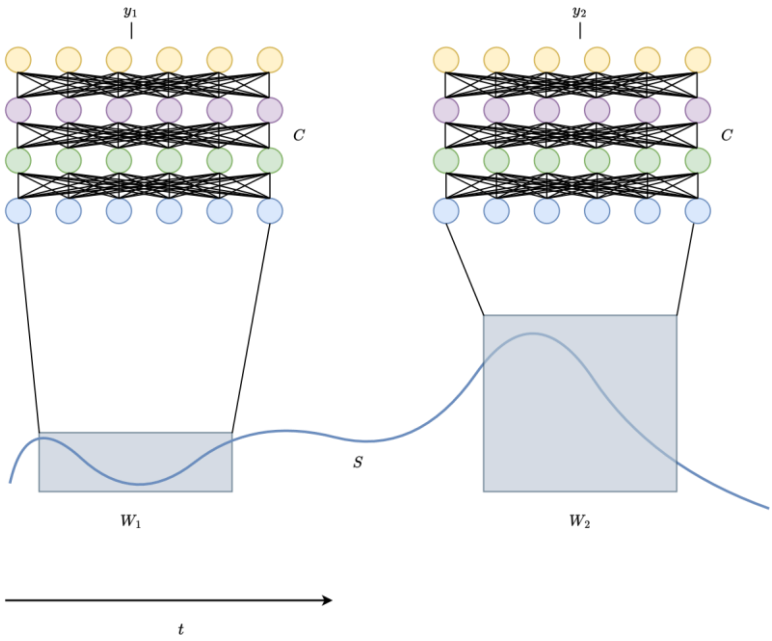


Figure 5.1: The current setup for all state of the art neural networks without a memory component. For illustrative purposes, I portray a memoryless neural network C as a feed-forward MLP, but it may also be a Transformer or CNN. When the network is tasked with learning a signal S in time, there is no memory component, and networks deal with the temporal component by means of *windowing*: the feed-forward network (Transformer, CNN) [Vaswani et al., 2017, LeCun et al., 1989] is fed a small snippet that is passed forward through the network. The only way to make this snippet, or window, 'see' the entire sequence is to make the input size large enough. These snippets are also referred to as 'context windows', in the context of Transformers and Large Language Models (LLMs). In LLMs, the focus on ignoring memory has led to an ever-increasing race to ramp up the size of the context window, although newer works are utilizing a "compressive memory", which in some sense takes us back to gradient-learned memory as in LSTMs [Munkhdalai et al., 2024, Dai et al., 2019]. As the network moves along the signal, it is first fed in a window, W_1, it makes a prediction, then moves on to W_2, and so forth, each window produces its own target output, labeled here as y. Because of the fixed structure of the network, the width of the windows must be the same. When W_2 is fed into the network, it contains no information about the first window from this current run - i.e, it contains no *working memory*. It perhaps contains *long term memory* in the form of weight updates, from being trained on a dataset that contains windows from all locations, but it does not contain memory of this specific W_1. To not keep a working memory is to toss valuable information for the task.

average of all of the possible outputs - which it theoretically cannot do. Take for example, an edge that shifts from white to black, and black to white, after a few pixels. A single slice of weights could not learn this temporal shift, only the shift along the other axis. See Fig. 5.2 for a visual depiction of this phenomenon. What is needed, then, is a system that is capable of *online control* of its own weights for the temporal component.

A path out of the trap is to notice that we need not, and should not, combine the feature learning for the state vectors with the error learning from the target. This will lead us in the next section to my final proposal of this thesis, the *Maelstrom Networks* paradigm, in the coming sections.

Current State of the Art Networks Ignore (Independent) Sequence Memory as well as an Executive

As I have pointed out in prior sections, the human brain is highly topologically *heterogenous*. While the cortex may appear to function more as a self-organizing mass, which is roughly what we model neural networks from, the rest of the brain is highly structured, with an inductive prior or prior distribution of connections highly driven by specific evolutionary pressures and needs, as well as functions. In deep learning, the network

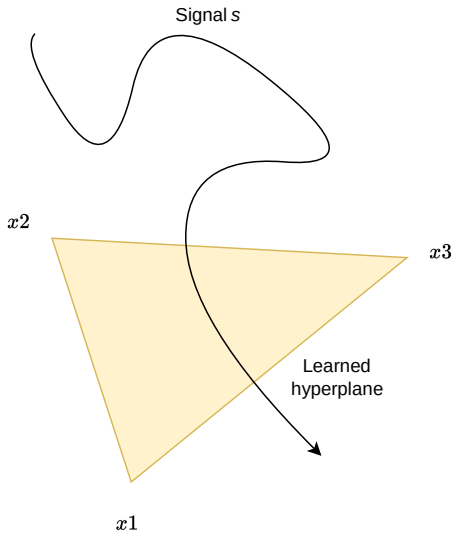


Figure 5.2: The problem with gradient-trained RNNs (here, a single layer example) is that the learned hyperplane of the weights is shared across all timesteps. The hyperplane is passed (screened, in a sense) across the signal *s*, and uses the same learned weights for each timestep, where in reality the curve has very different features depending on which timestep it is. For example, the frequency may be very high in one section of the signal, and very low in the next section, but the weights would not reflect this - they would rather take an average of the frequencies. I would argue here that this is one of the contributions of multiheaded attention transformer networks: they allow a per-timestep gating mechanism that allows the network to learn different weights for different timesteps, as gated by the inter-element relationships. This is in contrast to the sequence wide weights of LSTMs and RNNs. However, this is not enough, as shown by the fact that the original attention networks were actually LSTM networks with attention, and were outperformed by Transformer networks, which just involved removing the LSTM component.

is completely *homogenous*. The network has the same activation function, the same learning rate, the same neuron type, and for feed-forward networks the same information flow, across the entire network. There is no regulatory network, and no dedicated network for memory, association, or executive regulation. This structure, I will argue, is a key reverse-engineering point that we need now consider in advancing connectionist A.I. to the next phase. The essential characteristic that causes failure in a homogenous system, is what I would refer to as “tunnel vision” towards a single loss function, causing the entire network to hyper-fixate on a specific task, which reduces generalization, and degenerates the system to a mere pattern recognizer for a specific type of objects. Large Language Models have done a tremendous amount of work towards extracting the maximum amount of processing from implicit, feed-forward pattern recognition that it is simply remarkable. However, when Large Language Models make mistakes, it is apparent that they are simply implicit, throughput statistical models, and that their “memory” really relies on the vast training corpus that sustains it, not on some more intelligent topological organization of the system. While this is not true for some newer works that incorporate memory such as in

[Munkhdalai et al., 2024, Dai et al., 2019], this in my view tends to slide us back towards the original work on attention, which incorporated a gradient-learned state (LSTM) with an attention mechanism [Bahdanau et al., 2016]; the boost we observe from removing the recurrent component may just be the result of finding better hyperparameters, or better datasets, or removing the memory-to-memory connections (recurrent weight matrix) in favor of more of a "bucket" design, that simply throws memory into a compressed zone . One potential way to give a state to the system would be to have a separate memory module connected, which itself is a state of the system, running in time; something like this would be akin to having a running reservoir that sits next to other systems. This idea will be the key to understanding Chapter 7.

While memory is the main focus of this thesis, future directions will also incorporate how this relates to the executive. And with the executive, we will gain understanding of sentience and what it means for a machine to be self-aware, this I am sure of.

Memory Must be Distinct Learning or Associate Process Separate from Error Learning

Because we do not want the error learning process to dominate the memory learning, and have a situation where the state is being trained on the same error signal as the task, we naturally need to find a topology that separates out the sequence memory process, from the error learning. This sequence memory will be accessed via an executive which makes the final decision to act (the "free will" in the scenario), and the accessing of the memory thus becomes a separate act from accessing the implicitly-learned function mapped values. Figure 5.3 displays a diagram of the relationship between the executive, the implicit processing network (error-learning network), and the associative memory. Since the agent is living and acting in the real world, the output of its action feeds back into the input of the action at the next time step, creating a closed-loop control system which I will discuss in Chapter 6. While this does not offer a solution yet to how to integrate these, it offers a roadmap towards a class of networks that have segmented topologies. One possible instantiation is the *Maelstrom* paradigm, which I discuss in Chapter 7.

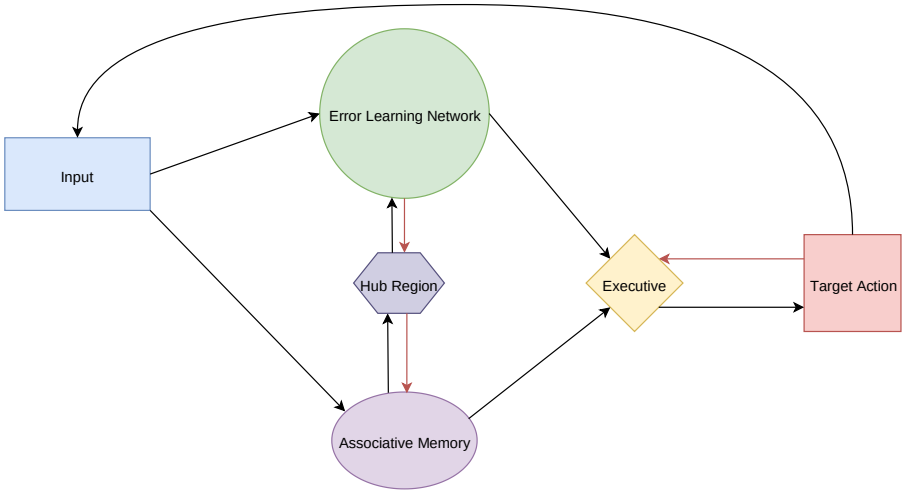


Figure 5.3: A higher-level diagram of the notion of both the separate learning paradigm for associative sequential memory, as well as an executive that can decide to access associative memory, and its integration with the implicit learned mappings. How this executive can elegantly fit into this process is an ongoing endeavor and will be left to future work, as I focus here on the memory module. The output from the action feeds back into the input, creating a control loop. The error-learning network is driven by a learning signal such as a dopaminergic projection, whereas the associative memory is learned via self-organizing methods, which can be guided by the error via an intermediary function, which allows for cross-talk but preserves independence of learning mechanisms.

Latent Learning

It can be argued that neural networks owe their learning capacity to their ability to create *representations* of the incoming data, representations that are ideally of a lower dimension than the original data. Some information theory-centric views of neural networks view this process as a natural result of the network's attempt to reduce its predictive error, per layer, by increasing mutual information while at the same time reducing the size of the encoding [Tishby et al., 2000], a theory called the Information Bottleneck. This is evident in the expressive power of neural networks that are underparameterized with respect to the training data, such as the case of modern deep learning systems [Alom et al., 2018] being able to learn huge datasets consisting of billions of pixels, with a deep network model of significantly less parameters. The neural network is able to approximate more data samples than it has parameters (underparameterized) due to the network's ability to learn latent representations of the underlying data - again, in the unsorted paradigm of data, with notions of temporal sequences missing. One could argue that any deep feed-forward neural network, as a function mapper, performs these two tasks: mapping stimulus to a lower-dimensional representation, and mapping the lower dimensional representation to a target. This representation learning is fundamental to any and all successful and widespread deep learning models in adoption: CNNs and Transformers (as other models such as Diffusion are based on these components). Of course, this setup gets increasingly more complex when you add recurrence and state: now the network is tasked with also learning a state, which, based on these two core tasks, is not part of its "skill set", so to speak. Gradient-trained RNNs such as LSTMs are not uniquely tasked with learning a latent representation of the data, as they must *also* learn to control gates which control the flow of information through time, which may be the root for some of their inconsistency with learning, and somewhat unstable learning mechanisms. This inability of the network to learn state alongside the label mapping is, as I have tried to argue, a core weakness of LSTMs, and requires us to think of new ways to remember stateful information. This, then, reduces the network's ability to learn sequence memory, as defined earlier - to learn sequence memory, we must have a way to save state across a temporal run. One promising direction to learning stateful sequence memory, however, is the idea that we can save these latent representations, and store them elsewhere. This fulfills my core request, in addition, that sequence memory be a separate unique facility from error learning. In the next section, I will delve briefly into one promising area, disentanglement, that can potentially optimally save these latents as a separate process, and could be one potential avenue for the implementation of the maelstrom.

Disentanglement

This latent-encoding property is also exemplified in the field of *disentanglement*, which aims to learn using neural networks *unique, separable, independent, and interpretable* underlying factors of variation in the data. These factors act like "sliding bars" that a person can adjust, and given any combination, should expect to see a sample that exists in the dataset. For example, if the data consisted of an image of a baseball, the factors of variation might be its size, position in the image (*x,y*), rotation, color, and shape. These factors are uniquely, given a value for each factor, able to reconstruct (using a decoder) the original image. This is a property of the *dataset* as well: given other instantiated values of the factors, we are able to generate *any* of the data samples within the dataset. The field of neural disentanglement takes advantage of the properties of *autoencoding* or *encoder-decoder* networks, which bottleneck the data to a smaller representational space, and use the networks own affinity to finding latent representations to force it to learn the optimal encoding of the data in that smaller latent, which generally does *not* actually coincide with a well-disentangled latent, but can approach one with appropriate constraints, some of which I address in my joint work on disentanglement [Patil et al., 2022, Patil et al., 2023]. This is of utmost importance for explainable AI, as truly disentangled representations would give us interpretable encodings that give us some idea of what the neural network is learning. It is the position of those in this field that disentangled representations also offer the most optimal latent encoding for a neural network to learn downstream tasks as well.

These disentangled representations can also, then, be used as a form of sequence memory, as the representations are separated and distinct from the error learning process if they are preserved and extracted from the network. In particular, this is the case when certain external memories are learned, such as in the case of clusters or prototypes (See Section **??** for ProtoVAE [Patil et al., 2023]). These "saved" disentangled representation persist across runs, allowing the network to remember past inputs; this is exactly what is needed for sequence

memory. This also offers a unique way of learning to detect new or unseen data: new data will have factors that do not fit into the learned factors, and thus a process of *continual* disentanglement must be performed to learn new factors in an online fashion as data arrives. Another interesting question with respect to disentanglement is, what happens when the data is temporal (as is most data in the real world)? What do temporal-disentanglement features look like, are they related to motion? This is a question I will be touching on in future work, where I will investigate what disentanglement can offer the maelstrom paradigm.

6 Embodiment and Memory as Contol

As I discussed in the prior sections, the notion of a state in the network is not one that we can simply disregard or use as a temporary nicety in designing networks. The state is critical in not only enabling a system to have sequential memory, but also endowing it with a sense of self for executive decisions, and eventual consciousness. Although not listed as a class of networks in their own rights, there do exist current models that encompass the ideas of state, time, and memory, albeit in a more ad hoc manner. In this chapter, I will introduce these models and connect them as one unifying class of embodied memory model. I will make connections between the idea of embodiment, time, and memory, and models that do contain a sense of state in the world, which inevitably turns the system from a function into one of a control system, or a dynamical system. The network now not only takes in stimulus and processes it from the outside world, but also has an internal state that contributes to its decision making process. These models exist as Reinforcement Learning (RL) and State Space Models (SSMs), and Recurrent Neural Networks (RNNs).

I argue here that stateful memory is, rather than a nicety, the core principle for this set of three disparate learning mechanisms, all with the underlying similarity that they deal with an embodied agent. In the following sections, I will list out these various schemas, and draw their connections to how they incorporate and rely on memory for their sense of embodiment.

State is The Necessary Condition for Embodiment

If you asked a typical artificial intelligence researcher what it meant to be intelligent, you would undoubtedly get some answers regarding an agent moving and taking actions in an environment. This is I would argue the correct view of intelligence, and contains within it the idea that intelligence is *embodied* - i.e., it *must* exist within the confines of a body that *can take actions in the world*. Notably, it is *not* a passive sensor that takes readings and outputs predictions or labels. As I have mentioned in Chapter 5, this idea is also one that Heinz von Foerster referred to as “throughput only” systems. This similar dichotomy of expectations is also present in trying to make a distinction between “artificial intelligence”, and “machine learning”. In fact, I would argue this is *the* key difference between artificial intelligence and machine learning: machine learning exists principally to study passive statistical models, whereas ’true’ artificial intelligence agents require a body.

What does it mean to have a body? From an intuitive sense, we would say that having a body means having limbs, which means having access to taking actions. But people missing their limbs are also embodied, so taking actions in the traditional sense must not be the core requisite for embodiment. I will argue that the core requirement for embodiment is rather having a state. This state also exists to continue the essential essence of the agent through time, such that we can say that we are the same person when we wake up, as when we go to sleep, or even the same person now versus five minutes ago. Systems that have a state have a sense of their own self, as an independent entity, and are not simply “following physics as a servant”, so to speak. Having a state means we

can make predictions on our internal state, rather than the environment, and it is this key defining feature that separates us from automatic creatures, such as insects.

We can attempt to list out some necessary and sufficient conditions for am embodied agent:

The ”agent” must have a state the ”agent” must perceive

features of the environment

The agent must take actions that correspond to altering the environment

The result of these actions must feed back into the state in a closed loop

Without the connections to deep learning, these ideas have begun to be investigated; the field of control theory has been studying these exact kind of agents for many decades, and the notion of feedback loops is integrally connected in the field of Cyberneticism; unfortunately the ideas of embodiment have slowly died out in the era of deep learning, and we are only left with the second point about features of the environment. In the next section I will lay out how these requirements of embodiment can be formally mathematically declared via a state space model. My end goal is to bridge the gap between connectionist models (read: pattern recognizers), and these state space models, and see if there is area to merge the ideas that have already been developed into this idea of a stateful agent running on connectionist pattern recognition as the core learning mechanism.

Connections to Control Theory

Much of what I am discussing can be seen through the lens of control theory, which has a rich history of investigating the relationship between an agent and its environment, and the steps (actions) necessary to maintain stability for the agent as well as guaranteeing it produces the desired output (survives, eats, reproduces). Within control theory, there is a mathematical construct known as the *State Space Model*, which describes a system that contains an input, a memory, and an output, and offers linear solutions to solving for the optimal matrix that keeps the system within bounds [Raibert, 1977]. Figure 6.1 depicts a canonical state space model, where A notably is the memory, which is the key features I have been advocating for thus far, which turns the system from a disembodied throughput function, to system that has its own embodiment.

Control versus Optimization

While this may be a subtle difference, many current theories offer the brain as a ”computational engine”, or an ”optimizer”, optimizing over some quantity, such as Free Energy, or predictive energy, as its main purpose. Most neural network models, especially the recent surge of Large Language Models [Brown et al., 2020], take this view: the ”brain” is simply optimizing over predictive error for a sequence, which lacks a state or feedback loop that sends its previous output back into itself (the latter being less difficult to implement in the large language model framework). When we think of a creature that is alive, we do assume that the agent is attempting to maximize its chance of survival, but it seems odd to argue that the agent is perfectly maximizing its behavior. If the agent is alive, it is alive; it does not need to further be “alive better” than it already is. I believe this idea that the brain is optimizing an absolute quantity does not mesh with current observations of the brain, and offer that instead of optimizing a specific value, the brain is rather *optimizing the values of a control loop*, with the stated goal *not* of optimizing a quantity, but of maintaining a form of *homeostasis*. The idea of the brain as a control loop allows for the possibility that the system (the brain) is not necessarily striving for the globally optimal solution, but is rather attempting to keep its own system stable. Thus, the brain in the control loop is attempting homeostasis, but not a global optimum. This is also the viewpoint of recent alternative theories to brain function as in [Vergara et al., 2019], where the goal of the system is not to maximize a quantity, but rather to make sure it is in an equilibrium state with respect to the environment.

The Brain as a Nonlinear State Space Model

Taking a page from Cyberneticists past, I view the brain as a large nonlinear state space model, where the ”goal” of the brain is to take *actions* that correspond to maintaining *homeostasis* within the environment. This homeostasis can be over a variety of metrics and theories about the brain's “true” target, from energy [Vergara et al., 2019] to predictions [Parr et al., 2022] to reward [Silver et al., 2021]. Whatever the value we choose for the system to optimize, we can in an abstract way write the entire brain in the same formulation as in Figure 6.1, but we can reduce the constraint that the system be linear. We know the brain is a highly chaotic dynamical system, so we can assume for the time being that some unknown nonlinear function can map the values of these states. In the Maelstrom paradigm, I will show that a deep neural network is a good nonlinear mapping function for these states.

A control loop can be defined as simply as a linear state space model, in which there is an input,

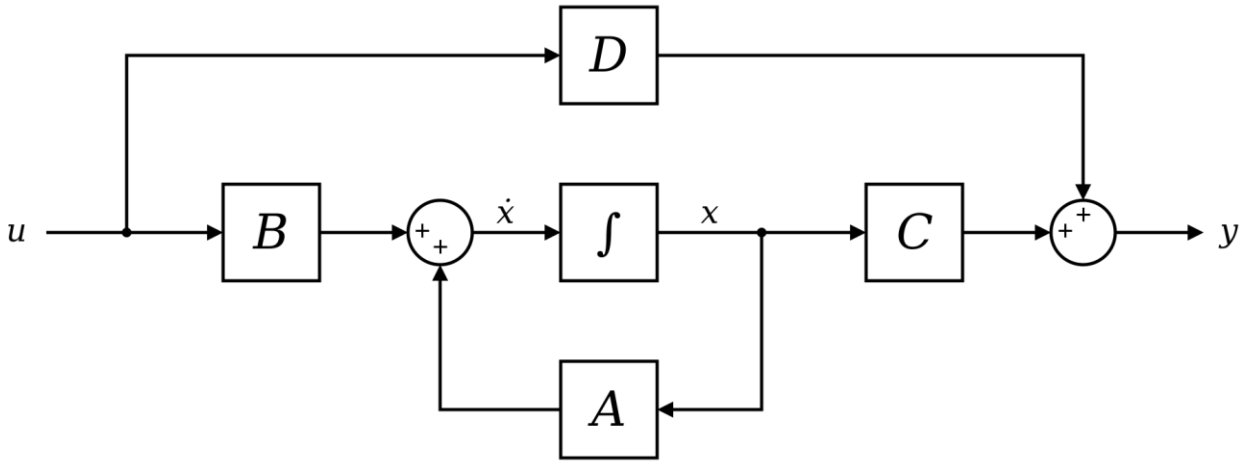


Figure 6.1: A canonical diagram of a linear state space model [Wikipedia contributors, 2024], which is actually the *same exact* formulation as the reservoir update equations. The input u is fed into an input matrix B (similar to the W_{in} matrix from my papers described later), which is summed with feedback from a previous timestep passed through a recurrent matrix A (identical to the W_{rec} weight matrix) This activity is then passed through a readout matrix C, which is used to perform a task. The input can also be passed through a skip connection matrix D, also identical to my work. Thus, in addition, the reservoir updates can be exactly defined as a state space model.

As we can see in Fig. 6.1, the control loop defines an input u, an input weight matrix B, a readout matrix C, a recurrent weight matrix A, and a skip connection D, *exactly the same* as the reservoir update matrix. The fact that the state space model is identical to the reservoir, and yet the fields of control and reservoir computing barely talk, is shocking to say the least.

This *state* is closely related to the idea of *memory*, memory is the agent's model of the state space, up to the current time (in fact, this is referred to as memory in the control literature). This state is represented by a variable x, which I to this point have hopefully convinced the reader of its necessity for the agent's survival in a dynamical, online environment, as well as for its sense of self and embodiment. This state variable represents a compact, ideally optimal, summary of the agent's current place in the world, *given* the perceptions it has made up to this point [Parr et al., 2022]. Formally, this state of the world, x, for a *linear* state space model, is defined as:

$$\dot{x} = A(t)x(t) + B(t)u(t)$$
$$y(t) = C(t)x(t) + D(t)u(t)$$

Here, x is our state variable, A is referred to as the *system matrix*, which dictates how memory is ”shuffled” around, B is the *input matrix*, C is the *output matrix*, and D is the ”feedforward” matrix (See Fig. 6.1). With respect

to reservoirs, there is a close almost 1-to-1 mapping of functions: x is the *reservoir activity*, B is the W_in matrix, C is the *readout*, and D is the skip connection, if necessary. Thus a reservoir model can be seen as a special case of a state space model. As I will expand upon later, this state space model offers a neat, compact formula for describing a generalized family of methods that combine neural networks with a memory module.

We can also look at these equations from an intuitive sense: some network B reads in the features of the world, and learns to incorporate it into a form of *memory* (x). This is analogous to our sensory cortices in the brain. This memory is then updated according to some rule, be it linear or nonlinear (A). This is analogous to the plethora of recurrent connections in the cortex and hippocampus, as well as the cross-region interneurons and other connections. And lastly, C and D are feed-forward networks that take an instantaneous snapshot of the memory and input, and learn a mapping to specific actions to take; this is analogous to the basal ganglia and motor cortex. Combined, these regions explain the non-executive functions of the brain. What is still missing, however, is the executive, as it is in deep learning: a region that can take executive actions and plan on hypotheticals, which is tightly related to notions of consciousness. This framework is, as I would call it, completely *implicit*: given a memory and a state, it "naturally" flows to a specific action taken; there is no room for actions that go against this implicit (sensory-up) processing. This introduction of the executive is left to future work, but is an extremely interesting line of investigation.

Framing this in the current state of the art of neural network research, it is apparent then what components current systems use, and which they don't. For systems like large language models, which do not contain memory, there is no x variable; the system simply updates the input, u, at each timestep, to account for the "memory" of the past. This is what I refer to, when I say that LLMs simply "pass all of time" forward at each "time step": $u(t)$ is updated to account for the new timestep. In contrast, autoregressive models such as LSTM do not augment $u(t)$: they contain a running memory x, which is the state of the model. However, for models such as LSTM, they conflate this memory variable x with the readout mechanism C, which is what I would argue, in addition to the training mechanism, the root of their problem; this idea has also been pointed out in more recent work that attempts to look at memory in neural networks [Gemici et al., 2017]. It is not wise however in this instance to throw the baby out with the bathwater; the idea of a memory mechanism is extremely important. Figure 6.2 depicts the two variants of neural network design. As we can see, the variant with a memory is the only one suitable for *embodied learning*:

However, because it is clear the brain is highly nonlinear, we can alter the update equations to include an arbitrary nonlinear function:

$$\dot{x} = M(x(t), u(t))$$
$$y(t) = O(x(t), u(t))$$

Where we can think of M as a "memory equation" that is responsible for updating the memory unit x; we can also think of this as a "controller", in the sense of a controller of a plant; and O is a "readout" mechanism, which takes in the current memory (or state) of the system at time t, as well as potentially the input (but not necessarily), and outputs an action to be taken at that time. Here, we can think of M and O as both combinations of deep neural networks. The memory equation contains not only the updates for the memory, but also what gets added to the memory from the input networks (the sensory regions). With this formulation, it allows us to reuse the power of nonlinear processing with deep learning that we have acquired, but use it in a more structured goal. The networks are not simply reading in the signal at time t and churning out an action. Rather, they are reading the *state of the system itself*, and producing an action. And, we are now taking full advantage of the pattern matching capabilities of the networks, but not on the original data; rather, we are focusing the pattern recognition to learn the highly complex dynamics of the state itself. Baked into this notion of O, we can think of an executive also, that takes into account not just the memory, but also its learned experience and current working memory, and makes an embodied, self-aware decision on its own free will. This notion of an executive adds much more complexity and will be beyond the scope of this thesis; the main point here is that this simple formulation not only allows for a lot of growth, but further gives us hints to how we can implement this in deep neural networks. This formulation will lead us in Chapter 7 to the notion of *Maelstrom Networks*.

Prior work on Deep State Space Models

It is worth noting, to be exhaustive, that in recent years, there has been renewed interest in combining the power of deep learning with "proper" state space models; some examples can be found in [Krishnan et al., 2017, Fraccaro et al., 2017, Rangapuram et al., 2018], and more recently [Lu et al., 2024] in the context of Deep RL. These models typically work by combining a Variational Autoencoder with a Recurrent Neural Network; this VAE addition allows for learning specific state transitions. In [Gu et al., 2021], the authors directly learn the matrices A, B, C, D using deep learning approaches. In the *Maelstrom* case, because we are modeling the system's state itself, we assume the transitions are like those of an RNN: we do not "meta learn" the systems' own transition matrix. We also do not force this linearity on the system, and allow the system to evolve as it sees fit - thus the maelstrom is more truly a non-linear state space, trained via the MIT Learning Rule. Thus, while it is important to note that while I take as inspiration, these are more formal versions of the connection of the brain to a state space model, and thus follow the formal constraints that slightly differ from the way that the brain may be a state space model in an abstract sense. This is an excellent direction, however, because all of these works tend to stick more closely to the original formulation of state space models, they rather than view the system as a form of implicit processing that augments the system, rather than a memory that augments the

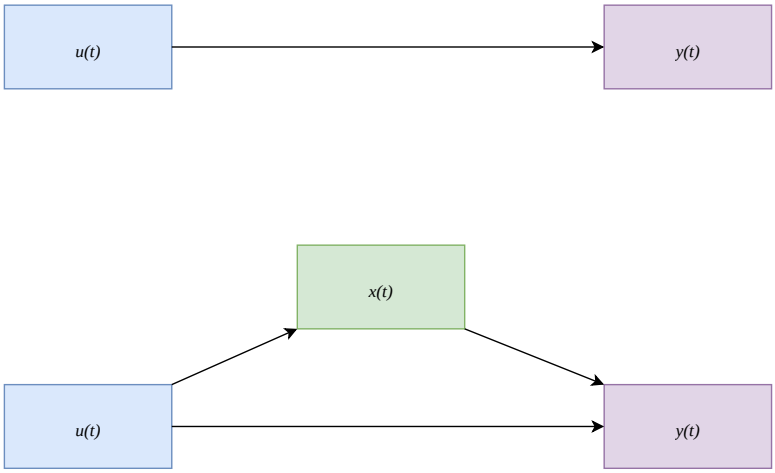


Figure 6.2: The two variants of neural networks: those that contain a memory (bottom) and those that do not (top). Following from the equations for a state space model as described in 6, the top model depicts the state of the art for large language models (LLMs) and other feed-forward (non-recurrent) models. These models contain an input u and an output y, and the model learns a 1-to-1 mapping from these two. In contrast, the memory-augmented models, represented at the bottom, contain both the input u, as well as a memory or state variable x, which is fed into the output network to learn an action or classification. From this, we can see that the current approach for LLMs is to augment $u(t)$ with all of the information from the past, the so-called *context-window*, rather than to integrate a state variable which summarizes this information compactly. Note the similarity to the state space model as in Fig. 6.1.

system. There is certainly interesting overlap between this and the *Maelstrom* paradigm, if there was an interest in figuring out ways to train the Hub interface as in Figure 5.3. The combination of learning the memory with the error from the predictive system, as well as not viewing the memory as a state of the system which is read out by the neural network, makes these slightly philosophically different. However, the idea that the system contains a running memory of its past, and that we can learn parameters that control the system, is one that I will take moving forward. The *Maelstrom* paradigm offers less restrictions, and thus allows slightly more room to grow more complex architectures, ones that may include an executive as well. As is, the Maelstrom paradigm is more focused on the control of a near-chaotic system via feed-forward networks and the fading sequential memory that it produces, than learning the fixed parameters of a set state space model using neural networks. It is also somewhat

similar in instantiation to the newly-discovered Memory-Augmented Neural Networks, to which I took a lot of inspiration in deriving the Maelstrom paradigm [Santoro et al., 2016, Graves et al., 2014]. In fact, these are probably the closest in nature to my proposed work. These networks have an input network, an output network, and a memory, which is also very similar to the state space model. The problem, however, is that the memory is still dealt with as a RAM-like storage, with the input network learning specific rows to store in memory. Here, the authors were more focused on creating memory that persisted *ad infinitum* rather than memory that dealt with fading temporal sequences. This realization that we need not treat memory as RAM, which solves the sequence memory problem but in a way that I do not believe biological agents solve it. Rather, I went the other direction, and asked, what if memory is still a fading trace, would an input controller and output readout still be able to control this system?

Connections to Reinforcement Learning

The idea of the brain as a state space model also has a strong and immediate connection to reinforcement learning (RL), which is also referred to as "optimal control" in the RL literature [Sutton et al., 1999], and was an offshoot of control theory itself, merged with findings from the behavioral sciences. Reinforcement learning deals with an embodied agent's pursuit of maximizing its *reward*, which is a signal sent by the environment (or, potentially, itself) to tell the agent that it did a "good" action, or a good set of actions, toward a terminal *goal* state. Here, "good" is exactly defined as those actions which maximize reward. Thus, unlike a homeostasis, reinforcement learning directly deals with a mathematical optimization. It is to this conclusion that we can wonder if it exactly matches human behavior. In many respects, where there is a clear goal, this paradigm makes perfect sense; however, in life, we have many "goals", and thus mapping all of life to a single goal is a dubious endeavor. Like neural networks, RL also has an original basis in cognitive science; its equations match the "classical conditioning" that experimenters observe in animals, which evolved from Pavlovian response theory.

"Classical" reinforcement learning deals with an agent and its memory of the environment; this memory is a specific kind of memory in that it only remembers to what degree an event or place is "good" or "bad"; it does not remember the sights and sounds. Similar to my endeavor here, "deep" reinforcement learning took classical reinforcement learning and augmented it with deep neural networks. In the same way that you can think of the to-be-described *Maelstrom* paradigm as a deep-learning infused state model, you can also think of it as a deep-learning infused memory model, where we actually want to model an explicit state of an agent, not just its reward model.

All of modern deep reinforcement learning is based on the *Bellman Equations*[Sutton et al., 1999], which itself is based on the framework of *Markov Decision Processes* (MDP), as described previously. The Bellman Equations stipulate that given a MDP, it is possible to derive the optimal *sequence* of actions for an agent, which is learned dynamically in an environment, that places the agent in a resulting final optimal *state* in the environment.

Reinforcement Learning Summarized

The field of Reinforcement Learning is intimately connected to control as well as memory: the goal of RL is to learn an optimal policy over a Markov decision process (MDP), where the process models the states and actions that an agent can take in a world environment, and the *goal* state that the agent is trying to get to. To recall, the MDP contains: A set of world states S, A set of Actions A that are possible to take at each state by the agent, $P_a(s,s')$, the probability of transitioning from state s to state s' for any two states s and s', and $R_a(s,s')$ given action a, the reward the agent gets from transitioning from s to s'. At each state, the a set of actions A lists the actions an agent can take, given the state, to transition to the next state. The task of reinforcement learning is to learn the best mapping of states s to actions a for each state, such that the reward is optimally reached, given that we have incomplete information of the world and only samples of the true reward.

In "classical" RL, the agent stores a reward model (we can think of this as a specific kind of memory) in itself; this model contains a mapping of states that the agent is in, to rewards that the agent got when it was in that state. Figure 6.3 displays the canonical example of the MDP with a grid-world, where each state is a coordinate x,y in a 5x5 grid. For each RL task, we must assign a specific goal to the agent to achieve based on the actions we endow the agent with. In this world, the actions follow the topology of the world, and thus we may only want to traverse

up, down, left, and right. The task of RL is to learn for each state a mapping of the best action to take, given the goal, and the world. The agent explores the world and takes actions, eventually (ideally) leading to the correct state. The grid of the values is then updated backwards to reflect the actions that led to the rewarding state, and the process is then repeated indefinitely - this is referred to as TD - learning [Sutton et al., 1999] . It can be proven that given an infinite number of trials, this process will find the optimal path with probability 1! However, of course, we do not have infinite trials, and we also do not just learn one goal. Still, this is a good formulation for the narrow problem of solving a single goal, and serves as a decent mathematical formulation for the observed

In "deep" RL, we use a formulation called *function approximation* to approximate this table. In some sense we can think of this function as learning the mapping that the table gives us in discrete terms, but in a continuous output from a nonlinear function. In deep RL we approximate this function using a deep neural network, and we use as input to the network features of the environment. We do this from a biological inspiration that we do not have direct access to the true state of the world, and must "guess" our state given the features we can sense. Thus, for example, a deep RL pipeline may be to take in an image from a robot (the features of the world), feed this image through a deep CNN, and this outputs a real-value corresponding to the value for the "state" the network thinks the agent is in. The neural network effectively skips the table step, to directly map the input features to a reward value. We can then use this network in the same way to update the parameters of the network.

RL Definition and the Explicit/Implicit Split

Intuitively, reinforcement learning contains two elements: an *implicit memory* element, where the input stimulus must be mapped to some reaction, and an *explicit* or *episodic* element, where the agent must construct a memory of a world that it previously experienced, and access this memory. This learning follows, in the same footsteps as deep learning, the implicit memory paradigm: the actions of the agent are completely implicit, in that they are stimulus driven, with the slight augmentation of the "memory" of the value of the world. For a given state, dictated by the stimulus, the agent will always deterministically output the same action. The agent is thus, once again, a servant of the laws of the universe. This is not all bad however, as we are slowly making progress with this method towards an embodied agent: there exists a state of the world that is stored in the agent, but not a state of the agent. A major difference between the Maelstrom paradigm and deep RL is this fact that in the Maelstrom paradigm as I will describe, the agent itself also has a state of itself, not just a mapping of the world. Notably, the fact that it is possible to model RL completely without a state (so-called *model free* RL) and actor only methods [Williams, 1992] hints that it is not an essential feature of the system, but rather is a helpful tool in solving AI.

This is an apt split for the differences between *explicit* and *implicit* memory that I will use as a foundation for the rest of the work as well. The implicit memory corresponds to immediate actions that do not require access to a chain of previously stored memories: the access is immediate and without pondering. Explicit memory corresponds to the requirement to follow chains of memories and extract those memories, for use by other implicit processes. In this light, one can also view explicit memory as a "chained" version of implicit memory. In the graph viewpoint, explicit memories are like following a path along the graph, whereas implicit memories are like accessing a single node. For the RL view, implicit memory is learning the action space that corresponds to the immediate response of being in the current state, without having to look back and hop through memory connections.

Now, how does this relate to our proposed Maelstrom paradigm? The agent's model of the world, the collection of states, is similar in function to the maelstrom. It preserves some state or memory of the past temporal experiences. This memory is a *disjoint* process from the action process - the actions may look at the state, but they are fundamentally stored in different variables. This is in contrast to LSTMs, where the memory and actions are intertwined. This is a good feature that we will collect and use for the *Maelstrom* paradigm. The stimuli that the agent receives while traversing the environment is a form of temporal signal, which arrives at the agent in order according to the state that the agent is in. This state is preserved through taking the repeated actions to hop from one state to the next. Analogously, the maelstrom represents the same temporal ordering of the signal, and maintains a state of itself as the signal arrives, similar to the MDP saved. In this classical RL MDP formulation, of course, a major difference is the lack of a neural network, as well as the state requirement that it is enumerable.

The maelstrom is similar to the MDP with function approximation, in that it is an evolving state that gives the agent different information with which to access implicitly learned behavior. Here, the implicit behavior of the

action-stimulus mapping in the MDP is analogous to the implicit learning of the maelstrom-readout. However, it is critically different in that (1) it does not have an explicit goal represented in the MDP, which is propagated backward, and (2) the state evolves naturally with the stimulus, rather than forced to change with actions. However, closing the loop with actions is an important next step to incorporate into the maelstrom network framework, although it is not a requirement for the system to function.

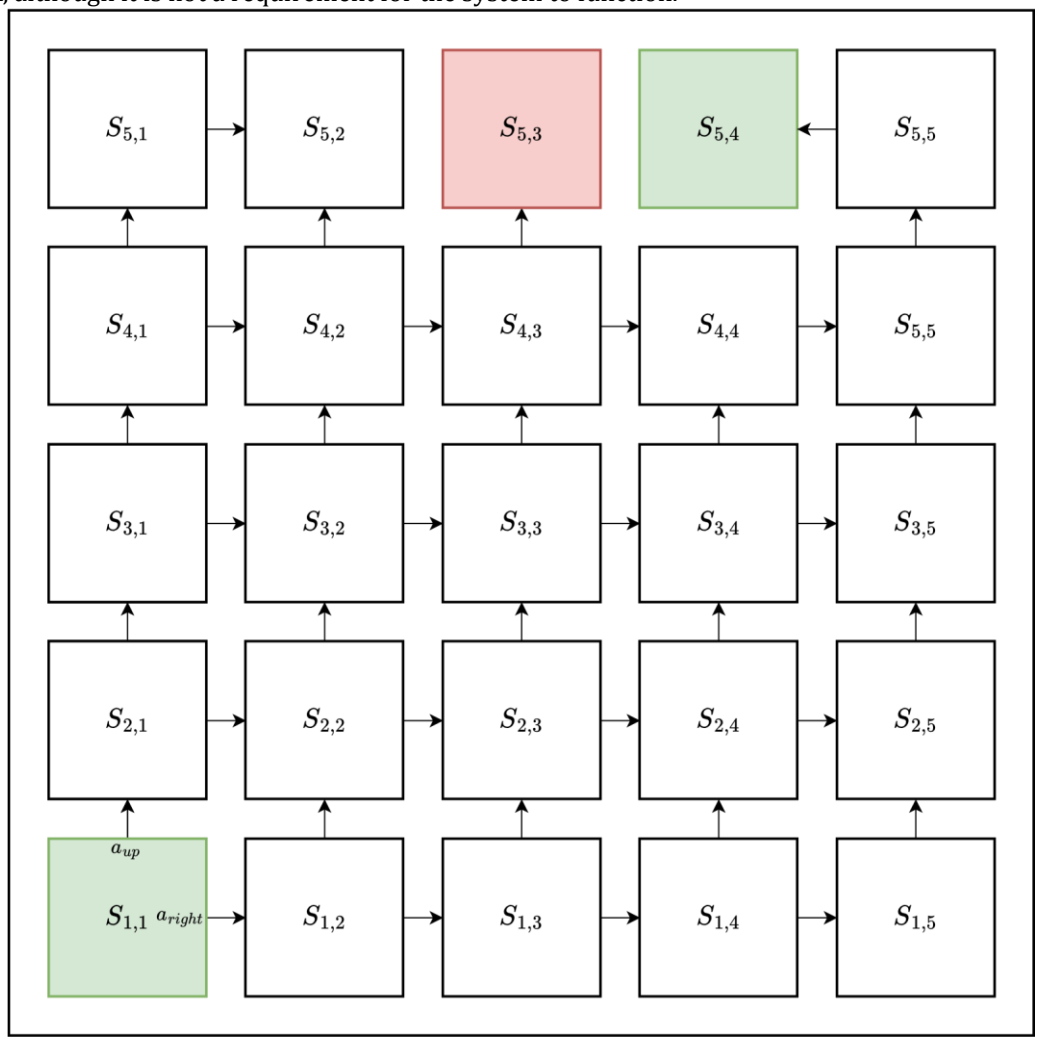


Figure 6.3: A canonical Markov Decision Process (MDP) representing the state space of a simple Reinforcement Learning (RL) agent's world model. In this commonly simplified view, the states $S_{x,y}$ are represented in a "grid world", where *x* represents row and *y* represents column. The columns start at the left and the rows start at bottom. In this world, the only information the agent cares about is its position in the world. One requirement of the RL view is that there must be a *goal* present that the agent is trying to achieve or optimize. There is no such thing as "just existing" in the RL world. The goal here is for the agent to get to state $S_{5,4}$, and the starting state or initial condition is that the agent is in $S_{1,1}$. The agent is able to modify the environment by taking *actions*, which transition the agent from one state to another. In this world, the actions are represented as arrows. For clarity the actions are only labeled for the starting state, with the action of "move up" being the top arrow, and "move right" being the right arrow. Also included is a "bad" state, that has no actions out in $S_{5,3}$, which breaks a symmetry in there being multiple optimal pathways. Here, the agent must learn that it has to go to $S_{5,5}$ and take a left. This reveals another important point about the state: the temporal state is often highly convoluted in state space.

7 The Maelstrom

Assuming I have convinced the reader thus far that sequence memory is critical for the next evolution of connectionist neural networks, that a recurrent component is critical; that current models do not address this shortfall including LSTMs; and that passing the gradient backwards through the recurrent component is an untenable approach to training this recurrent component, I will now shift to my proposal for a new paradigm for training recurrent networks that builds off of the insights of reservoir computing.

The Maelstrom

We now have all of the required components listed: we need to keep track of the sequence memory, we want to reuse the power of non-sequential pattern recognition using deep learning, and we want to avoid backpropagating the error, and we want the memory to be segmented out from the error learning of the network. From neuroscience ,we want to reverse-engineer the concept that stateful temporal sequences are represented as temporal Hebbian cell assemblies, in combination with a readout mechanism [Buzs´aki, 2010]. We want to treat the system as a control system, having feedback from the system potentially feed back into the system, and we want the system to run as a dynamical system in time. Reservoir computing gives us one potential seed to build off of: the fact that we can treat some sort of chaotic activity as its own segmented unit the fact that the reservoir acts as a dynamical system, and the fact that the reservoir system shares correspondences with a state space model. However, we do not need to stop at the basis of reservoir computing. We can take this idea as a base, and build off from there; this leads to the idea of the *Maelstrom*. We Figure 7.1 displays the overall schematic of the Maelstrom idea. It consists of, at a higher level, an input function, which is a non-linear mapping that takes and processes information from the environment (akin to the sensory cortices), a Maelstrom that is the temporal graph as discussed earlier, which collects temporal sequences, an interface that sends and receives from the maelstrom, and an output function, another non-linear mapping function which reads the current dynamical snapshot of the maelstrom and performs some action. Notably missing in this simpler case is the role of the executive; as I have explained, this is left to future but exciting work. As it is, the network is still completely stimulus driven; however, importantly, the network is also driven by the history of its own temporal trace of the stimulus. This can be thought of as the embodiment of the system. The specific topological structure of the maelstrom will lead to a different temporal trace in the graph, and will thus impart a unique characteristic to each maelstrom. Here, we can parameterize these input and output functions using deep learning.

The Maelstrom allows for Embodied Sequence Memory

The introduction of the maelstrom solves multiple “requests” of our embodied agent at the same time. It supplies a *state*, in the form of the activity of the maelstrom at any time, that persists across time in a continuous fashion; this gives the agent equipped with the maelstrom network a sense of continuity and self. Through the state, and the temporal traversals along the graph within the maelstrom (Fig 7.2), it provides the sequence memory that deep learning has so far lacked. It does so, I will note, while offering the full power of the feed-forward networks

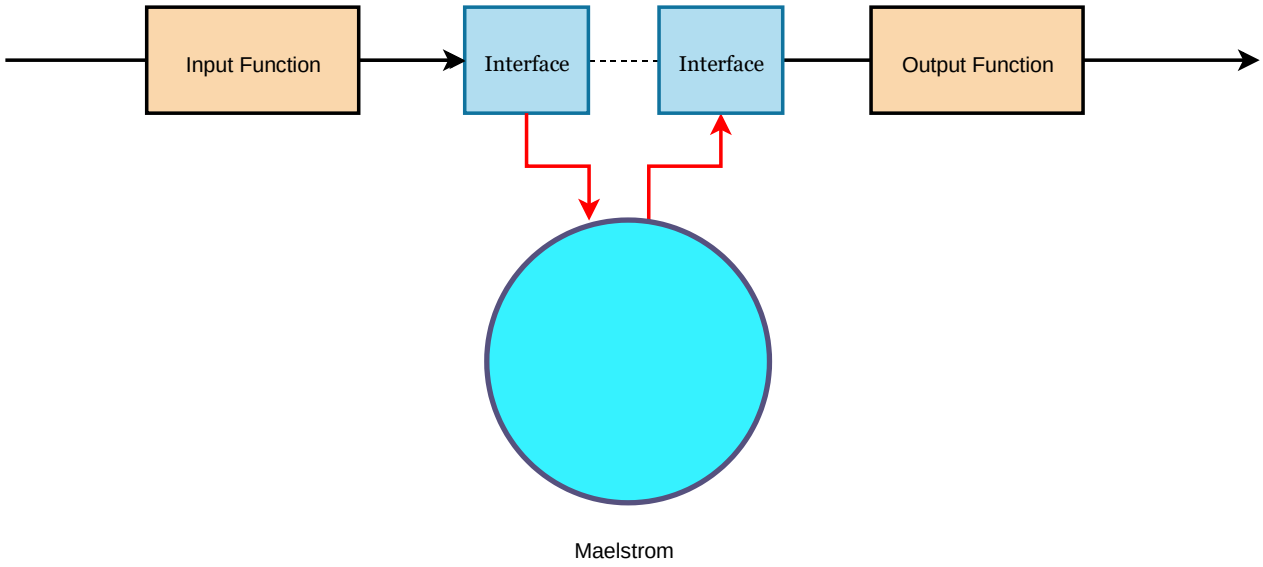


Figure 7.1: A diagram of the proposed *Maelstrom* paradigm for integrating the proposed sequence memory paradigm into connectionist networks. Connections may exist that can guide or apply gain control to the maelstrom, but direct credit assignment or gradient flowback does not occur through the maelstrom. The gradient can only flow backwards through the yellow arrows and the interface. The name *Maelstrom* symbolizes the chaotic, uncontrollable activity in our recurrently connected regions: the maelstrom is a storm of activity that is chaotic and at times random, but always dependent on the input signal, and activity is readout out from the maelstrom and sent to the output nodes. This is similar in philosophy to reservoir computing, where the information flow goes from the input, through the reservoir, to the readout. The name *maelstrom* was chosen to give more clarity to the storm of activity that occurs in the recurrent part, rather than just being a pool of sitting water, the maelstrom is constantly active from inputs, either from itself, other regions, or the stimuli. And unlike reservoir computing, we make no restraints on what the input nor the output can be. For example, the input and output can *both* be learned deep neural networks. The only constraint is that the gradient cannot flow backwards through the maelstrom. This learning rule, where the gradient only flows back through the top arrows and not the recurrent component, is also known as the MIT rule in controls.

at its disposal. For example, in the implementation I provide, the output and input function are both deep transformer models. In learning the specific flows of the maelstrom, the output network takes full advantage of its powerful ability to map patterns, but delegates the role of creating those patterns to the maelstrom! In this way, all components work in harmony towards the embodied state that deep learning needs to move forward.

The Maelstrom Represents Temporally-Activated Hebbian Assemblies

The Maelstrom directly applies principles from neuroscience, namely the idea of temporally-activated sequential Hebbian cell assemblies. These assemblies are then "read-out" by a non-linear pattern mapping [Buzs´aki, 2010], which in the brain is symbolized by the motor cortex, and which here is the readout network. The Maelstrom, as is, represents the collection of assemblies that are topographically locked to be separate from the sensory cortices and the motor cortex. In the brain, this would equate to the PFC, hippocampus, and other higher cortical areas. Thus, its stand-in here as a simple recurrent neural network is merely for proof-of-concept, as the general idea of a Maelstrom can encapsulate much more complex recurrent networks that will be the focus of my future work.

Role of the Error-Learned Deep Networks

One of the benefits of the maelstrom paradigm is that it allows for reuse of deep learning architectures towards a more integrated larger system; without throwing out the innovations that deep learning has achieved, but using the network "to its strengths" as part of a bigger picture. We thus parameterize the non-linear input and

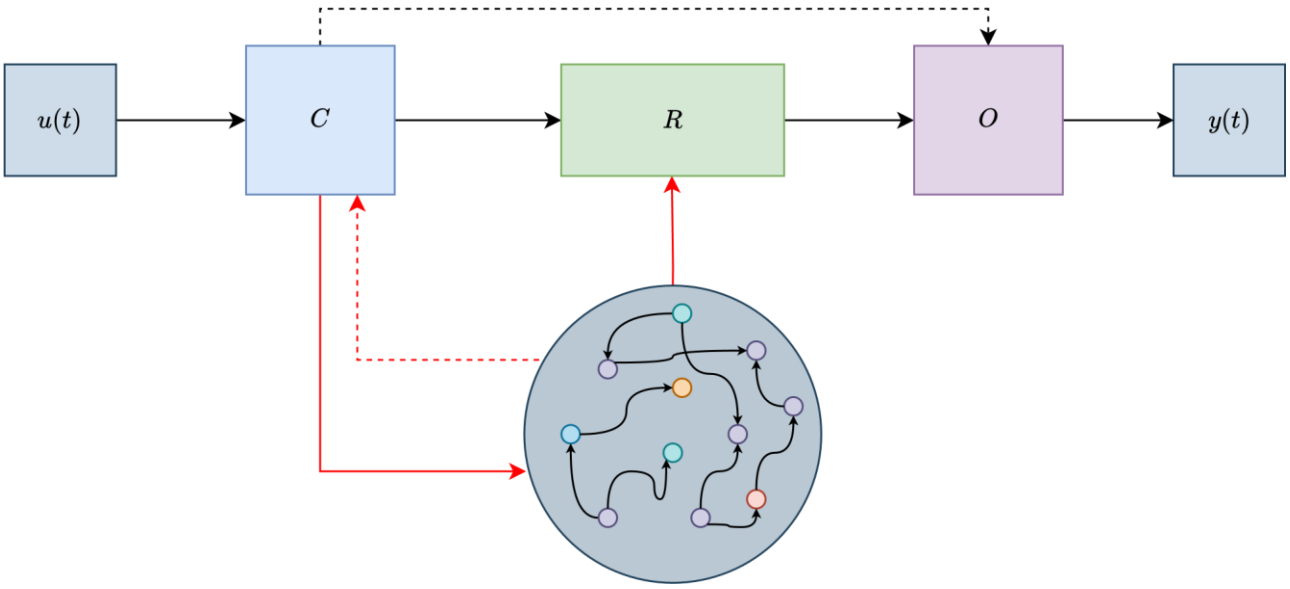


Figure 7.2: A diagram of an instantiation of the maelstrom in a deep neural network. Here, the simplest recurrent network is used as the maelstrom: a randomly initialized recurrent neural network with untrained weights, essentially a reservoir. *C* here is an input deep neural network, a "controller", that decides given the previous maelstrom state (dotted red line), and its input, what the next memory should be. *R* is a readout network that takes information from the maelstrom, this is separate from the classifier *O* network not for this instantiation, but for future more advanced versions that may want to separate this out from the action network. *O* is the final classifier/action network that performs the action given the input. In terms of analogues to the brain, we can think of *C* as the sensory cortices, the maelstrom as the prefrontal cortex, *R* as the entorhinal cortex or another hub area, and *O* as the motor cortex. The solid lines indicate the forward flow of information, the dotted red line is the feed-back from the maelstrom to the controller, and the dotted black line represents a potential skip connection; this turns out, is not necessarily important. The black lines indicate that gradient can pass backwards through those connections, and red means gradient is blocked.

output functions using deep learning. Here I will briefly describe the role and philosophy behind these input and output networks, and how these map to the larger functions of the brain. This is similar in nature, and indeed took inspiration from, Memory Augmented Neural Networks. However, key to the maelstrom network is the fact that the controller is not restricted to the RAM-like storage of the Memory Augmented Networks, and the maelstrom is kept separate as a dynamical system. The insight of the maelstrom network is that yes, the system still works if we let the controller and readout try to "control" a chaotic system.

Readout Network Learns Static Snapshots of the Maelstrom

To reiterate, critically, this network takes full advantage of deep learning *as is*, not to alter it, but to use it as one component of a larger system. The use of feed-forward networks here is the same as it always was: mapping a static pattern to a specific label. Here, however, the static pattern is a snapshot of the dynamical system of the maelstrom at a given time; the mapping is then done between this snapshot

If I were to be asked, "what is the one thing that deep learning does well", I would answer: it learns to map inputs to patterns; it is a pattern recognizing machine. This is at odds with current attempts to forcibly map large language models (which have no memory - they are feed forward networks) to some kind of reasoning or intelligence. It must be said that to have any kind of conscious intelligence, memory is a core requisite. Regardless, deep neural networks have completely excelled at pattern recognition. The tasks that they have overtaken humans on - face detection, game playing - have to do with mapping patterns to outputs (in this case, class labels and actions to take in an Atari video game). The Atari-playing deep RL network [Mnih et al., 2015] is an important connection, as this network bridges function approximation of RL with deep learning (see Chapter 6); this idea of using deep neural networks as pattern recognizers for a larger system is similar to the ideas presented in the Maelstrom paradigm. Because the feed-forward network excels at pattern recognition, it is as easy for the network to pick out the patterns in the raw data as it is to pick out projected patterns in a recurrent Maelstrom. Thus, the job of the deep network is to *continue* to do what it does best - i.e., map patterns to outputs - but to do so on this larger Maelstrom space, which has patterns that are hidden and non-obvious to a human observer. This readout mechanism's analogue to the brain would be the output cortices, such as the motor cortex, which are responsible for taking highly structured sequence memories and outputting a given command. What is missing, to reiterate, is a more complex executive function. However, given the maelstrom structure, this can be easily integrated into the larger "output function".

Input Network learns Static Gain Control of Maelstrom

On the input side, a separate deep network is doing the mapping of the input signal $u(t)$ (see Fig. **??**, and outputs the correct addition to the memory. This can act as a positive gain control to the memory, to keep memory active while processing is required. While there is no explicit mechanism to ensure this as of now, this can be left to future iterations of the network.

Training is done via the MIT Learning Rule

An important question to ask is, if the gradient is not flowing through the reservoir, how are the networks trained? As we can see in Figure 7.1, the gradient does flow through the output function into the input function. This has been a vexing issue for reservoir computing also, as we cannot simply train the input weight matrix. However, for this I note that this kind of problem has been addressed in 1961 with the advent of the *MIT Learning Rule* for control systems [Mareels et al., 1987]. If we view the system as a large nonlinear state space model, we can approximate the true unrolled gradient by just the state at the most recent timestep; the MIT rule itself already performs gradient descent, so we can view the deep networks as performing this step. In this sense, we cut the dependencies on the rest of the temporal chain, and only look at the maelstrom at a given time. The input function is given the gradient with respect to the output function and the maelstrom, at a given time, but this gradient is not propagated back in time through all timesteps. I show in follow-up result papers that this does actually work effectively to propagate the errors to the input function. Note of course that this is the first iteration, and more can be done to improve the system; but these results do prove that the system is capable of approximating the past, owing entirely to the fact that the past is summarized in the maelstrom.

Variants of the Maelstrom Proposed

Random Recurrent Maelstrom

The most direct connection to the work of the reservoir and the maelstrom network, is to use an untrained neural network as the maelstrom. Figure 7.2 demonstrates this instantiation. This has the downside of being more complex to control than the simpler Phasor memory, but has in my view a higher ceiling for potential. This has a lot of flexibility for topology: this maelstrom can be randomly connected and weighted (as in the case of the reservoir), it can be learned via unsupervised means (for example, via STDP or other clustering, or predictive coding). It can also include more detailed and advanced interconnected maelstroms, such as a chain of random recurrent networks with different timescales, or a larger memory complex. Here, we use the random recurrent network as the most simple of these, with the more complex variants left to future work.

Phasor Maelstrom

The second variant uses a memory matrix without interconnections, but with sin activations (Figure 7.3. In the brain, there are two theories to how the neurons can encode information, the *rate encoding* and the *temporal encoding*. Most of modern computational neuroscience is built around the rate encoding paradigm, but it is likely the case that the exact timing of neural firing plays a key role in many mechanisms [Engel et al., 1992]. Mathematically, we can represent the timing of neuron firing as a complex number, with the amplitude representing the strength and the angle representing the *phase*, or the time of arrival offset from a starting reference point. In addition, an issue with the memory unit itself, and common to reservoir computing, is how to normalize the reservoir activity such that it doesn't explode. A common tactic across all reservoir computing is to normalize the spectral radius (the maximum real eigenvalue) of the recurrent weight matrix to be between 0 and 1, such that the activity slowly dies out -this is what creates the "echo" in echo state networks. Here, inspired

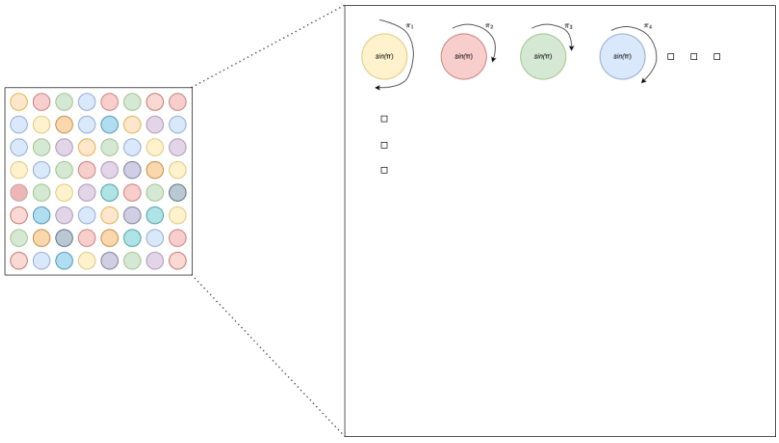


Figure 7.3: The proposed Phasor Memory schema that prevents value explosion and ensures memory stability. This memory is different in that it does not have a dynamical connection between neurons, but rather the trace of the memory is entirely contained within the resonance of each neuron. As such, the phasor memory is a much simpler variant. Shown on the left is a color depiction of the memory M, where each shade of color represents different values stored at each index. Zooming into this, we can see that the values stored are actually the phase (sin) of the absolute value (representing the angle π). Given that each function approximator is a powerful neural network, it is capable of learning hidden patterns in any medium. Therefore, it is no different a task for it to learn the patterns in the phases of the memory versus the absolute value.

by this phase angle of the complex number representing temporal coding, we take a new approach: rather than store memories in the memory matrix *as is*, we store them as the $sin(\cdot)$ of their input values. This causes the values themselves to spin around the unit circle, in effect turning memory into a large phase-bank of spinning knobs. We call this addition *phasor memory*, and demonstrate that its inclusion does indeed completely negate the need for any kind of negative weights or eigenvalue normalization. In addition to the reservoir memory, this is one possible variant of the network. The benefits of the phasor network is that we don't need to worry about the decay rate of the reservoir nor the spectral radius, however, the downside is that we lose the graph's ability to communicate with itself. To be certain, one of the inconsistencies with echo state networks is that they are fundamentally build upon the rate encoding framework (by including a squashing function and sum), but work in the temporal domain; this amounts to not changing activities over time, but rather change of rate of firing over time, an issue with LSTMS as well. This phase array also serves the purpose of aligning with the inherent rate encoding of the reservoir activation as it stands. The phasor memory is meant to be seen as a simple starting point to experiment with the convergence of the outer networks.

8 A Memory Theory for Stateful Brain Organization

While not the central point of the thesis, which focuses on the implementation of these in artificial systems, I believe the ideas flow backwards, and here will attempt to show one rough sketch of how this could inform investigations back into neuroscience. I will attempt to bind together the various thoughts on memory in the brain presented here, and present a theory for how this could exist in the brain. In doing so, I will offer the zoom level that I believe to be most relevant in reverse-engineering the brain for artificial intelligence applications, and how memory and the executive play a critical role. The main distinctive features are the interplay between memory, explicit and implicit, and the role of the executive in initiating sequence memories.

In this view, this controller selectively activates, or 'zaps', target memory engrams encoded in the cortex and hippocampus, and selects them in particular orders. The focus of attention to the selective activation is what gives us a sense of control over our conscious thoughts. This attention could be implemented as a gain control. And if you subscribe to Active Inference, as Frison says, this attention mechanism could be implemented as a gain on the precision of accuracy of measurements.

It is difficult here, then, to avoid opening the bag of worms slightly into the philosophy of consciousness, and posit a hypothesis for consciousness that fits in with this: *conscious awareness is the feeling that we get, as our controller (executive) implements the closed loop of active maintenance of working memory, or activates the memory chains.* This also has a fundamental consequence for our notions of free will and personhood in the legal system. Since Plato we have struggled with the idea that humans are half person half beast, always fighting the natural urges while attempting to rise above them. In this account, the control/executive accounts for the "control", whereas the implicit neuronal reactions based on the reward system account for the "nature" responses. Having a strong control over ones faculty, in the stoic sense, thus amounts to a strong controller.
The

The Right Zoom Level: Computational Principle of the Brain

Figure 8.1 displays my proposed zoom level for correct reverse-engineering of the brain. The correct zoom level is one which focuses on the core computational principles of intelligence and consciousness, rather than implementation or effect. At the lowest levels, while the ion channels and neurotransmitters are critical for implementation of the circuits in the brain, they do not get at the core computational principles that will aid us in discovering what makes the brain function. The learning rules, similarly, are a bit closer to this, and while they can offer what I will say are *clues* towards the solution, they are not themselves critical in discovering the correct solution. Similarly, on the other end, the cognitive output of the system gives us our desired product that we wish to achieve, and thus offers clues towards what kinds of behaviors we would like, but does not guide us toward a solution. My refusal to stick to this end is highly influenced by the failures of the logical AI approach that led to the AI winter in the 1960s and 1970s. The logical and classical AI were fully supported in their endeavors by wanting to recreate the effects of the cognitive system (i.e., they believed that the correct zoom level was the Cognitive one), however, just modeling the effects as the right zoom level, rather than simply an area for inspiration, turned out to be disastrous, as one can derive multiple different computationally elegant but poor models for emergent behavior and underlying core computation. The "core", I will argue, is something I have touched on before, which is that the network or graph of the simple units, and their assembly, organization, and topology, is the fundamental zoom that we need focus on. This guiding principle led me away from focusing on the debates between spikes and non-spikes, as these are more learning rule focused, but rather on the recurrent cycles that constitute the topology

of the network. These cycles, then, naturally lead to a notion of state, and this state leads to memory. I have attempted to color code these as gold, silver, and bronze, according to rankings of the importance. The flow from cognitive to the cell assembly is likely the most important, followed by the flow of learning rules clues' towards the assembly. This coincidentally follows a "middle path" approach that I believe well-suits many aspects of life that originated in Buddhism.

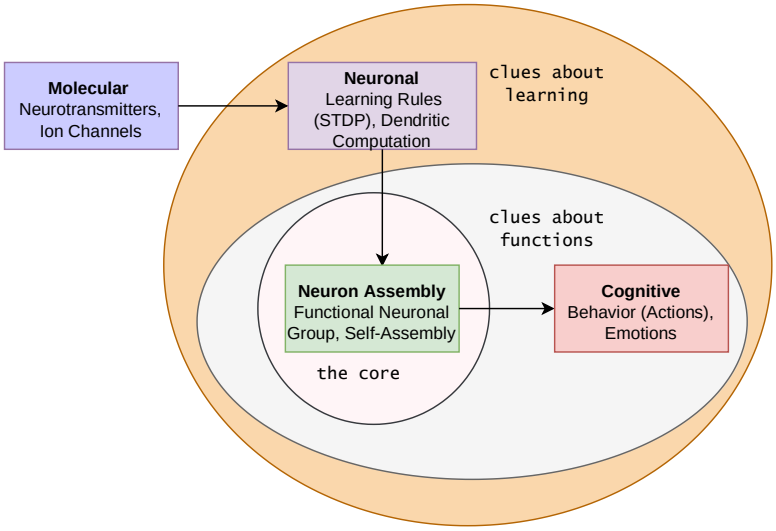


Figure 8.1: My proposed "correct zoom level" for reverse-engineering the brain which focuses on arriving at the core computational principles. The lowest levels of the glia and neurotransmitters offer no information about the system's recreation. The highest cognitive level offers clues for directions, but not for underlying computational principles. And the learning rules, similarly, offer clues for biological implementation, yet not computational principles. The only zoom level that offers the core computational principles, in line with my proposals for what actually drives the intelligent behavior, is at the cell assembly and graph, or topology, of neural connections. Through these topologies, we can derive not only principles for computation, but also principles for memory, state, and consciousness. The gold, silver, and bronze represent the ranked importance of the clues, with the gold being the core and most important.

Executive Controller and Cell Assemblies

The core two features of my account are the *executive controller* and the *sequence memory*, which I have spent the majority of the thesis defending and proposing an implementation of in artificial systems. The executive controller performs the so-called "executive functions" [Gilbert and Burgess, 2008] of the brain, which are involved with planning, correcting erroneous decisions, attention, and working memory. Importantly, executive functions are completely lacking in deep learning; the system is entirely implicit, with no active maintenance or decision making; the words that are espoused from a large language model are no different from the activity outputted by a reflex.[11] Of these, I believe that attention is the most critical, as it is this attentive state that we most associate intuitively with the phenomenological experience of consciousness. The sequence memory is, in my account, the core building block of other memories in the brain; this is not an outrageous statement, as it is clear the brain receives temporal signals, however this account I have not seen in prior work. In detailing these two mechanisms and their role in the brain, first I need to make a new distinction between explicit processing and implicit processing, which I argue is the core "split" between conscious and unconscious behavior.

[11] This is the main reason why I do not believe LLMs are actually "thinking" in any sense.

Assembly Resonant Memory: The Memory of Assemblies

At the lowest level of memory in the brain occurs at what I will call "resonant" memories, where the memory is resonant activity bouncing around a neural assembly. This builds directly off of the ideas of Hebb [Hebb, 2005] towards viewing neural assemblies as the building block of all memories; I simply am renaming these resonating assemblies as the core building block. This does not offer a new theory for how neurons operate. What this does is offer a reframing of the pipeline from stimulus to memory, and how the machinery of the brain can organize this activity towards stateful processing. It is known that the hippocampus receives temporal activity; what is less known is how this is done, or the structures and logic involved [Eichenbaum, 2013]. It is also hypothesized that the hippocampus is responsible for temporal cell assembly activation [Dragoi, 2020]; thus, this notion of temporal maintenance of cell assembly resonances is critical to understanding how the brain might be storing sequential memories.

This assembly originally is activated by spontaneous activity arising from the stimulus regions, where the activity is not guided by attention or a conscious executive. Once the executive focuses on this activity, the brain consciously and actively maintains the resonance by feeding in positive activity such that the resonance is maintained; this is the basis for "working memory". This resonance then moves from the cortex into the hippocampus, where it is encoded in unique assemblies there, which we refer to as "short term memory". During sleep, this memory is consolidated back from the hippocampus into the cortex, where it can more likely be triggered as a spontaneous resonance from stimulus in the future. All processing in the brain, thus, results from an initial resonant memory that reverberates within the brain's systems. This memory is analogous to a reservoir, although it likely is highly structured topologically and not a simple random weight matrix as reservoir computing approximates. Working memory, then, is defined in this sense as "maintained resonance memory"; the resonance is actively maintained by some executive process, such that it does not fade. An example of this is repeating a phone number in one's head to remember it; this is an example of conscious maintenance. Unconscious maintenance can involve keeping things in working memory while one is thinking about other tasks, in order to fuse the processing from multiple sources. The idea that the cell assembly "resonates" is also not new, as it was The ideas of sequential associations is also equally not new [Wess and R¨oder, 1977]. What I propose here is simply re-focusing on the resonating assemblies as the core block, and building up from there.

Implicit Memory: Those Processes Stimulus-Driven

Implicit memory, from this account, is processing based on synaptic learning that is completely stimulus driven; it does not involve "contemplation" as such, and occurs naturally the same as a physical processes occurs. An example of an implicit memory is a learned fear response reflex, recognition of a face that is familiar, or a skill that has been learned enough that it passes to lower levels of the brain and away from the working memory. Closely tied here is the idea that these forms of processing do not possess any form of working memory; they do not possess any "resonance" that bounces around. The more that the activity bounces, in this account, the more that the brain "contemplates" it. Resonance within the system in this account is analogous to processing time, as working memory is seen as a maintained resonance. An implicit memory or implicit processing is functionally feed-forward in this regard, and it should not contain loops.

Explicit Memory: Those Processes Driven on the System itself

Explicit memory, in contrast, involves memories and processes that are commenced via an explicit signal from a system that is running "above" the implicit system, which is able to make decisions based on its own set of criteria; usually this is the same criteria as the implicit system, but need not be. This system closely aligns with the "consciousness", the phenomenological feeling that we experience with the system choosing explicit decisions. This notion of explicit memory is unique to higher intellect beings, as is the notion of consciousness which it is tightly entangled with. If the implicit system runs on the stimulus, the explicit system runs on the system's dynamics itself, the resonating cell assemblies that I call Cell Assembly Resonant Memory. This, like the explicit system taking 'advice' from the implicit, runs at the core on implicit signals, but those implicit signals are themselves repeated activations of the brains own assembly resonances. This enables an explanation for how

explicit behavior can arise from implicit behavior: while the underlying machinery is implicit in both cases, the explicit behavior is implicit behavior that has been resonating many times over within the brain's dynamics. This resonance has slowly removed it from the causal influence of the external world, where it has become not external, but internal processing. Thus, the time that it takes for the information to resonate purifies this implicit signal into one that comes from us, internally.

Free Will and Consciousness

How does this dichotomy lead to directions for free will and consciousness? The critical issue in a connectionist account, is how an executive, and the phenomenological experience of consciousness, can arise from an implicit mechanism. At the core of the brain's functions, the entire brain is effectively implicit; the neurons are deterministic machines that respond predictably to the same input. The system is built from these implicit machines, so how is it possible that an executive can arise from a non-executive substrate? If the explicit system arises from the self-loops within the resonating system, then perhaps consciousness is the phenomenological feeling we experience of this transition from stimulus to internal processing. I suggest, then, that the executive functions arise from the implicit processing *exactly because they look onto the system itself, in a form of meta cognition*. **The phenomenological feeling that we get from consciousness, then, is the feeling we get from observing our own active maintenance of the resonances that the world produces**, rather than the observing the stimulus itself. This is, why I would argue, we look inward when contemplating; why we listen to "white noise" to help study, why careful deliberation effectively shuts off the stimulus from the outside world in favor of internal dynamics.

The Interplay Between Regions

Memory as a Separate Module from Task Learning Regions

As I have repeated often, it is critical that the associative memory is removed as a distinct learning mechanism from the error-based learning. A key finding from my investigations and theory thus far is that the different kinds of "memory" must be topologically and functionally separate from one another. The "memory" involved in task learning (error driven, implicit) is fundamentally different from the memory involved in associative learning in the hippocampus (unsupervised, explicit). These separate kinds of memories rest in different locations in the brain, and learn by different mechanisms, which evolved at later points along the phylogenetic tree. This suggests that we need to separate, in artificial intelligence as well, these different components into topologically separate and unique units, and we need to cleanly delineate the roles of each region. For example, we can cleanly delineate that the hippocampus is the self-associating processing region for short-term memories, and we can cleanly delineate that the neocortex is the maelstrom where the storm of activity resides for current awareness and attention.

Figure 8.3 displays the overall proposed brain topology for memory and executive processing. The main contribution is the addition of the *maelstrom* as the form of "resonant memory", which forms the fading trace of activity of the brain, and is the core building block of all other memories, both short and long. Resonant memory is fed into the hippocampus via the entorhinal cortex, where it becomes associated for processing. This activity is then fed back into the cortex during sleep for long-term memory consolidation. The important point is, that today's maelstrom resonance becomes tomorrow's long-term memory. In this way, long-term memories guide our maelstrom's responses in the future. A key distinction is between *implicit* neural learning, which I am associating here with instantaneous, atemporal dopamine-based updates, and *explicit* memories, which consist of sequences of learned experiences, except in the case of the basal ganglia, and which associates more with serotonin. Prior recent work has shown that the prefrontal cortex can be modeled as a reservoir, which is the core recurrent component of the maelstrom paradigm [Enel et al., 2016]. Input at each timestep of the real-world is sent into the brain via the various sensory cortices (visual, auditory, olfactory, taste, spinal cord). This information then flows to the executive, where it is sent to the prefrontal cortex, where it "resonates" around previously learned long-term memories. These resonances are either maintained by the executive via positive feedback for working memory, or are sent to the hippocampus for short-term associative memory. This memory, during sleep [Marshall and Born, 2007], is then consolidated *back* into the prefrontal cortex, where it then becomes more likely to

resonate with the new data, and the process repeats. For motor tasks, this short term memory also flows into the basal ganglia as well as from the cortex [Bolam et al., 2009], where it is associated with rewards and punishments. For motor tasks, this then flows from the basal ganglia into the cerebellum for motor skill hardening [Pelzer et al., 2013]. Omitted for simplicity are the feed-back connections from the controller to the input stimuli regions that aid in guiding the senses to *attend* to important inputs, such as eye saccades and auditory attention in a crowd [Semedo et al., 2022]. This input, implicit learning is *atemporal*, in that it reacts to the current stimulus without recurrent processing. As noted in [Semedo et al., 2022], there appears to be evidence to support this as the known feedback connections to lower levels seem to consist of a different population of neurons, suggesting that this could come from the controller. We also need to take into account the fact that some *sequences of implicit processing* are learned, such as the motor controls required to unconsciously ride a bike, or drive a car. This chained feed-forward also takes its stimulus from the controller (the enacter of tasks), but the dotted line also represents the potential interconnections between the explicit and implicit regions via areas such as the MTL, as described in [Dew and Cabeza, 2011]. This interplay could also be in the controller itself. This learned *chained implicit*, sequential reward-based pathway exists in the basal ganglia, which flows out of the controller (i.e., the ability to ride a bike on demand) which learns to associate the various instantaneous implicit signals. The Basal Ganglia, suspected region involved in reinforcement learning and reward-based learning [Schultz, 2016], projects to the cerebellum [Pelzer et al., 2013], where motor skills are "hardened", i.e., motor memories become more permanent.

Memory Chaining

Lastly, how does the brain actively maintain long term memories? Long term memories from this view, must be "calcified" versions of the sequence memories that we experienced before. It is known that the brain stores temporal memory as sequences of cell assemblies [Dragoi, 2020]. Following this theory of sequence memory, the original memories were temporal sequences of these resonating cell assemblies, as I have been arguing thus far. From our own experience in life and in practice, we know that to retrieve these memories, we must either "activate" a triggering memory or stimulus, such as a smell or sight. The temporal association, thus, with memory remains. What triggers this cascade of cell assemblies, then? In Figure 8.2, I give one possible hypothesis for how this could occur, using the ideas of the executive in combination with these chains of memories.

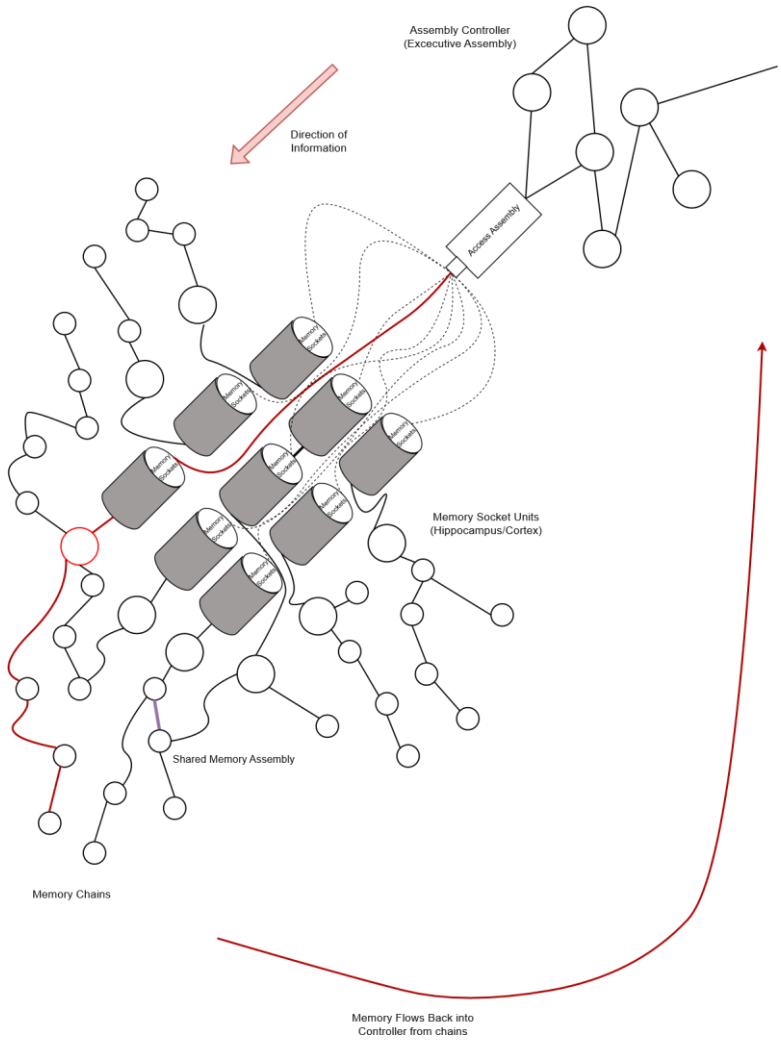


Figure 8.2: A hypothetical architecture that bridges memory chaining, executive control, and the notion of free will - which introduces the idea of the Memory Socket and Assembly Access. When we actively choose to "select" certain memories, i.e., when someone asks us the question "remember when...", our executive/controller module activates a specific pulse that travels through various cell assemblies (located in the cerebral cortex), to a *access assembly*, which is the focal point of access. This access assembly is similar to a read head in CPU architecture. It then sends a pulse down into receiving memory assemblies, which I call *memory sockets*, as they are similar to sockets receiving specific pulses from the read head. In red, I have shown a single chosen memory chain from among all the chains, and shown its potential pathway as it passes through the memory chains and back up. It must return back to the executive controller, as we are conscious of the memory that we retrieve from the access "request". This brings up an interesting philosophical point though, about whether or not we exist in the memory region as well, or just in the executive. If we exist just in the executive, then the memory must pass back up to re-enter the executive. However, if our phenomenological consciousness exists in the memory region as well, it may not need to pass back up, but the loop must occur somewhere. The act of willingly choosing to send a pulse to the memory socket restricts the location of where the "free will' might exist'. Whatever is sending the pulse is then, in the executive, is the agent making a free will decision. Free will is located thus in the agent making the decision to initiate the pulse that passes through the memory sockets if the agent is a free will agent, then free will drives this process; if the agent is a naturally flowing process that involves only expected causal relationships among particles (i.e., a ball rolling down a hill), then free will does not exist. As to where this executive controller sits in the brain is the real question that I am still discovering.

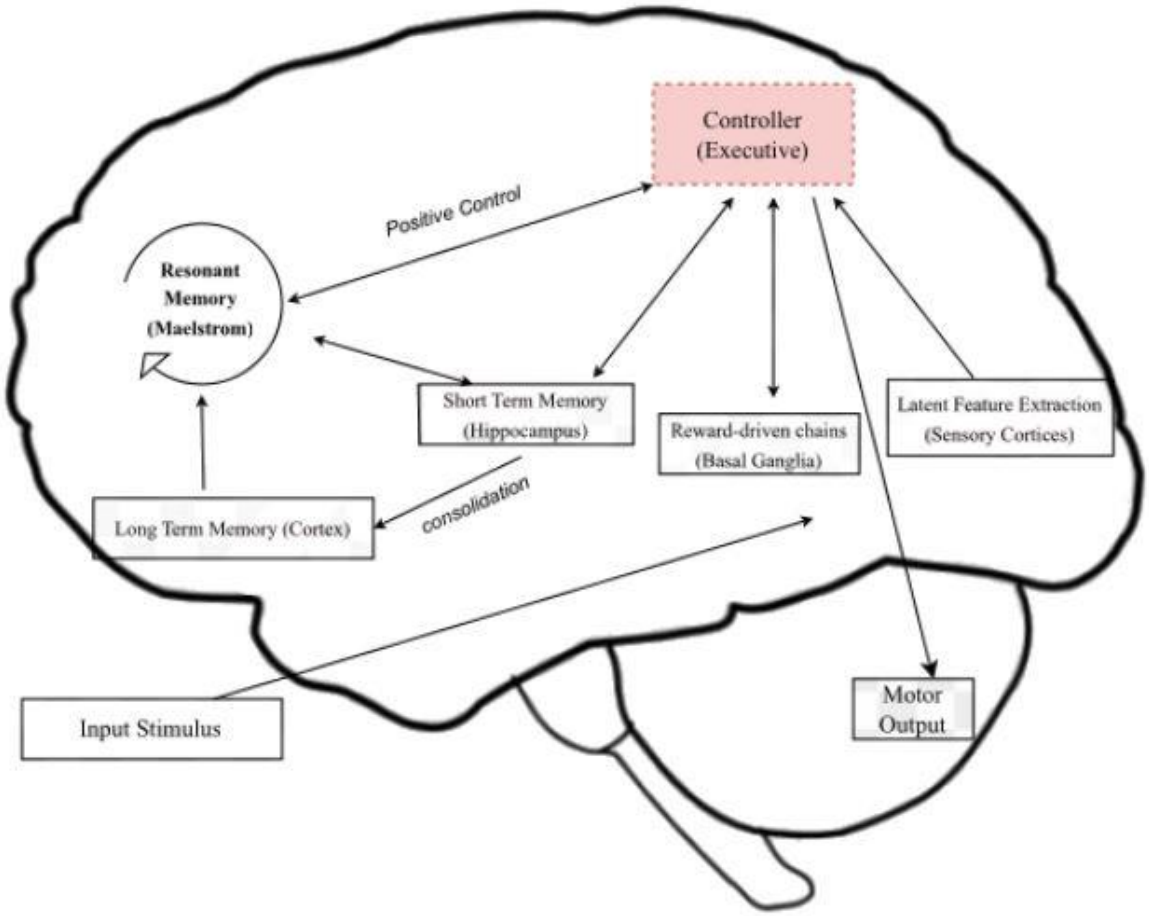


Figure 8.3: A diagram of overall proposed brain structure, as inspired by the maelstrom paradigm as well as prior works including executive brain functions [Hart et al., 1999]. The main contribution here is the suggestion of a kind of memory called *resonant* memory, which forms the building block of all other memories. The executive functions, which could reside in hub regions but is somewhat of a mystery as of now, actively maintain these resonances, and the phenomenological experience of consciousness and free will is the interplay between this executive region and the maintenance of these resonances. Similar to the ideas in Figure 2.4, we can also see the hierarchy of loops involved, with the smallest loops being the cell assembly activation in the resonant memory, and the larger ones being the loops between the hippocampus and the cortex that solidifies these memories. While "long term memory" and resonant memory are different elements here, they are located in the same neurons; I separate them here to display their unique functions. When the cell assemblies in the cortex are resonating, they are also reflecting the long term memory stored there. Note that the Short-term Memory has bidirectional connections with the resonating cell assemblies in the maelstrom, but long term memory does not flow backwards to short term; it must go through the resonating assemblies. Missing for simplicity is the role of the motor cortex and the cerebellum. The placement of the Controller is not meant to imply it sits in the motor cortex - in fact we do not know where this sits, and is potentially the biggest open question in neuroscience.

9 Completed Works

In this section, I list the works that led along a temporal chain of their own, towards the final proposal of the *maelstrom* thesis, as well as neurally plausible instantiations in the brain as cell assemblies.

Hybrid Backpropagation Parallel Reservoir Computing

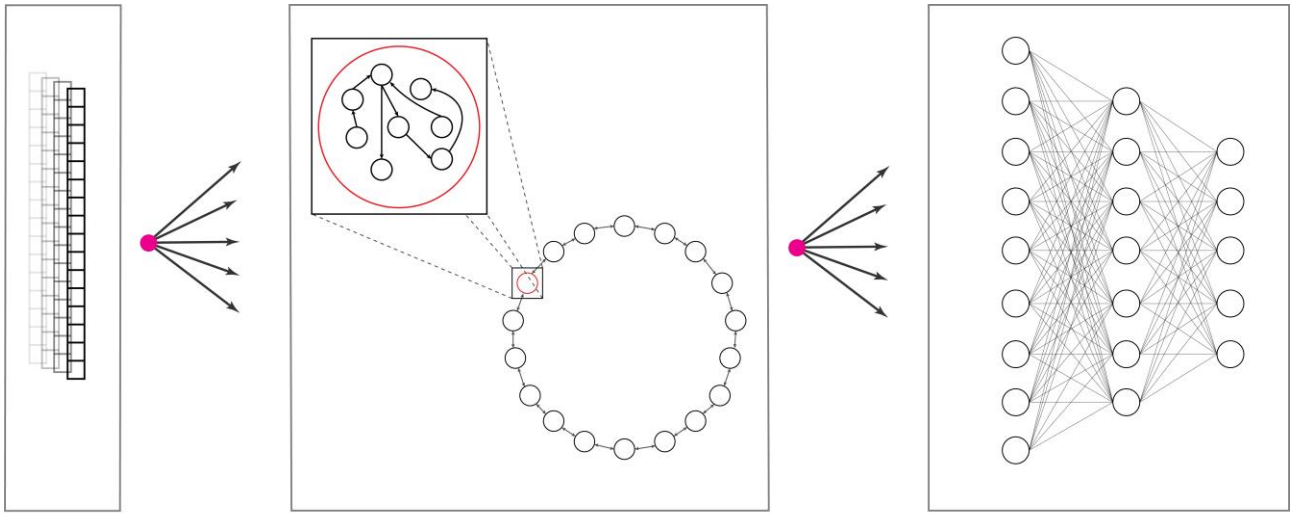


Figure 9.1: The HBP-ESN M-Ring architecture: (A) the input to the reservoir, with shading corresponding to time steps; (B) the parallel ring reservoir; (C) the backpropagated readout. The magenta dots and fan-out arrows between A, B, and C represent fully connected matrices. The ring in B is straightened and then flattened before being passed into C. Each sub-reservoir, shown zoomed in, is connected to its left and right neighbor in the shared weights structure, forming a ring of sub-reservoirs. In different experiments, some elements are swapped; shown here is the full contribution with the shared parallel cross-talk weights. HBP-ESN without M-Ring lacks the ring links. In the single reservoir experiment, part B consists solely of a large version of the zoomed-in sub reservoir. For the feed-forward only experiment, part B is removed entirely.

My first realization was that we can model the state of these resonating cell assemblies using tools from the "reservoir computing" paradigm:, where we instantiate a random pool of interconnected neurons, and let a readout mechanism learn the activations. On the way to the maelstrom paradigm, I first asked the question: what would it

look like if we allowed the network to learn more complex features from these resonating neurons? What if we used a powerful feed-forward neural network?

In "Hybrid Backpropagation Parallel Reservoir Computing" (In preparation for re-submission, after being rejected from Neurips 2020 by a small margin), I have devised a novel hybrid reservoir computing + deep learning framework, that aims to take advantage of the temporal processing power of the reservoir in addition to the readout, pattern-recognition power of modern rate-encoding deep neural networks. One of the proposed assumptions of reservoir computing is that the reservoir is able to perfectly "linearize" the input data for readout, or in other words, any input pattern, when passed into the reservoir, can be learned with a linear readout classifier. This work tested that hypothesis, with the idea that we don't exactly know the ideal size and topography of the reservoir for a given task, so it may benefit from using a readout that can learn non-linear mappings. In particular, it uses the full force of modern deep feedforward learning approaches, such as weight decay, batch training, and batch normalization. It also uses a parallel structure inspired by [Pathak et al., 2018], with the hope that each sub-reservoir can learn unique characteristic features. The sub-reservoirs were then connected in the simplest way possible - one large ring - with the same random connectivity matrix to save memory space. The results showed that this new ring hybrid reservoir outperformed LSTMS and GRUs on multiple difficult real-world time series tasks. See Fig. 9.1 for architecture details.

A variant of this work in [Evanusa et al., 2022] was accepted into *Springer Nature Computer Science*.

Event-based Attention and Tracking on Neuromorphic Hardware

In [Renner et al., 2019a], we introduced a fully end-to-end pipeline for processing raw DVS input data on a brain-inspired neuromorphic device, the Intel Loihi chip [Davies et al., 2018] (see related work for more details). The neural network framework used was built off of *dynamic field theory* [Sch¨oner, 2016], which aims to mimic the excitatory and inhibitory response of a population of neurons with winner take all dynamics, as a means of simulating cortical dynamics [Amari, 1989]. The core computational insight is that the entire brain can potentially be abstracted as a sequence of winner take all layers, where each layer contains excitatory neurons surrounded by locally inhibitory connections [Sandamirskaya, 2014]. For this task, we used the neural fields to successfully track an object as it moved across the visual field using the DVS camera (See Fig. 9.2). This is possible because the locally inhibitory connections "cancel out" noisy signals that are not relevant to the tracking task at hand, which also serves as a form of noise removal or whitening. The main novelty of this work was that this was entirely processed on the Loihi chip. This work could in theory open the doors to new methods for processing visual information using spiking networks on neuromorphic chips. This can be seen as a different approach to solving the sequence memory problem. Rather than saving the memory as a sequence with the maelstrom, dynamic field theory is used as the memory substrate. In both cases, the sequence of the object moving across time is reflected as dynamic activity of a separate substrate. In the case of dynamic field theory, it is the continuous-time equations that simulate the winner take all neural mechanism.

This work was accepted into the 2019 CVPR Workshop [Renner et al., 2019b].

A Deep 2-Dimensional Dynamical Spiking Neuronal Network for Temporal Encoding Trained with STDP

In [Evanusa et al., 2020], I introduced (what was, at the time, a novel implementation) of a deep, 2-dimensional spiking neural network for DVS gesture recognition. The goal was to learn to classify crude gestures directly from raw DVS input (See Fig. 9.3). The neurons in the network evolved according to the Izhikevich neuron model [Izhikevich, 2003] (see section 5.2). The synapses learn through Spike Time Dependent Plasticity (STDP) [Dan and Poo, 2004]. (For a visualization, see Fig. 9.4). The network was completely implemented on the GPU in custom CUDA code. This allowed the network to run in real time on millions of neurons (with hundreds of millions of synapses). The network was more biologically plausible than standard CNNs in that it did not have a shared convolutional kernel, each "kernel" was unique to its given area patch. In order to obtain translational invariance, further work is needed to incorporate an attention mechanism. Here, the memory of the system is also one of a dynamical system, except rather than being a dynamical memory unhooked from the error learning as in the Maelstrom paradigm, it was instead the dynamical memory of the membrane potential within the neuron. This is

one benefit of spiking neural networks: the implementation of the integrate and fire neurons “gives you” memory, for free, in that the neuron is itself a stateful system. The network, being trained with STDP, means that it is possible for this network to serve as the Maelstrom as well. At this time, I had not

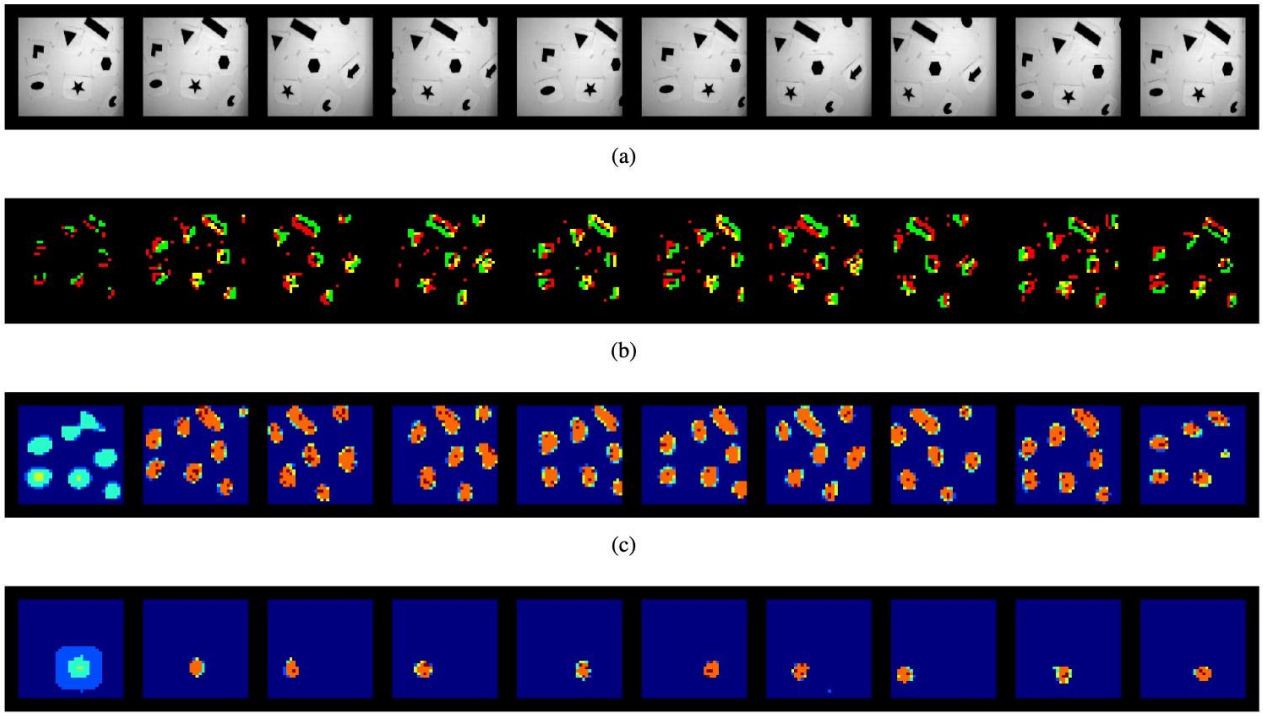


Figure 9.2: Tracking of an object using Dynamical Neural Field-based spiking neurons on the Intel Loihi chip. The bottom row demonstrates the final processed, tracked object.

devised the theory of the Maelstrom, but as is clear with my work, even before the final culmination, all work led to the same goal of a stateful, temporal system.

Deep Reservoir Networks with Learned Hidden Reservoir Weights using Direct Feedback Alignment

One of the major weaknesses of ”vanilla” reservoir computing is that it has not been able to upscale to the kinds of complex problems that are of interest to the community, such as image and video recognition, generative models, and complex robot control. Full-gradient RNNs are still used because the error can be backpropagated through the recurrent layers, so they are ”stackable” with other feed forward architectures. For most applications thus far, deep backpropagation networks still reign supreme. It is noted in this thesis that a goal of these works is to bring recurrent neural networks back into the mainstream via hybrid reservoir approaches. One of the major breakthroughs in artificial rate encoding networks was the introduction of multi-layered approaches to network learning. This innovation allowed deep networks to learn non-linearly separable data. [Rumelhart et al., 1986]. Along the same vein, a new line of research has risen, attempting to replicate this deep, hierarchical structure for reservoir computing [Gallicchio et al., 2017]. This is a positive step forward, however, these networks do not learn hidden representations, as ANNs do. This brings into question what the purpose of having a deep network is in the first place; in the case of a deep ANN, this would be akin to having multiple layers, but only learning on the last layer. While this approach is not entirely ineffective [Cao et al., 2018], it does not approach the performance of learned hidden representations.

The next logical step in deep or hierarchical reservoir computing is to learn the weights of hidden connections between reservoirs. The proposed network architecture can be seen in (Fig. **??**). This is not possible with normal

backpropagation, as the backup step through the reservoir is non-differentiable. This is precisely what makes training RNNs with backpropagation so difficult. This line of research is one step towards making recurrent neural networks more mainstream again, and more powerful. In addition, this affords the promise of stacking reservoirs in the same way that LSTMs can be stacked.

In this work, I implement a novel deep reservoir computer, where we can actually now learn the "hidden" inter-reservoir weights using the Direct Feedback Alignment, as the gradient need not pass through each reservoir. Thus, the separation between gradient learning and state learning is somewhat preserved, but the direct

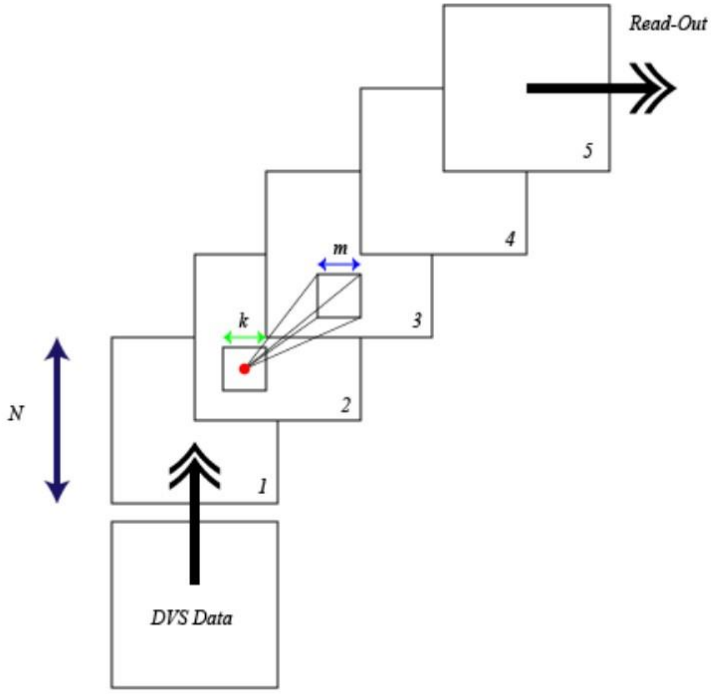


Figure 9.3: Diagram of the 2D Deep SNN architecture. Raw DVS data is fed in as input, and the network learns to organize the input via unsupervised STDP.

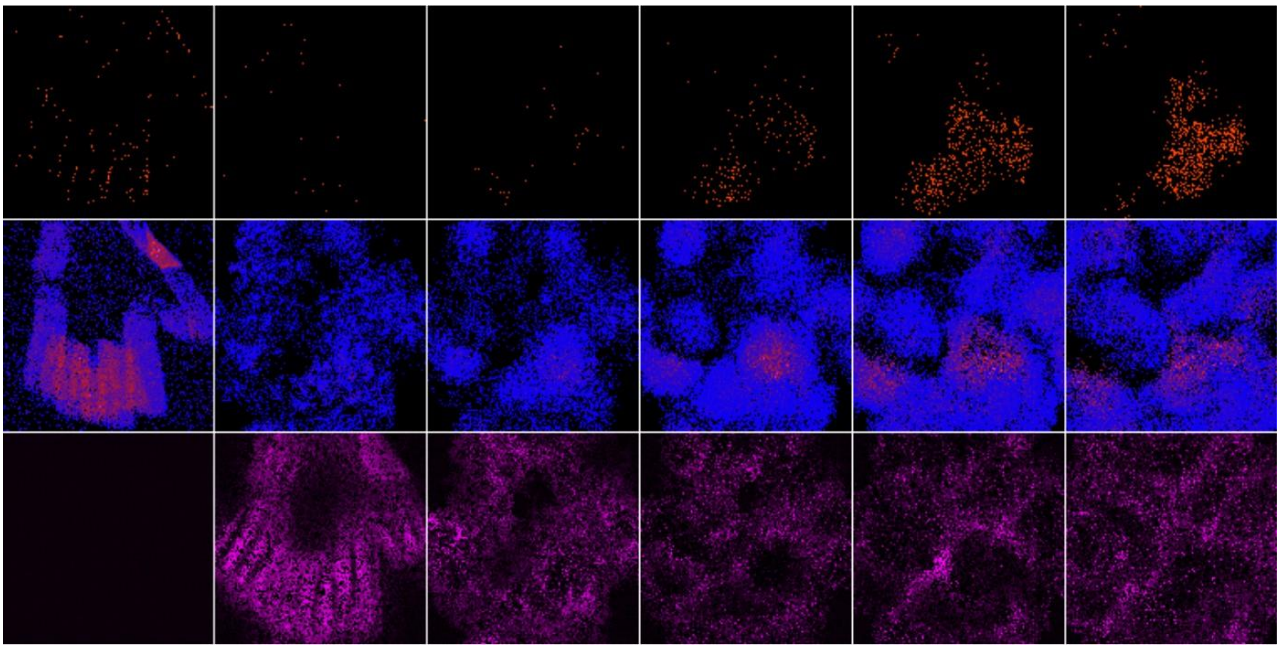


Figure 9.4: Visualization of the a novel deep SNN architecture. The image represents the 2D grid of neurons. The top row displays the neuron's firing, the second row displays a time average of the firing over a given firing window, and the bottom displays the network entropy.

gradients allow us some control over learning the dynamics of each reservoir. This is a compromise ground, and is more likely what I believe is occurring in the brain: separate learning for state-based sequence memory, with learnable control mechanisms. A diagram of the schema is presented in Figure 9.5.

This is not the first work to see the potential for biologically-plausible network using feedback alignment (see Sec. **??**). In [Kaiser et al., 2020, Neftci et al., 2017], feedback alignment was used to train a multi-layer spiking neural network. I have independently devised this mechanism to train a deep reservoir computer. However, this proposed work differs from [Kaiser et al., 2020] in two key ways. First, it is proposed for both rate-encoding and spiking models. Second, while [Kaiser et al., 2020] learns the recurrent weights of each layer, my proposed work will maintain the reservoir paradigm by learning the readout hidden layer weights of each reservoir-layer, keeping each reservoir-layer's recurrent connections random.

The contributions of this towards the maelstrom paradigm are that this may allow us to create more complex resonating cell assembly structures. The potential to use "skipping" gradients, which resemble dopamine projections, is huge, and with more refinement this could allow for an arbitrary state-based learning setup with specifically designed topologies of recurrent cell assemblies.

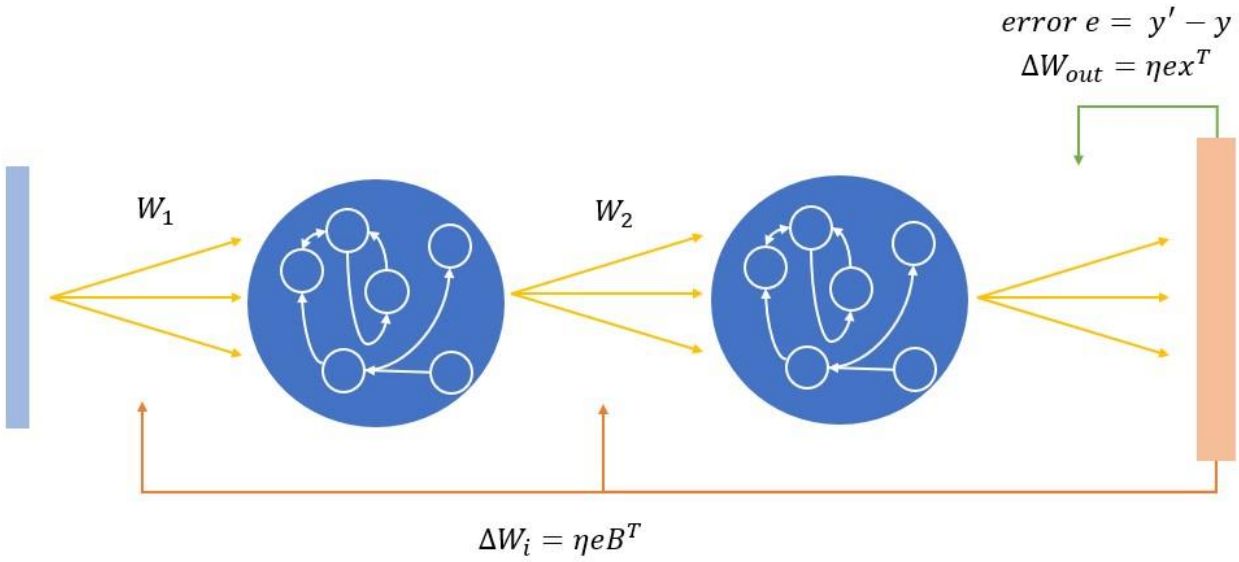


Figure 9.5: A diagram of the deep reservoir paradigm with learned weights using Direct Feedback Alignment. The error for the task (output layer) is calculated, and this error is used as a "projection" to the hidden inter-reservoir layers. This allows us to project a control signal to the cell assemblies, but without passing gradient through them.

This work was accepted to the Neurips 2020 Workshop "Beyond Backpropagation".

tConv-ESN

One of the weaknesses of the feed-forward deep readout for reservoirs was that the network size grows exponentially large as the sequence size increases (assuming we feed in the entire sequence at once - which is no longer the case for the maelstrom paradigm). The reason we need to feed the whole sequence, still, is that the fading memory of the random pool cannot persist across long time intervals - this is one of the reasons why LSTMs introduced the gating mechanism in the first place [Hochreiter and Schmidhuber, 1997]. To alleviate this, a special readout was introduced that learned a one-dimensional kernel for each reservoir neuron. This effectively turned the problem into a signal processing one, with a deep 1-D convolutional network assigning the patterns of the neuron's signals to specific tasks. This CNN readout drastically reduced the parameter count for the network, as well as increased performance.

This work, *t-ConvESN*, was presented at ICANN 2023 and published in Springer [Evanusa et al., 2023].

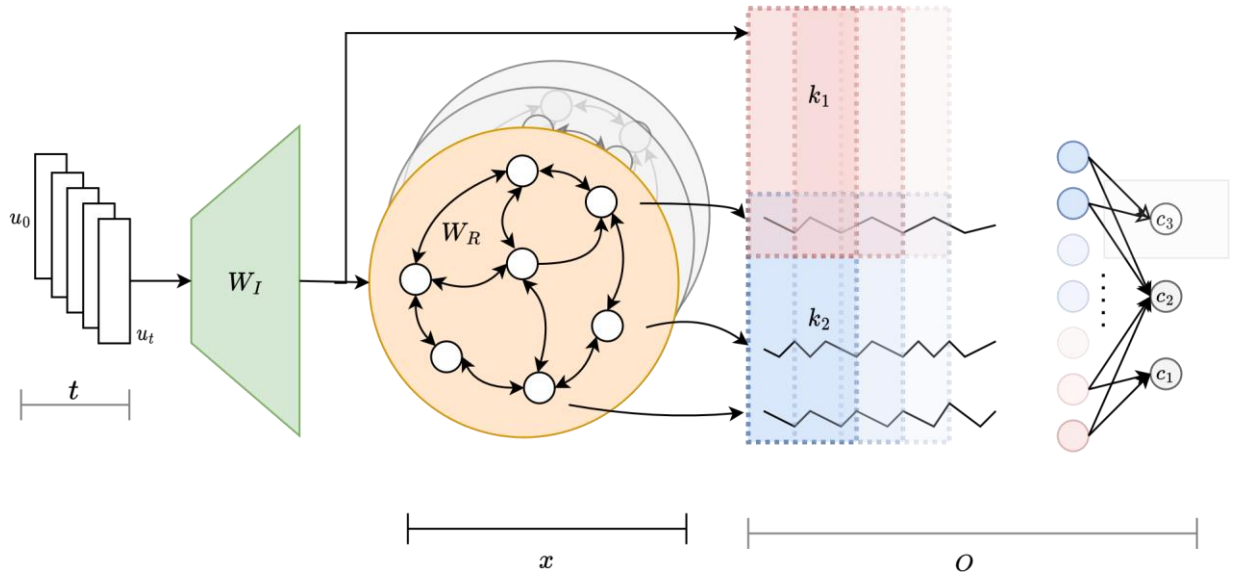


Figure 9.6: A diagram of the tConvESN architecture. The convolutional readout turns the problem of the readout into learning 1-dimensional filters for each neuron, and dramatically reduces the parameter count of the readout while increasing accuracy. We found this network to beat other state of the art time series classification networks, such as Transformers and LSTMs.

Maelstrom Networks

The penultimate combination of my work has been described in Chapter 7, the Maelstrom paradigm. It combines my work leading up to it with non-linear readouts for a stateful, randomly initialized cell assembly built off the theories of reservoir computing. As described, Maelstrom Networks is a paradigm for neural networks that contain a dynamical memory that is unhooked from the error processing of the task. While this offers a novel and potentially promising avenue for neural network design, it needs to be implemented to prove its efficacy. In this work, I take the first attempt to show that the network, with controller and readout parameterized by deep neural networks, is able to learn temporal signals significantly better than feed-forward networks. The memory of the network is kept separate and running while the signal comes in, with the feed-forward networks' role being specifically to learn a mapping from the current state of the memory to a target action. Note that this does not include an explicit executive system, simply a system that is "one step removed" from implicit: it reads out a fading trace of the system. Nonetheless, based on my definition of explicit vs. implicit, this can be seen as the simplest example one can give of an explicit system.

The testing for this system is somewhat novel: it involves taking a temporal signal and "chunking" it, such that the network only receives a small amount of the signal at each time. This more closely resembles the real world that the artificial agent would experience should it be let free. It also enables the system to do "online" learning, as the mapping that the readout learns does not depend on the entire length of the sequence, just on the particular snapshot of the maelstrom.

This work is submitted to *Neural Networks* journal.

DOT-VAE

On the issue of learning latent representations of memories, my joint work on DOT-VAE tackles the problem of Disentanglement that I have described in previous sections. If we are able to, in an unsupervised manner, learn optimal latent representations of the data, then this could serve as the basis for the Maelstrom's cell assemblies to optimally remember a given task. Because the cell assemblies are to be removed from the gradient learning

process, this means that an unsupervised learning of these elements would fit in perfectly. In this work, we tackle the novel problem not only of unsupervised disentanglement, which is already itself a challenging problem, but the problem of *continual* unsupervised disentanglement, which combines the difficulty of unsupervised disentanglement with the difficulty of continual learning with deep networks. The insight here is that a solution for one of these problems solves both problems: if we can learn to continually disentangle, then we can learn an optimal representation for continual learning in deep networks. In this work, we build off of the foundations of the Variational Autoencoder and β -VAE [Burgess et al., 2018], which show that increasing the KL-loss term reduces the model reconstruction capability (as we are reducing the contribution of the MSE loss), but increases the disentanglement in the latent space. Building from this, we propose "Disentangling One at a Time", or DOT, which gradually accumulates disentangled factors in the latent space, until the network gradually settles in an equilibrium point where it can no longer find further factors. We demonstrate that the network outperforms prior models, which had not been used yet for online disentanglement. The ability of the network to determine if a new factor helps is grounded in the idea of *interventions*, where we take actions in the latent space, i.e. change a latent value, and see what the resulting reconstruction looks like. This is highly related to predictive coding as well as contrastive coding or reasoning about counterfactuals, as we use the error we get from the counterfactual reasoning (i.e., changing a latent), to deduce how the model should update. Figure 9.8 displays the DOT-VAE architecture. DOT-VAE was accepted and presented at ICANN 2022.

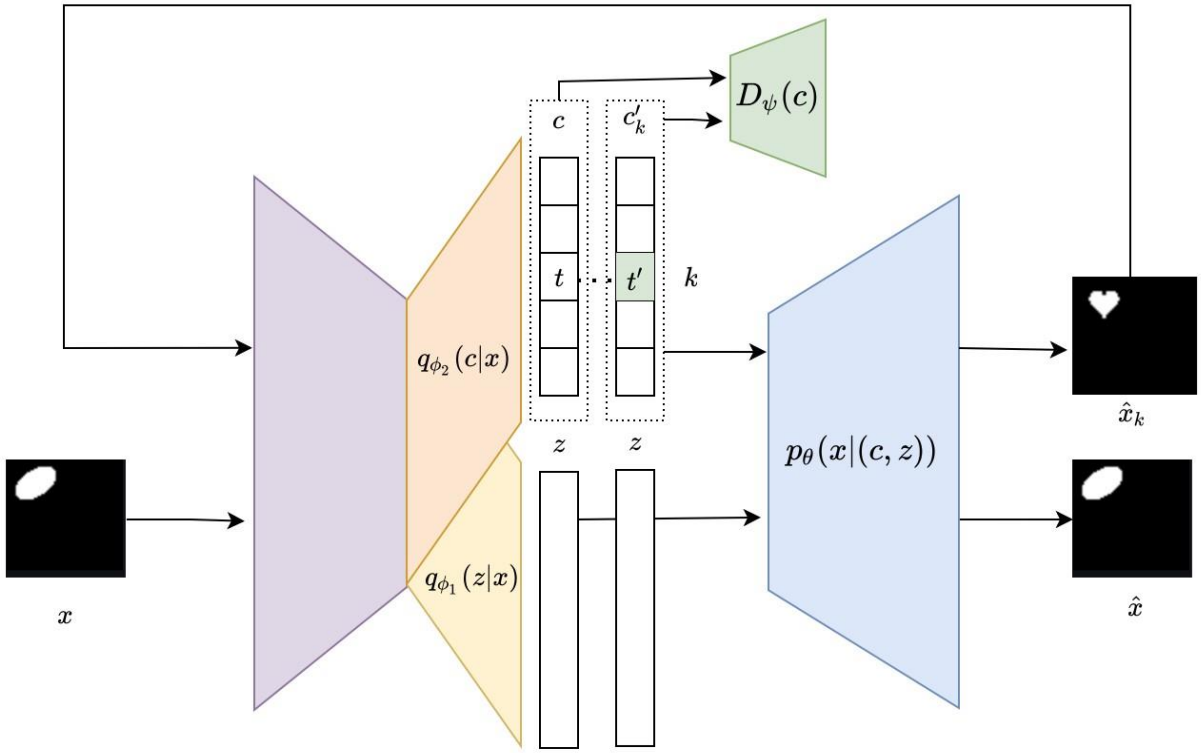


Figure 9.7: The DOT-VAE architecture. The core neural architecture is the autoencoder, with an encoder *q* and a decoder *p*. In contrast to prior work, we now have *two* latent spaces, one that is disentangled, *c*, and one that is entangled, *z*. The job of the network is to continuously pluck information from *z*, and place it in *c* in a disentangled fashion one at a time (hence the name of the architecture). To determine what remains to be disentangled, we perform *interventions* on the *c* space by automatically changing values of latents, and predicting the resulting reconstruction. We then use an adversarial loss from an adversarial network *D* as the loss update for the encoder *q*.

ProtoVAE

Similar to DOT-VAE, we aimed to solve unsupervised disentanglement. In this work, closer to the idea of sequence memory, we want to save the prior information about sequences in *prototypes*, or clusters, which are akin to a stateful memory that persists. This builds off of the work of Prototypical Networks, where the network learns set *prototypes* that best summarize the data [Snell et al., 2017]. In the context of this thesis, these prototypes can be seen as a form of persistent memory. It builds off of the prior work from DOT-VAE, and similarly uses interventions in the latent space, combined with two separate entangled and disentangled spaces, to allow the network to discern when it should settle in an equilibrium state. However, unlike DOT-VAE, we feed in the predicted generations to a prototypical network. The role of this network is to learn which factor changed; thus the network is learning the distance vector between the two images, and clustering them. In effect, it is learning to cluster actions on the latent space.

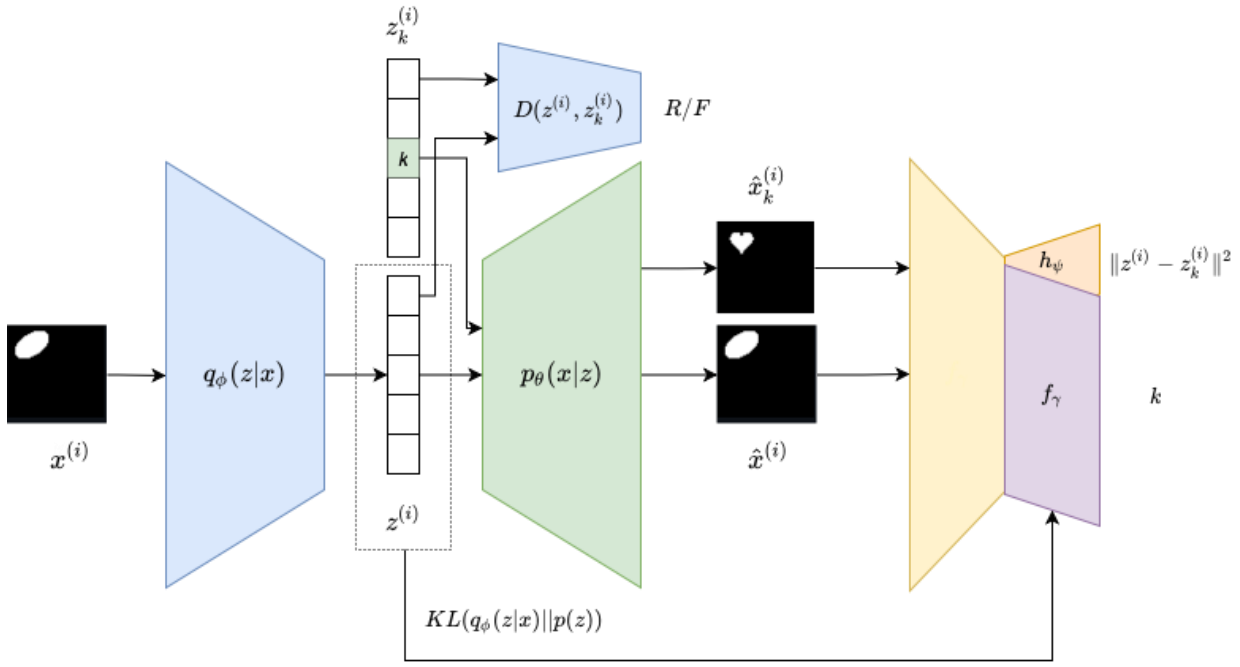


Figure 9.8: The ProtoVAE architecture. The core architecture builds off of our prior work on DOT-VAE, with the inclusion of a new network that learns to discern the changes that the autoencoding network made in the latent space. The encoding and decoding networks are similar to Figure 9.8, minus the prototypical network. The role of the prototypical network is to aid the network in credit assignment for learning by providing gradients that reflect the change of the latent variable.

10 Concluding Remarks

In conclusion, I have attempted to build up from neurodynamical and physical principles a foundation for reverse engineering the human brain towards the next state of artificial intelligence. I argue that events in the universe are not isolated, random events, but are rather highly temporally correlated. These temporal correlations, temporal sequences, must be a singular focus for the evolution of the living brain. I argue that the brain does in fact focus heavily on encoding temporal sequences, via mechanisms that were discovered by Donald Hebb in the middle of the 20th century. These resonating cell assemblies form the core building blocks of all sequential memory in the brain. This temporal, sequential activity of resonating cell assemblies combines with a readout mechanism in output cortices to produce the functions that we observe in living creatures.

However, we do not see a focus on temporal correlation as an inductive bias in the structure of design in deep learning. This lack of focus does not endow the system with a sense of self, or state, which allows it to persist across time. In an effort to endow deep learning with a state, and moving towards recreating this in artificial systems, I suggest that we combine this notion of the recurrent, resonating cell assembly as the persistent state, with a readout mechanism, which can be performed by current state of the art deep networks. To this end, I propose a new paradigm, *Maelstrom* networks, where the maelstrom is the set of these resonating cell assemblies, removed from the error learning component of the network. The error learning, instead, is performed by a control network, and a readout network, parameterized by deep neural networks. I demonstrate that this network is highly effective in temporal pattern recognition. It is my hope that this opens the door to future bi-directional research, with discoveries in neuroscience uncovering new avenues for deep learning, and visa versa. It is my hope that my suggestion shifts the zoom focus of reverse-engineering towards that of the cell-assembly and state. If we are able to show and model how the brain encodes memories in these cell assemblies, in addition to generating the next advancement in artificial agents, it may also be possible to repair disorders that originate from structural faults in the assembly system, such as in Alzheimer's' disorder. I believe the memories are still there encoded, but lack an appropriate access mechanism. Memories are who we are, and what sustains us. A deeper understanding of memory is tantamount to a deeper understanding of life itself.

A Deep 2-Dimensional Dynamical Spiking Neuronal Network for Temporal Encoding trained with STDP

Matthew S. Evanusa*
Dept. of Computer Science
University of Maryland
evanusa@cs.umd.edu

Yiannis Aloimonos
Dept. of Computer Science
University of Maryland
yiannis@cs.umd.edu

Cornelia Fermuller
Dept. of Computer Science
University of Maryland
fer@umiacs.umd.edu

Abstract

The brain is known to be a highly complex, asynchronous dynamical system that is highly tailored to encode temporal information. However, recent deep learning approaches to not take advantage of this temporal coding. Spiking Neural Networks (SNNs) can be trained using biologically-realistic learning mechanisms, and can have neuronal activation rules that are biologically relevant. This type of network is also structured fundamentally around accepting temporal information through a time-decaying voltage update, a kind of input that current rate-encoding networks have difficulty with. Here we show that a large, deep layered SNN with dynamical, chaotic activity mimicking the mammalian cortex with biologically inspired learning rules, such as STDP, is capable of encoding information from temporal data. We argue that the randomness inherent in the network weights allow the neurons to form groups that encode the temporal data being inputted after self-organizing with STDP. We aim to show that precise timing of input stimulus is critical in forming synchronous neural groups in a layered network. We analyze the network in terms of network entropy as a metric of information transfer. We hope to tackle two problems at once: the creation of artificial temporal neural systems for artificial intelligence, as well as solving coding mechanisms in the brain.

1 Background and Introduction

Many of the major advances in A.I. have been made due to reverse engineering biological brains, including Deep Neural Networks (DNNs) and Reinforcement Learning (RL) ([33]). Research into A.I. has been given a boost in recent decades due to success in reverse engineering neural connections and functions of the mammalian cortex, and in reverse, machine learning

approaches have helped computational neuroscientists in learning spike patterns from neural data. So-called “deep learning” frameworks ([24]) adopt the feed-forward, layered aspects of the connections in the brain, while some (convolutional networks) go further to mimic the convolutional connections that are present in many pathways. These approaches have demonstrated human level accuracy in some tasks, such as face detection from static images ([38]). However, because they rely on non-biological learning mechanisms, it is difficult to use these networks to uncover biological mechanisms for neural information encoding. In addition, these networks still struggle to classify temporal data; this will be a critical next step for A.I. systems of the future. In the neuroscience community, it is a current debate as to whether the information is encoded in the firing rate of the neuron, or if the information is encoded in the precise timing of incoming spikes ([32, 41, 42]). It has been shown that unsupervised learning rules, such as STDP (section 1.4), are capable of encoding this spike-timing information into the synaptic weights ([14, 22]). It has also been argued that through these learning rules, the neurons compete with one another to form groups in a selection process ([11]). Here we investigate whether biologically-inspired neural plasticity rules combine with precise timing of incoming firing rates to encode information about novel stimuli in the form of a biologically-inspired camera in a multi-layered network.

1.1 Rate-Encoding Models

Current ANNs, including deep-learning frameworks, are what computational neuroscientists refer to as "rate-encoding models", in that the real-value of the output from a unit (roughly approximating an integrating neuron) represents a firing rate, or a probability of firing, over time. These networks accept inputs all at once and are also referred to as "linear non-linear" models ([34]), in that they represent a combination of a linear operation (summation of incoming connections), as well as a non-linear operation called the "activation function" (some sort of sigmoidal, inverse tangent function, or rectified linear unit). The rate R of a neuron’s fire is algebraically:

$$R = N/T \qquad (1)$$

for N spikes in a given time-scale window T. As we can see from this interpretation, these rate encoding networks *divide out* time as a factor in their design; they are fundamentally time-less. Any rate-encoding network that attempts to encode a sequence of images as a video is not encoding 2-dimensional images in time, but rather the entire sequence of 2-dimensional frames at once as one 3-dimensional feature block.

1.2 Dynamical and Reservoir Computing

Our proposed network is closer to what is referred to as "reservoir computing" ([30]), in that it acts as a chaotic, dynamic system due to the neuron update, lateral connections, and randomness placed in the network. Reservoir computers are a type of recurrent neural network that do not learn on the recurrent connections within the random recurrent connections but rather only on a read-out layer. Reservoir computers have demonstrated high success in classification of difficult temporal patterns due to the integration of time information in each artificial neuron’s update rule. They have been shown to encode very complicated temporal input, even input that exhibits very chaotic patterns such as the Lorentz attractor. Although reservoir computers require a strong supervised signal to drive them towards a target, they still demonstrate that random weights and connections are a powerful factor in a network’s ability to encode temporal data. As such, they have been successfully used to analyze complicated recurrent neuronal dynamics in decision making ([7, 13]).

1.3 Spiking Neural Networks

The so-called “third generation” of neural networks, SNNs offer a promising way of encoding time information, although the dynamics are hardly new ([17]), and what constitutes a "spiking neural network" is broad, as well as the desired goal of biological plausibility or classifying power. SNNs remain at this point a loose collection of different neural-like networks that share the common feature that individual neurons operate as dynamical systems that individually collect voltage and “spike” an output at a given threshold, but differ in their layout and learning rules. SNN activation functions for the neuron vary widely, from neurons that simply integrate over time without any voltage leak ([40]) to those that model ion channels at individual compartments along the neuron’s dendrite ([35]). Analysis of the interconnection of these spiking units in larger networks, however, is a more recent undertaking ([15]). Efforts have been made to show that backpropagation can be implemented with SNNs as well ([3]), although this departs from the biological plausibility and principles of self-organization. Here we analyze the information-encoding potential of a semi-recurrent feed-forward cortical-like deep network of spiking neurons, and investigate whether information is stored in assemblies as Hebb postulated. We take the approach of using a network that has solid biological basis while trying to keep the operations as simple as possible.

1.4 STDP and iSTDP

Donald Hebb postulated that given a neuron’s spiking behavior, which is exhibited in SNNs, neurons organize into groups (what he called cell assemblies) that encode information ([16]). Hebb formulated a learning rule that has been boiled down to the phrase "neurons that fire together, wire together.", and is formalized for continuous (rate-encoding) networks as:

$$\delta w_{1,2} = \alpha N_1 N_2 \qquad (2)$$

For neurons N_1 and N_2, with learning rate α; the weight w changes only when both neurons fire. This rule allows a networks’ neurons, either rate-encoding or spiking, to learn in an unsupervised manner to reorganize their connections to match the input as there is no error term. However, this simple rule has been extensively studied and proven to be able to self-classify input data, for example in Hopfield Networks, although they have issues such as a limited patterns storage capacity([1]). Recent discoveries showed that neurons in the brain modify their connections via a *temporally asymmetric* learning rule, dubbed Spike Time Dependent Plasticity ([2, 37]), a timing-based modification of Hebb’s original learning rule that takes advantage of (and requires) spiking dynamics. STDP combines two phenomena that occur in neurons in vivo: Long Term Potentiation (LTP), which strengthens the connection (weight) between two neurons, and Long-Term Depression (LTD), which weakens it. The rule is implemented here using a common trace mechanism for STDP:

$$\delta w_{pre,post} = \begin{cases} A_+ w_{pre,post} T_{post} & \text{if } N_{pre} \text{ fires} \\ A_- w_{pre,post} T_{pre} & \text{if } N_{post} \text{ fires} \end{cases} \qquad (3)$$

for presynaptic neuron N_{pre} and post N_{post}. T is a trace that goes to some maximum value (this paper uses 2) and exponentially decays after the neuron fires, with time constant τ. A_+ and A_- are scaling factors that allow for the maximum LTP and LTD, respectively, and can be weighted.

The maximum excitatory and inhibitory weights were bounded; this is a natural balancing mechanism inspired by the fact that a given neuron can only put a maximum number of transmitter receptors physically on the outside of the neuron, and from studies that show that LTP works slower in high firing regimes [43]. STDP increases the weights for a connection between two neurons, with a given pre- and post-synaptic connection, only when the presynaptic neuron fires followed closely in time by a firing of the postsynaptic neuron. If a postsynaptic neuron fires before a presynaptic neuron, that connection is weakened ([2]). Because some synapses are also weakened, this results in a competitive process whereby only the connections that fire in lock-step with the input are strengthened. It has also been shown that STDP could be the fundamental mechanism that causes neurons, both in the brain and SNNs, to organize into Hebbian assemblies ([22]). It has already been shown that STDP is a powerful enough learning rule to allow network to categorize temporal data in the form of EEG ([23]), although the network setup was quite different from what will be described in Methods. Previous work also showed a 2-dimensional layered network that was able to classify static images of handwritten digits using STDP on a more simplified network ([20]).

However, it has also been argued that STDP alone is not enough to maintain a balanced network, as repeated high-frequency stimulus could trigger a continuous chain of LTP which infinitely increases the weights of the network ([43]). In order to attempt to counter this, we introduced both inhibitory neurons, as well as inhibitory plasticity, iSTDP, inspired by the implementation in [28]. The idea is that an inhibitory neuron should increase its firing to a post-synaptic excitatory neuron if that excitatory neuron fires too often.

As a consequence of STDP, certain "pathways" in the neural substrate begin to strengthen,; these are the cell assemblies that Hebb postulated. Because they are more tightly connected, a lesser input is required to reignite the group. In addition, because all the neurons are tightly bound, a reactivation in a small subset of the neurons will cause chain firing, ending up in the activation of the entire group[18]. A consequence of this is that if information can be successfully encoded in a neuronal group, it is extremely robust to noise and missing data.

2 Methods

2.1 DVS Camera

The temporal input being investigated comes in the form of the biologically-inspired DVS camera ([9]). The DVS sensor sends data in packets, called "events", rather than frames, and is a good candidate to take advantage of the independent nature of each neuron’s input in an SNN. The DVS camera is inspired by certain retinal ganglion cells that spike when a change in intensity is detected ([36]). Similarly, rather than send all pixels at the same time, as with RGB cameras, the DVS camera sends only events that correspond to large changes in pixel intensity. Because of the current instruction-based hardware, this network batches the events into discrete frames, although it could be transplanted into a neuromorphic chip hardware in the future that receives input asynchronously.

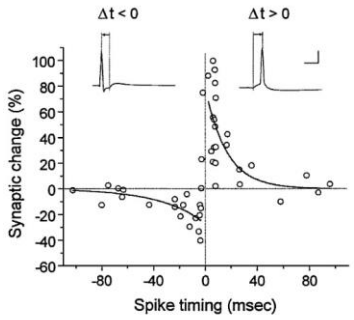


Figure 1: A typical STDP curve. Y axis is change in weight, X axis is time between post and pre-spike. A negative time difference (post fires before pre) will decrease the weight, and the inverse for a positive one. (from Bi and Poo, 2001)

Figure 2: Example DVS frame of a hand moving used in the input. The events capture the major shift in intensity of the edges of the hand as the hand moves across the background.

Recent work has been devoted to discovering good encoding mechanisms for DVS data for use in rate encoding network, whereas we make the argument that the raw DVS events themselves work well with the SNN network structure.

2.2 Neuron Dynamics and Type

The neuron activation is based off of the Izhikevich neuron ([21]), which is a variant of the quadratic integrate and fire model; it is able to reproduce many behaviors seen by neurons in vivo, except is more computationally efficient than more complex models such as Hodgkin-Huxley ([17]). Specific neurotransmitter conductances are not modeled explicitly. At each time-step, the voltages for each neuron at each layer are updated according to the following equations:

$$v^0 = 0.04v^2 + 5v + 140 - u + I_{syn} \tag{4}$$

$$u^0 = a(bv - u) \tag{5}$$

with the voltage being reset to reset parameter c, and the variable u being incremented by d, when the voltage goes over a threshold, $-30mV$, which indicates that the neuron spiked. We used the same parameters as in [21] to model a regular-spiking cortical neuron, with $a = .02$, $b = .2$, $c = -65mV$ and $d = 2$. In order to add randomness to the network, c is modified by $15r^2$ and d is modified by $-6r^2$, where r is a random number on the interval [0,1]. We did not model inhibitory neurons differently.

At each time-step, every neuron *j* updates its voltage as in eqn. 4, after calculating the incoming synaptic current. For layer 1, this current is equal to 0 or 1, depending on if an event occurred from the DVS frame. If the neuron is layers 2 or later, it calculates its voltage via the following:

$$I_{syn(i)} = \sum_{j=0}^{k^2} w_{j,i} s_j + \sum_{l=0}^{m^2} w_{l,i} s_l \qquad (6)$$

where $I_{syn(i)}$ is the incoming synaptic current into neuron i, k is the length of a side of the kernel for incoming connections and m is the length for horizontal (see Figure. 3); s_x is 1 if neuron x spiked and 0 else. Indices j and l are neurons that have synaptic connections to neuron i from feed-forward and lateral connections, respectively. If a neuron is inhibitory, its weight is multiplied by -1. The neurons' weights are set within a bounds, [0, w_{max}] for excitatory and [$-w_{min}$,0] for inhibitory.

2.3 Network Structure

The network consists of a set of connected "layers" of spiking neurons. Each layer is a 2-dimensional sheet of neurons of the same NxN dimensions (see Figure 3 for schematic). The code is completely custom-written, and is written expressly for the NVIDIA GPU using the CUDA programming language ([29]) to maximize speedup. The advent of GPU technology allows large-scale networks like this to run in real time, even while training: a 5-layered network of 612,000 total neurons with
122 million total synapses, on the NVIDIA Titan Black, runs at 38.5 Hz without STDP updates, and 20Hz while training with STDP.

The connections are inspired by how feed-forward connections in the mammalian brain are highly overlapping and convolutional ([12, 25]) and hierarchical ([44]), being the inspiration for Convolutional Neural Networks (CNNs)[26]. However, unlike CNN's, the kernels do not share weights, in an effort to remain biologically realistic, as well as to investigate if the non-shared quality of weights is an important factor in a spiking network. The connection schema is loosely based on a mix of [31] and modern CNN architectures. Each neuron has feed-forward connections to a small square block of neurons in the post-layer, exactly as in a CNN. This is done intentionally because the network is not learning a kernel using backpropagation, and needs as much variation as possible to aid in the STDP process. In addition, in attempting to reverse-engineer the fundamental principles of spiking, it is biologically implausible that neurons communicate some sort of shared weight across the cortex; weight updates are local by nature. The weights are initialized with a uniform random distribution, and given an equally-high starting weight that allows spikes to occur so that STDP can begin reshaping the weights, consistent with theories that state that intrinsically high starting activation is a major part of neuronal group formation ([18]). Excitatory weights are capped at +7 and inhibitory at -30, and STDP is only

performed on feed-forward connections. In addition to the feed-forward connection, each neuron also connects horizontally to a square of neurons in the same layer. This provides a degree of recurrence to the system, implementing a version of "re-entrant" connections as described by Edelman ([11]), although more long-distance reentry is left for future work. We refer to the system as "semi-recurrent" and not fully recurrent because it does not contain backward connections to previous layers.

The network features both excitatory and inhibitory neurons, to maintain balance. As per prior work
([22, 28]) the ratio of excitatory to inhibitory is set at 4:1, to mirror the ratio of neurons seen in the mammalian brain. Only 20% of feed-forward connections and 30% of lateral connections were kept non-zero, per calculations done about the probability of connections in the cortex being between .1 and .3 ([5, 27]).

2.4 Network Information and Entropy

Much work has been done to analyze the information capacity of neurons' firings using information theoretic metrics. ([4, 6, 41]). Here, we use a metric, network entropy, that has been used in networks analyzing cancer genome alterations ([39]), that analyzes the local entropy of the flux around each

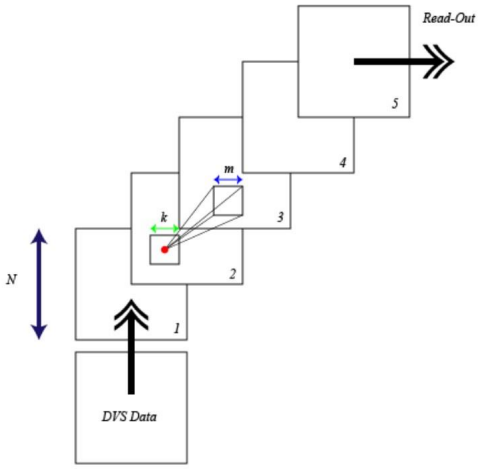


Figure 3: The network architecture. Each layer is a 2-dimensional sheet of Izhikevich neurons of dimension N x N. A given neuron, represented in red, has both horizontal connections to neurons in a k x k box around it in the same layer, as well as non-shared feed-forward connections in an m x m box in the next layer. The input DVS data is fed into the first layer, and the categorization is read out from groups formed in the last layer. Layer Numbers are indicated in the bottom right corner.

node. To calculate the entropy for one neuron, we calculate an averaged entropy over all the outgoing connections, one entropy for inhibitory and one for excitatory:

$$E_i = -1/log(K) \sum_{j \in N(i)} p_{i,j} log(p_{i,j}) \quad (7)$$

Where E_i is the entropy for a given neuron i, j is a neuron with a non-zero weight connected to neuron i, and K, the degree of i, is the number of non-zero connections to neuron i. The mutual information given by an input is analogous to a decrease in entropy after a stimulus is presented([8]).

3 Results and Discussion

Fig. 4 shows some results after training the network continuously. The top row shows the individual neurons spiking; the second row shows a spike count over a 300ms time window; the third and fourth row show the network entropy for each neuron. When the network starts running, the entropy plot is completely black; all the weights are uniformly random, so the entropy is at maximum. As STDP begins to strengthen the input patterns, and convolve itself forward into new patterns, it changes the weight distribution around a given neuron, thus altering the entropy.

Due to the effects of the convolutional connections and STDP, rapid pulses of activity propagate through the network that do not occur without learning; this activity is similar to activity demonstrated in cortical-like networks in the brain ([10]). Because the weights are capped, and due to inhibitory signals and the voltage decay, this means that these neurons must be activated by concurrent, synchronous firing at precise times in order to cause the postsynaptic neuron to fire. Unlike in [28], where uniformly low rates recruited depression, because the stimulus was driving the neurons, potentiation was dominant. The network exhibited local swells of activity in specific clusters, that would indicate a local synchrony, rather than a global synchrony ([19]). Because of the lateral connections, these clusters were able to move horizontally across the same layer. The groups also seem to fire at higher rates, which would give evidence that they are in fact Hebbian groups.

One point to note was that changing the parameters of the overlap size (the feed-forward convolution) had drastic effects on the activity of the network. In addition, the maximum and minimum weight

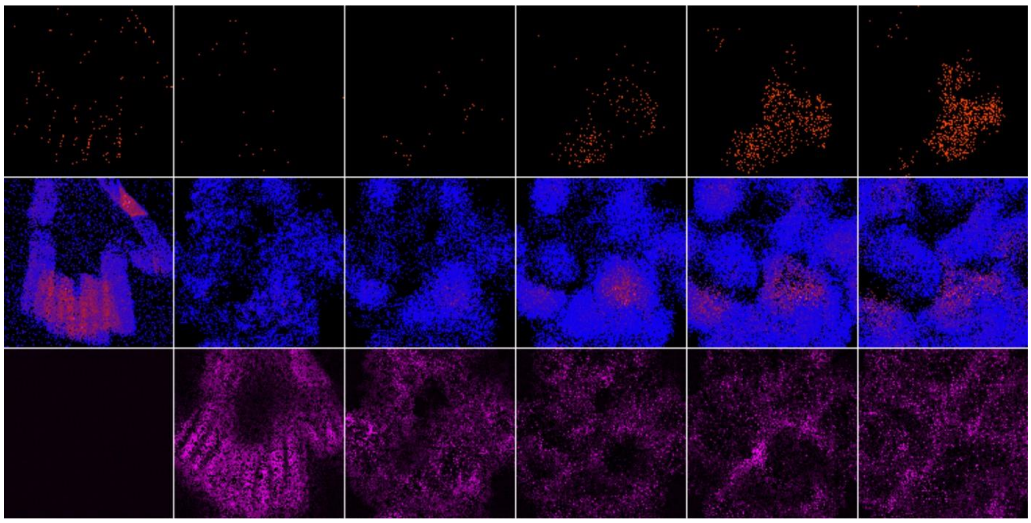


Figure 4: A display of the network running. The network consists of 96,774 neurons and 7.5 million synapses. Here we have turned off iSTDP; just excitatory STDP is running. The top row displays the firing, the second an average spike count analogous to a PSTH that gets refreshed every 300ms (red = high rate, blue = low), and the last row displays the network entropy, darker colors represent higher entropy for a given neuron. Also visible is a feed-forward propagation of activity from lower layers.

cutoffs were highly dependent on the size of the kernel: a larger kernel would overflow with activity with a much lower cutoff.

One assumption is that the network entropy carries information about what the weights are learning. The intuition is that if a neuron were to have equal-weighted outgoing connections, it would offer no processing power, as all of the outputs would be the same and indistinguishable. Because the network is learning a temporal signal, rather than a static one, we suppose that the network is learning "temporal features", in addition to traditional features such as lines, edges, and corners. Putting these together, we can see in Fig. 4 that as the spikes propagate through the network, they progress from being more local temporal-features, to more and more disperse; this would seem to correlate with the hypothesis that information is stored in a distributive fashion in the brain. These results would seem to aid credence to the hypothesis that temporally-precise firing can encode information about an input stimulus in a two-dimensional multi-layered network, because the synaptic weights encode the input stimulus.

When inhibitory STDP is added, the entropy seems to be inverted, at least in the second row: where the entropy is high for excitatory neurons it is low for inhibitory ones. We did not notice any difference in the firing of neural groups with the introduction of iSTDP, which is reflected in the relative similarity of the excitatory entropy between figures 4 and 5; this could either be because of the weight cutoff, or because inhibition works less in later layers. For both observations, they will be an interesting avenue of investigation for future work.

3.1 Future Directions

A realistic first step would be to perform simple machine learning techniques on the last layer of the network, such as using a linear SVM, to see if the features encoded in the last layer can

predict the class of input. Knowing that the network organizes its weights around the input stimulus, the next step is to integrate these networks into larger networks that could use the output of the final layers for further processing. For example, the last layer could be connected to a 1-D layer of lateral-inhibition neurons, which would act as a winner-take-all classification layer. Such neurons could be linked up to motor commands, for example moving a hand, and the network could be used to explore and validate any role STDP has in motor learning in development, both as investigation into humans as well as to train dexterous robots for quick and smooth maneuvers.

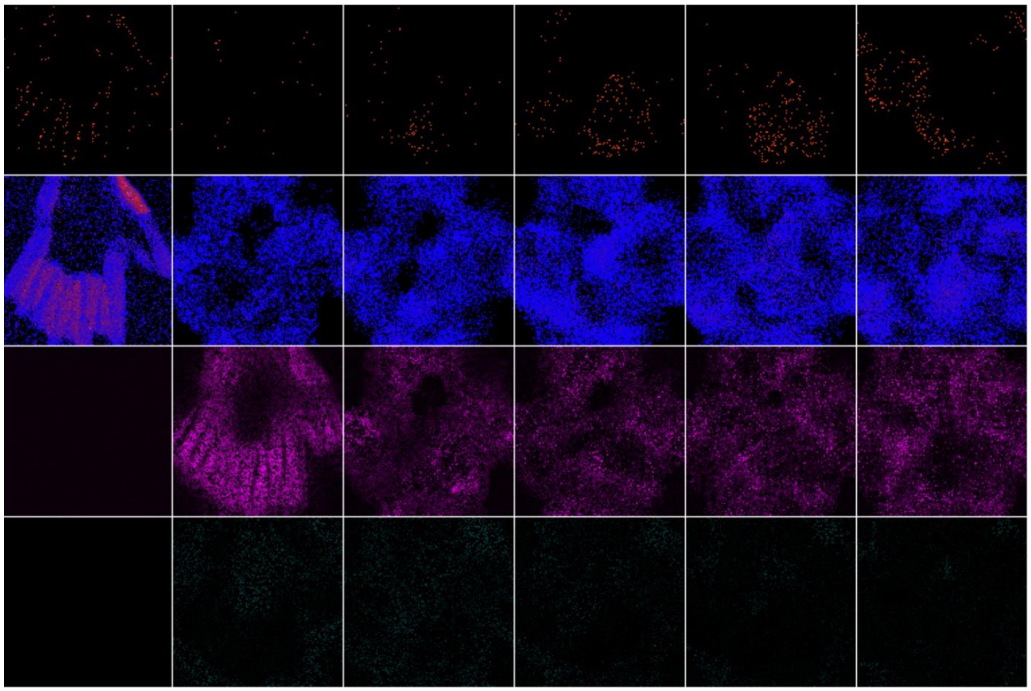


Figure 5: A display of the network running with iSTDP, inhibitory entropy is in the bottom row.

4 Conclusion

Here we have demonstrated a novel semi-recurrent feed-forward dynamical network and provided evidence that it is encoding information about temporal stimuli. Our hope is that this will lead to future work that takes advantage of the potential of temporal coding in spiking networks.

Conflict of Interests

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Acknowledgments

This work was supported partly by the University of Maryland COMBINE program and NSF award DGE-1632976. The authors would like to also thank Mrs. Greg Davis and Jesse Milzman at the University of Maryland for thoughtful conversation and advice.

Event-based attention and tracking on neuromorphic hardware

Alpha Renner
Institute of Neuroinformatics
University of Zurich and ETH Zurich
alpren@ini.uzh.ch

Matthew Evanusa
University of Maryland, MD, USA
matthew.evanusa@gmail.com

Yulia Sandamirskaya
Institute of Neuroinformatics
University of Zurich and ETH Zurich ysandamirskaya@ini.uzh.ch

Abstract

We present a fully event-driven vision and processing system for selective attention and tracking, realized on a neuromorphic processor Loihi interfaced to an event-based Dynamic Vision Sensor DAVIS. The attention mechanism is realized as a recurrent spiking neural network that implements attractor-dynamics of dynamic neural fields. We demonstrate capability of the system to create sustained activation that supports object tracking when distractors are present or when the object slows down or stops, reducing the number of generated events.

1. Introduction

Event-based sensors send out data packages, or events, from each pixel asynchronously when the pixel detects a local brightness change, rather than reading all pixels and sending out frames at a constant rate. Such event-based sensing allows us to perform some vision tasks extremely efficiently, reducing the amount of required computation, transmitted data, and power consumption. Many event-based vision pipelines and architectures have been developed over the last decade [1, 2], which address such vision tasks as stereo vision [3], 3D pose estimation [4, 5], or optical flow [6]. These event-based pipelines are typically implemented on conventional Von Neumann computer architectures. While such implementations try to make the best out of the event-driven nature of the sensor output, they cannot fully utilize its advantages: the clocked, sequential operation of a CPU, as well as separation between memory and processor stay in contrast to the highly parallel asynchronous temporal stream of events coming from an event-based sensor.

Neuromorphic hardware, in contrast to the conventional CPU, offers a massively parallel computing substrate that is inherently event-based, which matches the processing paradigm of event-based sensors [7, 8, 9, 10]. We aim to develop neuronal architectures that solve different tasks on such neuromorphic devices taking full advantage of the event-driven computation. In this work, we target a vision task of object tracking; in particular we show how a simple attention network can be configured on a neuromorphic hardware to select one object in an event-based input stream and to track this object in presence of equally salient distractors. While similar principles to the one realized here have been used in early days of neuromorphic engineering to design a dedicated spiking neuromorphic chip for attention [11], here we present its realization on a generic neuromorphic device Loihi that can also support other vision, cognitive, and motor control tasks [8].

Neuromorphic processors emulate dynamics of biological spiking neurons in hardware and thus allow us to run spiking neural network architectures in real-time, with a small energy footprint, and in small form-factor devices, making them a promising computing platform for event-based vision [7, 8]. However, most neuromorphic devices target offline computation, with applications in either computational neuroscience [10, 12] or data processing [9]. The Kapoho Bay – a USB-stick form-factor version of the Intel's latest neuromorphic research platform Loihi [8] – is among neuromorphic systems that offer a direct AddressEvent Representation (AER) interface to event-based

sensors [13]. This allows us to build a setup, in which a neuromorphic camera DAVIS [14] can be directly interfaced to Loihi, stimulating on-chip neurons configured in a network that can solve vision tasks. Here, we focus in particular on the task of object-centered attention and tracking.

While the event-based output of a DAVIS camera singles out a fast-moving object easily, two capabilities need additional processing: the ability to suppress distractors even if their salience changes and at times surpasses that of the target object and; keeping track of an object if it slows down or even stops. To gain these capabilities, the system needs a mechanism to hold a memory of the object's location in the field of view. While such a memory mechanism can be realized on a conventional computing system, doing so would alleviate the advantages of the low-power event-based computing. Here we explore a setup in which an event-based sensor is interfaced to an event-based processor running a recurrent neural network that is capable of creating memory states based on the incoming events.

Feed-forward artificial neural networks (ANNs) are stateless – they merely transfer inputs to outputs and if they do not receive input, the activity in the network fades away. In a spike-based network with integrate-and-fire neurons, the activity decays with a time-constant of neuronal dynamics, in a conventional ANN, even more radically, on the next clock cycle. In order to create a memory state, recurrence in the network is needed. A well-known model for working memory that has been studied in computational neuroscience and cognitive science is a Dynamic Neural Field (DNF) [15] – a neural population-based model. The DNF is a dynamical system that can be realized as a recurrent neural network with attractor dynamics, created by configuring a population of neurons with a winner-take-all connectivity [16]. In this connectivity pattern, neurons that encode similar values have an excitatory connection and neurons that encode different values – an inhibitory one. This simple connectivity pattern performs a selective amplification of a noisy input and, in an extreme case of strong interaction, can create sustained activation patterns that are kept active even if the initial input ceases completely. This property has been used as a model of working memory [15].

DNFs have been used previously to realize object tracking with on SCAMP – a smart camera with an in-focalplane processor array [17]. Here, we demonstrate object tracking with DNFs in an event-based setting using a spiking neuromorphic device Loihi.

2. The hardware setup

2.1. Neuromorphic Device Loihi

Intel Neuromorphic Computing Lab designed the neuromorphic research chip Loihi, in which spiking neural network models can be simulated in real-time efficiently [8]. The chip consists of a mesh of 128 neuromorphic cores, three embedded x86 processor cores, and an off-chip communication interface that allows to scale up architectures to multiple Loihi devices. Compartments are the main building blocks used to configure both single- and multicompartment neurons. In this work we only use single compartment neurons.

The external input to a network on Loihi is provided through spike generators. Spike generators are ports connected to compartments that can generate spikes at precise time-steps. Loihi provides an instrument for measuring the variables and sending them off the chip using "probes". For compartments it is possible to define probes to measure spike events, neuron's membrane voltage and input current. For the connections, probes can measure multiple synaptic variables, including weight, pre-synaptic and post-synaptic traces. Probing, however, affects the performance of the chip.

Loihi's Python NxSDK-0.8.0 API allows us to implement SNNs on the chip [18]. The NxNet API provides ways to define a graph of neurons and synapses and configure their parameters (such as decay time constants, spike impulse values, synaptic weights, refractory delays, spiking thresholds), inject external stimulus into the network, implement custom learning rules, and monitor and modify the network during runtime.

2.2. Dynamic Vision Sensor DAVIS

In this work, we used a Dynamic Vision Sensor type of a camera, the DAVIS240C [14]. The DAVIS camera emulates the dynamics of biological retinal cells in silicon using mixed-signal analog/digital technologies. There are 240×180 pixels integrated on the chip. Each pixel independently detects the brightness change in a small area of the visual

scene and emits an event if the brightness change passes a positive ("on" event) or a negative ("off" event) threshold. Each event is a digital data packet that carries the address of the pixel, the polarity of the detected brightness change, and the time of the event using Address Event Representation. Due to its high dynamic range, the sensor captures moving objects in its visual field in a wide range of lighting conditions.

To connect DAVIS to Loihi, the direct parallel AER interface on the device can be used. The events are captured and distributed to neurons on the neuromorphic cores through the embedded FPGA and x86 processors. For reproducibility, in this work, we use recorded spikes that we feed into Loihi using a spike generator from the NxNet API to generate some of the plots. The network was also tested with the direct AER interface between the DAVIS and Loihi.

3. Attractor Dynamics for Object Tracking: The Dynamic Neural Fields

A Dynamic Neural Field is a mathematical model that was derived to describe activity of large homogeneous populations of biological neurons [15]. The connectivity pattern in such a neuronal population is shown in Fig. 1. First, neurons are "aligned" according to a feature which they are

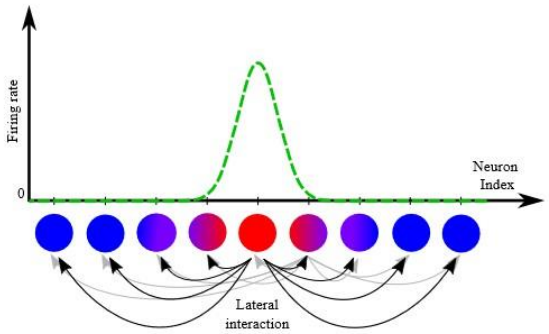


Figure 1. Schematic representation of a 1-dimensional winner take-all dynamic neural field. The lateral connections are all-to-all and the synaptic weights are defined by the kernel function that depends on the distance between the pre- and post-synaptic neurons.

sensitive to (here, the position in the camera's field of view). Second, neurons that are sensitive to similar values of the feature (i.e. are close to each other on this behavioral space) are connected via excitatory (positive weight) connections, and neurons that are sensitive to dissimilar features inhibit each other (negative-weight connections). This soft winner take-all connectivity pattern, combined with a non-linearity of the neuronal activation function, leads to formation of an attractor state in a DNF neuronal population. In particular, DNFs form a so-called bump-attractor – a localized activity peak centered over a salient value of the behavioral variable (green line in Fig. 1). Such bump-attractor networks have been used in the past both to explain activity patterns in biological neural networks and to build artificial cognitive systems for robot control [19, 15]. To realize DNF dynamics on Loihi, we create a group of compartments that are connected with synapses with weights matching the "Mexican hat" connectivity kernel.

The output of the DNF population is computed as a population vector using the instantaneous firing rate of neurons that is inversely proportional to the inter-spike intervals.

4. Results

4.1. Attractor dynamics on a neuromorphic Chip

First, we demonstrate the properties of a DNF realized in a spiking neural network on Loihi that are used in the object-tracking application. We have configured a small population of 12 neurons in a winner-take-all fashion, as shown in Fig. 1. We used two sets of parameters to demonstrate two dynamical configurations of the DNF: a selective input-driven regime, shown in Fig. 3a, b, c and a selfsustained regime shown in Fig. 3d. In each subfigure, the upper plot shows spikes from the spike generators that send

input to Loihi, the middle plot shows the output spikes from the DNF population, and the lower plot shows population

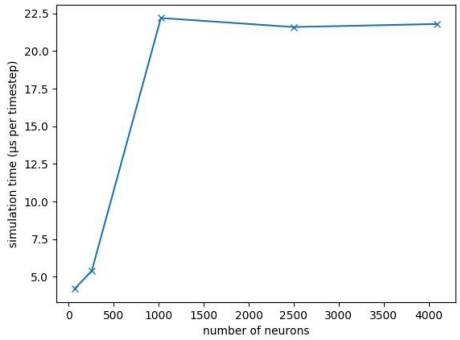


Figure 2. Performance evaluation. activity vector of the DNF population (the position of the mean of neuronal activity) over time of the experiment.

Fig. 3a shows that a DNF configured in a selective regime selects one of the input bumps in the case of a bimodal input distribution. For each pair of input bumps with equal strength (average firing rate), the DNF selects one of them randomly. Fig. 3b shows behavior of the same DNF population for an input sequence, in which one of the input bumps arrives first, is selected and then stabilized by the lateral interactions in the neuronal population. In this configuration, the second input bump is rejected by the DNF population and does not lead to any activity of the neurons in the respective region.

Fig. 3c and d contrast behavior of the DNF configured to be in an input-driven regime (c) versus in a selfsustained regime (d). For the same input, the input-driven DNF (Fig. 3c) follows the input activity, only rejecting noise around the activity bumps. The self-sustained DNF (Fig. 3d) keeps the position of the selected object and ignores input at different locations unless it is spatially proximal to the current activity bump. Thus, it shows tracking behavior.

To configure the two DNFs, we used the parameters on Loihi listed in the Table 1. In particular, we use Gaussian shaped connectivity profile for lateral connections in the WTA (of amplitude “Excitatory weight” and spread “Connectivity kernel σ”) and a direct global inhibition. Input and background-noise are generated as Poisson spikes.

4.2. Performance evaluation

Fig. 2 shows the duration of simulation on Loihi per time step depending on the number of neurons in a 2D DNF population with a single self-sustained bump without spike generators (apart from the initial one creating the bump) or probing. Neurons are distributed over the 2x128 cores. One can observe that the simulation time per time step stays on the level of 20μs for network sizes above 1000 neurons[12].

[12] Although quite fast already, the simulation time depends on the details of the network implementation and can be further optimized.

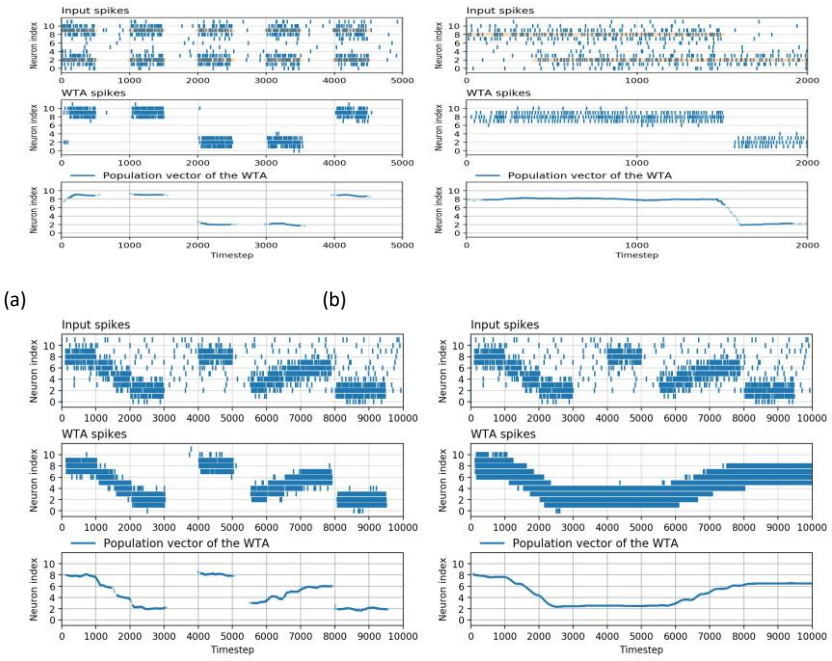


Figure 3. The plots show activity of spiking neurons on the Loihi chip, configured as a DNF in an input-driven (a, b, c) and self-sustained (d) regime. In each subfigure, the top plot shows input spikes generated on the computer, the middle plot shows output of the DNF population on Loihi, and the bottom plot shows the population vector (mean of the activity) of the DNF population.

4.3. Event-based attention and tracking

The tracking network consists of two two-dimensional WTA/DNF layers with 64x64 neurons each. The first layer has excitatory connections at a level at which peaks are self-sustained and weak global inhibition so that multiple bumps can form. Bumps form at the locations of the on-event input and follow the on-events when objects move over the field of view of the DVS. The off-events inhibit neurons in this population, decreasing activity in the bumps. This facilitates fast moving bumps avoiding a tail in the activation pattern. The second WTA layer receives input from the first layer through one-to-one connections. This layer has strong self-excitation and strong global inhibition which lead to the selection of a single bump. To this layer, we provide an excitatory initial input, cuing one of the objects in the beginning of the DVS stimulation; feature-based cues can also be used here [15] (Chapter 5). The WTA forms an activity bump over the selected object, which is moved by the excitatory input from the input layer when the selected object moves. In these experiments, parameters listed in

Parameter	Input driven	Selfsustained
Voltage threshold	$3000*2^6$	$3000*2^6$
Voltage decay time constant	150 ts	150 ts
Current decay time constant	10 ts	10 ts
Connectivity kernel σ	1.5	1.5
Self-excitation	no	yes
Excitatory weight	200	150
Global inhibitory weight	-160	-75
Input weight	200	200
Input firing rate (Poisson)	60 Hz	60 Hz

Noise firing rate (Poisson)	2 Hz	2 Hz

Table 1. DNF parameters used to produce plots in Fig. 3

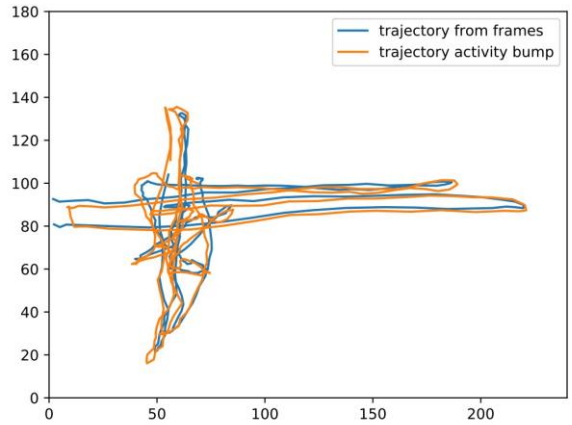


Figure 4. Trajectory of the selected object (star), obtained from the activity bump in the DNF on Loihi and from frames, used as ground truth. The average distance between the points in the two trajectories over all frames is 3.5 DAVIS pixels. The distance was calculated with an offset between frames- and events-trajectory of 15 ms, where the distance is minimal.

Table 2 were used. In particular, both DNF layers are self connected using a Mexican hat connectivity kernel specified by a difference of two Gaussians (with amplitude and σ of the excitatory and inhibitory kernels); a global inhibitory group of neurons is used to provide global inhibition. Every neuron in the inhibitory group has a given probability to be connected to any neuron in the excitatory layer.

Fig. 5 shows performance of the two-dimensional DNF on the Loihi chip in a tracking experiment. Here, the shapes_translation dataset [20] is used that contains a number of objects drawn on a wall in front of a moving DVS. Plots in Fig. 5b show input that is sent to Loihi over the course of the experiment: the DVS events are emitted at the edges of the objects.

Fig. 5c shows activity of the first, non-selective DNF layer: activity in this layer forms peaks over all objects, stabilizing this activity in moments with reduced motion and

Parameter	value
Voltage threshold	$640*2^6$
Voltage threshold global inhibition	$896*2^6$
Voltage decay time constant	20 ts
Current decay time constant	20 ts
Input connectivity kernel σ	1.5
Excitation kernel σ	2
Inhibition kernel σ	4
Excitatory weight non-selective	152
Inhibitory weight non-selective	-41
Excitatory weight selective	230
Inhibitory weight selective	-41
Excitatory weight to global inhibition	5
Global inhibitory weight non-selective	-20
Global inhibitory weight selective	-90
Excitatory weight non-selective to selective (1:1 connectivity)	740

Input weight on events	70
Input weight off events	-50
Refractory period	12 ts
Refractory period global inhibition	7 ts
Connection probability to/from global inhibition	0.6
Number of global inhibitory neurons	40

Table 2. DNF parameters used to produce plots in Fig. 5 and 6, if layer is not specified, the parameter applies to both.

weaker DVS output. Fig. 5d shows activity of the output layer of the tracking DNF. Here, one of the objects (the star) is selected (by a local boost to this layer in the beginning of the simulation) and is tracked throughout the experiment, despite presence of the distractors.

To obtain the spike plots, the spikes were filtered with a 50ms rectangular filter and snapshots were taken at regular intervals within the simulation of 6500 time steps. DAVIS input events to the first layer were down-sampled and binned into 1ms per time step (events that exceeded one event per bin, were discarded), i.e. the 6500 time steps correspond to 6.5s of DAVIS input.

Fig. 4 shows the trajectory of the selected object that is extracted from the activity of the output layer of the DNF model and the ground truth extracted from the input frames. The blue trace shows the center of the star object extracted from the DAVIS frames by thresholding (ground truth). The orange trace shows mean (i.e. population vector) of the instantaneous firing rate of neurons in the second (output) layer of the network (i.e. the middle of the tracked bump). Instantaneous firing rates are estimated based on the interspike intervals.

The DVS input to the network was down-sampled to 64x64 neurons, the trajectory was up-sampled to the DAVIS resolution of 240x180. The mean error was calculated as the mean of all distances between the locations of the bump activity and the locations of the frame based extraction at the timesteps of the DAVIS frames and amounts to 3.5 DAVIS pixels.[13]

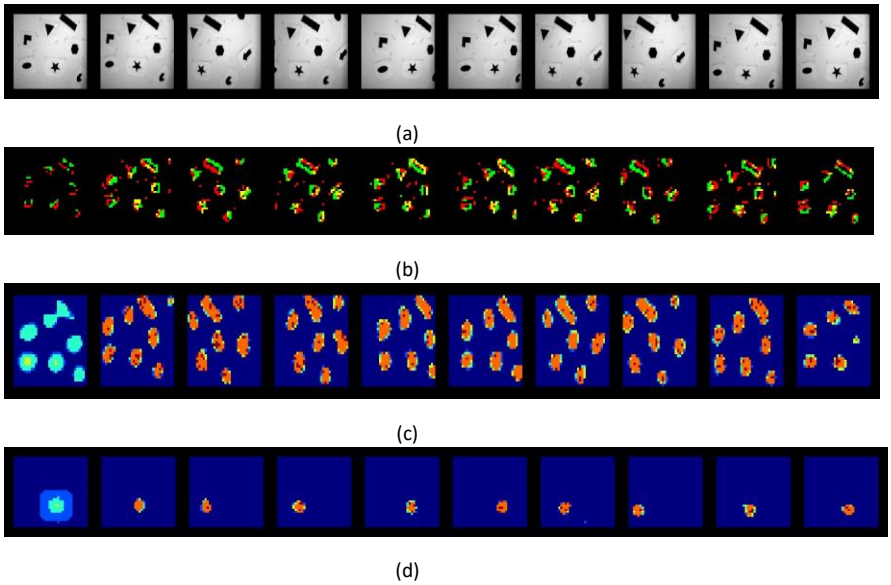


Figure 5. Object tracking experiment: (a) snapshots of input DAVIS frames (top); (b) DAVIS on (green) and off (red) events binned into
10ms frame (second from top); (c) Firing rate of first non-selective WTA layer on Loihi (third from top); and (d) second, selective WTA layer on Loihi (bottom).

Fig. 6 shows our second tracking experiment. Here, a ring with five identical objects is rotating in front of the DAVIS camera. The first layer of the tracking SNN architecture creates activity bumps for all five objects, while the second layer (bottom plots) only tracks the single object, selected by a localized activity boost in the beginning of the experiment (first pane in the plot). The same parameters were used here as for Fig. 5. The length

[13] The Figure was generated using a Brian2 simulation of the equations implemented in Loihi, as it is currently impossible to probe a large network for that many time steps, however, performance of the network can be observed in a live demo.

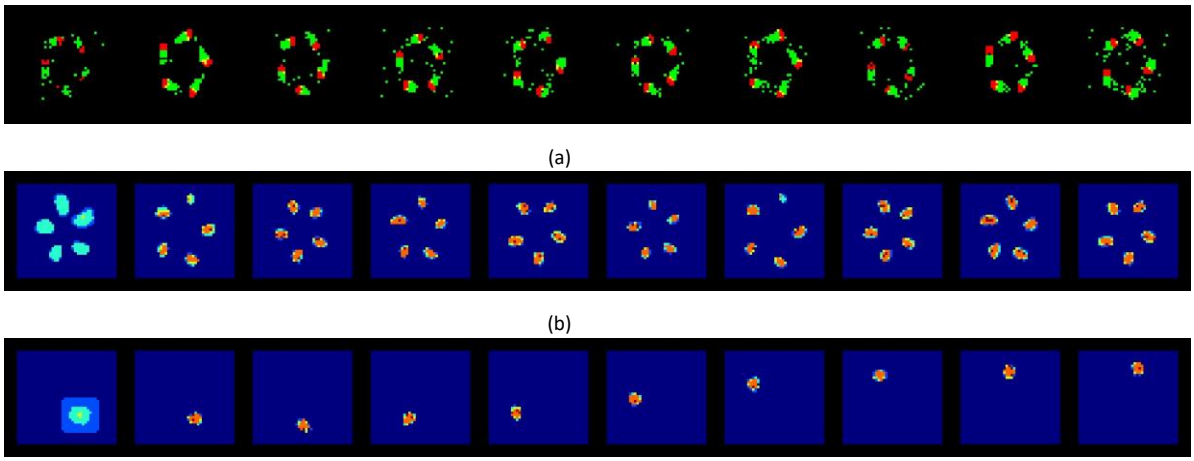
of simulation on Loihi was 3000 timesteps here. DAVIS input events were binned into 0.5ms per timestep, i.e. the simulation corresponds to 1.5s of DAVIS input.

5. Conclusion

In this work, we have shown for the first time a setup that interfaces an event-based camera DAVIS with the spiking neuromorphic system Loihi, creating a purely event-driven sensing and processing system. We have implemented a simple attention and tracking network on Loihi that allows to select a single object out of a number of moving objects in the visual field and track this object, even in cases when movement stops and the event stream is interrupted. Full evaluation of the system in terms of tracking speed and quality, as well as power efficiency and robustness is target of our current work and will be reported shortly, while functioning of the system can be observed in a live demonstration during the workshop.

Acknowledgements

We would like to thank Dr. Julien Martel and the Intel Neuromorphic Computing Lab for their help with the hardware and software setup used in this work. This project has received funding from the SNSF project Ambizione (PZ00P2 168183) and a ZNZ Fellowship from the Neuroscience Center Zurich.



(a)

(b)

(c)

igure 6. Object tracking experiment 2: DAVIS on (green) and off (red) events binned into 10ms frames (top); firing rate of the first on-selective WTA layer on Loihi (middle); output of the second, selective WTA layer on Loihi (bottom).

DOT-VAE: Disentangling One Factor at a Time

Vaishnavi Patil, Matthew Evanusa, and Joseph JaJa

University of Maryland, College Park, MD 20740, USA {vspatil, mevanusa, josephj}@umd.edu

Abstract. As we enter the era of machine learning characterized by an overabundance of data, discovery, organization, and interpretation of the data in an *unsupervised* manner becomes a critical need. One promising approach to this endeavor is the problem of *Disentanglement*, which aims at learning the underlying generative latent factors, called the factors of variation, of the data and encoding them in disjoint latent representations. Recent advances have made efforts to solve this problem for synthetic datasets generated by a fixed set of independent factors of variation. Here, we propose to extend this to real-world datasets with a countable number of factors of variations. We propose a novel framework which augments the latent space of a Variational Autoencoders with a disentangled space and is trained using a Wake-Sleep-inspired twostep algorithm for unsupervised disentanglement. Our network learns to disentangle interpretable, independent factors from the data "one at a time", and encode it in different dimensions of the disentangled latent space, while making no prior assumptions about the number of factors or their joint distribution. We demonstrate its quantitative and qualitative effectiveness by evaluating the latent representations learned on two synthetic benchmark datasets; DSprites and 3DShapes and on a real datasets CelebA.

Keywords: Deep learning · Representation learning · Unsupervised Disentanglement.

1 Introduction

Deep learning models, which are now widely adopted across multiple Artificial Intelligence tasks ranging from vision to music generation to game playing [16, 23,22], owe their success to their ability to learn representations from the data rather than requiring hand-crafted features. However, this self-learning of abstract representations comes at the known cost of the resulting representations being cryptic and inscrutable to human observers [8]. A more comprehensive representation of the data where the essential indivisible, semantic concepts are encoded in structurally disentangled parts could lead to successful domain adaptation and transfer learning [1] and facilitate robust downstream learning more effectively [27]. Learning these latent representations from the data alone without the need of laborious labeling by human observers constitutes the problem

of *Unsupervised Disentanglement*. In this work, we attempt to address the problem of unsupervised disentanglement via a novel Variational Autoencoder based framework and training algorithm.

Though there is no commonly accepted formalized notion of disentanglement or validation metrics [11], recent works have characterized disentangled representations, based on natural intuition. This intuition by [1] states that a disentangled representation is a representation of the data which encodes each *factor of variation* in disjoint sets of the latent representation. [21] state further that a change in a single factor of variation produces a change in only a subset of the learned latent representation which corresponds to that factor. Here, a factor of variation is an abstract human defined concept that assumes different values for different examples in the dataset. This intuition is closely related to the independent mechanisms assumption [26] which renders the informative factors as components of a causal mechanism. This assumption allows interventions on one factor without affecting the other factors or the representations corresponding to the other factors and thus can be independently controlled. In our work we use independent interventions on the learned disentangled representations, which encode the different factors, to generate samples and restrict the differences in corresponding representations, of the data and the sample, pertaining to that factor. This process of using interventions and generating new samples resembles the sleep phase of the wake-sleep algorithm.

Most current Variational Autoencoder (VAE) based state-of-the-art (SOTA) make the implicit assumption that there are a fixed number of independent factors, common for all the data points in the dataset. However in real datasets, in addition to the independent factors common to all points in the dataset, there might also be some correlated, noisy factors pertinent to only certain data points. While the approaches based on Generative Adversarial Networks (GAN) do not make an assumption, they learn only a subset of the disentangled factors, whose number is heuristically chosen. We believe however that one of the main goals of disentanglement is to glean insights to the data, and the number of factors of variation is generally one that we do not have access to. To this end, our method augments the entangled latent space of a VAE with a disentangled latent code, and iteratively encodes each factor, common to all the data points, in a single disentangled code using interventions. This process allows our model to learn any number of factors, without prior knowledge or ”hardcoding”, thus making it better suited for real datasets.

1.1 Main Contributions

Our contributions in the proposed work are:

- We introduce a novel, completely unsupervised method for solving disentanglement, which offers the mode-covering properties of a VAE along with the interpretability of the factors afforded by the GANs, to better encode the factors of variations in the disentangled code, while encoding the other informative factors in the entangled representations.
- Our proposed model is the first unsupervised method that is capable of learning an arbitrary number of latent factors via iterative unsupervised interventions in the latent space.
- We test and evaluate our algorithm on two standard datasets and across multiple quantitative SOTA metrics and qualitatively on one dataset. Our qualitative empirical results on synthetic datasets show that our model successfully disentangles independent factors. Across all quantitative metrics, our model generally outperforms existing methods for unsupervised disentanglement.

2 Disentangling One Factor at a Time using Interventions

We base our framework on the VAE which assumes that data x is generated from a set of latent features $z \in \mathbb{R}^d$ with a prior $p(z)$. A generator $p_\theta(x|z)$ maps the latent features to the high-dimensional data x. The generator, modeled as a neural network, is trained to maximize the total log-likelihood of the data, $\log p_\theta(X)$. However, due to the intractability of calculating the exact loglikelihood, the VAE instead maximizes an evidence lower-bound

(ELBO) using an approximate posterior distribution $q_\phi(z|x)$ modeled by an encoder neural network. For given data, this encoder projects the data to a lower-dimensional representation z, such that the data can be reconstructed by the generator given only the representation. Without any structural constraints, the dimensions of the inferred latent representation z are informative but entangled. **Disjoint Latent Sets:** To encode the factors of variation in an interpretable way, following [6] we augment the unstructured variables z with a set of structured variables $c = \{c_1, c_2, \cdots, c_K\}$ each of which is tasked with disentangling an independent semantic attribute of the data. We train our network to systematically discern the meaningful latent factors shared across the dataset into c, from the entangled representation z, both of which are important to maximize the log-likelihood of the data. However, it is only the most informative, common factors of variation encoded in c that we are interested in, as the remaining factors in z are confounded and contain the 'noise' in the dataset, i.e., features that only a few data samples contain.

The generator is now conditioned both on the disentangled latent codes and the entangled latent space (c,z) and describes a causal mechanism [27], where each causal component c_k is independently controllable and a change in a particular index k has no effect on any other index $c_j (j \neq k)$. Manipulating each c_k should correspond to a distinct, semantic change, corresponding to the factor encoded, in the generated sample, without entangling with changes effected by the other factors or with the entangled code z. To this end, we perform *interventions* as proposed in [27] (described in detail below) that go into the latent code, change a single dimension c_k of the disentangled code c while keeping the

0 Code available at https://github.com/DOTFactor/DOTFactor

other dimensions the same. We generate data from this intervened latent and then constrain the model to reconstruct the exact change.

To encourage the generator to make distinct changes for each c_k that is manipulated during interventions, we re-encode the data to recover the manipulated latent representation that was used to generate the data. Thus if the code c_i was manipulated, while keeping the rest $c_j (j \neq i)$ and z unchanged, the encoding of the corresponding generated data must reflect a change only in the manipulated code c_i.

Adversarial Latent Network: Moreover, in order to ensure that interpretable factors are encoded in the disentangled code c, the changes effected by each c_k must correspond to a semantic change in the generated data corresponding to a change which could be affected by a factor of variation. For this, the data generated from the disentangled code manipulation during interventions must lie in the true data distribution. To ensure this we train a discriminator, like in [14], in the latent space such that the latent representations after manipulations continue to lie in the distribution of the encodings of the true data. The encoder is trained to reduce the distance between the distribution of the representations, used by the generator to reconstruct the data, and the distribution of the representations after they have been intervened on. This in effect helps us train a generative model which can be conditioned to generate realistic, new samples with any desired values for the different latent factor. Figure 1 shows the overall framework which is trained with a two-step algorithm inspired by the wake-sleep algorithm.

Model Structure and Learning: In this section we will detail the training method, and how to combine the architectural components described earlier. We augment the latent space z to include a disentangled set of latent codes c with prior $p(c) = \prod_{i=1}^{K} p(c_i)$, where $p(c_i) \sim \mathcal{N}(0,1)$. The encoder $q_\phi(c,z|x)$ is split into two to model the distributions of the two latent variables as $q_{\phi_1}(z|x)$ and $q_{\phi_2}(c|x)$ respectively. Under this augmented space the evidence lower-bound of the data log-likelihood is as follows:

$$\mathcal{L}_{\theta,\phi} = -\mathbb{E}_{q(x)}[\mathbb{E}_{q_\phi(c,z|x)}[\log p_\theta(x|c,z)] + KL[q_{\phi_1}(z|x)||p(z)] + KL[q_{\phi_2}(c|x)||p(c)]] \quad (1)$$

where $q(x)$ is the empirical training data distribution. The first term in the above objective function minimizes the error of reconstructing the higher-dimensional data from its lower-dimensional representations. The KL divergence in the second and the third terms ensure that posterior distribution learned by the model is close to the uninformative prior distribution, and can be sampled from for generating new samples. These three terms together create an information bottleneck in the latent space where only the information relevant to recover the data is encoded in the representations. This ensures that all the informative factors of variations are encoded in the representations either in z or in c.

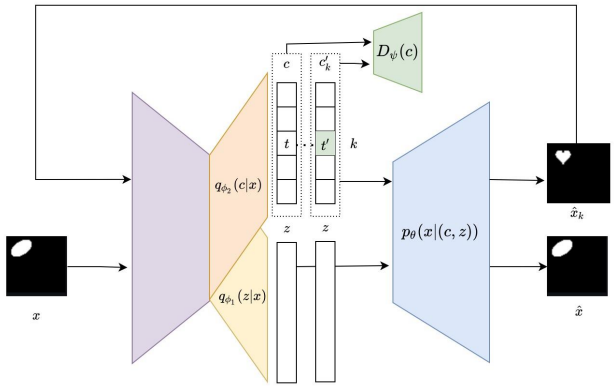


Fig.1: The full architecture of Dot-VAE. Input data is fed into an encoder, which projects the data into two disjoint latent spaces, a disentangled c and entangled z. *Interventions* change the value t of a single disentangled latent at index k to a new value t' while keeping the other latents unchanged to get the intervened representation (c'_k, z). The adversarial network ensures that the distribution of the real representations c and the intervened representations c'_k is close to each other. The decoder maps the intervened latents to generated data $\hat{x}_k$. This new generated data is passed back through the encoder to train the generator to make distinct and noticeable changes.

Learning One Factor at a Time Since we want each disentangled latent code $c_k \in c$ to encode a factor of variation, changes in the value of a latent code c_k, while keeping the others the same, should bring about semantic changes in the generated samples corresponding *only* to the encoded factor. The changes should be such that when the original data and the generated data (after interventions) are compared, the encoder must be able to discern (1) the index of the changed latent variable, and (2) by how much it was changed. We enforce these conditions by performing interventions on one code at a time and reconstructing the latents of the interventions, sequentially. At the beginning of training, we start with interventions on the first element of the disentangled space c_1. Once the objective in Equation 1 saturates for interventions with c_1, we shift the interventions to the next element in the set, c_2 along with c_1. As the training proceeds, more dimensions of c are intervened upon together and the network learns to encode the confounding information into z, while keeping only the disentangled information in c. For synthetic datasets with a fixed number of factors of variations, we continue iterating through the elements of c until the entangled part of the representation z encodes no information i.e. $KL(q_{\phi_1}(z|x)||p(z)) = 0$. For real-world datasets we can continue the iteration until we find a desired number of factors.

Formally, given a batch B of examples from the data distribution $\{x^i\}_{i=1}^B$, which we write as $\{x^i\}^B$ for brevity, we first encode the batch to find their

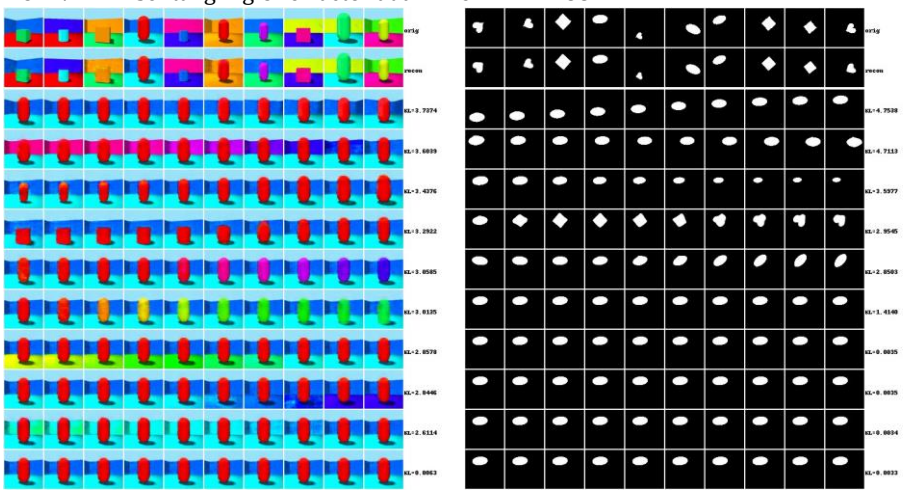


Fig.2: Latent traversals for the disentangled code *c* for 3Dshapes (Left) and Dsprites dataset (Right). Our method disentangles the informative factors and encodes them in the different dimensions of *c*. The top two rows are the original and reconstructed images, respectively. Each row below the second corresponds to a traversal. *3DShapes (Left):* Each row corresponds to a distinct independent factor. Rows three through ten correspond to orientation, wall hue, size, scale, object hue (two rows), floor color (two rows) respectively. This covers all of the exact factors of variation. In the last row, we can see the model discerned it had discovered all the factors of variation, and need not encode anything. *dSprites: (Right)* Similarly, in dSprites we can also see that the model discerns the factors in rows 3 through 7: x, y coordinate, size, shape, and orientation.

latent representation $\{(c^i,z^i) \sim q(c,z|x^i)\}^B$. We then select an index $k \in [K]$ and proceed to intervene/change the value of the element c_k, which we refer to as $\{t^i\}^B$, of the disentangled code, to $\{t'^i\}^B$ (i.e. to the value of another example $t^l (l \neq i)$ at the same dimension c_k), for all the representations in the batch, while keeping the other disentangled codes $c_j (j \neq k)$ and the entangled representations z the same. We change the value only for that code to obtain our intervened latent representation $\{c'^i_k = (t'^i, c^i_j (j \neq k))\}^B$. We then use these intervened representations to generate a batch of samples $\{\hat{x}^i_k \sim p_\theta(x|c'^i_k, z^i)\}^B$.

The generated samples $\{\hat{x}^i_k\}^B$ must differ from the data examples $\{x^i\}^B$ only in the factor corresponding to one encoded in c_k. To ensure this, we reencode the generated samples to reconstruct the intervened latent representation $\{\hat{c}^i_k \sim q_\phi(c,z|\hat{x}^i_k)\}^B$ and optimize the generator to minimize the reconstruction cost as follows:

$$L^\theta = \frac{1}{B} \sum_{i=1}^{B} \|\hat{c}^i_k - c'^i_k\|^2 \qquad (2)$$

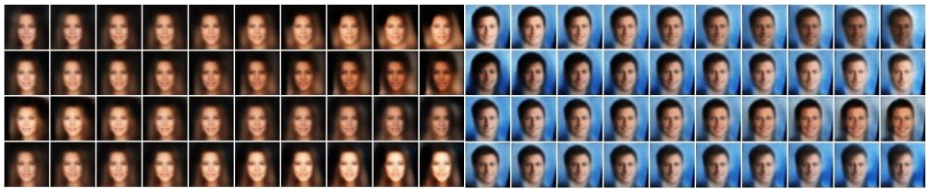


Fig.3: Traversals over latent space *c* for the *CelebA* real-world celebrity dataset. The dataset consists of cropped headshots of various celebrities. Given a seed image, we perform a traversal across various latents. Our model

successfully produces reconstructions that correspond to identifiable factors. Like for synthetic data, some of the latents remain unused; shown are exemplar latents that encoded factors of variation. The model learns many more factors as compared to previous methods such as hair color, skin tone, lighting, etc.

Since we enforce the model to exactly reconstruct the intervened latent representation, from the generated samples alone, the generator can only make the distinct and noticeable change corresponding to the factor encoded in the intervened disentangled code after converging.

Moreover, we want to ensure that manipulations in the disentangled code c translate to semantic changes in the generated samples thus making them interpretable. To this effect the generator should be constrained to map the intervened representations to examples which lie strictly in the true data manifold. Instead we constrain the encoder to reduce the distance between the posterior distribution $q_{\phi 2}(c)$ and the intervened latent distribution $p(c')$. To this effect we train an adversarial network D_ψ to distinguish between the original representations c and the intervened representations c' by maximizing the following objective:

$$\mathrm{L}_\psi = \mathrm{E}_{p(c')}[\log(D_\psi(c'))] + \mathrm{E}_{c\sim q_{\phi 2}(c|x)}[\log(1 - D_\psi(c))] \quad (3)$$

The encoder is in turn trained to fool the discriminator by making the approximate posterior distribution of c indistinguishable from the intervened representations c' by minimizing the following object:

$$\mathrm{L}_\phi = \mathrm{E}_{c\sim q_{\phi 2}(c|x)}[\log(1 - D_\psi(c))] \quad (4)$$

Since the VAE objective constrains the generator to map the representations to the true data distribution, to minimize the reconstruction cost in the image space, the generator now also maps the intervened representations also to the true data distribution.

We combine the losses in {1, 2, 4} linearly to train the encoder and the generator to minimize the following objective:

$$\mathrm{L} = \mathrm{L}_{\theta,\phi} + \lambda \mathrm{L}_\phi + \gamma \mathrm{L}_\theta \quad (5)$$

The latent discriminator D_ψ is trained jointly with the other networks to maximize the objective L_ψ. For the first few epochs, we train the model to maximize Table 1: Comparisons of the popular disentanglement metrics on the dSprites dataset. A perfect disentanglement corresponds to 1.0 scores. For all scores, **higher is better**. The scores are averages over 10 runs with different random seeds. The values for the GAN-based models were taken from [19]. Our model outperforms all the VAE-based models while performing comparably with the GAN-based models. DOT-VAE[1] corresponds to the model where intervened values are the values for other samples for the same intervened index whereas DOT-VAE[2] corresponds to the intervened values sampled from the prior distribution.

Model	FactorVAE	DCI	MIG	BetaVAE	Modularity	Explicitness
β-VAE (24)	0.65 ± .11	0.18 ± .10	0.11	0.83 ± .30	0.82 ± .03	0.81
β-TCVAE	0.76 ± .18	0.30 ± .05	0.18	0.88 ± .18	0.85 ± .03	0.83
FactorVAE(40)	0.75 ± .12	0.26 ± .04	0.15	0.85 ± .15	0.81 ± .02	0.80
InfoGAN*	0.82 ± .01	0.60 ± .02	0.22	0.87 ± .01	0.94 ± .01	0.82
InfoGAN-CR*	**0.88 ± .01**	**0.71 ± .01**	0.37	0.95 ± .01	**0.96**	0.85
DOT-VAE[1] (ours)	0.77 ± .12	0.66 ± .06	**0.38**	0.95 ± .13	0.86 ± .05	**0.86**
DOT-VAE[2] (ours)	0.72 ± .15	**0.72 ± .06**	0.34	**0.97 ± .18**	0.82 ± 0.02	0.84

the likelihood with the discriminator loss and only start the interventions after a few epochs of training.

3 Related Work

Various authors have attempted to learn unsupervised disentangled representations using generative models in recent years. SOTA for unsupervised disentanglement learning can be broadly classified into two categories based on the type of generative model used; one via Variational Autoencoders (VAE) [15,24], and another via Generative Adversarial Networks (GAN) [9]. Techniques that use VAEs have been modifications of the base VAE architecture to further facilitate a structured latent code. The β-VAE [10] heavily penalize the KL divergence term thus forcing the learned posterior distribution $q_\phi(z|x)$ to be independent like the prior. AnnealedVAE [2] controls the capacity of the latent encoding to allow for the independent encoding of the factors. Factor-VAE [14] and β-TCVAE [3] penalize the total correlation of the aggregated posterior $q_\phi(z)$ using adversarial and statistical techniques respectively. DIP-VAE [17] forces the covariance matrix of the aggregated posterior $q(z)$ to be close to the identity matrix by method of moment matching. Other works improved the performance augmenting the latent space to include the discrete factors [5,13], and use optimization techniques based on annealing to encode information effectively in the discrete and continuous factors.

Models based on GANs explicitly condition the generator with a set of independent latent variables c (by concatenation with random noise z), and train the generator to generate data which has high mutual information with c. The most prominent work is InfoGAN [4] which learns disentangled, semantically meaningful representations by maximizing a lower bound on the intractable mutual information between the conditioning latent variables c and the generated samples $G(z,c)$. InfoGAN-CR [19] add a contrastive regularizer to the InfoGAN model, which is trained to predict the changes in the latent space given only the pairs of images. [29] augment their objective with a similar self-supervised learning task. In [20], the authors add orthogonal regularization to encourage independent representations. In addition, a major difference between the disjoint sets of InfoGAN and our method is that we explicitly learn the conditioning variables c, whereas for the GAN-based methods they are pre-determined. Moreover, we do not need to know the number of factors before-hand and instead learn them as we continue training.

Table 2: Quantitative comparisons of the disentanglement metrics on the 3DShapes dataset averaged over 10 runs with different random seeds. The results for other method have been taken from [21]. For all metrics, a higher value indicates a more disentangled latent space. As we can see, DOT-VAE (our model) outperforms most of previous models.

Model	FactorVAE	DCI	MIG	BetaVAE	Modularity	Explicitness
β-VAE(32)	0.82 ± .24	0.58 ± .40	0.34	0.99 ± .06	0.94 ± .18	0.87
β-TCVAE	0.83 ± .20	0.68 ± .46	0.27	**1.00 ± .07**	**0.96 ± .23**	0.92
FactorVAE(20)	0.81 ± .26	0.64 ± .27	0.29	0.98 ± .07	0.95 ± .28	0.94
DOT-VAE[1] (ours)	**0.95 ± .18**	**0.80 ± .30**	**0.42**	**1.00 ± .04**	0.94 ± .19	**0.95**

The authors of [27] introduced the concept of interventions to study the robustness of the learned representations under the Independent Mechanisms (IM) assumption [26], while we use the method of interventions while training the model to disentangle common factors from their entangled set. In [18] a VAE is used to disentangle representations and then the representations are passed into a GAN to generate high-fidelity images; our approach instead uses both VAEs and GANs for disentanglement. In [6] the latent space of entangled representations is conditioned with a disentangled code, which is learned in a supervised way using specific attribute discriminators. As far as the authors are aware, we are the first to split the latent dimension into entangled and disentangled in a completely unsupervised way, as well as the first to combine this with interventions, and with incremental learning.

4 Empirical Evaluation

For evaluation, we run experiments on three benchmark datasets: two synthetic datasets generated from independent ground truth factors of variation; dSprites [30] and 3DShapes [31]. Each datapoint in these datasets can be exactly described using their factors of generation. For a real-world dataset, with unknown

factors of variation, we run our model on the CelebA dataset [32]. This dataset has a copious amount of noise in each sample, and cannot neatly be described as independent factors.

Qualitative Analysis To demonstrate the disentangling ability of our model, we perform *traversals* across the latent space, and plot the resulting reconstruction, in line with the standard methods for disentanglement. A model has better disentanglement capability if a traversal across the latent space matches with a change in a single generative factor. We demonstrate this both for the synthetic datasets (Fig. 2), as well as the real-world sets (Fig. 3).

Quantitative Analysis We also evaluate the learned representations using three kinds of quantitative metrics as described in [28]. Specifically, we use the FactorVAE[14], the BetaVAE[10]; Mutual Information Gap (MIG)[3] and Modularity[25]; and the Disentanglement-Completeness-Informativeness (DCI) [7] and Explicitness [25]. Results for all metrics are found in Table 1 for dSprites Table 2 for 3DShapes. We used the implementation from [21] to calculate all metrics.

4.1 Results and Discussion

As we can see in Figure 2, our model successfully disentangles the dSprites and the 3DShapes dataset, with the traversals showing a change in that latent value corresponds to a change in the true factor of variation. Importantly, the ”entangled” latent space z encodes no information, and the model was able to, one at a time, disentangle the relevant factors of variation and encode them in c. This is in line with our hypothesis for synthetic datasets as the data can be exactly coded as the independent factors of variation. Our model is able to completely capture all of the factors of variation for dSprites as well as 3DShapes. We believe part of the reason for the success is the iterative, one at a time decomposition, which allows the network itself to discover how many factors there are, contrasted with prior methods that force the network to factorize the data in a given sized latent space. While the latent traversals show near perfect disentanglement, some of the metrics still report a lower value. We believe that this is because the metrics expect each factor to be completely described by a single latent variable. We do not restrict a factor to correspond to be encoded by only one latent variable as doing so amounts to complete unsupervised disentanglement as described in [21] which has been proved to be impossible. Instead we encode it in a subset of the latent variables in c. Thus each dimension of c is encoding only one factor while each factor can be encoded in multiple dimensions of c. This is a common issue discussed in the community as elaborated by [7]. In Fig. 3, our results for CelebA show that the model can discern the different factors while maintaining a low reconstruction error.

4.2 Conclusion

In this work, we present DOT-VAE, or Disentangling One at a Time VAE, a method of disentangling latent factors of variation in a dataset without any *a priori* knowledge of how many factors the dataset contains. We demonstrate that DOT-VAE is on par with or outperforms state of the art methods for disentanglement on standard metrics, and generates crisp, clear and interpretable latent traversal reconstructions for qualitative evaluation. Future directions could be to disentangle multiple datasets together using DOT-VAE while ensuring that the model reuses the learned factors and does not catastrophically forget them when new datasets are introduced.

t-ConvESN: Temporal Convolution-Readout for Random Recurrent Neural Networks

Matthew S. Evanusa[1,3], Vaishnavi Patil[1], Michelle Girvan[2], Joel Goodman[3], Cornelia Fermüller[1], and Yiannis Aloimonos[1]

[1] Dept. of Computer Science, University of Maryland, College Park, MD USA

[2] Dept. of Physics, University of Maryland, College Park, MD USA

[3] U.S. Naval Research Laboratory, DC USA mevanusa@umd.edu

Abstract. While deep neural networks have excelled at static data such as images, temporal data - the data that these networks will need to process in the real world - remains an open challenge. Handling temporal data with neural networks requires one of three options: backpropagation through time using recurrent neural networks (RNNs), treating the time series as static data for a convolutional neural network (CNNs) or attention-based transformer architectures. RNNs are an elegant autoregressive network type that naturally keep a memory of the past while performing computations. Although recurrent networks such as LSTMs have shown strong success across a multitude of fields and tasks such as natural language processing, they can be difficult to train. Transformers and 1-D CNNs, two feed-forward alternatives for temporal data, have gained popularity but in their base forms lack memory to keep track of past activations. Random recurrent networks, also known as Reservoir Computing, have shown that one need not necessarily backpropagate the error through the recurrent component. Here, we propose a novel hybrid approach that brings together the temporal memory capabilities of a random recurrent network with the powerful learning capacity of a deep temporal convolutional readout, which we call *t*-ConvESN. We experimentally verify that although the recurrent component remains random and unlearned, its combination with a deep readout achieves superior accuracy on a number of datasets from the UCR time series classification dataset collection compared to other state of the art deep learning architectures. Our experiments also show that our proposed method excels in datasets in the low-data regime.

Keywords: Recurrent Neural Network · Reservoir Computing · Deep Learning · Machine Learning · Neuromorphic Computing

1 Introduction

With the advent of multilayered/deep feed-forward networks, learning generalizable features for many tasks, such as classification, approximation the state for reinforcement learning and data generation are now possible [1,2,22]. However, these networks are designed for and tend to excel at *static* tasks, such as image classification or generation. Even in cases where the state evolves, such as in RL, the network still learns a static mapping of features to approximate the state. The real world, in contrast, contains exclusively temporal data consisting of the sights, sounds, and video that constitute our daily experiences. Temporal data grows exponentially complex with time, but humans handle this overabundance by keeping a history of the past in the network as *memory*.

To this end, learning length-invariant, time-varying parametric representations of temporal data that efficiently manage a memory of past events remains an open challenge. A recent line of investigation has been opened into granting neural networks access to an explicit memory location, as in a Turing machine [35]. The mammalian brain is known to store its memory as a combination of dynamical activity [37] as well as synaptic weight changes [38]. This dynamical memory results from maintaining a *state* of the network, something lacking in feed-forward architectures. To introduce this dynamical state in a neural network, we can create cycles within the network, leading to a Recurrent Neural Network, or RNN. However, owing to the cycles in the network, training RNNs via backpropagation commonly requires an expensive and tricky unrolling of the

network into a synthetic deep feed forward network before training, where the backpropagation is performed - also known as Backpropagation Through Time (BPTT) [8]. Due to the difficulty of this training, many non-recurrent methods have taken over as the state of the art for temporal data, such as Transformers [14], feed-forward convolutional autoregressive networks such as WaveNet [16] and Temporal Convolutional Networks [21]. However, it has been argued that these feed-forward architectures do not generalize well to time series whose lengths are not encountered during training [32], have restricted expressivity as compared to their recurrent counterparts [33], [34], as well as issues of efficiency as well as representation learning [18]. Within the neuromorphic community who are interested in biologically realistic learning rules, BPTT in its current form is considered non-plausible. It is thus of interest to the general neural networks community - to both those interested in biologically plausible mechanisms as well as performance - to continue to investigate the benefits of recurrent neural networks. To this end, we aim to develop a model that both lends to the biological plausibility of recurrence, as well as their theoretical performative benefits.

Recently, *randomness* in neural network parameters and initialization has become an important direction of inquiry into discerning the true effectiveness of deep neural networks. Recent work on biologically-realistic backpropagation shows that randomly chosen feedback matrices are effective at training hidden layers [3]. Proposal of the *lottery ticket* hypothesis [4] argues that the true work of stochastic gradient descent is finding "winning" randomly initialized subnetworks within larger networks - that the values of the winning sub-trees did not change much even after training. Reliance on over tuning of architectures in deep learning supports the argument that it is the random initialization of the architecture (and the subsequent fine-tuning of hyperparameters) that creates the best-performing deep networks. Here, we take a step further towards demonstrating the power of randomness by merging ideas from Reservoir Computing [5,6], which has for decades rested on random, unlearned recurrent weights, with temporal convolutional deep networks, to show that random recurrent weights are sufficient for temporal learning.

We demonstrate that the temporal convolutional readout is able to more robustly learn the complex dynamics for time series classification tasks compared to other neural models both feed-forward as well as recurrent. We further demonstrate that our method significantly outperforms gradient-based BPTTtrained recurrent neural networks, and make the case that the recurrent weights need not necessarily be learned for temporal tasks. We believe this is of interest to both deep learning, as well as the neuromorphic and computational neuroscience community, as methods that do not require backpropagating through time are both more biologically realistic and more implementable in specialized neuromorphic hardware.

2 Our Proposed Method: t-ConvESN

The two main components of our architecture are an untrained, random recurrent neural network component modeled off of echo state network dynamics, and a deep temporal convolutional readout that learns multi-timescale features with a multi-layered fully connected classifier. Figure 1 shows an overview of the structure. The reservoir or random recurrent network can be seen as acting as a temporal kernel machine [20]. Another way to view the structure is that the W_I matrix is randomly expanding the input temporal sequence in data space into a much higher dimensional temporal reservoir space. Our hypothesis - following the initial reservoir hypothesis in [5] - is that this higher dimensional reservoir space is more separable than the initial data space. We replace this readout layer with a deep temporal convolutional network that learns multi-timescale features. The reservoir component effectively gives the convolutional readout access to a fading memory of the input sequence.

2.1 Standard Echo State Network Structure

In typical reservoir computing, there are three main components: an input projection, a recurrent pool, and a readout mechanism. Our proposed network consists of a randomly initialized input weight matrix and a recurrent weight matrix which together determine the *state* vector of an Echo State Network at any time step. "Neurons" in the reservoir are represented as the activity vector x_t, which is a weighted linear combination of

the activity at the previous time step and the projected activity from the current time step. The activity is calculated as:

$$x_t = (1-\alpha)x_{t-1} + \alpha f(iW_I u_t + rW_R x_{t-1}) \qquad (1)$$

where x_t is the N-dimensional vector of the activity states of each reservoir neuron at time t, α is the leak rate, $f(\cdot)$ is a saturating non-linear activation

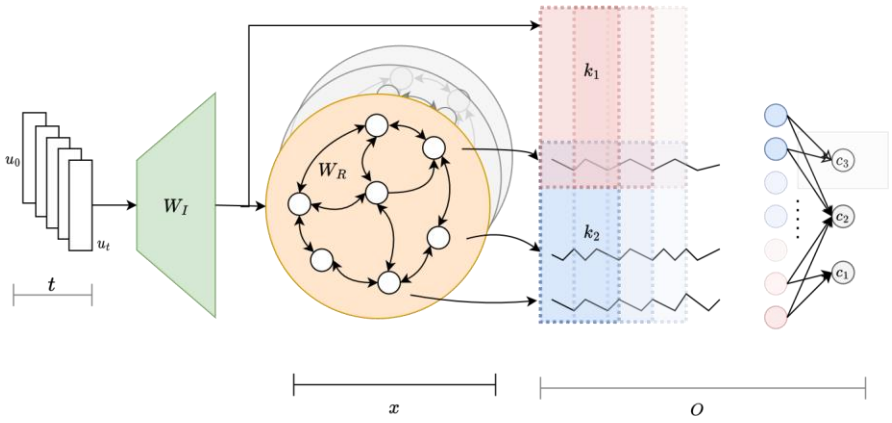


Fig.1. The *t-ConvESN* architecture for time series classification. Each element of the input sequence $u_0 \ldots u_T$ is fed in one time step at a time to the randomly weighted recurrent neural network *(center, orange)*, where T is the maximum length of each series. At each time step (See Eq. 1), the reservoir activity is updated as a combination of the input projected by matrix W_I and the past time step's activity at $t-1$ multiplied by the recurrent weights W_R. The reservoir is instantiated in parallel with other subreservoirs, shown in grey (middle). All sub-reservoirs' activity are concatenated before passing into the readout. The activity of each reservoir neuron, stored as a vector x_t at each time step t, is then fed into a temporal convolutional readout (depicted here as O, taking the place of the readout W_O in the standard ESN), along with the input along a "skip connection", where multiple filters are passed across the series in *reservoir space*. Shown here are two kernels k_1 in red and k_2 in blue, as they are shifted across the time domain. Finally, per traditional convolutional architectures, the resulting feature maps are flattened and passed into a fully-connected multi-layered perceptron for final softmax classification for class value c. Not shown here is the multi-reservoir structure: shown here is one reservoir for illustrative purposes.

function (here sigmoid), and W_I, W_R are the data-to-input and reservoir-to reservoir weights, respectively. The scalars i and r act as input and recurrent weight matrix scaling constants, which reduce or increase the gain on the activity. Formally, at each time step t, a vector-valued input $u_t \in \mathsf{R}^d$ is projected into a higher dimensional reservoir space by the input weight matrix $W_I: \mathsf{R}^d \rightarrow \mathsf{R}^D$ with ($d << D$). The $W_R: \mathsf{R}^D \rightarrow \mathsf{R}^D$, also randomly initialized, scales the state vector of the previous time step x_{t-1} such that the reservoir keeps a shallow memory of its past. This scaled previous activity is added to the projected input and this current activity is used to update the state vector x_t as per Eq. 1. The current state activity is a convex combination of the state vector of the previous time step and the current activity for smoother updates. As stated in [5], we normalize the maximum real eigenvalue of W_R, denoted the *spectral radius*, or σ, to be less than 1; this is a hyperparameter to be set. This ensures the *echo state property*, or that activity fades when there is no input.

Finally, for a vanilla ESN, the final classification or regression is performed by learning an output matrix W_O, which maps the state vectors to the corresponding labels y_t at each time step as in Eq. 1:

$$y_t = W_O x_t \quad (2)$$

For classification or regression tasks, the readout layer maps all the points of one signal to the same label. The weights of the readout layer W_O are trained with the appropriate loss function for the different downstream tasks. In this work W_O is by a sophisticated temporal convolutional readout, as described below and depicted in Figure 1. We provide a mathematical description of the readout in Sec. 2.2.

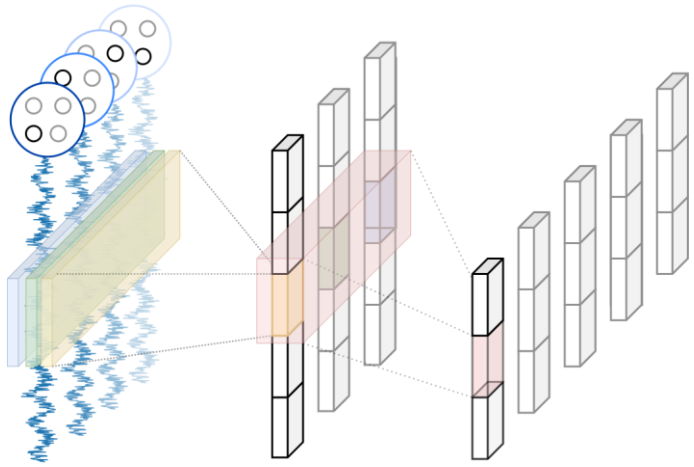


Fig.2. A zoomed in visualization of the convolutional filters being applied to different neurons in the reservoir. As in Fig. 2, the result of this convolution (the final layer, represented by vertical bars) is passed to a fully connected readout. The neuron shading depicts that the vertical signal output corresponds to a single neuron. All of the neurons collective outputs are passed over by multi-channeled 1-dimensional convolutional kernels, which output a vector feature map. The middle and left bars represent the second and third layers, respectively, of the CNN readout - for visualization purposes, as the number of layers can be changed - and the colors correspond to the value resulting from a particular convolution channel.

Parallel Sub-Reservoirs As in [12], we use a parallelized sub-reservoir structure. Instead of one large reservoir of dimension D, we pass in the input to multiple sub-reservoirs M with each reservoir of dimension H such that $D = M * H$, which are inter-connected amongst themselves but not with other sub-reservoirs. Mathematically, this is equivalent to one large reservoir with zeroes everywhere except for small blocks of weights around the diagonal. We find this is actually more computationally stable for the readout mechanism than a large reservoir, and produces more consistent and accurate results, especially for the convolutional readout. Unlike in [12] where concatenating the sub-reservoirs for a fully connected readout forced the readout to learn alternating combinations of subreservoirs, for the model in this work, a parallel structure has no effect on the convolution operation. This is because the convolutions are still applied across time, and the readout learns to map neurons from sub-reservoirs in non-linear combinations across time. The parallel reservoirs are analogous to the multiple heads in the transformer attention, or the multiple channels in a CNN architecture. By increasing the number of initializations in parallel, you increase the feature space on which you can search. Figure 2 visually depicts the interaction between the sub reservoirs and the convolutional readout.

2.2 Temporal Convolutional Readout

The novelty of our work lies in using a temporal convolutional readout, also known as a Time Delay Neural Network [9], or Temporal Convolutional Neural Network [21] (without the causal convolutions, for simplicity), as the readout for a random recurrent neural network. This readout consists of filters which convolve over the state vectors of the reservoir at different time steps to extract temporal features from the data. The different reservoir state vectors x are stacked to form the time series data matrix X, which is convolved with the temporal convolutional filters across time. A filter $k = \{k^1, k^2, \cdots, k^D\}$ such that k^i slides over the activity across time of neuron x^i according to the equation:

$$f(X)[t] = (X * k)[t] = \sum_{i=1}^{D} (x^i * k^i)[t]$$
$$(x^i * k^i)[t] = \sum_{m=-n}^{m=n} x^i_{t-m} k^i[m] \quad (3)$$

Figure 2 demonstrates the movement of the convolution filters across the output of the reservoir neurons. As for deep CNN architectures, we then stack such multiple 1d-convolution layers to extract hierarchical temporal features from the data for final classification, according to the equation:

$$f^i(X) = \sigma \odot (f^i(X)[0],\ f^i(X)[1],\ \cdots,\ f^i(X)[T])$$
$$y = W_O\ (f^n \circ \cdots \circ f^2 \circ f^i(X)) \quad (4)$$

where T is the number of time steps in an example of the time series, σ is the activation which is applied point-wise ($\odot$) to the output of a convolutional layer and n if the total number of 1-d convolution layers. Each convolutional layer f^i consists of multiple filters which convolve over the input to the layer in parallel, as is commonly done.

Through this, we are able to learn nonlinear combinations of the responses of individual neurons as a function of time in a much more compact way compared to all-to-all connections as in a multi-layered Perceptron. One way to view this interaction is that the readout is learning nonlinear filters of the response of the recurrent component; in this way the reservoir can be seen as acting as a random temporal kernel [20]. Due to the way convolution operates in one dimension, each channel of filters actually learns a combination of the responses of each neuron, which gives the readout more computational flexibility than being forced to learn a filter for each neuron. The convolutional kernels are stacked in layers, allowing the readout to extract hierarchical features across time. In addition, the kernels share weights, facilitating the network to learn shift-invariant features. Causal and dilated convolutions as in [16] are not used for simplicity, but are left for future work. The feature maps resulting from the convolutions are fed into a multi-layered feed forward network for final classification. While we will point out that we do train this readout with backpropagation, which is not biologically inspired as is the reservoir; we view this as a temporary placeholder for any future biologically-adjacent nonlinear network training rule, such as Feedback Alignment [3], as the backpropagation is performed in a feed-forward manner, rather than through time.

3 Related Work

Prior work has looked into using 1-D convolutional filters for time series learning [15]. Hybrid deep learning approaches to reservoir computing have been developed in the past several years to address the inability of "vanilla" reservoir computing to scale up to complex tasks. In [12], the authors showed using a combination of a traditional reservoir with multiple sub-reservoirs, as well as a deep artificial neural network with modern regularization techniques, could outperform state of the art methods on real world data tasks. Our work can be seen as a further evolution of this work that replaces the artificial neural network readout with a more sophisticated temporal convolutional readout. In [27], the authors combined a reservoir with a complex attention and memory mechanism, that allowed the reservoir access to longer periods of memory. In [23], the authors used untrained kernels, as in a CNN [11][10], to train a reservoir to recognize image data, in order to avoid backpropagating the signal through the reservoir. Previous work in [24] also combined the reservoir with other kinds of deep learning readouts, for time series classification. A different line of research aimed to

stack reservoirs in layers as an MLP stacks perceptrons in layers, leading to "deep reservoir" computing [25], and novel unsupervised ways to train them [26]. In [28], the authors use a multi-timescale convolutional readout that learns independent kernels of multi-timescale reservoirs. Our work is different in that rather than enforce multiple timescales in the collected states, we let a deep convolutional readout *learn* the multiple timescale features. Our kernels also learn to combine activity from multiple reservoirs, which aids in stability. To the best of our knowledge, our work is the first to incorporate a deep temporal convolutional readout in the time domain, for the neuron pool, that learns to combine multiple sub-reservoir activity across time as multi-channel kernels, to learn multi-timescale features.

Table 1. Comparison top-1 accuracy results of our proposed *t-ConvESN* method against four other deep neural network architectures: a standard MLP, LSTM [7], Transformer [14] encoder classifier, and a 1-d CNN (TCN) on 19 UCR datasets. All networks were run 10 times, with results posted as the mean of the runs with standard deviation to test for consistency. The top pane are univariate datasets, and the bottom pane are multivariate. All networks were trained on 1000 epochs. Our *t-ConvESN* shows surprising consistency even though the W_I and W_R weight matrices are chosen randomly, *and the 10 runs initialize a different random weight matrix each time*. The selected datasets are chosen for being more difficult, i.e., datasets that were considerably less than accuracies of 1 for most models. For consistency, the ReLU [19] activation function was used for all layers of all networks. By outperforming the TCN, we show that it is not just the convolutional readout, but the combination with the recurrent component, that performs well.

Dataset	MLP	LSTM	Transformer	TCN	**t-ConvESN**
Beef	.832 ± .02	.722 ± .04	.813 ± .03	**.967 ± .00**	**.967 ± .00**
Car	.805 ± .04	.488 ± .04	.816 ± .04	**.853 ± .01**	**.851 ± .01**
CinCECGTorso	.596 ± .03	.601 ± .03	.826 ± .03	.932 ± .01	**.968 ± .01**
ChlorineConc.	.872 ± .02	.582 ± .02	.801 ± .02	**.901 ± .01**	**.901 ± .01**
Computers	.521 ± .02	.582 ± .04	.533 ± .02	.613 ± .02	**.629 ± .02**
ECG200	.862 ± .02	.710 ± .02	.882 ± .03	.932 ± .01	**.942 ± .01**
ElectricDevices	.514 ± .01	.422 ± .01	.460 ± .01	.623 ± .02	**.652 ± .01**
ItalyPowerDemand	**.964 ± .01**	.943 ± .02	.958 ± .02	.950 ± .01	**.963 ± .01**
Lightning2	.675 ± .03	.631 ± .02	.708 ± .04	.736 ± .01	**.793 ± .02**
Lightning7	.582 ± .03	.557 ± .04	.625 ± .03	.682 ± .01	**.722 ± .03**
OliveOil	.933 ± .02	.401 ± .01	.872 ± .04	.936 ± .02	**.963 ± .01**
RefrigerationDev.	.353 ± .01	.477 ± .03	.361 ± .03	.490 ± .01	**.528 ± .01**
ERing	.368 ± .01	.623 ± .02	.371 ± .01	.586 ± .01	**.750 ± .01**
RacketSports	.400 ± .02	.532 ± .02	.521 ± .03	.543 ± .02	**.833 ± .01**
NATOPS	.374 ± .03	.448 ± .04	.500 ± .02	.565 ± .03	**.638 ± .02**
PEMS-SF	.514 ± .03	.551 ± .02	.568 ± .03	.684 ± .01	**.716 ± .02**
Libras	.763 ± .02	.458 ± .02	.762 ± .03	**.884 ± .01**	**.888 ± .01**
Epilepsy	.632 ± .02	.602 ± 0.03	.701 ± .02	.881 ± .01	**.958 ± .01**

4 Experiments

We compare our proposed model now against numerous state of the art methods to test its efficacy across multiple time series classification datasets in the UCR Time Series Classification dataset archive [13].

4.1 Comparison Methods and Setup

We compare against exemplar networks from the main sequential or temporal processing neural networks: Transformers, LSTMs, and Temporal Convolutional Neural Networks along with the fully connected multi-layer perceptron. We chose specific datasets that were more challenging, which showcase our model's performance in the low-data regime, and avoided "synthetic" time series datasets that were created out of rasterizing image data. For all runs, the Rectified Linear Unit activation function [19] was used for all networks. In Table 1, we list the chosen datasets which represent more natural signal data such as electrical power, sensor readings, and ECG, and show the top accuracy results from the different methods. For different datasets, different numbers of sub-reservoirs were used: on smaller datasets, we found that four small reservoirs performed better. We

perform training for the readout deep network using the AdamW [36] optimizer in pyTorch, using dropout as a regularizer in conjunction with batched training and batch normalization. Due to the large number of parameters in echo state networks, it is common to use some kind of search optimization to find optimal configurations [30]. These include α, σ, i, r, the number of sub-reservoirs, and the size of the sub-reservoir, in addition to other standard neural network hyperparameters like the learning rate. Here, we use the Optuna python library [31] to optimize said parameters. For a fair comparison, we also run the Optuna optimizer to find the best hyperparameters for the competing state-of-the-art models, such as the learning rate, dropout, number of layers, and number of heads for the Transformer. Experiments were run on the RTX A4000, A5000, and A6000 GPUs using pyTorch for the deep neural networks.

4.2 Results

Table 1 shows the top-1 accuracy across several datasets from the UCR time series classification dataset collection. For reproducibility, especially because initialization is critical for random recurrent architectures, we ran each network 10 times with different random seeds and report the mean accuracy and standard deviation. Our proposed *t-ConvESN* outperforms most other models consistently, with low standard deviation, even though the reservoir is chosen and kept random throughout training even though the reservoir is initialized randomly during each run. We hypothesize that this is empirical evidence that the random projection offers a rich set of dynamical features that are readily useable by a sophisticated readout mechanism. As in [12], we argue that standard echo state networks without a non-linear readout are not able to linearly separate some complex data. In particular, our network far outperforms the gold standard LSTM gradient-based RNN on classification tasks. We believe this is because while the datasets have lower spatial dimension, they contain long temporal sequences, and the LSTM is incapable of backpropagating far through time. On some datasets we see that fully connected models like the multilayer perceptron and Transformers outperform the sequential or temporal processing models. We believe that this is because the fully connected models have larger number of parameters and the corresponding labels for the datasets do not explicitly depend on the temporal properties. Furthermore, on some datasets we see that only the Temporal convolution network performs comparably with our t-ConvESN network. For these datasets, we speculate that the dataset does not really benefit from being projected into a higher dimensional space by the reservoir to be mapped to the corresponding labels. For some datasets which are easily separable, such as *Car* and *Beef*, our model matches the performance of the base TCN. We posit that this is because recurrence adds no additional benefit, and the class information is contained within a local window. However, for most of the other datasets, this is not the case: long-term information aids in generalizing to the test set. In addition, most of these datasets are in the extreme low-data environment, with less than 500 samples for training. We hypothesize our model particularly performs well in the low-data regime, even with a deep learning readout, because projecting the data into the random reservoir space allows the readout to learn separation planes with fewer data. However, more investigation is needed in this front. We also believe this low-data regime is the reason for the Transformer network (and the other deep learning set) relatively poor performance. While it remains to be seen if this generalizes in future work to more and more complex datasets, we believe this is a fruitful sign for neuromorphic and bio-inspired networks, that we do not have to choose between either biological inspiration or accuracy; that neuromorphic architectures can compete at the level with other state of the art methods.

5 Conclusion

In this work, we present a novel bio-inspired random recurrent neural network for temporal data, *t-ConvESN*, which projects the data into a space in such a way that gives a convolutional neural network access to a fading memory. We empirically demonstrate that the network exhibits robust and consistent performance even though the recurrent weights remain untrained, and can outperform many state of the art methods on time series. Future work can involve making the network fully biologically plausible by replacing backpropagation training of the readout with any of the newly-discovered backpropagation alternative gradient methods such

as Feedback Alignment variants [3,17]. While this work employs a relatively simple CNN readout, future work can incorporate more complex convolutions such as in WaveNet [16] or attentional mechanisms.

MAELSTROM NETWORKS

Matthew S Evanusa
University of Maryland, College Park
US Naval Research Laboratory, Washington DC matthew.evanusa@gmail.com

Cornelia Fermüller
University of Maryland, College Park

Yiannis Aloimonos
University of Maryland, College Park

ABSTRACT

Artificial Neural Networks has struggled to devise a way to incorporate working memory into neural networks. While the "long term" memory can be seen as the learned weights, the working memory consists likely more of dynamical activity, that is missing from feed-forward models. This leads to a weakness in current neural network models: they cannot actually process temporal data in time, without access to some kind of working memory. Current state of the art models such as transformers tend to "solve" this by ignoring working memory entirely and simply process the sequence as an entire piece of data; however this means the network cannot process the sequence in an online fashion, and leads to an immense explosion in memory requirements. In the decades prior, a separate track of research has followed recurrent neural networks that maintain a working memory via a dynamic state, although training these weights has proven difficult. Here, inspired by a combination of controls, reservoir computing, deep learning, and recurrent neural networks, we offer an alternative paradigm that combines the strength of recurrent networks, with the pattern matching capability of feed-forward neural networks, which we call the *Maelstrom Networks* paradigm. This paradigm leaves the recurrent component - the *Maelstrom* - unlearned, and offloads the learning to a powerful feed-forward network. This allows the network to leverage the strength of feed-forward training without unrolling the network, and allows for the memory to be implemented in new neuromorphic hardware. It endows a neural network with a sequential memory that takes advantage of the inductive bias that data is organized causally in the temporal domain, and imbues the network with a state that represents the agent's "self", moving through the environment. This could also lead the way to continual learning, with the network modularized and "'protected" from overwrites that come with new data. In addition to aiding in solving these performance problems that plague current non-temporal deep networks, this also could finally lead towards endowing artificial networks with a sense of "self".

1 INTRODUCTION

The ultimate goal of artificial intelligence is to recreate the intelligence that biological agents have displayed to remarkable degree, extract the core components of intelligence, and reproduce this in an artificial agent, potentially amplifying this. Since the early days of *connectionist* networks, where we attempt to solve this intelligence problem through networks (or graphs) of simple artificial neural units, the goal has somewhat drifted away from a general artificial agent towards more specialized tasks, namely, pattern recognition. Feed-forward neural networks, built off of the *Perceptron* framework (Rosenblatt, 1961), have excelled at pattern recognition, and have reoriented the entire field towards this mapping function. This has culimated in the current generation of Large Language Models, which are at the core, meta-networks of Perceptrons trained using backpropagation (Vaswani et al., 2017). What have we sacrificed in this focus on pattern recognition? This is the

question that we motivate this work with. We have, we argue, sacrificed the notion of the agent as a state, a sense of self, that persists across time, and the temporal correlations that accompany this.

Data is Temporally Organized

At the core of this issue is the current foundational viewpoint for deep learning that data in the universe is, in some sense, I.I.D, and has no underlying structure; it is up to the *network* to learn this structure from random noise. Creatures, however, evolved in the real world, with real physical limitations on the way that the system evolved to deal with, and survive in, the real world. One of the set conditions that nature gives us, in addition to the 3-dimensional structure of the world, is the temporal nature of cause and effect; data later in time are temporally correlated with the earlier data. Figure 1 gives a graphical depiction of this dichotomy. The data that is aligned temporo-causally along the same strands we refer to as temporal sequences. And the memory mechanisms in agents that is responsible for remembering points along the same strand we refer to as *Sequence Memory*.

The viewpoint of feed-forward connectionist networks, which is the current paradigm for deep learning, is that data must be assumed to be independently distributed - the I.I.D prior. We assume that the data is "given" to us in a completely random, jumbled state, with no inductive prior on the structure of the data, and it is up to the network to learn the structure of the data. This is what we expect from neural networks - the only inductive prior that we assume is that the data is structured *hierarchically*, which is what leads to our "deep" learning structures of multi-layered networks. We do not, however, flip this 90 degrees, and think about the structure of the data in time. The data in the real world, however, *is* structured: it follows clear causal relations between the data in time, which we can visualize as "strands" in time. Each of the data that is causally linked sits on the same "strand", which may branch off from one another. Figure 1 shows these two varying viewpoints of data in the world. The story of deep learning has shown that the structure of the network topology is paramount: the learning rule (backpropagation) as well as the activation of the neurons has largely remained unchanged in the decades since deep learning came about, only the topology of the network, the structuring of the linear layers into convolution or attention layers, has changed. This insight demonstrates that to effectively tackle networks in the real world, inductive biases must be given for the temporal domain and not just the spatial. This inductive bias takes the form of *Sequential Memory*: the ability of the agent to remember data that is causally linked accoring to the same temporal sequence strand. This bias is implemented using modules of resonating, recurrently connected graphs of neurons which we call the *Maelstrom*.

Brains Have State via Resonating Cell Assemblies

How do creatures persist across time, and remember the memory of temporal sequences as they occur? Of course, statistical pattern recognition is a key component, but another component must be the system's ability to retain the information such that a pattern "readout" can occur. This idea that the brain's mechanism for recognition is a combination of cell assemblies, which maintain activity, and a readout mechanism, which performs a mapping of activity to motor action and tasks, is well documented and is a current theory for brain organization (Buzsaki, 2010). According to´ the theories of Neural Assemblies by Donald Hebb (Hebb, 2005), the activity persists in groups of neurons, called assemblies, that resonate their activity across time according to some stimulus. It is this resonance, we argue, is the "stable state" that allows living creatures to persist across time. A separate readout mechanism, then, performs mapping of these states, akin to a function learning a mapping of states of a dynamical system to outputs.

Cell Assemblies are a Temporal Inductive Bias

The key insight for the brain is that these assemblies *model the temporal sequences by mirroring the cause-effect relationship of the input data, but in the recurrent patterns of the network*. Thus, the same temporal patterns that occur in the real world, are "echoed" in the temporal patterns of the recurrently-linked cell assemblies. This topographical structure of a recurrently connected graph structure that is modular and unhooked from the gradients of the sensory and motor areas are the temporal inductive bias in the brain, and in our proposed *Maelstrom* network, that allows it to maintain a state and capture sequence memory. Whereas the spatial inductive bias is that the hierarchical nature of the features in the data are represented as feed-forward layers of the network, the temporal inductive bias is that the looped or cyclical structure of the recurrent components captures "resonances" which

mirror the temporal-causal relationship of the data. As the topology of the network in the brain is fixed, the specific resonances that occur will occur with the same temporo-causal patterns that occurred in the real world, *except* that it collects cause-effect from across time.

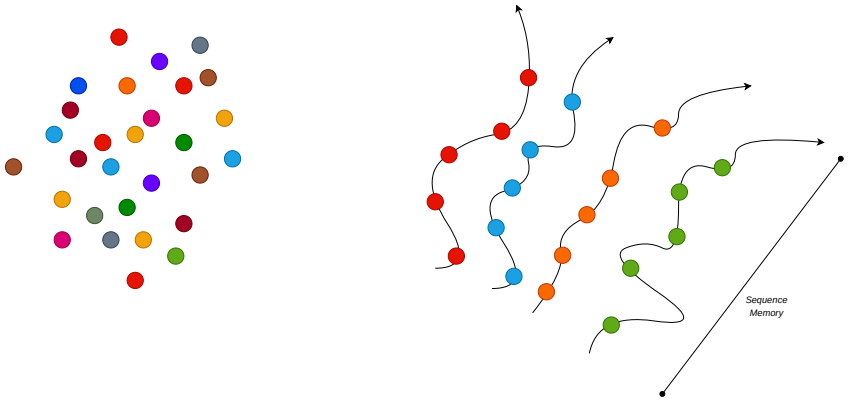


Figure 1: As opposed to the current zeitgeist of machine learning, data in the real world follows inductive biases in how the data is structured not only in the spatial domain - for which we have taken heavy account - but also in the temporal one. *Left:* The current view of machine learning which sees data as independently and identically distributed (I.I.D) in time; it is the job of the network to learn the features in time that correspond to each data point. This assumes the inductive bias of the spatial hierarchy of the data, but not the temporo-causal relationships of the data points along the same time thread. *Right:* The new view of networks as also accounting for the temporal inductive bias of the data along the temporal dimension. The points are related to one another on the same temporal thread via the referential frame of *action*: it is those actions which took a point from one location to another, as a result of the causal effect of the action. The ability of the network to recognize data along the same thread is what we refer to here as *Sequence Memory*.

Interestingly, the story of deep learning has been one, we argue, of inductive biases and nothing more, in the form of network topologies, in the static space. A deep multi-layered perceptron actually contains more parameters than the same layered transformer, however, the transformer performs remarkably better; this is generally understood to be because of the attention mechanism, which is just a special form of inductive bias on the structure of the linear layers; this structure biases the layers to learn representations that focus on attending to specific elements in a sequence.

2 Limitations of Current Approaches

Although deep neural networks have shown remarkable performance, their success masks some key missing elements if we are to achieve a thinking machine. Although some posit that Large Language Models and this is an open debate in the literature, the current pinnacle of feed-forward networks, have “reasoning” capabilities, we would argue here that they rather are simply powerful pattern mappers, and that many tasks we ascribed to reasoning in the past are actually just complex mapping tasks. This is a longer debate outside of the scope of this paper, but for the purposes of this work, we argue they cannot be true reasoners in the real world as they are not embodied. And to be embodied, they are missing one critical components: namely a memory of the past activations via a *state*.

2.1 Feed-Forward Networks Lack Sequence Memory

Memory, as a concept, is tricky to pin down, although there have been some interesting definitions of memory that bridge the biological with the computational (Zlotnik & Vansintjan, 2019). Here, we take *memory* in the context of connectionist neural networks to mean: *a mechanism by which the neural network maintains a state of itself, and contains information about the past from a temporal sequence*. This is what we refer to as described earlier, as *Sequence Memory*. The idea of sequence memory is intricately tied to the notion of time: to process time, an agent must record, in some state variable, some information of timesteps prior. As articulated in earlier sections, the idea

of a recurrent network, unhooked from the gradient, gives a temporal inductive bias to the processing of the data. This accompanies with it the important notion that *all* data we receive, as active agents (Friston et al., 2016; Aloimonos et al., 1988), is in the temporal domain. In our own brains, we do not have access to a "RAM-like" memory that computer systems have (as our entire system is the connectionist neural network), the neural network itself must keep a memory of its own activations, to serve as the state variable for temporal processing. This "state" of the neural network is then updated with new information, in a recursive fashion. This kind of architecture is mirrored in Recurrent Neural Networks (RNNs), as described below, and serves as an important foundation for our proposed ideas here.

Of course, incorporating a "state" that updates with the model introduces a host of challenges. It is not hard to see why the current state of the art neural networks, built on Transformers (Vaswani et al., 2017) which then became Large Language Models (Radford et al., 2019), for the most part eschewed the idea of recurrence (although some hybrid work has arisen in recent years attempting to fuse them (Hutchins et al., 2022)).

2.2 Memory Serves in the Service of Embodiment

To be an artificial embodied intelligence (Chrisley, 2003; Duan et al., 2022) entails placing the learning system in a chassis that can take actions in an environment. This is an active process (Aloimonos et al., 1988; Friston et al., 2016), with the agent contained in a control loop within the environment. And to realize these actions in an environment which is moving in time, the agent must have access to a running state of the previous activity. To not have this state would amount to the agent running with "blindfolds" on around their perception completely, only seeing the current timestep and making a decision, then moving onto the next sensory perception. This realization amounts to reducing agents to mere pattern recognizers - thus the idea of having a memory in a strong sense elevates agents above merely pattern matching entities. It has also been argued that the entire purpose of memory, rather than serving as some RAM-like appendage that stores abstract values for later use, completely serves embodiment (Glenberg, 1997).

3 Prior Work on Connectionist Memory

3.1 Memory in Connectionist Networks

As mentioned earlier, the way that researchers think about neural networks *in silico*, versus how neural networks work *in vivo*, is fundamentally different not only from the algorithmic perspective, but also with respect to how memory is stored. In computers, we simply store our neural network code in RAM or disk storage, which is a collection of buckets that can store any arbitrary numerical values (without respect to any task). All of the functions that require remembering (such as loading code, the weights, the dataset) are stored in this disk memory. In contrast, in the brain there is no arbitrary storage separate from the neural network - the storage *is* the neural network. This leads to some more complex and dynamical architectures to store values in "working memory", versus the "long term memory", which corresponds to the disk memory. Working memory *in silico* corresponds more closely to RAM in that a consistent voltage must be applied, although this memory is stable and unchanging, whereas dynamical attractors in the brain are constantly in flux. Regardless, in connectionist networks, this memory must be stored as a *state* of the network, or an abstract vector that updates in time as the network progresses through inputs. As the brain does not actually have this vector which itself would be stored in disk, the *state* of the brain is really the current snapshot of the voltages and accumulation of neurotransmitters for each neuron terminal and body. To read the state, the brain has only access to the action potentials or "spikes" that are emitted to send information between neurons (barring any unforeseen tricks by the glia). Thus, to truly implement this state in a neuromorphic way, the value of the state is simply the output activity being sent back to itself, creating a "self-loop". While in code it is possible to maintain in the disk the states of the "transmitters" of the neurons, this idea of recurrent connections is still critical as loops provide recurrent computation and allow for the memory to reverberate throughout the network. Thus, the network persists its state (or memory) of the temporal signal by reverberating or bouncing a reflection of the input inside itself to maintain a persistent activity.

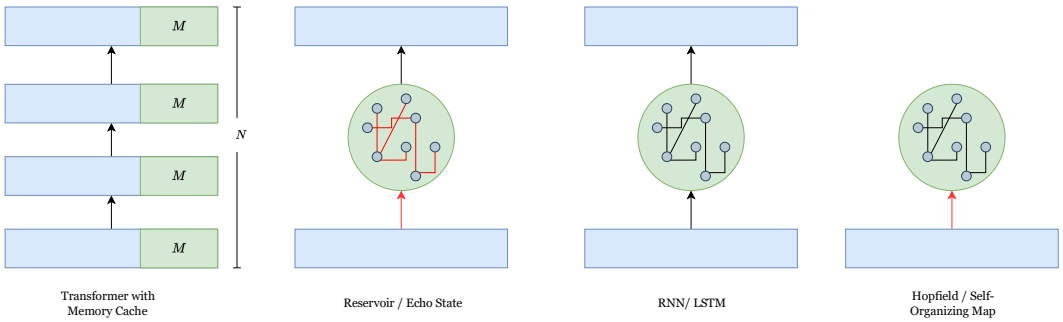


Figure 2: Comparing the various prior approaches to dealing with temporal sequential memory (or lack thereof). Blue rectangles indicate a feed-forward network layer (no memory), while green represent a recurrently connected one or memory that is accessible through timesteps. Black arrows represent a learnable weight, whereas red indicates unlearned, where the gradients do not pass backwards. *From left to right*: Transformers or CNNs (LeCun et al., 1998; Devlin et al., 2018; Vaswani et al., 2017) are in their vanilla state purely feed-forward networks, and are not represented here because they do not contain any form of sequence memory. Only newer variants of Transformers such as Transformer-XL, which contain a "cache" that is accessible to later timesteps, have what we would consider sequence memory. However, these are not "true" sequence memory as they do not solve the continual learning problem, due to the fact that they are still non-modular (Hadsell et al., 2020) (i.e., the memory component is still "hooked" to the motor networks) and thus any new gradients would overwrite previous timesteps. Reservoir networks (echo state networks or liquid state machines) have a recurrent component with unlearned recurrent weights and input, but a learned readout that can be a feed-forward network (Evanusa et al., 2023). RNNs and LSTMs have a recurrent component where every connection is learnable. Both LSTMs and Reservoirs can have multiple recurrent "layers", connected in a hierarchical structure (Gallicchio et al., 2017). Lastly, Hopfield (Hopfield, 2007) and Self Organizing Maps (Kohonen, 1990) are recurrent components without a learned readout, where the recurrent weights are trained using an unsupervised self-organizing rule.

3.2 Hopfield Networks

A result of Donald Hebb's seminal work on neural population and organization (Hebb, 2005), Hopfield derived a network that views the network as an energy minimization problem, where the activity that bounces around and settles into *attractor states*, and the learned patterns correspond to energy minima in this landscape. It has been shown that using the Hebbian learning rule mathematically equates to finding, for a given data sample, the energy minimum attractor state associated with that network architecture. This has the benefit of not needing labeled samples, is completely self-organizing, and is biologically relevant. The issues hampering it have been a lack of ability to extract latent codes - features - from large amount of samples, and the somewhat strict limits on memory that result from this. However, new work has shown that via gating mechanisms (Hochreiter & Schmidhuber, 1997; Davis et al., 2022) the memory can be increased. The gradient-based LSTMs also suffer the same problem as in the earlier section about the weights not being sensitive to the particular timestep. While in their basic form Hopfield networks have failed to match deep learning, newer architectures that incorporate continuous values and attention show correspondence between Hopfield learning and attention layers in Transformers, and are a promising direction for future RNN research (Ramsauer et al., 2020).

3.3 Gradient-Based RNNs

A simple approach to training a recurrently connected network is to treat it like a feed-forward network, and treat the time dimension as if the network were actually multiple layers deep. For a single-layered RNN, the network is run in time for a set number of t timesteps, and then the network is "unrolled" to have t layers corresponding to the t timesteps. The gradient is learned and then the updates are all applied to the same weight vector. This training mechanism is known as "Backpropagation Through Time" (BPTT), as the network through time is unrolled and treated as a deep neural network. It is generally accepted that BPTT is highly biologically unrealistic as the brain cannot unroll itself, but several approximations have been developed in recent years that try to argue for a mechanism in the brain that produces similar effects (Cheng & Brown, 2023; Manneschi & Vasilaki, 2020).

The issue, still, is that the memory and the feature vector are not truly topologically separated: the "memory" vector (represented ostensibly by the cell state) is still trained and *driven* by the same signal as the error for the feature vector. The only difference is that the cell state 'prefers' to stay unchanged for longer periods, allowing for long-term dependency learning; this is not a true philosophical "separation" of the memory and the feature vector. Thus, it is not surprising that the Gated Recurrent Unit or GRU (Cho et al., 2014), which fuses the cell state and the hidden state a process that if the memory were truly separated and the system depended on it, would destroy the function of the system - actually tends to perform the same or sometimes even better, with less overhead, and is the preferred RNN of choice in modern times when RNNs are used. Of course, RNN usage has been in recent times completely eclipsed by feed-forward networks - in particular Transformer models Vaswani et al. (2017).

3.4 Reservoir Computing

As a response to the requirement of stable attractor states from Hopfield Networks, Reservoir Computing (Jaeger, 2001; Maass, 2011) was developed to convert the problem from an attractor-based one to a mapping based one. Critically, it converts the problem of having the network settle, to having the network "observe" the dynamical state of the recurrent state (now called the *reservoir states* - but effectively the same as a hidden state of an RNN). The insight is that we need not propagate the error through the recurrent component. The component will produce some activity as a result of its initialization, and if that initialization is random, or is a sufficient basis set to cover the possible features, a sufficiently powerful readout is capable of mapping that activity to any predictive value. Here, it is also important to note that we need not learn, in the recurrent component, *what* the memory activity corresponds to; this is the task of the readout mechanism. The recurrent component simply is in charge of bouncing and persisting activity, just long enough for the readout to make a determination. This new paradigm fundamentally shifts around the requirements of a memory mechanism, and makes an important founding that we use for this work as well: that the memory and predictive processing components can potentially be split into two separate processes. While the memory necessarily must be tuned to help in the service of prediction as in (Glenberg, 1997), it is really the combination of this readout mechanism and the memory that constitutes the effects of memory. We take this idea of separating the memory and computation components, along with the idea that we need not propagate the gradients through the recurrent component, as the job of the "observer" is simply to perform a mapping.

One of the key insights of reservoir computing that will be extracted is that the memory unit is topologically separated, *unhooked*, from the gradient learning process of the *readout*, which is the network assigned to do the functional mapping from states to actions (labels, or values). We take this core idea from reservoir computing, and expand it outward in the *Maelstrom* paradigm into a much larger proposed theoretical structure, without the limitations placed by reservoir computing.

4 Maelstrom Networks

To solve the issue of sequence memory and to endow a neural network with its capabilities, we present here *Maelstrom Networks*. From control theory, we take the idea of a state space model that incorporates a memory state, an input function, and an output function. From reservoir computing we incorporate the notion that the temporal memory must be a process topologically separated from the readout mechanism, in terms of learning via the gradients. And we use feed-forward networks for their preferred role: to map inputs to outputs, and not to learn how to store things in memory. In Fig. 4 we show the schematic overview of the proposed approach. From an implementation standpoint, the input and output functions are easily parameterized by deep neural networks - blocks were shown here for visual simplicity. The key element which gives the network its name is the *Maelstrom*, a recurrently connected component which can be implemented as a reservoir - but not exclusive to - , that takes input from the input function (an input neural network), bounces this activity around, and passes this to an output function. we say "passes to", but in reality what is happening is, it is more akin to the output function reading out the maelstrom; you one can visualize

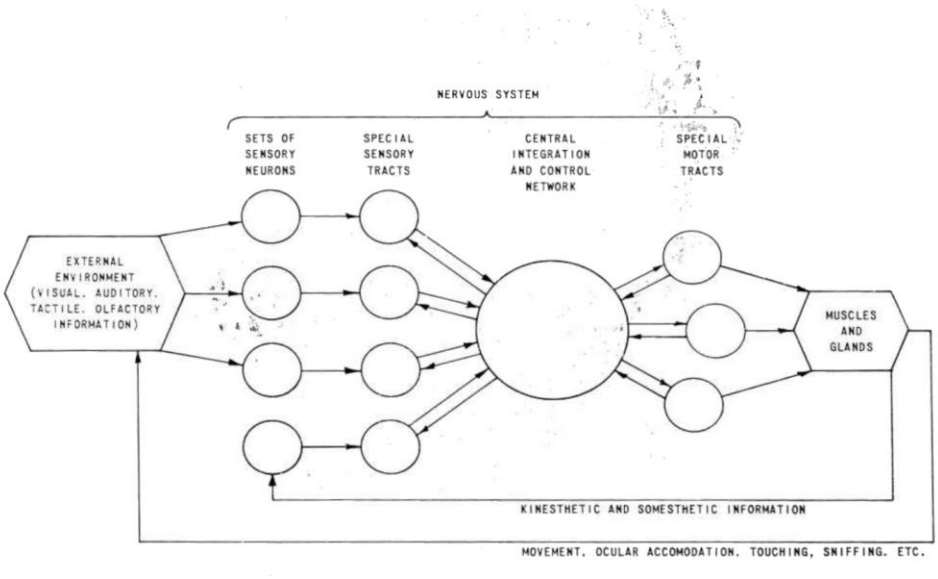


Figure 1 BASIC TOPOLOGICAL STRUCTURE OF THE NERVOUS SYSTEM AND ITS SOURCES OF INFORMATION

Figure 3: The topological organization of the nervous system as proposed by Frank Rosenblatt in 1961, taken from Rosenblatt (1961). This bears a strong resemblance to the Maelstrom Paradigm: The sensory tracts and memory represent the input function, the motor tracts represent the output function, and the integration network is the Maelstrom. The only difference, and the only thing lacking from Rosenblatt's account in our opinion, is the notion of sequence memory. The Maelstrom can be seen as an implementation of Rosenblatt's ideas, in combination with deep learning as well as notions of state and sequence memory.

this as the output function being a pair of eyes that observe the chaotic activity of the maelstrom. Because the network is untrained and recurrently connected, it is a potentially chaotic system - a storm that gives the maelstrom its name. It is the job of the input to "control" the activity of this storm and keep it within acceptable bounds.

Critically as well, the maelstrom possesses "top-down" feedback onto the input function, which allows the feed-forward input function a control loop over its own activity. This "closing the loop" on the maelstrom is what gives a meta-loop between the maelstrom and the network structure as a whole. These feedback connections are ubiquitous in the primate cortex (Zagha, 2020) and it is this loop that feeds back into the input which, in addition to the loop within the maelstrom, creates the "state" that we perceive as our continuous self. The inclusion of this loop is intended to, for the first time, imbue this sense of "self" with deep learning.

The stimulus from the outside world is passed first through the input function - parameterized by a deep *feed forward* network. This does exactly what neural networks are good at: function mapping; the job of the input network is to map the activity is passed from the input controller to an interface (another neural network) that may or may not connect around the maelstrom as a skip connection (dotted line - Fig. 4). The job of the key element is that the gradient does not flow through the maelstrom - it simply agglomerates activity from the neural network surrounding it. And critically, the output function *sees the maelstrom as if it were the input*, as the maelstrom does not have access to the gradients of the output. This is akin to the idea of "neural ensemble readout mechanism" as summarized in (Buzsaki, 2010). The maelstrom accrues a memory of activations, which can be´ guided by the input neural network as well, since those weights are also learned. The term *Maelstrom* represents the fact that this memory is a chaotic storm of activity that we can see from the outside, but cannot access the internals of. It's important to stress that this does not force a specific setup for an architecture, but rather a general class of architectures that need just follow this general setup of the gradient flows. The critical feature for the maelstrom network is a disjointed memory component where gradients cannot flow through, but that give the output function

access as an input. This bears a strong resemblance to Rosenblatt's early work on the topological structure of the nervous system, as seen in Fig 3.

The Maelstrom Network's only requirements are the structure of the input, and output, and recurrently-connected maelstrom that feeds back into the input of the input network. How one implements these is up to the user, however, it is clear that a deep feed-forward neural network is primely situated to serve as the input and output functions, and a recurrent network in some capacity is primed to serve as the maelstrom. How complicated these are, and how the gradient is passed around the maelstrom, is still an open area of research. In a more philosophical sense, we can think of the cell assemblies as solving the issue of sequential memory by "mirroring" the activity out of the outside world, but in the internal model of the agent.

It is our opinion that the best implementations for the maelstrom, however, build on the work that demonstrate that the brain does behave like a near-chaotic system; the ability of the output function to map chaotic network states to actions is a key capability that deep learning offers us, in parameterizing with them.

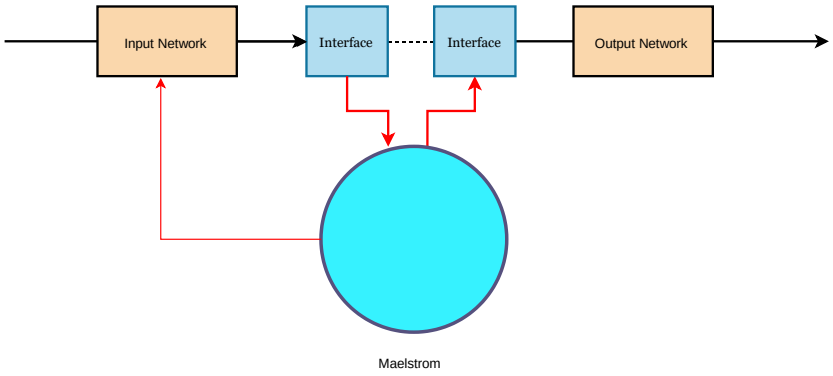


Figure 4: The Maelstrom Network paradigm. Arrows indicate the input is passed through an *input network*, a feed-forward neural network which maps input patterns to control actions on the maelstrom. This is passed then to an interface, which serves as the hub for talking to and from the maelstrom. The interface passes to the *maelstrom*, a recurrently connected state space that collects and conglomerates actions from the controller. The maelstrom bounces and maintains a state of the previous input. As the maelstrom is recurrent and unlearned, or unhooked, from the gradient of the output, it exhibits chaotic behavior - it is the job of the input network to control this activity. The interface then reads the maelstrom and passes this to an output function, which then produces an output. For neural network approaches, the input function, output function, and interface are all multilayered neural networks. Critically, when learning, the gradients cannot flow through the maelstrom; this entails the exhaustive unrolling of the network to compute accurate gradients, and its highly biologically unlikely. In contrast, the maelstrom does not need to unroll, which makes it more attractive as well for a biological model that is able to account for the randomness of connections while also preserving computational power. Black lines indicate connections where error can back-propagate and induce learning, red indicate connections that do not allow error to backpropagate. This also allows for a "skip" connection between the interface components to assist in gradient propagation (dotted line).

Relationship to Control Theory

The structure of the network bears a strong resemblance to two well-known results from control theory. The setup of the connection closely resembles a *state space model*, with the input function taking the place of the B matrix, the Maelstrom representing the state variables x, and the output function r matching the C output matrix. The maelstrom network can be seen as a case of *nonlinear state space models*.

In this setup, the input and output networks are trained using the MIT Learning rule (Mareels et al., 1987), where the system is aiming to control via samples from the Maelstrom (which it has no control over), and uses gradient-descent to lower its error over time. One way to view the Maelstrom setup is a fusion of the MIT learning rule, a recurrently updated state vector, and deep neural networks.

Relationship to Reservoir Computing

As the idea of the maelstrom was born about of reservoir computing (Evanusa et al., 2022), it is natural that it would be connected. In (Evanusa et al., 2022; 2023), what can be thought of as a "partial maelstrom network" had been introduced; this included ideas of the maelstrom - the recurrent component - and a readout (or, output network) for a task. It did not, however, include the missing component, which was the sensory cortices. With the input network attached, the network is "complete" in its development phase. However, this does not mean that a simple randomly connected reservoir is the only possible maelstrom - it was used for simplicity and proof of concept, but in our proposal here, we envision any complex recurrently connected state machine that is topologically disconnected from the readout mechanism.

5 Advantages of Maelstrom Networks

As the Maelstrom network paradigm is inspired by - and can be seen as an evolution of - the reservoir computing paradigm, it inherits many of the benefits. It also collects new advantages, as it incorporates deep learning into the mixture in ways that reservoir computing had not.

Provides a Model for Theoretical and Systems Neuroscience

As the Maelstrom paradigm was inspired by reverse-engineering the brain, it is possible to create a "two-way" highway between this proposed model and the brain function, where the brain informs AI research, and the AI research informs neuroscience. Figure 5 demonstrates our general view of the brain's interactions, and how this can map to the maelstrom paradigm. Each module of the Maelstrom Network maps either to a single area of the brain, or a group of areas. This modular structure of the brain, also echoed in (Hadsell et al., 2020), is essential to performing the complex tasks, in a way that does *not* overwrite weights, which occurs when we train networks using the I.I.D assumption on temporal data. The goal here is to create a system that both advances the field of A.I. *and* advances the field of neuroscience simultaneously. In addition, we propose that it is through these multiple-hierarchy loops at varying scales, i.e. within the maelstrom and then within the feedback loop from the maelstrom and the controller, which provides the "state" that serves as the continual sense of self that agents phenomenologically perceive. This could potentially link systems neuroscience structures with concepts of consciousness.

Encapsulates the Embodied Notion of Memory

It is clear that while computer memory relies on memory as an abstract storage, memory in the brain likely evolved for a specific goal, which was to enable the survival of the agent. This means that memory serves at the behest of embodiment, and not simply as an abstract bin that any information can be stored into, as argued in Glenberg (1997). This is also reflected here in the Maelstrom paradigm, in the combination of the memory and the interface. The maelstrom's memory itself is a meaningless string of information, that *only* has meaning with respect to a given readout - the readout which maps the information to a specific prediction, or task. This is one potential solution to the issue of *grounding* of the meaning of activity - while the activity itself is potentially meaningless, it is given embodied meaning by virtue of it being tied to a specific readout. This is also the view espoused in the Neural Assembly theory of Buzsaki (2010).´

Enables powerful training of feed-forward networks while combining Memory

It is abundantly clear that feed-forward neural networks work best with backpropagation, as Transformers (Vaswani et al., 2017) have completely overshadowed recurrent neural networks for sequential processing, even though Transformers contain no memory themselves in their vanilla form (see

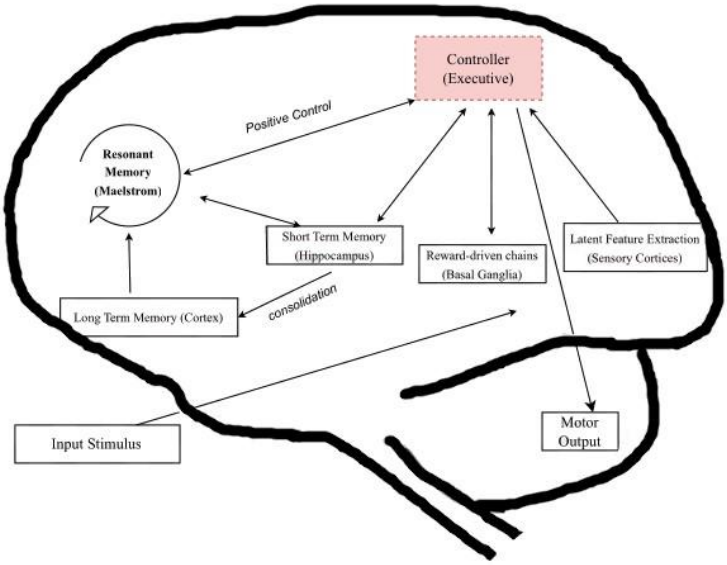


Figure 5: The proposed relationship of the Maelstrom paradigm with functional modules in the human brain. One of the main thrusts of the Maelstrom paradigm was to create a model that both provides advancements to artificial intelligence, and to human brain understanding, simultaneously; this is the ideal goal of artificial intelligence research. The stimulus enters the brain through the sensory cortices, such as the visual and auditory cortex. These cortices contain top-down feedback control (which is exemplified by the feedback from the maelstrom to input controller), but are seen as functionally feed-forward (i.e., the gradients do not pass recurrently). These sensory cortices map to the input network of the Maelstrom. These features are passed to the *executive* modules, which are a large open question in neuroscience as to their location, but we suggest they exist likely in the hub regions that control, receive from, and regulate multiple regions. In the Maelstrom Paradigm, the executive is rolled into the input controller network, however we envision future work as a separate executive module distinct from the sensory network. This network then sends data to the Maelstrom, where it bounces around recurrently in a "storm" of chaotic activity; this is in our view located as a mix of the prefrontal cortex as well as the hippocampus. Also left for future work is consolidation of memory, or learning of memory features, in the maelstrom, as it is left completely untrained in the current simpler iteration. Lastly, the output is sent (either from basal ganglia-learned actions, explicit control, or reflexes), through the executive, to the cerebellum where it learns the correct weights for mapping these to motor actions. The positive feedback control from the maelstrom back to the controller is represented in the maelstrom network via the feedback connection from the maelstrom to the input network, this creates a larger meta-loop within the system at a level above the loops within the maelstrom, and contributes as well to the phenomenon of "self" of the system.

Dai et al. (2019) for a version with a cached memory). While it is true that the Maelstrom removes the ability to attend to far-back sequence elements, this *is necessarily what must happen* if we want to process sequences in real-time, as described below - the only way to attend a weight for each past element is to give the network access to the past elements, which would in turn reduce the model back to a Transformer. In fact, combining a recurrent memory with attention actually *predates* Transformers themselves, as in (Chorowski et al., 2015), and the main innovations of the Transformer were simply *removing* the LSTM (Hochreiter & Schmidhuber, 1997) component (hence the "All you need" of the title) and adding the multi-headed features. In this case as in Chorowski et al. (2015), the readout attention mechanism still needs access to the entire length of the sequence, even though it has a memory, because of these limitations being placed by the readout attention, and the fact that it must pass gradients through the LSTM to learn the weights. The Maelstrom paradigm also allows for additions to learning or attending to the Maelstrom - its requirement is simply that the gradients cannot flow back through it. Naturally, adding in a memory necessarily means aggregating the past in some way into a compact latent representation, which does lose information in the process - there is no avoiding this. The fact that Maelstrom Networks are a general principle also

means there is room to incorporate new elements in the system. For example, to enable even more associative memory, the Maelstrom can be trained using the Self Organizing Map (Kohonen, 1990) procedure to agglomerate similar representations.

EASILY INCORPORATES INTO EXISTING FEED-FORWARD NETWORK STRUCTURES

Recent work as in (Hutchins et al., 2022; Wu et al., 2020) have begun to investigate using cached memory representations to alleviate the computational burden that Transformers have when dealing with long sequences. However, these architectures requires highly specific structures for the memory to incorporate them in, and some are highly tuned towards language models and not general temporal data. The Maelstrom paradigm, in contrast, requires no strict guidance on what the memory must look like. The key insight is that *the memory representation passing through the maelstrom is fed into the readout, in such a way that to the readout, it appears to be input stimulus.* This reflects a nested hierarchical view that Friston et al. (2016) touches on; this notion that to sub-regions of the network, input coming in appears the same as the stimulus does to the input layers of the network. This general paradigm, we believe, applies to all temporal data.

ALLOWS FOR REAL-TIME LEARNING AND INFERENCE OF TEMPORAL DATA

One of the major strengths of Transformers, their learning ability through feed-forward networks, is also one of their greatest weaknesses, as they must take the entire length of the sequence to process the sequence. While it is possible to train a Transformer to map all possible sub-sequences in a feed-forward manner, this becomes computationally infeasible very quickly, as the number of sub-sequences grows as N^2 with the number of sequences.

Adding this Maelstrom, however, that is completely unhooked from the gradient learning, turns the problem of learning sequences into a controls problem, where the input at each timestep is simply a feed-forward learning of the current memory state vector, and its mapping to a prediction. This means that the entire length of the sequence is *no longer needed*, which means that the sequence can be learned in an online fashion, *and* that inference can be performed in an on-line fashion. This is critical for any use-case where neural networks need to be running in time in the real world - which, by most measures, would include most real-world applications.

ALLOWS FOR NEUROMORPHIC HARDWARE IMPLEMENTATIONS

Lastly, as the maelstrom does not require gradients to backpropagate through the recurrent component, it is extremely amenable to newer neuromorphic and bio-inspired hardware implementations, such as through the Intel Loihi (Evanusa et al., 2019), or Memristor technologies (Thomas, 2013), or even FPGA implementations. These technologies eschew with the traditional CPU chip processors and instead solely work by modeling *just* the neuron activations. This reduction of the function of the chip severely limits the kind of processing it can do (for normal computer programs), but supercharges the use-cases when it runs neural networks. As the memristor and loihi chips in particular have specialized hardware, performing Backpropagation Through Time or other complex temporal operations are burdensome; running a maelstrom, however, that requires no internal updating and simply runs as an autonomous dynamical system, is much more doable. The fact that the Maelstrom is disjoint from the feed-forward component means that the feed-forward component can be implemented separately in GPU hardware, and read from the neuromorphic Maelstrom when needed. This would fully utilize the power saving capabilities of neuromorphic hardware (as the recurrent component is computationally expensive with BPTT), in a time when it is becoming abundantly clear that the energy usage for current deep learning is unsustainable.

ALLOWS FOR ONLINE OR CONTINUAL LEARNING

As the sequence memory, encapsulated in the maelstrom, is unhooked from the gradient learning process, this can allow future instantiations to deal with the *continual learning* problem. The continual learning problem is defined as the ability of a neural network to train indefinitely while it continues to perform testing inference. This is impossible with current deep learning networks due to the fact that the gradients from new samples completely overwrite the old weights with no regard to their importance. For example, a network trained on five image classes, but then continued to train on 5 new classes without access to the old classes, will completely forget the first data. It has been proposed in (Hadsell et al., 2020) that a modular architecture, for example the Maelstrom paradigm, can aid in the

continual learning problem, by unhooking the gradients, and in addition by removing the I.I.D assumption that underlies the current setup. The unhooked Maelstrom will allow for the network to not overfit to a given task along the sequence, while also not overwriting later sequence elements.

6 CONCLUSION

In this work, we propose *Maelstrom Networks*, a novel modular neural network architecture that unhooks a sequential memory, representing the prefrontal cortex and hippocampus, from the gradient pass of two feed forward networks used to control the memory. The input network acts as the sensory cortex to the system, inputting the correct features while maintaining balance of the maelstrom, and the readout takes outputs from the maelstrom and applies them to an action or task, similar to the motor cortex. we propose that this work is the natural evolution of work done in the early 1960s, which laid out a larger modular structure for the nervous system but left out the critical issue of sequence memory. we connect this work to control theory and note that the maelstrom is a nonlinear state space model, trained using the MIT control rule; this also relates the brain to notions of control as well, as we map each region of the brain to components of the maelstrom as well. It is our hope this helps usher in a new era of neural networks that specifically focus on sequence memory, not as an engineering trick to aid in performance, but as a fundamental property of networks.

MAELSTROM NETWORKS FOR TIME SERIES CLASSIFICATION

Matthew S. Evanusa
Department of Computer Science
University of Maryland
College Park, MD

Cornelia Fermuller¨
Institute for Adv. Computer Studies
University of Maryland
College Park, MD

US Naval Research Laboratory matthew.evanusa@gmail.com

Yiannis Aloimonos
Department of Computer Science
University of Maryland
College Park, MD

ABSTRACT

Neural networks have shown remarkable progress in the last decade on pattern recognition tasks. However, these networks work on the standard machine learning assumption of independently and identically distributed (IID) data, which does not take into account the natural temporal-causal relationship between data points in the real world. The human brain, owing to millions of years of evolution, has evolved naturally to deal with time via resonating cell assemblies, which gives us a stable state and our continued perception of realty. These assemblies also serve as a temporal inductive bias, which processes data not in an IID fashion, but according to temporal cause-effect. In this work, we take inspiration from Hebbian assemblies in the brain, as well as random neural networks, and utilize the newly proposed *Maelstrom* network paradigm for time series classification tasks. We demonstrate that the network, through its modular incorporation of the dynamical, chaotic *Maelstrom*, is able to maintain a compressed history of its past in a memory unhooked from the gradient error of the task. We evaluate the model's performance on a novel time series classification task using the UCI Time Series Classification data collection, where we test the network's ability to remember previous elements of the same sequence and use them for predictions. We evaluate and demonstrate the Maelstrom paradigm's effectiveness on several multivariate datasets, and compare against state of the art deep networks. We propose that this effectiveness shows the Maelstrom paradigm's use not only as a brain model, but as a new direction for deep learning. In endowing the network with a sense of its own past and self-awareness with respect to it, we also hope this paradigm allows future networks to approach closer to intelligence that displays what we may think of as phenomenological consciousness.

Distribution Statement A. Approved for public release: distribution is unlimited.

1 INTRODUCTION

Deep neural networks have in the past decade achieved remarkable results, but these successes mask the missing capabilities in the temporal domain. The networks are designed to work on, and excel at, static data, rather than the temporal data that we experience in the real world. Thus, getting them to work in the temporal domain involves re-tooling the same static networks to apply them to time data, such as via windowing, or zero-padding. These engineering feats solve some problems but do not progress research in deep learning towards fundamentally addressing temporal data, as they are augmentations of the data rather than imbuing the network with a persistent sense of time. To do this, here we argue we must go back to the origins of deep learning as in (Rosenblatt et al., 1962), and think about a different direction.

What is deep learning missing that biological agents possess with regards to time? What makes biological agents perceive a persistent state of time and self? These are the questions we aim to address in our work. Here, we demonstrate the efficacy of a new model, the *Maelstrom Network* paradigm, which attempts to address the shortfall of integrating notions of time directly into the network, rather than via augmentations of the data itself. We show that by giving a modular dynamical state to a specific network meta-topology, we can integrate time into the network and utilize the feed-forward networks to their best efficacy. We show that doing so is not simply an exercise in biological reverse engineering, but that this is the right focus to reverse-engineer that can produce real progress in deep learning systems towards more "true" artificial intelligence systems.

DATA IS TEMPORALLY ORGANIZED

While modern neural networks are built off of the concept of "independently and identically distributed" data, this ignores the fact that in the real temporal world, data is not organized as a collection of random data thrown about (See Fig. 1). Rather, the data is temporo-causally organized according to the same *strands* of time, which are related to one another via the frame of *action*. It is a given action that causes matter to shift from one state to the next, and thus, viewing through these actions ties points in a sequence along the same temporal strands. To address the temporal structure of data, new inductive biases, (i.e, new network structures) must be implemented that take these temporal relationships into account (Goyal & Bengio, 2022). We refer to systems that can remember data from the same temporo-causal strands as those displaying *sequence memory*. This is not the same as *memory*, which is a more general concept that just refers to the ability to learn from prior training data.

THE NEED FOR A TEMPORAL INDUCTIVE BIAS AND STATE

It has been shown repeatedly that inductive bias is a key, perhaps *the* key, element in neural network success. This is potentially because neural networks choose the simplest solution first (Mansour et al., 2019), which intuitively makes sense, as backpropagation is inherently an energy minimization process. In choosing these simpler solutions, it cannot "self-discover" more complex mechanisms for solving problems; it simply maps input to output in the lowest-energy method possible. This can be one reason why the story of deep learning has really been one of inductive bias, in the form of the topology of the network: the simplest structure, the multi-layered perceptron, has more learnable parameters than a similarly-layered Transformer (Vaswani et al., 2017) of the same depth, and yet the Transformer far outperforms the perceptron. This is likely, given this understanding of inductive bias, because the simple solution that the MLP takes does not learn the more advanced structures that the Transformer gives us for sequence processing.

This temporal bias, we propose, is implemented in living creatures by two mechanisms: a *state* of the system, which is a collection of variables or systems that persist activity across time vis recurrent connections. This state is also *topologically unique* from other regions that perform more task-mapping, such as the motor cortex. It is also distinct from the input areas that map the stimulus to additions to the state, such as the sensory cortices. We aim here, then, to model a topologically modular neural network that contains functionally feed-forward components, as well as distinct functionally recurrent state components that act as persistent temporal variables.

While neural networks have excelled at feed-forward pattern classification, one glaring omission remains in the state of the art models for temporal data: these models lack an internal memory

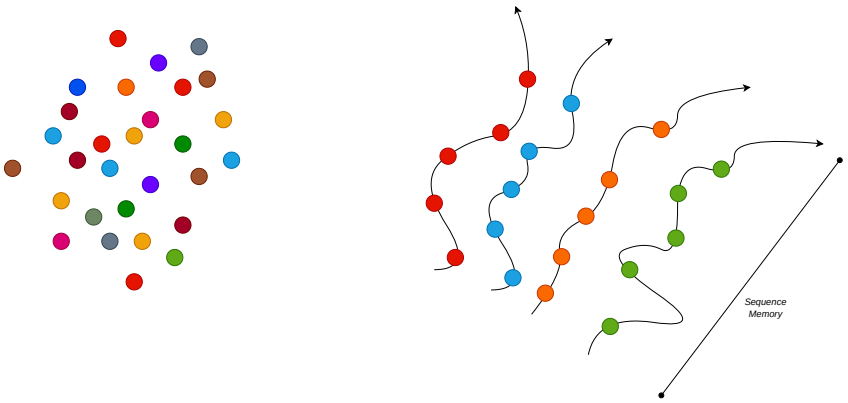


Figure 1: Data in the real world is not I.I.D and unstructured, but is organized according to the physical principles of temporal cause and effect. *Left:* The world if it were a jumbled set of uncorrelated data. This is the world of sequence-memory-less deep learning, where we only care about remembering data points from the population as a whole, and their correlations. This world assumes no underlying structure for the data in time. *Right:* The view of temporal sequences, where the data points are not seen as randomly uncorrelated data, but have a temporal-causal relationship along the stream of time, in line with the physics of the universe. The black line here represents points that are strongly temporally-causally related, for example, the apple's trajectory as it falls from the tree, or the joy one experiences when traveling to a new country. This relationship is with respect to a certain reference point, which is philosophically similar to the reference-oriented views of quantum mechanics. The reference point that I will argue is with respect to action; as action creates a temporal cause and effect relationship. The ability of a system to remember a point (of the same color in this visualization) at different points along the same temporal sequence is what we refer to as *Sequence Memory*

of their activations -i.e., they lack a *state* - which would allow them to remember and reuse their previous activity for processing sequences. For static data, such as images, the model does not need a memory, however this data is not representative of the kinds of data that we expect agents to experience in the real world. All biological agents have clearly evolved to handle *temporal* data based on the given laws of the universe, and have developed mechanisms for remembering the past inputs in the form of *memory*. This memory acts as a state variable, which evolves in time as a function of its past activity as well as its new inputs.

Most modern neural networks have completely eschewed the idea of using a state or sequence memory (with the exception of recurrent neural networks), and opted instead to go ”all-in” in feedforward networks, which now dominate even the temporal landscape. What used to be a connectionist landscape dotted with recurrent networks is now completely taken over by transformers and convolutional neural networks, which, by all measures, have shown remarkable success, but inherently do not possess (in their standard implementations) any inherent sequence memory or notion of temporal modeling. These networks either work by windowing (in the case of CNNs) or by simply passing in the entire length of the sequence, and use powerful gating techniques such as multi-headed attention to focus on particular segments. This latter approach has led to an arms race of ever larger ”context windows”, where the network is tasked with shouldering ever longer sequences, rather than to keep a latent representation of the past (memory), in which elements are encoded for future use. Intelligent agents in nature, however, evolved a state in the form of the dynamical network that sits in our working memory regions, and this state not only allows for online learning and predictions, but also gives us our sense of self and continuity through time. Recently, the Maelstrom Network paradigm was proposed, which aims to shift the landscape back towards the cyberneticist days of embodied AI, but using the powerful new tools that deep neural networks have given us.

Critically, while some newer network architectures are being imbued with a notion of state, they are doing so not as a critical element of the network, but merely as a feature to help compress the context window, for example in Transformer-XL (Dai et al., 2019). Notably, this does not address the problem of continual learning, where weights are overwritten by previous timesteps when data is presented sequentially. This seems to be the case when the network is trained entirely by the same gradient signal using backpropagation; the brain, however, is modular, with certain functions being topologically isolated from others, preserving their learning when other regions undergo

learning (Hadsell et al., 2020). It is argued that this modular approach is one possible solution to the continual learning and catastrophic forgetting problem in deep learning. In this work, we utilize the Maelstrom paradigm, which specifically utilizes a modular dynamical memory: this memory is not affected by the weight updates of the input (sensory) and output (motor) neural networks, and thus would not undergo catastrophic forgetting in a sequential learning problem.

To prove the network's veracity, in this work, we present an implementation of the Maelstrom Network paradigm using deep neural networks as the controller and readout functions, and apply it to an online time series classification task, in conjunction with a relatively simple variant of the maelstrom we deem the phasor. We demonstrate that including the state dramatically improves the predictive power of the network on multiple benchmark time series classification datasets. We also propose future directions for where to take these networks next.

2 Related Work

Early networks based around Hebbian learning, such as Hopfield networks (Hopfield, 2007) have a state consisting of resonating "cell-assembly-like" neurons, which are trained using Hebbian plasticity. Because the network does not incorporate an input or readout network, the patterns are required to be simple, and complex feature extraction is not possible, limiting their wider application in the real world. This model of the memory, however, has profound implications for how we design our memory module in the future. Most modern feed-forward networks do not contain any state, even those that are considered state-of-the-art for sequential tasks, such as CNNs and Transformer models (Vaswani et al., 2017; LeCun et al., 1998). In order to have a state, a separate memory, or activity that feeds back in a cycle to the network, must exist. Recurrent Neural Networks (Salehinejad et al., 2017) are the class of networks that contain a cyclical activity, however their performance has not matched feed-forward networks for modern tasks such as regression or classification. Improvements on vanilla RNNs in the form of Long Short Term Memory (Hochreiter & Schmidhuber, 1997) introduced multiple gates and two different state variables, with the aim of forcing the network to learn long-term dependencies. While these worked quite well and are the pinnacle of current RNNs, the fact that removing the cell state in the form of the GRU (Chung et al., 2014) performs as well shows that the problem must rest elsewhere. It is notable that the original attention mechanism was developed for these LSTMs, and the Transformer's main development was removing the state. Certainly, something is not working optimally with gradient learning and a recurrent state. Based on our intuition in the prior paragraphs, we can argue that the problem lies in the fact that the network is homogenous, and the "memory" is trained using the same gradient that the task is trained for. In doing so, the memory loses its inductive bias potential, as the network finds a low-energy equilibrium state that avoids the use of its memory. These networks also suffer from the same issues of continual learning as feed-forward backpropagation networks.

Newer work attempts to imbue memory into Transformer architectures, effectively bringing them back towards the LSTMs with attention. These works, such as Transformer-XL, are a bit closer to our model, but still suffer from the homogenous training of memory and task, and thus do not contain a separately trained memory unit and are weak to catastrophic forgetting.

The reservoir computing paradigm is the only modern model that contains a topologically distinct recurrent and readout component. However, it does not contain provisions for learning the input to the reservoir. and it does not (in its vanilla state - see (Evanusa et al., 2022)) contain a deep readout mechanism. Our work here can be seen as taking the insight of the topologically distinct recurrent memory as in reservoir computing, but placing it in a larger context of deep neural networks and state.

3 Maelstrom Network

Before we move on to the specific implementation, we must first define the general class of neural networks that we seek. To refresh, we seek a neural network that 1) has a dynamically altering set of state variables, 2) does not pass gradients from the task-learning procedure through to in the states (i.e., memory is *associative* in nature) 3) is able to learn control for these states nonetheless, and 4) combines the power of deep feed-forward networks. These ideas lead us to a new paradigm for recurrent learning that we combine here, which we term *Maelstrom Networks*. Borrowing from reservoir computing, we leave the recurrent component random and unchanged. Borrowing from control theory, we attempt to learn controlling mechanisms that help guide or steer the recurrent component, and

read out from its effects, without directly interfering with its activity. Borrowing from deep learning, we use a powerful deep neural network to learn the patterns of the memory (or maelstrom), and fully utilize this pattern capability in a very high dimensional complex space.

This has strong connections to earlier control theoretical approach to model control when the model was unknown and had to be approximated from data. In particular, this is related to the MIT control rule (Mareels et al., 1987), which was abandoned for a lack of convergence properties. In this case, we do not have fine-grained control over the maelstrom, just its inputs and its readout, and thus we treat the maelstrom as a somewhat uncontrollable chaotic system that can be guided but not set in detail. This is in contrast to the idea for recurrent neural networks, where we assume we can calculate exact gradients through the recurrent states; but is more similar to reservoir computing, where we do not. This is the origin for the name *Maelstrom*: the memory state vector itself is a chaotic mass of activity that rages like a storm; we cannot peer through it but we can attempt to control it from the boundaries.

In some sense, the *Maelstrom* idea is analogous to the idea of using deep neural networks to approximate patterns in an opaque chaotic system using the MIT control rule: where once the control could not be guaranteed, we now leverage the power of state-of-the-art neural networks with impressive mapping capabilities to perform this task.

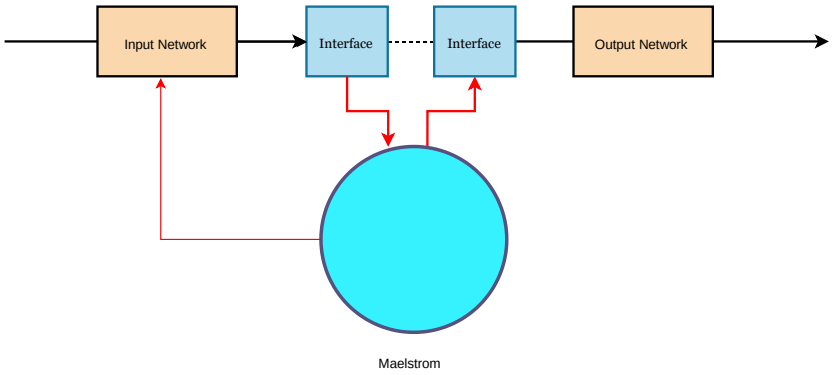


Figure 2: The Maelstrom Network paradigm. The input is fed from the world through a feedforward neural network to an interface neural network, which reads to and from the *Maelstrom*. The Maelstrom is a recurrently-connected state that agglomerates activity, dictated by the input network, and is *topologically unhooked from the gradient learning of the task*. This unhooking prevents the network from overwriting the inductive bias that we wish to endow the network with, which is a rich space to remember activity. It is also understood that overloading the memory units with feature learning units is not a good strategy (Gemici et al., 2017), thus we want to keep the memory unit (the maelstrom) separated from the task-learning (the output network). The gradient is passed across the top from the output to the input, and both networks train using the snapshot of the maelstrom at the given time. This is equivalent to the MIT learning rule Mareels et al. (1987), where the gradient is not unrolled for a state space. A feedback loop also occurs between the maelstrom and the input network, allowing the input to understand what the state of the maelstrom is before inputting to it. This closed loop between the maelstrom and the input network is, we argue, the most basic form of self-awareness that can exist in a neural system, and can serve as the foundation for more complex ones.

MAELSTROM MODELS INDUCTIVE BIAS VIA CELL-ASSEMBLY WITH A READOUT

In the brain, the core computational unit is hypothesized to be *cell assemblies* (Hebb, 2005), where small clusters of neurons capture the input data. It is less clear or understood how these cell assemblies actually perform the processing as a unit. One possible hypothesis is the "readout" paradigm, where the activity of the cell assemblies are seen as patterns, which are learned by a separate readout mechanism; this readout acts somewhat like eyes on the assemblies. (Buzsaki, 2010). A core´ idea is that the assemblies are topologically distinct from the readout mechanism, and further, that they act in time to resonate temporal signals within a state. Our working hypothesis here on which we build our system is that these cell assemblies, in addition to extracting features, serve as memory units, which provide the topologically distinct dynamical state from which a more functionally feed-forward system can learn the patterns. The state of these cell assemblies resonating in time also serves as the persistent, continual

"existence" of the agent, giving it a sense of itself through time. In the Maelstrom model, the cell assemblies are modeled by this unhooked dynamical, recurrent pool of neurons, which are then read-out by a functionally feed forward deep neural network.

MAELSTROM PRESERVES TEMPORAL INDUCTIVE BIAS

If we want to endow a network with a certain inductive topological bias, we must ensure that the network cannot alter this during the training procedure as discussed earlier, or it will simply choose to erase this structure in favor of the easiest solution. The Maelstrom paradigm, through the unhooked Maelstrom that is topologically separate from the task, performs this task by modularizing the network in such a way that the network cannot "be lazy" and overwrite the temporal inductive bias that we want to endow it with - in this case, the recurrently connected memory unit that performs the task of persisting the state across time, as well as agglomerating activity into a temporal averaging of sorts.

4 IMPLEMENTATION

We now present one implementation of a Maelstrom Network in a deep learning framework, which has the minimum necessary requirements to be classified as a Maelstrom network. It contains a learnable input-to-maelstrom controller network C, a Memory Maelstrom M in which gradients do not flow through, a readout interface from the maelstrom R that also takes input from the controller (and which allows gradients to flow through to the controller), and a summarizing output network O that performs the final prediction of the signal - for machine learning tasks, this is a regression or classification.

Building off of multiple lines of research, from recurrent neural networks, reservoir computing, and control theory, we present our proposed model, which we are calling "Generalized Memory Network" (to avoid confusion with *Memory Networks*, (Weston et al., 2014)). A successful temporal classifier can be thought of a composition of three components: a memory component, that keeps track of its own past activity, and a classification component, that looks at the memory component and decides based on the current input, and a combination of its past input, what the next class label should be. This is unsurprisingly also the critical basis for the Maelstrom paradigm. A generalized formula for a nonlinear state space model that instantiates a Maelstrom Network is thus:

$$M(t) = C(u(t))),B(M(t-1))$$

$$Y(t) = O(R(C(u(t)),M(t))$$

for input at time t denoted $u(t)$, the memory decay function B, controller function C, summarizing readout O, and memory readout R. This is related to the state space model representation from control theory as described earlier, where the memory matrix M is taking the place of the state variables x, the function q acts on the input like matrix B, and the function b acts on the past memory like matrix A. Here, each function is not a matrix but is rather a nonlinear function that can be approximated. Here we use function approximation to approximate the functions C, B, R, and O via artificial neural networks. The memory M in this work is a simple matrix; in future work, we will continue to elaborate on the formulation of the matrix memory, but for the purposes of this work we opted for simplicity. The matrix B acts like the reservoir's recurrent weight matrix, "bouncing" activity around. As with reservoirs, a huge amount of room is open for how to design this weight matrix, but for this work we keep it random. The skip connection from C to R is not technically needed as the gradient can flow through R, but we and others in previous work have found that a skip connection can be beneficial to learning gradients (Srivastava et al., 2015; He et al., 2016; Evanusa et al., 2023). We show a diagrammatic instantiation of this in Fig. 3.

4.1 RELATIONSHIP TO RNNS AND RESERVOIR COMPUTING

It is apparent that this formulation bears a strong resemblance to RNNs and reservoir computing, an overview of which can be found in (Jaeger, 2001). The q function is essentially the W_{in} matrix, feeding into the system, the memory M is our reservoir state space, g is our readout. There are however two new functions that we do not have in reservoir computing, namely b and c, which allow the model greater flexibility to control the dynamics.

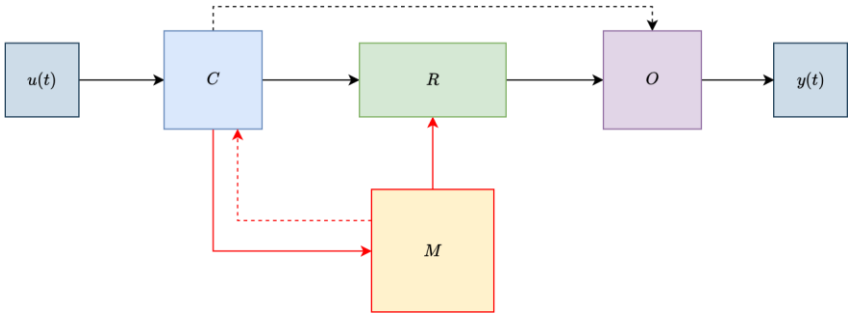


Figure 3: The Maelstrom Network paradigm implemented with deep neural networks as the controller and readout mechanism; $u(t)$ and $y(t)$ are the signal and target at time t respectively, C is the controller network that guides or steers the memory, M is the Maelstrom that is a memory component disconnected from the network's gradient flow as described above, R is the readout module from the maelstrom, and O is the task learning network that combines the readout and other signals into a final task output. Solid lines represent connections that are critically required for information to flow, while the dash lines are important but non-critical. Black lines indicate connections where gradient may flow backwards, and red indicate connections where gradient cannot flow; the red around the memory maelstrom M indicates M cannot pass gradients through itself in time as well. The top dash line represents a ”skip”connection that was found to be useful in reservoir applications in Evanusa et al. (2023), and the bottom dash represents a ”closing of the loop” between the memory and the controller, creating a sense of ”embodiment” that causes the input matrix to be aware of its own memory activity, in choosing activations. This is similar to theories of self-awareness of processing as promulgated by theories of cognition such as those from Friston or Piaget (Friston et al., 2016). Note that in the case without the skip connection on the top, the only gradients that the network can receive are through the

5 Phasor Maelstrom

The Maelstrom paradigm allows for any kind of learned or unlearned memory as long as it is topologically distinct from the gradient error of the task. For simplicity and ease of demonstration, we chose a rather simple, but powerful, variant where the memory is simply spinning “knobs” of activation that are put through sin activations. This memory does not have interactions between the memory nodes, but each memory “neuron” resonates in time. This allows the simplest possible trace of past activity on which to test our proposed network.

In the brain, there are two theories to how the neurons can encode information, the *rate encoding* and the *temporal encoding*. Most of modern computational neuroscience is built around the rate encoding paradigm, but it is likely the case that the exact timing of neural firing plays a key role in many mechanisms (Engel et al., 1992). Mathematically, we can represent the timing of neuron firing as a complex number, with the amplitude representing the strength and the angle representing the *phase*, or the time of arrival offset from a starting reference point. In addition, an issue with the memory unit itself, and common to reservoir computing, is how to normalize the reservoir activity such that it doesn't explode. If we rather encode the values as a sin of the activation, the values are bounded and continue to rotate around the unit circle indefinitely We call this addition *phasor memory*, and demonstrate that its inclusion does indeed completely negate the need for any kind of negative weights or eigenvalue normalization. The downside, however, is that we lose the graph's ability to communicate with itself. To be certain, one of the inconsistencies with echo state networks is that they are fundamentally build upon the rate encoding framework (by including a squashing function and sum), but work in the temporal domain; this amounts to not changing activities over time, but rather change of rate of firing over time, an issue with LSTMS as well. This phase array also serves the purpose of aligning with the inherent rate encoding of the reservoir activation as it stands. Future iterations of the network will include more complex memory, such as those with random recurrent pools of neurons, or recurrent pools of neurons that are trained via unsupervised, self-organizing methods.

6 Temporal Online Training and Testing Procedure

The training procedure operates similarly to standard deep learning frameworks; however, as we are interested in the memory capacity of the network in an online, temporal fashion, we make certain augmentations to the procedure. Namely, it is specialized to operate on the sequence in windowed frames (in an online fashion), rather than on a per-sample basis. We start with a dataset D of all of our training samples; which we split into a D_{train} and D_{test} train/test split. For each sample $S_i \in D_{train}$, we *windowize* the dataset into windowed "frames" that are passed through the model; these windows can be seen realistically as the smallest individual data sample that the agent receives at a given time t. As in biological agents, or neural networks deployed in the real world, each window itself passed through in a feed-forward manner, and it is the responsibility of the model to remember windows of information that occurred earlier in the sequence. We loop through each sample S_i for $i \in I$, the number of samples. For a given S_i, we windowize the sequence into an ordered set of windows ${S_w}^i$, for $i,i..j$, where j equals the max window index - i.e., the number of overlapping sub-windows in this sequence. For each window, in temporal order, such that the ordered index of $i < i+1$, we pass ${S_w}^i$ through the network; this constitutes the $u(i)$ from earlier. This ${S_w}^i$ passes through the controller C, then into M, and is read out by R and then O. Once O outputs a prediction, this is compared against the true prediction (here, a classification label for the window), and the loss is propagated backwards using crossentropy loss. For a regression task, we would use a mean squared error loss. As in Fig. 3, this loss is not backpropagated through the red connections (anything involving the memory), but is propagated to the controller. Following this backward pass, we do *not* clear memory until the sequence is complete; it is important that M persists throughout the whole sequence from index i to j. For the comparison studies for the Transformer and Temporal Convolutional Network Bai et al. (2018), each window is passed into the network in precisely the same manner.

7 EXPERIMENTS

To test our model against the state of the art deep neural networks for signal processing (or sequential) tasks, we use the benchmark UEA Multivariate Time Series Classification Collection (Ruiz et al., 2021). This collection contains multiple datasets spanning various tasks, such as brain scans, gesture recognition, and finger movement sensor detection. We examine the performance on multiple datasets from the collection to demonstrate the broad applicability of this memory approach; we list each dataset used in Table 1. We performed an analysis of the UCI dataset from Bagnall et al. (2016); Ruiz et al. (2021), where each dataset was described and thoroughly evaluated. The tabular results posted in their work do not apply to our task, as we do not look at the entire sequence as they do but rather *online windows* in sequence, but the general characteristics of the datasets were gleaned from analysis of the results and the class balances. Datasets in our evaluation were chosen for their difficulty (i.e., multiple channel inputs), non-trivialness (as in the case of *Coffee and the like, where*), and correctness (as in the case of *Heartbeat*, where the top performing networks' score is roughly equivalent to assigning the same label to all data). The resulting networks also show a large spread in performance, indicating that they are difficult tasks, rather than being difficult because the dataset is unlearnable or incomplete. As the strength of the Maelstrom network lies in its temporal inductive bias, this inductive bias must be learned via a higher-order process (such as evolution for living creatures). Here we use a Bayesian hyperparameter optimization from Optuna Gaussian Process optimization (Akiba et al., 2019) to find the correct hyperparameters for parameters such as the number of controller and readout layers, the size of the maelstrom memory, and scaling factors going into the Maelstrom for stability. We found that the deep comparison networks were not as sensitive to parameters, as they learn everything through one gradient signal, but we performed a grid-search for these parameters owing to the smaller parameter search space.

Table 1: Comparison F1 (micro/macro) of the dynamic, temporal online time series classification task, comparing Maelstrom networks with feed-forward variants. The sequences were run in temporal order, but windowed in sequential order in different sizes. A network with a temporal inductive bias, such as living creatures, should use the temporal causality of the sequence, and sequence memory therein, to remember previous fragments. Each result was averaged over five independent runs, for each network, for each dataset; the standard deviation for these runs are also shown in ±. As we can see, with full information (omniscient, or omni) of each timestep - in a sense, "cheating" at the memory task here - , deep learning becomes an IID pattern matching problem (Full Information), but once the network must remember the previous timesteps, the deep learning approaches suffer. Shown are different fragment sizes, 4-F and 2-F (for 4-Fragment and 2-Fragment). For the 2-Fragment, the network has very little temporal information and must use spatial information in the time slice. We can see that even though the Maelstrom network also only has access to this time slice, it's built in continual, unlearned memory gives it

information from the past. Interestingly, the LSTM, which actually does keep a memory, completely fails on the longer sequence tasks even with omniscient information such as in *Epilepsy*, as its gradient training likely results in vanishing gradients, and over-averaging of the sequence weights. The full sequence results for the Maelstrom are not shown, as this has been proven in prior work to be as good or better than the deep models, for full information (Evanusa et al., 2023). Due to the larger number of hyperparameters such as memory size, controller layers, etc., which are inductive biases, we used the Optuna Gaussian Process hyperparameter optimization (Akiba et al., 2019). For the deep networks, as there were only a small number of parameters (learning rate, number of layers) we did a simpler grid search.

Dataset (4-F)	MLP	LSTM	Transformer	TCN	Maelstrom Phasor	Maelstrom Random
ERing	.425/.374 ± .082/.074	.696/.668 ± .021/.022	.719/.702 ± .031/.023	.673/.647 ± .026/.030	.924/.917 ± .024/.021	.913/.901 ± .038/.037
RacketSports	.442/.383 ± .038/.032	.533/.505 ± .013/.037	.729/.709 ± .023/.031	696/.673 ± .032/.044	.893/.903 ± .044/.042	N/A
NATOPS	.623/.611 ± .019/.026	.575/.543 ± .033/.028	.596/.582 ± .038/.030	.613/.591 ± .033/.033	.782/755 ± .014/.029	N/A
Libras	.197/.131 ± .047/.031	.658/.606 ± .049/.063	.679/.624 ± .041/.039	.512/.450 ± .042/.038	.817/.796 ± .061/.069	N/A
Epilepsy	.873/.868 ± .038/.032	.936/.930 ± 0.012/.011	.932/.930 ± .022/.023	.909/.910 ± .032/.033	.951/.042 ± .019/.017	.959/.958 ± .017/.017
Dataset (2-F)	MLP	LSTM	Transformer	TCN	Maelstrom Phasor	Maelstrom Random
ERing	.409/.367 ± .047/.043	N/A	N/A	N/A	N/A	.876/.869 ± .020/.024
RacketSports	.483/.457 ± .051/.029	N/A	N/A	N/A	N/A	.884/.900 ± .056/.046
NATOPS	.587/.574 ± .018/.014	N/A	N/A	N/A	N/A	.709/.693** ± .054/.036
Libras	.080/.037 ± .037/.018	N/A	N/A	N/A	N/A	.755/721 ± .069/.095
Epilepsy	.510/457 ± .108/.101	N/A	N/A	N/A	N/A	.956/.954 ± .010/.013
Dataset (Omni)	MLP	LSTM	Transformer	TCN	Maelstrom Phasor	Maelstrom Random
ERing*	.973/.970 ± .04/.05	.940/.926 ± .014/.0282	.967/.969 ± .040/.041	..940/.931 ± .043/.040	N/A	N/A
RacketSports*	.768/.784 ± .042/.022	.877/.874 ± .035/.031	.916/.904 ± .037/.039	.845/.863 ± .083/.063	N/A	N/A
NATOPS*	.922/.914 ± .041/.035	.906/.896 ± .025/.026	.961/.960 ± .046/.049	.944/.941 ± .065/.065	N/A	N/A
Libras*	.517/.454 ± .100/.125	.794/.744 ± .042/.066	.889/.836 ± .043/.074	.872/.833 ± .037/.059	N/A	N/A
Epilepsy*	.857/.819 ± .056/.077	.528/.471 ± 0.150/.124	.964/.959 ± .025/.030	.964/.963 ± .036/.032	N/A	N/A

8 Results and Discussion

We can see from Table 1 that the Maelstrom network perform either on par with (for simpler dataset) or greatly exceeds the performance of the memory-less and non-online deep neural networks. This is because these networks do not have access to a running summary of their past, and in some cases, such as in the * variants, have access to the past but are incapable of using the gradients properly (as in the case of the LSTM). For the LSTM, the poor performance on the "full" task correlates to the length of the sequence: for longer sequences, the vanishing gradient problem kicks in, and the network begins to average too much over its weights for a given timestep. In the cases where the network only has access to a small window, which is the case for biological embodied agents, the networks that have "blinders" on, and only look at the features of a given window, fail to classify the task, such as in *RacketSports*, where the result is a random chance. This proves that the modular network is able to organize around an equilibrium point for a good solution to the task, even though it must rely on a dynamic memory module, *and* that memory does not propagate errors through itself. It must learn the weights of the controller network via the gradients from the output task, via the MIT learning rule. Future work will derive more optimal ways to train this controller network, but for the purposes of this study, we show that the gradient from the output is sufficient for the class of temporal tasks in these datasets. This shows both the strength (in terms of being able to classify when all data is present in a temporal instant) but also the major weakness of deep learning architectures when it comes to online temporal tasks, where the data comes in in specified windows - the conditions that are present if deep learning is ever deployed to real embodied systems.

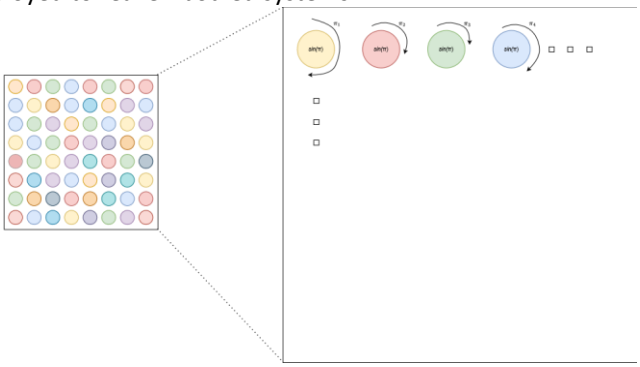


Figure 4: The proposed Phasor Memory schema that prevents value explosion and ensures memory stability. Shown on the left is a color depiction of the memory M, where each shade of color represents different values stored at

each index. Zooming into this, we can see that the values stored are actually the phase (sin) of the absolute value (representing the angle π). Given that each function approximator is a powerful neural network, it is capable of learning hidden patterns in any medium. Therefore, it is no different a task for it to learn the patterns in the phases of the memory versus the absolute value.

Deep Reservoir Networks with Learned Hidden Reservoir Weights using Direct Feedback Alignment

Matthew S. Evanusa *
Department of Computer Science
University of Maryland
College Park,
mevanusa@umd.edu

Cornelia Fermüller
Institute for Adv. Computer Studies
University of Maryland College
MD Park, MD

Yiannis Aloimonos
Department of Computer Science
University of Maryland
College Park, MD

Abstract

Deep Reservoir Computing has emerged as a new paradigm for deep learning, which is based around the reservoir computing principle of maintaining random pools of neurons combined with hierarchical deep learning. The reservoir paradigm reflects and respects the high degree of recurrence in biological brains, and the role that neuronal dynamics play in learning. However, one issue hampering deep reservoir network development is that one cannot backpropagate through the reservoir layers. Recent deep reservoir architectures do not learn hidden or hierarchical representations in the same manner as deep artificial neural networks, but rather concatenate all hidden reservoirs together to perform traditional regression. Here we present a novel Deep Reservoir Network for time series prediction and classification that learns through the non-differentiable hidden reservoir layers using a biologically-inspired backpropagation alternative called Direct Feedback Alignment, which resembles global dopamine signal broadcasting in the brain. We demonstrate its efficacy on two real world multidimensional time series datasets.

1 Introduction

1.1 Backpropagation Struggles with Temporal Data

While much progress has been made towards optimizing feed-forward networks with backpropagation [16], the workhorse for deep learning weight updates since its invention in the 1980s, research has struggled to expand this synchronous, time-less network structure to networks that are capable of learning temporal sequences. Effectiveness aside, while some in the community argue that backpropagation *can* be biologically plausible [8], the available evidence seems to indicate that the brain, while potentially performing an operation that reduces error according to some metric, cannot in fact backpropagate errors due to the synaptic structure and strict rules of symmetrical weight updates [10]. Because feed-forward networks contain no memory, one way to use feed-forward networks to encode temporal data is to remove the temporal component entirely, and flatten the entire "series" into one long vector. Alternatively, the current mode of training gradient-based recurrent neural networks (RNNs) is to directly apply the backpropagation techniques optimized for feed-forward non-temporal networks, to cyclical networks, in the form of RNNs or Long Short Term Memory Networks (LSTMs)

*

Corresponding author. Code available at: https://github.com/Symbiomancer

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

[6]. These networks can be difficult to train, may require large datasets, and suffer from issues of vanishing gradients and a move away from biological plausibility, although LSTMs aim to address the vanishing gradient concerns. More recent variants of LSTMs do away entirely with components in an attempt to simplify the architecture in the Gated Recurrent Unit (GRU) [3]. Other recent branches of backpropagation-led time series learning - Transformer architectures - have eschewed entirely with the recurrent network structure [18] and go back to learning "time" as a concatenated vector. Even with these supercharged LSTM, GRU, and Transformer architectures, time series learning remains an open challenge. Here, we encode time directly in the dynamics of simplified neurons with random connectivity, directly attempting to encode the dynamical system of the real world in the dynamical system of the network - with the hopes of gleaning insights into how the brain does this.

1.2 Reservoir Computing

Reservoir computing [11] is a paradigm for recurrent neural network learning that tries to move away and relax the rigid per-time-step gradient updates of LSTMs. In the reservoir paradigm, there are three components: an input weight matrix (generally untrained) that takes in the input and expands it into a higher dimensional "reservoir". This second component, the "reservoir", contains random feedback connections, with the maximum eigenvalue set to be less than 1, that bounces around the combined input and recurrent activity for a set time period (while the activity is required to continuously "die out", in order to wash out old inputs [11]). The reservoir keeps a history of the expanded temporal input as the different time steps are fed in. Critically, the recurrent weights of the reservoir *remain* random and untrained. Third, a "readout" mechanism looks at the patterns of activity of the reservoir and performs some learning, which can be a regression or classification task. For this paper, we show the results of a classification task, although chaotic time series are of high interest to the reservoir community [15].

1.3 Current Deep Reservoir Computing Frameworks Do Not Allow for Error-Based Hidden Temporal Representation Learning

It is well understood that the deep layers of the brain allow for increasingly more complex feature extraction, across multiple domain input types [1]. In the same way that deep ANNs leveraged deeper networks for more complex feature extraction, Deep ESNs [5] attempt to leverage the same hierarchy to extract temporal features from input time series. However, the current state of deep reservoirs does not learn hidden representations in the same manner that deep learning of ANNs does, for the simple reason that one cannot backpropagate gradients through the reservoir-layers. Unsupervised learning of these connections does exist in the form of PCA [12], although this does not allow for the same kind of powerful prediction-based error learning as in the hidden layers of deep ANNs. Here, we take one step towards a deep reservoir that can learn hidden representations with an error generated by the prediction, via direct feedback alignment. This affords the capability of learning complex *temporal* hidden representations from time series data.

2 Contributions

Our contributions are as follows:

- Introduce a novel Deep Reservoir Computer (Deep ESN), with novel trained hidden weights using Direct Feedback Alignment
- Demonstrate that Direct Feedback Alignment can induce learning through extremely nondifferentiable jumps, here over randomly connected neuron pools

3 Proposed Architecture

Reservoir networks consists of the rate encoding variant known as Echo State Networks (ESN) [7], and a spike (event) based variant, known as Liquid State Machines (LSM) [13], discovered independently. Here we adopt the ESN approach, although transferring to an LSM is in development. Each reservoir updates according to the following formula of a simplified reservoir taken from [17]:

$x_t = (1 - \alpha)x_{t-1} + \alpha f(W_{in}u_t + W_{rec}x_{t-1})$ (1)

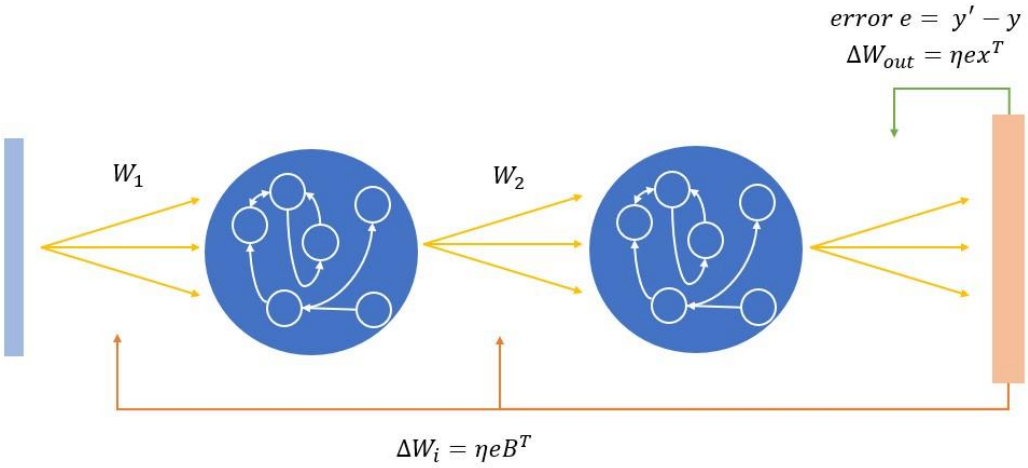


Figure 1: A schematic of the Deep Reservoir Computing with Learned Hidden Representations using Feedback Alignment framework. The hidden layer representations, stored as the W_{in} input matrices to the reservoir-layers, are learned through the direct feedback alignment learning mechanism [14]. The error for the last layer is directly fed back to the previous layers through a random matrix B into the corresponding W_{in} weight matrix for that reservoir-layer, and the network *learns to align* with this random matrix. The weights to the output from the last reservoir layer are computed using the standard delta rule for single-layer ANNs, where x is the activity vector of the last reservoir. The yellow lines indicate learned connection weights. The blue circles are the randomly connected reservoir with untrained recurrent weights for that layer (while shown with identical connectivity here, they are not necessarily identical). B is a random feedback matrix per [10, 14] with appropriate dimensions, η is the learning rate. Shown here are two reservoir-layers, although in our experiments we run with deeper networks effectively.

$y_t = W_{out}x_t,$ (2)

where y_t is the output vector at time t, x_t is the N-dimensional vector of the states of each reservoir neuron at time t, u_t is the input vector at time t, α is the leak rate in the interval (0,1), $f(\cdot)$ is a saturating non-linear activation function (here we use a sigmoid), and W_{in}, W_{rec}, W_{out} are the input-to-reservoir, reservoir-to-reservoir, and reservoir-to-output weight matrices, respectively. If the input vector has length A, and the output vector has length B, and the reservoir vector size is N, then the dimensions of W_{in}, W_{rec}, W_{out} are $N \times A, N \times N$, and $B \times N$, respectively,

3.1 Feedback Alignment and Direct Feedback Alignment

Feedback alignment (FA) has been shown to be a simple yet powerful tool in the quest to move towards more local-update and biologically inspired variants of backpropagation, while at the same time, achieving the same stated end goal of reduction of prediction error for some task. In the original paper [10], the authors point out a few of the major weaknesses of relating backpropagation to biological processes: the fact that these updates must occur in synchronous lock-step, that the synaptic updates must coordinate across multiple neuron layers in a chain to deliver the correct gradient update, and that these updates require weight updates that are symmetric with the output of the neuron. Feedback alignment attempts to tackle the issue of symmetric weight updates by showing one can simply use a random feedback matrix for the gradient updates, completely unrelated to the feed-forward activity, and still wind up with weight changes within 90 degrees of the true gradient update. Direct Feedback Alignment [14] takes this idea one step further, and does away with the locality requirements of FA. There, it was shown that not only can this matrix be random, but it need not come from the layer above: all the updates can be "projected" from the output layer. This addition throws FA and DFA much closer towards biological theories about reward-modulated

STDP with dopamine projections [9, 4], and other neuromodulators, with the error signal e acting as the globally-generated neuromodulator.

3.2 Training the Deep ESN with Direct Feedback Alignment

For the Deep ESN formulation [5], reservoirs are stacked in layers (which we refer to as "reservoir layers"), analogous to how perceptrons are stacked for deep ANNs. Each sub-reservoir is given its own unique weight matrices. The reservoir equations in (1) and (2) remain the same for all hidden reservoir-layers but there are two differences. First the size of x_t: for the input reservoir-layer, x_t is the size of the input vector at time t, whereas for "hidden" reservoirs, x_t is the size of the preceding reservoir-layer. This also is analogous to deep ANNs and perceptrons, as each layer takes in as input the output from the last layer. Second, in the deep formulation, there is only a single W_{out} matrix at the terminal reservoir-layer. The W_{in} of each subsequent non-terminal reservoir-layer serves as the W_{out} for the previous reservoir-layer. At regularly sampled time intervals, the correct label y is compared against the output of the network, y^0. The last layer of the network is trained using the traditional delta rule for perceptron learning (i.e., the last layer of a deep ANN) which reduces error via gradient descent; the W_{in} matrices of each preceding reservoir-layer are trained using the direct feedback alignment algorithm. The hypothesis is that the error from the last output layer will "align" these intermediary reservoir-layer weights to reduce the output error, *even though* the "true" gradient is not propagated backwards through all time steps. We illustrate the architecture in figure 1 for a simple two-reservoir-layer network. We refer to the deep ESN trained with DFA as DFA-DeepESN.

The values of the input weight matrix leading into reservoir i, W_i, are updated at each update step using the direct feedback alignment update:

$$\Delta W_i = \eta e B_i^T \qquad (3)$$

where η is the learning rate, e is the last-layer error, and B^T is the transpose of the random matrix of appropriate dimension (See Fig. 1).

4 Results

We tested our network on several challenging real-world high-dimensional input time-series classification datasets: BasicMotion and ERing, which are taken from the UEA Multivariate Time Series database [2] and are freely and publicly available online. The results are shown below. BasicMotions has 6 input dimensions per time step and 4 outputs corresponding to different movements (walking, running, standing, and playing badminton) taken from a HAR sensor. ERing has 4 input dimensions of a prototype finger sensor and 6 outputs corresponding to the finger. For all series, the reservoir activity was sampled at regular intervals for readout training to the target label for the given series. The weight change updates were performed as one large batch at the end of each epoch for both the last layer and the W_i weights. For BasicMotion, reservoirs of size 800 were used. For the deep networks, 4 reservoir-layers were used for all experiments. We set α to 0.1, weight decay at $10e^{-}9$, learning rate η to 0.01, and learning rate decay at $10e^{-}7$ per epoch.

BasicMotions	train	test	ERing	train	test
Single-Reservoir	82.5	77.5	Single-Reservoir	68.15	73.3
DeepESN w/out DFA	100	97.5	DeepESN w/out DFA	93.3	78.15
DFA-DeepESN	100.0	100.0	DFA-DeepESN	96.7	79.26

Table 1: Results of the accuracy of variants of the network, either without DFA or non-deep reservoirs, on both the BasicMotions and ERing datasets. Results are shown as % correct classification accuracy.

Conclusion and Future Work

Here we propose a novel deep reservoir network with learned hidden weight matrices (DFADeepESN), using the Direct Feedback Alignment method. Our hope is that this work will allow future recurrent networks that have highly non-differentiable components to train effectively towards temporal predictions. Future work will involve analysis of the hidden temporal features themselves, and testing of alternative backpropagation alternatives for the weight training.

Acknowledgments and Disclosure of Funding

The authors would like to thank Vaishnavi Patil for her invaluable assistance in this endeavor. This work was supported in part by NSF award DGE-1632976.

ProtoVAE: Prototypical Networks for Unsupervised Disentanglement

Vaishnavi Patil* Matthew Evanusa* Joseph JaJa
University of Maryland
College Park, MD
{vspatil, mevanusa, josephj}@umd.edu

Abstract

Generative modeling and self-supervised learning have in recent years made great strides towards learning from data in a completely unsupervised *way. There is still however an open area of investigation into guiding a neural network to encode the data into representations that are interpretable or explainable. The problem of unsupervised disentanglement is of particular importance as it proposes to discover the different latent factors of variation or semantic concepts from the data alone, without labeled examples, and encode them into structurally disjoint latent representations. Without additional constraints or inductive biases placed in the network, a generative model may learn the data distribution and encode the factors, but not necessarily in a disentangled way. Here, we introduce a novel deep generative VAE-based model, ProtoVAE, that leverages a deep metric learning Prototypical network trained using self-supervision to impose these constraints. The prototypical network constrains the mapping of the representation space to data space to ensure that controlled changes in the representation space are mapped to changes in the factors of variations in the data space. Our model is completely unsupervised and requires no a priori knowledge of the dataset, including the number of factors. We evaluate our proposed model on the benchmark dSprites, 3DShapes, and MPI3D disentanglement datasets, showing state of the art results against previous methods via qualitative traversals in the latent space, as well as quantitative disentanglement metrics. We further qualitatively demonstrate the effectiveness of our model on the real-world CelebA dataset.*

1. Introduction

One theory of the success of deep learning models for supervised learning revolves around their ability to learn mappings from the input space to a lower dimensional abstract representation space which are best predictive of the corresponding labels [31]. However, for the models to be robust to noise and adversarial examples, be transferable to different domains and distributions and interpretable, we need to impose additional constraints on the learning paradigm. As a promising solution to this, the models can be encouraged to focus on *all* the latent "distinctive properties" of the data distribution and encode them into a representation for downstream supervised tasks. These latent distinctive properties or *factors of variations* are the interpretable abstract concepts that describe the data. The intuitive notion of *disentanglement*, first proposed in [1], proposes to discover all the different factors of variations from the data, and encode each factor in a separate subspace or dimension of the learned latent representation. These disentangled representations are not only interpretable and give valuable insights into the data distribution but are also more robust for multiple downstream tasks [1,28] which might depend only on a subset of factors [29].

The problem of learning these disentangled representations in a completely *unsupervised* way is particularly challenging as we do not have access to the ground truth labels of factors nor are privy to the true number of factors or their nature. Recent works have proposed to solve this problem by training generative networks to effectively model the data distribution and in turn the factors of variations. From this generative perspective of disentanglement, higher dimensional data is assumed to be a non-linear mapping of these factors of variation, where each factor assumes different values to generate specific examples in the data distribution. [23] intuitively characterizes representations which encode the factors as *disentangled* if a change in a single underlying factor of variation in the data produces a change in a single

*These authors contributed equally.

factor of the learned representation (or a change in the subspace of the representation that encodes that factor). Conversely, from the generative perspective, for a representation to be disentangled, a change in a single subspace of the learned representation, when mapped to the data space, must produce a change in a single factor of variation.

For this generative mapping between changes in the representation space to the changes in the factors of variations (in the data space) to be injective, we propose constrains on the changes in the factors of variations for pre-determined changes in the representation space. Each separate subspace of the representation, when changed, must map to a change in a *unique* factor of variation which in turn encourages information about the different factors to be encoded in separate subspaces of the representation. Moreover, each separate subspace must *consistently* map to a change in a single factor throughout the subspace range. This encourages the different subspaces of the representation to encode information only about a single factor of variation. The recent work of [13] also demonstrated empirically that the concept of *local isometry* was a good inductive bias for unsupervised disentanglement, and it can aid generative models in discovering a "natural" decomposition of data into factors of variation. This local isometry constraint on the mapping enforces the changes in the data space to be proportional to any changes made in the representation space. In order to effectively impose the above constraints in an unsupervised manner, we turn towards deep metric learning.

In recent years, metric learning has emerged as a powerful unsupervised learning paradigm for deep neural networks, in conjunction with self-supervised data augmentation. One of the more successful metric learning models, Prototypical Networks, projects the data into a new metric space where examples from the same class cluster around a prototype representation of the class and away from the prototypes of other classes. We use this ability of the network to cluster the different changes in the data space mapped by the corresponding changes in the representation space and thereby enforce the above described constraints.

We develop a novel deep generative model, ProtoVAE, consisting of a Prototypical Network and Variational Autoencoder network (VAE). The VAE acts as the generative component, while the Prototypical Network guides the VAE in separating out the representation space by imposing the constraints for disentanglement.

To learn these representations in an unsupervised way, as the prototypical network needs labeled data for clustering, we train the prototypical network using generated selfsupervised datasets. To produce the self-supervised dataset, we perform *interventions* in the representation space, which change individual elements of the latent space and map the intervened representations to the data space. Owing to the self-supervised training, our model is able to disentangle without any explicit prior knowledge of the data, *including* the number of desired factors.

In this work, our core contributions are:

- We design a self-supervised data generation mechanism using a VAE that creates new samples via a process of intervention to train a metric-learning prototypical network.
- We design and implement a novel model, ProtoVAE, which combines a VAE and prototypical network to perform disentanglement without any prior knowledge of the underlying data.
- We empirically evaluate ProtoVAE on standard benchmark DSprites, 3DShapes, MPI3D, and CelebA datasets, showing state of the art results.

2. ProtoVAE

Our proposed model consists of a VAE [17, 27] as the base generative model (Section 2.1). The VAE consists of an inference network which encodes the data into lower dimensional latent representations and a generator network that maps the representations back into the data space. To implicitly encourage the inference network to encode disentangled representations, we impose constraints on the generative mapping from changes in the representation space to changes in the factors of variations in the data space. This generative mapping is determined by both the generator and the inference networks. To generate self-supervised data for the prototypical network, we perform interventions (Sec 2.2) which changes individual dimensions of the representation. Given a batch of latent representations encoded by the inference network, we first intervene on a dimension of the representation by changing its value to the value of another representation from the batch for the same dimension. The original representations and the intervened representations

are then mapped into the data space by the generator network and concatenated to form a pair of original data and generated data from interventions. Given that the original and the intervened representations differ in a single dimension, the generative mapping should be constrained to ensure that the corresponding pair of original and generated data differs only in a single factor of variation.

This constraint is enforced using a Prototypical network (Appendix A.2) which based on the idea that there exist an embedding in which examples from the same class cluster around a prototype representation for that class. Our proposed prototypical network (Section 2.3) takes as input pairs of data generated by the self-supervised process described above, and maps these pairs of data into a metric space in which pairs generated by intervening on the same dimension cluster together. These clusters which are identifiable with intervening dimensions in-turn become identifiable with the factors of variation that differ in value between the pair when a dimension is intervened upon. We further augment the prototypical network with a separate output head that enforces local isometry, by predicting the difference in the value of the intervened dimensions from the pair in the data space. Fig 5 gives the diagram overview of the complete model.

Lastly, for the intervened representations to be mapped into the data space such that only a *factor of variation* is changed, we constrain the generated data to lie in the true data distribution. This constraint can be effectively enforced in the representation space by minimizing the distance between the distribution of the original representations of the inference network and the intervened representations such that the generator network maps both the distributions to the true data distribution. We do so by training a discriminator network (Section 2.4) in the representation space to distinguish between the original and the intervened representations. The inference network which generates the original representations is then trained to fool the discriminator thus effectively bridging the distance between the distributions.

2.1. Variational Autoencoder

The base generative model consists of an inference network $q_\phi : \mathrm{R}^D \rightarrow \mathrm{R}^d$ that encodes the data x to a lower dimensional representation z and a generator network $p_\theta : \mathrm{R}^d \rightarrow \mathrm{R}^D$ which reconstructs the data $\hat{x}$ from the representations z. The inference and the generator network are trained together to maximize the evidence lower bound (ELBO) of the data log-likelihood as in eq. 1. $\max_{\theta,\phi} L_V(\theta,\phi) = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)||p(z))$

(1)

Maximizing the first term of eq. 1 ensures that the latent representation encodes all the information needed to faithfully reconstruct the data from the representation alone. This ensures that the representations encode all the different factors of variations in the data. The KL divergence term creates an information bottleneck which enforces optimal, compact encoding of the data by enforcing the posterior distribution to be similar to the independent, non-informative prior distribution. For more details please refer to Appendix A.1.

2.2. Self-Supervised Data Generation

The prototypical network works to cluster changes in the factors of variations in the data space and provides gradients to the generator and inference network to better separate out the factors in the representation space. To do so, the prototypical network requires a set of supervised examples, called the support set, from each class to compute a prototype around which examples from the same class cluster. Furthermore, a supervised query set is required to compute the distance of query examples from the target prototypes and the subsequent loss is used to update the network. For learning disentangled representations in an *unsupervised* way, we propose to generate these support and query sets using self-supervision. We describe the full algorithm in Appendix B.1.

Given a batch of data x, we first use the inference network to encode the data into the representation space $z \in$

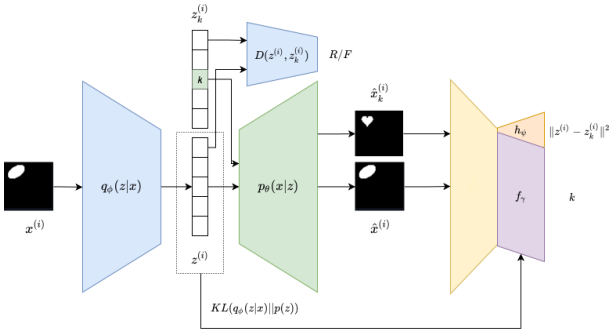


Figure 1. Architecture of our model consisting of a VAE, a discriminator and a Prototypical network. The representation z from the inference network of the VAE (Sec 2.1), is changed at a particular dimension k to get the intervened representation $\hat{z}_k$. A discriminator (Sec 2.4) is trained to distinguish between z and $\hat{z}_k$ and the inference network is updated to fool the discriminator. z and $\hat{z}_k$ are passed to the generator network to map it to the reconstructed data $\hat{x}$ and the intervened data $\hat{x}_k$. The original and the intervened data are concatenated to form the pair $(\hat{x}, \hat{x}_k)$, which is then passed to the prototypical network. The prototypical network (Sec 2.3) maps the pair closer to other pairs with the same dimension intervened. The prototypical network is updated by its ability to correctly predict the intervened dimension of the query examples and the magnitude of the change $\|z - \hat{z}_k\|$.

R^d. Following [29], we define *interventions* as the act of changing the value of a single dimension of the representation $k \in_R [d]$ while keeping the values of the other dimensions the same. We change the value of the intervened dimension to another example's value for the same dimension. The result of intervening on representation z in dimension k produces the intervened representation $\hat{z}_k$. The representation and the intervened representation are then mapped by the generator network to $\hat{x}$ and $\hat{x}_k$ respectively. This pair of $(\hat{x}, \hat{x}_k)$ forms the input data for the prototypical network with the intervened dimension k being the label and the difference in the representation space $|z - \hat{z}_k|$ as the label for the isometry head. The representation z is intervened upon every dimension to generate d support sets and is also intervened upon one dimension chosen uniformly from $[d]$ to generate the query set. The self-supervised data generation algorithm takes in a batch of data x and outputs d support sets $S = \{S_1, \cdots, S_d\}$, one query set Q, labels for the query set L and labels for training the isometric head I.

2.3. Prototypical Network

Our proposed prototypical network maps a pair of data generated by intervening on a single dimension of the representation to a lower dimensional metric space. By mapping *a pair of data* to the metric space, the prototypical network can focus on the factor differing in value between the pair while being invariant to the values of the other non-differing factors. Critically, the factor differing in value remains the same across pairs of different examples when the same dimension is intervened upon and hence should be mapped closer in the metric space. Thus, comparing a pair of data allows the prototypical network to focus on the *change* or *difference* that was brought about by the intervened dimension, and makes the central focus of the losses this change.

The prototypical network first takes in elements of the generated support set S (described in Section 2.2) and computes an m-dimensional representation through the embedding function $f_\gamma : \mathrm{R}^D \times \mathrm{R}^D \rightarrow \mathrm{R}^m$. In this m-dimensional space, the prototypical network computes a prototype embedding c_k for each element in the support set $S_k \in S$ using eq. 2:

$$c_k = \frac{1}{|S_k|} \sum_{s_k^{(i)} \in S_k} f_\gamma(s_k^{(i)}) \tag{2}$$

While the support set is used to compute the prototypes, the query set is used to compute the loss by calculating the distance of its embeddings in the metric space to the target prototypes. For each dimension of the representation to encode information about a *unique* factor of variation, each dimension when intervened upon and mapped to the data space must change a different factor of variation. Thus embeddings of pairs of data generated with the same intervening dimension of the representation must cluster closer in the metric space and away from the clusters of other dimensions. To enforce this, we introduce the *uniqueness* loss which is computed for each query $q^{(i)}$ example by calculating the negative log-likelihood of the true class l as in eq. 3:

$$\min_{\gamma,\phi,\theta} \mathcal{L}_U(\gamma, \phi, \theta) = -\frac{1}{|Q|} \sum_{q^{(i)} \in Q} \log p_\gamma(t = l | q^{(i)}) \cdot \mathrm{KL}(q_\phi(z_l|x) || p(z)) \tag{3}$$

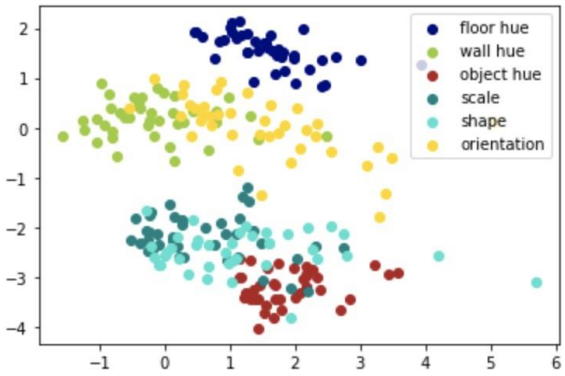
where the probability of each class $p_\gamma(t = l|q^{(i)})$ is calculated as a distribution over the Euclidean distance d to the prototypes as in eq. 4.

$$p_\gamma(t = l | q^{(i)}) = \frac{\exp\left(-d(f_\gamma(q^{(i)}), c_l)\right)}{\sum_{k'} \exp\left(-d(f_\gamma(q^{(i)}), c_{k'})\right)} \tag{4}$$

The loss for every intervening dimension of the query examples is multiplied by the KL-divergence of that dimension, averaged for the batch of examples. This ensures that the loss for the intervening dimensions is scaled by amount of information encoded by that dimension. For the dimensions that do not encode any information, and hence do not change any factor of variation upon intervention, the corresponding loss is scaled by zero. This is important as we do not need any

prior assumptions on the dimension of the representation needed to encode all the factors and the VAE can find the right number of dimensions needed.

In addition to the uniqueness loss, we want each dimension to consistently encode only a single factor of variation. When the representation z is first intervened



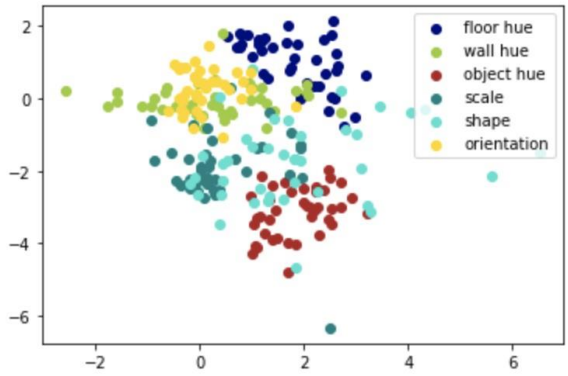


Figure 2. Output of the prototypical network with embedding dim $m = 2$ when the input is real pairs of data from the 3DShapes dataset which differ in a single factor of variation. Each color corresponds to a unique factor which differs in value amongst the pair. The network clusters the changes correctly on the pairs from the original dataset. This suggests that the prototypical network is clustering pairs of images based on the changed factor of variation. *Left*: $\lambda = 10$. *Right*: $\lambda = 5$.

on dimension k and mapped to the data space it makes a certain change in factor. When the representation is intervened again at dimension k by a different amount and mapped to the data space it should produce a change in the same factor, irrespective of the amount it was changed. When passed in to the prototypical network, the pair of data generated by the original z and intervened $\hat{z_k}$ must be embedded in the prototypical metric space closer to the pair generated by the representation intervened in the same dimension by a different amount. To enforce this we introduce the *consistency* loss in eq. 5 where the prototypes are replaced by the embeddings of the same example in the support set.

$$\min_{\gamma,\phi,\theta} \mathcal{L}_C(\gamma,\phi,\theta) = -\frac{1}{|Q|}\sum_{q^{(i)}\in Q} \log r_\gamma(t=l|q^{(i)}) \cdot KL(q_\phi(z_l|x)||p(z)) \quad (5)$$

With r_γ calculated as follows:

$$r_\gamma(t=l|q^{(i)}) = \frac{\exp\left(-d(f_\gamma(q^{(i)}), f_\gamma(s_l^{(i)}))\right)}{\sum_{k'}\exp\left(-d(f_\gamma(q^{(i)}), f_\gamma(s_{k'}^{(i)}))\right)} \quad (6)$$

The consistency loss and the uniqueness loss are added together to get a combined prototypical loss eq. 7

$$L_P(\gamma,\phi,\theta) = L_C + L_U \quad (7)$$

As an additional inductive bias, as proposed in [13], we constrain the generative mapping between original and intervened representation $(z,\hat{z_k})$ and the generated pair $(\hat{x},\hat{x_k})$ to be locally isometric [7]. Thus the factor changed in $\hat{x_k}$ when compared with $\hat{x}$ must differ in value proportional to the corresponding change in dimension k of z and the intervened $\hat{z_k}$. This serves as an imperative inductive bias for unsupervised disentanglement.

The additional head $h_\psi : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^d$, when given a pair of data, is trained to predict the difference in the values for the all the dimensions of z and $\hat{z_k}$ through the loss function in eq. 8. $\min_{\psi,\theta,\phi} L_I(\psi,\theta,\phi) = \|h_\psi((\hat{x},\hat{x_k})) - |z - \hat{z_k}|\|^2$ (8)

The training data for the isometry head generated in a self-supervised manner as described in section 2.2 where the support set S consists of data pairs and the set I consists of the corresponding targets. In the final implementation, f_γ and h_ψ share all hidden convolutional layers and differ only in the final fully connected layer.

2.4. Representation Space Discriminator

To restrict the realm of "changes" in the data space, made by intervened representations, to only the factors of variation present in the dataset we regularize the intervened representations to map to data in the true data distribution. By minimizing the reconstruction loss in eq. 1, the decoder learns to map the latent representations learned by the inference network $z \sim q_\phi(z)$ to the true data distribution $q(x)$. To enforce the decoder to also map the intervened representations $\hat{z}$

$\sim q(\hat{z})$ to the true data distribution we propose to minimize the distance between distributions of the true representations $q_\phi(z)$ and the representations after intervention $q(\hat{z})$ by minimizing the KL divergence between the distributions $\mathrm{KL}(q_\phi(z)||q(\hat{z}))$. To this effect we introduce a discriminator as proposed in [16] in the representation space which is trained to distinguishes samples from the two distributions by minimizing the loss in eq. 9). $\min_w \mathrm{L}_D(w) = -[\mathrm{E}_{\hat{z}}[\log(D_w(\hat{z}))] + \mathrm{E}_z[\log(1 - D_w(z))]]$ (9)

The inference network, which generates the representations $z \sim q_\phi(z)$, is regularized to fool this discriminator by minimizing the loss in eq. 10. This encourages the inference network to encode data into representations whose individual dimensions can be intervened on and mapped to the same data distribution.

$$\min_\phi \mathrm{L}_E(\phi) = \mathrm{E}_z[\log(1 - D_w(z))] \quad (10)$$

The final objective (eq. 11) of our method is a weighted sum of the different losses, and is optimized by the network parameters of the VAE and the prototypical network corresponding to each loss.

$$\min_{\phi,\theta,\gamma,\psi} \mathrm{L} = -\mathrm{L}_V(\phi,\theta) + \alpha \mathrm{L}_E(\phi) + \lambda \mathrm{L}_P(\gamma,\phi,\theta) + \kappa \mathrm{L}_I(\psi,\phi,\theta) \quad (11)$$

3. Empirical Evaluation

To empirically evaluate our method, we perform both quantitative and qualitative evaluation on two synthetic datasets and one real dataset with known factors of variation and qualitative evaluation on CelebA dataset [22]. The two synthetic and one real datasets are generated from independent ground truth factors of variation; DSprites [24] binary 64 x 64 images with 5 factors of variation: 3 shapes, 6 scales, 40 orientations, 32 x-positions and 32 y-positions; 3D Shapes [2] 64 x 64 x 3 color images with 6 factors of variations: 4 shapes, 8 scales, 15 orientations, 10 floor colors, 10 wall colors and 10 object colors; MPI3D real [10] 64 x 64 x 3 color images with 7 factors of variations: 6 colors, 6 shapes, 2 sizes, 3 camera heights, 3 background colors, 40 horizontal axis, 40 vertical axis, and one real world dataset; CelebA. The details of the architecture for the different components and the corresponding hyperparameters are listed in the supplementary material (Appendix B.2) and (Appendix B.3) respectively.

We qualitatively evaluate our model by intervening on the different dimensions of the learned representations and traversing the range of values of the dimension linearly in a fixed range $[-2,2]$. A model is better disentangled if the changes made in the data space while traversing a dimension are similar to the changes in a factor of variation in the data space.

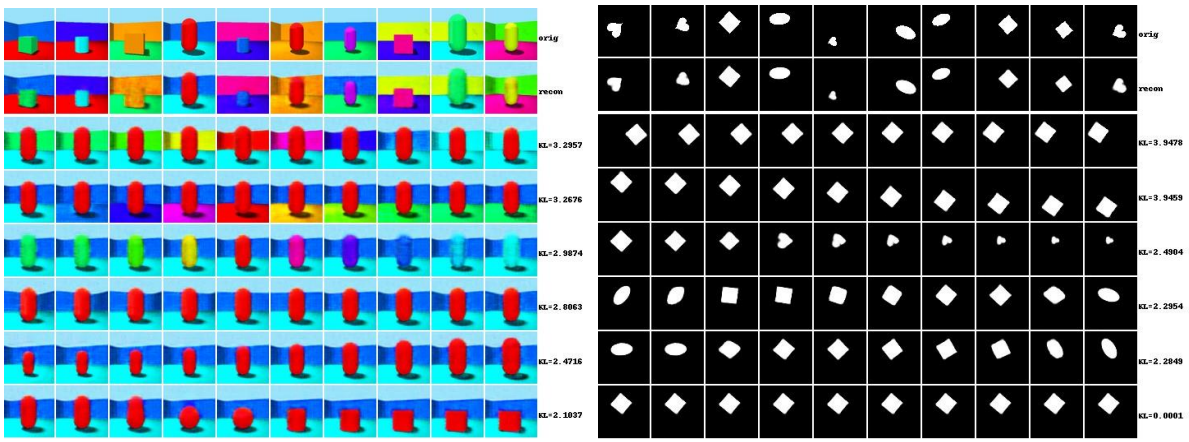


Figure 3. A comparison of latent traversals in latent space for the 3DShapes and Dsprites dataset. *Left:* 3DShapes, *Right:* Dsprites. ProtoVAE produces smooth, disentangled latent representations. Row 1 and 2 are some sample original images, and their reconstructions generated by our model, respectively. Rows 3 downward are the traversals for each latent element, as detailed below. For 3DShapes, we actually see a near-perfect traversal across all of the known factors of variation.

DATASETS	DSPRITES			3DSHAPES		
MODEL	FVAE	DCI	MIG	FVAE	DCI	MIG
β-VAE	0.51 ± .10	0.23 ± .10	0.15 ± .10	0.81 ± .10	0.44 ± .17	0.28 ± .18
ANNVAE	0.70 ± .10	0.28 ± .10	0.23 ± .10	0.84 ± .09	0.46 ± .16	0.31 ± .15
β-TCVAE	0.68 ± .10	0.35 ± .06	0.17 ± .09	0.88 ± .07	0.63 ± .10	0.40 ± .18
FVAE	0.74 ± .06	0.38 ± .10	0.28 ± .09	0.81 ± .06	0.47 ± .12	0.33 ± .14
GR-FVAE	0.75 ± .08	0.41 ± .07	0.31 ± .06	0.79 ± .06	0.49 ± .06	0.43 ± .11
PROTOVAE	0.70 ± .06	0.51 ± .04	0.37 ± .09	0.90 ± .06	0.84 ± .07	0.71 ± .11

Table 1. Various disentanglement metrics evaluated across a number of state of the art methods for the DSprites and 3Dshapes dataset. For all metrics, higher is better. The results for the other models are obtained using the hyperparameter settings and experimental conditions as described in [23]. The scores for all the models were averaged across ten runs with different random seeds, with standard deviation shown as ±. Gr-FVAE is the GroupifyVAE variant applied to the FactorVAE, as this is the closest variant of the GroupifyVAE to our model and the results for which are taken from [32]. The highest values in a column are written in bold. As we see, the ProtoVAE outperforms the state of the art on a majority of the metrics. ProtoVAE hyperparameters for DSprites and 3Dshapes results shown are $\{\alpha = 10, \lambda = 10, \kappa = 10\}$ and $\{\alpha = 20, \lambda = 20, \kappa = 20\}$

Our model both finds the correct number of factors and encodes them separately without any specific hyperparameter tuning. From Figure 3 we can see that our method ProtoVAE, produces disentangled traversals covering both the number of factors as well as the entirety of the range of the values for the Dsprites and the 3DShapes dataset. The latent traversals on the MPI3D dataset can be found in figure 4. Our method effectively separates the factors thus disentangling the learned latent representation without compromising on the reconstruction quality as seen from row 2. Owing to the unsupervised nature our method struggles to exactly disentangle the non-isometric discrete factor of shape in the DSprites dataset. For the 3DShapes dataset, in our traversals in figure 3, we achieved near perfect disentanglement, completely unsupervised. In the Appendix D, we show the performance of the ProtoVAE for a subset of the Dsprites dataset with only a few factors. We show traversals from models FactorVAE and β–VAE, along with our model, with only a few isometric factors for comparison. Our proposed ProtoVAE is the only model that does not conflate two factors and encodes them in separate dimensions of the representation.

Furthermore, we quantitatively evaluate the learned representation by calculating state-of-the-art disentanglement metrics. We choose metrics from each of the three kinds of metrics described in [33];

Intervention based FactorVAE [16], Predictor-based Disentanglement Completeness-Informativeness (DCI) [8] and Information-

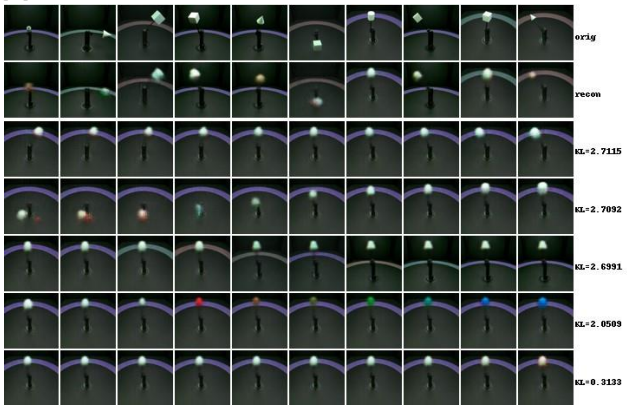


Figure 4. Latent traversals on the MPI3D real world disentanglement dataset. The data is collected via a camera that observes a jointed arm with known changed ground truth factors of variation. From top to bottom: original data, reconstruction, arm angle left/right, arm angle top/bottom, background height, arm end color, size. The KL values represent the amount of information encoded by that dimension of the representation.

Model	FVAE	DCI	MIG
β-VAE	.41 ± .05	.23 ± .04	.06 ± .03
AnnVAE	.29 ± .04	.12 ± .02	.07 ± .07
β-TCVAE	.45 ± .06	.27 ± .03	.16 ± .03
FVAE	.40 ± .04	.30 ± .03	.23 ± .03
DisCo	.39 ± .07	.29 ± .02	.07 ± .03
ProtoVAE	.46 ± .04	.38 ± .05	.25 ± .11

Table 2. Quantitative comparative metrics on the MPI3D dataset. ProtoVAE performs comparatively or better consistently across multiple metrics on a difficult real disentanglement dataset. See Fig. 4 for latent traversals on MPI3D. ProtoVAE hyperparameters for results shown are $\{\alpha = 10, \lambda = 2, \kappa = 2\}$. The numbers for DisCo have been borrowed from their paper [26] for the VAEbased methods.

based Mutual Information Gap (MIG) [5]. The metrics were implemented as proposed in [23] with the same hyperparameters. We refer the interested readers to [33] for an intuitive understanding of the metrics. We highlight here that our model achieves a higher DCI metric and a higher corresponding *completeness* and *informativeness* metric, which reflects the mode covering capabilities of the learned representation.

From table 1 we see that on the DSprites dataset, our method outperforms the state of the art models in a majority of the metrics. Similar performance of our model on the 3DShapes dataset as seen in table. Also, the variance in the metrics for the different runs is significantly lower than of the previous methods, thus ensuring a more robust way to disentangle representations. For the real disentanglement dataset of MPI3D [10] consisting of a camera taking photos of an object attached to a jointed arm, we see that our model consistently either matches or outperforms the state of the art. From the baselines, especially important is the FactorVAE model, which is the base model upon which we add our contributions for the ProtoVAE model and hence use it as a comparison to demonstrate the effectiveness of our contributions.

On the CelebA dataset, we find that across multiple runs, our model is able to find the same "natural" decompositions that correspond to human-interpretable factors of variation consistently (Fig. 5). We notice that the model is not constrained to completely encode one factor per latent dimension and the model might encode different ranges of a factor in different latents; we see this occur for example when it encodes half of the azimuth in one latent, and half in another. However, as we can see, for the most part, each latent dimension contains information only about one factor of variation and even in the unsupervised regime our model still encodes natural decompositions.

We visualize the embedding space of a trained prototypical network using our method in Fig. 2. We see that when input to the prototypical network is pairs of images from the dataset, with one ground truth factor differing in value between the pair, the prototypical network effectively clusters the pairs based on the differing factor. This clustering aligns with the labels based on the intervened dimensions during training and thus points to the effectiveness of the prototypical network for encouraging disentanglement.

We also performed quantitative and qualitative ablation studies on the 3DShapes dataset by changing the values of α, λ and κ to understand the effectiveness of each of the components and losses we introduce. The results of these ablations can be found in the supplementary material (Appendix C). Furthermore, we also perform ablation studies on the effect of dimension m of the metric space of the prototypical network on the metric scores. We also show in the Appendix (Section C) some limitation cases where the representations of the model did not axis align with a few factors but was rotated with respect to those

factors. We see that smaller values of the prototypical network metric space m performs better by encoding data in tighter clusters which in turn it imposes stronger constraints on the VAE. The discriminator and the corresponding L_E helps in confining the encoding of the factors into a single dimension whereas L_P alone fails to do this effectively as seen in ??.

4. Related Works

Many state-of-the-art unsupervised disentanglement methods extend the VAE objective function to impose additional constraints on the structure of the latent space to match the assumed independent factor distribution. β-VAE [12] and AnnealedVAE [3] heavily penalize the KL diver-

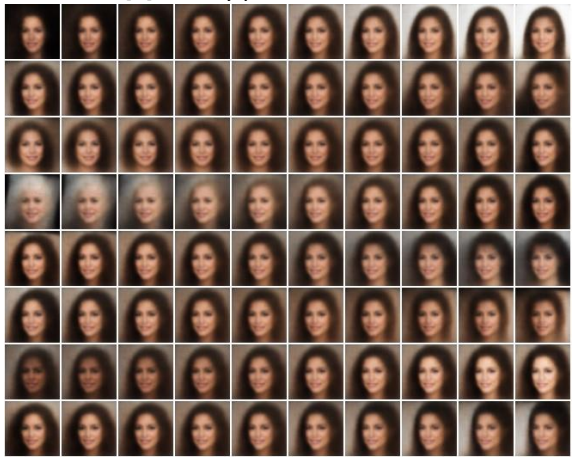


Figure 5. Latent traversals on the CelebA dataset ProtoVAE successfully captures ground-truth factors of variation on real-world data. From top to bottom: background color, hairstyle, head angle, age, hairstyle, hair color, skin color, face profile.

gence term thus forcing the learned posterior distribution $q_\phi(z|x)$ to be independent like the prior. Factor-VAE [16] and β-TCVAE [5] penalize the total correlation of the aggregated posterior $q_\phi(z)$. $TC = KL(q(z)||\prod_{i=1}^{K} q(z_i))$ where the aggregated posterior is calculated as $q_\phi(z) = \mathbb{E}_{p(x)}[q(z|x_i)] = \frac{1}{N}\sum_{i=1}^{N} q_\phi(z|x_i)$ using adversarial and statistical techniques respectively. DIP-VAE [18] forces the covariance matrix of the aggregated posterior $q(z)$ to be close to the identity matrix by method of moment matching. The changes that we described in the latent space are defined as intervention by [29] to study the robustness of the learned representations under the Independent Mechanisms (IM) [28] assumption. Most closely related to our work are VAE models that learn to disentangle by altering the latent code. In [15], the authors use a VAE or AE with a split double latent code, with a cycle consistent loss, but required that attribute labels be known *a priori*, which was also a requirement in [9] which learned by swapping out chunks of the latent code, and [30], which used the labels as a constraint to find unique disentanglement. The authors in [4] also used cycle-consistent loss, but again required labeling. In [14] the authors attempted unsupervised disentanglement with regular (non-variational) Autoencoder network models, by stacking one after another, our model instead uses a prototypical neural network. In [19] the authors derive a novel Jacobian loss combined with a student-teacher iterative training algorithm with an Autoencoder network model. In [25] the authors develop a latent-manipulating model aimed at human-interactive image manipulation tasks.

In [32] the authors use the group based definition by [11] and a cycle consistency loss to define the elements of a group. Our work differs significantly as we do not reencode the reconstructed data nor the generated data from interventions and instead use a prototypical network. [35] encode the latent space of a VAE using the commutative Lie group and enforce constraints on the latent space. A recent work [26] propose to learn disentangled representations from pre-trained models using contrastive methods.

The most prominent work from the GAN family is InfoGAN [6] which learns disentangled, semantically meaningful representations by maximizing a lower bound on the intractable mutual information between the conditioning latent variables c and the generated

Metric	$\alpha = 0$	$\alpha = 10$	$\alpha = 20$
FVAE	.93 ± .04	.85 ± .03	.88 ± .05
DCI	.78 ± .05	.81 ± .06	.81 ± .07
MIG	.59 ± .07	.63 ± .04	.65 ± .08
β-VAE	.92 ± .06	.88 ± .04	.90 ± .05

Table 3. Ablation results for different values of α, which shows that the discriminator helps in confining the encoding of the factors into a single dimension. This can be seen by a higher value of the β VAE and the FactorVAE metrics for $\alpha = 0$ but not for the MIG and the DCI metrics which require factors to be encoded in a single dimension.

samples $G(z,c)$. InfoGAN-CR [20] and [34] add a contrastive regularizer to the InfoGAN model to further encourage disentanglement. [21] add orthogonal regularization to encourage independent representations. However, all the GAN methods suffer from the limitation that they require *a priori* the number of factors to be discovered, in addition to the number of values for all the discrete factors. For fairness of comparison, we thus only compare against methods that do not require these priors.

5. Conclusion and Future Work

In this work, we proposed a novel generative model consisting of a VAE and a Prototypical Network for learning disentangled representations in a completely unsupervised way, inspired by recent discovery of sufficient inductive biases. We impose constraints on the structure of the representations learned by training the model in a selfsupervised manner to encode information about the different factors in separate dimensions of the representation. Our proposed method is able to outperform other state of the art networks on a number of metrics on three prominent disentanglement datasets. For future work, our method can be easily adapted to be trained in a weakly supervised regime with pairs of data differing in known number of factors being the prototypes for the prototypical network. The results can be possibly improved by intervening on multiple dimensions of the representations simultaneously. The importance of methods that can disentangle data *without labels* is critical as data is plentiful and the resulting representations give interpretable insights into the variations in the data distribution, and can be used for downstream tasks. Our hope is this work adds evidence that self-supervised generative methods are important in this endeavor.

Deep-Readout Random Recurrent Neural Networks for Real-World Temporal Data

Matthew Evanusa[1*], Snehesh Shrestha[1], Vaishnavi Patil[1], Cornelia Fermu¨ller[1,3], Michelle Girvan[2] and Yiannis Aloimonos[1]

[1]Department of Computer Science, University of Maryland, College Park, MD, USA.
[2]Department of Physics, University of Maryland, College Park, MD, USA.
[3]Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA.

*Corresponding author(s). E-mail(s): mevanusa@umd.edu;

Abstract

Echo State Networks (ESN) are a class of recurrent neural networks that can learn to regress on or classify sequential data by keeping the recurrent component random and training only on a set of readout weights, which is of interest to the current edge computing and neuromorphic community. However, they have struggled to perform well with regression and classification tasks and therefore, could not compete in performance with traditional RNNs, such as LSTM and GRU networks. To address this limitation, we have developed a novel hybrid network, called Parallelized Deep Readout Echo State Network (PDR-ESN) that combines the deep learning readout with a fast random recurrent component, with multiple ESNs computing in parallel. We show the PDR-ESN architecture allows for different configurations of the sub-reservoirs, leading to different variants which we explore. Our findings suggest that different variants of the PDR-ESN offer various advantages in different task domains, with some performing better in regression and others in classification. In all cases, our PDR-ESN architecture outperforms the corresponding gradient-based LSTM and GRU architectures in terms of training time as well as accuracy. To further evaluate, we also compared against a Transformer encoder classifier, where the PDR-ESN outperformed on all tasks. We conclude that our proposed network demonstrates a good

trade-off between the fast training times of traditional ESNs with the accuracy of deep backpropagation for real-world tasks. We hope that this architecture offers an alternative approach to sequential processing for edge computing as well as more biologically-realistic network development.

Keywords: Recurrent Neural Network, Time Series Analysis, Reservoir Computing, Deep Learning, Hybrid Neural Networks

1 Introduction

Learning temporal data is still a challenging task due to the explosive increase in the size of the data space in the temporal domain. In order to process temporal information with neural networks, the network needs to keep a memory of its past activity, either in the activity of the network or by looking at the entire sequence. The former by feeding back the activity from the previous state into the neuron at the next time step, as in the case of Recurrent Neural Network architectures. The latter can be accomplished by either feeding the entire sequence into a feed-forward network, as in the style of transformer-attention architectures [1].

The main question of concern is: *What is the optimal way to encode and learn temporal data using a neural-inspired network?* The dominant paradigm of machine learning for A.I. is the feed-forward approach: networks that do not contain cycles, even for temporal or sequential data [1], [2]. These networks are *functions*: they map any given input point into a new manifold after layers of nonlinear operations. These networks are easy to train with backpropagation, as the network need not be unrolled during training. While this feed-forward approach has shown impressive performance [3] there is an increasing awareness that these mega-architectures may not be sustainable, and are not applicable on edge, embedded, or mobile devices. In addition, these large networks incur long training times, which becomes an issue if we want to deploy time-sequence learning A.I. on robotic or active agents, where the agent needs to be able to learn in real-time or online, so-called "open-world learning" problems, or online learning tasks. Further, while transformer and feed-forward attention models work on language, it is unclear how well they will scale up to long sequences, as the number of connections grows quadratically with the sequence length. The brain circumvents these issues by introducing recurrent connections, maintaining a memory of past activity for future predictions. In the biological brain, there are countless recurrent and feedback connections, including highlevel processing areas such as the cortex [4]. Because of the recurrent connections, these networks are no longer functions but are actually *dynamic systems* [5], like the brain. Computationally, neural networks with recurrent connections are known as Recurrent Neural Networks (RNNs).

RNN architectures compact deeper networks into smaller ones [6], and in theory can contain more expressive power than deep networks [7]. From the

1.1 Main Contributions

biological standpoint, recurrent networks are ubiquitous in the biological brain across species. Thus, RNN research tackles two open fields, as insights into RNNs will also yield insights into the neuroscience behind dynamical cognition.

However, because of these recurrent connections' incompatibility with standard backpropagation, the main open challenge in RNN research is how to best train them, as the backward passes of backpropagation do not inherently fit with a recurrent network, which contains (potentially infinite) backward passes. This unrolling of the network, while being biologically implausible, also is the driving force behind much of the woes of gradient learning for RNNs. An alternative approach to RNN training that rests better with biology, known as *Reservoir Computing*, does not attempt to train the recurrent components but rather keeps them random and fixed, turning them into a sort of temporal kernel, expanding the data into a higher-dimensional space, where readout
weights are learned on this higher space. We will interchangeably refer to reservoir computing as *Random Recurrent Neural Networks* to highlight the large similarities between reservoir computing and machine learning RNNs. Here, we showcase the performance of new hybrid RNN architectures where the recurrent component is based on the *Reservoir* (or Random RNN) paradigm specifically, Echo State Networks [8], described below, and compare our network with current gradient-trained RNN networks - Long Short Term Memory and Gated Recurrent Unit networks (LSTM, GRU). We demonstrate that even without training the recurrent component, we can surpass the gradient-based RNN performance, at a fraction of the time, and offer an alternative direction for RNN research and development. Our work suggests that the job of the recurrent component need not necessarily be to fit its weights to the model, but rather to act as an expansion of the input data.

1.1 Main Contributions

Our work brings several novel contributions to the current state of the art: • Use of a parallel reservoir mechanism in conjunction with a backpropagated deep feedforward network readout, which we call a Parallelized Deep Readout Echo State Network (PDR-ESN)

- Two variants of PDR-ESN that allow for sub-reservoir cross-talk: a lateral ring weight matrix (PDR-ESN Ring), and a Hub-Reservoir dual network (PDR-ESN Hub)
- Demonstrate the PDR-ESN architecture for the tasks of gesture recognition and EEG regression, and show that it outperforms GRUs and LSTMs in terms of training speed and accuracy

2 Related Work

There have been numerous attempts to enhance random RNNs with more predictive power. Firstly, there have been attempts to replace the standard ridge regression in echo state networks with alternatives [9]. There also have been efforts to apply CNN techniques to the reservoir readout [10]. Recent work

has attempted to create hybrid models that integrate feed-forward networks into the reservoir structure [11].

A separate branch of work has focused on improving the reservoir itself, either by adding intrinsic plasticity (IP) to the recurrent neurons [12], combining IP with small-world network clustering [13], combining linear and nonlinear terms as in [14], or using genetic or evolutionary algorithms [15].

Other earlier works, like ours, have attempted to attach a multi-layered backpropagation learning structure to ESNs [16, 17]. However, neither of these
works used batch normalization, a parallelized sub-reservoir setup (see below), the bio-inspired hub mechanism, nor did they test on challenging datasets such as gesture classification or EEG regression. Recently, it was shown in [18, 19] that the issue of high dimensional inputs can be alleviated by introducing a parallel reservoir schema, where the work is divided between multiple reservoirs working in parallel. Our work extends this architecture for use with non-linear readouts and real-world data. The idea of using reservoirs in parallel can be seen as an extension and fusion of the ideas of deep feed-forward multilayered perceptrons, where each neuron is replaced by a reservoir. ESNs have also very recently been applied to EEG learning tasks [20]. Transformer encoders [21], as well as LSTM networks [22, 23], have also been recently used to learn EEG data.

As concerns the time-series prediction, efforts have been made to build off of the success of FORCE learning [24], via more complex network architectures [25]. Here instead, we focus on more traditional echo state network approaches to training that involves regression on the states of the reservoir, so our work differs in that regard to [25], in addition to the backpropagation mechanism. Recent work [26] used a similar ring structure with random neural connections to model the visual and prefrontal cortex. In [27–29] the authors demonstrate and provide evidence for our hypothesis that non-linear readouts of the reservoir can improve performance.

3 Recurrent Neural Networks

3.1 Types of Recurrent Networks

We will start with a brief background in RNN architectures necessary to explain our new architecture.

3.1.1 Fully Differentiable RNNs

Allowing the network to learn not just static features, but also temporal information increases the learning capacity of the network [30, 31]. Within the recurrent neural network (RNN) umbrella, we have two paradigms: a) fully differentiable recurrent networks, such as LSTMs [32], and b) Reservoir Computing (RC). We will not discuss here another paradigm, Spiking Neural Networks (SNNs) [33], which are by original design not a purely recurrent architecture, but can be made one (see below for LSMs), which muddies the classification. In fully differentiable RNNs, the training of recurrent connections

via gradient descent is usually achieved by unrolling the backpropagation (BPTT) [34], although newer alternatives to BPTT do exist [35], and local, non-BPTT methods exist for LSTMs [36]. In practice, given the abundance of computing power these days, BPTT is the main training method used with LSTMs and GRUs. However, local training methods for these networks, which were developed decades ago, maybe extremely valuable and potentially biologically realistic; for example, the Generalized Recirculation algorithm [37] and particularly the Almeida-Pineda local learning rule for RNNs [38] could potentially be promising for gradient-based temporal local learning.

In reservoir computing, on the other hand, the recurrent connections are kept random and fixed, and only the readout of the reservoir is trained (see Fig. 1), i.e., the states of the reservoir are mapped to targets using some kind of regression algorithm; this removes the need to backpropagate the error through the recurrent layers. Reservoirs or random RNNs can come in one of two ”flavors”: rate encoding real-valued reservoirs called Echo State Networks (ESN) [8], and spike or event-encoding reservoirs called Liquid State Machines (LSM) [39]. For this work, we will focus exclusively on ESNs.

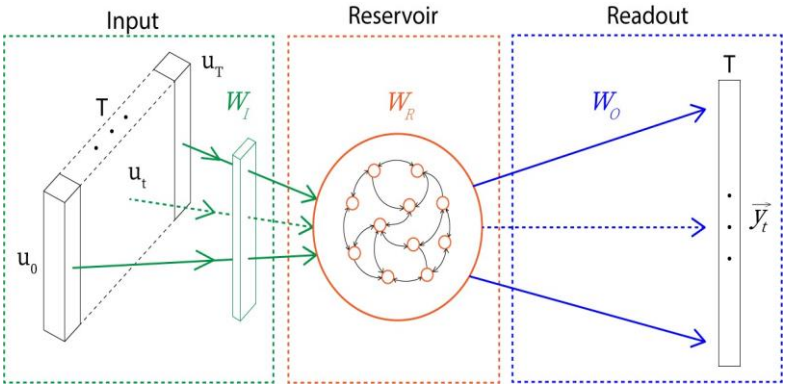


Fig. 1 The standard setup for learning with random recurrent neural networks with a readout mechanism (reservoir computing). There are three major components: the *input*, which is the time series vector at time *t*, denoted in the green box. The *reservoir*, which is the random pool of neurons, is denoted in the orange box. Lastly, the *readout*, which is a regression or least-squares mechanism that learns W_O, is denoted in the blue box. The input is fed into the reservoir via input weights W_I, which combines current input with past input passed in from the previous time step via W_R (see Section. 3.2) combined with some non-linearity, and bounces around the activity. The task essentially boils down to collecting the set of reservoir states at time t (x_t) and regressing against a target output vector y_t. Only W_O is trained during the process, all other weights remain fixed. The reservoir can be seen as acting as a temporal kernel machine [40].

3.2.1 Implementation

The most standard way to implement a random recurrent neural network with readout (reservoir computer) is to attach a set (vector) of recurrent, randomly weighted neurons that remain fixed and unlearned, to a regression readout solver that can be offline such as ridge regression [8], or batched as in stochastic gradient descent (SGD) [41] (we use SGD to train a multi-layer perceptron readout, see below). Here, we use the nonlinear leaky update equation from [42], which incorporates a leak rate that acts as a time constant τ for the reservoir dynamics, if this were written as an ODE. At each time step t, input (u_t) is fed into the reservoir through an input weight matrix W_I, fed back inward through recurrent connection W_R, and read out using W_O on the reservoir activity state vector. The output vectors are updated according to the following equations:

$$x_t = (1-\alpha)x_{t-1} + \alpha f(W_I u_t + W_R x_{t-1}) \quad (1)$$

$$W_O x_t = y_t, \quad (2)$$

where y_t is the output vector at time t (for our experiments, it is the same for all t), x_t is the N-dimensional vector of the states of each reservoir neuron at time t, α is the leak rate, $f(\cdot)$ is a saturating non-linear activation function (here hyperbolic tangent), and W_I, W_R, W_O are the input-to-reservoir, reservoir-to-reservoir, and reservoir-to-output weight matrices, respectively.

Given that we have I time series, each of length T, our classification task is to map each time series i to the corresponding label $y^{(i)}$. We first feed the individual time series into the reservoir and collect the resulting reservoir states $x^{(i)}_{t..T}$ for each time-step t. These resulting reservoir states are then concatenated

into a vector $x^{(i)}$ (Eq. 4) which is then used to map to the given label $y^{(i)}$ for that particular time series i. We stack the collected $x^{(i)}$ into one large matrix, which we denote X, similarly, we stack the labels into a matrix Y.

$$x_t^{(i)} = (1-\alpha)x_{t-1}^{(i)} + \alpha f(W_I u_t^{(i)} + W_R x_{t-1}^{(i)}) \quad (3)$$

$$x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \cdots, x_T^{(i)}] \quad (4)$$

$$W_O x^{(i)} = y^{(i)},$$
$$X = [x^{(1)}; x^{(2)}; \cdots, x^{(I)}]$$
$$Y = [y^{(1)}; y^{(2)}; \cdots, y^{(I)}] \quad (5)$$

$$W_O = (X^T X + \lambda I)^{-1} X^T Y \quad (6)$$

The readout weights, W_O, are then learned by fitting a linear regression curve between the collected matrix of reservoir states X, and Y , the collected set of output labels, using standard least-squares regression with a regularization constant. For standard reservoirs, L2 normalization via ridge regression is used, shown in closed form in Eq. 6, with a chosen normalization parameter λ. For our deep-readout variants, a multilayered perceptron (deep network) is used for the mapping from X to Y (Sec. 4.3). From the updates, one can see that the reservoir keeps a memory trace of its past activity through the recurrent weights W_R, which is a function of the leak rate α as well as the spectral radius of W_R. In order to make sure inputs vanish over time, it is necessary to ensure the *echo state property* of the network, which says that the spectral radius, or largest singular value of the recurrent matrix, is less than 1. Although this property has been re-analyzed [43] and the bounds are input-specific, for simplicity we will use the original recipe for calculating the desired spectral radius as in [8] and keep it less than 1. For simplicity of the model and ease of replication, we also do not hyper-tune the reservoir weights using any optimization techniques such as genetic algorithms [15, 44] or particle swarm optimization [45, 46], although those can readily be applied to this work to decrease the variance of the reservoir trials and improve performance as well.

Reservoir computing has been shown to train faster and use less training data than their fully-differentiable counterparts in some domains [47]. For real world tasks, ESNs have been successfully used for a variety of tasks ranging from direct robotic control [9], electric load forecasting [48], wireless communication [49], image classification [42], robot navigation [50], reinforcement learning [51], and recently action recognition using preprocessed inputs [52].

4 PDR-ESN Architecture

The PDR-ESN extends previous works to add a deep-readout mechanism to a collection of parallel sub-reservoirs, arranged in various ways with different topologies, as we will discuss below. The input is partitioned into segments and sent to individual sub-reservoirs, which process the information as a traditional reservoir, and then the resulting sub-reservoir activity is flattened, and passed to the deep readout mechanism (See Fig. 2). Of special interest is the PDR-ESN Hub architecture, where the Hub acts as an information passing, aggregating, and processing module but does not participate in the readout.

4.1 Parallel Reservoir Structure

4.1.1 Motivations

Biologically, the brain is partitioned into multiple sub-regions, which aids in its functions [53, 54]. On the other hand, there exist networks with massive cross connectionism such as the Hopfield networks [55], which have the issue of what is known as "catastrophic interference" [56], wherein learning new information causes old information to be lost. We build off of [18], which implements a parallel reservoir structure. Of course, we are not learning the weights of the recurrent network and thus cross-contamination is less of an issue, however, we take this simply as motivation. By partitioning one large sub-

reservoir into smaller networks, it was shown that this can prevent information from oversaturating other regions and enforce some sub-structural components [18]. This

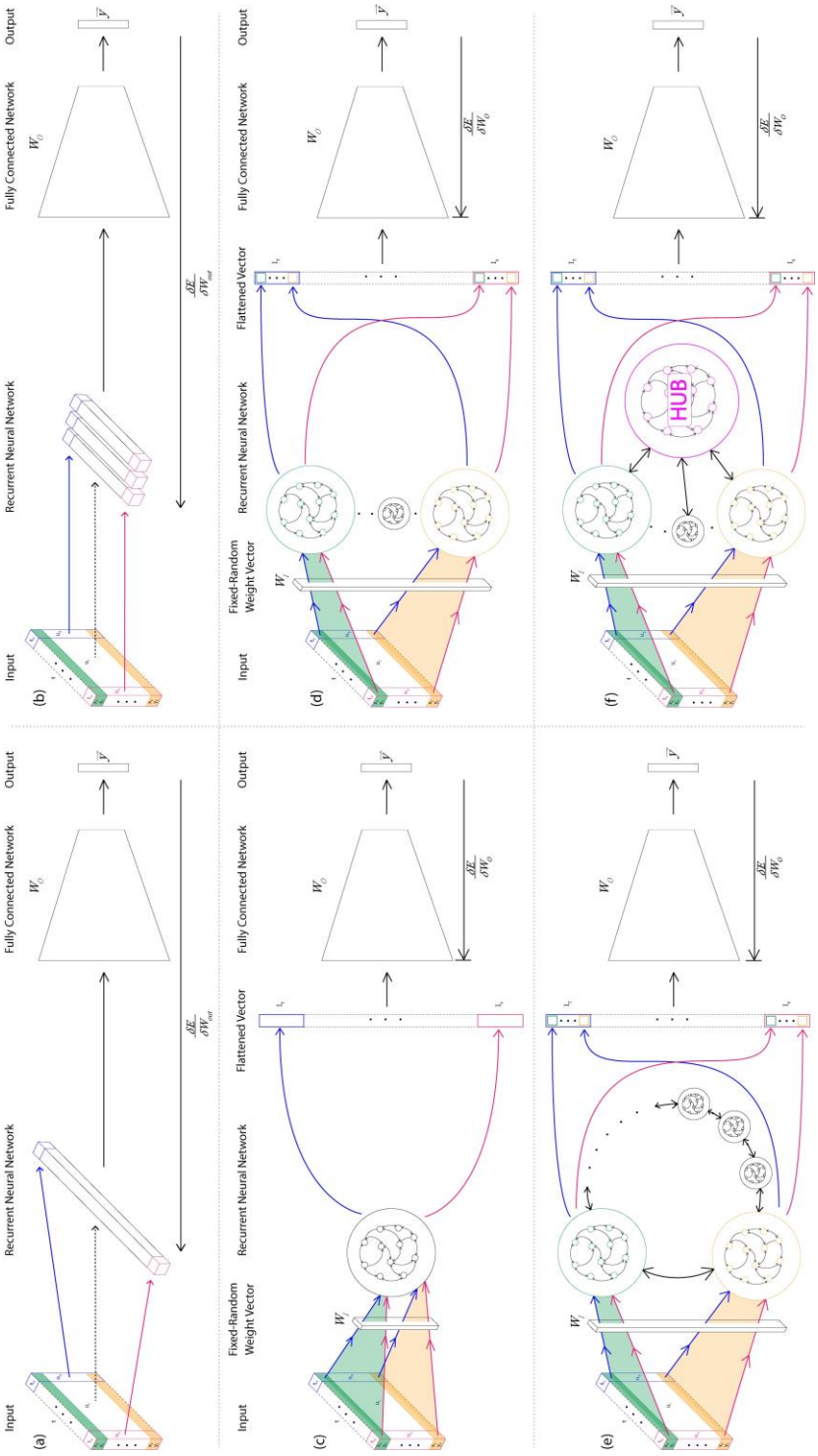


Fig.2 ShownareallvariantsofthevariousLSTM/GRUarchitecturesandESNdeep-readoutarchitecturesweusethroughoutthepaper:(a)1-layer LSTM/GRU(b)3-layersLSTM/GRU(c)SingleDeepRO:Singlereservoirwithdeep-readoutmechanism(d)PDR-ESN:ThePDR-ESNwithoutany connectionsbetweenthesub-reservoirs(e)PDR-ESNRing:ThePDR-ESNwithleftandrightconnectionsbetweensub-reservoirs,resultinginaring structure(f)PDR-ESNHub:PDR-ESNwithconnectionstoacentralhub,wherethehubisnotincludedinthereadout.

4.2 PDR-ESN Architectural Variants

catastrophic forgetting occurs not just in Hopfield networks, but in feed-forward MLP networks as well. There is empirical evidence that partitioning a network can alleviate this catastrophic forgetting [57]. A biologically-inspired reservoir network was shown to improve its capabilities by partitioning activity using gating [58].

Second, and more critically for applications, by separating into smaller subreservoirs, we can decrease the exponential increase of the size of one large reservoir. As the size of the reservoir increases, there simply is not enough memory to hold the recurrent weight matrix for one single large reservoir the size of W_R grows with N^2. As an example, for a large single reservoir of 3200 neurons, we would have 3200^2 recurrent weights, or on the order of 10^7. However, for the same number of total neurons organized into 8 sub-reservoirs with 400 neurons each, this only amounts to 4×400^2, or on the order of 10^6, a full order of magnitude smaller. This would allow for large scaling of network sizes. In practice, we have found that training single reservoirs of size 10^5 or greater starts to become untenable. Third, multiple reservoirs are better adept at handling data that is naturally already partitioned (see Experiments). Due to the shared connectivity matrix described in section 4.2.2, the sub-reservoirs are able to access information about other regions as well; for the PDR-ESN, this is accomplished through sideways connections, whereas for PDR-ESN Hub, it is done through a central hub reservoir. Figure 2 (c) through (f) shows the four different deep-readout architectures that we use in this work.

4.2 PDR-ESN Architectural Variants

In this work, we describe four variants of an ESN with deep readout. The first variant involves one single large reservoir, in a non-parallel setup (Fig 2(c)). We refer to this as Single DeepRO. The first parallelized sub-reservoir architecture involves splitting the large reservoir into multiple sub-reservoirs (Fig 2(d)), where sub-reservoirs have no communication with one another; we refer to this as PDR-ESN.

4.2.1 Parallelized Deep Readout

A basic implementation of a parallel reservoir structure does not allow for communication between reservoirs. This may be important because one sub-reservoirs' activity may benefit from information coming from other sub-reservoirs; for example spatially-located objects in a visual field, where subreservoirs take in information from 2D spatial location kernels. If we wanted to fully connect each sub-reservoir, we could additionally add cross-connection matrices between sub-reservoirs. However, by adding these recurrent weight matrices, we lose the memory efficiency that we gained from splitting into sub-reservoirs. Again dipping into deep learning, we can take inspiration from Convolutional Neural Networks [59], which use shared convolutional weights. Here, we use a *shared* recurrent weight matrix that all sub-reservoirs use to communicate with one another. The first variant where the reservoirs can

communicate (Fig 2(e)) involves a simple left and right connection with each sub-reservoir, resulting in a ring, referred to as PDR-ESN Ring. Lastly, our bio-inspired variant involves all sub-reservoirs having a (shared) fully connected weight matrix with a central hub, which doesn't participate in the readout (Fig 2(f)); we refer to this as PDR-ESN Hub.

4.2.2 Implementation and Update Equations

For all equations, the sub-subscript in capital letters denotes the sub-reservoir update equation for the given architecture. For notational simplicity, we denote the activity vector of each sub-reservoir x_t for PDR-ESN as P_t, for PDR-ESN Ring as R_t, and for PDR-ESN Hub as H_t. We denote each individual sub reservoir as r; we denote the recurrent matrix W_R for sub-reservoir r as $W_{R,r}$, input W_I as $W_{I,r}$, and the connection matrix from hub to reservoir W_{hub} as W_H, while the connections from the reservoir to the hub are symmetric ($W_H{}^T$). Equation 1 becomes, for the parallelized scheme without the cross-talk matrix:

$$P_{t,r} = (1 - \alpha)P_{t-1,r} + \alpha f(W_{I,r}\, u_t + W_{R,r}\, P_{t-1,r}) \quad (7)$$

For the PDR-ESN Ring, an additional ring input is added:

$$\delta_{ring} = \beta(f(W_{ring}\, P_{t-1,r-1}) + f(W_{ring}\, P_{t-1,r+1})) \quad (8)$$

$$R_{t,r} = P_{t,r} + \delta_{ring} \quad (9)$$

For the PDR-ESN Hub, each sub-reservoir additionally adds input from the hub:

$$H_{t,r} = (1 - \alpha)H_{t-1,r} + \alpha f(W_{I,r}\, u_t + W_{R,r}\, H_{t-1,r} + W_H\, x_{t-1,hub}) \quad (10)$$

The state of the hub itself is updated according to the equation:

$$x_{t,hub} = (1 - \alpha)x_{t-1,hub} + \alpha f\big(W_{R,hub}\, x_{t-1,hub} + W_H^T\big(\frac{1}{M}\sum_{r=1}^{M} H_{t-1,r}\big)\big), \quad (11)$$

where M is the number of sub-reservoirs. We assume that all inputs from the sub-reservoirs are scaled by the same factor, and because we multiply the same shared hub connection matrix, we can pull W_H^T out of the summation term and average the result by $\frac{1}{M}$. For a particular training sample $x^{(i)}$, the prediction of the label is made as follows:

$$x_{(i)} = [Res_{(t,i1)}, Res_{t,(i2)}, \cdots, Res_{(t,Ri)}] \quad (12)$$

$$y_{(i)} = W_O x_{(i)}$$

4.3 Deep Feed-Forward Network Backpropagation Readout

Where *Res* is either *P*,*R*, or *H* depending on the architecture used, for all sub-reservoirs *r* from 1 to *R*, where *R* is the number of sub-reservoirs, and ';' is the concatenation operator in Eq. 4. All sub-reservoirs are stored in an *R* dimensional array, referred to henceforth as the sub-reservoir array. For the PDR-ESN Ring, in equation 8, reservoir indexes -1 and +1 reflect the left and right index in the sub-reservoir array. The sub-reservoir connections are wrapped around, forming a ring structure. β is a parameter chosen to reflect how strongly to weigh the cross-talk connections. W_{shared} is the shared connectivity matrix used for all sub-reservoirs, which saves on memory usage. We use the same alpha for each sub-reservoir, although this can be parameterized as well.

4.3 Deep Feed-Forward Network Backpropagation Readout

In this work, as in Eqs. 5, the states of the reservoir are collected in a matrix *X*. The main difference between the deep readout and the standard readout is that instead of performing ridge regression, we map *X* to *Y* using a deep neural network, using stochastic gradient descent, as in [16]. The neurons perform a weighted linear sum of input activity and pass through a non-linear activation function; in our work, we use the Rectified Linear Unit (ReLU). The weights are then updated using standard batched Backpropagation [60], with added modern components (See Experiments). While this does increase the computational complexity versus ridge regression, we argue that the cost-benefit trade-off is desirable. The backpropagation still stops before the recurrent layer, and does not need to be propagated backwards through time. The multiple parallel reservoirs are concatenated at the end, and fed in as a single layer to the input layer of the deep network. As we will see, the computational costs of backpropagation with stochastic gradient descent are more expensive than a simple ridge regression. However, as our results show, and as argued in [27–29], the performance increases, in our case dramatically, when a nonlinear readout is applied. For smaller reservoir sizes, the multi-layer readout appears to be able to untangle confounding information hidden in the reservoir states that ridge regression alone is not powerful enough to untangle. This is particularly important for applications of these networks where larger networks are infeasible. In this current work, we perform all learning offline, per Eq. 5. One potential advantage of doing the learning this way - unsupervised filtering through the reservoir dynamics - is that the backprop network has access to the unsupervised features that the reservoir generates, - all interesting features throughout the time history - which can then be used for any task. In an LSTM, all of the features through the time history are fitted to one label, which makes re-using the LSTM on new tasks less effective. In theory, the last layers can learn task-specific tasks while the first layers of the deep network can be the same; this would allow for transfer learning across tasks.

5 Experiments

For consistency and reproducibility, we report the mean and the standard deviation of the results from 10 runs for all experiments, for each network type. In our experiments, we first test against the state-of-the-art gradient based recurrent neural networks, LSTMs [32] and GRUs [61], both single and "deep" stacked 3-layer. To further test the efficacy of our approach, we compare against a state-of-the-art feed-forward approach to temporal learning, the Transformer [1], which includes self-attention layers. As the original transformer was a sequence-to-sequence model involving a decoder and our tests involve classification and regression (i.e., a single scalar output, rather than a sequence), we removed the decoder of the transformer and replaced it with a fully connected layer, maintaining the powerful deep encoder with self-attention heads. We test against two variants of the transformer: a shallower 3 layer architecture with four "heads", and a deeper 6 layer architecture with 8 heads. As further ablation studies, we also compare against a standard deep MLP (ANN Baseline) (i.e., our readout without the reservoir) as well as a standard Echo State Network with ridge regression readout (ESN Vanilla), i.e., our reservoir without the readout.

As a control measure for the comparisons, after a hyper-parameter grid search, we select a set of fixed hyper-parameters with the best results for the LSTM and GRU networks. For the Transformer network EEG regression task, we found the network was unable to train with the same learning rate; the loss ballooned to infinity. We thus had to reduce the learning rate in order to get the model to converge. To compensate for this reduction and aid the transformer, we increased the number of epochs for the EEG task to 300, rather than 200 for the other networks. The same parameters are applied to all variants of the ESN as well, including the number of epochs, batch size, fully connected networks, optimizer, batch norm layer, computer hardware, and software configurations, etc. For within and between comparisons, we select six variants for each RNN a) the number of layers (single layer and deep stacked) and b) the number of neurons (1600, 2400, and 3200 for classification and 320, 480, and 800 for regression.) We split all data 80/20 for train/test.

We conduct an extensive ablation study with a number of parameters to establish the most salient parameters for all four variants of the PDR-ESN Figure 2 (c)-(f). We also include for both experiments a fully connected feedforward multilayered perceptron (ANN) as the baseline to demonstrate the necessity for a recurrent network for these tasks (essentially, just the deep readout of PDR-ESN). The full list of hyperparameters, as well as additional plots and images, can be found on the papers' Github page.

5.1 Airmarshal Gesture Classification

For our first experiment, we run a label classification on the ChaLearn [62] dataset subset of the Helicopter Aircraft Marshaling [63] gestures, where we extract 8 upper body skeleton 2D keypoints (16 element vector) using OpenPose [64]. We train our networks to learn to classify these gestures into the 9 labels

5.2 DEAP EEG Regression

Fig. 3 ChaLearn Dataset: Helicopter Airmarshal Subset with 8 upper body skeleton keypoints acquired from OpenPose containing 1792 samples - shown is label 3 (Take off) in Figure 4. These examples demonstrate variations due to camera angles, body types, individual styles, hands used, etc. that make the gestures difficult to learn.

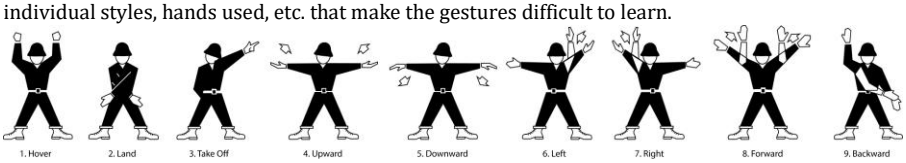


Fig. 4 Helicopter Airmarshal labels, a total of 9 classes, present in the ChaLearn dataset as shown in Figure 4. For this first experiment, we set all the parameters of the networks to be as identical as possible.

We can see from table 1 that all variants of the PDR-ESN outperform the LSTM, GRU, and Transformer self-attention multiheaded networks in terms of accuracy, and are roughly equal to the smaller transformer in terms of training time, outperforming the larger 6-layered transformer. We posit that given larger transformer or data sizes the training time of the transformer will continue to rapidly increase. All PDR-ESN were able to achieve comparable and even superior accuracy to the Single DeepRO, even though it uses an order of magnitude smaller weight matrices. While it may be counter-intuitive to think that transformers are slower, we posit that the architecture itself is faster to train than a gradient-based RNN because transformers are still feed-forward and avoid unrolling the network via backpropagation through time. The highest accuracy was found with the PDR-ESN Hub, with a hub size of 200. We found that for a problem as small as 16-dimensions, the backpropagation deep readout was able to make up for any shortcomings in separability issues as described in [18], and that as we can see with the Vanilla ESN, the reservoir does not do a good job of linearly separating the data. PDR-ESN variants all perform as well or better in accuracy as the Single DeepRO while using less memory. All PDR-ESNs used 8 sub-reservoirs. It is important to note that the best variants of the hub use a hub with a *smaller* time constant than the sub-reservoirs, meaning the hub updates slower. This means the hub acts as a longer timescale integrator of information, which mirrors results from neuroscience showing that higher level regions act at slower timescales [65].

5.2 DEAP EEG Regression

For our second experiment, we run a regression learning task on the DEAP EEG dataset [66] (Fig. 5). We use all 40 channels for training corresponding to the raw sensor recordings from 32 EEG electrodes and 8 body physiological sensors. The dataset has recordings from 32 participants, each with 40 trials,

Classification Methods	**acc (%) ± sd**	**time (s) ± sd**
ANN Baseline	70.52 ± 2.45	103 ± 2
LSTM 1-Layer	77.46 ± 1.71	1459 ± 21
LSTM Deep	77.5 ± 2.77	1516 ± 39
GRU 1-Layer	72.3 ± 3.36	1038 ± 7
GRU Deep	77.42 ± 2.04	1338 ± 66
ESN Vanilla	67.32 ± 0.46	42 ± 0
Single DeepRO	83.9 ± 0.84	262 ± 3
Transformer 3-Layer	82.30 ± 1.21	**292 ± 3**
Transformer 6-Layer	81.15 ± 1.98	576 ± 2
PDR-ESN	83.73 ± 0.97	**289 ± 9**
PDR-ESN Ring	82.93 ± 1.59	327 ± 3
PDR-ESN Hub	**84.08 ± 1.08**	301 ± 2

Table 1 Airmarshal gesture classification accuracy and training time (in seconds) for 1600-size networks. Deep LSTM and GRU networks used 3 stacked recurrent layers. ANN is the deep feed-forward ANN baseline (See Fig. 2 for visualization of each setup). For PDR-ESN Hub, a hub size of 300 was used. Each number is the mean over 10 runs, with std. deviation posted as ±. Our novel interconnected parallel variants offer a good trade-off of high accuracy while still drastically lowering training time, even outperforming Transformer architectures with self-attention. Our proposed PDR-ESN also outperforms both the 3- and 6-layer Transformer encoder architecture while maintaining roughly the same training speed.

Regression Methods	**mse ± sd**	**time (s) ± sd**
LSTM 1-Layer	1.48 ± 0.1	699 ± 38
LSTM Deep	1.28 ± 0.13	1391 ± 19
GRU 1-Layer	1.76 ± 0.1	551 ± 20
GRU Deep	1.46 ± 0.12	1299 ± 20
Transformer 3-Layer	1.79 ± 0.10	212 ± 0.4
Transformer 6-Layer	1.31 ± 0.10	369 ± 1
Single DeepRO	**1.27 ± 0.11**	**166 ± 4**

PDR-ESN Hub	1.41 ± 0.14	788 ± 20

Table 2 Results from the EEG emotion regression learning task for the 'arousal' label. Each network was run individually on all 32 participants 10 times each, and mean and standard deviation across all runs are posted. For the Transformer 6 layer model, we used the P6000 GPU model and for the others the NVIDIA 1080ti; the P6000 was needed as the model did not fit in the 1080ti memory. Therefore the time comparison is not entirely fair for the 6 layer transformer. We also made improvements to the code (not submitted due to length) that sped up the Hub speed (see other chart).

watching 1-minute video clips, where EEG and bio-sensor recordings were taken during the video watching session. Afterwards, participants were asked to rate emotions (valence, arousal, dominance, liking) on a 9-point scale. As in [67], we break up each 60-second trial into 1-second epochs, corresponding to 134-length vectors; each 1 second epoch is given the same label as the corresponding 60 second trial, giving 2400 samples per participant. Unlike most papers that run classification on high/low values, we run pure regression on the numerical label value and report results after running the network 10 times and showing the mean and std. dev. in Table 2. Lastly, shown here are the regression results

5.3 Discussion

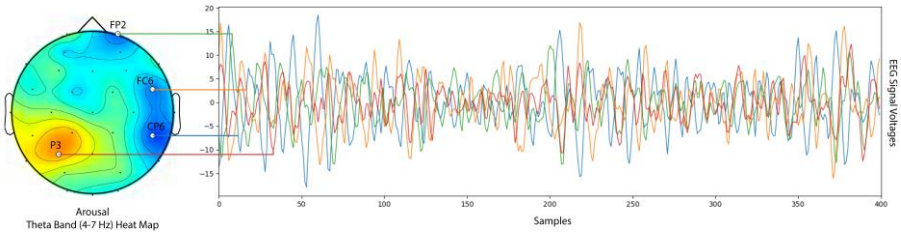


Fig. 5 Left: A heatmap depicting the EEG nodes overlaid with the correlations in the activity that correspond to the 'arousal' value. Red values indicate positive correlation and blue negative. For our purposes, we use all channels and let the network discern which channels are of importance.[66] Right: The raw EEG signal used for training, with lines showing the corresponding EEG node locations for example channels.

for "arousal", corresponding to the second index of the label array. All input data was channel-wise normalized between -1 and 1. Thus at each of 134time steps, a 40-element vector was fed into the networks corresponding to the 40 channels, and the network output was regressed to the arousal label. The network sizes for the EEG task were not uniform as in the gesture task, due to the challenging nature of the dataset; however, we feel that the runs demonstrate that architectures' performance.

5.3.1 Ablation Studies

Accuracy: We performed various ablation studies to determine and isolate the sources of variance in the network runs for the Airmashal classification task. First, to reduce the variance, all of our results show the average performance over 20 runs (see particular studies in Experiments for details). Our first ablation, evaluates the size of the reservoir, using a standard L2 readout for the Airmarshal dataset. We find that as the size of the reservoir decreases, the accuracy suffers. With the deep readout, we are able to achieve over 80% accuracy even with a reservoir of size 360.

For our second and larger ablation study, we study the classification of each of the four major variants of our PDR-ESN as a function of the total number of neurons, versus the single large reservoir with the same number of total neurons. Again, as expected, more neurons lead to generally higher accuracy, although there is a diminishing return once the size gets too large. We find that the PDR-ESN Hub generally performs as well as the Single DeepRO for a reservoir size of 480, which we hypothesize is due to its ability to transfer information between sub-reservoirs via the hub The Hub variant continues to perform well as the size of the reservoir increases, and the ring variant performs the best at the largest size.

Training time: Lastly, we tested the total training time among all variants of the reservoir, as a function of the total number of neurons. (Fig. 6). We can see that all variants of the deep readout reservoir are significantly less time

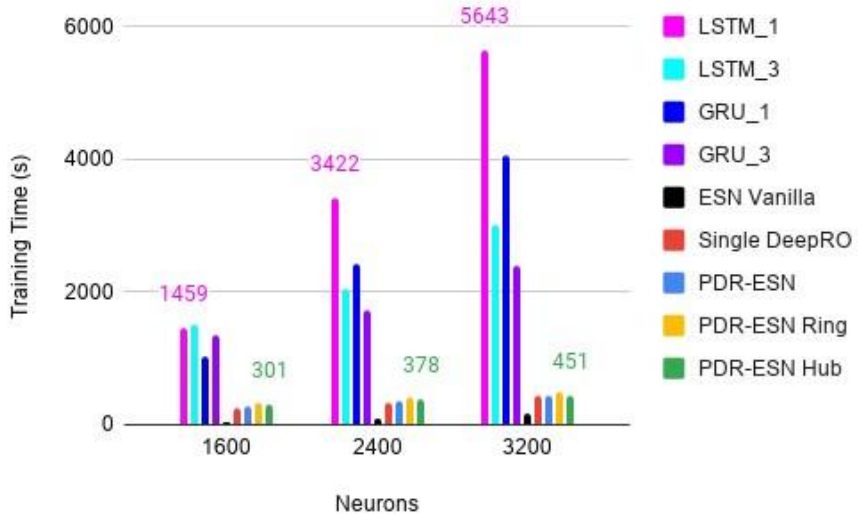


Fig. 6 This chart shows training time for three configurations of sizes of the recurrent neuron pool (total neuron count): 1600, 2400, and 3200. These numbers were chosen as multiples of 8, given our input data. As we can see, the training time is dramatically higher for the LSTM and GRU architectures than for the PDR-ESN architectures, as well as having lower accuracies than the PDR-ESN. The training time would and will be even lower for PDR-ESN with more optimized CUDA code (as pyTorch is optimized for LSTM/GRU) which will be forthcoming in the future.

consuming for training than the stacked gradient-based RNNs, with LSTMs taking the longest. While the 3-layer transformer architecture trains relatively quickly, it does not match the task performance of the PDR-ESN variants, and the base PDR-ESN trains roughly as fast. Further, when the transformer is increased in size, the training time nearly doubles for both datasets. In addition, the training time for the PDR-ESN grows at a much smaller rate with increasing neuron count. While the ESN Vanilla has the fastest time, this chart emphasizes that we aim to achieve a good trade-off between training time and accuracy, which are demonstrated in Fig. 6 and the results tables.

5.3.2 Analysis

While the ESN Vanilla is faster, that comes at a cost of extreme loss of accuracy: 67% versus 84% for the PDR-ESN, a net difference of 17%. The feed-forward network is also fast (twice as slow as the ESN Vanilla) but suffers equally from accuracy. The gradient RNNs (LSTM/ GRU) show good accuracy (far less than the PDR-ESN on the gesture, although the 3-layer LSTM does slightly outperform on the DEAP dataset) but at the cost of almost 5x slower training time. While the smaller 3-layer transformer architecture trains relatively quickly owing to the lack of backpropagation, we find that it suffers in terms of classification and regression accuracy. As the size of the transformer size doubles to 6 layers and 8 heads, the EEG accuracy increases but the training time also doubles. Our hypothesis is that for transformers to work, in line with other findings, huge datasets are required in addition to other fine-tuning and engineering of the transformer, and that these large datasets require larger networks, which in turn require large training times. We further hypothesize, though, that the PDR-ESN would continue to perform well given large amounts of data. We thus argue here that the PDR-ESN offers a good trade-off between high accuracy and fast train time.

6 Conclusion

We demonstrated the use of a backpropagation hybrid mechanism for parallel reservoir computing with various parallel architectures, and the application on two difficult real-world datasets. Experimental results show it balances effectively the trade-off between accuracy and training time versus gradient-based RNNs. We show that it can be used as an alternative to LSTMs, GRUs, and

Transformers for training time-series data, on both classification and regression tasks. Future work will expand upon and further integrate deep learning techniques into the reservoir learning framework. We offer this novel network as a new route to learning temporally changing or sequential data in a way that utilizes random recurrent matrices without the use of Backpropagation Through Time. We believe this can form the building blocks for future architectures that can modularize the reservoirs for hierarchical systems.

Acknowledgements

This work was supported by NSF awards DGE-1632976, BCS 1824198 and OISE 2020624.

Conflicts of Interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Bibliography

MAIN TEXT

[Aharonov and Vaidman, 1991] Aharonov, Y. and Vaidman, L. (1991). Complete description of a quantum system at a given time. *Journal of Physics A: Mathematical and General*, 24(10):2315.

[Aloimonos et al., 1988] Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. *International journal of computer vision*, 1:333–356.

[Alom et al., 2018] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., and Asari, V. K. (2018). The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*.

[Amari, 1989] Amari, S.-I. (1989). Dynamical stability of formation of cortical maps. In *Dynamic interactions in neural networks: Models and data*, pages 15–34. Springer.

[Bahdanau et al., 2016] Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-toend attention-based large vocabulary speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4945–4949. IEEE.

[Bellec et al., 2020] Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., and Maass, W. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625.

[Boden, 2002] Boden, M. (2002). A guide to recurrent neural networks and backpropagation. *the Dallas project*, 2(2):1–10.

[Bolam et al., 2009] Bolam, J., Brown, M., Moss, J., and Magill, P. (2009). Basal ganglia: internal organization. *Encyclopedia of Neuroscience*, 2:97–104.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[Bullier, 2003] Bullier, J. (2003). Cortical connections and functional interactions between visual cortical areas. *Neuropsychol. Vis*, pages 251–256.

[Burgess et al., 2018] Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. (2018). Understanding disentangling in β-vae. *arXiv preprint arXiv:1804.03599*.

[Buzs´aki, 2010] Buzs´aki, G. (2010). Neural syntax: cell assemblies, synapsembles, and readers. *Neuron*, 68(3):362–385.

[Buzs´aki and Kandel, 1998] Buzs´aki, G. and Kandel, A. (1998). Somadendritic backpropagation of action potentials in cortical pyramidal cells of the awake rat. *Journal of neurophysiology*, 79(3):1587–1591.

[Cao et al., 2018] Cao, W., Wang, X., Ming, Z., and Gao, J. (2018). A review on neural networks with random weights. *Neurocomputing*, 275:278–287.

[Carlson, 1997] Carlson, R. A. (1997). Meshing glenberg with piaget, gibson, and the ecological self. *Behavioral and Brain Sciences*, 20(1):21–21.

[Cauwenberghs, 2013] Cauwenberghs, G. (2013). Reverse engineering the cognitive brain. *Proceedings of the national academy of sciences*, 110(39):15512–15513.

[Cooper, 2015] Cooper, J. M. (2015). *Plato's Theaetetus*. Routledge.

[Cowan, 2008] Cowan, N. (2008). What are the differences between long-term, short-term, and working memory? *Progress in brain research*, 169:323–338.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

[Dan and Poo, 2004] Dan, Y. and Poo, M.-m. (2004). Spike timing-dependent plasticity of neural circuits. *Neuron*, 44(1):23–30.

[Davies et al., 2018] Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99.

[Dew and Cabeza, 2011] Dew, I. T. and Cabeza, R. (2011). The porous boundaries between explicit and implicit memory: behavioral and neural evidence. *Annals of the New York Academy of Sciences*, 1224(1):174–190.

[Diamond and Rose, 1994] Diamond, D. M. and Rose, G. M. (1994). Does associative ltp underlie classical conditioning? *Psychobiology*, 22(4):263–269.

[Dragoi, 2020] Dragoi, G. (2020). Cell assemblies, sequences and temporal coding in the hippocampus. *Current opinion in neurobiology*, 64:111–118.

[Dragoi and Buzs´aki, 2006] Dragoi, G. and Buzs´aki, G. (2006). Temporal encoding of place sequences by hippocampal cell assemblies. *Neuron*, 50(1):145–157.

[Edelman, 1987] Edelman, G. M. (1987). *Neural Darwinism: The theory of neuronal group selection.* Basic books.

[Eichenbaum, 2013] Eichenbaum, H. (2013). Memory on time. *Trends in cognitive sciences*, 17(2):81–88.

[Enel et al., 2016] Enel, P., Procyk, E., Quilodran, R., and Dominey, P. F. (2016). Reservoir computing properties of neural dynamics in prefrontal cortex. *PLoS computational biology*, 12(6):e1004967.

[Engel et al., 1992] Engel, A. K., K¨onig, P., Kreiter, A. K., Schillen, T. B., and Singer, W. (1992). Temporal coding in the visual cortex: new vistas on integration in the nervous system. *Trends in neurosciences*, 15(6):218–226.

[Evanusa et al., 2020] Evanusa, M., Fermuller, C., and Aloimonos, Y. (2020). A deep 2-dimensional dynamical spiking neuronal network for temporal encoding trained with stdp.

[Evanusa et al., 2022] Evanusa, M., Shrestha, S., Patil, V., Fermu¨ller, C., Girvan, M., and Aloimonos, Y. (2022). Deep-readout random recurrent neural networks for real-world temporal data. *SN Computer Science*, 3(3):222.

[Evanusa et al., 2023] Evanusa, M. S., Patil, V., Girvan, M., Goodman, J., Fermu¨ller, C., and Aloimonos, Y. (2023). t-convesn: Temporal convolution-readout for random recurrent neural networks. In *International Conference on Artificial Neural Networks*, pages 140–151. Springer.

[Feld and Born, 2017] Feld, G. B. and Born, J. (2017). Sculpting memory during sleep: concurrent consolidation and forgetting. *Current opinion in neurobiology*, 44:20–27.

[Figueredo, 2021] Figueredo, V. M. (2021). The ancient heart: What the heart meant to our ancestors. *Journal of the American College of Cardiology*, 78(9):957–959.

[Fraccaro et al., 2017] Fraccaro, M., Kamronn, S., Paquet, U., and Winther, O. (2017). A disentangled recognition and nonlinear dynamics model for unsupervised learning. *Advances in neural information processing systems*, 30.

[Fukushima et al., 1983] Fukushima, K., Miyake, S., and Ito, T. (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834.

[Gallicchio et al., 2017] Gallicchio, C., Micheli, A., and Pedrelli, L. (2017). Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99.

[Garcia and Buffalo, 2020] Garcia, A. D. and Buffalo, E. A. (2020). Anatomy and function of the primate entorhinal cortex. *Annual Review of Vision Science*, 6:411–432.

[Gemici et al., 2017] Gemici, M., Hung, C.-C., Santoro, A., Wayne, G., Mohamed, S., Rezende, D. J., Amos, D., and Lillicrap, T. (2017). Generative temporal models with memory. *arXiv preprint arXiv:1702.04649*.

[Gilbert and Burgess, 2008] Gilbert, S. J. and Burgess, P. W. (2008). Executive function. *Current biology*, 18(3):R110–R114.

[Glenberg, 1997] Glenberg, A. M. (1997). What memory is for. *Behavioral and brain sciences*, 20(1):1–19.

[Glenberg and Kaschak, 2002] Glenberg, A. M. and Kaschak, M. P. (2002). Grounding language in action. *Psychonomic bulletin & review*, 9(3):558–565.

[Graves et al., 2014] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[Gu et al., 2021] Gu, A., Goel, K., and R´e, C. (2021). Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*.

[Hampshire et al., 2012] Hampshire, A., Highfield, R. R., Parkin, B. L., and Owen, A. M. (2012). Fractionating human intelligence. *Neuron*, 76(6):1225–1237.

[Hart et al., 1999] Hart, T., Schwartz, M. F., and Mayer, N. (1999). Executive function: Some current theories and their applications. *The evaluation and treatment of mild traumatic brain injury*, pages 133–148.

[Hebb, 2005] Hebb, D. O. (2005). *The organization of behavior: A neuropsychological theory*. Psychology press.

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

[Izhikevich, 2003] Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572.

[Jaeger, 2001] Jaeger, H. (2001). The “echo state” approach to analysing and training recurrent neural networkswith an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13.

[Josselyn and Tonegawa, 2020] Josselyn, S. A. and Tonegawa, S. (2020). Memory engrams: Recalling the past and imagining the future. *Science*, 367(6473):eaaw4325.

[Kaiser et al., 2020] Kaiser, J., Mostafa, H., and Neftci, E. (2020). Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424.

[Kohonen, 2006] Kohonen, T. (2006). Self-organizing neural projections. *Neural networks*, 19(6-7):723–733.

[Krishnan et al., 2017] Krishnan, R., Shalit, U., and Sontag, D. (2017). Structured inference networks for nonlinear state space models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

[Krupa et al., 1993] Krupa, D. J., Thompson, J. K., and Thompson, R. F. (1993). Localization of a memory trace in the mammalian brain. *Science*, 260(5110):989–991.

[Lake et al., 2017] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, 40.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[Linnainmaa, 1970] Linnainmaa, S. (1970). *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master’s Thesis (in Finnish), Univ. Helsinki. [London and H¨ausser, 2005] London, M. and H¨ausser, M. (2005). Dendritic computation. *Annu. Rev. Neurosci.*, 28:503–532.

[Lu et al., 2024] Lu, C., Schroecker, Y., Gu, A., Parisotto, E., Foerster, J., Singh, S., and Behbahani, F. (2024). Structured state space models for in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

[Maass et al., 2002] Maass, W., Natschl¨ager, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

[Mareels et al., 1987] Mareels, I. M., Anderson, B. D., Bitmead, R. R., Bodson, M., and Sastry, S. S. (1987). Revisiting the mit rule for adaptive control. In *Adaptive Systems in Control and Signal Processing 1986*, pages 161–166. Elsevier.

[Markram, 2006] Markram, H. (2006). The blue brain project. *Nature Reviews Neuroscience*, 7(2):153–160.

[Marschall et al., 2020] Marschall, O., Cho, K., and Savin, C. (2020). A unified framework of online learning algorithms for training recurrent neural networks. *Journal of machine learning research*, 21(135):1–34.

[Marshall and Born, 2007] Marshall, L. and Born, J. (2007). The contribution of sleep to hippocampusdependent memory consolidation. *Trends in cognitive sciences*, 11(10):442–450.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[McSweeney and Murphy, 2014] McSweeney, F. K. and Murphy, E. S. (2014). The wiley blackwell handbook of operant and classical conditioning.

[Minsky and Papert, 2017] Minsky, M. and Papert, S. A. (2017). *Perceptrons: An introduction to computational geometry*. MIT press.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[Munkhdalai et al., 2024] Munkhdalai, T., Faruqui, M., and Gopal, S. (2024). Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*.

[Murray et al., 2018] Murray, E. A., Wise, S. P., and Graham, K. S. (2018). Representational specializations of the hippocampus in phylogenetic perspective. *Neuroscience letters*, 680:4–12.

[Neftci et al., 2017] Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017). Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in neuroscience*, 11:324.

[Nelson and Rinzel, 1995] Nelson, M. and Rinzel, J. (1995). The hodgkin-huxley model. *The book of genesis*, 2.

[Parr et al., 2022] Parr, T., Pezzulo, G., and Friston, K. J. (2022). *Active inference: the free energy principle in mind, brain, and behavior*. MIT Press.

[Pathak et al., 2018] Pathak, J., Hunt, B., Girvan, M., Lu, Z., and Ott, E. (2018). Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical review letters*, 120(2):024102.

[Patil et al., 2022] Patil, V., Evanusa, M., and JaJa, J. (2022). Dot-vae: Disentangling one factor at a time. In *International Conference on Artificial Neural Networks*, pages 109–120. Springer.

[Patil et al., 2023] Patil, V., Evanusa, M., and JaJa, J. (2023). Protovae: Prototypical networks for unsupervised disentanglement. *arXiv preprint arXiv:2305.09092*.

[Pelzer et al., 2013] Pelzer, E. A., Hintzen, A., Goldau, M., von Cramon, D. Y., Timmermann, L., and Tittgemeyer, M. (2013). Cerebellar networks with basal ganglia: Feasibility for tracking cerebello-pallidal and subthalamo-cerebellar projections in the human brain. *European Journal of Neuroscience*, 38(8):3106–3114. [Petersen and Posner, 2012] Petersen, S. E. and Posner, M. I. (2012). The attention system of the human brain: 20 years after. *Annual review of neuroscience*, 35:73–89.

[Popa and Ebner, 2019] Popa, L. S. and Ebner, T. J. (2019). Cerebellum, predictions and errors. *Frontiers in cellular neuroscience*, 12:524.

[Raibert, 1977] Raibert, M. H. (1977). *Motor control and learning by the state space model*. PhD thesis, Massachusetts Institute of Technology.

[Ram´on y Cajal, 1906] Ram´on y Cajal, S. (1906). Morfolog´ıa de la c´elula nerviosa. *Archivos de Pedagog´ıa y Ciencias Afines*, 1.

[Rangapuram et al., 2018] Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y., and Januschowski, T. (2018). Deep state space models for time series forecasting. *Advances in neural information processing systems*, 31.

[Rapoport, 1990] Rapoport, S. I. (1990). Integrated phylogeny of the primate brain, with special reference to humans and their diseases. *Brain Research Reviews*, 15(3):267–294.

[Renner et al., 2019a] Renner, A., Evanusa, M., and Sandamirskaya, Y. (2019a). Event-based attention and tracking on neuromorphic hardware. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1709–1716. IEEE.

[Renner et al., 2019b] Renner, A., Evanusa, M., and Sandamirskaya, Y. (2019b). Event-based attention and tracking on neuromorphic hardware. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[Rizzi-Wise and Wang, 2021] Rizzi-Wise, C. A. and Wang, D. V. (2021). Putting together pieces of the lateral septum: multifaceted functions and its neural pathways. *ENeuro*, 8(6).

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Rosenblatt, 1961] Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[Sah et al., 2003] Sah, P., Faber, E. L., Lopez de Armentia, M., and Power, J. (2003). The amygdaloid complex: anatomy and physiology. *Physiological reviews*, 83(3):803–834.

[Sandamirskaya, 2014] Sandamirskaya, Y. (2014). Dynamic neural fields as a step toward cognitive neuromorphic architectures. *Frontiers in Neuroscience*, 7:276.

[Santoro et al., 2016] Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Metalearning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR.

[Schilder et al., 2020] Schilder, B. M., Petry, H. M., and Hof, P. R. (2020). Evolutionary shifts dramatically reorganized the human hippocampal complex. *Journal of Comparative Neurology*, 528(17):3143–3170.

[Sch¨oner, 2016] Sch¨oner, G. (2016). *Dynamic thinking: A primer on dynamic field theory*. Oxford University Press.

[Schultz, 2016] Schultz, W. (2016). Reward functions of the basal ganglia. *Journal of neural transmission*, 123:679–693.

[Semedo et al., 2022] Semedo, J. D., Jasper, A. I., Zandvakili, A., Krishna, A., Aschner, A., Machens, C. K., Kohn, A., and Yu, B. M. (2022). Feedforward and feedback interactions between visual cortical areas use different population activity patterns. *Nature communications*, 13(1):1099.

[Silver et al., 2021] Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299:103535.

[Snell et al., 2017] Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30.

[Sporns, 2014] Sporns, O. (2014). Network neuroscience. *The Future of the Brain: Essays by the World's Leading Neuroscientists*, page 90.

[Squire, 2004] Squire, L. R. (2004). Memory systems of the brain: a brief history and current perspective. *Neurobiology of learning and memory*, 82(3):171–177.

[Squire, 2009] Squire, L. R. (2009). The legacy of patient hm for neuroscience. *Neuron*, 61(1):6–9.

[Stork, 1989] Stork (1989). Is backpropagation biologically plausible? In *International 1989 Joint Conference on Neural Networks*, pages 241–246. IEEE.

[Sutton et al., 1999] Sutton, R. S., Barto, A. G., et al. (1999). Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134.

[Theunissen and Miller, 1995] Theunissen, F. and Miller, J. P. (1995). Temporal encoding in nervous systems: a rigorous definition. *Journal of computational neuroscience*, 2:149–162.

[Tishby et al., 2000] Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. *arXiv preprint physics/0004057*.

[Tong and Tanaka, 2018] Tong, Z. and Tanaka, G. (2018). Reservoir computing with untrained convolutional neural networks for image recognition. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 1289–1294. IEEE.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L ., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[Vergara et al., 2019] Vergara, R. C., Jaramillo-Riveri, S., Luarte, A., Mo¨enne-Loccoz, C., Fuentes, R., Couve, A., and Maldonado, P. E. (2019). The energy homeostasis principle: neuronal energy regulation drives local network dynamics generating behavior. *Frontiers in computational neuroscience*, 13:49.

[Von Foerster and von Foerster, 2003] Von Foerster, H. and von Foerster, H. (2003). *What is memory that it may have hindsight and foresight as well?* Springer.

[Wang, 2021] Wang, J. X. (2021). Meta-learning in natural and artificial intelligence. *Current Opinion in Behavioral Sciences*, 38:90–95.

[Weiger, 1997] Weiger, W. A. (1997). Serotonergic modulation of behaviour: a phylogenetic overview. *Biological Reviews*, 72(1):61–95.

[Wess and R¨oder, 1977] Wess, O. and R¨oder, U. (1977). A holographic model for associative memory chains. *Biological Cybernetics*, 27(2):89–98.

[Wikipedia contributors, 2024] Wikipedia contributors (2024). State-space representation — Wikipedia, the free encyclopedia. [Online; accessed 24-May-2024].

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

A DEEP 2-DIMENSIONAL DYNAMICAL SPIKING NEURONAL NETWORK FOR TEMPORAL ENCODING TRAINED WITH STDP

[1] Y. Abu-Mostafa and J. S. Jacques. Information capacity of the hopfield model. *IEEE Transactions on Information Theory*, 31(4):461–464, 1985.

[2] G.-q. Bi and M.-m. Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual Review of Neuroscience*, 24(1):139–166, 2001.

[3] S. M. Bohte, J. N. Kok, and H. La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002.

[4] A. Borst and F. E. Theunissen. Information theory and neural coding. *Nature neuroscience*, 2(11):947, 1999.

[5] V. Braitenberg and A. Schüz. *Anatomy of the cortex: statistics and geometry*, volume 18. Springer Science & Business Media, 2013.

[6] D. A. Butts, C. Weng, J. Jin, C.-I. Yeh, N. A. Lesica, J.-M. Alonso, and G. B. Stanley. Temporal precision in the neural code and the timescales of natural vision. *Nature*, 449(7158):92, 2007.

[7] W. Chaisangmongkon, S. K. Swaminathan, D. J. Freedman, and X.-J. Wang. Computing by robust transience: how the fronto-parietal network performs sequential, category-based decisions. *Neuron*, 93(6):1504–1517, 2017.

[8] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[9] T. Delbruck. Frame-free dynamic digital vision. In *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, pages 21–26, 2008.

[10] M. Diesmann, M.-O. Gewaltig, and A. Aertsen. Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402(6761):529, 1999.

[11] G. M. Edelman. *Neural Darwinism: The Theory of Neuronal Group Selection.* Basic books, 1987.

[12] M. Eickenberg, A. Gramfort, G. Varoquaux, and B. Thirion. Seeing it all: Convolutional network layers map the function of the human visual system. *NeuroImage*, 152:184–194, 2017.

[13] P. Enel, E. Procyk, R. Quilodran, and P. F. Dominey. Reservoir computing properties of neural dynamics in prefrontal cortex. *PLoS computational biology*, 12(6):e1004967, 2016.

[14] W. Gerstner, A. K. Kreiter, H. Markram, and A. V. Herz. Neural codes: firing rates and beyond. *Proceedings of the National Academy of Sciences*, 94(24):12740–12741, 1997.

[15] S. Ghosh-Dastidar and H. Adeli. Spiking neural networks. *International journal of neural systems*, 19(04):295–308, 2009.

[16] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[17] A. L. Hodgkin and A. F. Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *The Journal of Physiology*, 116(4):449–472, 1952.

[18] A. Holtmaat and P. Caroni. Functional and structural underpinnings of neuronal assembly formation in learning. *Nature neuroscience*, 19(12):1553, 2016.

[19] R. Hosaka, O. Araki, and T. Ikeguchi. Stdp provides the substrate for igniting synfire chains by spatiotemporal input patterns. *Neural computation*, 20(2):415–435, 2008.

[20] T. Iakymchuk, A. Rosado-Muñoz, J. F. Guerrero-Martínez, M. Bataller-Mompeán, and J. V. Francés-Víllora. Simplified spiking neural network architecture and stdp learning algorithm applied to image classification. *EURASIP Journal on Image and Video Processing*, 2015(1):4, 2015.

[21] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[22] E. M. Izhikevich, J. A. Gally, and G. M. Edelman. Spike-timing dynamics of neuronal groups. *Cerebral Cortex*, 14(8):933–944, 2004.

[23] N. Kasabov. Evolving spiking neural networks and neurogenetic systems for spatio-and spectrotemporal data modelling and pattern recognition. In *Advances in Computational Intelligence*, pages 234–260. Springer, 2012.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[25] V. A. Lamme and P. R. Roelfsema. The distinct modes of vision offered by feedforward and recurrent processing. *Trends in neurosciences*, 23(11):571–579, 2000.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] D. T. Liley and J. J. Wright. Intracortical connectivity of pyramidal and stellate cells: estimates of synaptic densities and coupling symmetry. *Network: Computation in Neural Systems*, 5(2):175–189, 1994.

[28] A. Litwin-Kumar and B. Doiron. Formation and maintenance of neuronal assemblies through synaptic plasticity. *Nature communications*, 5:5319, 2014.

[29] D. Luebke. Cuda: Scalable parallel programming for high-performance scientific computing. In *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, pages 836–838. IEEE, 2008.

[30] M. Lukoševicius and H. Jaeger. Reservoir computing approaches to recurrent neural networkˇ training. *Computer Science Review*, 3(3):127–149, 2009.

[31] E. D. Lumer, G. M. Edelman, and G. Tononi. Neural dynamics in a model of the thalamocortical system. i. layers, loops and the emergence of fast synchronous rhythms. *Cerebral cortex (New York, NY: 1991)*, 7(3):207–227, 1997.

[32] Z. F. Mainen and T. J. Sejnowski. Reliability of spike timing in neocortical neurons. *Science*, 268(5216):1503–1506, 1995.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[34] S. Ostojic and N. Brunel. From spiking neuron models to linear-nonlinear models. *PLoS computational biology*, 7(1):e1001056, 2011.

[35] I. Segev, R. E. Burke, and M. Hines. Compartmental models of complex neurons. *Methods in neuronal modeling*, 63, 1989.

[36] T. Sejnowski and T. Delbruck. The language of the brain: The brain makes sense if our experiences by focusing closely on the timing of the impulses that flow through billions of nerve cells. *Scientific American*, 307(4):54, 2012.

[37] T. J. Sejnowski. The book of hebb. *Neuron*, 24(4):773–776, 1999.

[38] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[39] A. E. Teschendorff and S. Severini. Increased entropy of signal transduction in the cancer metastasis phenotype. *BMC systems biology*, 4(1):104, 2010.

[40] R. Urbanczik and W. Senn. Similar nonleaky integrate-and-fire neurons with instantaneous couplings always synchronize. *SIAM Journal On Applied Mathematics*, 61(4):1143–1155, 2001.

[41] R. R. d. R. van Steveninck, G. D. Lewen, S. P. Strong, R. Koberle, and W. Bialek. Reproducibility and variability in neural spike trains. *Science*, 275(5307):1805–1808, 1997.

[42] R. VanRullen, R. Guyonneau, and S. J. Thorpe. Spike times make sense. *Trends in neurosciences*, 28(1):1–4, 2005.

[43] A. J. Watt and N. S. Desai. Homeostatic plasticity and stdp: keeping a neuron's cool in a fluctuating world. *Frontiers in synaptic neuroscience*, 2:5, 2010.

[44] S. Zeki and S. Shipp. The functional logic of cortical connections. *Nature*, 335(6188):311, 1988.

EVENT-BASED ATTENTION AND TRACKING ON NEUROMORPHIC HARDWARE

[1] X. Lagorce and R. Benosman, "STICK: Spike Time Interval Computational Kernal, a Framework for General Purpose Computation Using Neurons, Precise Timing and Synchrony," *Neural computation*, vol. 27, pp. 2261–2317, 2015. 1

[2] X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman, "HOTS: A Hierarchy Of event-based Time-Surfaces for pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8828, no. c, pp. 1–1, 2016. [Online]. Available: http://ieeexplore.ieee. org/lpdocs/epic03/wrapper.htm?arnumber=7508476 1

[3] M. Osswald, S.-H. Ieng, R. Benosman, and G. Indiveri, "A spiking neural network model of 3D perception for event-based neuromorphic stereo vision systems," *Scientific Reports*, vol. 7, no. December 2016, p. 40703, 2017. [Online]. Available: http://www.nature.com/articles/ srep40703 1

[4] D. R. Valeiras, G. Orchard, S. H. Ieng, and R. B. Benosman, "Neuromorphic Event-Based 3D Pose Estimation," vol. 9, no. January, pp. 1–15, 2015. 1

[5] J. Carneiro, S. H. Ieng, C. Posch, and R. Benosman, "Eventbased 3D reconstruction from neuromorphic retinas," *Neural Networks*, vol. 45, pp. 27–38, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2013.03.006 1

[6] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan, "Asynchronous frameless event-based optical flow." *Neural networks : the official journal of the International Neural Network Society*, vol. 27, pp. 32–7, mar 2012. [Online]. Available: http://www.ncbi.nlm.nih. gov/pubmed/22154354 1

[7] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015. 1

[8] M. Davies, N. Srinivasa, T.-h. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-k. Lin, A. Lines, R. Liu, D. Mathaikutty, S. Mccoy, A. Paul, J. Tse, G. Venkataramanan, Y.-h. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: a Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82– 99, 2018. 1, 2

[9] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, D. S. Modha, C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, D. Rasmussen, M. Mishkin, L. G. Ungerleider, K. A. Macko, B. V. Benjamin, M. Mahowald, R. Douglas, G. Indiveri, R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, H. S. Seung, S.-C. Liu, T. Delbruck, J. Backus, R. J. Douglas, K. A. Martin, S. B. Laughlin, T. J. Sejnowski, V. B. Mountcastle, D. H. Hubel, T. N. Wiesel, T. Ohno, T. Hasegawa, T. Tsuruoka, K. Terabe, J. K. Gimzewski, M. Aono, M. M. Shulaker, G. Hills, N. Patil, H. Wei, H. Y. Chen, H. S. Wong, S. Mitra, D. S. Modha, R. Singh, S.-B. Paik, D. L. Ringach, L. A. Palmer, T. L. Davis, E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, S. B. Furber, P. Merolla, J. Arthur, R. Alvarez, J.-M. Bussat, K. Boahen, A. G. Andreou, and K. A. Boahen, "Artificial brains. A million spiking-neuron integrated circuit with a scalable communication network and interface." *Science (New York, N.Y.)*, vol. 345, no. 6197, pp. 668–73, 2014. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/25104385 1

[10] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2012. 1

[11] G. Indiveri, "Modeling Selective Attention Using a Neuromorphic Analog VLSI Device," *Neural Computation*, vol. 12, no. 12, 2000. [Online]. Available: http://dx.doi.org/ 10.1162/089976600300014755 1

[12] S. A. Aamir, Y. Stradmann, P. Muller, C. Pehle, A. Hartel, A. Grubl, J. Schemmel, and K. Meier, "An accelerated LIF neuronal network array for a large-scale mixed-signal neuromorphic architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4299–4312, 2018. 1

[13] K. A. Boahen, "Point-to-Point Connectivity Between Neuromorphic Chips using Address-Events," *Ieee Transactions on Circuits & Systems*, vol. 47, no. 5, pp. 416–434, 1999. [Online]. Available: https://web.stanford.edu/group/ brainsinsilicon/pdf/00{ }journ{ }IEEEtsc{ }Point.pdf 1

[14] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck, "A 240 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014. 1, 2

[15] G. Schoner and J. P. Spencer, Eds.," *Dynamic Thinking: A Primer on Dynamic Field Theory*. Oxford University Press, 2015. [Online]. Available: https://books.google.ch/books?id=iLVPAwAAQBAJ{&}printsec=frontcover{&} hl=de{&}source=gbs{ }ge{ }summary{ }r{&}cad=0 2, 3, 4

[16] Y. Sandamirskaya, "Dynamic Neural Fields as a Step Towards Cognitive Neuromorphic Architectures," *Frontiers in Neuroscience*, vol. 7, p. 276, 2013. 2

[17] J. N. P. Martel and Y. Sandamirskaya, "A neuromorphic approach for tracking using dynamic neural fields on a programmable vision-chip," in *Proceedings of the 10th International Conference on Distributed Smart Camera (ICDSC), ACM*, 2016. 2

[18] T.-H. Lin, G. N. Chinya, M. Davies, C.-K. Lin, A. Wild, and H. Wang, "Mapping spiking neural networks onto a manycore neuromorphic architecture," *ACM SIGPLAN Notices*, vol. 53, no. 4, pp. 78–89, 2018. 2

[19] Y. Sandamirskaya, S. K. U. Zibner, S. Schneegans, and G. Schoner, "Using Dynamic Field Theory to extend the em-¨ bodiment stance toward higher cognition," *New Ideas in Psychology*, vol. 31, no. 3, pp. 322–339, 2013. 3

[20] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM," *International Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, 2017. 5

DOT-VAE: DISENTANGLING ONE FACTOR AT A TIME

1. Bengio, Y. (2013, July). Deep learning of representations: Looking forward. In International conference on statistical language and speech processing (pp. 1-37). Springer, Berlin, Heidelberg.
2. Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., & Lerchner, A. (2018). Understanding disentangling in β-VAE. arXiv preprint arXiv:1804.03599.
3. Chen, R. T., Li, X., Grosse, R. B., Duvenaud, D. K. (2018). Isolating sources of disentanglement in variational autoencoders. Advances in neural information processing systems, 31.
4. Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. Advances in neural information processing systems, 29. Chicago
5. Dupont, E. (2018). Learning disentangled joint continuous and discrete representations. Advances in Neural Information Processing Systems, 31.
6. Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., & Xing, E. P. (2017, July). Toward controlled generation of text. In International conference on machine learning (pp. 1587-1596). PMLR.
7. Eastwood, C., & Williams, C. K. (2018, February). A framework for the quantitative evaluation of disentangled representations. In International Conference on Learning Representations.
8. Geirhos, R., Jacobsen, J. H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., & Wichmann, F. A. (2020). Shortcut learning in deep neural networks. Nature Machine Intelligence, 2(11), 665-673.
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.
10. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., ... & Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework.
11. Higgins, I., Amos, D., Pfau, D., Racaniere, S., Matthey, L., Rezende, D., & Lerchner, A. (2018). Towards a definition of disentangled representations. arXiv preprint arXiv:1812.02230.
12. Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.
13. Jeong, Y., & Song, H. O. (2019, May). Learning discrete and continuous factors of data via alternating disentanglement. In International Conference on Machine Learning (pp. 3091-3099). PMLR.
14. Kim, H., Mnih, A. (2018, July). Disentangling by factorising. In International Conference on Machine Learning (pp. 2649-2658). PMLR.
15. Kingma, D. P., Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.
16. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25. Chicago
17. Kumar, A., Sattigeri, P., Balakrishnan, A. (2017). Variational inference of disentangled latent concepts from unlabeled observations. arXiv preprint arXiv:1711.00848.
18. Lee, W., Kim, D., Hong, S., Lee, H. (2020, August). High-fidelity synthesis with disentangled representation. In European Conference on Computer Vision (pp. 157174). Springer, Cham.
19. Lin, Z., Thekumparampil, K., Fanti, G., & Oh, S. (2020, November). Infogan-cr and modelcentrality: Self-supervised model training and selection for disentangling gans. In International Conference on Machine Learning (pp. 6127-6139). PMLR.

20. Liu, B., Zhu, Y., Fu, Z., De Melo, G., & Elgammal, A. (2020, April). Oogan: Disentangling gan with one-hot sampling and orthogonal regularization. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 34, No. 04, pp. 4836-4843).
21. Locatello, F., Bauer, S., Lucic, M., Raetsch, G., Gelly, S., Scho¨lkopf, B., & Bachem, O. (2019, May). Challenging common assumptions in the unsupervised learning of disentangled representations. In international conference on machine learning (pp. 4114-4124). PMLR.
22. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.
23. Oord, A. V. D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499. Chicago
24. Rezende, D. J., Mohamed, S., & Wierstra, D. (2014, June). Stochastic backpropagation and approximate inference in deep generative models. In International conference on machine learning (pp. 1278-1286). PMLR. Chicago
25. Ridgeway, K., & Mozer, M. C. (2018). Learning deep disentangled embeddings with the f-statistic loss. Advances in neural information processing systems, 31.
26. Scho¨lkopf, B., Janzing, D., Peters, J., Sgouritsa, E., Zhang, K., & Mooij, J. (2012). On causal and anticausal learning. arXiv preprint arXiv:1206.6471.
27. Suter, R., Miladinovic, D., Scho¨lkopf, B., & Bauer, S. (2019, May). Robustly disentangled causal mechanisms: Validating deep representations for interventional robustness. In International Conference on Machine Learning (pp. 6056-6065). PMLR.
28. Zaidi, J., Boilard, J., Gagnon, G., & Carbonneau, M. A. (2020). Measuring disentanglement: A review of metrics. arXiv preprint arXiv:2012.09276. Chicago
29. Zhu, X., Xu, C., & Tao, D. (2020, August). Learning disentangled representations with latent variation predictability. In European Conference on Computer Vision (pp. 684-700). Springer, Cham.
30. Matthey L, Higgins I, Hassabis D, Lerchner A. dSprites: Disentanglement testing Sprites dataset. Published 2017. https://github.com/deepmind/dsprites-dataset/
31. Burgess C, Kim H. 3D Shapes Dataset. Published 2018. https://github.com/deepmind/3dshapes-dataset/
32. Liu Z, Luo P, Wang X, Tang X. Deep Learning Face Attributes in the Wild. In: Proceedings of International Conference on Computer Vision (ICCV). ; 2015.

t-CONVESN: TEMPORAL CONVOLUTION-READOUT FOR RANDOM RECURRENT NEURAL NETWORKS

1. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.
2. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
3. Lillicrap, T. P., Cownden, D., Tweed, D. B., & Akerman, C. J. (2014). Random feedback weights support learning in deep neural networks. arXiv preprint arXiv:1411.0247.
4. Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635.
5. Jaeger, H. (2001). The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148(34), 13.
6. Maass, W. (2011). Liquid state machines: motivation, theory, and applications. Computability in context: computation and logic in the real world, 275-296.
7. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

8. Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10), 1550-1560.
9. Lang, K. J., Waibel, A. H., & Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. Neural networks, 3(1), 23-43.
1. 10.
10. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
11. Evanusa, M., Shrestha, S., Patil, V. et al. Deep-Readout Random Recurrent Neural Networks for Real-World Temporal Data. SN COMPUT. SCI. 3, 222 (2022).
2. https://doi.org/10.1007/s42979-022-01118-9
12. Bagnall, A., Lines, J., Bostrom, A. et al. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Disc 31, 606–660 (2017). https://doi.org/10.1007/s10618-016-0483-9 14. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
11. Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014, June). Time series classification using multi-channels deep convolutional neural networks. In International conference on web-age information management (pp. 298-310). Springer, Cham.
12. Oord, A. V. D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.
13. Evanusa, M., Fermu¨ller, C., & Aloimonos, Y. (2020). Deep Reservoir Networks with Learned Hidden Reservoir Weights using Direct Feedback Alignment. arXiv preprint arXiv:2010.06209.
14. Sundaram, S., Sinha, D., Groth, M., Sasaki, T., & Boix, X. (2021). Symmetry Perception by Deep Networks: Inadequacy of Feed-Forward Architectures and Improvements with Recurrent Connections. arXiv preprint arXiv:2112.04162.
15. Agarap, A. F. (2018). Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375.
16. Tino, P. (2020). Dynamical Systems as Temporal Feature Spaces. J. Mach. Learn. Res., 21, 44-1.
17. Pandey, A., & Wang, D. (2019, May). TCNN: Temporal convolutional neural network for real-time speech enhancement in the time domain. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 6875-6879). IEEE.
18. Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021, July). Zero-shot text-to-image generation. In International Conference on Machine Learning (pp. 8821-8831). PMLR.
19. Tong, Z., & Tanaka, G. (2018, August). Reservoir computing with untrained convolutional neural networks for image recognition. In 2018 24th International Conference on Pattern Recognition (ICPR) (pp. 1289-1294). IEEE.
20. Bianchi, F. M., Scardapane, S., Løkse, S., & Jenssen, R. (2017). Bidirectional deepreadout echo state networks. arXiv preprint arXiv:1711.06509.
21. Gallicchio, C., & Micheli, A. (2017). Deep echo state network (deepesn): A brief survey. arXiv preprint arXiv:1712.04323.
22. Ma, Q., Shen, L., & Cottrell, G. W. (2020). DeePr-ESN: A deep projectionencoding echo-state network. Information Sciences, 511, 152-171.
23. Ma, Q., Zheng, Z., Zhuang, W., Chen, E., Wei, J., & Wang, J. (2021). Echo Memory-Augmented Network for time series classification. Neural Networks, 133, 177-192.
24. Ma, Q., Chen, E., Lin, Z., Yan, J., Yu, Z., & Ng, W. W. (2019). Convolutional multitimescale echo state network. IEEE Transactions on Cybernetics, 51(3), 16131625.
25. Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415.
26. Maat, J. R., Gianniotis, N., & Protopapas, P. (2018, July). Efficient optimization of echo state networks for time series datasets. In 2018 International Joint Conference on Neural Networks (IJCNN) (pp. 1-7). IEEE.
27. Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 2623-2631).
28. Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L . (2018). Universal transformers. arXiv preprint arXiv:1807.03819.

29. Hahn, M. (2020). Theoretical limitations of self-attention in neural sequence models. Transactions of the Association for Computational Linguistics, 8, 156-171.
30. Tran, K., Bisazza, A., & Monz, C. (2018). The importance of being recurrent for modeling hierarchical structure. arXiv preprint arXiv:1803.03585.
31. Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. arXiv preprint arXiv:1410.5401.
32. Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101.
33. Cohen, J. D., Perlstein, W. M., Braver, T. S., Nystrom, L. E., Noll, D. C., Jonides, J., & Smith, E. E. (1997). Temporal dynamics of brain activation during a working memory task. Nature, 386(6625), 604-608.
34. Bear, M. F. (1996). A synaptic basis for memory storage in the cerebral cortex. Proceedings of the National Academy of Sciences, 93(24), 13453-13459.

MAELSTROM NETWORKS

John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. Active vision. *International journal of computer vision*, 1:333–356, 1988.

Gyorgy Buzs¨ aki. Neural syntax: cell assemblies, synapsembles, and readers.´ *Neuron*, 68(3):362– 385, 2010.

Huzi Cheng and Joshua Brown. Replay as a basis for backpropagation through time in the brain. *bioRxiv*, pp. 2023–02, 2023.

Kyunghyun Cho, Bart van Merrienboer, C¸aglar Gulc¸ehre, Fethi Bougares, Holger Schwenk, and¨ Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.
1078.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

Ron Chrisley. Embodied artificial intelligence. *Artificial intelligence*, 149(1):131–150, 2003.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Gregory P Davis, Garrett E Katz, Rodolphe J Gentili, and James A Reggia. Neurolisp: High-level symbolic programming with attractor neural networks. *Neural Networks*, 146:200–219, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244, 2022.

Matthew Evanusa, Yulia Sandamirskaya, et al. Event-based attention and tracking on neuromorphic hardware. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Matthew Evanusa, Snehesh Shrestha, Vaishnavi Patil, Cornelia Fermuller, Michelle Girvan, and¨ Yiannis Aloimonos. Deep-readout random recurrent neural networks for real-world temporal data. *SN Computer Science*, 3(3):222, 2022.

Matthew S Evanusa, Vaishnavi Patil, Michelle Girvan, Joel Goodman, Cornelia Fermuller, and Yian-¨ nis Aloimonos. t-convesn: Temporal convolution-readout for random recurrent neural networks. In *International Conference on Artificial Neural Networks*, pp. 140–151. Springer, 2023.

Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, Giovanni Pezzulo, et al. Active inference and learning. *Neuroscience & Biobehavioral Reviews*, 68:862–879, 2016.

Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.

Arthur M Glenberg. What memory is for. *Behavioral and brain sciences*, 20(1):1–19, 1997.

Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory.¨ *Neural computation*, 9(8): 1735–1780, 1997.

John J Hopfield. Hopfield network. *Scholarpedia*, 2(5):1977, 2007.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Blockrecurrent transformers. *Advances in Neural Information Processing Systems*, 35:33248–33261, 2022.

Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to´ document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Wolfgang Maass. Liquid state machines: motivation, theory, and applications. *Computability in context: computation and logic in the real world*, pp. 275–296, 2011.

Luca Manneschi and Eleni Vasilaki. An alternative to backpropagation through time. *Nature Machine Intelligence*, 2(3):155–156, 2020.

Iven MY Mareels, Brian DO Anderson, Robert R Bitmead, Marc Bodson, and Shankar S Sastry. Revisiting the mit rule for adaptive control. In *Adaptive Systems in Control and Signal Processing 1986*, pp. 161–166. Elsevier, 1987.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Hubert Ramsauer, Bernhard Schafl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas¨ Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlovic, Geir Kjetil Sandve, et al. Hopfield´ networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

Andy Thomas. Memristor-based neural networks. *Journal of Physics D: Applied Physics*, 46(9): 093001, 2013.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Qingyang Wu, Zhenzhong Lan, Jing Gu, and Zhou Yu. Memformer: The memory-augmented transformer. 2020.

Edward Zagha. Shaping the cortical landscape: functions and mechanisms of top-down cortical feedback pathways. *Frontiers in Systems Neuroscience*, 14:33, 2020.

Gregorio Zlotnik and Aaron Vansintjan. Memory: An extended definition. *Frontiers in psychology*, 10:2523, 2019.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Anthony Bagnall, Aaron Bostrom, James Large, and Jason Lines. The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version, 2016.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018. URL http: //arxiv.org/abs/1803.01271.

Gyorgy Buzs¨ aki. Neural syntax: cell assemblies, synapsembles, and readers.´ *Neuron*, 68(3):362– 385, 2010.

Junyoung Chung, C¸aglar Gulc¸ehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation¨ of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Andreas K Engel, Peter Konig, Andreas K Kreiter, Thomas B Schillen, and Wolf Singer. Tem-¨ poral coding in the visual cortex: new vistas on integration in the nervous system. *Trends in neurosciences*, 15(6):218–226, 1992.

Matthew Evanusa, Snehesh Shrestha, Vaishnavi Patil, Cornelia Fermuller, Michelle Girvan, and¨ Yiannis Aloimonos. Deep-readout random recurrent neural networks for real-world temporal data. *SN Computer Science*, 3(3):222, 2022.

Matthew S Evanusa, Vaishnavi Patil, Michelle Girvan, Joel Goodman, Cornelia Fermuller, and Yian-¨ nis Aloimonos. t-convesn: Temporal convolution-readout for random recurrent neural networks. In *International Conference on Artificial Neural Networks*, pp. 140–151. Springer, 2023.

Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, Giovanni Pezzulo, et al. Active inference and learning. *Neuroscience & Biobehavioral Reviews*, 68:862–879, 2016.

Mevlana Gemici, Chia-Chun Hung, Adam Santoro, Greg Wayne, Shakir Mohamed, Danilo J Rezende, David Amos, and Timothy Lillicrap. Generative temporal models with memory. *arXiv preprint arXiv:1702.04649*, 2017.

Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266):20210068, 2022.

Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory.¨ *Neural computation*, 9(8): 1735–1780, 1997.

John J Hopfield. Hopfield network. *Scholarpedia*, 2(5):1977, 2007.

Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to´ document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Tarek Mansour et al. *Deep neural networks are lazy: on the inductive bias of deep learning*. PhD thesis, Massachusetts Institute of Technology, 2019.

Iven MY Mareels, Brian DO Anderson, Robert R Bitmead, Marc Bodson, and Shankar S Sastry. Revisiting the mit rule for adaptive control. In *Adaptive Systems in Control and Signal Processing 1986*, pp. 161–166. Elsevier, 1987.

Frank Rosenblatt et al. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, volume 55. Spartan books Washington, DC, 1962.

Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449, 2021.

Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

Rupesh Kumar Srivastava, Klaus Greff, and Jurgen Schmidhuber. Highway networks.¨ *arXiv preprint arXiv:1505.00387*, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

DEEP RESERVOIR NETWORKS WITH LEARNED HIDDEN RESERVOIR WEIGHTS USING DIRECT FEEDBACK ALIGNMENT

[1] C. A. Atencio, T. O. Sharpee, and C. E. Schreiner. Hierarchical computation in the canonical auditory cortical circuit. *Proceedings of the National Academy of Sciences*, 106(51):21894– 21899, 2009.

[2] A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

[3] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] N. Frémaux and W. Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9:85, 2016.

[5] C. Gallicchio and A. Micheli. Deep echo state network (deepesn): A brief survey. *arXiv preprint arXiv:1712.04323*, 2017.

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

[7] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[8] B. J. Lansdell, P. R. Prakash, and K. P. Kording. Learning to solve the credit assignment problem. *arXiv preprint arXiv:1906.00889*, 2019.

[9] R. Legenstein, D. Pecevski, and W. Maass. A learning theory for reward-modulated spike-timingdependent plasticity with application to biofeedback. *PLoS Comput Biol*, 4(10):e1000180, 2008.

[10] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1):1–10, 2016.

[11] M. Lukoševicius and H. Jaeger. Reservoir computing approaches to recurrent neural networkˇ training. *Computer Science Review*, 3(3):127–149, 2009.

[12] Q. Ma, L. Shen, and G. W. Cottrell. Deep-esn: A multiple projection-encoding hierarchical reservoir computing framework. *arXiv preprint arXiv:1711.05255*, 2017.

[13] W. Maass. Liquid state machines: motivation, theory, and applications. In *Computability in context: computation and logic in the real world*, pages 275–296. World Scientific, 2011.

[14] A. Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1037–1045, 2016.

[15] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical Review Letters*, 120(2):024102, 2018.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[17] Z. Tong and G. Tanaka. Reservoir computing with untrained convolutional neural networks for image recognition. In *24th International Conference on Pattern Recognition (ICPR)*, pages 1289–1294. IEEE, 2018.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

PROTOVAE: PROTOTYPICAL NETWORKS FOR UNSUPERVISED DISENTANGLEMENT

[1] Yoshua Bengio. Deep learning of representations: Looking forward, 2013. 1

[2] Chris Burgess and Hyunjik Kim. 3d shapes dataset. https://github.com/deepmind/3dshapes-dataset/, 2018. 5

[3] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae, 2018. 7

[4] Jiawei Chen, Janusz Konrad, and Prakash Ishwar. A cyclically-trained adversarial network for invariant representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 782–783, 2020. 8

[5] Ricky T. Q. Chen, Xuechen Li, Roger Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders, 2019. 7, 8

[6] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016. 8

[7] David L. Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003. 4

[8] Cian Eastwood and Christopher K. I. Williams. A framework for the quantitative evaluation of disentangled representations. In *International Conference on Learning Representations*, 2018. 6

[9] Zunlei Feng, Xinchao Wang, Chenglong Ke, An-Xiang Zeng, Dacheng Tao, and Mingli Song. Dual swap disentangling. *Advances in neural information processing systems*, 31, 2018. 8

[10] Muhammad Waleed Gondal, Manuel Wuthrich, Djordje Miladinovic, Francesco Locatello, Martin Breidt, Valentin Volchkov, Joel Akpo, Olivier Bachem, Bernhard Scholkopf,¨ and Stefan Bauer. On the transfer of inductive bias from simulation to the real world: a new disentanglement dataset. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche-Buc,´ E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 5, 7

[11] Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations, 2018. 8

[12] I. Higgins, Lo¨ıc Matthey, A. Pal, C. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017. 7

[13] Daniella Horan, Eitan Richardson, and Yair Weiss. When is unsupervised disentanglement possible? *Advances in Neural Information Processing Systems*, 34, 2021. 2, 4

[14] Qiyang Hu, Attila Szabo,´ Tiziano Portenier, Matthias Zwicker, and Paolo Favaro. Disentangling factors of variation by mixing them, 2018. 8

[15] Ananya Harsh Jha, Saket Anand, Maneesh Singh, and VSR Veeravasarapu. Disentangling factors of variation with cycleconsistent variational auto-encoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 805–820, 2018. 8

[16] Hyunjik Kim and Andriy Mnih. Disentangling by factorising, 2019. 5, 6, 8

[17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014. 2

[18] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. In *International Conference on Learning Representations*, 2018. 8

[19] Jose Lezama.´ Overcoming the disentanglement vs reconstruction trade-off via jacobian supervision. In *International Conference on Learning Representations*, 2018. 8

[20] Zinan Lin, Kiran Koshy Thekumparampil, Giulia Fanti, and Sewoong Oh. Infogan-cr and modelcentrality: Selfsupervised model training and selection for disentangling gans, 2020. 8

[21] Bingchen Liu, Yizhe Zhu, Zuohui Fu, Gerard de Melo, and Ahmed Elgammal. Oogan: Disentangling gan with one-hot sampling and orthogonal regularization, 2020. 8

[22] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang.
Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 5

[23] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Scholkopf, and Olivier¨ Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *International Conference on Machine Learning*, pages 4114–4124, 2019. 1, 6, 7

[24] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017. 5

[25] Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei Efros, and Richard Zhang. Swapping autoencoder for deep image manipulation. *Advances in Neural Information Processing Systems*, 33:7198–7211, 2020. 8

[26] Xuanchi Ren, Tao Yang, Yuwang Wang, and Wenjun Zeng. Do generative models know disentanglement? contrastive learning is all you need. *arXiv preprint arXiv:2102.10543*, 2021. 7, 8

[27] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models, 2014. 2

[28] Bernhard Schoelkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris Mooij. On causal and anticausal learning, 2012. 1, 8

[29] Raphael Suter, Dorde Miladinovic, Bernhard Sch´ olkopf, and¨ Stefan Bauer. Robustly disentangled causal mechanisms: Validating deep representations for interventional robustness, 2019. 1, 3, 8

[30] Attila Szabo,´ Qiyang Hu, Tiziano Portenier, Matthias Zwicker, and Paolo Favaro. Challenges in disentangling independent factors of variation. *arXiv preprint arXiv:1711.02245*, 2017. 8

[31] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle, 2015. 1

[32] Tao Yang, Xuanchi Ren, Yuwang Wang, Wenjun Zeng, and Nanning Zheng. Towards building a group-based unsupervised representation disentanglement framework. In *International Conference on Learning Representations*, 2021. 6, 8

[33] Julian Zaidi, Jonathan Boilard, Ghyslain Gagnon, and MarcAndre Carbonneau.´ Measuring disentanglement: A review of metrics, 2021. 6, 7

[34] Xinqi Zhu, Chang Xu, and Dacheng Tao. Learning disentangled representations with latent variation predictability, 2020. 8

[35] Xinqi Zhu, Chang Xu, and Dacheng Tao. Commutative lie group vae for disentanglement learning. In *International Conference on Machine Learning*, pages 12924– 12934. PMLR, 2021. 8

DEEP-READOUT RANDOM RECURRENT NEURAL NETWORKS FOR REAL-WORLD TEMPORAL DATA

[1] Vaswani, A. *et al.* Guyon, I. *et al.* (eds) *Attention is all you need*. (eds Guyon, I. *et al.*) *Advances in Neural Information Processing Systems*, Vol. 30 (Curran Associates, Inc., 2017).

[2] Oord, A. v. d. *et al.* Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016) .

[3] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018) .

[4] Douglas, R. J. & Martin, K. A. Recurrent neuronal circuits in the neocortex. *Current Biology* **17** (13), R496–R500 (2007) .

[5] Lukoˇseviˇcius, M. & Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review* **3** (3), 127–149 (2009) .

[6] van Bergen, R. S. & Kriegeskorte, N. Going in circles is the way forward: the role of recurrence in visual inference. *arXiv preprint arXiv:2003.12128* (2020) .

[7] Khrulkov, V., Novikov, A. & Oseledets, I. Expressive power of recurrent neural networks. *arXiv preprint arXiv:1711.00811* (2017) .

[8] Jaeger, H. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* **148** (34), 13 (2010) .

[9] Polydoros, A., Nalpantidis, L. & Kru¨ger, V. Advantages and limitations of reservoir computing on model learning for robot control. *IROS Workshop on Machine Learning in Planning and Control of Robot Motion, Hamburg, Germany* (2015) .

[10] Ma, Q. *et al.* Convolutional multitimescale echo state network. *IEEE Transactions on Cybernetics* (2019) .

[11] Zhao, Z. *et al.* Combining forward with recurrent neural networks for hourly air quality prediction in northwest of china. *Environmental Science and Pollution Research International* (2020) .

[12] Schrauwen, B., Wardermann, M., Verstraeten, D., Steil, J. J. & Stroobandt, D. Improving reservoirs using intrinsic plasticity. *Neurocomputing* **71** (7-9), 1159–1171 (2008) .

[13] Xue, F., Li, Q., Zhou, H. & Li, X. Reservoir computing with both neuronal intrinsic plasticity and multi-clustered structure. *Cognitive Computation* **9** (3), 400–410 (2017) .

[14] Inubushi, M. & Yoshimura, K. Reservoir computing beyond memorynonlinearity trade-off. *Scientific Reports* **7** (1), 1–10 (2017) .

[15] Ferreira, A. A. & Ludermir, T. B. Genetic algorithm for reservoir computing optimization. *2009 International Joint Conference on Neural Networks* 811–815 (2009) .

[16] Woodward, A. & Ikegami, T. A reservoir computing approach to image classification using coupled echo state and back-propagation neural networks. *International conference image and vision computing, Auckland, New Zealand* 543–458 (2011) .

[17] Bianchi, F. M., Scardapane, S., Løkse, S. & Jenssen, R. Bidirectional deep-readout echo state networks. *arXiv preprint arXiv:1711.06509* (2017) .

[18] Pathak, J., Hunt, B., Girvan, M., Lu, Z. & Ott, E. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical review letters* **120** (2), 024102 (2018) .

[19] Qiao, J., Li, F., Han, H. & Li, W. Growing echo-state network with multiple subreservoirs. *IEEE Transactions on Neural Networks and Learning Systems* **28** (2), 391–404 (2016) .

[20] Jeong, D.-H. & Jeong, J. In-ear eeg based attention state classification using echo state network. *Brain Sciences* **10** (6), 321 (2020) .

[21] Kostas, D., Aroca-Ouellette, S. & Rudzicz, F. Bendr: using transformers and a contrastive self-supervised learning task to learn from massive amounts of eeg data. *Frontiers in Human Neuroscience* **15** (2021) .

[22] Wang, P., Jiang, A., Liu, X., Shang, J. & Zhang, L. Lstm-based eeg classification in motor imagery tasks. *IEEE transactions on neural systems and rehabilitation engineering* **26** (11), 2086–2095 (2018) .

[23] Xing, X. *et al.* Sae+ lstm: A new framework for emotion recognition from multi-channel eeg. *Frontiers in neurorobotics* **13**, 37 (2019) .

[24] Sussillo, D. & Abbott, L. F. Generating coherent patterns of activity from chaotic neural networks. *Neuron* **63** (4), 544–557 (2009) .

[25] DePasquale, B., Cueva, C. J., Rajan, K., Escola, G. S. & Abbott, L. fullforce: A target-based method for training recurrent networks. *PloS One* **13** (2), e0191527 (2018) .

[26] Bouchacourt, F. & Buschman, T. J. A flexible model of working memory. *Neuron* **103** (1), 147–160 (2019) .

[27] Ganguli, S., Huh, D. & Sompolinsky, H. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences* **105** (48), 18970–18975 (2008) .

[28] Charles, A. S., Yin, D. & Rozell, C. J. Distributed sequence memory of multidimensional inputs in recurrent networks. *The Journal of Machine Learning Research* **18** (1), 181–217 (2017) .

[29] Charles, A. S., Yap, H. L. & Rozell, C. J. Short-term memory capacity in networks via the restricted isometry property. *Neural Computation* **26** (6), 1198–1235 (2014) .

[30] Walter, F., R¨ohrbein, F. & Knoll, A. Computation by time. *Neural Processing Letters* **44** (1), 103–124 (2016) .

[31] Izhikevich, E. M., Gally, J. A. & Edelman, G. M. Spike-timing dynamics of neuronal groups. *Cerebral Cortex* **14** (8), 933–944 (2004) .

[32] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9** (8), 1735–1780 (1997) .

[33] Ghosh-Dastidar, S. & Adeli, H. Spiking neural networks. *International Journal of Neural Systems* **19** (04), 295–308 (2009) .

[34] Werbos, P. J. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE* **78** (10), 1550–1560 (1990) .

[35] Bellec, G. *et al.* Biologically inspired alternatives to backpropagation through time for learning in recurrent neural networks. *arXiv preprint arXiv:1901.09049* (2019) .

[36] Monner, D. & Reggia, J. A. A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural Networks* **25**, 70–83 (2012) .

[37] O'Reilly, R. C. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation* **8** (5), 895–938 (1996) .

[38] Pineda, F. J. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters* **59** (19), 2229 (1987) .

[39] Maass, W. Liquid state machines: Motivation, theory, and applications 275–296 (2011) .

[40] Tino, P. Dynamical systems as temporal feature spaces. *Journal of Machine Learning Research* **21** (44), 1–42 (2020) .

[41] Jaeger, H. Discovering multiscale dynamical features with hierarchical echo state networks. Tech. Rep., Jacobs University Bremen (2007).

[42] Tong, Z. & Tanaka, G. Reservoir computing with untrained convolutional neural networks for image recognition 1289–1294 (2018) .

[43] Yildiz, I. B., Jaeger, H. & Kiebel, S. J. Re-visiting the echo state property. *Neural networks* **35**, 1–9 (2012) .

[44] Ferreira, A. A. & Ludermir, T. B. Comparing evolutionary methods for reservoir computing pre-training. *The 2011 International Joint Conference on Neural Networks* 283–290 (2011) .

[45] Chouikhi, N., Ammar, B., Rokbani, N. & Alimi, A. M. Pso-based analysis of echo state network parameters for time series forecasting. *Applied Soft Computing* **55**, 211–225 (2017) .

[46] Basterrech, S., Alba, E. & Sna˜sel, V. An experimental analysis of the echo state network initialization using the particle swarm optimization. *Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC)* 214–219 (2014) .

[47] Neofotistos, G. *et al.* Machine learning with observers predicts complex spatiotemporal behavior. *Frontiers in Physics* **7**, 24 (2019) .

[48] Bianchi, F. M., De Santis, E., Rizzi, A. & Sadeghian, A. Short-term electric load forecasting using echo state networks and pca decomposition. *IEEE Access* **3**, 1931–1943 (2015) .

[49] Jaeger, H. & Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science* **304** (5667), 78–80 (2004) .

[50] Antonelo, E. A. & Schrauwen, B. On learning navigation behaviors for small mobile robots with reservoir computing architectures. *IEEE Transactions on Neural Networks and Learning Systems* **26** (4), 763–780 (2014) .

[51] Chang, H. & Futagami, K. Convolutional reservoir computing for world models. *arXiv preprint arXiv:1907.08040* (2019) .

[52] Soures, N. & Kudithipudi, D. Deep liquid state machines with neural plasticity for video activity recognition. *Frontiers in Neuroscience* **13**, 686 (2019) .

[53] Rypma, B. & D'Esposito, M. The roles of prefrontal brain regions in components of working memory: effects of memory load and individual differences. *Proceedings of the National Academy of Sciences* **96** (11), 6558–6563 (1999) .

[54] Jensen, J. *et al.* Separate brain regions code for salience vs. valence during reward prediction in humans. *Human Brain Mapping* **28** (4), 294–302 (2007) .

[55] MacKay, D. J. & Mac Kay, D. J. *Information theory, inference and learning algorithms* (Cambridge University Press, 2003).

[56] French, R. M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* **3** (4), 128–135 (1999) .

[57] Masse, N. Y., Grant, G. D. & Freedman, D. J. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences* **115** (44), E10467–E10475 (2018) .

[58] Rikhye, R. V., Gilra, A. & Halassa, M. M. Thalamic regulation of switching between cortical representations enables cognitive flexibility. *Nature Neuroscience* **21** (12), 1753–1763 (2018) .

[59] LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86** (11), 2278– 2324 (1998) .

[60] Rumelhart, D. E., Durbin, R., Golden, R. & Chauvin, Y. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications* 1–34 (1995) .

[61] Cho, K. *et al.* Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014) .

[62] Wan, J. *et al.* Chalearn looking at people rgb-d isolated and continuous datasets for gesture recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* 56–64 (2016) .

[63] Foundation, W. *Aircraft Marshalling* (2019). URL https://en.wikipedia. org/wiki/Aircraft marshalling.

[64] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S. & Sheikh, Y. A. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019) .

[65] Murray, J. D. *et al.* A hierarchy of intrinsic timescales across primate cortex. *Nature Neuroscience* **17** (12), 1661–1663 (2014) .

[66] Koelstra, S. *et al.* Deap: A database for emotion analysis; using physiological signals. *IEEE Transactions on Affective Computing* **3** (1), 18–31 (2011) .

[67] Pan, C., Shi, C., Mu, H., Li, J. & Gao, X. Eeg-based emotion recognition using logistic regression with gaussian kernel and laplacian prior and investigation of critical frequency bands. *Applied Sciences* **10** (5), 1619 (202

ProQuest Number: 31559878

INFORMATION TO ALL USERS
The quality and completeness of this reproduction is dependent on the quality
and completeness of the copy made available to ProQuest.

Distributed by
ProQuest LLC a part of Clarivate (2025).
Copyright of the Dissertation is held by the Author unless otherwise noted.

This work is protected against unauthorized copying under Title 17,
United States Code and other applicable copyright laws.

This work may be used in accordance with the terms of the Creative Commons license
or other rights statement, as indicated in the copyright statement or in the metadata
associated with this work. Unless otherwise specified in the copyright statement
or the metadata, all rights are reserved by the copyright holder.

ProQuest LLC
789 East Eisenhower Parkway
Ann Arbor, MI 48108 USA