university of groningen

Function generation from a Sum of Oscillator Signals

an Evaluation of Algorithms

Rafael Tappe Maestro
s3258734

Master Thesis
Artificial Intelligence

Supervisors
Prof. Dr. Lambert Schomaker
Promovendus MSc D. Cipollini

University of Groningen
Netherlands
August 9, 2023

Abstract

Energy demand in data-intensive applications is an ever-growing concern. Training a recent large language model consumes energy in the order of hundreds of US households per year. Remarkably, the human brain is orders of magnitude more energy efficient than modern day digital computers. Taking inspiration from the brain, the neuromorphic paradigm aims to build more efficient computing systems using analog devices.

In this work, an ensemble of oscillators is designed and simulated with the goal of arbitrary time-series approximation. Each oscillator-neuron is formed by a vanadium dioxide memristor in series with a resistor-capacitor (RC) circuit. Three approaches to simulate an ensemble are developed using the SPICE circuit simulator. Because the propagation of gradient information through electric circuits is a hard problem, multiple gradient-free optimization algorithms are explored which perturb oscillators' frequency, gain, phase and offset. We fit a range of real-world and synthetic target functions with varying frequency bands and durations. Root Mean Squared Error (RMSE) between a target and ensemble signal is used to evaluate fit. As a benchmark algorithm we rely on linear regression which produces an analytical solution for the choice of circuit parameters.

We show that a vanadium-dioxide oscillator ensemble is suitable for function generation when the target's frequency band lies within the frequency band of the oscillator-neurons. The approximation of chirp signals is difficult, none of the real-world targets can be fit. The system benefits from broad phase and frequency diversity. In contrast, a wide dynamic range leads to exponential loss growth for most algorithms. Surprisingly, an increase in the number of oscillators tends to increase loss. Of the gradient-free algorithms, a Las Vegas and a random walk algorithm perform best; they outperform advanced algorithms such as Simulated Annealing, Basin Hopping and Differential Evolution. We achieve a RMSE of 0.02 with the Las Vegas algorithm, fitting a sine target function of amplitude 1; this is our best result. Linear regression finds a RMSE that is 13 orders of magnitude smaller.

We identify plateaus and steep steps in oscillators' frequency band as one of two hinderances to better fit. The strongest argument for this are the diminishing returns in adding oscillators to an ensemble while solving analytically. Therefore, vanadium-dioxide oscillators alone seem insufficient for realizing an arbitrary function approximator in the frequency domain. Second, we find that gradient-free algorithms would benefit from negative oscillator gains, as this is where the distribution of circuit parameters between gradient-free algorithms and linear regression primarily differs.

Contents

1 Introduction

1.1 Problem description and goals

Artificial intelligence (AI) begins in the middle of the 20th century and is kickstarted at a 1956 conference hosted by John McCarthy (Rockwell, 2017). At the time, computing was expensive beyond comparison to todays standards. And yet again we face high costs in computing. We are at a different point than we were 70 years ago. The imitation game, better known as Turing test, has podium contestants; in particular transformer models (Vaswani et al., 2017) such as BERT (Devlin et al., 2019) and GPT-3 (Brown et al., 2020). More recently, DALL · E 2 showcases the ability to transform text descriptions into images; MusicLM does the same for music (Agostinelli et al., 2023) and Dreamix introduces diffusion models as general purpose video editors (Molad et al., 2023). ChatGPT is an example of these technologies becoming readily available to developers and non-experts alike (Leiter et al., 2023). We are at a point where AI is not only a research topic anymore, but a common tool spawning a new suite of applications. As the adoption of AI grows, problems of scale and efficiency are amplified.

The hallmark deep learning (DL) models incur a high cost in wall-clock training time and energy spent. $\text{BERT}_{\text{base}}$, a 12 attention head and 110 million parameter model, is estimated to produce energy costs around 1.5 MWh (Strubell et al., 2019) training on 64 NVIDIA V100 graphics processing units (GPUs). In comparison the 2021 per capita energy consumption in the United States was around 77 MWh per annum[1]. Figure 1 shows that year over year, the size of language models grows exponentially. While BERT was born in 2018, GPT-3 boasts 3 orders of magnitude more parameters 2 years later, with energy costs scaling along. The given estimates don't include training costs for development, hyperparameter optimization or neural architecture search; considering these, multiple orders of magnitude must be added to the cost estimation. The achievements of said works are beyond fascinating. Yet, they also reveal limitations of the current Turing and von Neumann computing paradigm for AI applications. Efficient deep learning is an increasingly pressing topic not only because of increasing AI adoption but also amidst an energy crisis and globally surging awareness around sustainability.

Neuromorphic computing promises the energy efficiency that is found in biological systems. The arguments are strong; albeit silicon chip technology has made smartphones more capable than super computers of past

[1] "Energy Use per Person", n.d.

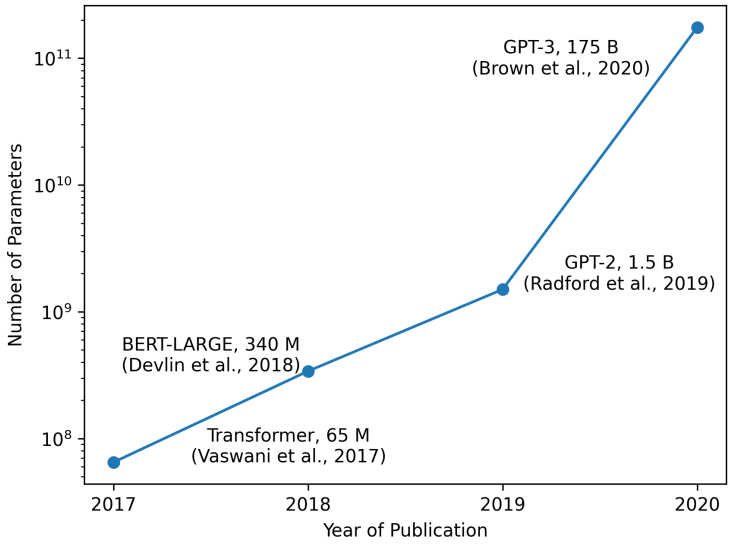


Figure 1: Size of large language models.

days, the energy efficiency at which the human brain operates while exerting general intelligence remains unparalleled by multiple orders of magnitude (Mead, 1990). Classical, digital computing and the Turing and von Neumann paradigms are unlikely to go anywhere. Yet for the purpose of some tasks, typically tackled by AI systems, the existence of a more energy efficient solution is known to be possible in the form of biological brains. The question arises, how to exploit biological principles of computation for our purposes. Devising such systems is difficult because the problem is inherently interdisciplinary and takes place on the level of algorithms and hardware. On the algorithmic level, examples are reservoir computing, spiking neural networks and local learning rules such as spike timing dependent plasticity (STDP). On the hardware level Intel's Loihi (Davies et al., 2018) and IBM's True North (Akopyan et al., 2015) demonstrate already existing neuromorphic chips.

Current AI systems successfully tackle a broad range of discrimination and generation tasks across multiple modalities, for example those mentioned in the first paragraph. In this work we focus on the problem of function approximation of one-dimensional signals. Signal approximation is the process of generating a signal that closely resembles a target signal. In audio and speech processing, signal approximation is used to compress audio signals to make their transmission and storage more efficient thus enabling technologies such as high quality music streaming and voice calls. Compression is achieved by approximating an original (target) signal with less coefficients, while still preserving important features of the original signal. Traditional

approaches to signal approximation rely on digital signal processing (DSP) methods; the established digital methods such as the Fast Fourier Transform (FFT) are already efficient for discrete one-dimensional signals. However, the time-complexity of classic methods such as the Fourier transform is exponential in the number of dimensions (Cooley & Tukey, 1965), or more precisely the tensor rank of the signal. This means that they are not well suited for the decomposition (and approximation) of high-dimensional signals. Furthermore, FFT algorithms assume discrete signals. Discrete signal representations may be prohibitively expensive in terms of memory and computation not only when dealing with high-dimensional signals but also when dealing with high frequencies signals requiring fast sampling rates.

The aim for this work is to explore the potential of a neuromorphic device based on vanadium dioxide (VO_2) for the task of signal approximation. We focus on the approximation of one-dimensional audio signals in order to test the feasibility of the proposed system. Demonstrating higher computational efficiency and lower energy consumption are out of the scope of this work, but form an overarching goal behind neuromorphic computing and this work, too.

1.2 Proposed solution

Figure 2 shows a schematic of the proposed system. The system consists of a set of N uncoupled oscillators. Each oscillator has a constant frequency, amplitude, offset and phase. Together the oscillators form an ensemble. The signals generated by the oscillators in the ensemble are summed to form a composite signal. A target signal is given as input to the system. The system is trained to produce a composite signal that is as close as possible to the target signal. The distance between the target and composite signal is measured by a loss function, here the Root Mean Square Error (RMSE). The system is trained by applying perturbations to the oscillators in the ensemble. A perturbation causes a change in an oscillator signal's parameters; its frequency, amplitude, offset and phase. A perturbation is then accepted or rejected based on the change in the loss function. The system is trained by applying perturbations until the loss function is minimized or a maximum number of perturbations is reached. The details of the training procedure vary depending on the algorithm used to train the system.

Vanadium dioxide (VO_2) is an anorganic compound whose phase transition properties make it interesting for neuromorphic applications. The material is used to produce a VO_2 memristor, a device that takes metal and insulator phases; the phase can be controlled by an externally applied voltage. In a resistor-capacitor (RC) circuit (see Figure 7), the VO_2 memristor

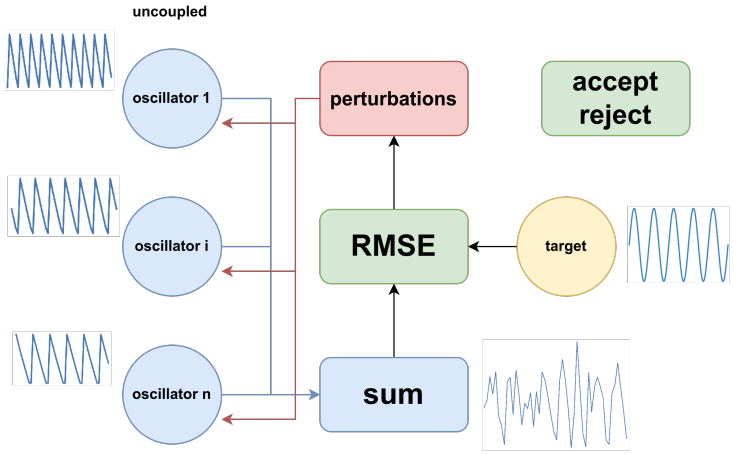


Figure 2: Oscillator ensemble.

produces high frequency oscillations of 0.1 to 1 GHz (Maffezzoni et al., 2015). The focus of this work is not on the VO_2 material itself; instead we will be working with an equivalent circuit modeled after the material's electric properties. The circuit is simulated in SPICE, a circuit simulator software. A central goal is to approximate signals that show both frequency and amplitude modulation from the ensemble of VO_2 oscillators.

In order to simplify the simulation effort, meta-programming is used to control the circuit elements. SPICE works by interpreting a netlist file, a text file that describes a circuit's elements and their properties. A general purpose programming language, here Python, is used to generate such netlists. As oscillators are uncoupled, electric interactions between them are assumed to be negligible, so that time-series generated in SPICE are further processed with Python. Once a perturbation to the circuit parameters has been computed, it's applied to the circuit by modifying the netlist file. The algorithms that control the perturbations are also implemented in Python and miss a circuit level implementation.

The perturbations of the oscillator ensemble and the resulting acceptance or rejection of changes are controlled by an optimization algorithm. Multiple optimization algorithms are explored to approximate target signals. The propagation of gradient information in electric circuits is difficult because the exact state of the system is not known at all times. Therefore backpropagation of a loss function requires a work-around; when this work-around relies on digital computing, the system is inefficient (Boon et al., 2021). For this reason stochastic optimization algorithms are explored, namely multiple Monte Carlo algorithms and a Las Vegas algorithm (Babai, 1979). Among the Monte Carlo algorithms are Simulated Annealing (Kirk-

patrick et al., 1983), Basin Hopping (Wales & Doye, 1997) and the Random Walk algorithm (Pearson, 1905). Furthermore a population based algorithm, Differential Evolution (Storn & Price, 1997), is explored. The algorithms are compared to an analytical solution which is computed with linear regression by the Ordinary Least Squares (OLS) method (Levenberg, 1944) as a benchmark.

The above algorithms have been selected with the following reasons in mind. First, they are simple. While the algorithms don't need to be simple for the digital domain, when thinking about analog circuitry, simple algorithms correspond to simple circuits. Among the chosen algorithms the Random Walk algorithm is one of the simplest with little memory requirements, in contrast to the population based Differential Evolution algorithm which maintains a population of candidate solutions. Simpler algorithms yield more freedom in designing a physical system as less control circuitry is necessary. Second, although the system is simple conceptually and observable, the individual role of each oscillator is not known throughout the training process. This means solving a black box optimization problem where only the composite signal is observed. Stochastic optimization algorithms are well suited for black box optimization problems as they make no assumptions about the system. Third, stochastic optimization algorithms are well suited to set a benchmark in optimization (Bergstra & Bengio, 2012) before advanced methods are attempted. A benchmark should be established as the VO_2 based oscillator ensemble is a new system. Lastly, random search and evolutionary algorithms have been successfully used to optimize neuromorphic systems (Schuman et al., 2022).

A range of synthetic and real world target signals are used to evaluate the performance of the system and algorithms. Synthetic signals are simple to work with because they are easy to generate and can be controlled freely. The selected synthetic signals range from simple periodic signals to more complex signals with frequency and amplitude modulation. Simple periodic target signals are sinusoids, square, triangle and sawtooth waves. More difficult is the approximation of a chirp signal, which is a sinusoid with a linearly increasing frequency. The difficulty arises because the frequency of the ensemble's oscillators is constant and the number of oscillators is finite while the frequency of the target changes continuously. More difficult, still, is the damped chirp signal, which is a chirp signal with a linearly decreasing amplitude. Real world signals have been selected to test the system's ability at speech and audio synthesis. For this purpose, a short human speech sample and the call of a magpie (Pica pica) have been selected. These signals are also difficult to approximate because they exert frequency and amplitude modulation. Furthermore, they are non-stationary, meaning that their underlying

statistical properties vary with time.

1.3 Research questions and contribution

The research questions following this approach target four levels of the developed signal approximation system. First, on the level of a single oscillator, what range of

1. frequency diversity,
2. phase diversity,
3. gain diversity,
4. offset diversity

minimizes the RMSE in target approximation? The diversity of an attribute refers to the range of values that attribute can take. We expect that an increase in an attribute's diversity will lead to a decrease in RMSE. Second, on the level of the oscillator ensemble, what is the relationship between

1. the number of oscillators,
2. the number of perturbations

on the RMSE in target approximation? We hypothesize that an increase in the number of oscillators and perturbations will lead to a decrease in RMSE. Third, on the level of targets, what is the relationship between

1. the target function,
2. the target duration,
3. the target's frequency content

on the RMSE in target approximation? We expect that more difficult target functions yield a larger RMSE. This means aperiodic target signals, sharp edges, frequency and amplitude modulation. Furthermore, we hypothesize that the duration of periodic target signals doesn't impact the RMSE. We also hypothesize that target signals which are slower than the oscillating frequency of VO_2 are more difficult to approximate. Fourth, we aim to evaluate the choice of optimization algorithm by minimal RMSE. More finely, we can distinguish algorithms by some of their properties and ask: What is the relationship between

1. the number of simultaneous perturbations,
2. the perturbed property of the oscillators, meaning frequency, phase, gain or offset or combinations thereof,
3. the candidate acceptance criterion, meaning by what criteria a change to the ensemble after perturbation is accepted or rejected

on the RMSE in target approximation? Note that some of these questions are responded to in the Appendix 10 for brevity of the main text.

Guided by the presented questions this work aims to contribute to the field of artificial intelligence. Currently, digital compute approaches tackling a range of AI problems face physical constraints, affecting both runtime and energy efficiency. One of these problems is the approximation of signals, in particular when signals are fast or high-dimensional. It's unclear how the existing obstacles can be overcome within the digital paradigm. This work relies on the field of neuromorphic computing, which exploits aspects of biological brains, to overcome said obstacles. Currently, neuromorphic computing is in a phase of exploration. VO_2 is under active investigation as a neuromorphic material, encouraging further research. Uncoupled ensembles of VO_2 oscillators for the purpose of signal approximation have not yet been demonstrated. The goal of this work is to test their capabilities. While the approach is tested in simulation, the results could inform the design of physical prototypes.

1.4 Outline

Subsequent sections of this work are structured as follows. Section 2 will present related work beginning with Section 2.1 to give a historic account of neuromorphic computing and a discussion of terminology. Following is Section 2.2, to give an idea of current neuromorphic approaches. For this purpose I cover gradient based approaches in Section 2.2.1. This is followed by Section 2.2.2 which covers reservoir computing. Next, we move on to Section 2.3, with an algorithmic view on search and optimization in a neuromorphic context. Herein, multiple stochastic optimization algorithms are covered. The related works are followed by a description of methods in Section 3. Here we begin with Section 3.1 which covers the generation of signals. Three approaches have been explored. First is a SPICE only approach, described in Section 3.1.1; this approach was abandoned due to numerical instability. Second is a Python only approach, which is described in Section 3.1.4; this approach allows for free control of oscillator signals but is a less realistic account of the VO_2 oscillators. Section 3.1.6 describes a hybrid approach that

combines the advantages of the two previous approaches. For each approach, signal generation is described on the level of a single oscillator and on the level of the ensemble. It follows a detailed description of one optimization algorithm in Section 3.2, adapted for the proposed system and an overview of all tested algorithms in Section 3.4. Lastly, Section 3.3 covers the target signals. Section 4 aggregates the parameters to the conducted experiments. Next, Section 5 lays out the results. A discussion of said results follows in Section 6. We conclude this work presenting limitations and suggestions for future investigation. The Appendix is divided into three sections. First, Section 9 holds material related to the methods and related work sections. Second, Section 10 includes additional figures and results as well as discussions thereof. Third, Section 11 contains source code and addresses a possible erratum.

2 Related work

2.1 Introduction to neuromorphic computing

This section makes an attempt at a definition of neuromorphic computing (NC), touching on history and modern day approaches. A precise definition of the term neuromorphic computing is difficult to come by (Muller et al., 2020), so we begin with an exploration of adjacent terminology to gain clarity.

Opposite to digital computing there is analog computing. One of the earliest accounts of an analog computer is the Antikythera mechanism (Freeth et al., 2006). Dated to the second century BC it was capable of predicting the position of earth, moon and sun using 37 bronze gears. Analog computing implies analogy between a modelled system and its model; analog computing also carries the notion of continuos representation and infinite precision (within the boundaries of physics). Digital computing in contrast relies on symbols[2], such as bits and bytes, instead of analogy; the available number of bits limit the precision of todays digital computers.

Unconventional computing, is a more recent term that intersects with analog computing. It captures any approaches to computing that fall outside of the modern day computer architecture devised by John von Neumann. This includes digital ternary computers, as well as computation based on spintronics, optics, DNA, fungi, fluids and chemical reactions, to name a few (Nakajima & Fischer, 2021). Not so unconventional anymore, quantum computing has been argued to have developed into a sufficiently mature field of its own (Jaeger, 2021). Gordon Pask, an English 20th century scientist, gave his unconventional computers the name maverick machines. He, for example, evolved an electrochemical ear of iron threads in a dish of ferrous sulphate solution and platinum electrodes. Using positive reinforcement, with rewards in the form of increased current, the system was capable of learning to discriminate between two frequencies (Bird & Paolo, 2008). Maverick machine is not the only alternative term given to unconventional computing. Emergent, in-materio, natural, physical and alternative computing have also been probed (Jaeger, 2021). Pask was involved with the cybernetics movement. Popular figures of cybernetics such as Stafford Beer worked alongside him building the electrochemical ear. Describing the prototype of a cyberneticist Pask wrote:

> "... we are concerned with brain-like artifacts, with evolution growth and development; with the process of thinking and getting to know about the world. Wearing the hat of applied science, we

[2]Veritasium, 2021.

> aim to create ... the instruments of a new industrial revolution - control mechanisms that lay their own plans". (Bird & Paolo, 2008)

Brain-like, a paraphrase also used by Jaeger (2021), is an apt description of neuromorphic that places it in its niche of computing. Neuromorphic captures hardware and algorithms and therefore also takes place on digital computers, for example in the form of artificial neural networks (ANN). NC distinguishes itself from the broad fields of AI and machine learning (ML), because AI and ML algorithms are not necessarily developed with brain-likeness in mind; consider the support vector machine (SVM) (Cortes & Vapnik, 1995) and k-nearest-neighbors (kNN) (Altman, 1992) algorithms.

Today's research goals for neuromorphic computing are bigger than brain-like algorithms. Through the development of hardware and algorithms alongside each other the aim is a leap over the glooming limitations of digital computing. Energy efficiency is not the only promise of neuromorphic computing but also parallelism and adaptability (Jaeger, 2021). Digital computers are serial systems first, before being parallel, and so was most interaction with them, *feed input - wait (sometimes long) - receive output - repeat*, until the appearance of personal computers, graphical user interfaces and smartphones. In contrast the brain operates in a parallel fashion, integrating the experience of multiple sensory organs at all times while exercising control over multiple muscles. The brain is also adaptable; the case of Phineas Gage, who's skull was fully penetrated by a metal rod in a work accident, demonstrates that the brain continues to function and sustain life even after severe injury (Reisberg, 2022). Pask's electrochemical ear shows adaptability in a similar fashion; conducting metal threads grow around obstacles and regrow after taking damage. Figure 3 shows one of Pask's original sketches of a similar device. A network of metal threads manifests on the layer labelled as signal network.

Research in neuromorphic computing also takes place outside of academia, for example at Intel and IBM. Intel Loihi (Davies et al., 2018) is a neuromorphic chip that relies on spiking neural networks (Tavanaei et al., 2019) to perform machine learning tasks with the advantages of asynchronous and event-based processing while consuming less energy than a digital computers for equivalent tasks. Version 1 of the chip, presented in 2018, has already been succeeded by Loihi 2 in 2022 (Frady et al., 2022). IBM Zurich focuses on three areas within neuromorphic computing, from lower to higher levels of abstraction. First, is the exploration of new materials with possibly interesting properties for neuromorphic computing, for example barium titanate (Kormondy et al., 2017) or vanadium dioxide, explored in this work. Sec-

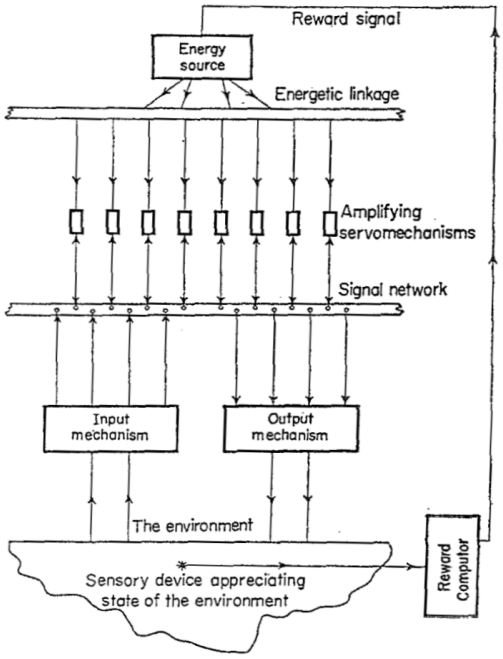


Figure 3: One of Pask's electrochemical learning machines. Figure taken from Pask (1960).

ond, are device level approaches where a material's properties are exploited to realize small scale computation; an example is demonstrating memory using barium titanate (Abel et al., 2017). Third, neuromorphic architectures integrating the device level are explored, this includes the development of learning algorithms. An example is a reservoir computing system based on barium titanate (Stark et al., 2020).

Neuromorphic computing also seems to build commercial momentum. BrainChip, founded in 2011, has a market capitalization of AU$ 1.5 billion[3] as of September 2022. The company claims that their Akida chip is the first commercial neuromorphic processor[4]. In 2022, Mercedes Benz partnered with BrainChip for the production of the Mercedes-Benz Vision EQXX concept car, where BrainChip technology handles speech recognition. Mercedes claims that the wake word "Hey Mercedes" can be evoked 5 to 10 times more efficiently compared to digital hardware[5]. PROPHESEE, another neuromorphic tech startup founded in 2014 specializes in computer vision. In collabo-

[3]"Brainchip Stock", 2022.
[4]Brainchip, 2022.
[5]Moll et al., 2022.

ration with Sony, the company developed event-based vision sensors[6]. Digital cameras typically use frame-based sensors; event-based sensors promise reduced energy consumption alongside higher temporal resolution and dynamic range. The commercial endeavours around neuromorphic computing benefit in ways as promised by neuromorphic research.

2.2 Approaches in neuromorphic computing

In this section I will discuss why gradient based learning algorithms are difficult in a neuromorphic context. Additionally I will present a non-gradient based machine learning method that is popular in neuromorphic computing, namely reservoir computing (RC). The approach to signal generation that I am taking in this work falls in neither category as it is more simple. Yet, a peek into RC, at this point, gives an idea of what kinds of algorithms are promising in neuromorphic research and why.

2.2.1 Gradient based optimization in-materio

Deep learning is built on the back of gradient descent and backpropagation algorithms. Gradient descent algorithms describe how to interpret gradients, thus, how a neural network can learn using gradient information, while the backpropagation algorithm is an efficient method for the computation of gradients (Goodfellow et al., 2016). A canonical example of gradient descent algorithms is stochastic gradient descent (SGD). Equation 1 describes the gradient estimation rule while Equation 2 describes the weight update rule; both equations are taken from Goodfellow et al. (2016).

$$\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)}) \tag{1}$$

$$\theta \leftarrow \theta - \epsilon_k \hat{g} \tag{2}$$

Where in Equation 1, ϵ is the learning rate with dimension 1, ∇_θ is gradient with respect to θ, m is the number of samples in a minibatch of samples, L is the loss function, f is the result of the forward pass or prediction, x is a sample or data point, y is the target label or ground truth, i is the index of a sample to label pair, $\hat{g}$ is the average gradient estimate over m samples;

[6]"Sony to Release Two Types of Stacked Event-Based Vision Sensors with the Industry's Smallest 4.86 Micrometer Pixel Size for Detecting Subject Changes Only Delivering High-Speed, High-Precision Data Acquisition to Improve Industrial Equipment Productivity", 2021.

it has the same dimension as x and y. And where in Equation 2, θ refers to weights between nodes, k is the index of an epoch.

Stochastic gradient descent and more generally, gradient based algorithms are not feasible to implement in a neuromorphic system. To illustrate, consider an electric circuit implementing a simple fully connected neural network with a system of resistors and amplifiers as weights. In Equation 1, the term $f(x^{(i)})$ refers to the state of our circuit; this means weights and activations of artificial neurons in the circuit. In order to calculate the derivative of the loss, exact information about the state of the system is necessary. Obtaining exact knowledge of the state of physical systems is a difficult problem (Boon et al., 2021). In our case we may ask, to what level of accuracy can we determine the voltage and current of each circuit element, the exact resistance of a resistor and the gain added by an operational amplifier? Boon et al. (2021) discuss multiple methods of measuring or calculating gradients.

Alspector et al. (1992) perturbate weights in a physical system in order to measure changes in the system's loss. A problem with this approach is that each backward pass requires multiple measurements of the system (Boon et al., 2021); the number of measurements scale at best linearly with the number of parameters in the model; as such, this method has undesirable time complexity.

Another approach for training a physical system using gradient descent is presented by Ruiz Euler et al. (2020). Direct training on the physical system is avoided by using a surrogate model. The surrogate is an artificial neural network on a digital computer trained on the input-output behavior of the physical system. The surrogate model is then further trained on a task that the physical system should solve. This process produces parameters in the model which can be transferred onto the physical system. Problematic with this approach is that the training suffers from the DL efficiency constraints that have already been laid out in Section 1.

More recently, Boon et al. (2021) present another approach for in-materio backpropagation and demonstrate successful learning of boolean functions. Their method relies on perturbations of the input signal that are reconstructed from the output. It remains to show whether the reconstruction of input signals scales to complicated real world signals, beyond boolean functions.

2.2.2 Reservoir computing

Reservoir computing (RC) systems (Lukoševičius & Jaeger, 2009; Lukoševičius et al., 2012) belong to the class of recurrent neural networks (RNN). Currently, RNNs are often associated with training by gradient descent; for

an example, consider the use of Long-Short Term Memory (LSTM) based language models (Hochreiter & Schmidhuber, 1997). A RC-system can be trained without gradient information. Furthermore, the reservoir component of such systems is typically a (somewhat) random network of neurons. RC systems have been successfully applied to a wide range of tasks such as speech and image recognition as well as time-series prediction. Both of these properties, (1) no need for gradient information and (2) network randomness, make RC systems attractive for the development of neuromorphic devices.

The details of RC are not necessary to understand the rest of this work. However a brief overview of RC is given here to give the reader a sense of the kinds of algorithms that are popular in neuromorphic computing. Figure 4 shows the basics of a RC system. On a high level, the reservoir consists of an input layer, a reservoir layer and an output layer. The input layer presents an input to the system. Typically, each input node of the reservoir is connected to one or more neurons of the reservoir. The weights of these connections are randomly initialized and are not optimized (trained). Inside the reservoir, connections between neurons are randomly initialized; these connections also typically remain constant during training. The details of neuron connectedness and random initialization are interesting and complex (Loeffler et al., 2020), but are not important for understanding the concept of RC. Similarly to the input layer, all neurons of the output layer are connected to some neurons of the reservoir. The connections between the output layer and the reservoir are also randomly initialized. During training, these connections are optimized in order to reduce some loss function. Typically, linear regression is used for this purpose, but other optimization methods are feasible.

While the RC approach is simple on a high level, the details are plenty. In particular when it comes to the development of in-materio systems, the RC landscape becomes difficult to oversee. This is because expert domain knowledge becomes necessary in multiple fields. While RC can be explained purely mathematically and is applicable on digital computers, physical RC systems will require knowledge of complex domains such as circuit design, optics and spintronics to name a few. As hinted on, the connectedness of RC systems is also an area of research in itself and can be studied from a graph theoretical perspective (Loeffler et al., 2020). For example, small-worldness has been shown to be a common property in biological neural networks (Watts & Strogatz, 1998). Furthermore, a broad class of RC systems, in particular Echo State Networks (ESN) typically assume a leaky integrator and fire (LIF) neuron model, however a variety of neuron models are possible.

Nakajima and Fischer (2021) have surveyed a broad range of neuromorphic RC systems. In their book, they present Figure 5 which illustrates the

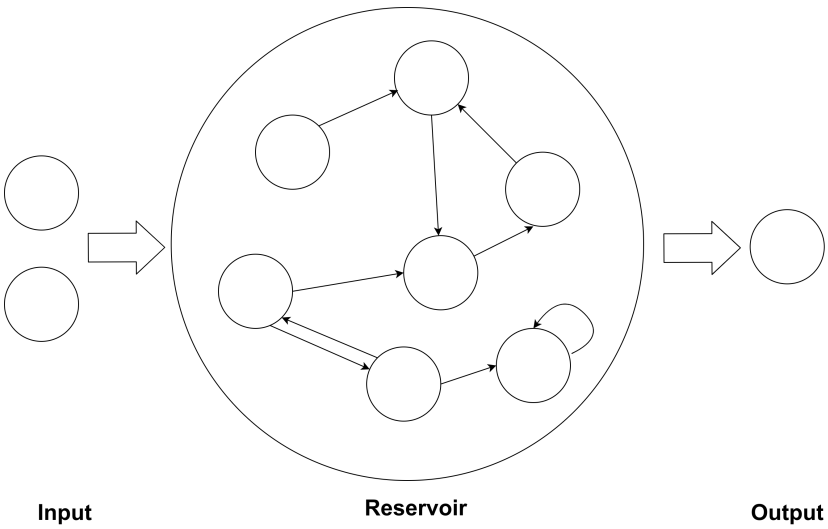


Figure 4: A simplified reservoir computing system.

interdisciplinary nature of reservoir computing. Their diagram can be generalized to the broader field of neuromorphic computing. It illustrates that the development of a neuromorphic compute system draws on methods of machine learning, materials science and mathematics of nonlinear dynamical systems, among others.

2.3 Optimization

The presented approach in this work relies on classic machine learning methods for optimization. This section explores optimization algorithms and adds a neuromorphic context. When hardware considerations are appropriate, they focus on electric circuits.

2.3.1 Monte Carlo algorithms

The most naive and trivial approach to solve an optimization problem is random search (Bergstra & Bengio, 2012). Albeit conceptually simple, as laid out in Section 1.2, random search is powerful. Random search algorithms are commonly divided into two classes, Monte Carlo and Las Vegas algorithms (Goodfellow et al., 2016). Monte Carlo algorithms use resources deterministically while the found solution contains a random amount of error. By adding compute resources, Monte Carlo methods yield output with less error. In this context, a resource concerns, for example, the run time, memory or energy consumption. In contrast, Las Vegas algorithms don't show deterministic

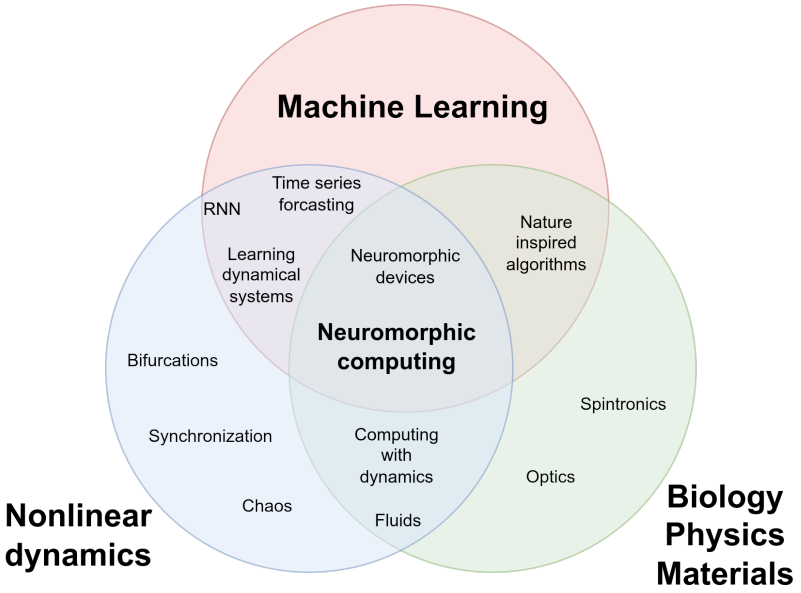


Figure 5: The neuromorphic landscape around reservoir computing. Figure adapted from Nakajima and Fischer (2021).

resource usage and produce a correct result or report failure. Correctness is up to specification and may for example be defined as an error value having obtained a value below some threshold.

In this section we focus on Monte Carlo algorithms. Monte Carlo algorithms are commonly applied for optimization and search; more formally they are a class of algorithms that sample from a distribution. In some cases, the sampled distribution is unknown and a sampling algorithm is used to estimate it. For example, consider a coin that we know is manipulated, so we don't know the exact distribution of heads and tails. We may repeatedly throw the coin (sample) and count the number of heads and tails (estimate the distribution). Now consider our method of signal approximation laid out in Section 1.2. We aim to approximate a signal by summing an ensemble of oscillator signals. The signal we aim to approximate is the unknown distribution. By repeated sampling — by repeatedly replacing the oscillators, we can estimate the signal (distribution). The ideal oscillator configuration is the point of equilibrium where the error between the target signal and the sum of the oscillators is minimized. In these terms, adding of resources corresponds to sampling more often, thus reducing the error of an approximation.

A variety of Monte Carlo algorithms exist, here we are interested in two classes of Monte Carlo samplers. First, direct sampling, where the distribution of interest is sampled directly (e.g. we toss a coin). Second, Markov

chain sampling, where the target distribution is sampled indirectly by sampling a Markov chain that converges to the target distribution.

2.3.2 Naive random search

Naive random search shown in Listing 1 is the simplest form of direct sampling Monte Carlo algorithms. Let P_X be a sampled distribution and X be the N-dimensional random variable associated with drawing from P_X. Let x^j denote the j-th element of x and x_i denote the i-th draw from X. The algorithm initializes a candidate solution x_1 with a random configuration. Throughout runtime, the algorithm stores the best solution x_{best}. Then, the algorithm iteratively samples a new candidate solution x_i. $l(x)$ is the loss function, which is minimized by the algorithm. When a new candidate solution x_i has a loss lower than the current best solution x_{best}, the algorithm updates the best solution. This process is repeated until a stopping criterion is met. A typical stopping criterion is a maximum number of iterations.

Listing 1: Naive random search

```
1  # draw(): draw a random sample from the search-space
2  # l(x): loss function
3  # crit: stopping criterion
4  # evaluate(): evaluate whether the stopping criterion is met
5
6  best = draw()
7  while not crit:
8      temp = draw()
9      if l(temp) < l(best):
10         best = temp
11     crit = evaluate()
```

As an example consider an initial draw x_1 and a subsequent draw from X with x being a 3-dimensional vector shown in Equation 2.3.2. The given values are exemplary. While the chosen values here are all distinct, a draw from X may also yield the state as a previous draw.

$$x_1 = \begin{pmatrix} 0.1 \\ -0.3 \\ 0.5 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 0.9 \\ -0.7 \\ 0.4 \end{pmatrix} \tag{3}$$

Naive random search doesn't require domain knowledge and is expected to optimize (at least to some extent) with a broad range of loss functions and distributions of the random variable X. Furthermore, the algorithm is simple to implement and doesn't require tuning of hyper parameters. Difficulty of implementation typically refers to writing code in a high-level programming

language but also translates to a neuromorphic context. Intuitively, a circuit-level implementation seems simpler, when compared with the gradient-based approaches previously discussed in Section 2.2.1. Naive random search is an uninformed (blind) search algorithm (Russell et al., 2010) as it doesn't reuse information about previous attempts or the currently best solution x_{best} to generate a new candidate solution x_i. The algorithm suffers the curse of dimensionality, meaning that its performance decreases exponentially as the dimensionality of the problem increases (Russell et al., 2010). Therefore, it is best applied to low-dimensional problems.

Naive random search is non-complete because finding the global minimum in the loss landscape is not guaranteed. This property is shared with all Monte Carlo algorithms. For continuous problems such as the one tackled in this work, the time complexity of the algorithm is difficult to estimate analytically as an exact solution is not guaranteed to exist. However a relationship between the number of iterations and the loss function can be established. This will be done empirically in Section 5. The space complexity of the algorithm is determined by the memory required to store the current best solution and the candidate solution being evaluated. The algorithm is not optimal as (1) it's not complete and (2) a solution if found is not guaranteed to be found in the least possible amount of iterations (Russell et al., 2010).

To improve the performance of the naive random search algorithm, several variations have been proposed. A well known variation is the random walk algorithm, where only one of N variables in a state x are replaced instead of replacing N variables. Another variation is the Simulated Annealing algorithm, which introduces a temperature parameter to allow for occasional loss-increasing moves to avoid getting stuck in local minima. Moreover, evolutionary algorithms, such as genetic algorithms, have also been proposed as extensions to the naive random search algorithm. For brevity Simulated Annealing and an evolutionary algorithm are discussed in the Appendix, see Section 9.1. In order to evaluate advanced Monte Carlo algorithms, naive random search can serve as a baseline.

2.3.3 Random Walk

Random walk is a variation of the naive random search algorithm. The algorithm is shown in Listing 2. In a first step, the algorithm initializes a candidate solution x_1 with a random configuration, as in naive random search. Similarly, the best candidate solution is maintained throughout search. The algorithm distinguishes itself from naive search by taking a single action at each iteration (Russell et al., 2010). For this purpose we define an action as

a single draw for x^j in the range of x^1 to x^N. While naive random search replaces all N variables in a state x with a new random value, random walk replaces only one variable in a state x. See, Equation 2.3.3 for an example where only x^3 is replaced.

$$x_1 = \begin{pmatrix} 0.1 \\ -0.3 \\ 0.5 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 0.1 \\ -0.3 \\ 0.4 \end{pmatrix} \tag{4}$$

Being similar to naive random search, random walk shares many of its properties. For this reason, the random walk is not necessarily better than naive random search; for some problems, random walk may outperform naive random search.

Listing 2: Random walk

```
# draw_full(): draw N dimensional sample from the search-space
# draw(sample): redraw one variable from the search-space
# l(x): loss function
# crit: stopping criterion
# evaluate(): evaluate whether the stopping criterion is met

best = draw_full() # initial random sample in search-space
while not crit:
    temp = draw(best)
    if l(temp) < l(best):
        best = temp
    crit = evaluate()
```

2.3.4 Las Vegas algorithms

Las Vegas algorithms constitute a broad class of algorithms. Both Las Vegas and Monte Carlo algorithms are stochastic. While Monte Carlo algorithms gamble on the quality of a solution, Las Vegas algorithms gamble resources, in particular runtime (Goodfellow et al., 2016; Luby et al., 1993). Last Vegas algorithms are described as always returning a correct solution. For a continuous optimization problem, a Las Vegas algorithm would typically find the global optimum. The notion of correctness can also be interpreted flexibly. For example then, we may define a correct solution as any solution with a loss below a certain threshold. In doing so, we find that Monte Carlo algorithms can be rephrased as Las Vegas algorithms by changing the stopping criterion.

Listing 3: Las Vegas Random Walk

```
# draw_full(): draw N dimensional sample from the search-space
```

```
2  # draw(sample): redraw one variable from the search-space
3  # l(x): loss function
4  # crit: stopping criterion
5  # evaluate(): evaluate whether the stopping criterion is met
6  # threshold # threshold for stopping criterion
7
8  def evaluate(sample):
9      if l(sample) < threshold:
10         return True
11     return False
12
13 best = draw_full() # initial random sample in search-space
14 while not crit:
15     temp = draw(best)
16     if l(temp) < l(best):
17         best = temp
18     crit = evaluate(sample)
```

Listing 3 shows pseudo code for a Las Vegas adapted random walk. Indeed, the loop is identical to the Monte Carlo random walk algorithm in Listing 2. The algorithms are only distinguished by the stopping criterion. Where, Listing 2 stops when a fixed number of iterations is reached, while Listing 3 stops when a solution with a loss below a certain threshold is found. As optimization part of the algorithm is identical, their algorithmic properties are the same. The given code snippet serves as a simple example to introduce the notion of relatedness between Las Vegas and Monte Carlo algorithms, although the example is somewhat artificial. A well known, real-world Las Vegas algorithm is quicksort (Hoare, 1962). The runtime of the algorithm depends on the selection of pivot elements[7]. The pivot element is that element around which the to be sorted array is partitioned, with smaller elements to one side and larger elements to the other.

[7] Har-Peled, 2015.

3 Methods

In this section I will describe the method to build and test the oscillator ensemble as introduced in Figure 2. The purpose of the developed system is to generate signals in order to approximate a target signal. The method can be divided into three parts. In a first part, three approaches to generate signals are developed. Second, the mechanisms to fit the generated signals to a target signal are explored; these mechanisms correspond to the application of optimization algorithms. Note that, a shorthand overview of all tested optimization algorithms is given in Section 3.4. In a third part, multiple target signals are discussed. All experiments are performed in simulation.

A high level overview of the developed system is given in Section 1.2. To summarize, the general idea is repeated here, briefly.

1. The oscillator ensemble starts out as a set of random uncoupled oscillators (random frequency, phase, gain and offset).
2. Oscillators signals are summed to form a composite signal.
3. The composite signal is compared to a target signal by RMSE.
4. A subset of oscillators is perturbed to form a candidate ensemble.
5. A composite signal is formed from the candidate ensemble and its RMSE is computed.
6. The candidate ensemble is accepted or rejected by comparing its RMSE to the RMSE of the original ensemble.
7. The process is repeated so long as resources are available.

3.1 Signal generation

In order to test the outlined hypotheses, three different approaches to signal generation are developed. Each approach presents a different set of advantages and disadvantages. We should think of the three developed approaches as primarily different on an implementation level but similar on a conceptual level. The first approach, we can call it SPICE simulation approach, is aimed at being close to a physical circuit composed of multiple VO_2 devices. The second approach, call it Python approach, is aimed at accessibility from a digital computing point of view. The third approach, from here on named hybrid approach, is a combination of the first two approaches with the aim of mitigating the disadvantages of the first two. While the SPICE section best

illustrates details of the developed system, ultimately the hybrid approach is chosen for the presentation of results in Section 5.

3.1.1 SPICE generator

The SPICE approach best illustrates the underlying method of signal generation that is shared between the three approaches. The SPICE approach is also the most accurate representation of a possible physical device and therefore serves as inspiration for the other two approaches. The approach to signal generation is best described in three parts. First, on the level of an individual VO_2 oscillator. Second, on the level of multiple VO_2 oscillators in a parallel circuit. Third, in terms of fitting the sum of multiple oscillators signals to a target signal; this last part is reserved for Section 3.2.

3.1.2 SPICE: VO_2 oscillator

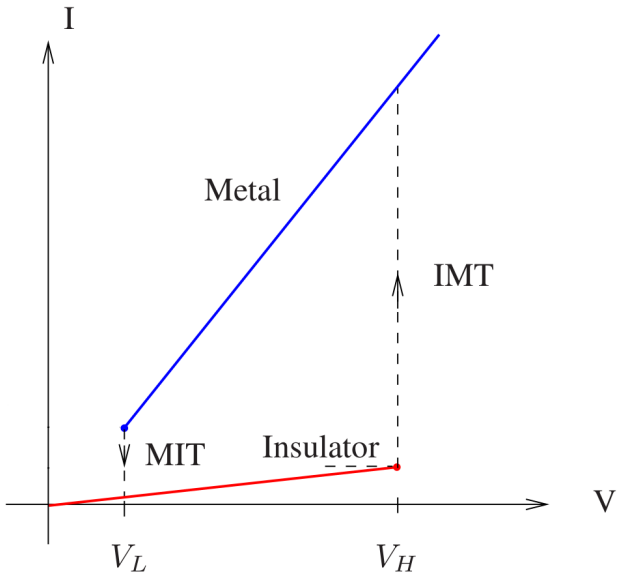


Figure 6: Phase transitions in the VO_2 device. Figure by Maffezzoni et al. (2015).

Vanadium Dioxide is a voltage gated two-phase material; its two phases are the metal phase and the insulator phase. A VO_2 device is a two terminal electric component that contains a VO_2 element; the device inherits the two-phase nature of the material. Radu et al. (2015) give an introduction to the manufacturing of VO_2 devices; further investigation on this account is left to the reader. For the purpose of this work, an equivalent circuit model of the

VO_2 device is used. The equivalent circuit is described by Maffezzoni et al. (2015).

In the metal phase, the VO_2 device is conductive while it's resistive in the insulator phase. Figure 6 describes the phase transitions in the VO_2 device. The x-axis indicates the applied voltage with arbitrary scale while the y-axis indicates current, also on an arbitrary scale; see Núñez et al. (2021) for voltage and current values. When no voltage is applied to the VO_2 device, it tends towards the insulator phase (Núñez et al., 2021). Taking the insulator state as starting point, and applying a voltage, the VO_2 device is in the high resistance state below V_L. Around the higher voltage V_H, the VO_2 device undergoes an insulator to metal transition (IMT); the transition to the low resistive state is abrupt but not instantaneous. Then, as the VO_2 device is in the metal state, the device remains conductive until the voltage is reduced to the lower voltage V_L. As applied voltage decreases from V_H to V_L, the flow of current decreases linearly. At V_L, the VO_2 device undergoes a sudden metal to insulator transition (MIT) and returns to the high resistive state.

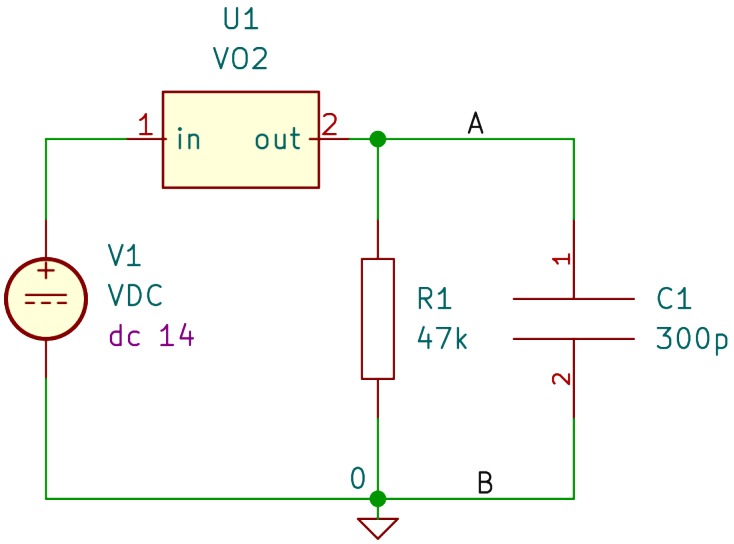


Figure 7: A VO_2 RC-circuit forms an oscillator unit (relaxation oscillator).

The phase-transition properties of the VO_2 device can be used to build VO_2 oscillators. An example of a VO_2 relaxation oscillator, inspired by Maffezzoni et al. (2015), is shown in Figure 7. The unit V1 refers to a direct current voltage source; the default voltage is set at 14 V in line with the authors. The unit R1 refers to a resistor with a default resistance of 47 kΩ. This value is chosen because the oscillatory behavior of the RC-circuit is probabilistic, and the probability of oscillation is maximal around 47 kΩ (Maffezzoni et al., 2015). The unit C1 refers to a capacitor with a

default value of 300 pF in accordance with the authors. U1 refers to the VO_2 device described in Figure 6. Variability between VO_2 devices is a problem which affects computational approaches that rely on the (phase) synchrony of multiple VO_2 oscillators such as demonstrated by Núñez et al. (2021) and Maffezzoni et al. (2015). For the purpose of this work, variability between VO_2 devices is not problematic. This is because frequency diversity is a desired property of the signal generation approach. The component that is altered for the purpose of fitting a target signal is the resistor R1. Adjusting R1 influences the oscillation frequency of the circuit at point A. In this sense R1 acts as frequency controller. Viewing the RC-circuit as an oscillatory unit, the point A serves as output terminal, where voltage modulation is observed.

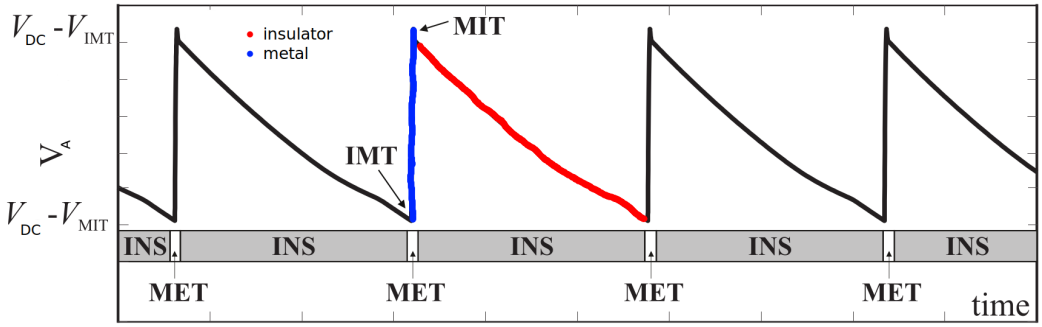


Figure 8: Voltage output of the VO_2 device at point A. Figure modified from Núñez et al. (2021).

Figure 8 shows the voltage of the VO_2 oscillator at point A in the RC-circuit. The x-axis shows time in arbitrary units [a.u.] while the y-axis shows voltage [a.u.]. V_{DC} is the voltage source, V_A is the output or oscillatory voltage; the names map to the circuit in Figure 7. V_{MIT} is the voltage at which the metal to insulator transition occurs. At V_{IMT} the insulator to metal transition occurs. As the VO_2 device enters the metal phase, the output voltage V_A sharply increases. Then, at point MIT, the VO_2 device enters the insulator phase and the output voltage of the VO_2 RC-circuit decreases at a sub-linear rate. The period of the oscillation is described by the equation (Maffezzoni et al., 2015)

$$T = \tau_{\text{met}} \cdot \ln\left(\frac{U_{\text{eq}}^{\text{met}} - V_{\text{IMT}}}{U_{\text{eq}}^{\text{met}} - V_{\text{MIT}}}\right) + \tau_{\text{ins}} \cdot \ln\left(\frac{U_{\text{eq}}^{\text{ins}} - V_{\text{IMT}}}{U_{\text{eq}}^{\text{ins}} - V_{\text{MIT}}}\right), \tag{5}$$

where,

- T is the period of the oscillation,

- τ_{met} and τ_{ins} are time constants of the metal and insulator phase respectively,
- $U_{\text{eq}}^{\text{met}}$ and $U_{\text{eq}}^{\text{ins}}$ are the voltages that capacitor C1 tends towards when the VO_2 device is in the metal or insulator phase; a more precise definition is found in the Appendix, "eq" stands for equivalency following Thévenin's equivalency theorem,
- V_{IMT} is the voltage at which the insulator to metal transition occurs in the VO_2 device,
- V_{MIT} is the voltage at which the metal to insulator transition occurs in the VO_2 device.

See Section 9.3.1 in the Appendix for a more detailed discussion of the equation's terms and a more in-depth description of the circuit's behavior.

Figure 9 shows an augmented circuit over the one shown in Figure 7. The augmented circuit adds a negative voltage source, V1_NEG that drives two operational amplifiers, U2_BUFF and U3_AMP. Op amp U2_BUFF[8] serves as a voltage buffer. It shields the oscillation generated around the label "Frequency" from the rest of the circuit. This is necessary to prevent multiple oscillators in an ensemble from influencing each other, hence making them uncoupled. The point at "Frequency" corresponds to point A in Figure 7. This makes combining multiple oscillators possible without them influencing each other. Therefore the output voltage at PIN 3 is the same as the voltage at "Frequency". Op amp U3_AMP[9] allows change in gain and offset for the oscillator generated signal. The achieved gain is given by the ratio of the resistors R1_GAIN and R2_GAIN. This relation is described in Equation 6[10].

$$\text{Gain} = 1 + \frac{R_2}{R_1} \tag{6}$$

Gains in the range of 0.1 to 10 are reasonably attainable with this circuit. The displayed values, 9 kΩ and 1 kΩ are exemplary for a 10 fold gain. During the optimization process, their values are intended to change in order to control the contribution of an oscillator to the ensemble's composite signal. The voltage offset at PIN 3 of U3_AMP is controlled by the voltage applied at PIN 1 of the op amp, labelled "Offset". When the voltage at PIN 1 is larger than that of PIN 2, the output voltage at PIN 3 is offset in a

[8] "CIRCUIT060021 Design Tool | TI.Com", n.d.
[9] "CIRCUIT060022 Design Tool | TI.Com", n.d.
[10] Sculley, n.d.

positive direction. When the voltage at PIN 1 is smaller than that of PIN 2, the voltage at PIN 3 is shifted negatively. Further, the voltage at PIN 1 is controlled by the ratio of R1_BIAS and R2_BIAS. This relationship is described in Equation 7.

$$U_{\text{OFFSET}} = V_{1\_\text{POS}} - R_{1\_\text{BIAS}} \cdot \frac{(V_{1\_\text{POS}} + V_{1\_\text{NEG}})}{(R_{1\_\text{BIAS}} + R_{2\_\text{BIAS}})} \tag{7}$$

Where,

- U_{OFFSET} is the voltage at PIN 1 of U3_AMP,
- V_{1_POS} and V_{1_NEG} are a voltage sources,
- R_{1_BIAS} and R_{2_BIAS} are resistors.

The outgoing signal of U3_AMP at PIN 3 then is a function of U_{OFFSET} and the voltage modulation at PIN 2 with the oscillation generated in the RC-circuit. The values of R1_BIAS and R2_BIAS at 1 kΩ are exemplary and are best chosen such that the oscillator's signal is centered around 0 V. R1_BIAS and R2_BIAS, as well as R1_GAIN and R2_GAIN may be replaced by potentiometers; for clarity they are shown as resistors in Figure 9.

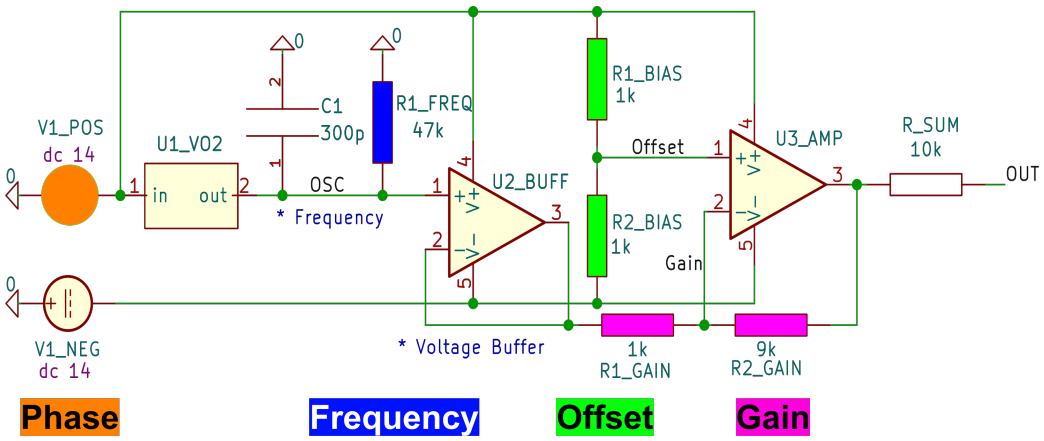


Figure 9: Control circuit for a single VO_2 oscillator.

Now we have a fully controllable oscillator. The frequency of the oscillator is controlled by the RC-circuit's time constant. The gain and offset of the oscillator are controlled by the op amps. The time constant can be adjusted by changing the value of the resistor R1_FREQ. The gain and offset can be adjusted by changing the values of the resistance pairs R_GAIN and

R_BIAS. Lastly, the phase of a single VO_2 oscillator can be controlled by delayed onset of the voltage source V1_POS or by providing the capacitor C1 with an initial charge.

3.1.3 SPICE: Oscillator Ensemble

The output voltages of multiple VO_2 oscillators can be summed in order to form a composite signal that approximates a target signal. This approach shares similarities with Fourier synthesis, see Section 9.2 in the Appendix for further elaboration. Figure 10 shows a circuit that sums the output voltages of N VO_2 oscillators forming an oscillator ensemble. The circuit is an extension of the circuit in Figure 7 by aggregating N VO_2 oscillators in parallel. For simplicity, the operational amplifiers added in Figure 9 are omitted. It's important to consider that the oscillators are uncoupled and independently controlled in their frequency, phase, gain and offset. Note that for a single oscillator, its capacitor C_i and resistor R_i are in parallel, forming a RC-circuit as shown in Figure 7. The ensemble circuit is driven by a single power source V1 for simplicity. However, powering each oscillator independently is viable and enables controlling the phase of each oscillator separately. Instead of controlling the offset of each oscillator individually as shown in Figure 9, it may be sufficient to control the offset of the sum of oscillators using a single op amp following the summation of oscillator signals.

Figure 10 misses a circuit-level representation of the comparison between composite and target signals via RMSE. Furthermore, the circuit avoids an explicit description of the mechanisms that change the resistive components which control oscillators' frequency, offset and gain in order to fit the system towards the target signal. Fully developing these components on a circuit level is beyond the scope of this work. In order to proceed, and to gauge the value of such a system, these open ends are filled by digital computing in the simulation.

Meta-programming is used to generate variations of SPICE netlists that represent the ensemble circuit. A SPICE netlist is a text file that unambiguously describes a circuit and its components. A netlist can be interpreted by the SPICE simulator, which is a circuit simulation software that can solve for the voltages and currents of a circuit. SPICE is capable of solving Maxwell's equations for the lumped network circuit model and the transmission line circuit model (C. Wong, 1994). The software is furthermore capable of transient (time-domain) analysis and frequency analysis. In particular, the transient analysis feature of SPICE is used to simulate the behavior of the circuit over time. C. Wong (1994) points out that the transient analysis feature of SPICE

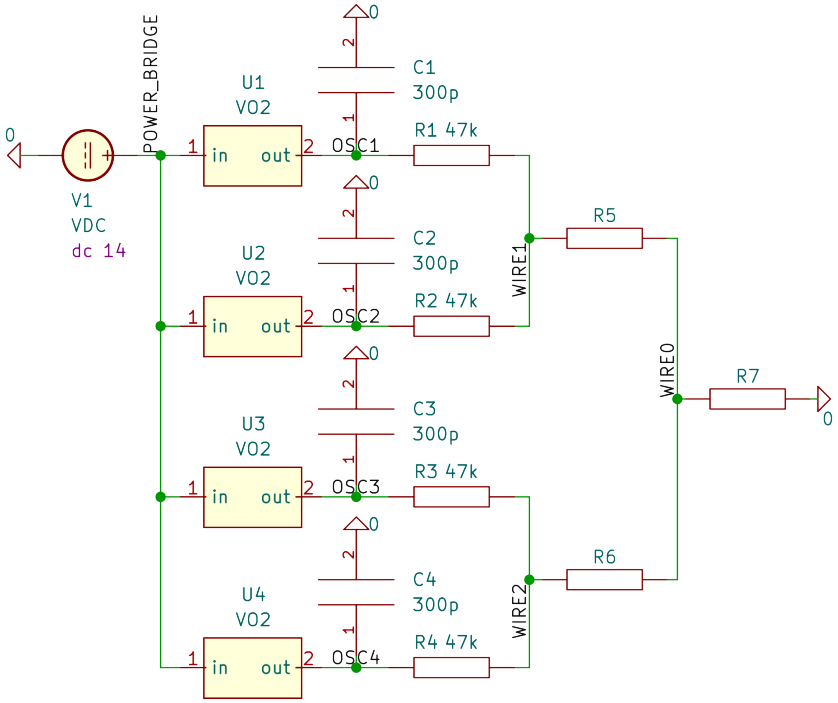


Figure 10: Ensemble of $N = 4$ VO_2 oscillators.

is inefficient due to its adaptive time-stepping algorithm. This observation is important for the development of the Python and hybrid signal generation approaches. Before fitting the circuit to a target signal, the capabilities of SPICE to simultaneously simulate increasingly large numbers of oscillators is tested.

Initial exploration of the simulation capabilities of SPICE shows that the simulation of a single oscillator is possible, but prohibitively slow. Section 9.4 in the Appendix elaborates on the details of this exploration. As a result of found limitations the Python signal generator is developed in order to proceed with simulation efforts.

3.1.4 Python generator: VO_2 oscillator

Equations 19 and 20 produced by the VO_2 device describe an exponential function. As an approximation to this function a left-skewed triangle wave, close to an inverse sawtooth function, is used for the Python signal generator. Equation 8 describes the left-skewed triangle wave for a single cycle[11].

[11] Weisstein, n.d.-b.

$$f_{\text{triangle}}(t, a, p, b) = \begin{cases} a\frac{m(t+p)}{L} + b & \text{for } 0 \leq t \leq \frac{L}{m} \\ a[1 - \frac{m}{(m-1)L}\left(t + p - \frac{L}{m}\right)] + b & \text{for } \frac{L}{m} \leq t \leq 2L - \frac{L}{m} \\ a\frac{m}{L}(t + p - 2L) + b & \text{for } 2L - \frac{L}{m} \leq t \leq 2L. \end{cases} \tag{8}$$

Where,

- a is amplitude,
- L is the duration of a period,
- p is phase shift in radians,
- b is offset,
- t is time,
- m is the skew factor.

Here, $m = 1$ yields a sawtooth, $m = 2$ yields a triangle wave, and $m \to \infty$ yields a left-skewed triangle wave approaching an inverse sawtooth. A left skewed triangle wave is preferred over the inverse sawtooth due to the steep step of the inverse sawtooth function. For convenience, the `scipy.signal.sawtooth` (Virtanen et al., 2020) function with `width=0.15` as skew factor is used.

In order to align with the op amp circuit design in Figure 9 an explicit weight term w is introduced. Throughout this work, the terms gain and weight are used interchangeably. The weight term is used to scale the amplitude, thus taking the role of the gain modifying op amp U3 in the circuit. Thus the factor a in Equation 8 corresponds to the amplitude at point "Frequency" in Figure 9, whereas $a \cdot w$ refers to the amplitude in the circuit at PIN OUT. The phase shift p corresponds to the onset time of the signal in the circuit. The offset b corresponds to the bias at PIN OUT as modulated by R1_BIAS and R2_BIAS. No additional term for offset at point "Frequency" is introduced in Equation 8 as the offset at PIN OUT is fully controlled by U3 and thus is not dependent on the bias at point "Frequency". In contrast the amplitude at PIN OUT is dependent on the amplitude at point "Frequency". The model for a single oscillator is then given by

$$f_{\text{oscillator}}(t) = w \cdot f_{\text{triangle}}(t, a, p, b) \tag{9}$$

In the Python signal generation approach frequency, phase, amplitude and offset can be directly controlled; this is opposite to the SPICE approach,

where function properties are controlled indirectly by the circuit parameters. As a downside of the Python approach, the circuit parameters corresponding to an oscillator's signal are not directly observable. An oscillator signal s is discretely represented by a row vector of length d, where d is the total number of samples, and a weight scalar w; see Equation 10.

$$s \cdot w = \begin{bmatrix} s_1 & s_2 & \dots & s_d \end{bmatrix} \cdot w \tag{10}$$

Frequency [Hz]	Weight [a.u.]	Phase [π]	Offset [a.u.]
$U(1e5, 1e6)$	$U(0, 10)$	$U(0, 2)$	0

Table 1: Default Python parameters for generating an oscillator signal.

The default set of parameters for the Python signal generator is given in Table 1. The frequency distribution is derived from SPICE simulations of the VO_2 RC-oscillator circuit. Both graphs in Figure 11 show the relationship between resistance and frequency for the VO_2 RC-oscillator circuit. On the x-axis, both plots report the resistance of the R1 element in the RC-circuit. The y-axis reports the frequency of the oscillation as a response. Whereas Maffezzoni et al. (2015) (left) report frequency for the resistance range of 30 to 65 kΩ, we (right) show a larger range of 20 to 140 kΩ. Across the shared range of 30 to 65 kΩ, Maffezzoni et al. (2015) report a frequency range from 112 to 29 kHz while we find a range of 800 to 500 kHz. For the resistance value R1=47 kΩ the frequency reported by Maffezzoni et al. (2015) is 69 kHz, whereas we produce 690 kHz. The resistance value R1=47 kΩ is particularly interesting as Maffezzoni et al. (2015) also report measurements on a physical VO_2 RC circuit with the same frequency. The left plot shows a linear relationship between resistance and frequency. Instead, we observe a moderately exponential relationship across the broad range of resistances. Note that the staircase pattern present in our figure (right) is not the result of large resistance steps, as a step size of 1 kΩ is used. It's unclear why the steps emerge in our figure. It's also unclear why we find frequency values that are an order of magnitude greater than Maffezzoni et al. (2015). The case of the authors is stronger however, as their SPICE model's frequency at 47 kΩ aligns with the results of physical measurements for a VO_2 RC circuit. Furthermore, the authors report that oscillation is probabilistic with a probability near 1 for resistances around 47 kΩ and near 0.5 around 30 kΩ and 65 kΩ. We don't observe the probabilistic nature of the VO_2 oscillation; the occurrence of oscillation appears equally probable for all resistances in the tested range. Note that the probabilistic nature of oscillation is not reported as grounded in physical measurement by Maffezzoni et al. (2015).

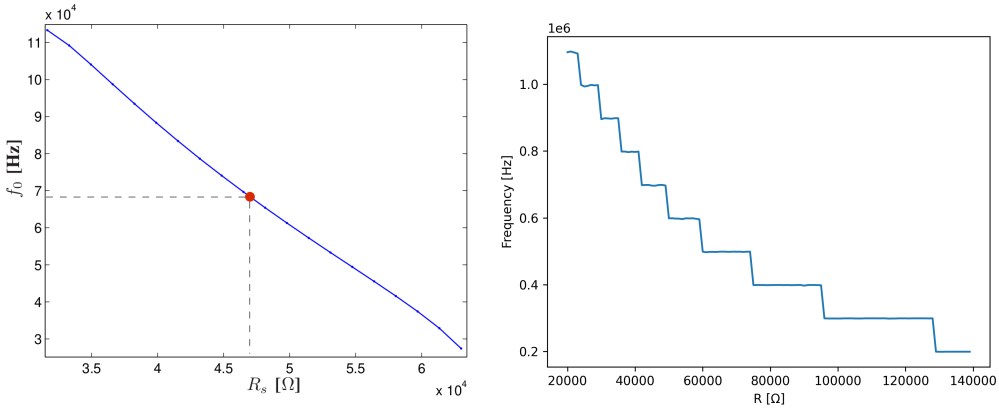


Figure 11: Frequency vs. resistance of a single VO_2 oscillator. Left, figure from Maffezzoni et al. (2015). Right, our own results.

Although the results of Maffezzoni et al. (2015) are backed by experimental results, we proceed by generating signals with frequencies matching our own findings; see Table 1. We proceed with a model of deterministically occurring oscillation, as the authors' claim in this regard is based in simulation; furthermore, it is computationally more efficient to assume oscillation is always present for simulation purposes, as zero-signals are avoided. Although we observe an exponential relationship between resistance and frequency, we assume a linear relationship as reported by Maffezzoni et al. (2015) for the sake of simplicity; this allows us to draw signals' frequencies from a uniform distribution. Weights, w, are drawn from a uniform distribution in the range of 0 to 10 [a.u.]. The chosen distribution is determined via hand-tuning as we observed that dampening (resistive only weighting) in the range of 0 to 1 is insufficient for target signal approximation. Note that this may be in part due to the small amplitude around 0.5 V of the VO_2 RC-oscillator circuit. The phase, p, is drawn from a uniform distribution in the range of 0 to 2π. This choice reflects the desire to maximally increase diversity between signals. Offset, b, is set to 0 [a.u.], as it is assumed that target signals are centered around a 0 offset, too.

3.1.5 Python generator: Oscillator ensemble

Analogous with Section 3.1.3 the sum of N signals is generated by summing the uncoupled oscillator signals along the time domain. Formally, a sum S of N oscillators is computed from an oscillator ensemble E, a weight vector W and a bias b, see Equation 11. The ensemble E has dimensions N, D, while W is of length N. Here, D refers to the number of samples in the time

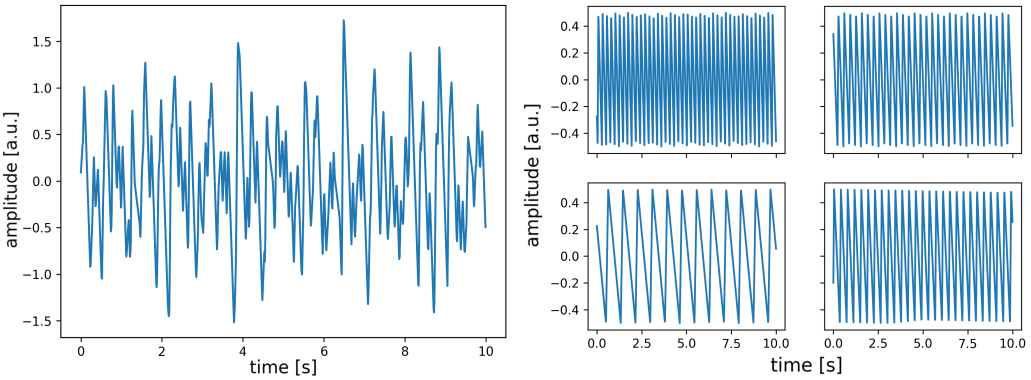


Figure 12: Sum of four oscillators (left) and individual oscillators (right).

domain. It is assumed that the individual oscillator offsets are 0 and thus the offset of the sum of signals is modulated by a scalar b.

$$S = E \cdot W + b = \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,D} \\ s_{2,1} & s_{2,2} & \dots & s_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N,1} & s_{N,2} & \dots & s_{N,D} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} + b \tag{11}$$

An example of a sum of four oscillators is given in Figure 12. On all subplots, the x-axis gives time in seconds and the y-axis reflects the amplitude of the signal in arbitrary units, albeit the amplitude is modeled after the VO_2 RC-oscillator circuit as discussed in Section 3.1.4. The sum S is the result of the sum of the four signals s_1, s_2, s_3 and s_4 shown on the right of Figure 12. The frequencies of the individual oscillators shown here are in the range of 1 to 10 Hz; this choice is for purpose of demonstration, not following the parameters in Table 1. Furthermore, the oscillators are weighted equally with a weight of 1.

A sampling rate f_s of 100 Hz is chosen for the sake of demonstration. In order for a continuous signal to be reconstructable after sampling, the sampling rate must be at least twice the maximum frequency of the signal; this sampling rate is known as Nyquist rate. When a signal is sampled at the Nyquist rate, the signal can be reconstructed without error. The reconstruction requires Whittaker-Shannon interpolation (Shannon, 1949), also known as sinus cardinals or sinc interpolation.

3.1.6 Hybrid generator

The hybrid signal generation method is a combination of the SPICE and Python methods described in the previous sections. In this approach a VO_2

oscillator circuit as shown in Figure 7 is simulated in SPICE to generate a time series. In contrast to the SPICE method, only one circuit is simulated at a time, so that oscillator signals are not summed in SPICE. Instead, the sum of the oscillator signals is computed in Python. Hereby we circumvent the limitations of SPICE when simulating multiple oscillators simulatenously. This also avoids the circuit level implementation of the impedance buffered circuit (Figure 9) with its phase, gain and offset control. Furthermore, simulation in SPICE is prohibitively slow at high sampling rates in the order of 1 GHz as needed for the VO_2 RC-oscillator circuit. In order to accelerate generating oscillator signals, caching and extrapolation are used.

3.1.7 Hybrid: VO_2 oscillator

Within the hybrid approach three modes of signal generation are available as a result of introducing caching. In order of slowest to fastest, measured in wall-clock time, we have 1) full simulation in SPICE, 2) simulation of a single period in SPICE and extrapolation of the signal in Python and 3) loading a cached SPICE period from file and extrapolation of the signal in Python. Table 2 gives an overview of the three signal generation modes and their speed. For each table entry, a $N = 100$ oscillator ensemble was simulated on an Intel i7-1065G7 CPU. The results are not averaged as runtime differences are large between modes. A duration of 1e-5 s corresponds to the period of a slow VO_2 oscillator with a resistor near 140 kΩ, in line with Figure 11 (right). The first mode, full simulation in SPICE, is the slowest and identical to the SPICE method described in Section 3.1.2. Here, a SPICE netlist is generated within Python using string manipulation. Then, the netlist is written to a file and a call to SPICE is made in order to simulate the circuit. The table shows that the duration of simulation scales with the duration of the simulated signal. Although the relationship is sub-linear, the simulation time is prohibitively long. The second mode, simulation of a single period in SPICE and extrapolation of the signal in Python, is faster than the SPICE approach for signal durations longer than a single period. Extrapolation can be used as the oscillations are periodic in the SPICE simulation. When increasing the signal duration, the simulation runtime remains nearly constant. Third, the use of caching further reduces simulation time; it is two orders of magnitudes faster compared to the extrapolation approach. In order to build the cache, RC oscillators are simulated taking resistance steps of 100 Ω, for each step a single period is simulated in SPICE and stored. Once built, the cache is loaded into Python as a resistance to signal lookup table. Then, an oscillator signal of desired duration is generated by extrapolation. While speed is increased, a tradeoff is made in accurately representing the pool of

oscillators. Figure 11 (right) shows that this tradeoff seems acceptable due to the frequency plateaus; it may be argued that the narrow steps between plateaus are underrepresented using this approach. Comparing the extrapolation and cache modes shows that spawning a SPICE process and simulating a netlist for a short duration is slow. With the first two modes, when SPICE simulation fails due to numerical instability (this occurs regularly at around each 10 oscillators) the call to SPICE is repeated with the same netlist.

Table 2: Wall-clock times of hybrid generator modes with 100 oscillators.

Duration	SPICE	Extrapolation	Cache + Extrapolation
1e-5 s	8.04 s	6.99 s	0.04 s
1e-3 s	586.03 s	7.05 s	0.07 s

After generating an oscillation, post processing is applied in Python to remove the DC offset by averaging the signal and subtracting the average. Furthermore phase and gain are added as described in Sections 3.1.4 and 3.1.5. The default hybrid parameters for generating a single oscillator signal are given in Table 3, combining parameters of the SPICE and Python approaches. The bounds of the resistance distribution are chosen far apart in order to maximize frequency diversity. Furthermore, compared to the Python parameters a larger than zero offset diversity is chosen to increase degrees of freedom in the optimization.

Resistance [kΩ]	Weight [a.u.]	Phase [π]	Offset [a.u.]
$U(20, 140)$	$U(0, 10)$	$U(0, 2)$	$U(-10, 10)$

Table 3: Default hybrid parameters for generating an oscillator signal.

3.1.8 Hybrid: Oscillator ensemble

The representation of an oscillator ensemble in the hybrid generator approach is identical to the Python approach described in Section 3.1.5. To summarize, individual oscillator signals are summed, weighted and an offset is added. The hybrid approach with caching is nearly as fast as the Python approach but more realistic with respect to the properties of the VO_2 material. Critically, the Python approach doesn't model the frequency staircase observed in Figure 11 (right). Therefore, the hybrid approach is preferred overall, an the focus for the remainder of this work.

3.2 Signal optimization towards a target

This section is concerned with describing the process of fitting the composite signal of an oscillator ensemble to approximate a target signal. Previous sections describe the process of generating oscillator signals and the formation of the ensemble signal. Generating an oscillator ensemble has been described as a random draw within the constraints of the RC-circuit in Figure 7. Optimization of the oscillator ensemble is the process of finding a set of oscillators whose sum best approximates a target signal. The optimization of an oscillator ensemble is an iterative process characterized by four steps.

1. Compute the loss between the sum of oscillators and the target signal.
2. Perturb the oscillator ensemble.
3. Recompute the loss between the oscillator sum and the target signal.
4. Accept or reject the perturbation, informed by the loss.

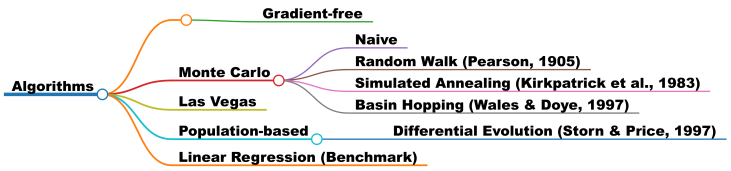


Figure 13: Explored classes of optimization algorithms.

Multiple algorithms have been implemented for the purpose of optimization. In particular non-gradient based algorithms have been selected as they are more feasible in the neuromorphic domain, eyeing the possibility of hardware implementation. Algorithms have been selected from three classes. A broad overview of these classes is given in Figure 13. First, are Monte Carlo algorithms. Of particular interest among the Monte Carlo algorithms are naive Monte Carlo search, Simulated Annealing (Kirkpatrick et al., 1983), Basin Hopping (Wales & Doye, 1997) and variants of a random walk (Franz et al., 2001; Pearson, 1905). The random walk algorithms are also inspired by Hill Climbing and Gibbs Sampling (Geman & Geman, 1984; Neal, 1993) algorithms. Second are evolutionary algorithms, by the example of Differential Evolution (Storn & Price, 1997). Third, is a variant of a Las Vegas algorithm (Babai, 1979). Lastly, linear regression by Least Squares (Levenberg, 1944) is included as an analytical baseline. The selected algorithms have already been described on a general level in Section 2.3. In this section the Las Vegas algorithm will be described in greater detail as it provides the best results

in our experiments. Compared to the general description, the pseudo code of the Las Vegas algorithm in this section is more verbose and shows details specific to the implementation in Python source code. Short descriptions of other algorithms are found in Section 3.4, furthermore all source code will be made publicly available (Tappe Maestro, 2023).

3.2.1 Las Vegas implementation

Listing 4 shows the implementation of a Las Vegas algorithm. The algorithm requires three inputs: a target signal, a maximum number of oscillators `n` in the ensemble and a maximum number of iterations `k` that the inner loop is allowed to run. The maximum number of iterations is determined according to Listing 6. This in turn requires a total number of perturbations `z` that the algorithm is allowed and a number of perturbations that the algorithm performs on each iteration. The number of perturbations that is applied per iteration is determined according to Listing 5. A perturbation is a change in the phase, frequency or gain of an oscillator. Here, a perturbation is applied to a candidate oscillator that is to be added to the ensemble. In the case of Listing 4 a single oscillator is drawn on each iteration of the inner loop, thus `r` equals 1. For the offset one perturbation is counted, as it's assumed that the offset is controlled on level of the composite ensemble signal. For the algorithm described here, all four attributes are perturbed; thus we count 4 perturbations for one iteration of the inner loop.

The objects `best`, `base` and `temp` represent ensembles. The ensemble `best` is the best ensemble found so far and `base` is the ensemble currently being optimized. Ensemble `temp` is a temporary copy that is used to store a perturbation of the `base` ensemble. The RMSE of an ensemble is accessed by the `rmse` attribute; for simplicity, no function call is shown to update the RMSE of an ensemble. Both the `best` and `base` ensemble are initialized with an empty ensemble via the function `init_empty_ensemble` such that the RMSE is equal to the RMSE between a zero signal and the target signal.

The variable `k_j` is a counter that is incremented each time the inner loop is run, it is a stopping criterion for both the inner and outer loop that evaluates to false when the maximum number of iterations is reached. The variable `i` is a counter that is incremented each time a new oscillator is accepted to the `base` ensemble. A new oscillator is accepted if the RMSE of the `temp` ensemble is less than that of the `base` ensemble. The function `add_oscillator` adds a new oscillator to the `base` ensemble at position `i` meaning that once added, an oscillator is not removed from the ensemble. The optimization of the `base` ensemble terminates when `i` is equal to the maximum number of oscillators `n` in the ensemble or when the number of

iterations `k` is exhausted. Lastly, the `base` ensemble is compared to the `best` ensemble by RMSE. The `best` ensemble is overridden if the RMSE of the `base` ensemble is lower.

Listing 4: Implementation of a Las Vegas algorithm

```
target # time series
k # maximum number of iterations
n # number of oscillator in ensemble

best = init_empty_ensemble()
k_j = 0 # number of iterations
while k_j < k:
  base = init_empty_ensemble()
  i = 0 # number of replaced weights
  while i < n and k_j < k:
    k_j += 1
    temp = add_oscillator(base, i)

    if temp.rmse < base.rmse:
      i += 1 # move to next row
      base = temp

  if base.rmse < best.rmse:
    best = base
return best
```

Listing 5: Determining the number of perturbations of an iteration

```
r # number of perturbed oscillators per iteration
properties # list of perturbed properties
perturbations = 0

if "gain" in properties:
    perturbations += r
if "frequency" in properties:
    perturbations += r
if "phase" in properties:
    perturbations += r
if "offset" in properties:
    perturbations += 1
return perturbations
```

Listing 6: Infering the maximum number of iterations

```
z # maximum number of perturbations
loop_cost # perturbations consumed by one iteration
k = z / loop_cost
return k
```

Note that while this algorithm is here on called Las Vegas, only the inner loop constitutes a Las Vegas algorithm as it's uncertain how many iterations are required to fill the ensemble. The outer loop is a Monte Carlo wrapper that allows runtime to be controlled. Also, the notion of a solution that is typically more strict for a Las Vegas algorithm is relaxed here; such that a solution is an ensemble where the addition of each oscillator has reduced the RMSE of the ensemble.

3.3 Target signals and preprocessing

Synthetic and real world target signals are tested against the oscillator ensemble. Fit between a composite signal and the target is quantitatively evaluated using Root Mean Square Error (RMSE). First we will discuss synthetic, then real world target signals.

The synthetic signals are generated using the Python libraries `numpy` and `scipy` (Virtanen et al., 2020). In order to test basic fitting capabilities, a range of synthetic periodic functions are approximated. Namely, sine, triangle, square, sawtooth, inverse sawtooth and beat signals are tested. A beat signal is formed from the interference of two sine waves with similar frequency. In order to assess the ensemble's capability of producing frequency modulation, a chirp signal serves as target. A chirp signal is a sine function with an increasing or decreasing frequency over time. To test the system's ability to additionally adapt to amplitude modulation, a damped chirp signal is used, where as frequency increases, the amplitude decreases over time. To assess the ability of fitting non-periodic functions, samples are drawn from a uniform and a normal distribution, respectively. We test signal approximation on the unmodified noise signals; additionally we apply a 10-point averager over the noise signals and also test fit on those.

We also aim to test the ensemble's ability at approximating real world signals. For this purpose an audio sample of a birdcall and a word of human speech are selected. The call of the magpie, who is among one of few animals capable of recognizing themselves in a mirror (Prior et al., 2008), is used. Since the magpie signal[12] has been used as target throughout the development of the Python signal generation pipeline, it can be considered the development sample; in line with the notion of a development set. The human speech sample has been held out until algorithms were implemented; similar to the notion of a test set. The sample of human speech articulates the word "yes"[13]. A property of human speech is that vowel regions are

[12]H F, n.d.

[13]PacDV, 2022.

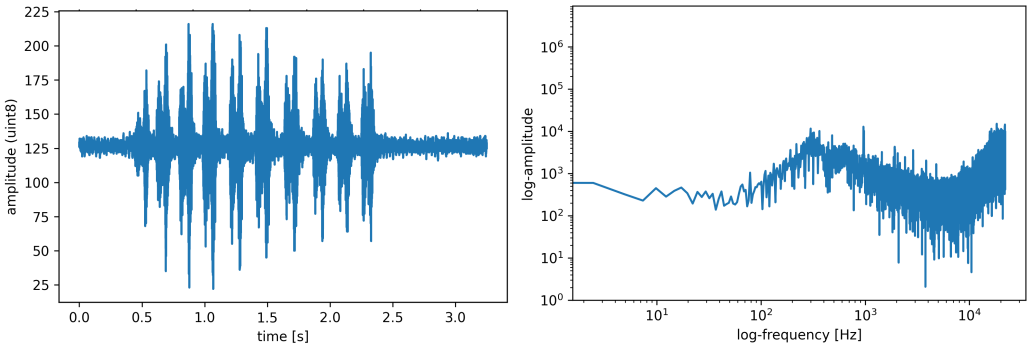


Figure 14: Magpie time-series (left) and Fourier transform (right).

nearly periodic (McClellan et al., 2017). This is likely to aid in the fitting of speech. Next to evaluation via RMSE, the fit of the real world signals is evaluated by listening to the approximated signal; this is achieved by conversion to a `.wav` audio file. To a human listener the speech audio samples are particularly interesting in this regard as humans are trained to recognize speech even under noisy conditions.

Target	Sampling frequency [Hz]	Duration [s]	Bit-dept	Channels
magpie	11025	3.25	8	1
yes	48000	0.67	16	2

Table 4: Properties of the original audio files.

A summary of the target signals' properties is given in Table 4. The parameters for the real world signals given in the table represent those of the original audio files. In order to reduce the computational cost of the experiments, the signals are downsampled to 1 kHz. Furthermore, only the center third of the magpie signal is approximated. Further preprocessing steps involve the conversion of the "yes" signal to mono and the removal of the y-offset of the magpie signal originating from it's original 8-bit encoding. Figure 14 shows the time-series and Fourier transform of the unmodified magpie recording. The magpie call shows elements of both amplitude and frequency modulation.

3.4 Algorithm shorthand

This section gives an overview of the algorithms used across experiments in Tables 5 and 6. In the name of brevity only a short description is given. Further details can be found in the publicly available code repository (Tappe

Maestro, 2023) that accompanies this thesis. The first column denotes the internal name of the algorithm, the second column gives a description. The third column describes the cost function; in this case whether only lower RMSE solutions are accepted, corresponding to greedy, or whether higher RMSE solutions are accepted too, corresponding to ergodic. The fourth column contains a reference to the algorithm or method that inspired the implementation.

The method for determining the maximum number of iterations from a fixed perturbation budget Z described in Section 3.2.1 applies to all manually implemented algorithms. Differential Evolution, Basin Hopping, SciPy Anneal and SciPy Dual Anneal are more difficult to evaluate fairly as SciPy implementations of the algorithms have been used. For Differential Evolution the maximum number of generations is computed as

$$\text{max\_iter} = \frac{Z - N}{\text{populations} \cdot N}. \tag{12}$$

Where, Z is the number of perturbations, N is the number of oscillators and populations refers to the number of candidate solutions maintained per generation of the evolutionary algorithm. We use SciPy's default value of 15 for the number of populations. The maximum number of iterations for Basin Hopping is determined as

$$\text{max\_iter} = \frac{Z}{N \cdot 20}. \tag{13}$$

Here, 20 is a hand-picked parameter that results in a run time similar to the Las Vegas and Exploit algorithms on a digital computer. For SciPy Anneal and SciPy Dual Anneal the maximum number of iterations is set to

$$\text{max\_iter} = \frac{Z}{N}. \tag{14}$$

Name	Shorthand description	Loss	Reference
One Shot	Perturbs all oscillators simultaneously. A perturbation randomly changes an oscillator's frequency, phase and gain. After a perturbation, a step is accepted when the new RMSE is lower than before. Else a new perturbation is performed. The algorithm stops when a budget of perturbations is exhausted.	Greedy	Random Search
One Shot Weight	Like One Shot, but only perturbs gain.	Greedy	Random Search
Exploit	Like One Shot, but only perturbs one oscillator before evaluating RMSE.	Greedy	Random Walk
Exploit J10	Like Exploit, but perturbs 10 oscillators simultaneously.	Greedy	Random Walk
Exploit Weight	Like Exploit, but only perturbs gain.	Greedy	Random Walk
Exploit Neighbor Weight	Like Exploit Weight, but samples a Gaussian distribution that is formed from the mean gain of a random oscillator and it's adjacent neighbors.	Greedy	Random Walk
Exploit Decoupled	Like Exploit, but randomly changes either an oscillator's gain or it's frequency and phase.	Greedy	Random Walk
Grow Shrink	Draws an ensemble of oscillators. Perturbs a single oscillator's gain at a time. Random chance of increased or decreased gain by a fixed factor. Change is accepted if RMSE is decreased.	Greedy	Random Walk, Genetic Algorithms
Dampen	Like Grow Shrink, but only dampens gain.	Greedy	Random Walk, L1 regularization (Goodfellow et al., 2016)
Purge	Like Dampen, but only sets gains to zero.	Greedy	Random Walk, Pruning methods
Oscillator Anneal	Like Exploit but uses a schedule to reduce the number of oscillators over time following a linear schedule.	Greedy	Simulated Annealing
Oscillator Anneal Weight	Like Oscillator Anneal, but only perturbs gain.	Greedy	Simulated Annealing
Oscillator Anneal Log	Like Oscillator Anneal, but the number of oscillators follows a logarithmic schedule.	Greedy	Simulated Annealing
Oscillator Anneal Log Weight	Like Oscillator Anneal Log, but only perturbs gain.	Greedy	Simulated Annealing
Differential Evolution	Population based optimization algorithm that relies on mutation, crossover and selection to minimize a fitness function.	Greedy	Storn and Price (1997)
Linear Regression	Finds optimal solution analytically by minimization of squared differences between prediction and target.	Analytical	Ordinary Least Squares linear regression

Table 5: Algorithm shorthand, first part.

Name	Shorthand description	Loss	Reference
Las Vegas	Begins with an empty ensemble and only adds an oscillator to the ensemble if RMSE is reduced. Sets each oscillator once. Then draws a new ensemble. Best ensemble is selected.	Greedy	Las Vegas
Las Vegas Weight	Similar to Las Vegas. Begins with a random best sample. Updates each oscillator's gain once.	Greedy	Las Vegas
Exploit Fast	Like Las Vegas until all oscillators have been changed at least once, then like Exploit.	Greedy	Las Vegas, Random Walk
Exploit Ergodic	Like Exploit, but accepts perturbations with larger RMSE given a small chance.	Ergodic	Metropolis-Hastings (Russell et al., 2010)
Exploit Anneal	Like Exploit Ergodic, but the acceptance probability of larger RMSE values decreases according to a temperature schedule.	Ergodic	Simulated Annealing
Exploit Anneal Weight	Like Exploit Anneal, but only perturbs gain.	Ergodic	Simulated Annealing
Basin Hopping	Similar to Simulated Annealing with a second, local, optimization step. Here Constrained Optimization BY Linear Approximation (COBYLA) is used for local optimization.	Ergodic	Basin Hopping
SciPy Anneal	Conceptually similar to Exploit Anneal Weight. Uses SciPy's implementation of Generalized Simulated Annealing.	Ergodic	Generalized Simulated Annealing
SciPy Dual Anneal	Like SciPy Anneal, but uses a local optimization step after each annealing iteration. Local optimization via the Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Bound constraints (L-BFGS-B) method.	Ergodic	Generalized Simulated Annealing

Table 6: Algorithm shorthand, second part.

4 Experimental Setup

All experiments make use of the default parameters presented in Table 3 on the oscillator level. On the ensemble level, the parameters in Table 7 are used as default. Here, N denotes the number of oscillators, Z the number of perturbations, T. the target function, amp. means amplitude, f frequency, f_s sampling rate and d. means duration. A sine of 500 kHz centered around zero with an amplitude of 1 has been chosen as default target as 500 kHz lies within the VO_2 frequency band. For each experiment, the results are averaged over L runs by default.

L	N	Z	T.	T. f	T. bias	T. amp.	T. d.	f_s
10	100	20 k	sine	500 kHz	0 [a.u.]	1 [a.u]	2e-5 s	2.5 GHz

Table 7: Default ensemble parameters.

Tables 8 and 9 list the conducted experiments. For each reported diversity value (div.) in Table 8, the parameters are drawn from a uniform distribution. The reported diversity value is the difference between the upper and lower bound of said distribution.

Name	Description and goal	U(a, b) [kΩ] a	b	div.
Resistance diversity	Oscillation frequency is controlled by resistor R_FREQ in the RC-circuit. To test whether a broader band increases fit.	40	54	14
		33	61	29
		26	68	42
		19	75	56
		19	95	76
		19	125	106
Phase diversity	Phase is controlled by the onset of the voltage source V1. To test whether a broader band increases fit.	0	0	0
		0	0.66	0.66
		0	1	1
		0	2	2
		0	4	4
Gain diversity	Gain is controlled by resistors R_GAIN via an op amp. To test whether a broader band increases fit.	0	1	1
		0	5	5
		0	10	10
		0	50	50
		0	100	100
Offset diversity	Offset is controlled by resistors R_OFFSET via an op amp. To test whether a broader band increases fit.	0	0	0
		-25	25	50
		-50	50	100
		-75	75	150
		-100	100	200

Table 8: Dependent oscillator variables for the hybrid generator experiments.

For example, resistance diversity (div.) is the difference between the upper (b) and lower bound (a) of the uniform distribution U from which RC-circuits' resistances R are sampled (see Figure 7). Resistance diversity influences the frequency band produced by the oscillator ensemble and thus influences the ability to fit target functions. The relationship between frequency and resistance is given by the RC-circuit's time constants and is described by Equation 17; this relationship is also visualized in Figure 11 (right). The lower bound doesn't extend below 19 kΩ as oscillation doesn't occur reliably for lower values. The upper bound has been chosen at twice the maximum range explored by Maffezzoni et al. (2015).

Similarly, we perform experiments where increasing amounts of phase, gain and offset diversity are sampled in order to test whether the ability to fit target functions is improved. Phase allows shifting an oscillator's signal in time. Gain or weight modulation allows dampening the contribution of oscillators not matching a target signal, and amplifying the contribution of those that do; it is also known as dynamic range. Variability in offset allows approximating target functions whose mean is not zero. More oscillators add more tuneable parameters while more perturbations add optimization iterations.

We try various target functions (see Section 3.3), frequencies and durations to test our hypotheses. For the target function experiment shown in Table 9, we test the ability to fit target functions ordered by anticipated

Name	Description and goal	Values
Number of oscillators	Number of decoupled oscillators in an ensemble. To test whether a larger number increases fit.	50, 100, 200, 500, 1000
Number of perturbations	Number of perturbations applied to an ensemble. To test whether a larger number increases fit.	0, 500, 1 k, 5 k, 10 k, 50 k
Target function	Waveform of the approximated target function. To test the ability of fit on a set of targets ordered by difficulty.	Sine, square, triangle, sawtooth, inv. sawtooth, beat, chirp, damp. chirp, uniform noise, Gaussian noise, 10-pt avg. uniform noise, 10-pt avg. Gaussian noise, "magpie", "yes"
Target duration	Duration of the approximation target function. To test if long periodic signals are approximated as well as short ones.	1e-4, 1e-3, 1e-2 s
Target frequency	Frequencies of sines and damped chirp signals. To test the ability of fit on various frequencies.	1, 10, 100, 1 k, 10 k, 100 k, 1 MHz

Table 9: Dependent ensemble variables for the hybrid generator experiments.

difficulty. For a given target function, the mean and standard deviation of the RMSE is computed over the number of L runs with each algorithm; note that this includes all algorithms listed in Section 3.4. Therefore, given 25 algorithms and 3 runs per algorithm, the mean and standard deviation is computed over 75 RMSE values for each target function. Furthermore, for the sine, triangle, sawtooth, inverse sawtooth, square wave and beat targets, signals at 4 different frequencies are tested, from 1 kHz to 1 MHz, with steps in powers of 10. Therefore, for targets with various frequencies, the mean and standard deviation is computed over $4 \cdot 75 = 300$ RMSE values. The noise targets have no frequency as they are sampled from a uniform or Gaussian distribution until the number of samples is equal to a specified duration multiplied by the sampling frequency. For the smoothened noise targets, a 10 point average is applied. In the target function experiment, the chirp and damped chirp signals use a starting frequency of 1 Hz and a stopping frequency of 1 MHz.

For the target frequency experiment sines and damped chirps are used. The frequencies of the sines are directly reported in Table 9. The damped chirps have a start and stop frequency which is set between two adjacent frequencies, for example from 1 Hz to 10 Hz, from 10 Hz to 100 Hz, etc.

Lastly, for each experiment multiple algorithms are applied. Therefore, a comparison of the algorithms workings is made linked to the results of previous experiments. In this comparison we focus on the hypotheses formulated in Section 1.3.

5 Results

In this section we present experimental results addressing the research questions introduced in Section 1.3. The focus lies on results of the hybrid signal generator pipeline for reasons discussed in Section 3.1. The figures presented in this section contain the names of algorithms as devised for the development of simulation code. The given names don't claim that the algorithms are novel; however they are kept instead of literature names as they are effective at descriptively naming algorithm variations. Furthermore while names in the plots are written in pascal case, in the text they are written apart. Additionally, the prefix "MC", short for Monte Carlo, is dropped in the text. A short-hand description of each algorithm with reference to literature is found in Section 3.4.

5.1 Summary

We find that the Las Vegas algorithm performs best across a range of experiments.

Multiple hypotheses on oscillator properties are tested. The results show that resistance diversity has a relatively small effect on the RMSE. We find that the RMSE increases alongside resistance diversity. Phase diversity has a large effect on the RMSE. The RMSE is minimal when phase shifts between 0 and 2 π are sampled. Dynamic range has a large effect on the RMSE in particular when the dynamic range is large. The RMSE increases with increasing dynamic range. No change in RMSE is observed with increasing offset diversity for a majority of tested algorithms (Section 10.2 in the Appendix).

On the ensemble level, an increase in the number of oscillators tends to increase the RMSE. The Las Vegas algorithm is not affected by an increase in the number of oscillators. An increase in the number of perturbations reduces the RMSE across stochastic algorithms.

Real world target signals cannot be approximated with the hybrid pipeline, while fitting a sine wave of 500 kHz is possible. Fitting longer durations of periodic signals doesn't increase RMSE, see Section 5.5 in the Appendix.

Algorithms perturbing one oscillator at a time perform best. No clear trend is observed regarding algorithms that perform weight-offset perturbation compared to weight-offset-frequency-phase perturbation. Greedy algorithms outperform those that probabilistically accept RMSE increasing states. See Section 10.4 in the Appendix for details.

5.2 Oscillator properties

5.2.1 Resistance diversity

Figure 15 displays resistance diversity (x-axis) against RMSE (y-axis) for a broad set of tested algorithms. The three worst performing algorithms are MCPurge, MCDampen and MCGrowShrink. The best performing algorithm is linear regression that serves as a benchmark. For algorithms in bracket B, the mean RMSE appears lowest in the 14 kΩ band and largest in the 106 kΩ band; a good example is the One Shot algorithm. For algorithms in bracket A, the mean RMSE appears lowest in the 56 kΩ band and largest in the 106 kΩ band; a good example is the Exploit Decoupled algorithm.

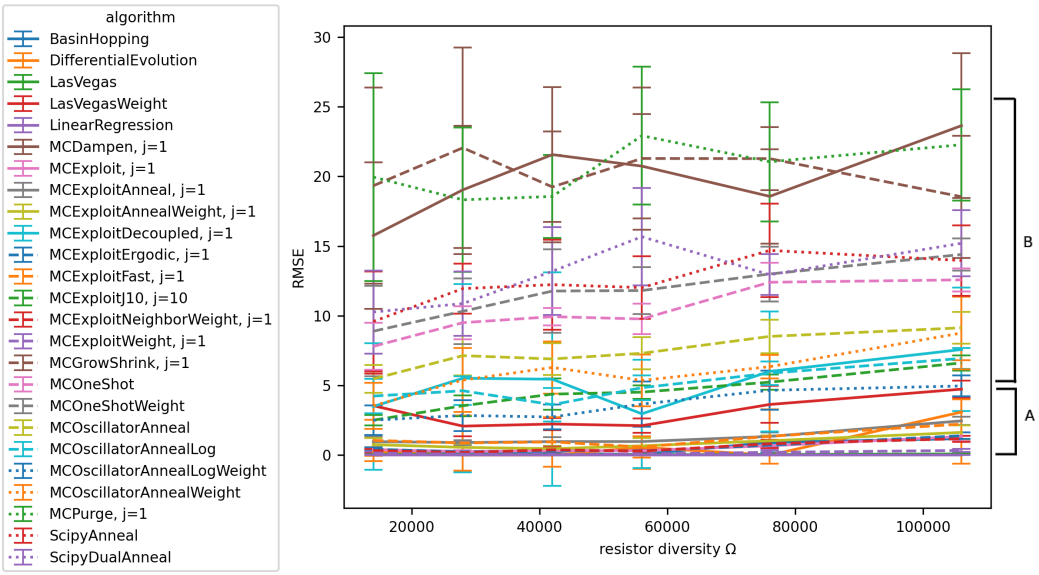


Figure 15: RMSE as a function of resistance diversity for a broad set of algorithms. Mean and standard deviation over 10 runs. See Table 10 for tabular data.

As Figure 15 is difficult to interpret, Table 10 highlights the results of the four best algorithms. In the table, B. Hop. refers to Basin Hopping. The three stochastic algorithms with lowest RMSE in the 56 kΩ band are the Las Vegas algorithm, the Exploit Weight algorithm and Basin Hopping. The Las Vegas algorithm is best across all resistance diversity bands with the exception of the 42 kΩ band. Here, Differential Evolution, not shown in the table, achieves a mean RMSE of 0.0007 with standard deviation (SD) 0.0013. In the 106 kΩ band, the Exploit-Neighbor-Weight algorithm achieves a mean RMSE, lower than Basin Hopping, of 1.18 with SD=0.22. In the 76

kΩ band, the Exploit algorithm has the third lowest mean RMSE (M) of the stochastic algorithms with M=0.66 and SD=0.27. The RMSE dip at 56 kΩ and the maximum RMSE at 106 kΩ visible in Figure 15 is also present in the tabular data.

			RMSE							
R ∼U(a, b) kΩ			Linear Regression		Las Vegas		Exploit Weight		B. Hop.	
a	b	div.	M	SD	M	SD	M	SD	M	SD
40	54	14	1.78e-15	5.94e-17	0.05	0.01	0.15	0.12	0.22	0.15
33	61	28	1.76e-15	3.54e-16	0.03	0.01	0.06	0.07	0.08	0.12
26	68	42	1.74e-15	1.67e-16	0.02	0.01	0.05	0.04	0.11	0.13
19	75	56	2.12e-15	1.32e-15	0.03	0.01	0.08	0.06	0.05	0.06
19	95	76	1.85e-15	5.43e-16	0.04	0.01	0.2	0.14	0.92	0.73
19	125	106	1.35e-15	1.96e-16	0.08	0.04	0.33	0.11	1.18	0.22

Table 10: RMSE as a function of resistance diversity with four select algorithms. See Figure 15 for the full set of algorithms.

Figure 16 displays the frequency distribution of four well-performing algorithms. We observe a frequency band from 0.2 to 1.2 MHz across algorithms in line with the VO_2 frequency band. Only Las Vegas shows frequencies below 0.2 MHz; these can be attributed to zero-frequency oscillators which are used to initialize the Las Vegas ensemble. Gaps in the frequency distribution are visible across algorithms. The probability densities are similar between algorithms, also for Las Vegas when not considering its zero-frequency oscillators.

5.2.2 Phase diversity

Table 11 shows the RMSE as a function of phase diversity for the three stochastic algorithms with the lowest RMSE and linear regression as a reference. Here, Diff. Evol. and Exploit W. denote Differential Evolution and Exploit Weight, respectively. Out of the stochastic algorithms, Differential Evolution shows the smallest mean for 0π phase diversity. Notably, the RMSE increases for Differential Evolution at 2π and 4π by one order of magnitude compared to when phase diversity is $\leq 1\pi$. The overall smallest RMSE out of the stochastic algorithms is achieved by the Las Vegas algorithm at 2π phase diversity. The RMSE doesn't further decrease beyond 2π phase diversity. The effect of increased phase diversity is least pronounced in the Exploit Weight algorithm. See Section 10.1 in the Appendix for the full set of algorithms.

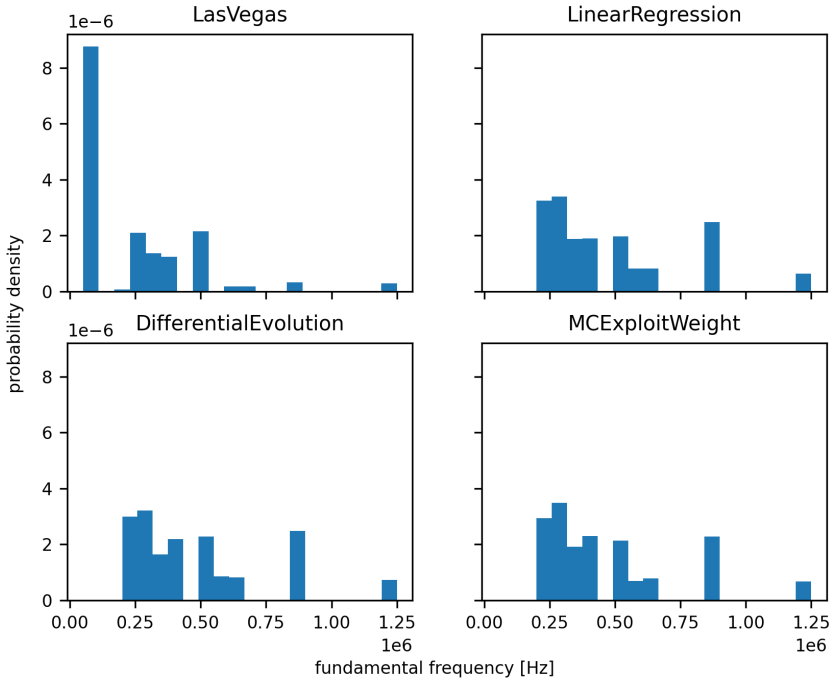


Figure 16: Frequency distributions after optimization. Average over 10 runs.

	RMSE							
	Linear Regression		Las Vegas		Diff. Evol.		Exploit W.	
Phase div.	M	SD	M	SD	M	SD	M	SD
0.00	3.44E-10	3.40E-10	0.47	0.03	0.39	0.01	0.86	0.07
0.67	7.88E-14	8.70E-14	0.52	0.07	0.40	0.01	0.74	0.07
1.00	7.38E-15	1.06E-14	0.44	0.05	0.35	0.02	0.62	0.04
2.00	1.35E-15	1.25E-16	0.10	0.05	5.72	3.78	0.52	0.12
4.00	1.62E-15	5.30E-16	0.11	0.03	3.58	3.59	0.55	0.13

Table 11: RMSE as a function of phase diversity with four select algorithms. See Figure 26 for the full set of algorithms.

5.2.3 Dynamic range

Figure 17 shows the RMSE as a function of dynamic range (gain or weight diversity) for a broad set of algorithms. Dynamic range is shown on a logarithmic scale whereas RMSE is on a linear scale. The RMSE increases with an increase in dynamic range for all stochastic algorithms. We can identify three brackets of algorithms, A, B and C. Algorithms in bracket C respond strongest to an increase in dynamic range. The RMSE is response is exponential for both the B and C brackets. The exponential response is stronger in bracket C. The Exploit Decoupled algorithm stands out from the C bracket as it shows a decrease in RMSE between a dynamic range of 50 and 100. An outlier in bracket B is Basin Hopping which shows a steeper increase in RMSE similar to bracket C. Bracket A is formed by Las Vegas and linear regression which respond sub-linearly.

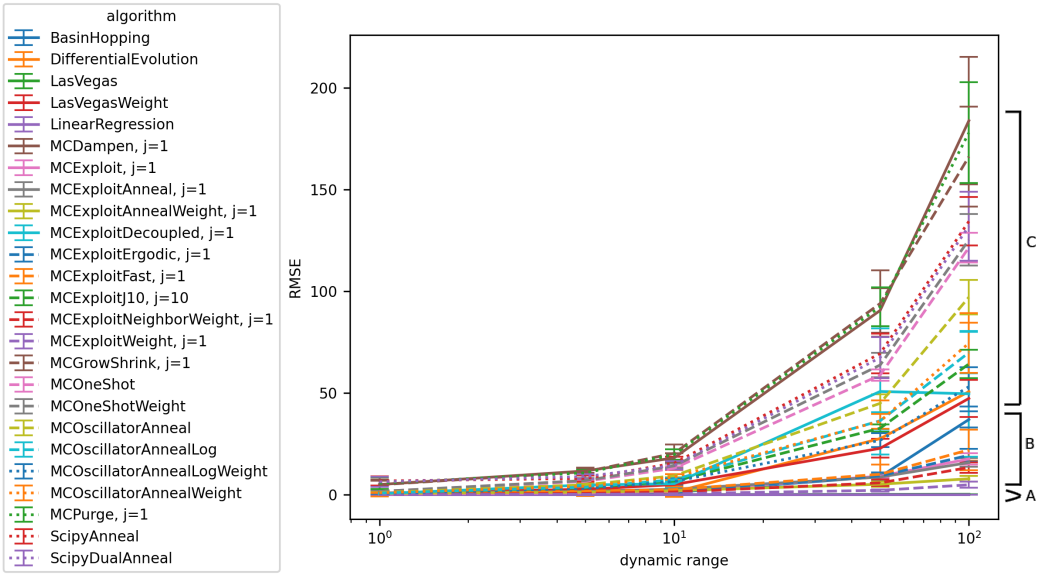


Figure 17: RMSE as a function of dynamic range for a broad set of algorithms. Mean and standard deviation over 10 runs. See Table 12 for tabular data.

Table 12 shows RMSE as a function of dynamic range for the three stochastic algorithms with the lowest RMSE and linear regression. In the table Exploit N. W. refers to the Exploit Neighbor Weight algorithm. The Las Vegas algorithm shows the lowest RMSE across all dynamic ranges among the stochastic algorithms. The distance between Las Vegas and the other algorithms increases as dynamic range increases. A hundredfold increase in dynamic range increases the RMSE of the Las Vegas algorithm by a factor

of four. In comparison, the RMSE of Exploit Weight increases by a factor of 41.9.

	RMSE							
	Linear Regression		Las Vegas		Exploit W.		Exploit N. W.	
Dyn. range	M	SD	M	SD	M	SD	M	SD
1	1.33E-15	2.48E-16	0.07	0.04	0.10	0.05	0.17	0.03
5	1.54E-15	5.95E-16	0.08	0.03	0.23	0.08	0.65	0.12
10	1.57E-15	5.01E-16	0.10	0.03	0.52	0.19	1.39	0.15
50	1.56E-15	4.46E-16	0.21	0.06	2.19	0.54	5.86	1.62
100	1.62E-15	5.3E-16	0.28	0.08	4.91	1.50	13.42	2.60

Table 12: RMSE as a function of dynamic range (dyn. range) with four select algorithms. See Figure 17 for the full set of algorithms.

Figure 18 shows the distribution of gain for the algorithms listed in Table 12. The distribution of gain is shown for a default dynamic range of 10 [a.u.] In each of the subfigures, probability density is shown on a logarithmic y-axis with gain on the x-axis. Linear regression shows a probability density near 1000 around 0 gain; positive and negative gains are observed. The distribution is not entirely symmetric, appearing heavier on the positive side of the x-axis. Both Las Vegas and Exploit Weight display no negative gains, as per their respective distributions. With Las Vegas the probability density is maximal around 0 gain; the probability density exponentially decreases and approaches zero as the gain approaches 2.5. The probability density around zero of Las Vegas is in the same order of magnitude as that of linear regression. The probability densities of Exploit Weight and Exploit Neighbor Weight are smaller by two and three orders of magnitude, respectively. Exploit Weight observes a decrease in probability density up to 2.5, from there on the density remains constant. Lastly, Exploit Neighbor Weight produces the flattest distribution with a mode around 2 gain instead of 0; we observe negative weights. The decay of probability to the left tail is more linear and steep compared to the right tail.

5.3 Ensemble properties

5.3.1 Number of oscillators

Figure 19 shows the RMSE as a function of the number of oscillators in the ensemble. Where discernible in this figure, the error grows with the number of oscillators across all algorithms. We observe two brackets in terms of algorithms' RMSE response. RMSE grows more rapidly in bracket B.

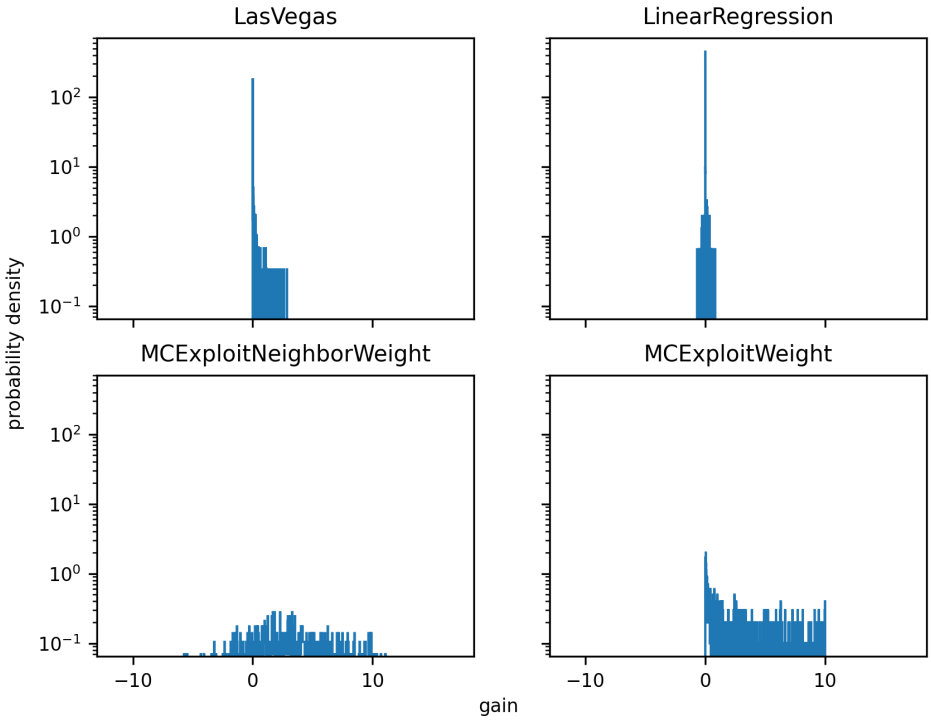


Figure 18: Weight distributions after optimization. Average over 10 runs.

Notably, Differential Evolution and Basin Hopping are part of group B although both algorithms start out with a RMSE < 1 for $N = 50$ oscillators. The Basin Hopping algorithm stands out in particular as its response curve is exponential while other algorithms in the bracket show a sub-linear response. Algorithms in bracket A show a weaker and more linear response to increasing N. Within bracket A, Exploit J10, Las Vegas Weight and Exploit Fast respond the strongest to an increase in the number of oscillators.

Table 13 shows the RMSE as a function of the number of oscillators in the ensemble for the four algorithms with lowest RMSE. The Las Vegas algorithm doesn't respond to an increase in the number of oscillators. The RMSE of Differential Evolution is initially low. Between 50 and 100 oscillators, the RMSE increases by a factor of 16. The growth rate drops between 100 and 200 oscillators to a factor of 4.

5.3.2 Number of perturbations

Figure 20 shows the RMSE as a function of the number of perturbations Z. For a majority of algorithms the RMSE decreases with an increase in the number of perturbations. This trend doesn't apply to the Purge and

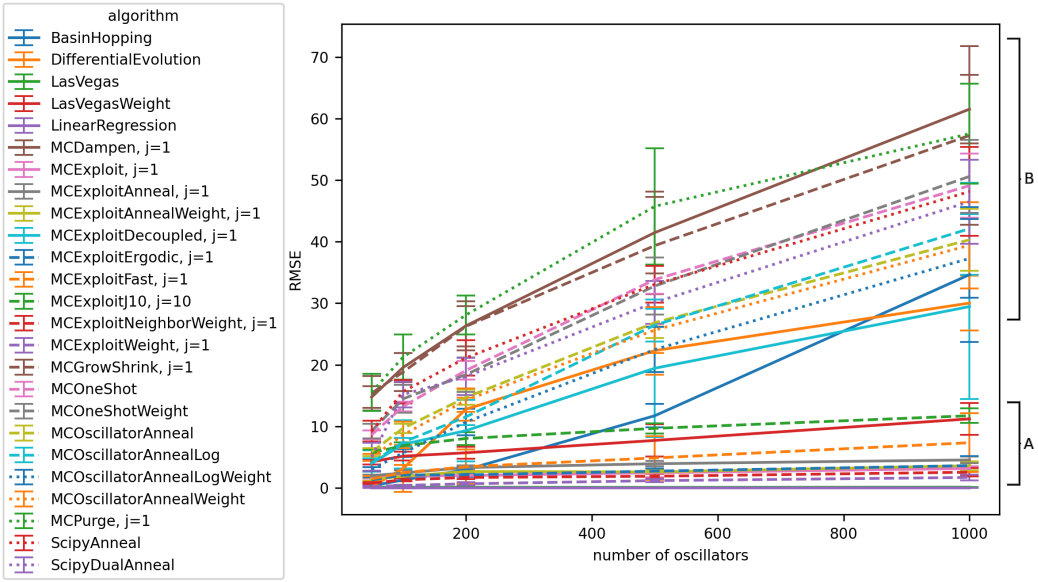


Figure 19: RMSE as a function of the number of oscillators N in an ensemble for a broad set of algorithms. Mean and standard deviation over 10 runs. See Table 13 for tabular data.

	RMSE							
	Linear Regression		Las Vegas		Diff. Evol.		Exploit W.	
N	M	SD	M	SD	M	SD	M	SD
50	1.30E-14	3.21E-14	0.14	0.07	0.19	0.18	0.23	0.14
100	1.52E-15	8.40E-16	0.11	0.04	3.16	3.82	0.41	0.12
200	1.16E-15	1.59E-16	0.10	0.03	12.79	1.87	0.67	0.23
500	1.33E-15	3.20E-16	0.08	0.03	22.31	3.88	1.18	0.27
1000	2.33E-15	7.10E-16	0.12	0.05	30.03	4.47	1.70	0.48

Table 13: RMSE as a function of the number of oscillators N with the three lowest-RMSE algorithms and linear regression. See Figure 19 for a larger set of algorithms.

Dampen algorithms that are only capable of decreasing oscillator gains. For algorithms benefitting from an increase in the number of perturbations, we find that the rate of decrease slows down with an increasing number of perturbations. This is expected as the algorithms are stochastic and the probability of finding a better solution decreases with an increasing number of perturbations. The SciPy Dual Anneal algorithm stands out with a steep decline in RMSE between 10 k and 50 k perturbations. Furthermore, Oscillator Anneal Log and Exploit Decoupled stand out for steeply descending between 10 k and 50 k perturbations. Although we can identify gaps between algorithms along the y-axis at 50 k perturbations, there are no unique patterns of RMSE shrinkage to distinguish groups.

Table 14 shows the RMSE for the four lowest-RMSE algorithms. The table shows that the RMSE decreases with an increasing number of perturbations for the three stochastic algorithms. Note that, the linear regression algorithm doesn't respond to a change in Z as it is not a stochastic algorithm; thus perturbations are not allocated or tracked. Since the other algorithms are stochastic they benefit from an increase in the number of perturbations. Exploit Weight shows a 68 fold RMSE decrease between 0 and 50k perturbations. Similarly, Exploit Neighbor Weight shows a 17 fold decrease. The decrease is smallest for the Las Vegas algorithm with an 11 fold decrease. It also stands out that the Las Vegas algorithm sets out with a lower RMSE than Exploit Weight achieves after 10 k perturbations; the Las Vegas algorithm is therefore more efficient in the number of perturbations needed.

	RMSE							
	Linear Regression		Las Vegas		Exploit W.		Exploit N. W.	
Z	M	SD	M	SD	M	SD	M	SD
0	1.29E-15	1.63E-16	0.71	1.11E-16	19.08	2.40	18.61	2.50
500	1.31E-15	1.44E-16	0.49	0.10	9.76	2.47	7.12	1.44
1000	1.32E-15	3.25E-16	0.44	0.12	3.97	0.79	3.60	0.52
5000	1.41E-15	3.26E-16	0.19	0.06	1.13	0.33	1.92	0.26
10000	1.27E-15	1.57E-16	0.15	0.04	0.73	0.25	1.51	0.37
50000	1.34E-15	1.17E-16	0.06	0.03	0.28	0.12	1.05	0.16

Table 14: RMSE as a function of the number of perturbations Z with the four lowest-RMSE algorithms. See Figure 20 for a larger set of algorithms.

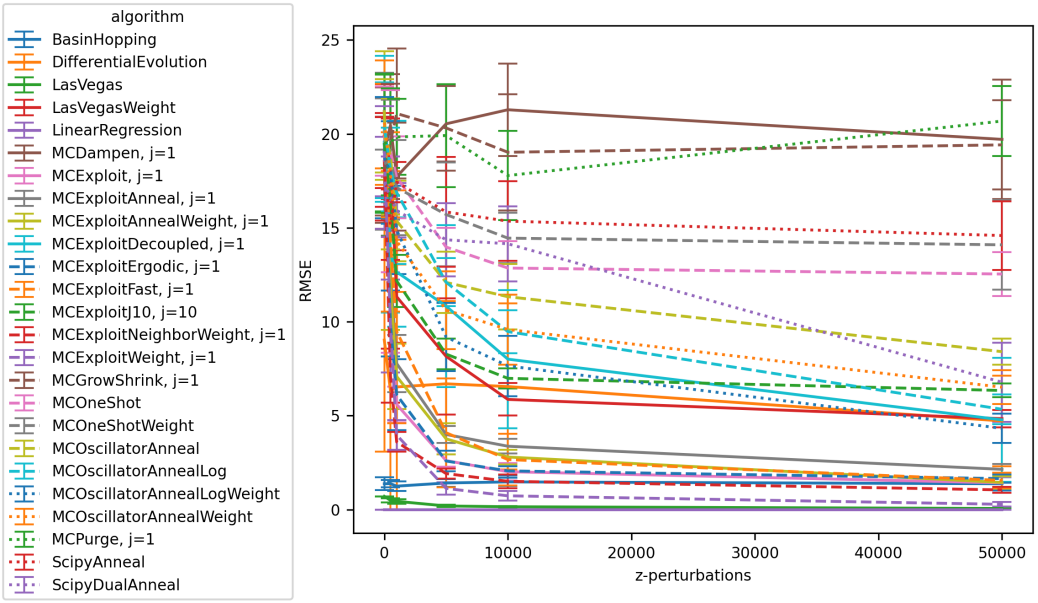


Figure 20: RMSE as a function of the number of perturbations Z applied to an ensemble for a broad set of algorithms. Mean and standard deviation over 10 runs. See Table 20 for tabular data.

5.4 Targets

5.4.1 Target frequency

Figure 21 shows the RMSE as a function of target frequency. Both sinusoids and damped chirp targets are approximated. For each target, the RMSE is the mean over four algorithms and five runs per algorithm. The algorithms are linear regression, Las Vegas, Exploit Weight and Exploit.

First, we observe that a majority of the sine targets yield a lower RMSE compared to the chirps. Following the target function results in Section 10.3, this is not surprising. A constant change of frequency requires a larger ensemble size to be approximated well. Second, we observe that sine targets from 1 Hz to 1 kHz yield similarly low errors. The 1 MHz sine follows the trend of previous sinusoids, while the 0.1 MHz sine stands out as having the largest RMSE of all targets and a near two-fold RMSE compared to the 1 Hz sine.

Figure 22 shows three target sinusoids approximated with the Las Vegas algorithm without averaging. The remaining parameters are kept the same as in Figure 21. In the figure, time is on the x-axis and signal amplitude is on the y-axis. The original target signal is shown in blue while the approximated signal appears in orange, labelled as prediction. The findings contrast the

results of Figure 21.

Firstly we find that the 1 Hz sine is not approximated at all, although the RMSE is low. The low RMSE is explained by the low amplitude of the target signal at the sampled interval of the function. Furthermore, because 1 Hz is slow in comparison to the observed time-frame of 2e-5 s, the target signal appears as a straight line.

Second, we find that the 100 kHz target is also not approximated. The explanation for the large RMSE while approximating the 0.1 MHz sine lies in the larger amplitude of the target signal in the observed time frame compared to lower frequency targets. Note that linear regression also fails to approximate the target with a similar RMSE of 0.69 for a single run. This indicates that there exists no better solution given the oscillators in the ensemble. The inability to fit targets of 100 kHz and slower is likely explained by the VO_2 frequency band of the oscillators. More specifically, (1) the upper and lower bound of the band, which are 1.2 MHz and 0.2 MHz, respectively and (2) the staircase-like shape of the VO_2 frequency band with a large probability of similar-frequency oscillators.

Third, we observe that the 1 MHz sine can be approximated. Its RMSE is lower than the 100 kHz sine's, but larger than the 1 Hz sine's, both of which are not approximated. Note that variations in amplitude of the target sine in the 1 MHz figure are artifacts of sampling. Cardinal sine interpolation has been used to smoothen both the target and the prediction in the given figure for plotting; the RMSE is calculated on raw data.

5.5 Target duration

Figure 23 shows the RMSE as a function of the target duration for a subset of algorithms. We find that with the exception of Differential Evolution, the RMSE remains constant as the duration is increased. Although not included here for clarity of the figure, we find a similar result for the remaining algorithms. Differential Evolution shows a larger standard deviation and an increase in RMSE beyond target durations 10^{-4} s compared to the other algorithms in the figure. This may be partially explained by the sample size of 3 runs per algorithm.

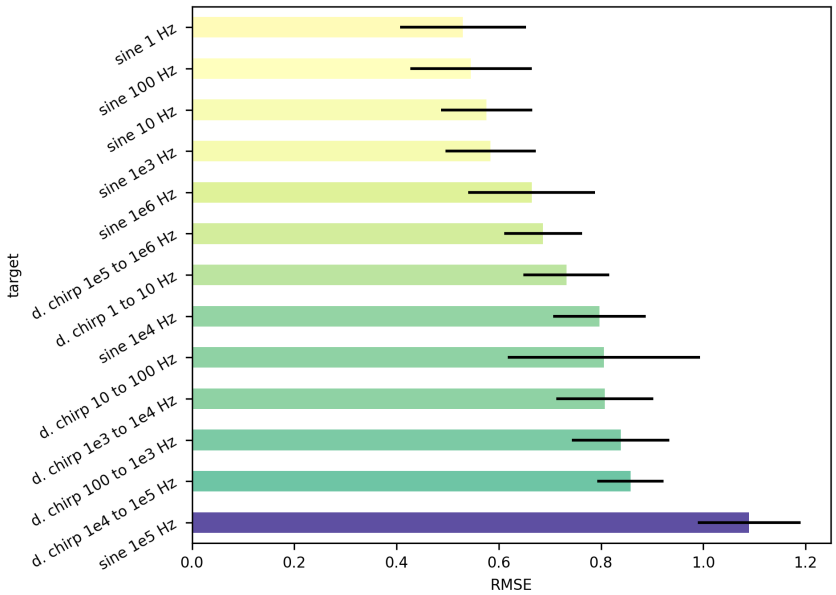


Figure 21: RMSE as a result of targets with varying frequency. Mean and standard deviation over 5 runs.

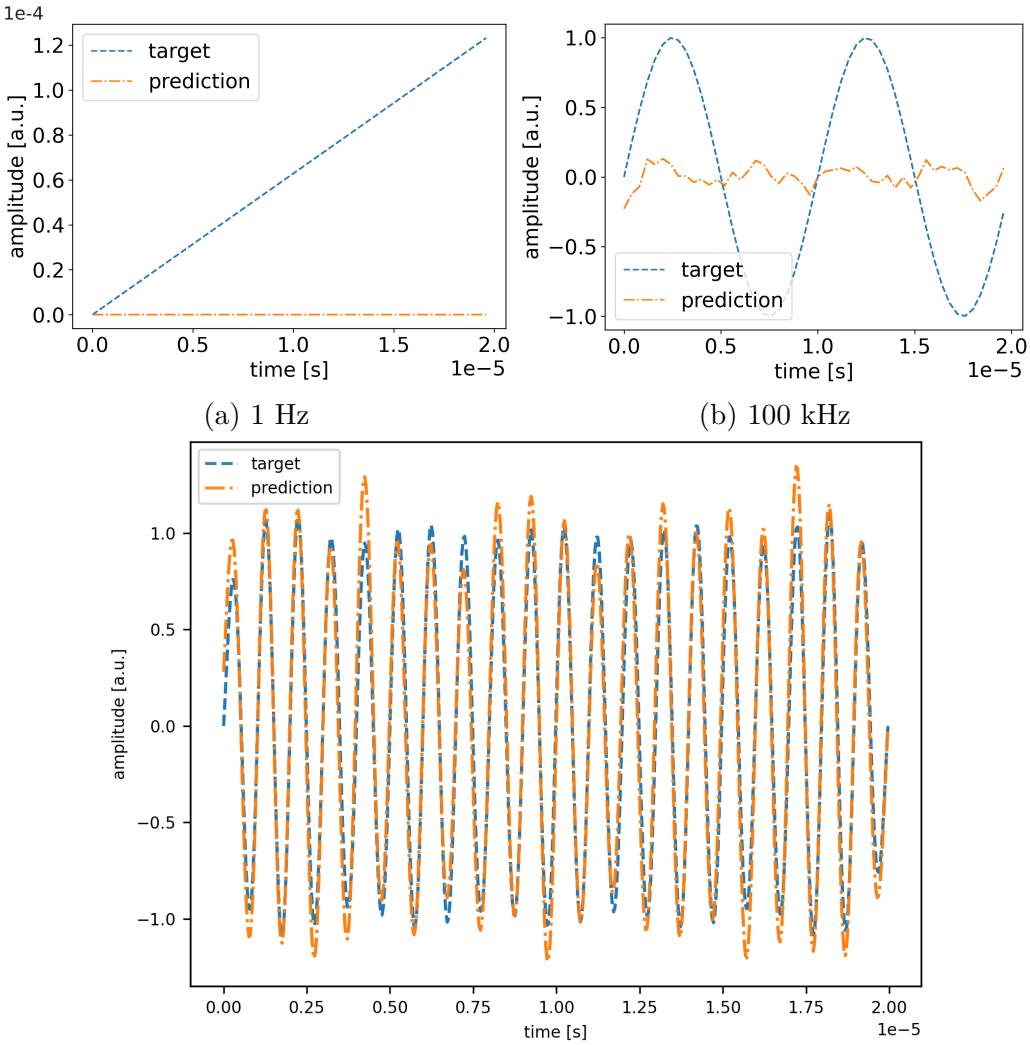
6 Discussion

6.1 Results discussion

6.1.1 Resistance diversity

The resistance of an oscillator directly controls its frequency. Therefore increased resistance diversity increases the bandwidth of the frequency response. We asked whether a wider band of resistances will lower the RMSE. The results in Section 5.2.1 allow two observations. First, the RMSE increases when oscillators slower than the target-frequency are added to the ensemble. Second, the RMSE decreases when faster than target signal oscillators are added; this was only visible for algorithms in group A. We present two possible explanations.

First, a wider band of resistances increases the search space. Meanwhile, the number of perturbations is kept constant. Thus, there may be insufficient resources to traverse the search space. This explanation seems appropriate given that the effect cannot be observed with linear regression. Linear regression finds an analytical solution that is not constrained by the number of perturbations. Furthermore, the Las Vegas algorithm is second best; because the algorithm starts out with an empty ensemble and only adds RMSE reducing oscillators, it doesn't need to remove RMSE increasing oscillators



(a) 1 Hz

(b) 100 kHz

(c) 1 MHz

Figure 22: Fitting sinusoids of 1 Hz with RMSE=7.15e-5, 100 kHz with RMSE=0.71 and 1 MHz with RMSE=0.14 using the Las Vegas algorithm. Result of a single run.

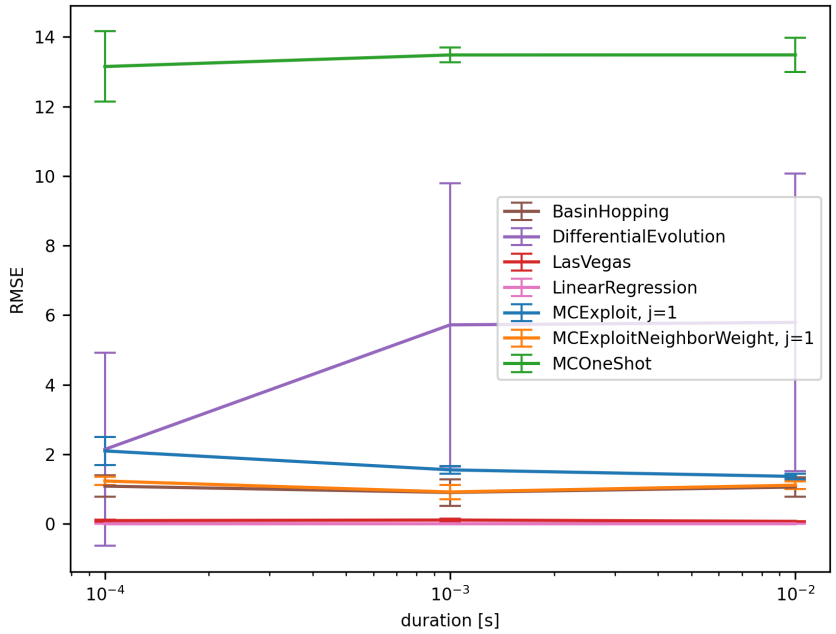


Figure 23: RMSE as a result of the target duration for a selection of algorithms. Mean and standard deviation over 3 runs.

from the ensemble. This implies that the non-gradient algorithms struggle with filtering out non-useful oscillators efficiently.

Second, the target signal is a sinusoid at 500 kHz. This means that oscillators with a frequency below 500 kHz are not helping to reduce the RMSE. The results show that the increase in RMSE occurs above 75 kΩ. Resistances of 75 kΩ and higher correspond to frequencies below 500 kHz, as shown in Figure 11 (right).

6.1.2 Phase diversity

We hypothesized that increasing phase diversity lowers the RMSE. The results in Section 5.2.2 are conclusive and show that the RMSE decreases with increasing phase diversity. As expected, an increase beyond 2π does not further decrease the RMSE because the sawtooth generated by the oscillators in the ensemble is periodic. The findings apply to all algorithms except for one outlier. It's unclear why Differential Evolution shows a larger RMSE with increasing phase.

6.1.3 Dynamic range

The hypothesis that dynamic range lowers RMSE must be rejected following the results of Section 5.2.3. We find two possible explanations for the increase in RMSE as a function of dynamic range.

First, the amplitude of the target signal is constant at 1 [a.u.]. Additionally, the amplitude of the VO_2 oscillation is near 1 V. An amplification of 0.1 V by a factor of 10 is already sufficient to reach the target signal amplitude. Second, the number of perturbations is constant at 20 k. While an increase in dynamic range widens the search space that can be explored, the number of steps to explore remains constant. Lastly, we observe that linear regression relies on negative gains. Therefore, allowing negative gains for all stochastic algorithms may further decrease RMSE.

6.1.4 Number of oscillators

We hypothesized that an increase in the number of oscillators decreases the RMSE. The results in Section 5.3.1 allow us to reject this hypothesis. The following explanations come to mind.

First, adding more oscillators increases the dimensionality of the search space. While a better solution can be found, more resources are necessary to find a good solution. More resources are not available however, as the number of perturbations is kept constant.

Second, the probability of nulling out bad oscillators may be too low under the uniform distribution from which an ensemble's gains are drawn. The Las Vegas algorithm circumvents this by only adding RMSE-decreasing oscillators to an initially empty ensemble.

Third, the oscillators' frequency response as a function of RC-circuit resistance is not continuous. The frequency response is staircase-like as shown in Figure 11 (right). As the plateaus capture the largest density of oscillators, the probability of adding an oscillator with a sufficiently unique frequency response is low.

6.1.5 Number of perturbations

We hypothesized that an increase in the number of perturbations to an oscillator ensemble decreases the RMSE. The results in Section 5.3.2 confirm this hypothesis. We also find that the Las Vegas algorithm is the most efficient stochastic algorithm in terms of the number of perturbations needed to achieve a low RMSE.

From these arguments we conclude that adding oscillators comes at an expense without a corresponding benefit. It remains open to investigate the

optimal number of oscillators. Although the number likely is below 50, it also likely is equal to or greater than the number of pleateaus in the frequency response. This indicates that materials beyond VO_2 should be explored.

6.1.6 Target frequency

We hypothesized that targets slower than the VO_2 frequency band are more difficult to approximate. Our results in Section 5.4.1 confirm this hypothesis. The findings show that when the target frequency is within the VO_2 frequency band, the target can be approximated.

We didn't expect that signals below, but close to, the VO_2 frequency band could not be approximated. Intuitively, more oscillators are necessary to approximate such targets. However as discussed in Section 6.1.4, adding more oscillators does not necessarily lead to a lower RMSE due to the staircase-like frequency response of the VO_2 oscillators.

6.1.7 Target duration

We hypothesized that increased duration of periodic target signals doesn't affect the RMSE. Our results confirm this hypothesis. This showcases an advantage of the neuromorphic over the digital compute paradigm.

In the digital domain, either higher resolution or a longer duration of a target signal correspond to more samples in the time domain. An increase in the number of samples is coupled to runtime cost; this is a limitation on the hardware level of serial digital computers. Meanwhile, the oscillator ensemble is inherently parallel and the approximation of a target is performed in the frequency domain. Thus it is independent of signal duration and resolution for periodic signals.

Note that we also don't observe an increase in RMSE for linear regression. This is explained by linear regression not using a perturbation budget that is enforced for the stochastic algorithms. An implementation of linear regression with a fixed budget of operations is necessary to show an increase in RMSE.

6.2 Conclusion

We find that signal approximation with an ensemble of VO_2 oscillators is possible for fast frequencies between 0.2 Ghz and 1.2 GHz. We also conclude that VO_2 is not suited for approximating slow frequency targets and by that account most real world signals.

The advantages of the neuromorphic approach are its parallel nature and its independence of signal duration and resolution for periodic signals. Additionally, signal approximation with an ensemble of oscillators promises to be more energy efficient than digital approaches. The need for such systems is exemplified by the rise of machine learning on microcontrollers (Warden & Situnayake, 2020).

We find two stochastic algorithms, Las Vegas and Exploit Weight, that perform particularly well while maintaining minimal state. This shows that non-gradient algorithms are worth exploring in a neuromorphic context.

6.3 Limitations and Future Work

We recognize that the presented results are the outcome of simulation. We make assumptions regarding the behavior of the VO_2 oscillator by disregarding its probabilistic nature reported by Maffezzoni et al. (2015). We also use caching of oscillator signals to speed up the simulation. It may be argued that the narrow steps between VO_2 frequency plateaus are underrepresented as a result. Furthermore, we acknowledge that the integration of a number of VO_2 oscillators in the order of 100 oscillators is yet to be achieved in physical circuits (Corti et al., 2020). Yet, the simulation results are motivating to continue with such efforts.

Due to the limitations of the VO_2 material, the exploration of other materials for the purpose of signal approximation is recommended. We expect that an improved oscillator ensemble can benefit from a variety of oscillator circuits covering different frequency bands. This in turn may allow for the approximation of lower frequency signals. Another approach to approximate slower targets may be the use of a low-pass filter applied to the summed ensemble signal to filter out high frequency components.

The perturbation of the oscillator ensemble relies on fast manipulation of resistors in order to change frequency, gain and offset. It remains to be explored how this can be achieved in a physical system. Possible solutions include the use of potentiometers or an extended use of memristive devices that replace resistors in the oscillator circuit.

In the chosen approach, the combination of oscillator signals is exclusively linear and oscillators themselves are uncoupled. It is well known that nonlinearities play an important role in the function approximation capabilities of neural networks. Coupling of oscillators, so that the output of one oscillator drives the input of another, is another avenue to explore which allows drawing from literature on spiking neural networks and reservoir computing.

We hope that this work contributes to a future of machine learning guided by energy efficiency and inspired by the wonders of biological life.

References

Abel, S., Stark, D. J., Eltes, F., Ortmann, J. E., Caimi, D., & Fompeyrine, J. (2017). Multi-Level Optical Weights in Integrated Circuits. *2017 IEEE International Conference on Rebooting Computing (ICRC)*, 1–3. https://doi.org/10.1109/ICRC.2017.8123672

Agostinelli, A., Denk, T. I., Borsos, Z., Engel, J., Verzetti, M., Caillon, A., Huang, Q., Jansen, A., Roberts, A., Tagliasacchi, M., Sharifi, M., Zeghidour, N., & Frank, C. (2023, January 26). *MusicLM: Generating Music From Text*. arXiv: 2301.11325 [cs, eess]. https://doi.org/10.48550/arXiv.2301.11325

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.-J., Taba, B., Beakes, M., Brezzo, B., Kuang, J. B., Manohar, R., Risk, W. P., Jackson, B., & Modha, D. S. (2015). TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557. https://doi.org/10.1109/TCAD.2015.2474396

Alspector, J., Meir, R., Yuhas, B., Jayakumar, A., & Lippe, D. (1992). A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks. *Advances in Neural Information Processing Systems*, *5*. Retrieved September 29, 2022, from https://proceedings.neurips.cc/paper/1992/hash/1595af6435015c77a7149e92a551338e-Abstract.html

Altman, N. S. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, *46*(3), 175–185. https://doi.org/10.1080/00031305.1992.10475879

Babai, L. (1979). Monte-Carlo algorithms in graph isomorphism testing. *Université de Montréal Technical Report*.

Bäck, T., & Schwefel, H.-P. (1993). An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, *1*(1), 1–23. https://doi.org/10.1162/evco.1993.1.1.1

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of machine learning research*, *13*(2), 25.

Bird, J., & Paolo, E. D. (2008). Gordon Pask His Maverick Machines. *The Mechanical Mind in History*, 185. Retrieved September 11, 2022, from https://www.academia.edu/54223516/Gordon_Pask_His_Maverick_Machines

Boon, M. N., Euler, H.-C. R., Chen, T., van de Ven, B., Ibarra, U. A., Bobbert, P. A., & van der Wiel, W. G. (2021, May 15). *Gradient*

Descent in Materio. arXiv: 2105.11233 [cs]. Retrieved September 5, 2022, from http://arxiv.org/abs/2105.11233

Brainchip. (2022, September 19). *The world's first commercial producer of neuromorphic IP*. BrainChip. Retrieved September 19, 2022, from https://brainchip.com/

Brainchip Stock. (2022, September 19). Retrieved September 19, 2022, from https://investors.brainchip.com/stock

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020, July 22). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs]. https://doi.org/10.48550/arXiv.2005.14165

CIRCUIT060021 Design tool | TI.com. (n.d.). Texas Instruments. Retrieved January 7, 2023, from https://www.ti.com/tool/CIRCUIT060021

CIRCUIT060022 Design tool | TI.com. (n.d.). Texas Instruments. Retrieved January 7, 2023, from https://www.ti.com/tool/CIRCUIT060022#overview

Cooley, J. W., & Tukey, J. W. (1965). An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of computation*, *19*(90), 297–301.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Corti, E., Khanna, A., Niang, K., Robertson, J., Moselund, K. E., Gotsmann, B., Datta, S., & Karg, S. (2020). Time-Delay Encoded Image Recognition in a Network of Resistively Coupled VO on Si Oscillators. *IEEE Electron Device Letters*, *41*(4), 629–632. https://doi.org/10.1109/LED.2020.2972006

Dahlke, K. (2002–2022). *Sines and Cosines are Orthogonal*. Math Reference Project. Retrieved January 3, 2023, from http://www.mathreference.com/la-xf-four,orth.html

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., … Wang, H. (2018). Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, *38*(1), 82–99. https://doi.org/10.1109/MM.2018.112130359

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May 24). *BERT: Pre-training of Deep Bidirectional Transformers for Language Under-*

standing. arXiv: 1810.04805 [cs]. https://doi.org/10.48550/arXiv.1810.04805

Energy use per person. (n.d.). Our World in Data. Retrieved September 3, 2022, from https://ourworldindata.org/grapher/per-capita-energy-use

Frady, E. P., Sanborn, S., Shrestha, S. B., Rubin, D. B. D., Orchard, G., Sommer, F. T., & Davies, M. (2022). Efficient Neuromorphic Signal Processing with Resonator Neurons. *Journal of Signal Processing Systems*, *94*(10), 917–927. https://doi.org/10.1007/s11265-022-01772-5

Franz, A., Hoffmann, K. H., & Salamon, P. (2001). Best Possible Strategy for Finding Ground States. *Physical Review Letters*, *86*(23), 5219–5222. https://doi.org/10.1103/PhysRevLett.86.5219

Freeth, T., Bitsakis, Y., Moussas, X., Seiradakis, J. H., Tselikas, A., Mangou, H., Zafeiropoulou, M., Hadland, R., Bate, D., Ramsey, A., Allen, M., Crawley, A., Hockley, P., Malzbender, T., Gelb, D., Ambrisco, W., & Edmunds, M. G. (2006). Decoding the ancient Greek astronomical calculator known as the Antikythera Mechanism. *Nature*, *444*(7119), 587–591. https://doi.org/10.1038/nature05357

Geman, S., & Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-6*(6), 721–741. https://doi.org/10.1109/TPAMI.1984.4767596

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

H F, C. (n.d.). *HKBWS - Bird Call, Magpie*. Retrieved May 26, 2023, from http://www.hkbws.org.hk/web/eng/bird_call_eng.htm
Download at http://www.hkbws.org.hk/web/chi/birdcall/Magpie.wav.

Har-Peled, S. (2015). Intro, Quick Sort and BSP. *University of Illinois at Urbana–Champaign, 598: Randomized Algorithms*. Retrieved May 26, 2023, from https://sarielhp.org/teach/10/a_rand_alg/lec/01_intro.pdf
Unpublished lecture notes.

Hoare, C. A. R. (1962). Quicksort. *The Computer Journal*, *5*(1), 10–16. https://doi.org/10.1093/comjnl/5.1.10

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Iwamatsu, M., & Okabe, Y. (2004). Basin hopping with occasional jumping. *Chemical Physics Letters*, *399*(4), 396–400. https://doi.org/10.1016/j.cplett.2004.10.032

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

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. Retrieved February 17, 2023, from https://www.jstor.org/stable/1690046

Kormondy, K. J., Popoff, Y., Sousa, M., Eltes, F., Caimi, D., Rossell, M. D., Fiebig, M., Hoffmann, P., Marchiori, C., Reinke, M., Trassin, M., Demkov, A. A., Fompeyrine, J., & Abel, S. (2017). Microstructure and ferroelectricity of BaTiO$\less$sub$\greater$3$\less$/sub$\greater$ thin films on Si for integrated photonics. *Nanotechnology*, *28*(7), 075706. https://doi.org/10.1088/1361-6528/aa53c2

Leiter, C., Zhang, R., Chen, Y., Belouadi, J., Larionov, D., Fresen, V., & Eger, S. (2023, February 20). *ChatGPT: A Meta-Analysis after 2.5 Months*. arXiv: 2302.13795 [cs]. https://doi.org/10.48550/arXiv.2302.13795

Levenberg, K. (1944). A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics*, *2*(2), 164–168. Retrieved May 20, 2023, from https://www.jstor.org/stable/43633451

Loeffler, A., Zhu, R., Hochstetter, J., Li, M., Fu, K., Diaz-Alvarez, A., Nakayama, T., Shine, J. M., & Kuncic, Z. (2020). Topological Properties of Neuromorphic Nanowire Networks. *Frontiers in Neuroscience*, *14*. Retrieved January 4, 2023, from https://www.frontiersin.org/articles/10.3389/fnins.2020.00184

Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, *47*(4), 173–180. https://doi.org/10.1016/0020-0190(93)90029-9

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149. https://doi.org/10.1016/j.cosrev.2009.03.005

Lukoševičius, M., Jaeger, H., & Schrauwen, B. (2012). Reservoir Computing Trends. *KI - Künstliche Intelligenz*, *26*(4), 365–371. https://doi.org/10.1007/s13218-012-0204-5

Maffezzoni, P., Daniel, L., Shukla, N., Datta, S., & Raychowdhury, A. (2015). Modeling and Simulation of Vanadium Dioxide Relaxation Oscillators. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *62*(9), 2207–2215. https://doi.org/10.1109/TCSI.2015.2452332

McClellan, J. H., Schafer, R. W., & Yoder, M. A. (2017). *DSP First* (Second edition, global edition). Pearson.

Mead, C. (1990). Neuromorphic Electronic Systems. *PROCEEDINGS OF THE IEEE*, *78*, 8.

Molad, E., Horwitz, E., Valevski, D., Acha, A. R., Matias, Y., Pritch, Y., Leviathan, Y., & Hoshen, Y. (2023, February 2). *Dreamix: Video Diffusion Models are General Video Editors*. arXiv: 2302.01329 [cs]. https://doi.org/10.48550/arXiv.2302.01329

Moll, M., Olma, R., & Müller, T. (2022, January 3). *VISION EQXX – taking electric range and efficiency to an entirely new level*. marsMediaSite. Retrieved September 19, 2022, from https://group-media.mercedes-benz.com/marsMediaSite/en/instance/ko/VISION-EQXX--taking-electric-range-and-efficiency-to-an-entirely-new-level.xhtml?oid=52282663

Muller, L. K., Stark, P., Offrein, B. J., & Abel, S. (2020). Neuromorphic Systems Design by Matching Inductive Biases to Hardware Constraints. *Frontiers in Neuroscience*, *14*. Retrieved September 13, 2022, from https://www.frontiersin.org/articles/10.3389/fnins.2020.00437

Nakajima, K., & Fischer, I. (Eds.). (2021). *Reservoir Computing: Theory, Physical Implementations, and Applications*. Springer Singapore. https://doi.org/10.1007/978-981-13-1687-6

Neal, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, ON, Canada.

Ngspice, the open source Spice circuit simulator. (n.d.). Retrieved January 22, 2023, from https://ngspice.sourceforge.io/

Núñez, J., Avedillo, M. J., Jiménez, M., Quintana, J. M., Todri-Sanial, A., Corti, E., Karg, S., & Linares-Barranco, B. (2021). Oscillatory Neural Networks Using VO2 Based Phase Encoded Logic. *Frontiers in Neuroscience*, *15*, 655823. https://doi.org/10.3389/fnins.2021.655823

PacDV. (2022). *Voices and Vocal Wav Sound Effects*. Pacific Digital Video. Retrieved January 4, 2023, from https://www.pacdv.com/sounds/voices-5.html
files yes-5.wav and okay-7.wav.

Pask, G. (1960). The natural history of networks. *Self-organizing systems*, 232–263.

Pearson, K. (1905). The Problem of the Random Walk. *Nature*, *72*(1865), 294–294. https://doi.org/10.1038/072294b0

Prior, H., Schwarz, A., & Güntürkün, O. (2008). Mirror-Induced Behavior in the Magpie (Pica pica): Evidence of Self-Recognition. *PLOS Biology*, *6*(8), e202. https://doi.org/10.1371/journal.pbio.0060202

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. *OpenAI blog*.

Radu, I. P., Govoreanu, B., Mertens, S., Shi, X., Cantoro, M., Schaekers, M., Jurczak, M., Gendt, S. D., Stesmans, A., Kittl, J. A., Heyns, M., & Martens, K. (2015). Switching mechanism in two-terminal vanadium dioxide devices. *Nanotechnology*, *26*(16), 165202. https://doi.org/10.1088/0957-4484/26/16/165202

Reisberg, D. (2022). *Cognition: Exploring the science of the mind* (8e). W. W. Norton & Company, Inc.
OCLC: 1251737663.

Rockwell, A. (2017, August 28). *The History of Artificial Intelligence*. Science in the News. Retrieved September 2, 2022, from https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

Ruiz Euler, H.-C., Boon, M. N., Wildeboer, J. T., van de Ven, B., Chen, T., Broersma, H., Bobbert, P. A., & van der Wiel, W. G. (2020). A deep-learning approach to realizing functionality in nanoelectronic devices. *Nature Nanotechnology*, *15*(12), 992–998. https://doi.org/10.1038/s41565-020-00779-y

Russell, S. J., Norvig, P., & Davis, E. (2010). *Artificial intelligence: A modern approach* (3rd ed). Prentice Hall.

Schuman, C. D., Kulkarni, S. R., Parsa, M., Mitchell, J. P., Date, P., & Kay, B. (2022). Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, *2*(1), 10–19. https://doi.org/10.1038/s43588-021-00184-y

Sculley, D. (n.d.). *Non-Inverting Amplifier*. Tufts University. Retrieved January 7, 2023, from https://www.eecs.tufts.edu/~dsculley/tutorial/opamps/opamps2.html
Kaggle CEO.

Shannon, C. (1949). Communication in the Presence of Noise. *Proceedings of the IRE*, *37*(1), 10–21. https://doi.org/10.1109/JRPROC.1949.232969

Sony to Release Two Types of Stacked Event-Based Vision Sensors with the Industry's Smallest 4.86 micrometer Pixel Size for Detecting Subject Changes Only Delivering High-Speed, High-Precision Data Acquisition to Improve Industrial Equipment Productivity. (2021, September 9). Sony Semiconductor Solutions Group. Retrieved September 19, 2022, from https://www.sony-semicon.com/en/news/2021/2021090901.html

Stark, P., Horst, F., Dangel, R., Weiss, J., & Offrein, B. J. (2020). Opportunities for integrated photonic neural networks. *Nanophotonics*, *9*(13), 4221–4232. https://doi.org/10.1515/nanoph-2020-0297

Storn, R., & Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of*

Global Optimization, *11*(4), 341–359. https://doi.org/10.1023/A:1008202821328

Strubell, E., Ganesh, A., & McCallum, A. (2019, June 5). *Energy and Policy Considerations for Deep Learning in NLP*. arXiv: 1906.02243 [cs]. https://doi.org/10.48550/arXiv.1906.02243

Tappe Maestro, R. (2023, July 1). *Function generation from a Sum of Oscillator Signals* (Version 1.0.0). https://github.com/Wehzie/master-thesis Commit ed1650e163da8aafd7e6e4ba5b8da7b76b589a19.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47–63. https://doi.org/10.1016/j.neunet.2018.12.002

Tsallis, C., & Stariolo, D. A. (1996). Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, *233*(1), 395–406. https://doi.org/10.1016/S0378-4371(96)00271-3

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems*, *30*. Retrieved September 3, 2022, from https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Veritasium (**typedirector**). (2021, December 21). *The Most Powerful Computers You've Never Heard Of* [Video]. Retrieved August 30, 2022, from https://www.youtube.com/watch?v=IgF3OX8nT0w

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … van Mulbregt, P. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wales, D. J., & Doye, J. P. K. (1997). Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *The Journal of Physical Chemistry A*, *101*(28), 5111–5116. https://doi.org/10.1021/jp970984n

Warden, P., & Situnayake, D. (2020). *TinyML: Machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers* (First edition). O'Reilly. tinymlbook.com
OCLC: on1104044619.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, *393*(6684), 440–442. https://doi.org/10.1038/30918

Weisstein, E. W. (n.d.-a). *Generalized Fourier Series*. Retrieved January 22, 2023, from https://mathworld.wolfram.com/

Weisstein, E. W. (n.d.-b). *Triangle Wave*. Retrieved January 24, 2023, from https://mathworld.wolfram.com/FourierSeriesTriangleWave.html

Wong, C. (1994). "SPICE" as a Maxwell's equations solver. *1994 Second International Conference on Computation in Electromagnetics*, 303–306. https://doi.org/10.1049/cp:19940077

Wong, W. (2015, June 10). *Answer to "Fourier transform with non sine functions?"*. Mathematics Stack Exchange. Retrieved January 3, 2023, from https://math.stackexchange.com/a/1319755

Xiang, Y., Sun, D. Y., Fan, W., & Gong, X. G. (1997). Generalized simulated annealing algorithm and its application to the Thomson model. *Physics Letters A*, *233*(3), 216–220. https://doi.org/10.1016/S0375-9601(97)00474-X

7 Acknowledgements

A thesis rarely emerges alone. For this one I've been supported by at least the following. Professor Lambert Schomaker, who opted to supervise this project together with promovendus Davide Cipollini. Professor Schomaker had a clear vision of the project from the start. I'm glad that he guided me through the fog of uncertainty with examples, ideas and understanding. Davide accompanied me through the project at each step and shared his own experiences about becoming a researcher. Dirk Bischof offered his extensive expertise in electrical engineering, aided with circuit design and first inspired a passion for electronics a long time ago. Jonas Zimmermann provided me with advice on multiple details of the project including Markov Chain Monte Carlo methods and Fourier Series; I'd like to thank him for his tireless explanations. I'd like to thank Nils Kruse for reviewing source code and pointing out bugs in a forest full of trees. Furthermore, Gonçalo Hora de Carvalho listened and uplifted my spirits when it was needed. Elisabetta Blyumin provided essential insights for the colloquium, drawing on her physics perspective. Nikolas Rieder, Robin Müller and Alia Schönberg asked questions which I didn't know to ask. Thank you Barbara, German, Claudia, Luz, Fabio, Rüdiger, Milena, Luuk, Aaron, Marta for attending. My thanks also go to the RUG staff for making the colloquium possible on short notice and to the university for being a place of academic freedom. Lastly, I thank my parents for the support that they have provided me with throughout my studies and beyond, my mother for understanding without words and my father for being an anchor of reason.

8 Abbreviations

Table 15 lists abbreviations used throughout this thesis.

Abbreviation	Expansion
DL	deep learning
ML	machine learning
NC	neuromorphic computing
RC	reservoir computing
RC-circuit	resistor-capacitor circuit
ESN	echo state network
LSTM	long-short term memory
a.u.	arbitrary units
AM	amplitude modulation
FM	frequency modulation
RMSE	root mean square error
M	mean
SD	standard deviation
T.	target
div.	diversity
CPU	central processing unit
GPU	graphics processing unit

Table 15: Abbreviations

9 Appendix A

9.1 Additional Algorithms

9.1.1 Simulated Annealing

Simulated Annealing is an improvement over both random search and the random walk. The algorithm takes inspiration from processes in statistical mechanics, where a material's ground states are found by slow cooling (Kirkpatrick et al., 1983) as otherwise defects and only partially stable structures are formed. The algorithm is describe in Listing 7.

Listing 7: Simulated Annealing

```
# draw_full(): draw an N dimensional sample from the search-space
# draw(sample): redraw one variable from the search-space
# l(x): loss function
# crit: stopping criterion
# evaluate(): evaluate whether the stopping criterion is met
# exp(): exponential function
# rand(): random real number between 0 and 1

best = draw_full() # initial random sample in search-space
temperature = 1
while not crit:
    temperature = schedule(temperature, crit)
    candidate = draw(best)

    if l(candidate) < l(best):
        best = candidate
    else:
        diff = l(candidate) - l(best)
        acceptance_ratio = exp(diff / temperature)
        decision = rand() <= acceptance_ratio
        if decision == True:
            best = candidate

    crit = evaluate(temperature)
```

Similar to the random walk, the algorithm starts by initializing a candidate solution x_1 with a random configuration; furthermore, a temperature parameter is initialized (Russell et al., 2010). This parameter is updated on each iteration as a function. The temperature schedule is important for a good outcome of the optimization and dependent on the problem at hand; it constitutes a tuneable hyper parameter. When a new candidate solution x_i has a loss lower than the current best solution x_{best}, the new candidate solution is always accepted. If the new candidate solution x_i has a higher

loss than the current best solution x_{best}, the new candidate solution is accepted by some chance. This probability of acceptance is decreased as (1) the temperature decreases and (2) the loss of a candidate solution x_i increases. Typically, the algorithm stops when the temperature reaches a predefined minimum value; hence the stopping criterion evaluation function is described as a function of temperature, albeit other stopping criteria are possible.

In this work we explore Simulated Annealing for a neuromorphic compute system. Interestingly, Simulated Annealing also played a central role in the development of digital computers, in particular in solving the very large scale integration (VLSI) problem (Kirkpatrick et al., 1983). The VLSI problem is the problem of placing transistors on a chip such that the total wire length is minimized, a problem which resembles that of the traveling salesman problem.

The Simulated Annealing algorithm developed by Kirkpatrick et al. (1983) is now known as the Classic Simulated Annealing (CSA) algorithm. More recent developments are the fast Simulated Annealing (FSA) algorithm and the Generalized Simulated Annealing algorithm (GSA) (Tsallis & Stariolo, 1996; Xiang et al., 1997). Another notable algorithm that builds on top of Simulated Annealing is Basin Hopping (Iwamatsu & Okabe, 2004; Wales & Doye, 1997). In Basin Hopping, an iteration involves two steps. Firstly a random jump that is similar to an iteration of Simulated Annealing. Secondly, a local minimization step is performed to find the local minimum within the current basin discovered by the hop. For local minimization, any local optimization algorithm can be used. Similarly to Simulated Annealing, the choice of the cooling schedule is important for good results.

9.1.2 Differential Evolution

Another branch of Monte Carlo methods are evolutionary algorithms. Broadly, evolutionary algorithms take inspiration from biological evolutionary processes. A central idea of evolutionary algorithms then is the concept of a population of candidate solutions. The field of Differential Evolution is wide; Bäck and Schwefel (1993) give an early overview of the field and identify three main branches: genetic algorithms, evolution strategies and evolutionary programming. A more recent type of evolutionary algorithm is Differential Evolution (Storn & Price, 1997) which shares similarities in particular with genetic algorithms and evolution strategies.

Listing 8: Differential Evolution

```
1 # draw_full(): draw N dimensional sample from the search-space
2 # recombine(): randomly take three individuals from the
      ↪ population and recombine them
```

```
3  # crossover(): mix the original sample with the recombined (
       ↪ mutated) individual
4  # l(x): loss function
5  # crit: stopping criterion
6  # evaluate(): evaluate whether the stopping criterion is met
7  # size_pop: number of individuals in the population
8
9  pop = [draw_full() for _ in range(size_pop)] # draw size_pop
       ↪ initial random samples
10 while not crit:
11     for indiv in pop:
12         indiv_mutate = recombine(indiv, pop)
13         indiv_trial = crossover(indiv, indiv_mutate)
14         if l(indiv_trial) < l(indiv):
15             indiv = indiv_trial
16     crit = evaluate()
```

Listing 8 shows pseudo code for a basic variant of the Differential Evolution algorithm (Storn & Price, 1997). Similar to random search, the algorithm starts by drawing random samples. Instead of drawing a single sample, Φ samples are drawn. Φ then corresponds to the number of individuals or the population size. If a sample or individual is a vector of N dimensions, then a population is a matrix of N by Φ dimensions. In the recombination step, three unique individuals are taken from population space. Let x_i denote the i-th individual and let F be a real constant. $x_1 + F * (x_2 - x_3)$ forms a mutant individual by recombination. F then controls the extent to which a difference between individuals x_2 and x_3 is amplified. In the crossover step, the mutant individual is crossed with the original population. Let x^j denote the j-th scalar of an N dimensional vector that forms an individual. Then for each entry j in the range 1 to N, either x^j_{original} or p^j_{mutant} is chosen to form the trial vector. Whether the original or the mutant value at j is carried to the trial vector depends on CR. CR is the crossover ratio hyperparameter; it is a real constant that controls the extent to which mutant individuals are crossed with their original counterparts. Lastly, the trial vector is compared with the original vector. If the trial vector is better than the original vector, the trial vector is accepted and replaces the original one. In the vocabulary of evolutionary algorithms, selection takes place. Differential Evolution is often run for a fixed number of iterations, termed generations; other stopping criteria are feasible.

Although more advanced than naive random search, Differential Evolution shares algorithmic properties such as non-completeness. In a neuromorphic context, evolutionary algorithms that maintain a host of candidate solutions are more difficult to implement compared to annealing algorithms. This is primarily explained by the need for a physical representation of the

population. In the case of circuit development, this means more difficult circuit design.

9.2 Relation to Fourier Methods

The oscillator ensemble approach resembles a Fourier series in materio. Here we discuss some of the mathematics behind Fourier series and Fourier synthesis. We aim to argue that Fourier synthesis is possible with non-sinusoidal basis functions because the VO_2 device produces an inverse sawtooth wave. The following discussion focuses on continuous time signals.

A Fourier series is a sum of harmonic sinusoids which can be used to approximate a periodic signal. Typically, sinusoids are used to synthesize periodic signals. This may be in part, because a proof of a Fourier series' convergence to a periodic signal is relatively straightforward when summing sinusoids; see McClellan et al. (2017) for said proof. The derivation relies on the properties of complex exponentials which by Euler's formula

$$e^{i\theta} = \cos(\theta) + i\sin(\theta). \tag{15}$$

decompose into a real cosine and an imaginary sine part. The sine function has further advantageous properties. Namely, sinusoids are symmetric, are differentiable at any point, their basis is orthonormal and an eigenvector (Dahlke, 2002–2022; W. Wong, 2015). However, other periodic functions, for example a triangle wave or sawtooth wave, can be used to approximate a periodic signal, too. Note that the type of convergence achieved with an arbitrary periodic function isn't necessarily the same as that of a sinusoid. Discussing the nuances of Fourier series' convergence types is out of the scope of this work, as is proving convergence for non-trigonometric periodic functions. For some mathematical evidence, consider the generalized Fourier series[14]. The generalized Fourier series requires that its basis functions form a complete orthogonal system on a closed interval. Sine and cosine form such a system in the interval $[-\pi, \pi]$ with the integral over their product equal to 0. In the case of sinusoids this is easy to see, as $\int_{-\pi}^{\pi} \sin(x)dx = 0$ and $\int_{-\pi}^{\pi} \cos(x)dx = 0$. By their vertically symmetric wave form it's intuitively clear that we can find similar biorthogonal systems with triangle and square waves. We may be able to find a biorthogonal system with a sawtooth and inverse sawtooth wave, too. Empirically we can test this, by applying linear regression using the Least Squares method, to fit a sum of sawtooths to a sinusoid target signal, given that the signal diversity is sufficient. Signal diversity refers to a finite number of unique elements in a set of signals, where

[14] Weisstein, n.d.-a.

each signal is a periodic function with constant phase, frequency, amplitude and offset; two signals are unique when they vary in at least one of the aforementioned properties. See Listing 10 in the appendix for an example.

In order to synthesize aperiodic signals, the Fourier synthesis formula 16 is used. It states that an arbitrary function can be approximated by a finite sum of sinusoids of varying frequencies, phases, amplitudes and offsets. The Fourier synthesis formula is a generalization of the Fourier series formula from the interval $[-\pi, \pi]$ to the interval $[-\infty, \infty]$. For time-series, finite or infinite, it should be noted that any aperiodic signal can be considered periodic by selecting a window and considering it to be the interval $[0, 2\pi]$ of a periodic signal. Effectively, this extends the signal to the periodic domain. Therefore, Fourier synthesis is a function approximator for finite time-series.

$$f(x) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi_k) \tag{16}$$

9.3 Supplement to the Methods

9.3.1 Elaboration on the VO_2-circuit and oscillation function

The VO_2 device's time constants presented in Equation 5 are defined as

$$\tau_{\text{phase}} = R_{\text{eq}}^{\text{phase}} \cdot C_1, \tag{17}$$

where,

- τ_{phase} is the time constant of the metal or insulator phase respectively,
- $R_{\text{eq}}^{\text{phase}}$ is the combined resistance of the VO_2 device and the resistance of R1 during a given phase,
- C_1 is the capacitance of the capacitor C1.

Elaborating on Equation 5 the voltages that the capacitor C1 tends to, depending on the VO_2 device's state, are described by the equation

$$U_{\text{eq}}^{\text{phase}} = V_1 \cdot \frac{R_1}{R(U1)_{\text{phase}} + R_1}, \tag{18}$$

where,

- V_1 is the voltage of the DC voltage source V1,
- R_1 is the resistance of the resistor R1,

- $R(U1)_{\text{phase}}$ is the resistance of the VO_2 device U1 for a given phase.

Equations 5 and 17 describe how the oscillation period is related to the RC-circuit elements R1 and C1. Since the oscillation frequency f at the output terminal A is defined as inverse of the period T, we have $f = \frac{1}{T}$. Doubling of R1's resistance halves the time constants. While an increase in resistance does decrease oscillation frequency, this relationship is not linear as R1 also controls $U_{\text{eq}}^{\text{met}}$ and $U_{\text{eq}}^{\text{ins}}$. Also, doubling the capacitance of C1 halves the time constants with the relationship on frequency being linear. Still, C1 is kept constant for the purpose of this work as R1 is considered to be more easily changed. From Equation 18, we see that the voltage that the capacitor C1 tends towards is the product of the DC voltage and the combined resistance of U1 and R1.

Figure 6 describes the I-V characteristic of the VO_2 device as a function of applied voltage between the two terminals of the VO_2 device. Meanwhile Figure 24 describes the capacitor I-V characteristic seen at the terminals of the capacitor C1. Here, $U_{\text{eq}}^{\text{met}}$ and $U_{\text{eq}}^{\text{ins}}$ are the voltages that the capacitor C1 tends towards when the VO_2 device is in the metal or insulator phase respectively; compared to Maffezzoni et al. (2015) the notation U is preferred over E to describe a voltage. When the VO_2 device is in the metal phase, the voltage of the capacitor moves toward the DC voltage of V1. While when the VO_2 device is in the insulator phase, the capacitor discharges and the voltage at node A moves toward zero. When the VO_2 device is in the insulator phase and near V_H, the capacitor's voltage is at $V(C)_L$ with a small current flowing. Similarly, when the VO_2 device is in the metal state near V_L, the capacitor's voltage is at $V(C)_H$ with current flowing. The hysteresis cycle in the capacitor is clockwise while the hysteresis cycle in the VO_2 device is counter clockwise.

Concluding, the sequence of events is as follows. First, the VO_2 device starts in the insulator phase; the capacitor is not charged. Second, as the DC voltage is applied, the VO_2 device transitions to the metal phase and the capacitor charges. Third, as the capacitor charges, the voltage across the VO_2 device falls off and the VO_2 device transitions back to the insulator phase. Fourth, the capacitor discharges (see Equation 18) and voltage builds up across the VO_2 device again. Fifth, the VO_2 device transitions to the metal phase and the capacitor is charged again as the voltage in the VO_2 device falls off. This sequence of events repeats itself indefinitely. The resulting signal resembles an inverse sawtooth wave, as shown in Figure 8.

The waveform of the oscillator is described by the following two equations. Equation 19 describes the signal as a function of time during the VO_2's metal phase, while Equation 20 describes the signal during the insulator phase.

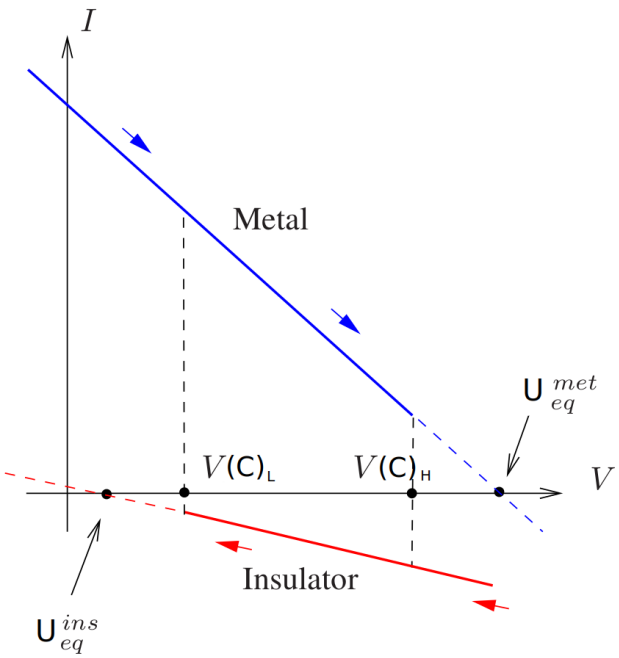


Figure 24: Voltage at the capacitor C1 as a function of the VO_2 device's state. Figure modified from Maffezzoni et al. (2015).

Note a slight deviation in the last summand from Maffezzoni et al. (2015); the change is justified in the Appendix, see Listing 9.

$$V_{\text{out}}^{\text{met}}(t) = (V_{\text{min}} - U_{\text{eq}}^{\text{met}}) \cdot \exp[-t/\tau_{\text{met}}] + U_{\text{eq}}^{\text{met}} \tag{19}$$

$$V_{\text{out}}^{\text{ins}}(t) = (V_{\text{max}} - U_{\text{eq}}^{\text{ins}}) \cdot \exp[-(t - T_{\text{met}})/\tau_{\text{ins}}] + U_{\text{eq}}^{\text{ins}} \tag{20}$$

where,

- $V_{\text{out}}^{\text{phase}}(t)$ is the output voltage of the oscillator at point A in Figure 7 while the VO_2 device is in a given phase,
- t refers to the time,
- V_{min} and V_{max} are the minimum and maximum output voltage of the oscillator,
- $U_{\text{eq}}^{\text{phase}}$ is the voltage that the capacitor C1 tends towards when the VO_2 device is in a given phase,
- τ_{phase} is the RC time constant of the circuit during the VO_2's phase,
- T_{met} is the duration of a metal phase.

9.3.2 Derivation of VO_2 offset

Equation 7 describes the offset voltage of the VO_2 device. We can derive the equation as follows. First, consider the subcircuit from V1_POS via R1_BIAS and R2_BIAS to V1_NEG as a series circuit. According to Kirchoff's current law, the current through this circuit is the same at any point. Furthermore, following Ohm's law, the current through a resistor is the voltage across the resistor divided by its resistance. Combining voltages and resistances in series, we can derive the following equation.

$$I = \frac{(V_{1\_\mathrm{POS}} + V_{1\_\mathrm{NEG}})}{(R_{1\_\mathrm{BIAS}} + R_{2\_\mathrm{BIAS}})} \tag{21}$$

The voltage at $U_{\mathrm{R1\_BIAS}}$ is similarly given by Ohm's law.

$$U_{\mathrm{R1\_BIAS}} = R_{1\_\mathrm{BIAS}} \cdot I \tag{22}$$

Upon substitution of I from Equation 21, into Equation 22 we arrive at the following equation.

$$U_{\mathrm{R1\_BIAS}} = R_{1\_\mathrm{BIAS}} \cdot \frac{(V_{1\_\mathrm{POS}} + V_{1\_\mathrm{NEG}})}{(R_{1\_\mathrm{BIAS}} + R_{2\_\mathrm{BIAS}})} \tag{23}$$

Let us call the voltage between R1_BIAS and R2_BIAS with respect to ground U_{OFFSET}, which applies at PIN 1 of U3_AMP. Then we subtract the voltage originating from V1_POS by the voltage drop across R1_BIAS to obtain U_{OFFSET}.

$$U_{\mathrm{OFFSET}} = V_{1\_\mathrm{POS}} - U_{\mathrm{R1\_BIAS}} \tag{24}$$

By substituting Equation 23 into Equation 24, we arrive at Equation 7.

9.4 Limitations of SPICE simulation

Initial exploration of the simulation capabilities of SPICE shows that the simulation of an oscillator is slow. Table 16 shows measured and estimated wall clock times of simulating a single oscillator for increasing signal durations. Table 17 shows three sets of parameters for VO_2 circuits. The first row of parameters is used to generate the results in Table 16; the choice of parameters is taken from Maffezzoni et al. (2015). The column headers map to the elements of a single RC-circuit as shown in Figure 7. N refers to the number of oscillators in the circuit. The form $U(a, b)$ refers to the use of meta-programming, where the circuit parameters are drawn from a uniform distribution ahead of simulation. The time step column refers to the time

step used in the transient analysis. By-hand experimentation shows that the time step must be sufficiently small to accurately capture the oscillation; in the case of the VO_2 oscillator, time steps in the order of 1e-9 s are required although the VO_2 frequency doesn't exceed 1.2e6 Hz. This small time step makes the simulation of longer durations prohibitively slow. The second row of parameters shows the largest number of non-identical oscillators in a single circuit that I was capable of simulating. The parameters were found by hand-tuning. Whether a SPICE transient analysis succeeds is not deterministic for a fixed set of parameters as SPICE sometimes fails citing numerical instability. As the number of oscillators increases, the likelihood of failure increases. Other times, SPICE stops an ongoing simulation without reporting failure. The third row of Table 17 shows that SPICE is capable of simulating large numbers of oscillators when their parameters are identical.

signal duration	1 s	1e-1 s	1e-2 s	1e-3 s	1e-4 s	1e-5 s
wall clock time	est. 2 h	est. 700 s	67 s	7 s	1 s	<1 s

Table 16: Wall clock times to various durations of SPICE transient analysis for a single VO_2 oscillator circuit.

N osc.	DC [V]	R [kΩ]	min C [pF]	step [s]	duration [s]
1	14	47	300	5e-9	1e-5 to 1e-2
100	4	8	$U(40, 60)$	5e-9	1e-5
500	12	47	300	5e-9	1e-5

Table 17: Parameters for SPICE netlist generation with the purpose of testing SPICE's simulation capabilities.

9.5 Reproducibility

An open-source implementation of SPICE called ngspice[15], version 37, is used for this work. For meta-programming, Python version 3.10 is used. All source code will be made publicly available (Tappe Maestro, 2023). In single threaded execution of the simulation a constant seed of 5 is used for random number generation. For multi-threaded execution, seeding has not been implemented; all results presented in Section 5 are generated by single threaded execution.

[15]"Ngspice, the Open Source Spice Circuit Simulator", n.d.

9.6 Classification

We find that other tasks can be realized with VO_2 ensembles, for example classification. An ensemble forest shown in Figure 25 consists of trained ensembles. Each ensemble is trained to approximate a target class with a stochastic algorithm as laid out in Section 3.2. Note that the training phase is not shown in this diagram. After training, a test signal is compared to the output of each ensemble by RMSE. The ensemble with the lowest RMSE is chosen as the predicted class. The ensemble forest is inspired by the random forest algorithm (Breiman, 2001). We successfully trained two ensembles with the Las Vegas algorithm to distinguish between a sine and a square wave. The approach seems to tolerate frequency deviations between the train and test signals. We acknowledge that the presented design doesn't satisfy the ethos of neuromorphic computing as training and classification phases are strictly separated. Such a dichotomy isn't found in biological systems. Exploration in this direction is left for future work.

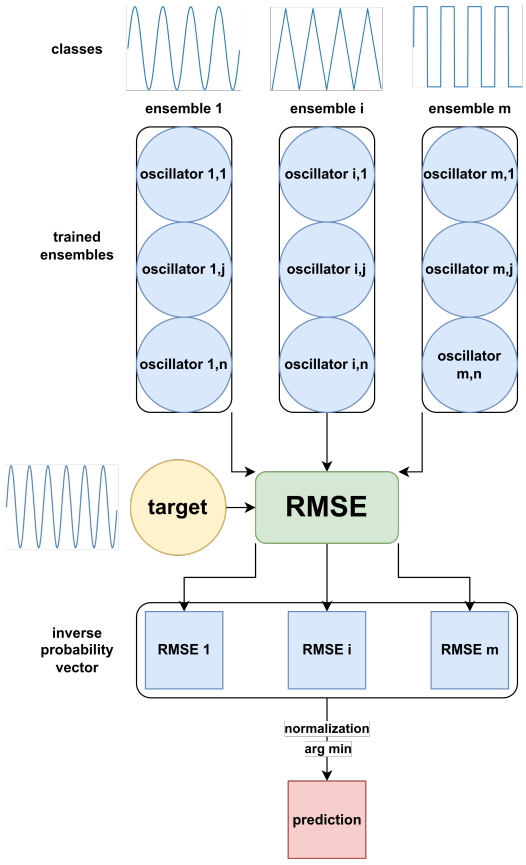


Figure 25: A forest of ensembles acts as a classifier.

10 Appendix B: Additional Results

10.1 Phase diversity

Figure 26 shows the RMSE as a function of phase diversity for a broad set of algorithms. The figure shows that the RMSE decreases with increasing phase diversity up to 2π for a majority of algorithms. A notable exception is Differential Evolution which shows larger RMSE at phase diversity 1π and at 2π compared to lower phase diversities. For a majority of algorithms no improvement in RMSE is visible beyond 2π phase diversity. Furthermore we can distinguish two groups of algorithms. Bracket B denotes algorithms with large RMSE at 0π phase diversity; we observe a steep decrease in RMSE with increasing phase diversity. Bracket A denotes algorithms with initially lower RMSE. Here, the effect of phase diversity on RMSE is less pronounced.

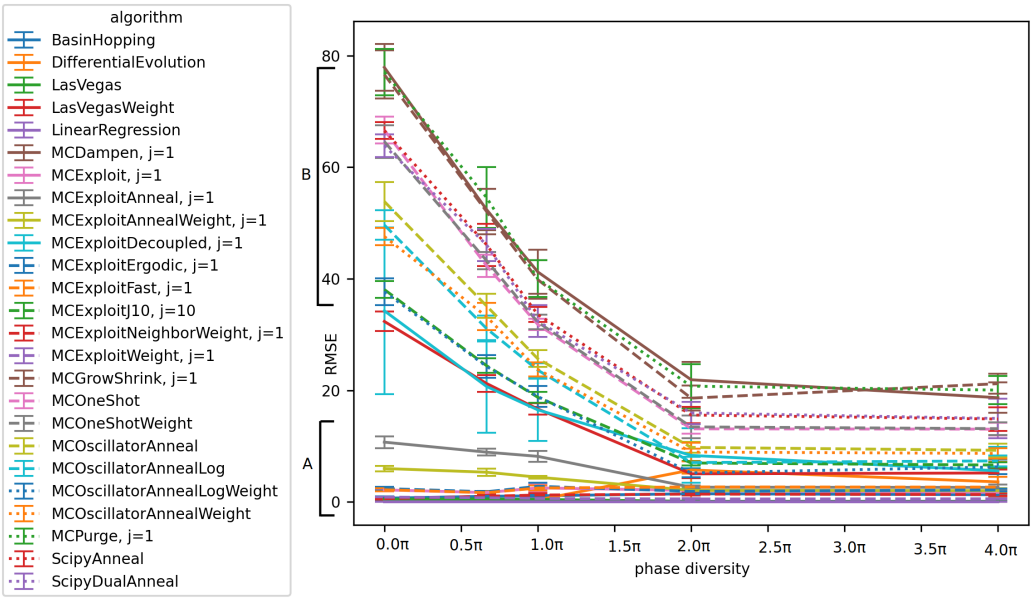


Figure 26: RMSE as a function of phase diversity for a broad set of algorithms. Mean and standard deviation over 10 runs. See Table 11 for tabular data.

10.2 Offset diversity

Figure 27 shows RMSE as a function of offset diversity for a broad set of algorithms. We can identify two brackets of algorithms, A and B. For algorithms

in bracket A, an increase in offset diversity has little effect on RMSE. A notable outlier in this bracket is Exploit Fast which shows a RMSE decrease by a factor of 4 between 0 and 50 offset diversity. Considering the sample size of 10 and standard deviations in the order of 10 and larger, the observation may be explained by chance. Bracket B shows an increase in RMSE as offset diversity increases. This means the Purge, Dampen, Grow-Shrink, SciPy Anneal and SciPy Dual Anneal algorithms; they have the largest RMSE at offset diversity 200. The increase in RMSE with offset diversity is approximately linear. Furthermore, the algorithms in bracket B show the highest RMSE at 0 offset diversity, too. This indicates that these algorithms generally perform poorly, regardless of offset diversity.

Table 18 shows the RMSE as a function of offset diversity for the four algorithms with the lowest RMSE. For linear regression, Las Vegas and Exploit Weight we observe an increase in mean RMSE and standard deviation as offset diversity increases; this relationship is linear. Although mean RMSE is lowest at 0 offset diversity, no clear trend is visible for the Exploit Neighbor Weight algorithm.

	RMSE							
	Linear Regression		Las Vegas		Exploit W.		Exploit N. W.	
Offset div.	M	SD	M	SD	M	SD	M	SD
0	1.33E-15	2.48E-16	0.10	0.03	0.44	0.13	1.15	0.21
50	1.55E-15	5.95E-16	0.12	0.04	0.44	0.15	1.45	0.30
100	1.57E-15	5.01E-16	0.10	0.03	0.59	0.21	1.41	0.15
150	1.56E-15	4.46E-16	0.13	0.04	0.61	0.26	1.30	0.23
200	1.62E-15	5.30E-16	0.14	0.06	0.88	0.31	1.55	0.32

Table 18: RMSE as a function of offset diversity with the three best stochastic algorithms and linear regression. See Figure 27 for a larger set of algorithms.

We hypothesized that an increase in offset diversity would reduce RMSE. Following the results, we can't accept our hypothesis. We believe that this is due the experiment design.

In this experiment, the target signal has a mean of zero; thus its offset is zero and an optimization of offset is not necessary. Although we can conclude that the experimental design is not ideal to answer the research question, we find that algorithms which perform well under increased offset diversity are capable of efficiently optimizing offset. We maintain that, for an offset-diverse set of targets, offset diversity will likely lower the RMSE. We also find that the SciPy algorithms don't optimize the offset at all due to an incorrect implementation. Note that this error also affects the results of other experiments for these two algorithms, albeit to a smaller extent, as

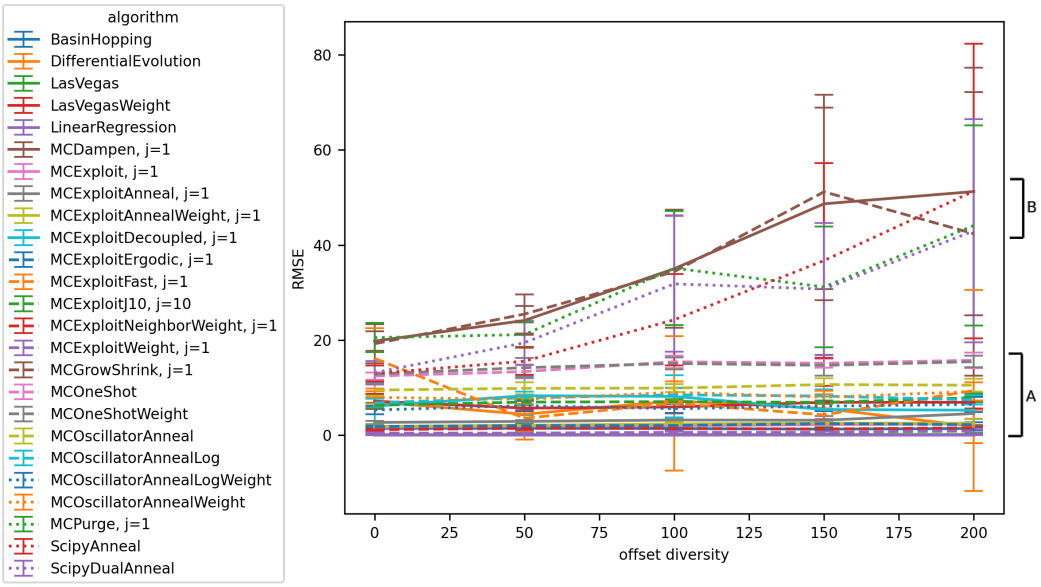


Figure 27: RMSE as a function of offset diversity for a broad set of algorithms. Mean and standard deviation over 10 runs. See Table 18 for tabular data.

the default offset diversity is 20. We don't consider this to be problematic as Exploit Anneal implements the same algorithm as SciPy Anneal, and does so correctly.

10.3 Target function

We find that the approximation of the real world target signals is not possible using the hybrid generator. We are able to approximate real world signals with the Python generator, however those results don't realistically reflect the properties of the VO_2 material. For this reason, Figure 28 shows the RMSE for a set of synthetic target functions. We note that the noise targets appear to be the most difficult to approximate. However, the similarity in mean RMSEs and a large standard deviations across target functions, makes it difficult to draw further conclusions.

Therefore we repeat the experiment with a smaller set of algorithms shown in Figure 29. The tested algorithms are Las Vegas, Exploit, Exploit Weight and linear regression. For each algorithm, 5 runs are sampled, resulting in 20 RMSE values per target function, again multiplied by 4 for targets with various frequencies as described in Section 4. Compared to Figure 28, the means and standard deviations are smaller. This is explained

by the selection of algorithms which are better at approximating the target functions.

Most notably, the Gaussian noise target remains the most difficult to approximate, similar to the previous figure. Additionally, the square wave is the second most difficult target, this is likely due to the sharp edges of the square wave. In comparison the edges of a triangle wave are smoother. Since the sawtooth and inverse sawtooth targets also have a sharp edge, they too may be more difficult to approximate than the triangle wave.

The sine wave yields a lower RMSE in comparison, also being smooth. The smoothened noise targets have the lowest RMSE in this experiment. This could be explained by the the smoothening of the signals with a 10 point average, which acts as a low-pass filter. Given that the sampling frequency is 2.5 MHz, the noise targets are filtered to a frequency of 250 kHz which is within the frequency band of a single oscillator, see Section 5.2.1. Since the other constant frequency targets are drawn from a range of frequencies, starting at 1 kHz, the low RMSE on the smooth noise targets may be primarily explained by frequency rather than the shape of the target function.

Lastly, the damped chirp target yields a lower RMSE than the chirp target although it also has a varying amplitude. This could be explained by the damped signal being close to zero towards the end of the signal, which may be easier to approximate.

We hypothesized that the difficulty of the target function correlates with increasing RMSE. Our results confirm this hypothesis.

We find that smooth functions are approximated well while sharp functions were more difficult. Furthermore, frequency modulated signals are more difficult to approximate than constant frequency signals. We can't show that amplitude modulation signals are more difficult than constant amplitude signals. The damped chirp is more easily approximated than the chirp signal due to its zero deflection towards the end of the signal. A constant frequency sine with growing amplitude may be a better target function to test the effects of amplitude modulation.

Lastly we find that real world target signals can't be approximated with the hybrid pipeline. We believe this to be due to (1) their low frequency in comparison to the VO_2 frequency band and (2) the staircase frequency response of the VO_2 oscillators, see Figure 11 (right).

10.4 Algorithms

Figure 30 shows the RMSE as a function of the number of perturbations. The data is the same as presented in Figure 20, however in this section we

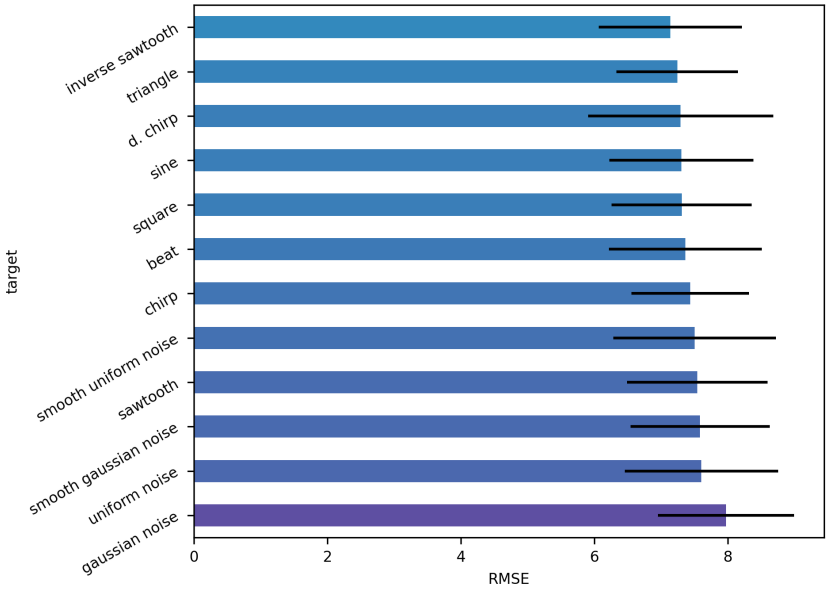


Figure 28: RMSE as a result of the target function. Aggregate of 25 algorithms and 3 runs per algorithm.

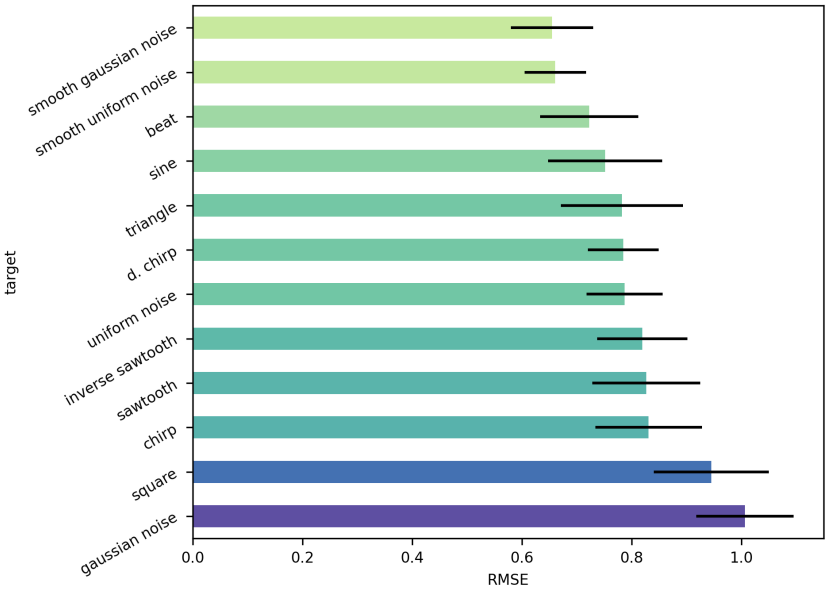
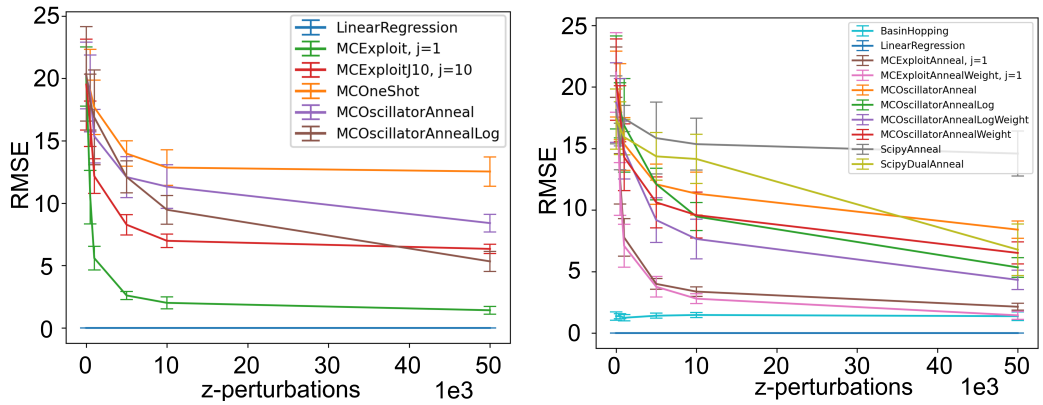
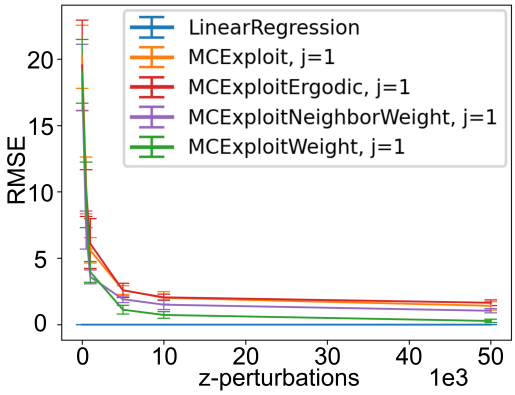


Figure 29: RMSE as a result of the target function. Aggregate of 4 algorithms and 5 runs per algorithm.

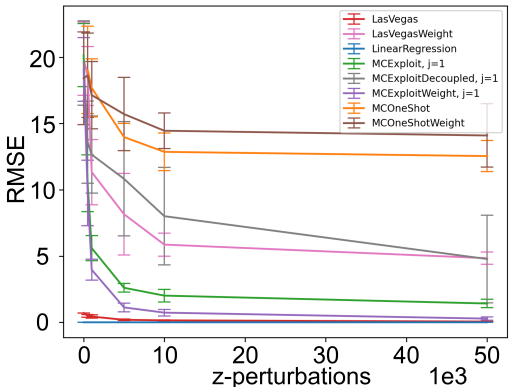


(a) Simultaneous perturbations.

(b) Annealing algorithms.



(c) Ergodicity and regularization.



(d) Gain-only perturbation.

Figure 30: RMSE as a function of the number of perturbations Z applied to an ensemble for varying subsets of algorithms. Data as in Figure 20.

focus on subsets of algorithms and discuss differences. In each figure linear regression is included as a baseline.

10.4.1 Number of perturbed oscillators

Figure 30a shows the RMSE for stochastic algorithms with varying numbers of simultaneously replaced oscillators. The One Shot algorithm perturbs all oscillators at each iteration. Meanwhile the Exploit algorithm perturbs one oscillator per iteration. Intermediate variations are the Exploit J=10 algorithm which perturbs 10 oscillators per iteration and the Oscillator Annealing algorithms which use a temperature schedule to decrease the number of perturbed oscillators per iteration.

We find that Exploit performs best. It is the fastest algorithm to converge

to a low RMSE and it continues to improve at 50 k perturbations. Given sufficient perturbations, the Oscillator Anneal Log and its linear counterpart follow. The Exploit J=10 algorithm stops improving after 10 k perturbations. It is overtaken at 40 k perturbations by the Oscillator Anneal Log algorithm. The worst performing algorithm is One Shot.

10.4.2 Annealing algorithms

Figure 30b shows the RMSE for stochastic algorithms with varying annealing schedules. The best performing stochastic algorithm is Basin Hopping. Exploit Anneal Weight achieves a similar RMSE at 50 k perturbations; Exploit Anneal Weight doesn't use a second local optimization step as Basin Hopping. The Exploit Anneal algorithms decrease the acceptance probability of worse solutions as the temperature decreases and perturb one oscillator at a time. The Oscillator Anneal algorithms decrease the number of perturbed oscillators as the temperature decreases. Here we find that the logarithmic schedule performs better than the linear one. Yet, the Oscillator Anneal algorithms all perform worse than the Exploit Anneal algorithms. For all annealing algorithms, the weight and offset perturbing variant outperforms the weight, offset, frequency and phase perturbing variant. The SciPy Anneal algorithms perform worst up to 10 k perturbations. Beyond, Dual Annealing improves, using a second local optimization step.

10.4.3 Perturbed oscillator property

Figure 30d shows the RMSE for stochastic algorithms with varying perturbed oscillator properties. Exploit perturbs oscillators' frequency, phase, offset and gain. Meanwhile, Exploit Weight perturbs gain and offset starting out with a random ensemble. The same principle applies to One Shot and One Shot Weight and the Las Vegas and Las Vegas Weight algorithms.

There appears to be no clear trend, as Exploit Weight achieves lower RMSE than Exploit. Meanwhile One Shot Weight performs worse than One Shot and Las Vegas Weight performs worse than Las Vegas. The Exploit Decoupled algorithm randomly alternates between weight-offset perturbations and frequency-phase perturbations. It performs worse than Exploit Weight with a larger standard deviation. Positively, the rate of RMSE decrease at 50 k perturbations is larger compared to other algorithms.

10.4.4 Acceptance criterion

Figure 30c shows two additional variations of the Exploit algorithm. Exploit Ergodic introduces an acceptance criterion which accepts perturbations

that increase the RMSE without a temperature schedule. Exploit Neighbor weight samples neighbor gains, forms a Gaussian distribution and samples the distribution to obtain a new gain. Here, neighbor refers to a neighboring oscillator in the ensemble.

We find that Exploit Ergodic performs similarly to the Exploit algorithm. Furthermore, Exploit Neighbor Weight performs similarly to Exploit Weight. Both variants are outperformed by the Exploit Weight algorithm.

10.4.5 Discussion on Algorithms

In Section 10.4 we test the performance of various algorithms and compare them by various properties. Although Basin Hopping and Differential Evolution perform well, they are outperformed by the Las Vegas and Exploit Weight algorithms. Both algorithms are also more suitable for hardware implementation as they require less state to be maintained. While gain-offset perturbation is compared to gain-offset-frequency-phase perturbation, the effect of phase perturbation alone remains to be tested. Futhermore, the effect of perturbation strength is only tested to a limited extend with the Neighbor Exploit and Grow Shrink algorithms; a more thorough investigation may be worthwhile.

11 Appendix C

Listing 9: Possible typographical error in Maffezzoni et al. (2015)

```
"""This Python script explores equations (13) and (14) in Maffezzoni et al., 2015.
The purpose is the check whether equation (14) has a typo in it.
Namely, the term +E_eq^met could possibly be +E_eq^ins.

The evidence points towards equation 14 indeed being wrong.
The minimum and maximum voltages of the two phases should meet.
They do this when correcting equation 14 with the summand +E_eq^ins.
They don't meet when the summand is +E_eq^met.
"""

import numpy as np
import matplotlib.pyplot as plt

def f_metal(t, V_min, U_met, tau_met):
    """equation (13) from the paper"""
    return (V_min - U_met) * np.exp(-t / tau_met) + U_met

def f_insulator_paper(t, V_max, U_ins, tau_ins, T_met, U_met):
    """equation (14). this is how the insulator equation is described in the paper"""
    return (V_max - U_ins) * np.exp(-(t-T_met) / tau_ins) + U_met

def f_insulator_hypothesis(t, V_max, U_ins, tau_ins, T_met):
    """this is a hypothesis for the correct insulator equation """
    return (V_max - U_ins) * np.exp(-(t-T_met) / tau_ins) + U_ins

# the values are loosely inspired by the paper but disregard timescales
t = np.linspace(0, 100, 1000)
V_max = 13 # max voltage of the output oscillation
V_min = 11 # min voltage of output oscillation
U_met = V_max +1 # voltage the capacitor tends to during metal phase of vo2
U_ins = V_min -1 # voltage capacitor tends to during insulator phase of vo2
R = 47 # resistor in RC circuit
C = 300e-3 # capacitor in RC circuit
tau_met = R*C # RC time constant during metal phase
tau_ins = 1*tau_met # RC time constant during insulator phase
T_met = 1 # period of metal phase
```

```
37
38 plt.plot(t, f_metal(t, V_min, U_met, tau_met), label='metal')
39 plt.plot(t, f_insulator_paper(t, V_max, U_ins, tau_ins, T_met,
       ↪ U_met), label='insulator paper')
40 plt.plot(t, f_insulator_hypothesis(t, V_max, U_ins, tau_ins,
       ↪ T_met), label='insulator test')
41 plt.legend()
42 plt.show()
```

Listing 10: Approximation of a sine function by periodic non-trigonometric functions.

```
1  """
2  This Python code demonstrates approximation of a sinusoid
3  with a linear combination of non-trigonometric functions of
       ↪ varying phase and frequency
4  using linear regression.
5  """
6  import matplotlib.pyplot as plt
7  import numpy as np
8  from scipy import signal
9  import matplotlib.pyplot as plt
10 from sklearn.linear_model import LinearRegression
11
12 RNG = np.random.default_rng()
13
14 def compute_weighted_sum(X: np.ndarray, coef: np.ndarray,
       ↪ intercept: float) -> np.ndarray:
15     """generate approximation of target, y"""
16     fit = np.sum(X.T * coef, axis=1) + intercept
17     return fit
18
19 def compute_rmse(p: np.ndarray, t: np.ndarray) -> float:
20     """
21     Compute root mean square error (RMSE) between prediction and
           ↪ target signal.
22     """
23     rmse = np.sqrt(((p-t)**2).mean())
24     return rmse
25
26 def regress1d(p: np.ndarray, t: np.ndarray):
27     """apply linear regression"""
28     r = p.T
29     reg = LinearRegression().fit(r, t)
30     return reg
31
32 t = np.linspace(0, 10, 50000)
33 target = np.sin(2 * np.pi * 1 * t)
34 plt.plot(t, target, label='target')
```

```
35
36 signals = []
37 for i in range(1, 100):
38     f = RNG.uniform(0.1, 1)
39     phase = RNG.uniform(0, 2) * np.pi
40     # s = signal.square(phase + 2 * np.pi * f * t)
41     s = signal.sawtooth(phase + 2 * np.pi * f * t, width=0.1)
42     signals.append(s)
43
44 matrix = np.array(signals)
45 reg = regress1d(matrix, target)
46 weights = reg.coef_
47 offset = reg.intercept_
48 pred = compute_weighted_sum(matrix, weights, offset)
49 rmse = compute_rmse(pred, target)
50 print(rmse)
51
52 plt.plot(t, pred, label='pred')
53 plt.legend()
54 plt.show()
```