A Spiking Neural Network with Continuous Local Learning for Robust Online Brain Machine Interface

Elijah A. Taeckens and Sahil Shah

Abstract—Objective. **Spiking neural networks (SNNs) are powerful tools that are well suited for brain machine interfaces (BMI) due to their similarity to biological neural systems and computational efficiency. They have shown comparable accuracy to state-of-the-art methods, but current training methods require large amounts of memory, and they cannot be trained on a continuous input stream without pausing periodically to perform backpropagation. An ideal BMI should be capable training continuously without interruption to minimize disruption to the user and adapt to changing neural environments.** ***Approach.*** **We propose a continuous SNN weight update algorithm that can be trained to perform regression learning with no need for storing past spiking events in memory. As a result, the amount of memory needed for training is constant regardless of the input duration. We evaluate the accuracy of the network on recordings of neural data taken from the premotor cortex of a primate performing reaching tasks. Additionally, we evaluate the SNN in a simulated closed loop environment and observe its ability to adapt to sudden changes in the input neural structure.** ***Main results.*** **The continuous learning SNN achieves the same peak correlation ($\rho = 0.7$) as existing SNN training methods when trained offline on real neural data while reducing the total memory usage by 92%. Additionally, it matches state-of-the-art accuracy in a closed loop environment, demonstrates adaptability when subjected to multiple types of neural input disruptions, and is capable of being trained online without any prior offline training.** ***Significance.*** **This work presents a neural decoding algorithm that can be trained rapidly in a closed loop setting. The algorithm increases the speed of acclimating a new user to the system and also can adapt to sudden changes in neural behavior with minimal disruption to the user.**

I. Introduction

Brain machine interfaces (BMI) are systems that process neural signals from the human brain and use the processed signals to control external devices, such as prosthetic limbs [1], [2]. These technologies can be very beneficial to users who have lost limbs or suffered spinal cord injuries by allowing them to have improved mobility and reduced dependence on caretakers. Surveys have indicated that paralyzed patients have generally positive attitudes towards BMIs [3], [4].

BMI systems consists of three major parts: signal acquisition and processing, feature decoding, and device output [5]–[7], as shown in Fig. 1. The best results in kinematic decoding are typically achieved when signal acquisition is performed by an implanted electrode the brain, which allows enough spatial resolution to distinguish the activity of individual neurons [8]. After signal acquisition, there exist a variety of different decoding algorithms that can be used to extract kinematic information from the neural signals. The Kalman filter is often considered the gold standard in neural decoding, having demonstrated superior accuracy to linear decoders in several studies [9]–[11], but machine learning techniques have also recently shown promise [12], [13], especially recurrent networks such as LSTMs [14]. However, these recurrent networks often take several epochs to train, which can require large amounts of memory [15].

Spiking neural networks (SNNs) are a type of artificial network that are particularly natural to apply in a BMI application due to their biological nature. SNNs propagate information in the form of discrete sequences of events called spike trains, and process these spike trains using models that closely mimic the action potentials that drive information flow in real biological neurons [16], [17]. This makes them well suited for BMI tasks since the input data from the electrodes is already in the form of biological spikes, so no additional preprocessing is needed. Additionally, SNNs have been shown to be very power efficient due to the sparse nature of the spiking data [18]–[20], which can be critical for implanted chips, where power dissipation must not exceed 40 mW/cm^2 [21].

Previous work has explored the use of SNNs for neural decoding [22]–[25]. In [24], authors demonstrated an SNN that could replicate the behavior of a Kalman filter using the Neural Engineering Framework. More recent work has shown that SNNs trained with surrogate gradient descent can also provide highly accurate results. In [22] authors demonstrated that SNNs can achieve superior performance to Kalman filters and artificial neural networks for decoding finger velocity, while [25] showed an SNN decoder with comparable accuracy to an LSTM for decoding arm velocity or body position. However, there are still several challenges that need to be addressed before SNNs can become viable decoding options for a robust and accurate BMI. In previous studies, the SNNs were trained in parallel using short training samples recorded at different points in time. This technique, known as batching, is commonly used to make machine learning algorithms more robust [26], but it is less practical for real world BMI use, in which the system would ideally be trained on chronologically sequential data as the user learns to use it. These SNNs also require more memory as the duration of training data increases, since they must store past spiking events to perform backpropagation. This large memory storage during the training process is not practical for a hardware implementation, which is one of the main motivations for developing SNN neural decoders.

E. Taeckens and S. Shah are with the Department of Electrical and Computer Engineering at University of Maryland, College Park, MD, USA e-mail: (sshah389@umd.edu).

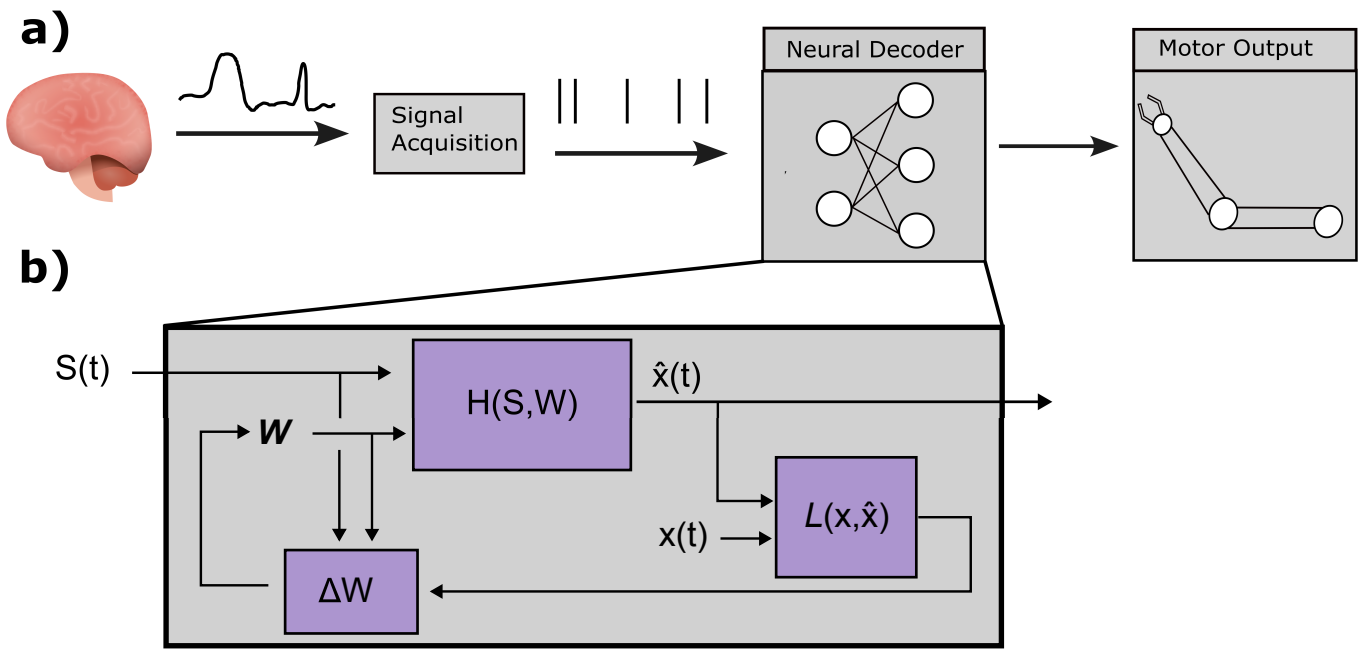

Fig. 1. a)An overview of the main components present in a BMI system. A BMI begins with signal acquisition, in which raw voltage readings from electrodes in the brain are recorded and organized into spike sequences via threshold encoding and spike sorting algorithms. These spike sequences are then sent to a decoder, which extracts kinematic information from the neural data. Predicted kinematics are then sent to a motor output, such as a prosthetic limb. b) A more detailed view of the systems present in a neural decoder with continuous learning. The decoder consists of a function which takes neural spiking data $S(t)$ and a matrix of weights W as inputs. The decoder function outputs a predicted kinematic state, $\hat{x}(t)$. In BMI operation, $\hat{x}(t)$ is used to control a motor output. During training, the predicted kinematics can be compared with ground truth kinematic data $x(t)$ using a loss function, $L(x, \hat{x})$. The loss is used to update the weight values.

Additionally, SNN decoders have not been tested in an online BMI setting, in which the user attempts to use the BMI in real time and receives information about the output of the BMI. Neural decoders trained offline can have quite different results when applied an online setting due to the adjustments made by the user in response to feedback from the BMI system [27]–[30]. Even decoders that perform well initially in an online setting may see a decrease in performance over time due to changes in the structure of nearby neurons or sudden loss of neurons [31], [32]. Additionally, current online decoding methods often require a period of offline training first, in which the user imagines performing movements [30]. A BMI that can be trained entirely online would require no need for imagined movement on behalf of the user, potentially lowering the difficulty for new users to acclimate to a BMI.

This work presents a spiking neural network for neural decoding that can be trained continuously without the need to pause training and perform backpropagation. The SNN utilizes a novel weight update algorithm that does not require storage of past spiking information in memory, allowing it to be trained continuously on arbitrarily long sequences of data. This allows the model to be trained in real time, even in a closed loop setting, giving it the potential to adapt to changes in neural behavior. The model is evaluated in two stages: first, we train it on a data set containing neural and kinematic data from the premotor cortex of macaque primates [33] and benchmark it against the performance of an existing SNN decoder algorithm in an offline training setting. We show that the SNN with continuous learning can achieve comparable accuracy to previous SNN algorithms with significantly reduced peak memory consumption. We also evaluate the model's ability to learn from a single continuous input sequence on the same dataset. Additionally, we demonstrate the algorithm's ability to perform online learning in a simulated closed loop environment and benchmark its performance against a ReFIT Kalman filter, a standard closed loop decoding method [30]. The results from these experiments show that the continuous learning SNN is capable of quickly adapting to sudden changes in neural behavior and can also learn entirely online with no prior training.

II. Spiking Neural Network Design

A. Neuron and Synapse Models

SNNs take as input a sequence of discrete events known as a spike train, defined formally as $S(t) = \sum_{k \epsilon C} \delta(t - t^{(k)})$, where C is the set of all events and $t^{(k)}$ represents the timing of event k in C. SNN layers consist of neuron models that perform operations on the spiking input and produce their own spike trains that propagate on to the next layer. There are a variety of neuron models, all of which seek to imitate the action potentials found in biological neurons. For this study, we use the leaky integrate-and-fire neuron model [34] due to its relative simplicity to implement, but the work can easily be generalized to other models.

The LIF neuron maintains two internal values: the synaptic current, I(t), and the membrane potential, U(t). The synaptic

current and membrane potential of the i-th neuron in a layer L in the network are governed by following:

$$\tau_{syn}\frac{dI_i^l}{dt} = -I_i^l(t) + \sum_j W_{ij}S_j^{l-1}(t) \tag{1}$$

$$\tau_{mem}\frac{dU_i^l}{dt} = (-U_i^l(t) + I_i^l(t))(1 - S_i^l(t)) \tag{2}$$

$$S_i^l(t) = \Theta(U_i^l(t) - \nu) \tag{3}$$

where W represents the matrix of weights applied to the input spikes and τ_{syn} and τ_{mem} are time constants [35]. The neuron emits a spike whenever the membrane potential, U(t), crosses a threshold value, at which point the membrane potential is reset to 0.

B. SNN Architecture

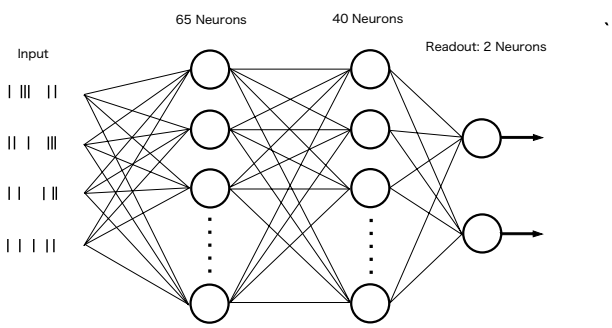

Fig. 2. A graphical depiction of the spiking neural network architecture described below.

The SNN used in this study consists of an input layer of 46 input neurons, two hidden layers of 65 and 40 neurons, respectively, and a readout layer. The sizes of the hidden layers and other parameters were optimized using Bayesian Optimization [36]. A graphical representation of the SNN architecture is shown in Fig. 2.

C. Readout Function

SNNs are typically used either for classification tasks, in which the readout layer consists of several spiking neurons and the output is determined by selecting the neuron with the highest spiking rate [37]–[39], or for modelling spike rates [40]. Neural decoders, however, are designed for regression tasks, in which the decoder outputs two continuous real-valued functions corresponding to the x and y components of the predicted arm velocity. These predicted outputs are then compared with the ground truth x and y velocities to calculate the loss function, typically using mean-squared error. SNN decoders can be adapted to produce real-valued functions by adding a readout layer consisting of two leaky-integrate neurons that never emit spikes [22], [25], where the output is given by the neuron's membrane potential. This method of computing the output has several desirable properties, such as the fact that the membrane potential is a continuous and differentiable function, so there will be no unnatural instantaneous changes in the predicted velocity. The k-th readout value Y_k^l for hidden layer l is given by:

$$\begin{aligned} \tau_{syn}\tfrac{dJ_k^l}{dt} &= -J_k^l(t) + \textstyle\sum_i R_{ki}S_i^l(t) \\ \tau_{mem}\tfrac{dY_k^l}{dt} &= -Y_k^l(t) + J_k^l(t) \end{aligned} \tag{4}$$

where R_{ki} represents a fixed readout weight between the i-th hidden layer neuron and k-th readout neuron.

D. Learning Rules for Training Spiking Neural Network

Traditionally, spiking neural networks were trained using Spike-Timing-Dependent Plasticity learning rule [41]. STDP is based on the observation that when a neuron fires shortly after an input spike, the weight of the synapse connecting the input to the neuron tends to increase [42]. However, the STDP learning rule tends to have lower accuracy, when used for training a network of neurons compared to the error back-propagation. But the spike trains present in the spiking neural networks are non-differentiable.

To address these challenges recent work have shown high levels of accuracy when trained with surrogate gradient descent, a modified version of gradient descent [35]. However, the calculation of these gradients typically involves backpropagation through both the layers of the network and backpropagation through time to account for the effect of previous spiking events on the output. This requires a large amount of memory that is prohibitive for use in a continuous learning setting. In [38], a technique known as deep continuous local learning (Decolle) is introduced that addresses the issues of backpropagation through both space and time. Local learning removes the need for backpropagation through the neural network by assigning a local readout layer with fixed weights to each hidden layer in the network. This technique was previously demonstrated to be successful in a neural decoding setting [25]. However, the SNN in that study used backpropagation through time to calculate the weight updates, since the Decolle algorithm only works for SNNs that are trained on classification problems. Here, we introduce a weight update algorithm for SNNs that are trained on regression problems that does not rely on backpropagation through time. The general formula for determining updating weights via local learning is given by

$$\Delta W_{ij} = -\eta\frac{\partial L^l}{\partial W_{ij}^l} \tag{5}$$

where η is the learning rate and L^l is the loss function for the readout of layer l. When using mean squared error loss, this term becomes

$$\frac{\partial L^l}{\partial W_{ij}^l} = \sum_k (Y_k^l - \hat{Y}_k)\frac{\partial Y_k^l}{\partial W_{ij}^l} \tag{6}$$

where Y_k^l is the kth readout value and $\hat{Y}_k$ is the corresponding target value. Since Y_k^l is a function of the output spikes S_i^l of hidden layer l, $\frac{\partial Y_k^l}{\partial W_{ij}^l}$ can be expressed in terms of the derivative

of $S_i^l(t)$ with respect to the weights. Due to the discontinuous nature of spike trains, the derivative of the spiking function, S_i^l, is not well defined. To fix this issue, the heaviside step function $\Theta(U_i^l(t))$ is replaced with a differentiable function $\sigma(U_i^l(t))$ that closely approximates it to perform surrogate gradient descent, as shown in [35]. With this adjustment, the derivative of the spiking output can be expressed as:

$$\frac{\partial S_i^l(t)}{\partial W_{ij}^l} = \sigma'(U_i^l(t) - 1)\frac{\partial U_i^l(t)}{\partial W_{ij}^l} \tag{7}$$

To find $\frac{\partial U_i^l(t)}{\partial W_{ij}^l}$ without using backpropagation though time, the Decolle algorithm rewrites the equations for I(t) and U(t) using the following equivalent expressions:

$$\begin{aligned} \tau_{syn}\frac{dQ_j^l}{dt} &= -Q_j^l(t) + S_j^l(t) \\ \tau_{mem}\frac{dP_j^l}{dt} &= -P_j^l(t) + Q_j^l(t) \\ U_i^l(t) &= \textstyle\sum_j W_{ij}^l P_j^{l-1} \end{aligned} \tag{8}$$

where $\frac{\partial U_i^l(t)}{\partial W_{ij}^l}$ is simply equal to $P_j^{l-1}(t)$.

In standard Decolle learning, this is enough to fully solve for the weight update, since each Y_k^l is simply a linear function of the the spiking output S_i^l [38]. However, for a SNN with real-valued continuous outputs (in case of regression task such as mapping neural data to kinematics), Y_k^l is not a linear function of $S_i^l(t)$, since it is defined using the same differential equations that govern U(t) for a hidden layer neuron. This means that Y_k^l depends upon past spiking events from the hidden layer neurons. The equivalent expressions used in Decolle to find the derivative of U_i^l cannot be used here, because the derivative is taken with respect to the spikes, not the weights, and unlike the weights the spikes are being integrated. However, we propose to modify the equation to shown that

$$\frac{\partial Y_k^l}{\partial W_{ij}^l} = R_{ki}^l G_{ij}^l \tag{9}$$

where

$$\begin{aligned} \tau_{syn}\frac{dH_{ij}^l}{dt} &= -H_{ij}^l(t) + \frac{\partial S_i^l(t)}{\partial W_{ij}^l} \\ \tau_{mem}\frac{dG_{ij}^l}{dt} &= -G_{ij}^l(t) + H_{ij}^l(t) \end{aligned} \tag{10}$$

where the difference between equations 8 and 10 is that H_{ij}^l integrates the derivative of the output spikes, rather than the spikes themselves. These stored values replace the need for storing the spiking history of the neurons, allowing for regression learning with a fixed cap on memory. Equation 9 is derived from the fact that the dynamics governing the synaptic current and membrane potential for a spiking neuron can be equivalently expressed as the convolution of S_i^l with a kernel function, K(t) [43].

$$Y_k^l(t) = R_{ki}^l \sum_\tau S_i^l(\tau)K(t-\tau) \tag{11}$$

where K(t) is given by

$$K(t) = e^{-t/\tau_{mem}} - e^{-t/\tau_{syn}} \tag{12}$$

Using the property of derivatives of convolutions, it can be shown that

$$\frac{\partial}{\partial W_{ij}}(S_i^l(t) * K(t)) = \frac{\partial S_i^l(t)}{\partial W_{ij}} * K(t) \tag{13}$$

and substituting the derivative of $S_i^l(t)$ into equation 11 yields equation 9. Combining all of these equations yields the full weight update equation:

$$\Delta W_{ij} = -\eta G_{ij}^l(t)\sum_k (Y_k^l - \hat{Y_k})R_{ki} \tag{14}$$

This weight update equation captures accounts for all previous states of both the hidden layer neuron and the readout neurons without requiring any memory overhead to store the historical states of these neurons. It does require supplementary functions in the form of G_{ij}^l and H_{ij}^l, but these only increase the memory complexity to $O(NL)$, where N is the number of input neurons and L is the input layer size. The memory complexity of BPTT is $O(NT)$, where T is the input duration. In BMI applications, L is typically significantly less than T, since SNN performance don't increase significantly after the layer size increases beyond about 50-60 neurons [25], whereas the number of time steps could easily enter the hundreds of thousands for a continuous experiment that lasts longer than an hour, assuming a neural sampling rate of 100 Hz.

Additionally, because the memory does not have to store past states and produces weight update values at every time step, the learning rule proposed in this work is capable of online continuous learning. When training with backpropagation through time, a SNN must first do a forward pass on a training example, and then stop and backpropagate through the training example to calculate the gradients before moving on to the next example. The continuous learning algorithm proposed in this work evaluates the forward pass and weight updates simultaneously, allowing the weights to be updated at every time step without ever needing to pause the experiment.

III. Experimental Procedure and Results

A. Offline Learning

1) Evaluation Data:: The SNN was trained and tested on a data set containing intracortical neural recordings and corresponding kinematic data from macaque primates performing reaching tasks [33], [44]. For each reaching task, the monkey moved a cursor towards an indicated target on a 20 cm by 20 cm computer screen. The velocity of the on-screen cursor was sampled at 1000 Hz. The neural recordings were preprocessed to form spike trains using threshold crossing and spike sorting algorithms. Both the neural and kinematic data were placed into 10 ms bins before being passed to the network, with neural data limited to at most 1 spiking event per bin. This data set was also used to evaluate a neural decoding SNN in [25], allowing for a comparison between the SNNs trained using two learning rules.

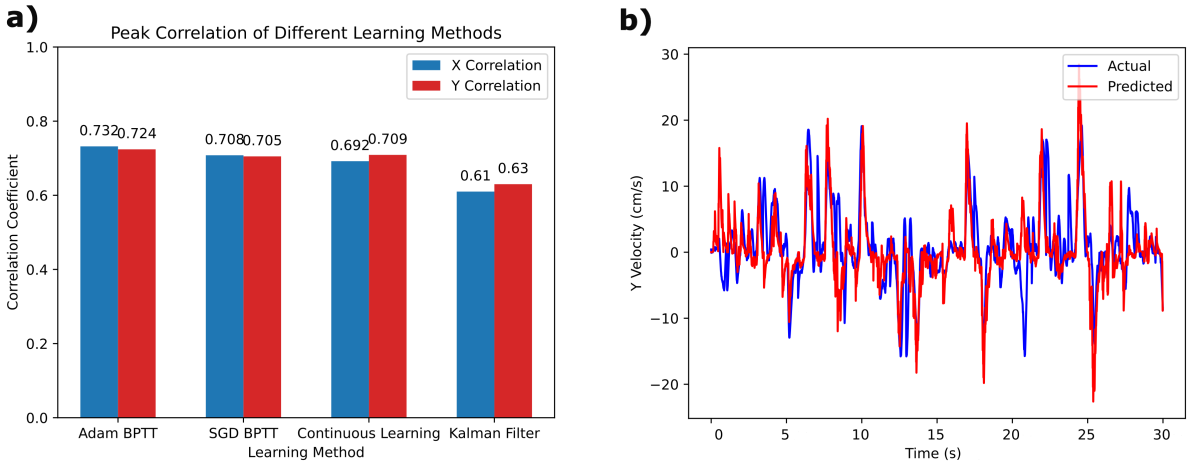

Fig. 3. a) Comparison of the peak correlations achieved by three learning methods: backpropagation-through-time with the Adam optimizer, backpropagation-through-time with stochastic gradient descent, and deep continuous learning. All 3 methods show very close performance and achieve higher peak correlation than a standard Kalman filter. b) A 30 second sample comparing the ground-truth kinematic data with the predicted kinematic data output by continuous learning SNN. The network used to generate this example was trained for 50 epochs.

2) Batched Learning Comparison:: SNN decoders traditionally have been trained using batched learning [22], [25], so the continuous learning model was also trained using batches to provide the best possible comparison. For both models, the data was split into testing and training data sets, and the training data was further split into distinct reaches of at most 1.7 seconds in duration. A batch size of 10 was used, meaning that the models were trained on 10 training examples in parallel at any given time. The models were then evaluated on the testing data using the Pearson Correlation Coefficient to determine the correlation between the model and output and ground truth kinematic data. When training using standard stochastic gradient descent, the continuous learning model and the backpropagation-through-time (BPTT) model achieved approximately the same peak correlation of $\rho = 0.70$, with the continuous learning model requiring fewer training epochs to converge. When using more advanced optimization algorithms, such as Adam, the BPTT model is able to achieve a slightly higher correlation, since the continuous learning rules currently only perform standard gradient descent. Additionally, all methods show a moderately higher correlation than the result achieved by a Kalman filter. Fig. 3 shows the peak correlations achieved by each training method.

3) Memory Footprint: The continuous learning SNN shows a significantly reduced memory footprint compared to the BPTT learning methods. Reach tasks from the data set were typically 1.7 seconds long or less; for an input with a 1.7 second duration and a batch size of 1, the stochastic gradient descent BPTT algorithm had a minimum memory requirement of 2,145 KB to store all necessary spiking data and gradients. By comparison, the continuous learning SNN had a minimum memory requirement of just 161 KB to store all necessary information, regardless of the duration of the input. The BPTT memory requirements also scale linearly with input duration and network size, while the continuous learning memory requirements are invariant to input duration, but scale linearly with input neurons, as shown in Fig. 4. Even with large input sizes of up to 1000 neurons or electrode input channels, the continuous learning method uses less memory when the training duration is longer than about 0.6 seconds. In practice, when analyzed with memory profilers, the cpu required more memory to perform learning with both algorithms. When analyzed with pympler, the continuous learning algorithm required 189 KB of memory. BPTT learning was performed with the Pytorch machine learning library, so we analyzed it with the Pytorch memory profiler. Pytorch used over 92 MB of total memory to perform gradient descent on a 1.7 second input.

4) Continuous Learning: : Current state-of-the-art decoders use batched training, which requires simultaneously training on data that was recorded at different points in time for several epochs. To reduce calibration time and disruptions to the user, a BMI that can learn continuously would make more sense. Surveys conducted on patients with motor impairments show that users prefer BMI systems that offer high accuracy, minimal daily setup, and multifunctionality [3], [45], [46]. To test the continuous learning capabilities of the SNN, we again evaluated it on the premotor cortex dataset, but this time with no batching and without splitting the recording into discrete training examples. Additionally, to test the ability of the network to maintain performance after training stops, we ran the experiment multiple times with different training times. The results show that the SNN converges rapidly, performing most of the learning within the first 300 seconds. The accuracy does not noticeably decline during the testing phase, regardless of when training stops. The peak correlation achieved is less than the peak correlation achieved after multiple epochs of training, but it does achieve the same peak correlation as

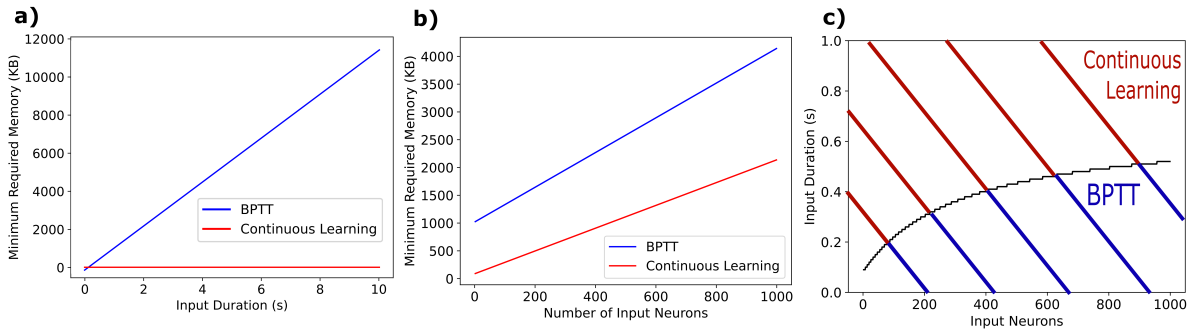

Fig. 4. a) Memory complexity for each learning method when the input size is held at a constant 46 neurons. The memory required to perform continuous learning is constant regardless of input duration, whereas the memory required by BPTT scales linearly with input duration. b) The memory required for both learning methods when input duration is held constant at 1.0 seconds, with an input timestep of 10 ms. scales linearly with number of input channels. The slope of the line for continuous learning is determined by the size of the hidden layers, whereas the slope of the continuous learning line is determined by the input duration. c) The conditions under which each learning method requires less memory, assuming 10 ms time step and hidden layer sizes as specified in section 2.2. Even with a large number of input neurons, BPTT is more efficient only when the input duration is less than 0.6 seconds.

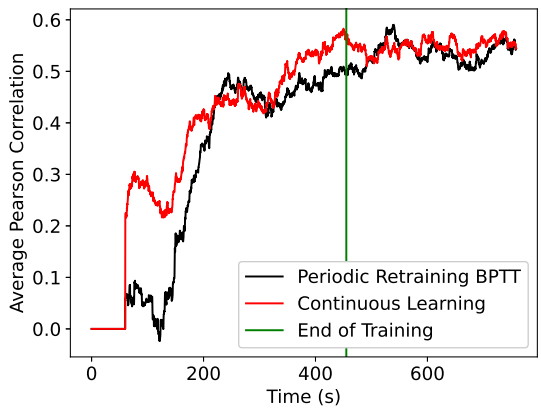

Fig. 5. A sliding window average of the Pearson Correlation achieved by two different SNNs trained on a single epoch of reaching data. A sliding window with a 60 second duration is used to smooth out noise in the plots. The X and Y correlations are averaged for each SNN. Although both models eventually reach the same peak correlation, the continuous learning SNN learns more quickly. Both models maintain their performance after training stops. The figure does not include the time it takes for BPTT to pause to perform backpropagation (an average of 35ms for a reach of 1.7 seconds).

the batched learning did after a single training epoch. We benchmark the continuous learning performance against that of an SNN trained with BPTT. The BPTT algorithm cannot learn continuously, so we instead allow it to pause and perform backpropagation after each individual reaching task (duration 1.0-1.7 seconds). Both methods eventually achieve the same peak correlation, but the continuous learning mechanism learns more quickly, as shown in Fig. 5. Additionally, BPTT takes longer than indicated in the figure, since it must pause after each reach to perform backpropagation. Backpropagation for one reach takes an average of 35 ms, which introduces a delay in training after hundreds of reaches and also interrupts the user.

B. Closed Loop Experiments

While offline experiments on pre-recorded data are commonly used to evaluate neural decoder performance, they do not always correlate well with performance in an online, closed-loop setting [28]–[30]. In a closed loop environment, the output of the BMI is fed directly to the output device, allowing the user to attempt to correct for any errors in the BMI output in real time. This user feedback can have significant impacts on the performance of a BMI system.

1) Experimental Procedure:: Performing a real closed loop experiment on human subjects can be expensive and time consuming. Neural simulations have been shown to produce neural data that more closely mimic the neural responses of real human subjects using a closed loop BMI when compared to real neural data taken from offline experiments [47]. For this work, we adapt the online prosthetic simulator (OPS) described in [29]. The OPS uses the cosine tuning model to determine the spiking rates of neurons [48], in which each neuron has a "preferred direction" and is more likely to fire when the direction of motion is in its preferred direction. The firing rate λ_t of the k-th neuron at time t is given by:

$$\lambda_t = (\lambda_{max} - \lambda_{min})c_k * x_t + \lambda_{min} \tag{15}$$

where λ_{min} and λ_{max} are the minimum and maximum firing rates, respectively, c_k is a unit vector in the neuron's preferred direction, and x_t represents the desired acceleration vector at time t. λ_t was constrained to be greater than or equal to zero. Acceleration was used to determine firing rates because the real neural data used in the previous section showed a much stronger correlation with acceleration than with velocity or position. We sampled direction vectors uniformly from the unit circle. Maximum and minimum firing rates were sampled between 40 and 100 and between 0 and 5 spikes per second, respectively. These values were again chosen to match the firing rates found in real neural data, and match closely with values used in previous OPS simulations. In [29], a Poisson distribution with parameter λ_t was used to determine the number of spikes in each time bin. Since SNNs operate on

individual spikes, and since Poisson random variables can be represented as an approximation of a sum of many Bernoulli variables, we modeled each neuron as a Bernoulli random variable with spiking probability $p = \lambda_t * t_s$ using a spike sampling period t_s = 10 ms.

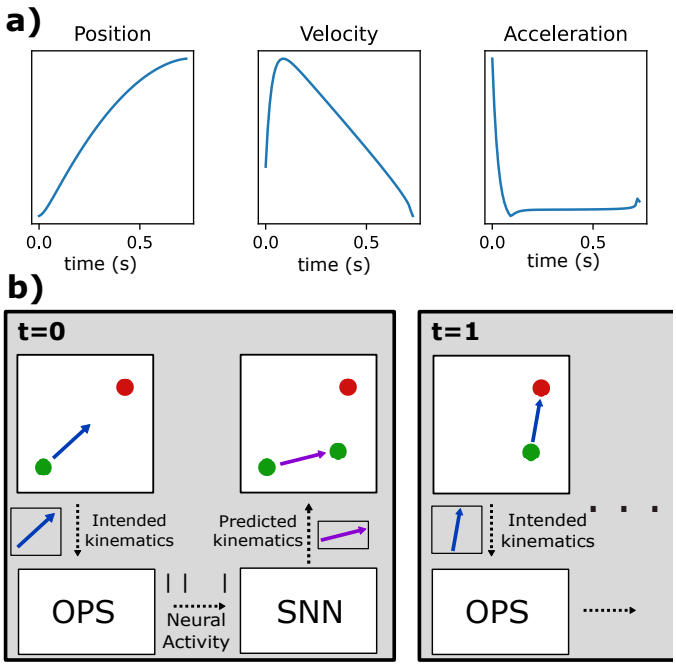

Fig. 6. a) A plot of the magnitudes of the intended position, velocity, and acceleration profiles for a sample reach in which the simulator travels along its intended path. The units of time are in seconds (on x-axis), while the distance units are arbitrary. b) Graphical depiction of the feedback system used by the closed loop simulator. The system starts by generating an intended acceleration vector in the direction of the target, which is sent to the OPS to generate simulated neural spikes. The spiking activity is decoded by the SNN to produce a predicted velocity, which is used to update the current position. At the next time step, a new target acceleration is generated based on the updated position.

The OPS was previously used in [29] to generate simulated neural activity for people performing center-out reaching tasks, with ground truth kinematic data used as the input to generate simulated neural activity. In the absence of real human test subjects, we use a virtual center-out reach task simulation in which targets are placed at random locations a fixed distance away from a central starting point. The ground truth kinematic data that is sent to the OPS is generated by assuming that a real user would always attempt to move in the direction of the target. This assumption has previously been used to train online neural decoders to a high level of accuracy [30]. This, along with constraints on the velocity and acceleration, is used to generate intended kinematic profiles, as shown in Fig. 6a. Target acceleration vectors are then sent to the OPS to generate neural activity, which is in turn decoded by the SNN to produce a predicted velocity and update the position. At the next time step, the new velocity and acceleration vectors go from the current position to the target, as shown in Fig. 6b.

For all experiments described below, a reach is considered successful if the decoder is able to stay within a unit distance of the target for at least 500 ms (the targets were placed 4 unit distances away from the central starting location). Reach tasks are allowed to proceed for a maximum of 3.0 seconds, after which point they will be considered failed. These values are generally in line with previous center-out reach experiments [30], [44].

2) Neural Variability and Disruptions: : One of the main challenges with using BMIs in a closed loop setting is the decrease in decoding quality that can occur as a result of neural disruptions [31]. We performed several closed loop simulations to evaluate the effects of various neural disruptions on the SNN decoding performance. Prior to the closed loop experiments, the SNN was trained on 400 simulated reaches for 5 epochs using batched learning. The reaches were generated using the online prosthetic simulator described above. To provide a benchmark for SNN performance, a ReFIT Kalman filter was also trained offline on 400 simulated reaches. The ReFIT Kalman filter is a modified Kalman filter decoder designed specifically for closed loop BMI experiments that accounts for the lack of uncertainty in the measured position, since the position is determined by the output of the decoder [30].

After offline training, the SNN and Kalman filter both performed 3 closed loop simulation experiments. The performance of each decoder is measured using average time to secure the target, a common metric for evaluating closed loop BMI performance [27], [29], [30]. In the first experiment, 30 out of 46 input neurons were removed after 50 reaches. This simulates a loss of connection to neurons that is common in real BMI systems due to electrodes shifting or becoming damaged [28], [49]. Both decoders showed only a slight increase in average time to target, as shown in Fig. 7, demonstrating that the system can handle sudden neural dropout with minimal disruption.

In the second experiment, the preferred directions of 30 neurons were randomly reassigned. Although real neurons do not suddenly change behavior like this, it is common for electrodes used in implantable BMI to shift during use and start picking up data from a new set of neurons [28], [31], which can be simulated by changing the properties of the existing neurons. Both decoders showed a significant decrease in quality following neural reassignment. The SNN was allowed to train using continuous learning for 30 subsequent reaches, while the Kalman filter was re-calibrated on neural data from the first 30 reaches after reassignment. Here, the SNN with continuous learning can learn without any pause in the usage of the BMI device where as in case of the Kalman filter the BMI system has to be paused for calibration. The SNN was able to return to close to pre-disruption performance, while the Kalman filter performance remained significantly altered even after re-training (Fig. 7). Additionally, the SNN training can be more easily implemented in a real BMI use setting; if a user notices a sudden decrease in quality, they can perform reaching tasks with continuous learning until quality improves. The Kalman filter cannot learn continuously, so to fix a performance drop a user must instead perform an arbitrary number of reaches and then wait for the filter to be retrained

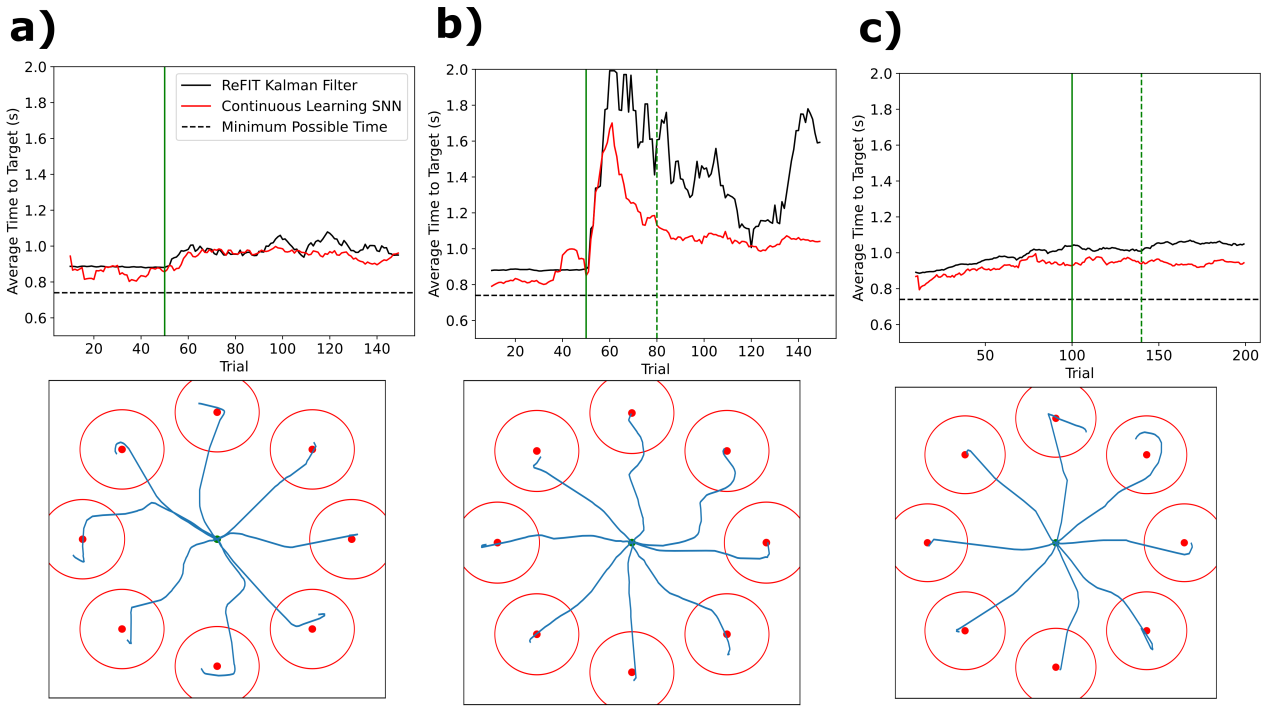

Fig. 7. Plots of average time to target (top) and selected reaching paths (bottom) of simulated closed loop experiments testing three different kinds of neural disruption: neural dropout (a), changing kinematic mapping (b), and shifting firing rates (c). The top graphs show the average time required to secure the target for the SNN (red) and ReFIT Kalman Filter (black), with a sliding window average of 10 reaches used to smooth out the results. The dotted black line shows the minimal possible time under the kinematic constraints of the simulation. The bottom graphs show the path taken by the SNN decoder on eight example reach tasks after any neural disruption and retraining. The red circles around each target represent the window that the decoder has to stay inside for the reach to be considered successful. In (a) and (b), the solid green line represents the time at which the neural disruption occurred, while in (c), it represents the time at which firing rates stabilized. In (b) and (c), SNN continuous training starts at the solid green line and stops at the dotted green line. The Kalman filter is also retrained after the dotted green line.

on the new data.

In the last experiment, the maximum firing rates of the input neurons were allowed to change for the first 80 reaches. Real neurons have been shown to demonstrate changes in their firing rates over time [50], and degrading electrode quality can also cause apparent neural firing rates to decrease an an electrode ages. For this experiment, neurons were assigned a new "target" maximum firing rate between 0 and 30 spikes per second. The neurons approached the new firing rates exponentially with time constants chosen to be normally distributed with a mean of 30 seconds. After firing rate drift, the SNN trained continuously for 40 reaches, while the Kalman filter was re-calibrated after 40 reaches. Both systems show a moderate decrease in performance which does not appear to improve after retraining. This suggests that although the decoder is fairly robust to changing firing rates, it is not capable of compensating for those changing rates by additional training.

3) Closed Loop Learning: : An additional simulation was performed to see if the SNN was capable of learning in a closed loop setting without prior offline training. An untrained SNN was trained online using continuous learning for 20 reaches, and then tested for an additional 80 reaches post training; Fig.8 shows the average time to secure the target. The SNN converged rapidly, achieving average times under 1.0 seconds (the minimum possible time to secure the target is 740 ms). For comparison, an untrained ReFIT Kalman filter was also subjected to a similar trial. The first 20 reaches were conducted online using parameters that were randomized and constrained to a range of values that would produce reasonable outputs. After 20 reaches, the Kalman filter was fit to the neural data from the twenty training reaches and tested for 80 additional online reaching tasks. Unlike the SNN, the Kalman filter was unable to learn from the online reaching tasks, and only managed to reach the target within the required 3 second time window on 6 out of 80 testing reaches. This demonstrates that the continuous learning SNN is capable of learning entirely online, without a need for imagined offline reaches, which could make BMI calibration much easier for new users.

IV. Discussion and Conclusion

A. Significance

The continuous weight update mechanism for neural decoding SNNs presented in this paper has two significant applications for practical brain machine interfaces. First, it allows training on an arbitrary duration input with a fixed amount of memory. This makes it significantly easier to

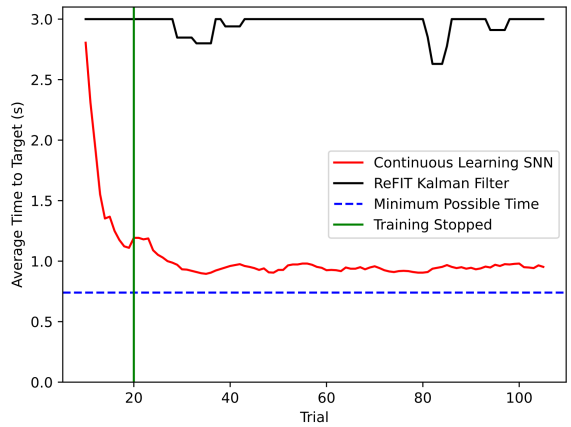

Fig. 8. A plot showing the average time taken to reach the target for a continuous learning SNN and Kalman filter that received no prior offline training. A sliding window average of 4 trials is used. The SNN learns rapidly, requiring only 20 reaches of online trials to achieve the same average time to target achieved by the decoders that were trained offline in the previous experiment. The Kalman filter is unable to learn from online trials, and typically fails to acquire the target, taking the maximum 3.0 seconds for most trials.

implement the algorithm in hardware, since there is no need for dynamic allocation of memory or introducing a hard limit on the input duration. One of the primary motivations for using SNNs is their power efficiency [25], which could enable safe implantation of a decoder directly inside a user's brain, with no need for connection to an external device. In order to realize this power efficiency, a hardware implementation will be necessary, and this work takes an important step in that direction by introducing learning with a finite amount of memory. Future work should aim to implement the algorithm in a hardware system and measure its true power efficiency.

Additionally, the continuous nature of the weight update mechanisms presented here make the SNN ideal for use in a closed loop system. Adaptability is an important trait for online decoders, since implanted electrodes tend to shift over and degrade in quality over time, which can cause decreases in neural performance. The SNN presented here demonstrates an ability to respond to multiple different types of neural disruptions. It's performance does not change significantly in response to neural dropout or changing neural firing rates, and it is able to quickly adapt when the kinematic mapping of the input neurons changes. The SNN can also train rapidly in an exclusively online setting, with no need for prior offline training, an ability not shared by standard online decoders such as the ReFIT Kalman filter. This could reduce the difficulty of BMI calibration tasks for new users by removing the need for offline training in which the user must imagine a movement without physical feedback.

B. Future Research

Future research should explore methods to optimize the SNN learning algorithm. Currently, the weight update algorithm is based on stochastic gradient descent. As shown in results, the continuous learning algorithm falls slightly short of the peak accuracy achieved by BPTT when trained offline with the Adam optimizer. Future work should investigate whether the accuracy of the continuous learning SNN can be improved by implementing more sophisticated optimization techniques, such as momentum.

Subsequent studies should also investigate the performance of the SNN in a real closed loop trial using animal model. Simulations have been demonstrated to be reliable predictors of actual closed loop performance, but a verification is still an important step before the algorithm is ready for use in an actual clinical care setting.

V. Acknowledgments

This work was funded in part by the UMD MTECH ASPIRE scholarship.

References

[1] S. Musallam, B. D. Corneil, B. Greger, H. Scherberger, and R. A. Andersen. Cognitive Control Signals for Neural Prosthetics. *Science*, 305(5681):258–262, July 2004. Number: 5681 Publisher: American Association for the Advancement of Science Section: Report.

[2] A. Jackson, C.T. Moritz, J. Mavoori, T.H. Lucas, and E.E. Fetz. The neurochip bci: towards a neural prosthesis for upper limb function. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14:187–190, 2006.

[3] J. L. Collinger, M. L. Boninger, T. M. Bruns, K. Curley, W. Wang, and D. J. Weber. Functional priorities, assistive technology, and brain-computer interfaces after spinal cord injury. *J. Rehabil. Res. Dev.*, 50, 2013.

[4] Jacob Lahr, Christina Schwartz, Bernhard Heimbach, Ad Aertsen, Jorn Rickert, and Tonio Ball. Invasive brain–machine interfaces: a survey of paralyzed patients' attitudes, knowledge and methods of information retrieval. *J. Neural Eng.*, July 2015.

[5] Simanto Saha, Khondaker A. Mamun, Khawza Ahmed, Mostafa Raqibul, Ganesh R. Naik, Sam Darvishi, Khandoker Ahsan H., and Mathias Baumert. Progress in brain computer interface: Challenges and opportunities. *Frontiers in System Neuroscience*, 15, 2021.

[6] Jerry Shih, Dean J. Krusienski, and Jonathan R. Wolfpaw. Brain-computer interfaces in medicine. *Mayo Clinic Proceedings*, 87, 2012.

[7] Andrew B. Schwartz, X. Tracy Cui, Douglas J. Weber, and Daniel W. Moran. Brain-controlled interfaces: Movement restoration with neural prosthetics. *Neuron*, 52, 2006.

[8] Shoeb Shaikh and Arindam Basu. Intelligent intracortical brain-machine interfaces: Next generation of scalable neural interfaces. In Mohamad Sawan, editor, *Handbook of Biochips*, pages 869–889. Springer New York, 2022.

[9] W. Wu, M. Black, Y. Gao, M. Serruya, A. Shaikhouni, , J.R. Donoghue, and Elie Bienenstock. Neural decoding of cursor motion using a kalman filter. In *Advances in Neural Information and Processing Systems*. NeurIPS, 2002.

[10] W. Wu, A. Shaikhouni, J.R. Donoghue, and M.J. Black. Closed-loop neural control of cursor motion using a kalman filter. In *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2004.

[11] Sung-Phil Kim, John D Simeral, Leigh R Hochberg, John P Donoghue, and Michael J Black. Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia. *Journal of Neural Engineering*, 5:455–476, 2008.

[12] Joshua I. Glaser, Ari S. Benjamin, Raeed H. Chowdhury, Matthew G. Perich, Lee E. Miller, and Konrad P. Kording. Machine Learning for Neural Decoding. *eNeuro*, 7(4), July 2020. Publisher: Society for Neuroscience Section: Research Article: Methods/New Tools.

[13] Fangyu Liu, Saber Meamardoost, Rudiyanto Gunawan, Takaki Komiyama, Claudia Mewes, Ying Zhang, EunJung Hwang, and Linbing Wang. Deep learning for neural decoding in motor cortex. *J. Neural Eng.*, September 2022.

[14] Sahil Shah, Benyamin Haghi, Spencer Kellis, Luke Bashford, Daniel Kramer, Brian Lee, Charles Liu, Richard Andersen, and Azita Emami. Decoding Kinematics from Human Parietal Cortex using Neural Networks. In *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 1138–1141, March 2019. ISSN: 1948-3554.

[15] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997.

[16] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, October 1990. Conference Name: Proceedings of the IEEE.

[17] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic Silicon Neuron Circuits. *Frontiers in Neuroscience*, 5, 2011.

[18] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R. Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V. Arthur, Paul A. Merolla, and Kwabena Boahen. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, 102(5):699–716, May 2014.

[19] Jongkil Park, Sohmyung Ha, Theodore Yu, Emre Neftci, and Gert Cauwenberghs. A 65k-neuron 73-Mevents/s 22-pJ/event asynchronous micro-pipelined integrate-and-fire array transceiver. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, pages 675–678, October 2014. ISSN: 2163-4025.

[20] Andreas G. Andreou, Kwabena A. Boahen, Philippe O. Pouliquen, Aleksandra Pavasovic, Robert E. Jenkins, and Kim Strohbehn. Current-mode subthreshold MOS circuits for analog VLSI neural systems. *IEEE Transactions on neural networks*, 2(2):205–213, 1991. Publisher: IEEE.

[21] Julie Dethier, Paul Nuyujukian, Stephen I Ryu, Krishna V Shenoy, and Kwabena Boahen. Design and validation of a real-time spiking-neural-network decoder for brain–machine interfaces. *Journal of Neural Engineering*, 10:036008, 2013.

[22] Jiawei Liao, Lars Widmer, Xiaying Wang, Alfio Di Mauro, Samuel R. Nason-Tomaszewski, Cynthia A. Chestek, Luca Benini, and Taekwang Jang. An energy-efficient spiking neural network for finger velocity decoding for implantable brain-machine interface. In *4th International Conference on Artificial Intelligence Circuits and Systems*. IEEE, 2022.

[23] Julie Dethier, Paul Nuyujukian, Chris Eliasmith, Terrence Stewart, Shauki Elasaad, Krishna V Shenoy, and Kwabena A Boahen. A Brain-Machine Interface Operating with a Real-Time Spiking Neural Network Control Algorithm. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[24] Julie Dethier, Paul Nuyujukian, Stephen I. Ryu, Krishna V. Shenoy, and Kwabena Boahen. Design and validation of a real-time spiking-neural-network decoder for brain–machine interfaces. *Journal of Neural Engineering*, 10(3):036008, April 2013. Publisher: IOP Publishing.

[25] Elijah Taeckens, Ryan Dong, and Sahil Shah. A biologically plausible spiking neural network for decoding kinematics in the hippocampus and premotor cortex. In *11th International IEEE EMBS Conference on Neural Engineering*. IEEE, 2023.

[26] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information and Processing Systems*, 2018.

[27] Shinsuke Koyama, Steven M Chase, Andrew S Whitford, Meel Velliste, Andrew B Schwartz, and Robert E. Kass. Comparison of brain-computer interface decoding algorithms in open-loop and closed-loop control. *Journal of Computational Neuroscience*, 29, 2010.

[28] Siddharth Dangi, Suraj Gowda, Rodolphe Heliot, and Jose M. Carmena. Adaptive kalman filtering for closed-loop brain-machine interface systems. In *5th International IEE/EMBS Conference on Neural Engineering*. IEEE, 2011.

[29] John P. Cunningham, Paul Nuyujukian, Vikash Gilja, Cindy A. Chestek, Stephen I. Ryu, and Krishna V. Shenoy. A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces. *Journal of Neurophysiology*, 105, 2011.

[30] Vikash Gilja, Paul Nuyujukian, Cindy A. Chestek, John P. Cunningham, Byron M. Yu, Joline M. Fan, Stephen I. Ryu, and Krishna V. Shenoy. A brain machine interface control algorithm designed from a feedback control perspective. In *International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2012.

[31] Michael D Linderman, Gopal Santhanam, Caleb T. Kemere, Vikash Gilja, Stephen O'Driscoll, Byron M. Yu, Afsheen Afshar, Stephen I. Ryu, Krishna V. Shenoy, and Teresa H. Meng. Signal processing challenges for neural prostheses. *IEEE Signal Processing Magazine*, 2008.

[32] David Sussillo, Sergey D. Stavisky, Jonathan C. Kao, Stephen I. Ryu, and Krishna V. Shenoy. Making brain–machine interfaces robust to future neural variability. *Nature Communications*, 7:13749, 2016.

[33] Matthew G. Perich, Patrick N. Lawlor, Konrad P. Kording, and Lee E. Miller. Extracellular neural recordings from macaque primary and dorsal premotor motor cortex during a sequential reaching task, 2018.

[34] Giacomo Indiveri. A low-power adaptive integrate-and-fire neuron circuit. In *Proceedings of the 2003 International Symposium on Circuits and Systems*, pages IV–820–IV–823, Bangkok, Thailand, 2003. IEEE.

[35] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate Gradient Learning in Spiking Neural Networks. *arXiv:1901.09948 [cs, q-bio]*, May 2019. arXiv: 1901.09948.

[36] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, Lake Tahoe, Nevada, December 2012. Curran Associates Inc.

[37] John J. Wade, Liam J. McDaid, Jose A. Santos, and Heather M. Sayers. Swat: A spiking neural network training algorithm for classification problems. *IEEE Transactions on Neural Networks*, 21, 2010.

[38] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE). *Frontiers in Neuroscience*, 14:424, May 2020. arXiv:1811.10766 [cs].

[39] Sayma N. Chowdhury and Sahil Shah. Hardware aware modeling of mixed-signal spiking neural network. In *20th IEEE Interregional NEWCAS Conference*. IEEE, 2022.

[40] H Rostro-Gonzalez, B Cessac, and T Vieville. Parameter estimation in spiking neural networks: a reverse-engineering approach. *J. Neural Eng.*, March 2012.

[41] Natalia Caporale and Yang Dan. Spike timing-dependent plasticity: a Hebbian learning rule. *Annual Review of Neuroscience*, 31:25–46, 2008.

[42] Guo-qiang Bi and Mu-ming Poo. Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type. *The Journal of Neuroscience*, 18(24):10464–10472, December 1998.

[43] Tian Gao, Bin Deng, Jiang Wang, and Yi Guosheng. Highly efficient neuromorphic learning system of spiking neural network with multi-compartment leaky integrate-and-fire neurons. *Frontiers in Neuroscience*, 16, September 2022.

[44] P.N. Lalower, M.G. Perich, L. Miller, and K.P. Kording. Linear-nonlinear-time-warp-poisson models of neural activity. *Journal of Computational Neuroscience*, 2018.

[45] G.J. Snoek, M.J. IJzerman, H.J. Hermens, D. Maxwell, and F. Biering-Sorensen. Survey of the needs of patients with spinal cord injury: impact and priority for improvement in hand function in tetraplegics. *Spinal Cord*, 42, September 2004.

[46] K.D. Anderson. Targeting recovery: priorities of the spinal cord-injured population. *J. Neurotrauma*, 21, October 2004.

[47] Rudolphe Heliot, Karunesh Ganguly, Jessica Jimenez, and Jose M. Carmena. Learning in closed-loop brain-machine interfaces: modeling and experimental validation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40, 2011.

[48] A.P. Georgopoulos, A.B. Schwartz, and R.E. Kettner. Neuronal population coding of movement direction. *Science*, 26, 1987.

[49] Vadim S. Polikov, Patrick A. Tresco, and William M. Reichart. Response of brain tissue to chronically implanted neural electrodes. *J. Neurosci Methods*, September 2005.

[50] Karunesh Ganguly and Jose M. Carmena. Emergence of a stable cortical map for neuroprosthetic control. *PLoS Biology*, 7, July 2009.