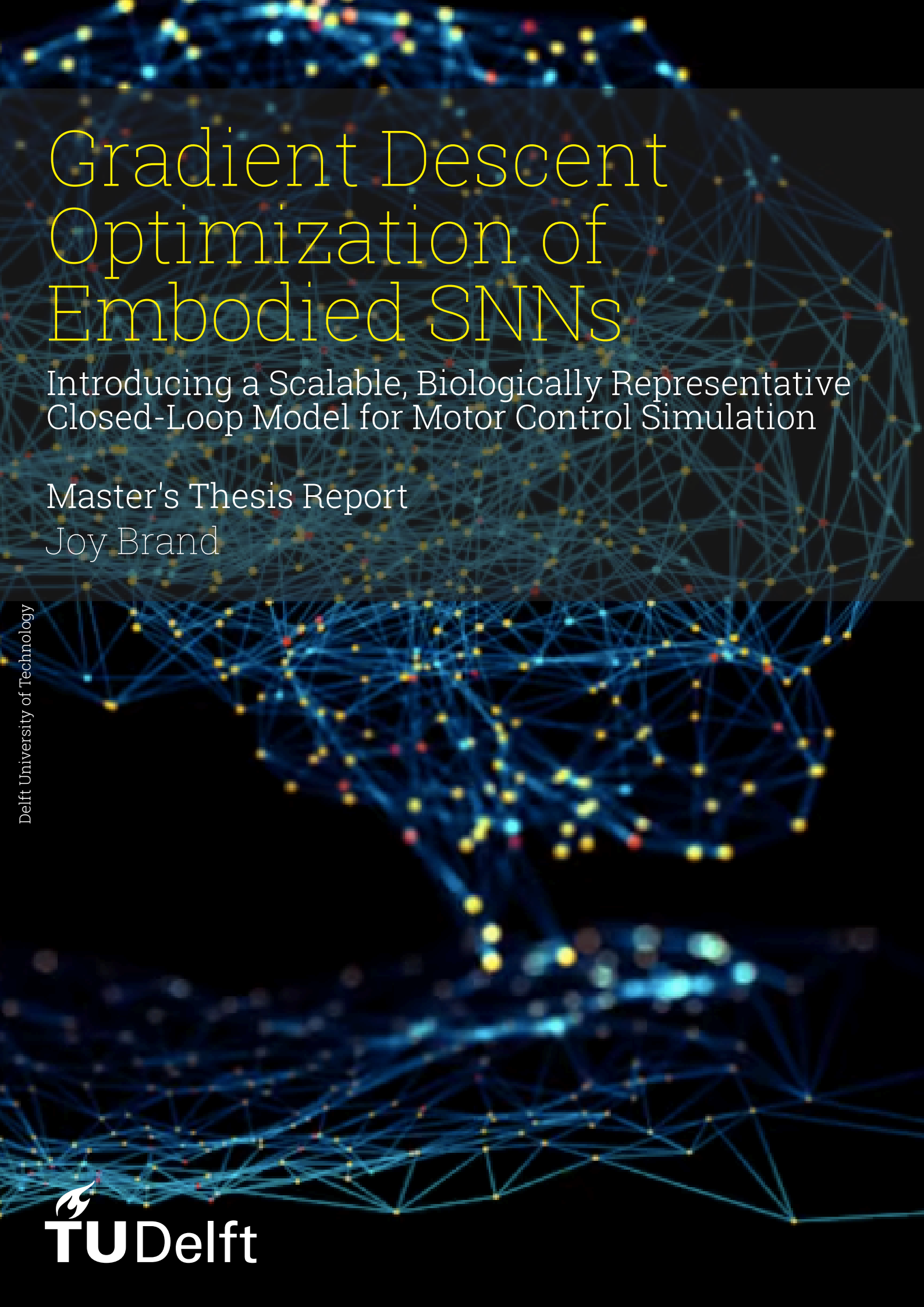

Gradient Descent Optimization of Embodied SNNs

Introducing a Scalable, Biologically Representative Closed-Loop Model for Motor Control Simulation

Master's Thesis Report

Joy Brand

Delft University of Technology

TU Delft

Gradient Descent Optimization of Embodied SNNs

Introducing a Scalable, Biologically Representative Closed-Loop Model for Motor Control Simulation

by

Joy Brand

Student Name	Student Number
Brand	4716795

Supervisor: Dr. W. Mugge, TU Delft & Dr. M. Negrello, Erasmus MC
Daily Supervisor: L.P.L. Landsmeer, Erasmus MC & E.M. Fernandez Santoro, Erasmus MC
Project Duration: February 15, 2024 - Month, Year
Faculty: Faculty of Biomedical Engineering, Delft

Cover: Tam Nguyen. What is a neural network? A computer scientist explains. en-EUROPE. Dec. 2020. [33]

TU Delft

Preface

Neurons are fundamental units of the nervous system, processing and transmitting information. They enable brain-muscle communication, allowing organisms to interact with their environment. The cerebellum plays a key role in motor control by fine-tuning movements and coordinating precise actions. Understanding its relationship with the body is crucial for unraveling neural functions and related impairments, but its complex neural networks present significant research challenges.

Researchers study brain activity using *in vivo*, *in vitro*, and *in silico* methods. *In vivo* experiments record directly from living brains but offer limited control over variables and difficulty isolating individual neuron functions. *In vitro* experiments use cultured neurons in controlled settings but face technical challenges and limited reproducibility due to neuron viability. To overcome these challenges, a third approach—*in silico* experiments—has emerged, transitioning from traditional electrophysiological methods into the realm of computational neuroscience. *In silico* experiments employ computational models to simulate neural activity, overcoming many biological constraints. This approach allows for hypothesis testing, prediction of neural behavior, and integration of large data sets, complementing traditional methods and enhancing our understanding of brain function.

Embodied Spiking Neural Networks (SNNs) simulate neural circuits by modeling neurons that communicate through discrete spikes, capturing temporal neural dynamics more accurately than traditional models. Accurate parameterization of these models becomes complex as they incorporate detailed brain-environment interactions. Traditional optimization algorithms struggle with memory limitations and scalability. Gradient descent optimization offers an efficient solution by adjusting thousands of parameters to minimize errors between simulated and observed behaviors. In the context of embodied brain simulations this approach could enable researchers to test and validate hypotheses in scalable embodied brain models encompassing the mechanisms of motor control.

In collaboration with NeuroComputing Lab at the Erasmus Medical Center, which specializes in the brain dynamics of the cerebellum, the current Master's Thesis Project focuses on developing and validating a novel gradient descent optimization approach for parameter tuning in an embodied SNN model. The project involves building a simulation to validate the efficacy of gradient descent in optimizing SNNs within an embodied context. A simulated model of a mouse forearm, including both flexor and extensor muscles, is created to serve as the environment for this proof of concept, providing a foundation for more comprehensive simulations that integrate the cerebellum and the entire musculoskeletal system of the mouse. By aiming to simulate the intricate mechanisms of motor control with high precision, this research aspires to make contributions to the field of computational neuroscience.

The thesis is organized into five key sections: the introduction provides an overview of the scientific background, including the significance of gradient descent optimization and the role of embodied SNNs in modeling the processes of motor control; the methods section explains the theories and techniques that are used and details the implementation of it; the results section presents the outcomes of the subsystems of the simulations and the full system, examining informative gradient calculations, biological representative behaviour and computational performance of the system; the discussion summarizes the project's outcomes and contributions to the field and outlines directions for future research; and finally, the conclusion.

Joy Brand
Delft, September 2024

Abstract

This study introduces an approach to optimizing large-scale embodied spiking neural networks (SNNs) for simulating the brain in a closed-loop environment, crucial for validating theoretical neuroscience hypotheses about the brain-body relationship. Accurately modeling this relationship at scale allows for the simulation of neural plasticity, temporal dynamics, and spike timing. Traditional parameter tuning methods are impractical for complex SNNs due to their non-differentiable nature and computational challenges. To address these issues, we apply gradient descent optimization with forward propagation through time, enhanced by surrogate gradient techniques, enabling efficient and scalable SNN tuning. We demonstrate this approach with a proof-of-concept system comprising a three-layer leaky-integrate-and-fire neural network with recurrent connections, integrated with a 2D musculoskeletal model using Hill-type muscle representations. All components are fully differentiable, allowing for gradient calculations through the system. The results demonstrate that the weights are updated and the performance of the embodied SNN increases as it learns to stabilize the arms angle to zero degrees. Together with the improved motor control behavior, these results indicate that the optimization approach can handle the non-linearities of the muscle model. Spike activity show representative spike firing frequencies during the training process. The system operates within zero memory constraints and has an easily adjustable and well structured software architecture enabling scalability of the system. These findings support gradient descent optimization with forward propagation through time as a viable and scalable approach for embodied SNNs in motor control simulations, paving the way for more extensive applications like closed-loop cerebellum modeling.

Contents

Preface i

Abstract ii

1 Introduction 1

2 Methodology 4
2.1 Gradient Descent Based Learning . . . 4
2.2 Brain Model . . . 4
2.2.1 Leaky Integrate-and-Fire Neuron Model . . . 4
2.2.2 Surrogate gradient via SuperSpike formalism . . . 5
2.3 Musculoskeletal system . . . 5
2.3.1 Body model . . . 5
2.3.2 BRAX environment . . . 6
2.3.3 Activation dynamics . . . 6
2.3.4 Integration of stimulation signal in activation dynamics . . . 7
2.3.5 Hill-type Muscle Model . . . 7
2.4 Integration and Parameter Mapping of the Musculoskeletal System and SNN . . . 8
2.4.1 Angle Reformulation . . . 8
2.4.2 Force to Torque Conversion . . . 8
2.4.3 Determination of Muscle Length . . . 8
2.4.4 Tendon-Muscle Ratio . . . 8
2.5 Implementing Optimization Strategies for training SNNs . . . 10
2.5.1 Numerical Computing Library: JAX . . . 10
2.5.2 Optax's Stochastic Gradient Descent optimizer . . . 10
2.5.3 Forward Propagation Through Time . . . 10
2.6 Full model Design . . . 11
2.7 Hardware . . . 12

3 Results 15
3.1 Validation gradient descent optimization in the embodied SNN . . . 15
3.2 Learning behavior of the embodied SNN . . . 16
3.3 Verifying firing frequencies . . . 17
3.4 Scalability of the system . . . 18
3.5 Neural motor control across different muscle stuffiness's . . . 19

4 Discussion 22

5 Conclusion 25

References 26

A Steady state behavior and learning behavior of the SNN in isolation 31

1

Introduction

Motor control relies on the intricate interaction between the brain, muscles, skeleton, and surrounding environment [38]. Together they form a complex network to enable coordinated movements [38]. Understanding the underlying mechanisms of motor control is crucial for diagnosing neurological diseases and developing therapies or interventions that restore or compensate for motor function [16]. We already have great insights into the individual mechanisms of motor control such as reflex arcs [34]. Nevertheless, aspects such as integration of sensory feedback, coordination, and higher-level regulation of movements remain not fully understood [9, 21, 27]. This is due to technical limitations, ethical constraints, and the high level of complexity [26, 30]. Computational models are useful to gain insight into motor control [34]. By reverse engineering the behavior of biological processes, researchers can provide precise control over experimental parameters and test hypotheses to complement where *in vivo* and *in vitro* experiments fall short.

Computational models often focus on isolated neural circuits or specific muscle activations, aiming to understand how individual components contribute to movement [4, 13]. Spiking neural networks (SNNs) are commonly used in computational models to simulate the biological processes of the brain's neural circuits. SNNs closely mimic the biological neural spiking behavior by incorporating the event-based nature of spikes and synaptic states [37]. It is particularly popular because of its low computational cost.[15, 44] Another model of this type, the Hill model, mimics the dynamics of muscle contraction and is considered relatively simple [32]. Even though it contains nonlinearities it is still effective in describing muscle behavior in most applications [32]. Decoupled brain and muscle models offer valuable insights into individual mechanisms of motor control. However, motor control is a dynamic process that relies on continuous feedback between the brain, body, and environment [38]. Neglecting or overly simplifying these relationships can paint a distorted picture of reality. To fully capture motor control, the brain and muscles must be connected in a closed-loop environment, capable of scaling to encompass all systems involved. In short, scalable embodied brain simulations are in demand.

As we model more components that are involved in motor control we increase the number of parameters leading to a more complex system [34, 35]. In general, to create computational models, observed behavior or experimental data are translated into mathematical equations or algorithms to replicate biological processes. In such models, parameters represent key variables —such as synaptic strengths in neural networks— that determine the behavior and dynamics of the system. For example, the body adjusts its actions through regulation of the synaptic strength —a process known as synaptic plasticity— to achieve a task [47]. Tuning of such parameters allows the model to produce realistic outputs that match empirical observations. Parameter tuning can be done manually, i.e. iteratively running simulations, then manually updating parameters based on the output. However, this approach becomes infeasible when dealing with the large number of parameters in complex models [40]. This makes automated parameter tuning essential.

Hence, the focus has shifted towards automated optimization techniques that in general adjust parameters to optimize a predefined objective function to achieve a specific task or criterion [14, 15]. In the field

of computational neuroscience, tuning is commonly performed using zero-order algorithms [2]. These algorithms rely on direct evaluation of the objective function without requiring gradient information, adjusting parameters iteratively by evaluating their performance at each step [8, 45, 46]. However, this process is known to suffer from *the curse of dimensionality*; it only scales to small dimensional problems and becomes computationally infeasible with many parameters, often requiring large-scale computing resources [2]. On the other hand, derivative-based methods, known as a first-order algorithm, offer more efficient and scalable approaches to parameter tuning. Derivative-based optimization allows to adjust multiple parameters simultaneously, allowing to tune large numbers of parameters[2] in large-scaled computational models.

To capture the underlying mechanisms of motor control, we need scalable embodied brain models that allow gradient-based automatic parameter tuning, leading to the following requirements. These models must incorporate a neural network that replicates neural spiking behavior, a musculoskeletal body that replicates actual physical measurements of joints and muscles, and a virtual environment that simulates the kinematics of the body allowing it to perform tasks and interact with external forces. Additionally, a muscle model is required to simulate the nonlinear dynamics of muscle force generation depending on the state of the body, muscle and neuronal input. Therefore muscle activation dynamics and real-time sensory feedback must be included. Furthermore, parameter knowledge must be systematically gathered, with hyperparameter tuning employed to optimize broader model settings, and a comprehensive gradient-based parameter tuning framework must be integrated for automated tuning[15]. To ensure scalability, the model must be capable of simulating long periods and scaling up while maintaining acceptable simulation speeds. Together, these all components must be differential to allow gradient-based optimization and should enable a biologically plausible simulation of motor control within a closed-loop environment. In prior art, we find multiple works aiming to build an embodied brain simulation although none meet all these model requirements [10, 34, 35, 40],

Gradient-driven optimization is already widely used in optimizing artificial neural networks (ANNs), but of limited use in optimizing SNNs [29, 36]. Gradient-driven optimization in ANNs is particularly used in the field of image processing and reinforcement learning [29]. Optimizing SNNs is primarily based on biologically inspired learning rules, such as Spike-Timing-Dependent Plasticity (STDP), rather than gradient-based methods [9, 21, 27]. Unlike ANNs that use continuous activation functions, SNNs contain discrete spiking events, making the system non-differentiable [31, 54]. Gradient descent requires the system to be differentiable due to its reliance on gradient calculations, which makes the discrete nature of SNNs unsuitable. Therefore, methods such as surrogate gradients have been proposed to approximate gradients, overcoming the non-differentiable nature of spiking neurons and enabling gradient calculations through SNNs [31, 54]. Studies have shown that gradient descent effectively optimizes both the micro-level dynamics, such as individual neuron spiking, and the macro-level behavior, including task-level performance, in SNNs [15, 18, 24, 25].

Furthermore, SNNs incorporate temporal dynamics which play a role in information processing and learning, requiring optimization techniques to account for temporal dependencies and delays in neural responses [1, 31]. To manage the temporal dynamics, backpropagation through time (BPTT) and forward propagation through time (FPTT) are developed. Here, FPTT is considered as a combination of forward propagation with BPTT. BPTT involves ”unrolling” the network through time, ensuring accurate gradient calculations that capture long-term dependencies. However, this comes with high memory consumption, as the entire sequence of gradients must be stored. In contrast, FPTT calculates gradients incrementally, requiring only the information from the current iteration, making it more memory-efficient. However, this approach can lead to less accurate optimization outcomes, as it does not retain the entire history of gradients. Recent work has shown FPTT outperforming BPTT in optimizing recurrent networks [20] and SNNs on various benchmark tasks, promising memory-friendly, online training for long sequences [51].

Libraries exist that allow for automatic gradient calculation, including Theano, TensorFlow, PyTorch, and JAX. Among these libraries, JAX offers the most flexibility in composing automatic gradients and performs well across a wide range of hardware [42]. Additionally, a gradient processing and optimization library named Optax is developed for JAX to optimize parametric models such as SNNs. Furthermore, BRAX is a library for rigid body simulation written in JAX. This physics engine is fully differentiable, allowing users to take advantage of JAX’s automatic differentiation features and integration of Optax.

BRAX offers prepackaged environments while also supporting custom ones. However, despite its compatibility with JAX and Optax, BRAX faces limitations in simulating muscle dynamics, as it currently does not support muscle actuators [11].

To conclude, gradient descent optimizations have been found viable in optimizing SNNs [15, 18, 24, 25]. However, gradient-based optimization encounters memory issues for simulation of long-sequences. FPTT has the potential to address memory constraints, yet it remains under-researched in SNNs [20]. Current tools available allow investigating gradient-driven optimization of SNNs including body physical dynamics, but lack mimicking muscle dynamics [11]. Thus, while the first steps toward gradient descent optimization in SNNs have been made and available tools provide the possibility of simulating embodied brain models to some extent, gradient descent optimization with FPTT has not been explored in embodied brain simulation while promising to understand the underlying mechanisms of motor control.

The goal of the current study is to demonstrate gradient descent optimization through FPTT in embodied SNNs. We approach this by developing an embodied brain model from scratch that retains all required components while simplifying it to its fundamental form. SNN with Leaky Integrate-and-Fire (LIF) neuron model is used and a customized musculoskeletal system replicating a two-dimensional arm including one joint is created in BRAX. To incorporate the muscle dynamics, a self-defined Hill-type muscle model is integrated into the full model. All components of the full model are differentiable to enable gradient calculations. The libraries JAX and Optax are used to facilitate the construction of the optimization framework in an easy and computationally efficient manner. We design this model based on rodent measurements.

The current paper aspires to create a scalable and biological representative model to lay the groundwork for a platform that allows researchers to test and validate the underlying mechanisms of motor control within a gradient-driven optimized closed-loop environment. The term ‘scalable’ means that the system must operate without memory constraints—implying no increased memory usage during simulation—and must have a robust software architecture. In this work, “biologically representative” refers to realistic firing frequencies, specifically between 0 and 100 Hz. This is determined by dividing the number of spikes by the time in seconds. In addition, the muscle model must exhibit non-linear behavior including passive and active force generation. Next to assessing gradient descent optimization via FPTT in the embodied SNN, the system is also examined based on its scalability and biological representation.

This work makes the following contributions to science:

- Provide a full differentiable closed-loop embodied SNN model.
- Provide an optimization framework to automatically tune parameters within a zero memory constraint.
- Provide an optimization framework that can handles the non-linearity of the muscles.
- Provide a class-based structured architecture that enable adjusting and scalability of the system.
- Provide insights into the motor control behavior and its performance of optimized SNNs across different muscle stiffnesses

The current paper is structured as follows. The methodology outlines the theories and methods used to build the embodied SNN. This is followed by a detailed description of the full model design and the rationale behind specific design choices. The results section first validates gradient descent optimization in the complete system, then explores motor control behavior as the system learns. Where after it verifies firing frequencies and evaluates system scalability, concluding with an analysis of motor control performance across different muscle stiffnesses. Finally, the discussion addresses our findings, biological plausibility limitations, and recommendations for further research, which is summarized by the conclusion.

2

Methodology

This study trained a SNN of LIF neurons connected to a musculoskeletal system, consisting of a vertical arm connected to a hinge joint operating in a two-dimensional plane with the flexor and extensor muscles connected on opposite sides of the arm, using gradient descent optimization. The libraries JAX, Optax, and BRAX were utilized for numerical computing and automatic differentiation, gradient processing and optimization, and simulation of the physics of our body, respectively. The aim of this study is to develop a differentiable, scalable, and biologically representative closed-loop simulation that provides a simplified embodied brain model for investigating gradient descent optimization through FPTT. The parameters and measurements of the model are based on rodent models. This section describes the architecture of our system and the choices made during this study.

2.1. Gradient Descent Based Learning

Gradient descent is used as an optimization technique to strengthen or weaken neural connections based on their contribution to error reduction, emulating the biological process of synaptic plasticity. The process begins with an initial set of parameters and a predefined loss function, which measures the error—the difference between the model's output and the actual target values. Using the gradient of the loss function, the algorithm determines the direction and magnitude of changes needed to reduce the error. It adjusts the model parameters in the opposite direction of the gradient, "descending" along the slope of the loss function. This process is performed iteratively, refining the parameters to find a minimum and, in turn, optimizing the model's performance [39]. This update rule can be mathematically represented as:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t; x_i, y_i) \tag{2.1}$$

where θ_t represents the parameters at iteration t, η is the learning rate, and $\nabla_\theta L\theta_t; x_i, y_i)$ is the gradient of the loss function L with respect to θ_t, computed using a single training example (x_i, y_i).

2.2. Brain Model

2.2.1. Leaky Integrate-and-Fire Neuron Model

A leaky integrate-and-fire (LIF) model is implemented for the simulation of neurons due to its biological representation of neuronal behavior and computational efficiency [37]. The LIF model mimics the electrical characteristics of a neuron, where the membrane potential integrates incoming synaptic inputs over time. When the membrane potential reaches a certain threshold, the neuron fires (a spike) and its potential is reset. The "leaky" aspect of the model represents the natural decay of the membrane potential due to passive ion flow, mimicking the neuron's tendency to return to its resting state. The LIF model consists of two main components: an equation describing the evolution of the membrane potential $U(t)$, and a mechanism to generate spikes.

The continuous-time linear differential equations describing the evolution of the membrane potential

$U(t)$ and synaptic current $I(t)$ are:

$$\tau_{\text{mem}} \frac{dU(t)}{dt} = -U(t) + RI(t) \tag{2.2}$$

$$\tau_{\text{syn}} \frac{dI(t)}{dt} = -I(t) + I_{\text{syn\_in}} \tag{2.3}$$

where τ_{mem} is the membrane time constant, R is the membrane resistance, τ_{syn} is the synaptic time constant, and $I_{\text{syn\_in}}$ is the input synaptic current.

To make Equations 2.2 and 2.3 suitable for iterative computation, they are discretized using the exponential Euler method. The update rule for the membrane potential U and I is given by:

$$U_{t+1} = (1 - S_t) \cdot (\beta U_t + I_t \Delta t) \tag{2.4}$$

$$I_{t+1} = \alpha I_t + I_{\text{syn\_in}} \tag{2.5}$$

where Δt is the discrete time step, $\beta = e^{-\frac{\Delta t}{\tau_{\text{mem}}}}$ and $\alpha = e^{-\frac{\Delta t}{\tau_{\text{syn}}}}$, including τ_{mem} a membrane time constant and τ_{syn} a synaptic time constant.

The LIF model describes action potentials as events, ignoring the shape of the action potential. If the membrane potential exceeds the threshold criterion, a spike is generated:

$$S_t = H(U_t - v_{\text{th}}) \tag{2.6}$$

where H is the Heaviside step function and v_{th} is the threshold voltage for a spike. After a spike, the membrane potential is reset to zero.

2.2.2. Surrogate gradient via SuperSpike formalism

The Heaviside step function used in Equation 2.6 introduces non-differentiability (discontinuous function) into the spiking mechanism. To address this, surrogate gradients are implemented using the SuperSpike formalism, inspired by the work of Zenke and Ganguli [54]. Surrogate gradients approximate the gradient of the Heaviside step function, making it a continuous and differentiable function, essential for gradient descent optimization.

The Heaviside function $H(x)$ is defined as:

$$H(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.7}$$

Equation 2.7 is approximated as:

$$\frac{dH}{dx} \approx \frac{1}{(|x| + 1)^2} \tag{2.8}$$

This approximation ensures that the Heaviside function is a continuous function, enabling the gradient calculations required for gradient descent optimization.

2.3. Musculoskeletal system

The musculoskeletal system contains two components; the body and the muscle model. The muscle model is divided into the activation dynamics and contraction dynamics of the muscle. The contraction dynamics are modeled as a Hill-type model [48, 52] (Figure 2.2A.). The body is simulated within a BRAX environment which is discussed in section 2.3.2. Since BRAX does not support muscle actuators, we developed our own differentiable muscle model and integrated it with BRAX, allowing us to leverage the benefits of the differentiable physics engine. Section 2.3.3 and 2.3.5 describes the muscle model and section 2.4 the integration of it in the full model.

2.3.1. Body model

The body consists of a vertical arm connected to a hinge joint that is actuated by two muscles (Figure 2.1). This model operates in a two-dimensional plane, with the hinge joint allowing a range of motion $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$. Two muscles, each attached to the opposite sides of the vertical arm representing a flexor and an extensor. The muscles are structurally identical but mirrored across the y-plane, with their position and length determined by the connection points on the arm ($con1$) and the ground ($con2$).

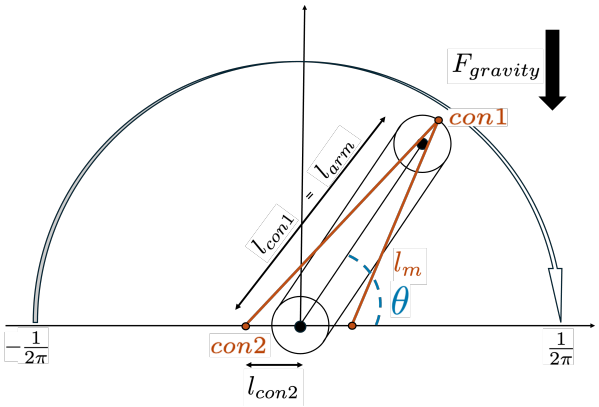

Figure 2.1: Illustration of the body model showing the muscle connections. The arm is depicted in black, forming an angle θ with the ground, and has a range of motion between $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$. The arm's movement is influenced by the force of gravity ($F_{gravity}$), which acts downward. The muscles are shown in red, with connection points on the arm ($con1$) and the ground ($con2$). The lengths of the lines extending from the connection points (l_{con1} and l_{con2}) are used, along with the angle θ, to calculate the muscle length (l_m).

2.3.2. BRAX environment

An environment in BRAX is created to simulate the dynamics of our body discussed in section 2.3.1. BRAX is used as physics engine because it is written in JAX, allowing to make use of the automatic differentiation and gradient processing via JAX, essential for gradient descent optimization [11]. The body, or in other words, the skeleton is described in a xml file that contains all the measurements and connections of the model. The joint of the arm is modeled as a torque actuator, since BRAX does not support muscle actuators. Here the joint properties such as the range of motion, damping, armature (inertia) and gain and gravity are specified (see Table 2.1 for specifications). These parameters are tuned based on the time that the arm falls down when there is no torque applied. This is approximately 1.5 seconds. The simulation environment in BRAX includes an action space, observation space, and a starting state. Each time step, the body takes an action after which the state of the system is obtained. The observation space consists of the joint's angle and angular velocity. The starting state of the arm is at an angle of 0 and there is a reset state implemented to set the state to a randomly determined angle within the predefined range with uniform random noise added for stochasticity. In addition, a maximum angle is defined. If the arm exceeds this angle before the maximum number of timesteps is reached, the epoch terminates.

2.3.3. Activation dynamics

The activation dynamics of a muscle is modeled using first-order differential equations to describe the relationship between neural activation and muscle force generation (Fig. 2.2B). The activation and deactivation time of the muscle differs, therefore there are two time constants specified τ_{act} and τ_{deact}. For modelling the muscle activation the equation is used:

$$a = \begin{cases} stim - \tau \frac{dt}{\tau_{act}}, & \text{for } stim > a \\ stim - \tau \frac{dt}{\tau_{deact}}, & \text{for } stim \leq a \end{cases} \tag{2.9}$$

where a is the muscle activation level, $stim$ is the stimulation signal reflecting the neural input received from the SNN, τ_{act} is the activation time constant and τ_{deact} is the deactivation time constant that represents the speed of the muscle's response to the neural input.

To make Equation 2.9 suitable for iterative computation, they are discretized, to enable gradient computation. This results in the muscle activation update rule:

$$a_{t+1} = stim_t \cdot (1 - (1 - a_t \cdot e^{-\frac{t_{pulse}}{\tau_{act}}}) + (1 - stim_t) \cdot a_t \cdot e^{-\frac{dt}{\tau_{deact}}}) \tag{2.10}$$

where t_{pulse} is the duration of the activation in ms.

2.3.4. Integration of stimulation signal in activation dynamics

The spikes that the SNN executes are converted into a square step signal with a duration based on the time it takes for the system to reach 95% of maximum activation level (Fig. 2.2B). We call this squared signal the stimulation signal ($stim$). This is implemented to capture the exponential growth of the muscle's activation level in addition to the first-order activation dynamics. Modeling activation dynamics using only a first-order approach results in a linear increase towards the activation level, missing the activation path of the muscle.

The stimulation signal is produced by the use of some kind of a counter t_{stim} and the use of the SuperSpike formalism method described in in section 2.2.2. The stimulation time is the counter which is increased by dt_{stim} upon receiving a spike from the SNN, and decreased by 1 if no spike is received:

$$t_{stim,t+1} = t_{stim,t} + (S-1) \cdot 1 + S \cdot dt_{stim} \tag{2.11}$$

where $t_{stim,t+1}$ is the stimulation time for the next time step, S the spike [0,1] received from the SNN and dt_{stim} duration of the stimulation of the muscle per single spike. This function is clipped between [-1, dt_{stim}] every time step to ensure the stimulation time do not become infinite. The stimulation counter (t_{stim}) is used as input in the Heavyside step function (Equation 2.7) which generate a unit step. This unit step is the stimulation signal ($stim$) and is approximated (such as Equation 2.8) to provide a continuous function to enable gradient calculations:

$$\frac{dstim}{dt_{stim}} \approx \frac{1}{(|t_{stim}|+1)^2} \tag{2.12}$$

This approach accounts for multiple spikes and captures the activation behavior of the muscle in a more biologically plausible manner compared to using first-order activation dynamics alone.

2.3.5. Hill-type Muscle Model

The contraction dynamics of the muscle are modeled by a Hill-type model (Figure 2.2A.). The Hill model was chosen to keep the computational cost low and to avoid unnecessary complexity, while still accurately representing muscle behavior at the macroscopic level [32]. The focus was not on the specific characteristics of the muscle model but rather on utilizing a simplified Hill-type model that simulates force-length and force-velocity relationships using differentiable functions. Instead of traditional if-else statements commonly found in most Hill-type models, we designed custom functions that aligns with the general behavior of Hill-type models, ensuring gradient calculation throught the Muscle model.

The contraction dynamics of the muscle is described by the Hill-type model. The Hill-type model assumes that, at each instant, the total force produced by the skeletal muscle, results from the contribution of a passive elastic element (PE), representing the rigidity of the structures physically in parallel with the muscle fiber, and an active Hill active contraction element (CE) (Fig. 2.2A.) [48, 52].

The total force is expressed by:

$$F_M = CE + PE \tag{2.13}$$

where F_M is the total force, CE is the force produced by the contractive element, and PE is the force produced by the passive element. The CE and PE are functions of l_{CE} and v_{CE} normalized with respect to l^0_{CE} and v^0_{CE}, resulting in $\tilde{l}_{CE}$ and $\tilde{v}_{CE}$. Note that the $\tilde{l}_{CE}$ and $\tilde{v}_{CE}$ and $\tilde{l}_{PE}$ and $\tilde{v}_{PE}$ are the same, $\tilde{l}_{CE}$ and $\tilde{v}_{CE}$ are only used in the equations for clearance.

The CE depends on the maximum force produced by the muscle, activation level, force-length and force-velocity relationship:

$$CE = F_{max} f_l f_v a(t) \tag{2.14}$$

where the f_v and f_l are calculated by the equations:

$$f_l = \frac{e^{-(\tilde{l}_{CE}-1)}}{0.5^2} \tag{2.15}$$

$$f_v = \frac{-\tan^{-1}(10\tilde{v}_{CE})}{1.5+1} \tag{2.16}$$

where the force-length relationship is a Gaussian function. The force of PE is given by a half parabolic function:

$$PE = F_{max} \cdot \begin{cases} 0, & \text{for } \tilde{l}_{CE} \leq 1 \\ \tilde{l}_{CE}^2, & \text{for } \tilde{l}_{CE} > 1 \end{cases} \tag{2.17}$$

To enable gradient computation, the derivative of Equation 2.17 is approximated via the Spike formalism method described in section 2.2.2. This results in:

$$\frac{dPE}{d\tilde{l}_{CE}} \approx F_{max} \cdot \frac{\tilde{l}_{CE}^2}{|\tilde{l}_{CE} - 1| + 1} \tag{2.18}$$

The resulting behaviour of the force-length and velocity-length relationship is shown in Figure 2.2C and D respectively. This muscle model is differentiable and non-linear.

2.4. Integration and Parameter Mapping of the Musculoskeletal System and SNN

2.4.1. Angle Reformulation

To translate the integer values from the body model simulated using BRAX into currents for the SNN, the angle θ is reformulated using a ReLU function and a normalization method. In each simulation time step, BRAX updates the angle θ, producing an integer that can range from $-\infty$ to ∞. Since LIF neurons are not been stimulated by negative currents, these negative values are converted to positive values using a ReLU function. Additionally, the output values are normalized to correspond to spike frequencies between 10-100 Hz. The normalization parameters are determined prior to the simulation. This implementation, although simple, ensures that the neurons spike in a biological realistic manner and enables real-time interaction between the musculoskeletal model and the SNN.

2.4.2. Force to Torque Conversion

To connect the muscle model with BRAX a conversion from force to torque is implemented because the muscle model generates muscle force, but BRAX only supports torque actuators. This conversion is based on the geometric configuration of the muscle and joint, where the torque τ is calculated as:

$$\tau = l_{con1} \cdot \sin\left(\cos^{-1}\left(\frac{-l_{con2}^2 + l_m^2 + l_{con1}^2}{2l_m \cdot l_{con1}}\right)\right) \cdot F_M \tag{2.19}$$

Here, l_{con1} and l_{con2} are the lengths of the connection points to the muscle, l_m is the length of the muscle, and F_M is the muscle force. See Figure 2.1 for illustration.

2.4.3. Determination of Muscle Length

The executed force by the muscle is dependent on the length of the muscle. Although the muscle itself is ”imaginary” and not directly simulated in BRAX, its current length can be derived from the geometric configuration of the muscle and joint. This is achieved using the law of cosines. The muscle length l_m is calculated based on the joint angle and the distances between the muscle’s attachment points. The equation used is:

$$l_m = \sqrt{l_{con1}^2 + l_{con2}^2 - 2l_{con1} \cdot l_{con2} \cos(\theta)} \tag{2.20}$$

where l_{con1} and l_{con2} are the lengths from the joint to the muscle attachment points, and θ is the joint angle.

2.4.4. Tendon-Muscle Ratio

The tendon is not modeled in the Hill-type muscle model, even though its stiffness affects muscle behavior. To account for this, a tendon-muscle ratio is implemented. This ratio adjusts the relative lengths and stiffness of the muscle and tendon, simulating their relationship. In this model, the tendon is modeled as stiff, while the muscle retains its elastic elements based on the Hill-type model. Increasing the tendon-muscle ratio lengthens the tendon, resulting in a stiffer overall muscle-tendon unit, illustrated in Figure 2.2E. This increased stiffness affects force and torque generation, altering the dynamics of muscle contractions and their impact on movement.

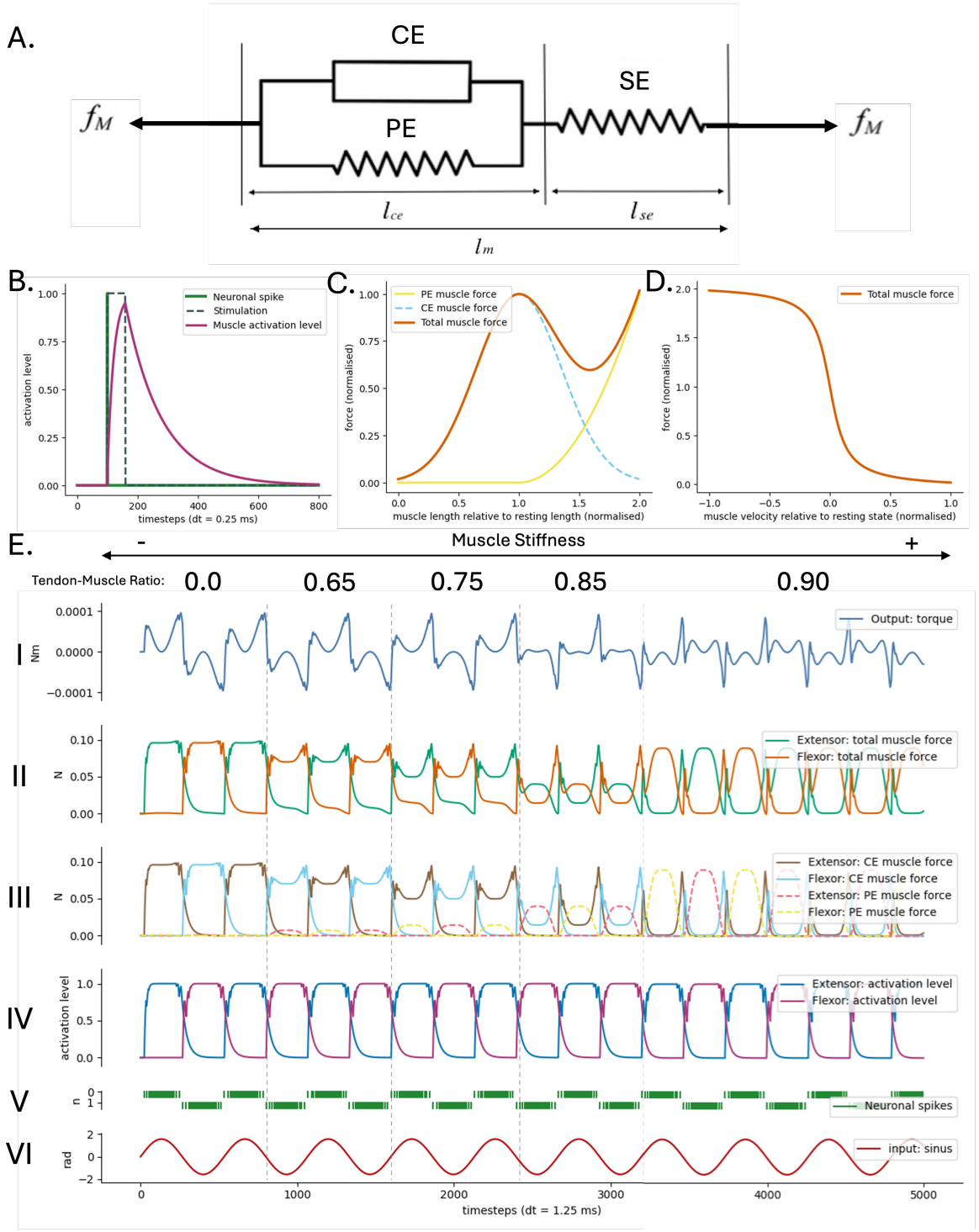

Figure 2.2: Overview muscle dynamics. A) Hill-type muscle model containing the contractile element (CE), whose length is indicated as l_{ce}, the passive parallel element (PE) connected in parallel which has the same length as the CE element and the elastic element (SE) connected in series whose length is indicated as l_{se}. The resulting force indicated as f_M and the total length of the muscle as l_m. B) Activation dynamics of the muscle in response to a neuronal spike. The green line represents the neuronal spike, the dashed line shows the duration of stimulation, and the purple line indicates the muscle activation level. The muscle activation rises quickly to its peak following the spike and stimulation, then gradually decays back to baseline over 800 timesteps (200 milliseconds). C) Force-length relationship of the Hill-type model, illustrates the normalized total muscle force (orange), passive elastic (PE) muscle force (yellow), and contractile element (CE) muscle force (blue dashed) as functions of muscle length relative to resting length. D) Force-velocity of the Hill-type model shows the normalized total muscle force as a function of muscle velocity relative to the resting state, demonstrating how force decreases with increasing velocity. E) Various stiffnesses of the muscle effecting the force and torque output. The muscle is stimulated by the input of a sinoisodal (VI), converted to spikes in the (V), activation level of the muscle (IV), passive and active force produced by the muscle (III), total muscle force (II) and the output torque based on the both forces (I). Both, the extensor and flexor muscle is simulated whereby the muscle length of them changes according to the angle of the sinusoidal. The stiffness of the muscle is based on the ratio between tendon and muscle (presented at the top of the graph), by increasing this ratio, the stiffness of the muscle increases and thereby the force and torque output.

2.5. Implementing Optimization Strategies for training SNNs

Gradient descent optimization requires gradient calculations to optimize the system. This section describes the strategies used in our optimization framework to ensure a scalability as the current study desires. The use of JAX, Stochastic Gradient Descent (SGD), and Forward Propagation Through Time (FPTT) is described in the current section.

2.5.1. Numerical Computing Library: JAX

JAX is a high-performance numerical computing library that enables automatic differentiation (autograd) and Just-in-Time (JIT) compilation for Python functions. The main reasons for using JAX in the current study are its ability to handle complex data structures effortlessly and its performance improvements for large-scale computations [17]. JAX's ability to handle pytrees simplifies the management and manipulation of nested data structures commonly encountered in advanced models like SNNs. Additionally, JAX's autograd is based on two fundamental concepts: the Jacobian-Vector Product (JVP) and the Vector-Jacobian Product (VJP), which directly compute the desired products, avoiding unnecessary computations. This allows for the automatic, effortless, and computationally efficient computation of gradients required in gradient descent optimization [17].

2.5.2. Optax's Stochastic Gradient Descent optimizer

Optax, a library for gradient-based optimization in JAX, is used to automatically update the weights of the SNN [5]. One of the optimizers that Optax provide is Stochastic Gradient Descent (SGD). SGD is a variation of the standard gradient descent optimization algorithm. Unlike standard gradient descent, which uses the entire dataset to compute gradients, SGD updates parameters only a single data point, allowing for faster iterations [39]. SGD also includes support for momentum, and Nesterov acceleration, as these are standard practice when using stochastic gradient descent to train neural networks. The SGD including momentum and Nesterov acceleration returns an update μ of the form:

$$\mu \leftarrow -\eta(g + \mu) \tag{2.21}$$

where η is the learning rate, g is gradient of the loss function, μ is the momentum parameter. Momentum and Nesterov acceleration is applied in the current work to accelerate convergence.

2.5.3. Forward Propagation Through Time

Forward Propagation Through Time calculates the gradients necessary for adjusting the weights in gradient descent optimization. FPTT was considered for implementation because of its memory efficient characteristic [51]. FPTT is formulated based on the system's state and parameters, this is described as $X_{t+1} = F(X, \theta)$ in a continuous system. To simulate this, the equation is discretized to:

$$F(X, \theta) = X + \Delta t \cdot f(X, \theta) \tag{2.22}$$

where Δt is the time step duration, and $f(X, \theta)$ represents the state transition function. Here we formulate FPTT as explicit integration of the sensitivity matrix, the update rule is given by:

$$\left\langle \frac{\partial X_{t+1}}{\partial \theta} \right\rangle = \frac{\partial F(X,\theta)}{\partial \theta}\bigg|_{X=X_t} + \frac{\partial F(X,\theta)}{\partial X}\bigg|_{X=X_t} \left\langle \frac{\partial X_t}{\partial \theta} \right\rangle \tag{2.23}$$

where $\frac{\partial F(X,\theta)}{\partial \theta}$ represents the direct and indirect effect of the parameters on the state transition, while $\frac{\partial F(X,\theta)}{\partial X} \left\langle \frac{\partial X_t}{\partial \theta} \right\rangle$ captures the effect propagated through the state. This approach explicitly takes into account the full system sensitivity, capturing how changes in parameters affect the state over time.

The term $\frac{\partial F(X,\theta)}{\partial X} \left\langle \frac{\partial X_t}{\partial \theta} \right\rangle$ involves a matrix-matrix multiplication, which is computationally expensive. To optimize this, the Jacobian-Matrix Product (JMP) is expressed as a series of Jacobian-Vector Products:

$$\left(\left(\frac{\partial F(X)}{\partial X} \left\langle \frac{\partial X_t}{\partial \theta} \right\rangle \right)^T \right)_i = \partial F(X) \left[D_i^T \right] \Big|_{\theta=\theta} \tag{2.24}$$

where $\left(\left(\frac{\partial F(X)}{\partial X} \left\langle \frac{\partial X_t}{\partial \theta} \right\rangle \right)^T \right)_i$ is the transpose of the matrix product between the Jacobian of F and the sensitivity matrix with i the i-th element of the transposed product, D_i^T is a vector that represents the

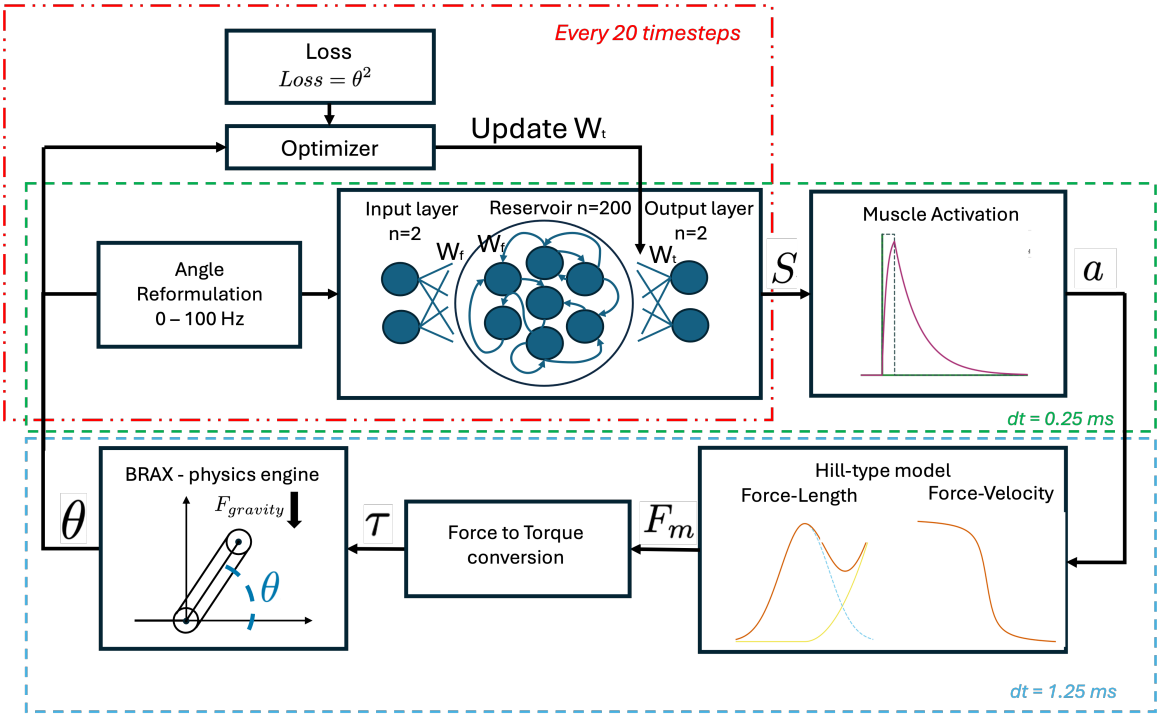

Figure 2.3: Technical Overview of the closed-loop model: The system contains a SNN consisting of an; input layer, a reservoir, and an output layer that feeds neural spikes S into the muscle activation dynamics which provides the activation level a that is used as input for the Hill-type muscle model; simulating force-length and force-velocity properties resulting in the muscle force F_m that is converted to torque τ using basic geometric principles which controls the arm in the BRAX environment of which the state of the angle θ is reformulated to an ampere that generates spikes in the range of 0 and 100 Hz, used as input for the SNN - closing the loop. The system operates with two different time steps; the muscle model and the body simulation in BRAX run at a timestep of 1.25 ms and the SNN and muscle activation dynamics at 0.25 ms, looping five times through the subsystems per timestep of 1.25 ms. The optimization loop contains the loss function, optimizer and calculated gradients by FPTT. Every 20 timesteps the weight matrix W_t is updated based on the loss function using gradient descent optimization. This overview clearly illustrates the subsystems and their connections, forming the complete closed-loop system containing the fundamental components of an embodied brain model.

i-th column of the sensitivity matrix transposed, and $\partial F(X)\left[D_i^T\right]\Big|_{\theta=\theta}$ is the Jacobian-vector product evaluated at the parameter θ.

The conversion of JVP to Jacobian-Matrix Product is achieved using jax.vmap and jax.tree.map in combination with jax.tree.transpose. Using this JMP circumvents full Jacobian calculation in the intermediate steps, speeding up both compile times and runtime. As JVP extent over matrices and vectors to general linear spaces, this also allows us to more easily define the system state as a series of JAX PyTrees.

2.6. Full model Design

The full loop system is presented in Figure 2.3, with its corresponding parameters listed in Table 2.1 and the optimization procedure used in Algorithm 1. This section outlines the design choices made in the current work leading to the development of the closed-loop embodied brain model in its fundamental form.

The SNN is modeled based on a LIF model (see section 2.2.1) in a multilayer feedforward reservoir architecture. The network comprises an input layer (n=2), a reservoir layer (n=200) and an output layer (n=2) to control the two muscles. Recurrent connections are included in the second layer to enable temporal processing and signal modulation, which are essential for generating diverse output signals [6]. All the initial weights in the SNN are uniformly random distributed between a defined range and the recurrent weights are additionally masked to introduce sparsity. Moreover, the synaptic connections of the LIF cells are modelled based on the biophysical characteristics of AMPA receptors as these play

a major role in excitatory neurotransmissions in motor control (Table 2.1) [7, 50]. Furthermore, the number of neurons in each population is determined by balancing the need to minimize computational costs with the requirement to maintain a stable and controllable system.

The SNN and activation dynamics have been chosen to be simulated with a dt of 0.25 ms, while the muscle model and BRAX run at a dt of 1.25 ms (the SNN loops five times for each muscle-BRAX timestep). This approach captures the necessary detail of neural activity with smaller time steps, while the longer timesteps for the mechanical components reduce computational load. This creates an computational efficient and accurate simulation of the components.

For every 20 timesteps (dt = 1.25 ms) the trainable weights are updated based on the loss function and the calculated gradients (see Figure 2.3 and Algorithm 1). We have selected the weights between the reservoir and output layer as trainable weights, while the other weight remains fixed. This approach was chosen because updating weights across multiple layers would complicate the system with numerous potential solutions, requiring additional constraints to maintain stability. This is not included in our design requirements and do not directly contribute to the goal. Therefore, to enhance stability and minimize computational costs, we concentrated on a single weight update point. To avoid exploding gradients, the loss gradient and differential matrix are reset every 20 timesteps. To speed up the optimization loop, the @jax.jit decorator is used to JIT compile functions that are repeatedly used in the simulation, including the step functions of the LIF model, the muscle model, and the update functions.

The experimental setup used to investigate gradient descent optimization via FPTT is as follows. The muscles are initialized at a zero degrees angle with the muscle-tendon ratio of 0.75. This ratio was selected to give the SNN greater control over the arm compared to the passive forces at maximum angles, while still accounting for the influence of passive muscle forces in the simulation. The system is tasked with stabilizing the arm to a zero-degree angle, using the following loss function:

$$Loss = \theta^2 \tag{2.25}$$

The system has an initialization period of 20 timesteps, after which the arm's angle is reset randomly within the ranges of (-0.6, -0.4) or (0.4, 0.6) rad and a velocity in the range of (-0.01, 0.1) rad/s.

2.7. Hardware

The simulations and computations for the present study were conducted on a MacBook Air (13-inch, Early 2015). This machine is equipped with a 2.2 GHz Dual-Core Intel Core i7 processor, 8 GB of 1600 MHz DDR3 memory, and an integrated Intel HD Graphics 6000 GPU with 1536 MB of shared VRAM (GPU is not used for simulation). The primary storage device is a Macintosh HD, which served as the startup disk. While this hardware setup is modest compared to modern high-performance computing systems, it was sufficient to perform the necessary simulations and optimizations for this research.

Algorithm 1 Training SNN with FPTT

Require: $B = \{x_i, y_i\}_{i=0}^{T}$, # Epochs E
Require: Trainable params, Simulation states, and Simulation params
Require: Optimizer and learning rate η
Require: Initial number of simulation steps # Steps T
Require: Number of optimization epochs # Epochs E

Initialize Fixed Weights W_f
Initialize Trainable Weights W_t
for each $e \in E$ **do**
    Initialize Body state θ_i by f_{start}
    Initialize Neuron states u_i, i_i
    Initialize Muscle states $a_i, \tau_{m,i}$
    Initialize the differential matrix D_0
    Initialize the derivative of the loss L_0
    Reset L: $L = L_0$
    Reset D: $D = D_0$
    for each $i \in T$ **do**
        if $i \mod 20 == 0$ **then**
            Randomly reset state body angle θ_i by f_{reset}
        end if
        if $i \mod 20 == 0$ **then**
            Reset L: $L = L_0$
            Reset D: $D = D_0$
        end if
        Update $i_i, u_i, a_i, \tau_{m,i}, \theta_i = \hat{f}\left(\hat{f}_n(\theta_{i-1}, [u_{i-1}, i_{i-1}] \parallel W), \hat{f}_m(i_i, [a_{i-1}, \tau_{m,i-1}]), \hat{f}_b(\tau_{m,i}, [\theta_{i-1}])\right)$
        Update Loss gradient L: $L(W_t) = L(W_t) + \nabla_x \ell(x_i) \cdot D$
        Update Differential matrix D: $D = \nabla_x f(x_{i-1}, W_{t,i-1}) \left[D_i^T\right] \Big|_{\theta=\theta} + \nabla_W f(x_{i-1}, W_{t,i-1})$
        if $i \mod 20 == 19$ **then**
            Update W_t: $W_{t,i+1} = W_{t,i} - \eta(\nabla_W L(W_t)\big|_{W_t=W_{t,i}} + \mu m_t)$
        end if
    end for
    Update W_t: $W_{t,i+1} = W_{t,i} - \eta(\nabla_W L(W_t)\big|_{W_t=W_{t,i}} + \mu m_t)$
end for

Component	Parameters	Initial States
Spiking neural network	$v_{th} = 1.0$ mV $\tau_{syn} = 5.0$ ms $\tau_{mem} = 20.0$ ms	$I = jnp.zeros(n)$ mA $U = jnp.zeros(n) + 1e - 10$ mV $W_{f\_input} = \mathcal{U}(0.0, 0.1)$ $W_{f\_reservoir} = \mathcal{U}(-0.001, 0.00125) > 0.90$ $W_t = \mathcal{U}(-0.005, 0.005)$
Activation Dynamics	$t_{act} = 10.0$ ms $t_{deact} = 30.0$ ms $dt_s = 25.0$ ms $dt_{stim} = 60.0$ ms	$a = 0.0$ (dimensionless)
Muscle Activation Dynamics	$F_{max} = 0.1$ N $v_{max} = 0.2$ m/s	$F_m = 0.0$ N $v_m = 0.0$ m/s
Body Model - BRAX	$l_{arm} = 0.01$ m $l_{con1} = 0.01$ m $l_{con2} = 0.001$ m $joint\_range = [-\frac{1}{2\pi}, \frac{1}{2\pi}]$ rad $joint\_damping = 5 \cdot 10^{-5}$ Nm/rad/s $joint\_armature = 5 \cdot 10^{-6}$ kg·m² $joint\_gear = 1.0$ (dimensionless) $F_{gravity} = 0.981 m/s^2$	$f_{start} = 0.0$ rad, 0.0 rad/s $f_{reset} = \mathcal{U}(-0.6, -0.4), (0.4, 0.6))$ rad, $\mathcal{U}((-\frac{\pi}{100}, \frac{\pi}{100})$ rad/s $\tau = 0.0$ Nm

Table 2.1: Parameters and initial states for each component of the system. v_{th}: threshold voltage (mV), τ_{syn}: synaptic time constant (ms), τ_{mem}: membrane time constant (ms), t_{act}: activation time constant (s), t_{deact}: deactivation time constant (s), dt_s: timestep for activation dynamics (ms), F_{max}: maximum muscle force (N), v_{max}: maximum muscle velocity (m/s), l_{arm}: length of arm (m), l_{con1}: muscle connection point at the arm side (m), l_{con2}: muscle connection point at the ground side (m), $joint\_range$: range of joint motion (rad), $constraint\_stiffness$: constraint stiffness, $constraint\_limit\_stiffness$: constraint limit stiffness, $spring\_mass\_scale$: spring mass scale, $spring\_inertia\_scale$: spring inertia scale, $solver\_maxls$: solver max line search, I: synapse current (mA), U: membrane voltage (mV), W_{f_input}: fixed input weights, $W_{f\_reservoir}$: fixed reservoir weights, W_t: initial trainable weights, a: initial activation state (dimensionless), F_m: initial muscle force (N), v_m: initial muscle velocity (m/s), f_{start}: initial joint state (rad, rad/s), f_{reset}: reset joint state (rad, rad/s), τ: initial torque (Nm), $joint\_damping$: joint damping (Nm/rad/s), $joint\_armature$: joint armature (kg·m²), $joint\_gear$: joint gear (dimensionless).

3

Results

The goal of this study is to demonstrate gradient descent optimization through FPTT in embodied SNNs in a biologically representative and scalable manner. The design of the system we use is discussed in section 2.6. The current chapter first validates gradient descent optimization in the embodied SNN and explores its learning behavior. Next, the firing frequencies are verified to ensure that the system is biologically representative, as defined in this paper, with frequencies below 100 Hz. Scalability is assessed through computational performance analysis and evaluation of the software architecture. Finally, we examine the impact of muscle stiffness on motor control behavior and performance in optimized SNNs across different angles. For information on the steady-state behavior of the SNN in isolation, gradient descent validation via FPTT, and the learning behavior of the isolated SNN, see Appendix A.

3.1. Validation gradient descent optimization in the embodied SNN

The goal of this section is to demonstrate the effectiveness of the gradient descent optimization using FPTT in training the embodied SNN to control the arm muscles and stabilize the arm angle at zero degrees. Specifically, we aim to assess whether the SNN can learn to manage the non-linear dynamics of the muscle model. We approach this by examining our optimization strategy across five initial weight ranges; *U(-0.005, 0.005)*, *U(-0.05, 0.05)*, *U(-0.5, 0.5)*, *U(-1.0, 1.0)*, *U(-0.1, 0.1)*, *U(-2.0, 2.0)* to avoid reliance on a ”lucky” initialization and to assess the robustness of the optimization technique. The SNN was tasked to stabilize the arm at zero degrees for 100 iterations each with a maximum of 4000 timesteps. For each initial weight configuration, we tracked the loss and weight updates over iterations. A successful learning process is indicated by updated weights and loss decreasing over the 100 iterations. For each initial weight range, the loss should ideally converge to a small value, indicating the arm is stabilized around zero degrees.

Figure 3.1A. shows fluctuating, not steadily decreasing, averaged loss of the last 2000 timesteps over the iterations. This suggests that the degree of difficulty to solve the problem varies and affects the system’s performance. Figure 3.1B. presents the weight update of a single weight per initial weight range, indicating that the gradients are calculated through the system. To examine whether the gradients are informative, meaning that the gradients result in better performance of the system aka the system learns, the averaged loss over the first 10 iterations (3.1C.) and last 10 iterations (3.1D.) are compared per initial weight range. The results show decreased average losses for the initial weight ranges *U(-0.005, 0.005)*, *U(-0.05, 0.05)*, *U(-0.5, 0.5)*, and *U(-1.0, 1.0)*, while an increased average loss is observed for *U(-2.0, 2.0)*. This indicates that the SNN does not learn to stabilize the arm at zero degrees within 100 iterations when initialized with weights in the range *U(-2.0, 2.0)*, whereas it does for the other configurations. Furthermore, the initial weight range *U(-0.05, 0.05)* presents the lowest average loss over the last 10 iterations, indicating that this configuration results in the best performance within the given iterations of training.

The results demonstrate that the gradient descent optimization is effective for training the embodied SNN to stabilize the arm at zero degrees, successfully handling the non-linearity of the muscle model. In

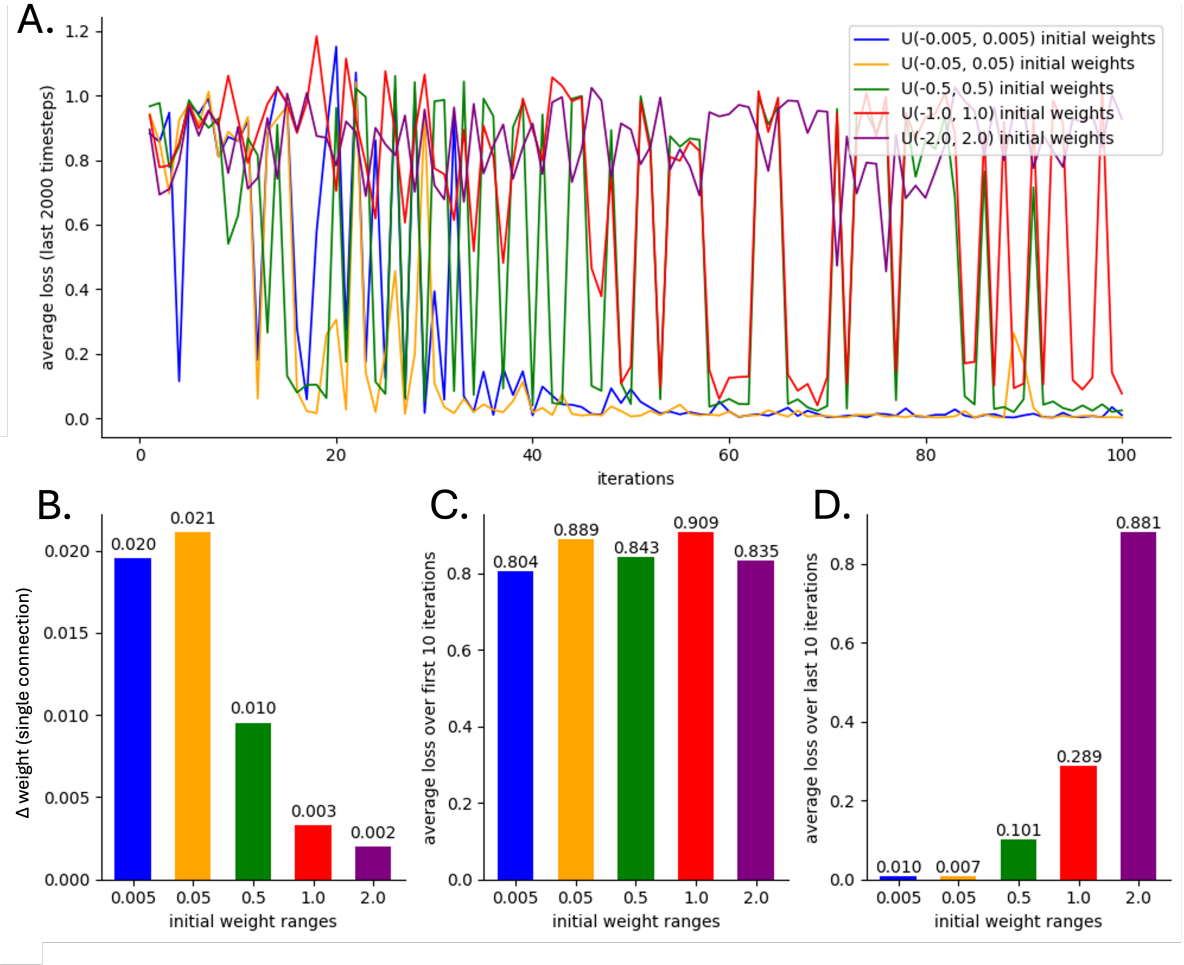

Figure 3.1: Learning analyses of the full loop system in which an arm learns to balance by controlling two muscles with 5 different initial weight ranges; *U(-0.005, 0.005)*, *U(-0.05, 0.05)*, *U(-0.5, 0.5)*, *U(-1.0, 1.0)*, *U(-0.1, 0.1)*, *U(-2.0, 2.0)*, each going through the same training process under the same conditions (100 iterations consisting of 4000 timesteps). A) Average loss per iteration of the last 2000 timesteps. B) Presents the Δ weight of the single weight that exhibit the greatest change between iteration 1 and 2, reflecting the gradient calculations for all initial weights. C) Shows the average loss of the first 10 iterations and D) of the last 10 iterations (of the last 2000 timesteps) per initial weight range. This shows that the gradients are informative for all the initial weight ranges with exception of *U(-2.0, 2.0)*. In summary, all weight ranges present to have a gradient, however for initial weight range *U(-2.0, 2.0)* the gradient is not informative, for the other initial weight ranges; *U(-0.005, 0.005)*, *U(-0.05, 0.05)*, *U(-0.5, 0.5)*, *U(-1.0, 1.0)*, *U(-0.1, 0.1)*, the gradients are informative, meaning the SNN is learns to balance the arm.

addition, the results indicate that the system is sensitive to the initial weight range, revealing a limitation of our optimization approach in that it can optimize the embodied SNN only within a certain range of initial weights. The initial weight range *U(-2.0, 2.0)* represent a more diverse set of initial conditions compared to the smaller weight ranges which result in poor gradient information making it challenging to optimize the embodied SNN. Overall, the results validates that the optimization strategy is robust across a range of initial weights, with *U(-0.05, 0.05)* yielding the most effective learning outcome, underpinning its suitability for training the embodied SNN in this context.

3.2. Learning behavior of the embodied SNN

This section demonstrates how the embodied SNN adapts its neural activity over training iterations to stabilize the arm, illustrating changes in behavior as it learns. We compare the spike activity, torque output, and arm angle at the beginning and end of the training process using the initial weight range *U(-0.05, 0.05)*, as it shows the best performance in previous results. Two iterations with similar initial angle offsets, iteration 11 and 82 (3.2), are selected and divided into four phases to illustrate the progression in learning to stabilize the arm.

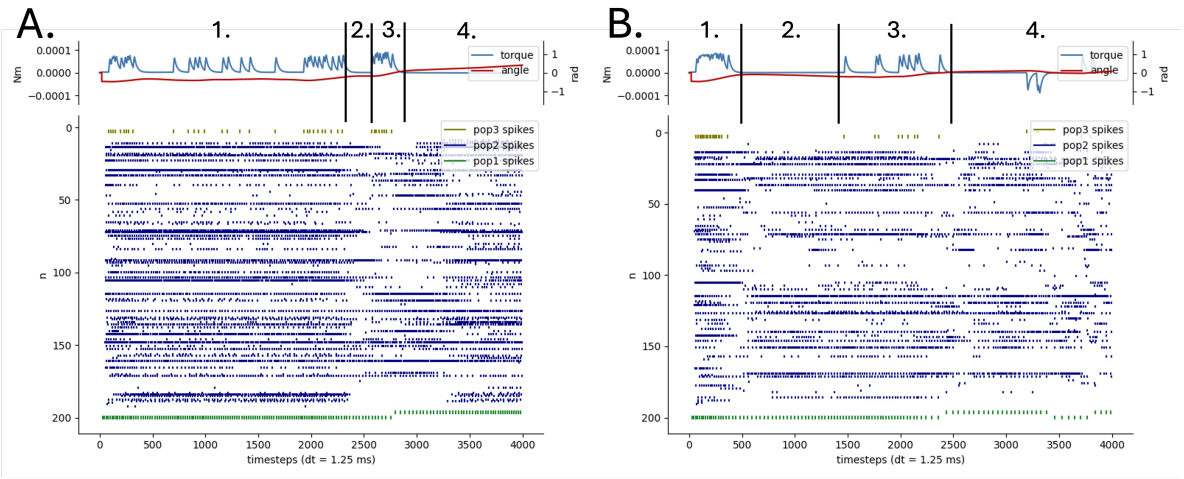

Figure 3.2: Learning behavior of the SNN with initial weight range *U(-0.05, 0.05)* at A) iteration 11 and B) iteration 82, simulated for 5 seconds (4000 timesteps with a time step of 1.25 ms). Each plot represents the SNN's activity across three populations: population 1 (n=2, green), population 2 (n=200, blue), and population 3 (n=2, yellow). The neural activity influences the torque output (blue line), which in turn controls the angle of the arm (red line). Both figures are divided into four phases for analysing its behavior: Phase 1 represents the system's initial response to the randomly set arm angle; Phase 2 occurs when the SNN stops spiking, assuming that the angle will reach zero with the given stimulation; Phase 3 begins when the SNN detects that the angle has not reached zero and resumes spiking to correct it; and Phase 4 is the final phase where the SNN ideally balances the arm around the zero-degree angle. Over time, the SNN adapts its spiking behavior to balance the arm, as reflected in the changes in torque and angle stabilization between the two iterations.

Phase 1 represents the system's initial response to the randomly set arm angle; Phase 2 occurs when the SNN stops spiking, here the systems assumes that the angle will reach zero with the given stimulation; Phase 3 begins when the SNN detects that the angle has not reached zero and resumes spiking to reach a zero degrees angle; and Phase 4 is the final phase where the SNN ideally balances the arm around the zero-degree angle.

Figure 3.2A. shows iteration 11 and 3.2B. iteration 82. The following observations can be made: Phase 1 is longer for iteration 11 compared to iteration 82, indicating that iteration 82 converges more quickly to zero degrees. Additionally, the angle achieved in phase 1 is closer to zero in iteration 82, demonstrating better precision in arm control. The magnitude of the error after the initial response reflects the duration of phase 2, which is shorter for iteration 11 compared to iteration 82. In phase 3, iteration 11 exhibits high-frequency spiking over a short period, resulting in overshooting. In contrast, iteration 82 shows low-frequency, controlled spiking that more precisely adjusts the angle, with minimal overshooting. In phase 4, iteration 11 shows no activity, leading to an unbalanced arm, while iteration 82 maintains stability around zero degrees, precisely adjusting the arm as needed.

The results demonstrate that the embodied SNN improves its control of the arm over training iterations, as seen by more precise and stable behavior in iteration 82 compared to iteration 11. The SNN learns to reduce spiking frequency and avoid overshooting, leading to better stabilization of the arm.

3.3. Verifying firing frequencies

This section examines whether the embodied SNN exhibits biologically representative firing frequencies, specifically below 100 Hz. We use data from the training process with the initial weight range *U(-0.05, 0.05)*. The average and maximum spike frequencies are measured and compared against a 100 Hz threshold. The system is considered biologically representative if the firing frequencies remain below this threshold.

The majority of the firing frequencies measured are between 0 and 100 Hz (Figure 3.3). However, frequencies above 100 Hz, with a maximum of 1200 Hz, are observed in populations 2 and 3 (highlighted by red circles). These high frequencies indicate bursting activity. In the first 18 iterations, population 3 exhibits low to near-zero spike frequencies, while population 2 shows frequencies above 100 Hz, indicating a near-zero output signal and no muscle control. These measurements do not reflect

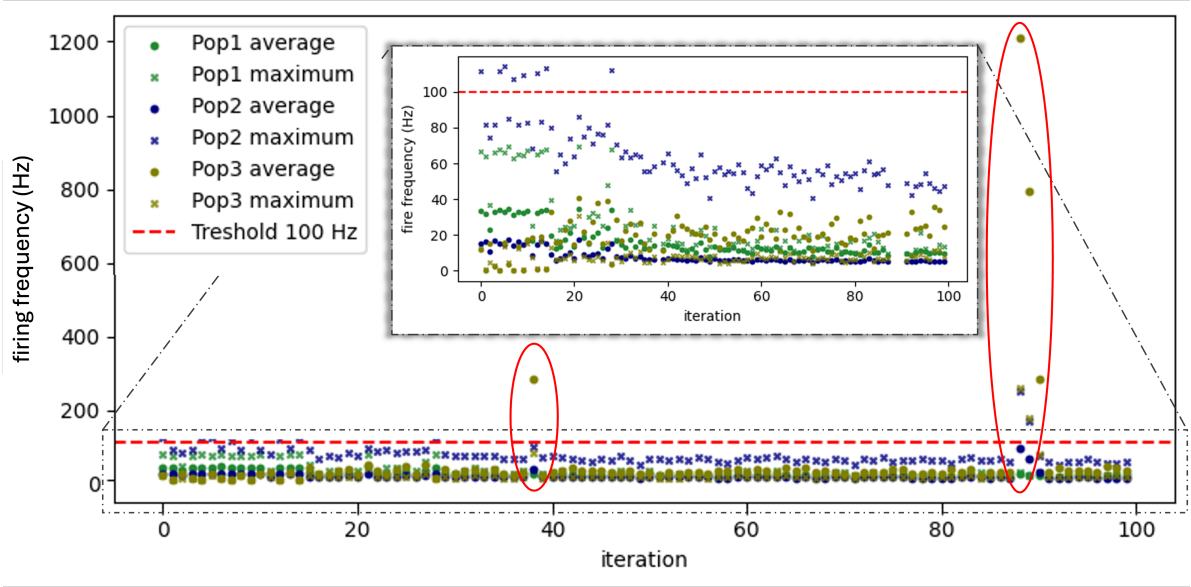

Figure 3.3: Firing frequency of the full system with initial weight range *U(-0.05, 0.05)* per iteration (100 iterations of 4000 timesteps). The average and maximum spike frequency per neuron for each population are presented, along with the 100 Hz threshold (red dotted line). The zoom-in view shows spike frequencies under 100 Hz in detail. While most neurons maintain firing frequencies below 100 Hz, some iterations show neurons exceeding this threshold.

the system's firing characteristics during active muscle control; therefore, the first 18 iterations are excluded from the analysis of whether the system maintains biologically representative firing frequencies under 100 Hz.

Overall, the results show that the majority of firing frequencies are below 100 Hz, with a few outliers indicating burst spikes. The cause of these bursts will be discussed in section 4; the discussion.

3.4. Scalability of the system

The goal of this section is to evaluate the scalability of the embodied SNN by analyzing its computational performance and software architecture. We approach this by measuring the total compile time and the runtime required to simulate 1 biological second for neuronal populations of 50, 100, 200, 500, and 1000 neurons, for the full system, the system without the muscle model, and the system without optimization. This analysis provides insights into the runtime behavior of the system as the SNN scales. Additionally, memory usage is monitored over 200,000 timesteps to verify that it remains stable and does not increase, ensuring the system is not constrained by memory issues. Furthermore, a simplified overview and description of the software architecture are provided to assess whether the architecture supports the addition of components and adjustments of parameters to change or scale the system.

The results exhibit (3.4A.) that both the total compile time and the time required to simulate 1 biological second increase as the neuronal population grows. Additionally, the results indicate that the gradient descent optimization technique contributes the most to the total compile time and runtime, while the muscle model has only a minimal impact. Overall, the results suggest an exponential growth in both compile time and runtime as the neuronal population increases.

The memory usage measurements show (3.4B.) memory usage starts at 1004.03 MB, and decreases until it stabilizes at 134.47 MB after approximately 140,000 timesteps. This behavior is explained in the section 4, the discussion. The results demonstrate that the memory does not increase, concluding the system is not constrained by memory limitations.

Figure 3.4D. illustrates the software architecture, structured into two levels: the Simulation Loop (main) file and the Params file. The Simulation Loop (main) file integrates high-level components through classes like TrainableParams for synaptic weights, SimulationState for dynamic states of neurons, muscles, and the BRAX environment, and SimulationParams for static configurations. It executes a single

simulation step function that updates all states in each timestep. This file builds on lower-level dynamics defined in the Params file, which includes classes for LIF neuron states, single muscle activation states, and muscle contraction states and their corresponding parameters and step functions. The physical body configuration and characteristics are defined in a xml file, which the BRAX Environment file uses to create a simulated setting supported by the BRAX simulator. The environment's step function communicates changes back to the Simulation Loop. Here, in the main file the optimization is handled using the Optax optimizer, JAX gradient processing features, and the support of the PyTree functions from the Utility file.

To conclude this section, the system exhibits zero memory constraints and features a well-structured, class-based software design, ensuring adaptability and scalability without changes to the core framework. These results demonstrate the potential to scale up our embodied brain simulation.

3.5. Neural motor control across different muscle stuffiness's

The goal of this section is to gain knowledge of the system's ability to control and stabilize the arm after training while considering the non-linearities of the muscle model across different muscle stiffnesses. We aim to observe distinct differences in motor control behavior and performance at varying stiffness levels, demonstrating the impact of muscle properties on the SNN. Three SNNs with tendon-muscle ratios of 0.0, 0.75, and 0.85, were trained under the same conditions for 100 iterations, each lasting 5 seconds (4000 timesteps). After training, the optimized weights were applied to non-learning versions of the embodied SNN for each tendon-muscle ratio. These SNNs were then tasked to stabilize the arm's angle at zero degrees starting from six different initial angles: 30°, 20°, 10°, -10°, -20°, and -30°. These angles were chosen because they fall within the range of offset angles used during training process. To assess whether the embodied SNNs could compensate for the non-linear dynamics of the muscle, their ability to stabilize the arm at zero degrees was observed (Figure 3.5A.). Performance was quantified by calculating the sum of the loss over the first 1000 timesteps and 4000 timesteps (Figure 3.5B.) for each initial angle and tendon-muscle ratio. The lower the sum of the loss, the better the SNN's performance at stabilizing the arm.

The results show that regardless of the initial angle, the SNN brings the arm closer to zero degrees. Furthermore, it shows that the neural control behavior varies with different tendon-muscle ratios (Figure 3.5). Specifically, the tendon-muscle ratio of 0.0 tends to produce burst spikes in short bursts, followed by moments of inactivity to control the arm. In contrast, ratios of 0.75 and 0.85 generate an initial burst behavior in response to the offset angle, followed by single or few spikes to balance the angle around its zero degree angle. This pattern reflects a more refined and controlled strategy, where the SNN maintains stability with minimal activity. Despite the initial angle, the SNN consistently guides the arm toward zero degrees, demonstrating its capability to handle the non-linearities of the muscle model effectively.

This result indicates that while the tendon-muscle ratio of 0.75 exhibits the highest performance in the short term (first 1000 timesteps), over the entire duration (4000 timesteps), the ratio of 0.0 performs better (Figure 3.5B.). The long-term performance of the tendon-muscle ratio 0.0 suggests that its lower stiffness, which results in minimal passive forces, making it easier for the SNN to maintain control over an extended period. This implies that the SNN find it easier to manage less complex, more linear dynamics compared to stiffer, more non-linear muscle behaviors over the long-term, while in the short term, the difference in performance is less pronounced. To conclude, the results demonstrate that the optimized embodied SNNs can control and stabilize the arm across different tendon-muscle ratios and initial angles, accounting for the non-linearities of the muscle model.

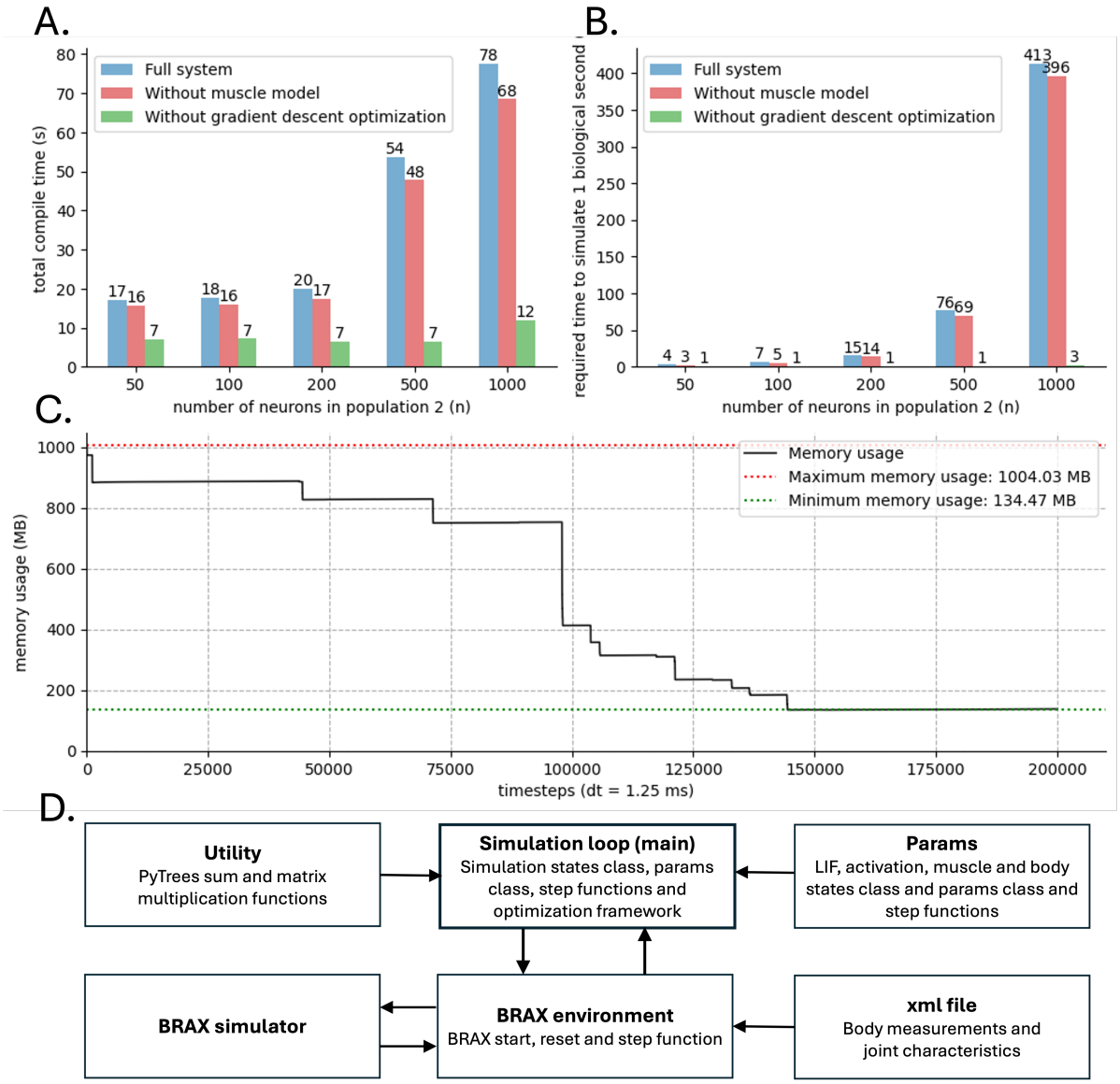

Figure 3.4: Computational performance and software architecture of the embodied SNN for system scalability analysis. A,B) Total compile time (s) (first iteration of 100 timesteps) and runtime to simulate one biological second (s) (averaged over 6 iterations) for various neuron population sizes in population 2. Blue bars represent the full system, red bars exclude the muscle model, and green bars exclude gradient descent optimization. C) Memory usage during 50 iterations of 4000 timesteps. The black line shows memory usage, with maximum (1004.03 MB) and minimum (134.47 MB) levels marked by red and green dotted lines, respectively. D) Software architecture overview. The Params file defines classes and step functions for LIF neuron dynamics, muscle activation, and contraction models, used in the Simulation Loop (main) file for managing neural populations, muscle activations, and body-BRAX states. The BRAX environment manages the body defined in an XML file, which customizes skeletal systems and joint characteristics. JAX and Optax libraries support the system. The results show stable memory usage over time and a flexible software architecture that facilitates scaling by adding new components and adjusting parameters without altering the core system.

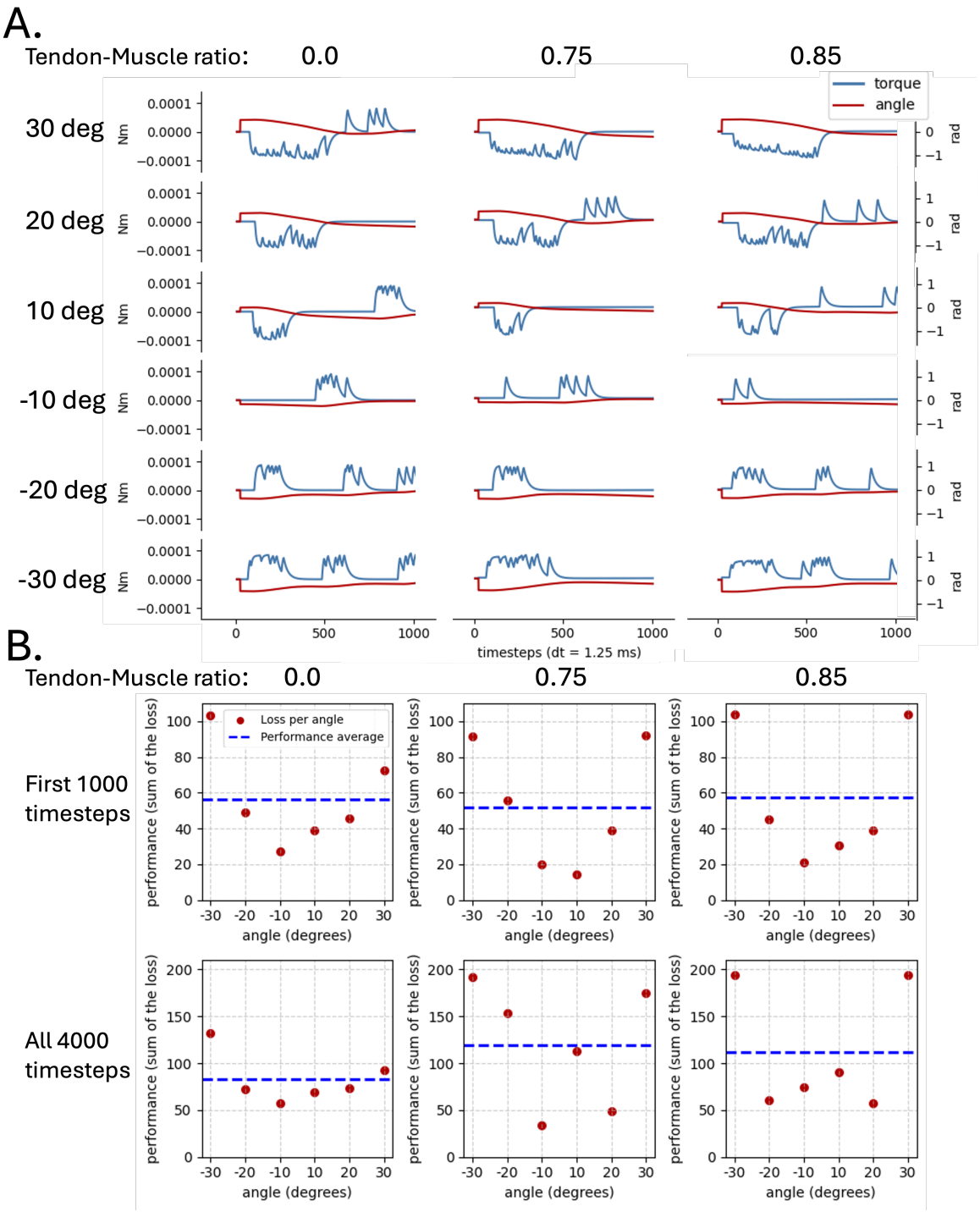

Figure 3.5: Motor control behavior and corresponding performance of optimized embodied SNNs in stabilizing the arm across different tendon-muscle ratios and angles. Embodied SNNs with the tendon-ratio 0.0, 0.75 and 0.85 are 100 iterations of 5 seconds (4000 timesteps) long trained, where after the optimized weights were applied to non-learning versions of the embodied SNN for each tendon-muscle ratio and tasked to stabilize the arm's angle at zero degrees starting from 30°, 20°, 10°, -10°, -20°, and -30°. Each tendon-muscle ratio reflects a different level of stiffness, the higher the ratio, the stiffer the muscle. A) Shows the motor controlling behavior of the three different tendon-muscle ratio's with its torque output (blue) and arm angle (red) over the first 1000 timesteps for the six initial angles. B) Quantifies the performance of the SNNs by calculating the sum of the loss over the first 1000 timesteps (top row) and over all 4000 timesteps (bottom row) for each tendon-muscle ratio. Lower loss values indicate better stabilization performance. The performance is compared for each initial angle and tendon-muscle ratio, demonstrating the SNN's ability to handle the non-linearities of the muscle model.

4

Discussion

The current paper demonstrated gradient descent optimization using FPTT in embodied SNNs to address challenges related to scalable and biologically representative modeling. Computational models for studying motor control often isolate subsystems, neglecting the relationship between body, brain, and environment [27, 21, 9]. Capturing the full closed-loop dynamics of motor control requires large-scale models with numerous parameters, which are difficult to tune due to memory constraints. This study developed a simplified embodied brain model that retains essential motor control components to demonstrate effective gradient descent optimization via FPTT. This approach aimed to create a scalable and biologically representative framework to support optimization in large-scale simulations to capture the underlying mechanisms of motor control. The results indicate that the system is fully differentiable, enabling effective gradient-based optimization. As the embodied SNN learned, its performance improved, and most neurons exhibit biologically realistic firing frequencies below 100 Hz. The software's zero memory constraints and modular, class-based structure demonstrates the scalability of our model. Additionally, the SNN demonstrated varying motor control behavior across different muscle stiffness levels, showing its capability to adapt to muscle properties and underscoring the significance of incorporating muscle dynamics into brain models.

From the results, we identified three notable findings for further discussion: the limited performance of gradient descent with an initial weight range of *U(-2.0, 2.0)*, the occurrence of high firing rates, and the observed decrease in memory usage over time. Gradient descent optimization was effective only within the small ranges of initial weights. This can be explained using the concept of sparsity, a biological phenomenon where only a small fraction of neurons are active at any given time [41]. In a network with larger initial weights, too many neurons might be active simultaneously, leading to less distinct and less informative gradients, resulting in poor optimization. Smaller weight ranges help maintain a sparser, more biologically realistic activity pattern, leading to better gradient descent optimization [14, 22]. Bursts of firing rates above 100 Hz likely occur due to sudden input changes that produce large gradient values. This behavior is observed when the arm is stabilized at zero degrees for a period, resulting in no gradients. After such a period, a slight deviation from zero can then cause exploding gradients, leading to large weight updates and burst activity. One solution is to implement a constant baseline spiking input, which could mitigate this [6, 43]. Gradient clipping could also be applied, as another solution to avoid exploding gradients [49]. In our work, normalizing the arm angle to always stimulate the SNN, even at zero degrees, could be the simplest solution to avoid burst spikes. Although burst activity may impact training and performance, it does not significantly affect the overall validity of the results. The memory usage that decreases over time can be explained by Python's garbage collection [12]. During the initial stages, memory usage is high due to JAX compiling numerous functions and initializing various states and data structures required for the simulation. As the simulation progresses, Python's garbage collector removes unneeded temporary objects, resulting in stabilized memory usage that reflects the actual requirements of the ongoing process. Thus, the system's eventual memory usage is stable and not constrained by memory limitations.

Furthermore, we must critically evaluate our approach before considering gradient descent a viable

technique for embodied brain simulation in a broader context, as this study focuses on optimizing a single task, tuning a single set of connections with a single SNN architecture. This work used a balancing task with muscles calibrated to zero degrees, raising the question of whether the passive forces contributed more to stabilization than the control of the SNN itself. Although the muscles exhibit passive forces when stretched, the chosen muscle stiffness does not generate passive forces exceeding the gravitational pull for the tested angles. This minimizes the influence of passive forces while keeping them as part of the muscle system. Though it would be insightful to re-calibrate the muscles to a different angle and observe the SNN's motor control when tasked with balancing the arm back to zero degrees. Focusing on tuning a single set of connections with a single SNN architecture raises questions about whether this optimization method can effectively optimize multiple connections or different SNN architectures. However, several studies have demonstrated that gradient descent is a viable method for optimizing SNNs [15, 18, 24, 25]. Adding to this, our study shows that gradient descent can handle non-linearities of the muscle model, which is also supported by prior research [4, 19, 28]. Based on both literature and our findings, we conclude that gradient descent optimization via FPTT is an effective approach for parameter tuning in embodied SNNs.

The embodied brain model introduced in this study lacks biological plausibility in terms of proprioceptive feedback and error encoding. Our system provides proprioceptive feedback by normalizing the arm angle. However, in biology, sensory feedback is obtained from muscle spindles and Golgi tendon organs. Spindles respond to changes in muscle length and velocity, while Golgi tendon organs measure changes in muscle tension [53]. To improve biological accuracy, muscle length and tension changes should be used as inputs for the SNN, as suggested by [23]. The current model can be easily modified to use the muscle length instead of the arm's angle as input for the SNN to enhance biological realism. In addition to that, the error used to optimize the embodied SNN in the current study is based on the difference between the desired and actual arm angles, which implies position control. However, this approach is slightly too straightforward, and can be adjusted to enhance biological realism of the system. As the exact mechanisms of motor control are not fully understood—motivating this study—there is general consensus among theorists that the nervous system aims to minimize energy expenditure during movement. [3]. This is not integrated into our system, which currently focuses solely on positional error. To provide deeper insights into motor control mechanisms, incorporating an energy efficiency constraint within the loss function for both spiking activity as well as muscle contraction would better align with behavior found in biology.

Moreover, our embodied SNN model is constrained by its two-dimensional environment, lacking the dynamics of real-world three-dimensional space. Simulating in 3D introduces additional challenges as the model's complexity and range of motion increases, expanding the number of possible solutions and necessitating proper parameter initialization and system constraints. However, to simulate a biological representative model, the musculoskeletal model needs to be expanded to a 3D environment. To extend the system to 3D, we need to include additional degrees of freedom for joint movements. In its simplest form, this can be achieved by adding another hinge joint to the existing joint, rotated over the other spatial plane, and attaching two muscles on the sides of the pole to actuate the arm. This setup could represent a simplified hip joint connected to the bone. Adding the hinge joint can be done easily in the XML file and adding the muscle pair in the simulation class int he main file. Additionally, the SNN needs to be extended to control four muscles instead of two, which involves adding two populations to the simulation class defined in the main file (see section 3.4). This approach is easy to implement while allowing investigation of the effect of a 3D environment on the system's behavior and gradient descent optimization.

Despite the discussed limitations in biological realism, the model presented in this paper incorporates various biologically plausible components for studying motor control. First of all, our embodied SNN system captures the relation between body, muscle and environment as gravity is applied to the system which lacks major computational motor control models [27, 21, 27]. Secondly, the conversion of spikes to muscle activity through simulation of the activation dynamics incorporating the differences in activation and deactivation dynamics. Compared with the integration of the squared stimulation time, the activation dynamics are biological plausible mimicked compared to first-order dynamics as used in comparable embodied brain models [34]. Additionally, the integrated Hill-type muscle model, combined with a tendon-muscle ratio, enables simulation of non-linear muscle contraction dynamics including adjustable muscle stiffness [48]. Furthermore, the SNN utilizes LIF cells, which capture the event-based

nature of spikes and synaptic states [37].

Our method shows promise for enhancing the understanding of motor control and has potential applications in fields such as computational neuroscience. Since the systems show zero memory constraints and well-structured, class-based software design, the system can be easily adjusted, extended or connected with differentiable brain models. The components of the system can be used as building-blocks to create your own embodied brain model to validate or test hypothesis in a controlled and applicable environment. Additionally, existing brain models such as that of the cerebellum can be connected to investigate open standing questions such as; how does the cerebellum contribute to motor learning and adaptation, and what are the specific roles of synaptic plasticity and neural coding in this process or how do disruptions in cerebellar circuits contribute to disorders like ataxia, and how can computational models help in understanding these dysfunctions? The embodied SNN model that the current paper introduced together with the ability of optimizing parameters by gradient descent optimization provides a valuable tool for exploration of the underlying mechanisms of motor control within a closed loop environment.

For further research it is recommended to implement a constant stable spike activity to avoid burst spikes and thereby biologically implausible firing frequencies. Secondly, it is advised to expand the musculoskeletal system to 3D model to validate gradient descent optimization via FPTT within a three dimensional embodied SNN. Furthermore, it is recommended to implement biological plausible proprioceptive feedback mechanism that is based on the change in muscle length and tension. Additionally, it is recommended to adjust the loss function to account for the efficient nature of biology by implementing energy constraints for spiking activity. The current study aimed to lay the groundwork for a platform that allows researchers to test and validate the underlying mechanisms of motor control within a gradient-driven optimized closed-loop environment and successfully made the first step towards it by demonstrating gradient descent via FPPT for embodied SNN optimization. We aspire to set the next step toward such a platform by extending and enhancing biological plausibility of our system.

5

Conclusion

This study presents a fully differentiable, closed-loop embodied Spiking Neural Network (SNN) model that captures motor control behavior. We conclude that gradient descent optimization using Forward Propagation Through Time (FPTT) is an effective approach for parameter tuning in embodied SNNs. The optimization framework employed has zero memory constraints and can be easily scaled. The study demonstrates different motor control behaviors across various muscle stiffness levels, highlighting the importance of simulating the brain-body relationship when studying motor control. However, the current model has certain limitations, including a lack of biologically representative proprioceptive feedback and the absence of dynamics found in real-world three-dimensional spaces. Future work should focus on enhancing biological realism and extending the system to a 3D environment to create embodied brain simulation platforms that enable simulation, testing, and validation of the underlying mechanisms of motor control.

References

[1] Guillaume Bellec et al. *Long short-term memory and learning-to-learn in networks of spiking neurons*. arXiv:1803.09574 [cs, q-bio]. Dec. 2018. DOI: 10.48550/arXiv.1803.09574. URL: http://arxiv.org/abs/1803.09574 (visited on 07/18/2024).

[2] Mathieu Blondel and Vincent Roulet. *The Elements of Differentiable Programming*. en. arXiv:2403.14606 [cs]. Mar. 2024. URL: http://arxiv.org/abs/2403.14606 (visited on 07/18/2024).

[3] Hillel J. Chiel et al. "The Brain in Its Body: Motor Control and Sensing in a Biomechanical Context". In: *The Journal of Neuroscience* 29.41 (Oct. 2009), pp. 12807–12814. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.3338-09.2009. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2794418/ (visited on 09/27/2024).

[4] Friedl De Groote and Antoine Falisse. "Perspective on musculoskeletal modelling and predictive simulations of human movement to assess the neuromechanics of gait". In: *Proceedings of the Royal Society B: Biological Sciences* 288.1946 (), p. 20202432. ISSN: 0962-8452. DOI: 10.1098/rspb.2020.2432. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7935082/ (visited on 09/27/2024).

[5] DeepMind and Babuschkin, Igor and Baumli, Kate and Bell, Alison and Bhupatiraju. *GitHub - google-deepmind/optax: Optax is a gradient processing and optimization library for JAX.* 2020. URL: https://github.com/google-deepmind/optax (visited on 07/20/2024).

[6] Vyacheslav Demin and Dmitry Nekhaev. "Recurrent Spiking Neural Network Learning Based on a Competitive Maximization of Neuronal Activity". English. In: *Frontiers in Neuroinformatics* 12 (Nov. 2018). Publisher: Frontiers. ISSN: 1662-5196. DOI: 10.3389/fninf.2018.00079. URL: https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2018.00079/full (visited on 09/23/2024).

[7] A Destexhe et al. "Kinetic Models of Synaptic Transmission". en. In: ().

[8] Mikael Djurfeldt, Örjan Ekeberg, and Anders Lansner. "Large-scale modeling - a tool for conquering the complexity of the brain". English. In: *Frontiers in Neuroinformatics* 2 (Apr. 2008). Publisher: Frontiers. ISSN: 1662-5196. DOI: 10.3389/neuro.11.001.2008. URL: https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/neuro.11.001.2008/full (visited on 07/17/2024).

[9] Benedikt Feldotto et al. "Deploying and Optimizing Embodied Simulations of Large-Scale Spiking Neural Networks on HPC Infrastructure". English. In: *Frontiers in Neuroinformatics* 16 (May 2022). ISSN: 1662-5196. DOI: 10.3389/fninf.2022.884180. URL: https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2022.884180/full (visited on 07/21/2024).

[10] Benedikt Feldotto et al. "Evaluating Muscle Synergies With EMG Data and Physics Simulation in the Neurorobotics Platform". English. In: *Frontiers in Neurorobotics* 16 (July 2022). Publisher: Frontiers. ISSN: 1662-5218. DOI: 10.3389/fnbot.2022.856797. URL: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2022.856797/full (visited on 09/16/2024).

[11] C. Daniel Freeman et al. *Brax – A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. en. arXiv:2106.13281 [cs]. June 2021. URL: http://arxiv.org/abs/2106.13281 (visited on 09/16/2024).

[12] *Garbage collector design*. en. URL: https://devguide.python.org/internals/garbage-collector/ (visited on 09/27/2024).

[13] Han Q et al. “Spinal cord maturation and locomotion in mice with an isolated cortex.” eng. In: *Neuroscience* 253 (2013). Place: United States, pp. 235–44. ISSN: 1873-7544 (Electronic). DOI: 10.1016/j.neuroscience.2013.08.057. URL: https://pubmed.ncbi.nlm.nih.gov/24012835/ (visited on 01/01/0012).

[14] Torsten Hoefler et al. *Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks*. en. arXiv:2102.00554 [cs]. Jan. 2021. URL: http://arxiv.org/abs/2102.00554 (visited on 09/27/2024).

[15] Dongsung Huh and Terrence J Sejnowski. “Gradient Descent for Spiking Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/hash/185e65bc40581880c4f2c82958de8cfe-Abstract.html (visited on 07/18/2024).

[16] Masao Ito. “Control of mental activities by internal models in the cerebellum”. en. In: *Nature Reviews Neuroscience* 9.4 (Apr. 2008). Number: 4 Publisher: Nature Publishing Group, pp. 304–313. ISSN: 1471-0048. DOI: 10.1038/nrn2332. URL: https://www.nature.com/articles/nrn2332 (visited on 12/13/2023).

[17] James Bradbury and Roy Frostig and Peter Hawkins and Matthew James Johnson and Chris Leary and Dougal Maclaurin and George Necula and Adam Paszke and Jake Vander{P}las and Skye Wanderman-{M}ilne and Qiao Zhang. *google/jax*. original-date: 2018-10-25T21:25:02Z. July 2024. URL: https://github.com/google/jax (visited on 07/21/2024).

[18] Ilenna Simone Jones and Konrad Paul Kording. *Efficient optimization of ODE neuron models using gradient descent*. en. arXiv:2407.04025 [q-bio]. July 2024. URL: http://arxiv.org/abs/2407.04025 (visited on 07/17/2024).

[19] Abdelhamid Kadiallah, David Franklin, and Etienne Burdet. “Generalization in Adaptation to Stable and Unstable Dynamics”. In: *PloS one* 7 (Oct. 2012), e45075. DOI: 10.1371/journal.pone.0045075.

[20] Anil Kag and Venkatesh Saligrama. “Training Recurrent Neural Networks via Forward Propagation Through Time”. en. In: *Proceedings of the 38th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2021, pp. 5189–5200. URL: https://proceedings.mlr.press/v139/kag21a.html (visited on 07/20/2024).

[21] Christian Keysers and David Perrett. “Demystifying social cognition: A Hebbian perspective”. In: *Trends in cognitive sciences* 8 (Dec. 2004), pp. 501–7. DOI: 10.1016/j.tics.2004.09.005.

[22] W. B. Levy and R. A. Baxter. “Energy efficient neural codes”. eng. In: *Neural Computation* 8.3 (Apr. 1996), pp. 531–543. ISSN: 0899-7667. DOI: 10.1162/neco.1996.8.3.531.

[23] Si Li et al. “Coordinated Alpha and Gamma Control of Muscles and Spindles in Movement and Posture”. English. In: *Frontiers in Computational Neuroscience* 9 (Oct. 2015). Publisher: Frontiers. ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00122. URL: https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2015.00122/full (visited on 09/27/2024).

[24] Yang Li and Yi Zeng. *Efficient and Accurate Conversion of Spiking Neural Network with Burst Spikes*. en. arXiv:2204.13271 [cs]. May 2022. URL: http://arxiv.org/abs/2204.13271 (visited on 09/16/2024).

[25] Yang Li et al. “Directly training temporal Spiking Neural Network with sparse surrogate gradient”. In: *Neural Networks* 179 (Nov. 2024), p. 106499. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2024.106499. URL: https://www.sciencedirect.com/science/article/pii/S0893608024004234 (visited on 09/16/2024).

[26] Henry Markram. “Seven challenges for neuroscience”. In: *Functional Neurology* 28.3 (2013), pp. 145–151. ISSN: 0393-5264. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3812747/ (visited on 07/20/2024).

[27] Henry Markram et al. “Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs”. In: *Science (New York, N.Y.)* 275 (Feb. 1997), pp. 213–5. DOI: 10.1126/science.275.5297.213.

[28] Ross H. Miller et al. “Muscle forces during running predicted by gradient-based and random search static optimisation algorithms”. en. In: *Computer Methods in Biomechanics and Biomedical Engineering* 12.2 (Apr. 2009), pp. 217–225. ISSN: 1025-5842, 1476-8259. DOI: 10.1080/10255840802430579. URL: http://www.tandfonline.com/doi/abs/10.1080/10255840802430579 (visited on 09/27/2024).

[29] Niklas Muennighoff et al. *Scaling Data-Constrained Language Models*. arXiv:2305.16264 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2305.16264. URL: http://arxiv.org/abs/2305.16264 (visited on 07/18/2024).

[30] Oliver Müller and Stefan Rotter. “Neurotechnology: Current Developments and Ethical Issues”. In: *Frontiers in Systems Neuroscience* 11 (Dec. 2017), p. 93. ISSN: 1662-5137. DOI: 10.3389/fnsys.2017.00093. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5733340/ (visited on 12/13/2023).

[31] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. “Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks”. In: *IEEE Signal Processing Magazine* 36.6 (Nov. 2019). Conference Name: IEEE Signal Processing Magazine, pp. 51–63. ISSN: 1558-0792. DOI: 10.1109/MSP.2019.2931595. URL: https://ieeexplore.ieee.org/document/8891809 (visited on 07/18/2024).

[32] *Neural prostheses : replacing motor function after disease or disability*. eng. New York : Oxford University Press, 1992. ISBN: 9780195072167. URL: http://archive.org/details/neuralprostheses0000unse_m3i7 (visited on 09/18/2024).

[33] Tam Nguyen. *What is a neural network? A computer scientist explains*. en-EUROPE. Dec. 2020. URL: http://theconversation.com/what-is-a-neural-network-a-computer-scientist-explains-151897 (visited on 08/25/2024).

[34] Javier Pérez Fernández et al. “A biological-like controller using improved spiking neural networks”. In: *Neurocomputing* 463 (Nov. 2021), pp. 237–250. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2021.08.005. URL: https://www.sciencedirect.com/science/article/pii/S0925231221011899 (visited on 07/17/2024).

[35] Shravan Tata Ramalingasetty et al. “A Whole-Body Musculoskeletal Model of the Mouse”. In: *IEEE access : practical innovations, open solutions* 9 (2021), pp. 163861–163881. ISSN: 2169-3536. DOI: 10.1109/access.2021.3133078. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8865483/ (visited on 09/27/2024).

[36] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. en. In: *Nature* 323.6088 (Oct. 1986). Publisher: Nature Publishing Group, pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (visited on 07/18/2024).

[37] Sanaullah et al. “Exploring spiking neural networks: a comprehensive analysis of mathematical models and applications”. In: *Frontiers in Computational Neuroscience* 17 (Aug. 2023), p. 1215824. ISSN: 1662-5188. DOI: 10.3389/fncom.2023.1215824. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10483570/ (visited on 09/23/2024).

[38] Andrew B. Schwartz. “Movement: How the Brain Communicates with the World”. In: *Cell* 164.6 (Mar. 2016), pp. 1122–1135. ISSN: 0092-8674. DOI: 10.1016/j.cell.2016.02.038. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4818644/ (visited on 09/27/2024).

[39] Sebastian Ruder. *An overview of gradient descent optimization algorithms/optimizing-gradient-descent/index.html on 19 January 2016.* en. Jan. 2016. URL: https://ar5iv.labs.arxiv.org/html/1609.04747 (visited on 07/18/2024).

[40] Yusuke Shinji, Hirotsugu Okuno, and Yutaka Hirata. “Artificial cerebellum on FPGA: realistic real-time cerebellar spiking neural network model capable of real-world adaptive motor control”. English. In: *Frontiers in Neuroscience* 18 (Apr. 2024). ISSN: 1662-453X. DOI: 10.3389/fnins.2024.1220908. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2024.1220908/full (visited on 09/16/2024).

[41] Anton Spanne and Henrik Jörntell. “Questioning the role of sparse coding in the brain”. In: *Trends in Neurosciences* 38.7 (July 2015), pp. 417–427. ISSN: 0166-2236. DOI: 10.1016/j.tins.2015.05.005. URL: https://www.sciencedirect.com/science/article/pii/S0166223615001198 (visited on 09/27/2024).

[42] Thomas M. Summe and Siddharth Joshi. *Slax: A Composable JAX Library for Rapid and Flexible Prototyping of Spiking Neural Networks*. en. Apr. 2024. URL: https://arxiv.org/abs/2404.05807v1 (visited on 07/21/2024).

[43] Clarence Tan, Marko Šarlija, and Nikola Kasabov. “Spiking Neural Networks: Background, Recent Development and the NeuCube Architecture”. en. In: *Neural Processing Letters* 52.2 (Oct. 2020), pp. 1675–1701. ISSN: 1573-773X. DOI: 10.1007/s11063-020-10322-8. URL: https://doi.org/10.1007/s11063-020-10322-8 (visited on 09/26/2024).

[44] Hoyoung Tang et al. “Spike Counts Based Low Complexity SNN Architecture With Binary Synapse”. In: *IEEE Transactions on Biomedical Circuits and Systems* 13.6 (Dec. 2019). Conference Name: IEEE Transactions on Biomedical Circuits and Systems, pp. 1664–1677. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2019.2945406. URL: https://ieeexplore.ieee.org/abstract/document/8859229?casa_token=SYSQavfv31EAAAAA:L1dQhQUdNukNIy36JS1BMxPCcnCqNu59hZZk2ps21n5taLyDVMXTOVrOeunuMJ22lkmDRbVyeg (visited on 09/27/2024).

[45] W. Van Geit, E. De Schutter, and P. Achard. “Automated neuron model optimization techniques: a review”. en. In: *Biological Cybernetics* 99.4 (Nov. 2008), pp. 241–251. ISSN: 1432-0770. DOI: 10.1007/s00422-008-0257-6. URL: https://doi.org/10.1007/s00422-008-0257-6 (visited on 07/18/2024).

[46] Werner Van Geit et al. “BluePyOpt: Leveraging Open Source Software and Cloud Infrastructure to Optimise Model Parameters in Neuroscience”. English. In: *Frontiers in Neuroinformatics* 10 (June 2016). Publisher: Frontiers. ISSN: 1662-5196. DOI: 10.3389/fninf.2016.00017. URL: https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2016.00017/full (visited on 07/18/2024).

[47] Wen TC et al. “Plasticity in One Hemisphere, Control From Two: Adaptation in Descending Motor Pathways After Unilateral Corticospinal Injury in Neonatal Rats.” eng. In: *Frontiers in neural circuits* 12 (2018). Place: Switzerland, p. 28. ISSN: 1662-5110 (Electronic). DOI: 10.3389/fncir.2018.00028. URL: https://pubmed.ncbi.nlm.nih.gov/29706871/.

[48] Jack M. Winters. “Hill-Based Muscle Models: A Systems Engineering Perspective”. en. In: *Multiple Muscle Systems: Biomechanics and Movement Organization*. Ed. by Jack M. Winters and Savio L-Y. Woo. New York, NY: Springer, 1990, pp. 69–93. ISBN: 9781461390305. DOI: 10.1007/978-1-4613-9030-5_5. URL: https://doi.org/10.1007/978-1-4613-9030-5_5 (visited on 07/08/2024).

[49] Xinyi Wu et al. “Optimizing Recurrent Neural Networks: A Study on Gradient Normalization of Weights for Enhanced Training Efficiency”. en. In: *Applied Sciences* 14.15 (Jan. 2024). Number: 15 Publisher: Multidisciplinary Digital Publishing Institute, p. 6578. ISSN: 2076-3417. DOI: 10.3390/app14156578. URL: https://www.mdpi.com/2076-3417/14/15/6578 (visited on 09/27/2024).

[50] Richard Naud and Liam Paninski Wulfram Gerstner Werner M. Kistler. *3.1 Synapses | Neuronal Dynamics online book*. URL: https://neuronaldynamics.epfl.ch/online/Ch3.S1.html (visited on 06/28/2024).

[51] Bojian Yin, Federico Corradi, and Sander M. Bohte. *Accurate online training of dynamical spiking neural networks through Forward Propagation Through Time*. arXiv:2112.11231 [cs]. Nov. 2022. DOI: 10.48550/arXiv.2112.11231. URL: http://arxiv.org/abs/2112.11231 (visited on 07/20/2024).

[52] F. E. Zajac. “Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control”. eng. In: *Critical Reviews in Biomedical Engineering* 17.4 (1989), pp. 359–411. ISSN: 0278-940X.

[53] Niccolò Zampieri and Joriene C. de Nooij. “Regulating muscle spindle and Golgi tendon organ proprioceptor phenotypes”. In: *Current opinion in physiology* 19 (Feb. 2021), pp. 204–210. ISSN: 2468-8673. DOI: 10.1016/j.cophys.2020.11.001. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7769215/ (visited on 09/27/2024).

[54] Friedemann Zenke and Surya Ganguli. “SuperSpike: Supervised learning in multi-layer spiking neural networks”. In: *Neural Computation* 30.6 (June 2018). arXiv:1705.11146 [cs, q-bio, stat], pp. 1514–1541. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco_a_01086. URL: http://arxiv.org/abs/1705.11146 (visited on 07/10/2024).

A

Steady state behavior and learning behavior of the SNN in isolation

To understand the steady state behaviour of the system, we first simulated the isolated SNN responding to a sine wave. When training the isolated SNN on a sinusoidal input, we find the following behaviour (Fig. A.1A.). Population 1, 2, and 3 exhibit spike frequencies (aeraged over the number of neurons per population) of 38, 17, and 36 Hz respectively. Population 1 present a spike activities that corresponds to the periodic behavior of the sinusoidal; the spike frequency increase as the input signal increase and decrease as the input signal decreases. Population 2 exhibits the greatest variation in spiking patters. These neurons have irregular spiking patterns with denser bursts of activity corresponding to specific phases of the sinusoidal input. The activity in population 1 results in both low and high spike frequencies in population 2. Some neurons in population 2 do not spike at all, while others simultaneously spike across the entire time sequence. These dynamics appear in population 3, expressed by relatively sparse spiking activity, with spikes occurring at specific intervals unlike the spiking patterns of population 1. These results shows that the SNN within a non-learning isolated environment is capable of generating a range of spiking patterns.

To validate our optimization approach with gradient descent, we trained the isolated SNN to replicate a sine wave (a translation of the sinusoidal input). As there is only a single output, the SNN is simulated with only a single output LIF (instead of two LIFs in the full system). Additionall, this LIF does not have the ability to spike, making it a LI. In this way, we can directly take the voltage level of the neuron as output signal of the SNN. The adjusted SNN is trained for 15 iterations each of a 2 seconds long simulation (1600 timestep) starting with two different initial weight ranges. In Figure A.1B. the averaged loss per iteration is shown of the simulation with initial weight ranges set at *U(-0.1, 0.1)* and *U(-0.001, 0.001)*. For both ranges the averaged loss decrease over iterations. The initial weight range *U(-0.001, 0.001)* shows a smaller average loss at both iteration 1 and iteration 15, suggesting that this range starts closer to the optimized weight values. The larger initial weight range *U(-0.1, 0.1)* requires more iterations to achieve the same level of optimization as *U(-0.001, 0.001)*. Figure A.1C. illustrates the learning process of the isolated SNN with initial weight range *U(-0.001, 0.001)* by presenting the output signal of iteration 1 and iteration 15. The output signal of the SNN match the target more closely in iteration 15 in contrast to iteration 1. In Iteration 1, there is a noticeable delay between the output signal and the target signal. By Iteration 15, this delay is no longer evident, suggesting that the network has learned to better align its output with the target signal. Thus, we both accuracy and temporal alignment with the target signal improving as training progresses; this must happen through informative gradients. This validates the ability of our optimization approach to train the SNN.

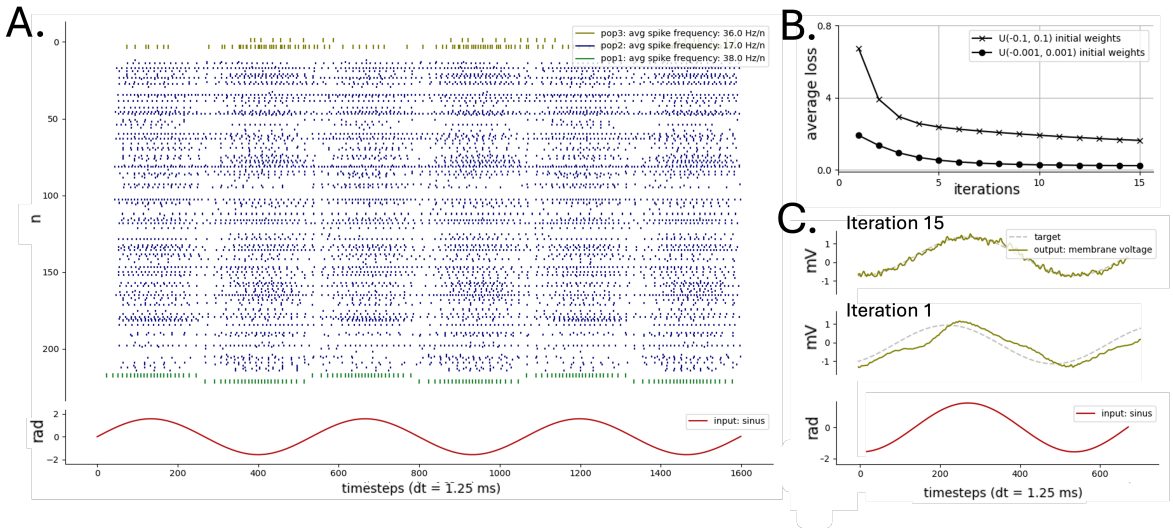

Figure A.1: Isolated SNN behaviour in response to sinusoidal input signal. A) Spiking behavior of the isolated SNN in a non-learning environment that shows spikes in population 1, 2 and 3 with an average spike frequency per neuron of 38.0, 17.0, 36.0 Hz respectively. B) Average loss across 15 iterations consisting of 1600 timesteps that is calculated by the error between the target signal (two-third of the sinusoid input signal) and output signal of the SNN (the membrane voltage of population 3) with initial weights randomly uniformly distributed between *U(-0.001, 0.001)*and *U(-0.1, 0.1)*, showing decreased loss over iterations for both initial weight ranges. C) Learning behavior of the isolated SNN at iteration 1 and 15 starting with weights between *U(-0.001, 0.001)*. Showing spike frequencies between 0 and 100 Hz and the SNN. Isolated SNN system shows spiking behvior and learns.