Article | Received 15 March 2025; Accepted 19 June 2025; Published 30 June 2025
https://doi.org/10.55092/neuroelectronics20250006

Low-complexity reinforcement learning decoders for autonomous, scalable, neuromorphic intra-cortical brain machine interfaces

Aayushman Ghosh[1,2,†], Shoeb Shaikh[3,†], Biyan Zhou[4], Pao-Sheng Vincent Sun[4], Camilo Libedinsky[5,6], Rosa Q. So[7,8] and Arindam Basu[4,*]

[1] Department of Electronics and Telecommunication Engineering, Indian Institute of Engineering Science and Technology, Shibpur, India

[2] Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign, Urbana, Illinois 61801, USA

[3] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore

[4] Department of Electrical Engineering, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong SAR, China

[5] Institute of Molecular and Cell Biology (IMCB), Agency for Science, Technology and Research (A*STAR), Singapore 138673, Singapore

[6] Department of Psychology, National University of Singapore, Singapore 117570, Singapore

[7] Institute for Infocomm Research (I^{2}R), Agency for Science, Technology and Research (A*STAR), Singapore 138632, Singapore

[8] Department of Biomedical Engineering, National University of Singapore, Singapore 117583, Singapore

†These authors contributed equally to this work.

* Correspondence author; E-mail: arinbasu@cityu.edu.hk.

Highlights:

- Comparing the computational cost of Banditron and Banditron-RP against state-of-the-art RL algorithms: AGREL, HRL, and Deep Q-Learning.
- Comparing discrete-state decoding performance of neuromorphic online learning algorithms like Banditron and Banditron-RP against state of the art batch training based RL algorithms across four datasets spanning multiple days of recording.
- Investigating the impact of erroneous and sparse nature of reward signal on the above reported RL algorithms.
- Comparing RL algorithms decoding performance against popular supervised classification methods – Linear Discriminant Analysis (LDA) and Support Vector Machines (SVM). Do note that this is not

a one-to-one comparison as RL and supervised methods employ altogether different paradigms of learning.

- Introduce two new open source primate datasets for neural decoding.

Abstract: Intra-cortical brain machine interfaces (iBMIs) with wireless capability could scale the number of recording channels by integrating an intention decoder to reduce data rates. However, the need for frequent retraining due to neural signal non-stationarity is a big impediment. This paper presents an alternate neuromorphic paradigm of online reinforcement learning (RL) with a binary evaluative feedback in iBMIs to tackle this issue. This paradigm eliminates time-consuming calibration procedures. Instead, it relies on updating the model on a sequential sample-by-sample basis based on an instantaneous evaluative binary feedback signal. Such online learning is a hallmark of neuromorphic systems and is different from batch updates of weight in popular deep networks that is very resource consuming and incompatible with constraints of an implant. In this work, using open-loop analysis on pre-recorded data, we show application of a simple RL algorithm—Banditron in discrete-state iBMIs and compare it against previously reported state of the art RL algorithms—Hebbian RL (HRL), Attention Gated RL (AGREL), deep Q-learning. Owing to its simplistic single-layer architecture, Banditron is estimated to be at least two orders of magnitude of reduction in power dissipation compared to state of the art RL algorithms. At the same time, offline analysis performed on four pre-recorded experimental datasets procured from the motor cortex of two non-human primates performing joystick-based movement-related tasks indicate Banditron performing significantly better than state of the art RL algorithms by at least ∼5%, 10%, 7% and 7% in experiments 1, 2, 3 and 4 respectively. Furthermore, we propose a non-linear variant of Banditron—"Banditron-RP", which gives an average improvement of 6% and 2% in decoding accuracy in experiments 2 and 4 respectively with only a moderate increase in computations (and concomitantly power consumption).

Keywords: brain-machine interface; neuromorphic; reinforcement learning; hardware-friendly

1. Introduction

It is estimated that around 1 in 50 people worldwide are afflicted with paralysis [1]. Intra-cortical brain-machine interfaces (iBMIs) are essential to restore quality of life for patients suffering from debilitating paralytic conditions such as tetraplegia, quadriplegia, stroke induced paralysis among others. Neural spikes recorded from the surface of brain areas associated with movement (e.g. primary motor cortex – M1) serve as an input to iBMI systems. The effector outputs vary depending on the application. For example, in order to enable communication, real-time demonstrations such as employing a cursor to type out messages have been exhibited [2–4]. Likewise for locomotion, real time demonstrations in the form of wheelchair [5] full-body exoskeleton [6] have been reported.

Notwithstanding the impressive advances made in the field of iBMIs, practical challenges persist in clinical translation. Most iBMI systems use wired connection to bulky computers reducing patient mobility and increasing risk of infection [7]. Wireless iBMIs suffer from scalability issues due to exploding data rates [8]. A potential solution for data compression is including an intention decoder in the implant

[7] while using special design techniques to reduce power dissipation of the decoder circuit. Table 1 compares different recently reported decoders. Most approaches use simple 1–2 layer networks to fit the power budget of implants. However, online learning is used in a spiking neural network hardware [9], but complex decoding tasks involving human or non-human primate models are not demonstrated.

However, the need for frequent retraining to counter neural signal non-stationarity makes the decoder-integrated implants difficult to use. The current state-of-the-art iBMI systems involve time-consuming daily calibration procedures amounting to several minutes at the beginning of a given day/session, which greatly affects the ability of iBMI users, suffering from debilitating impairments, to remain alert during these prolonged calibration routines [10]. Lastly, these systems employ supervised learning algorithms to learn the mapping from input neural signal to output effector movement [11]. The supervised learning paradigm requires a ground truth target label to be available at every instance of time to train an appropriate model. In case of patients who cannot move their limbs to control effector kinematics, getting this ground truth target label is non-trivial. It involves carefully designed calibration trials that are designed with the assumption that the ground truth target label is always pointing towards the direction of the eventual goal in the designed trial [12,13]. These requirements often necessitate the supervision of a neural engineer to ensure smooth operation of the iBMI system.

Table 1. Comparison of intention decoder integrated circuits.

Author (Journal)	**Chen (TBCAS'16)** [14]	**Boi (Frontiers'16)** [9]	**An (TBCAS'22)** [15]	**Shaeri (JSSC'24)** [16]
Channels	128	15	93	512
Feature	Spike rate	Spike rate	Spike band power	Spike rate
Online learning	N	Y	N	N
Closed-loop	N	Y	Y	N
Decoder Model	Extreme Learning Machine, 2 layer NN	Spiking Neural Network, 1–2 layers	Steady State Kalman Filter, 1 layer	Distinct Neural Codes + Linear Discriminant Analysis, 2 layer
Task	Finger movement, Human	Prosthetic Control, Rodent	Finger movement, Primate	Handwriting, Auditory, Human/Rodent
Power (μW)	0.4*	4000*	588	225

* Power of MCU not included.

One approach to tackle the problem of non-stationarity is to train a complex deep network with sufficient generalization capability [17]. Yet another recent approach is to use domain adaptation to align the previous domain (past data) to the new domain (current data) by using networks such as cycle GAN [18]. However, such complex decoders cannot be integrated in an implant due to tight area and power constraints. Hence, in this work we propose applying online reinforcement learning based low-complexity methods that (a) learn the neural input to effector movement mapping on incoming samples on a sample-by-sample basis, thereby getting rid of explicit calibration procedures, and (b) do not require explicit measurement of effector kinematics and learn with a simple scalar reward that is obtained while interacting with the external environment. The proposed approach is neuromorphic since

it adopts a continuous update or online learning strategy like the brain, and unlike batch learning as used in conventional deep learning. Second, the learning algorithm is chosen to be hardware efficient, another hallmark of neuromorphic systems [19]. Closed-loop biomedical interfaces is expected to be a prime application for adaptive neuromorphic systems [20–22] and to the best of our knowledge, this is the one of the first works demonstrating its usage for continuous learning in iBMI.

Promising RL based iBMI implementations have been reported in the past. This work aims to build up and extend on the reported techniques. Popular RL algorithm used by the DeepMind team in creating AlphaGo [23], Q-learning, was reported in [24] wherein rats controlled a prosthetic arm to choose between two targets. The limitation of this approach is that the large neural state-action pair gives rise to the curse of dimensionality problem leading to generalization difficulty. To overcome this problem, [25] proposed applying Attention-gated reinforcement learning (AGREL) and its variants [26,27]. The improvements are reported over Q-learning involving one NHP performing a center-out cursor task. An important point to bear in mind is that the reported approaches trained the RL-models in batches implying that these methods would still require an explicit block of training similar to the state of the art supervised methods. Alternatively, a recent effort presented an iBMI based on Hebbian Reinforcement learning (HRL) algorithm that is trained in an online fashion at each time-step, which allows us to get rid of an explicit calibration block and results have been reported in two NHPs performing a two-target discrete task [28]. Furthermore, a simplified version of a binary scalar reward to train the RL-model and preliminary results have reported biological sources to contain this reward information [28–30]. However, all the reported approaches use multi-layer neural networks that are prone to the local optimum problem [27].

Besides generalization issues, multi-layer neural networks based approaches are found to be too resource intensive to be deployed in an implantable fashion [7,31]. To overcome the issues of generalization and implantability, we present the application of Banditron [32] – a lightweight, single-layer, online prediction algorithm in an iBMI setting. Furthermore, we propose and present a non-linear variant of Banditron, Banditron-RP (Random Projection). The RP method is related to neuromorphic population decoding approaches popular under the names of neural engineering framework [33] or reservoir computing [34]. We further show detailed comparisons of the computational complexity of our approaches with traditional deep RL methods in Section 5.4.

Thus, in summary, we are presenting an alternate online BMI paradigm along the lines of [28], wherein we do not explicitly collect training, validation data as supervisory training scheme (see Figure 1(a)). Rather, we propose to train our model sequentially with incoming data on a sample by sample basis (see Figure 1(b–d)). Conventional Deep Learning (DL) requires a large amount of data for training – this means the patient cannot use the decoder properly in this duration. Whenever the decoder performance degrades due to neural variability, this retraining procedure has to be run which suspends the decoder. On the other hand, our proposed method is constantly learning from mistakes and keeps on updating while performing the decoding thus removing or reducing fatigue-inducing calibration routines. Many studies have found user independence to be a top priority for patients [35]. Hence, suspending the use of the decoder for retraining with the latest neural recording data is not good from an user perspective. In the experiments reported in this paper, the time required to conduct calibration routines ranges from approximately 5 to 14 minutes which is approximately one-third of the total experiment time in our

experiments. This methodology is consistent across all the reported RL algorithms.

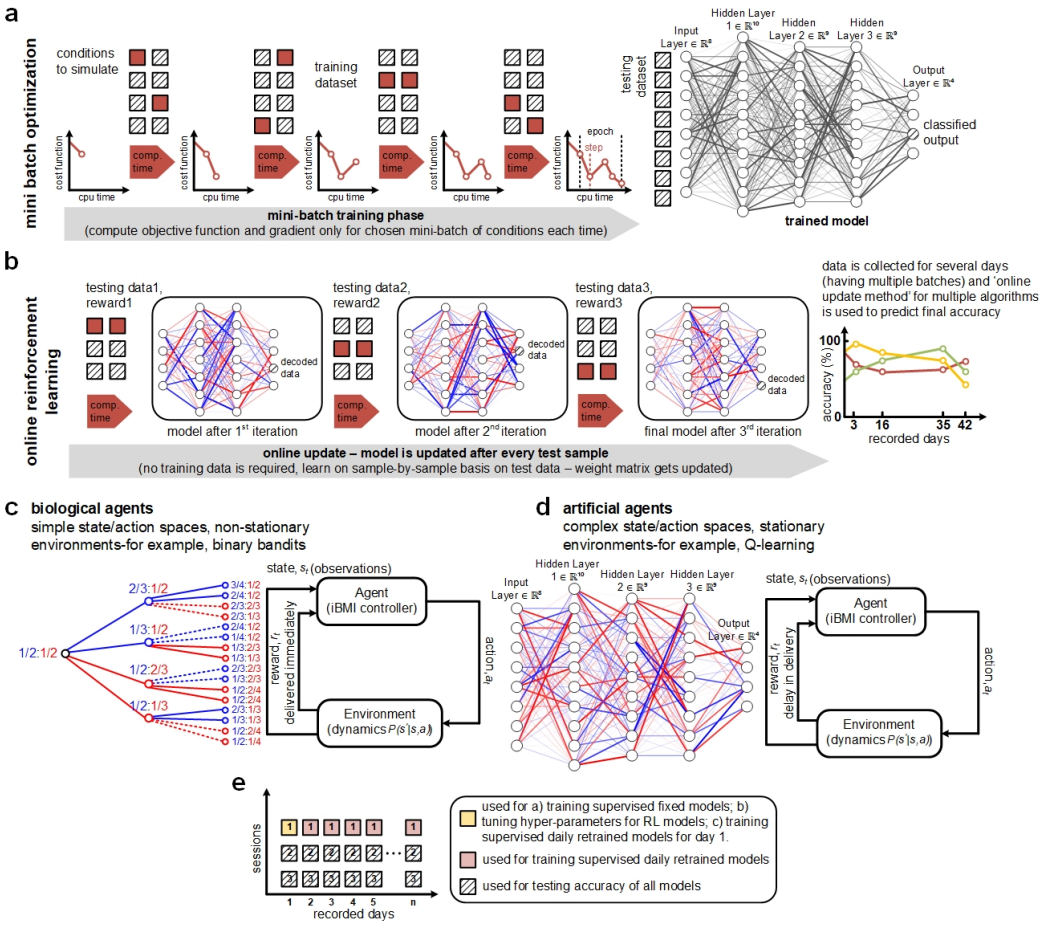

Figure 1. **(a)** Visualization of mini-batch optimization based methods. The independent experimental conditions are randomly divided into disjoint subsets, the mini-batches (depicted as squares). Per the optimization step, only the contribution of the chosen mini-batch is evaluated (red squares). Hence, possibly many optimization steps can be performed during one epoch, which is the time until the whole dataset has been evaluated. **(b)** Neuromorphic Online Reinforcement Learning based learning schemes. Online reinforcement learning-based models [7] do not employ only training data for learning. Instead, they learn also during the test phase by virtue of a scalar feedback at each discrete time interval $t = i$. **(c)** Learning in biological agents. RL is based on the interaction between between an agent and its environment. Agents choose actions, a_t, which lead to changes of the state, s_t, and rewards, r_t, which are returned by the environment. The agent's goal is to maximize rewards over a defined time horizon. **(d)** Learning in Artificial agents. The same agent–environment distinction is important in artificial systems. In state-of-the-art artificial RL systems, action values are estimated by training deep networks. **(e)** *Train-Test* split. The first session's data of every recorded day was used for training *retrained* supervised models across all pre-recorded datasets. First session's data of the first day of recording was used to build *fixed* supervised models as well as tune hyper-parameters associated with Reinforcement Learning models. Sessions 2 and 3 of each day are used as test data across all techniques.

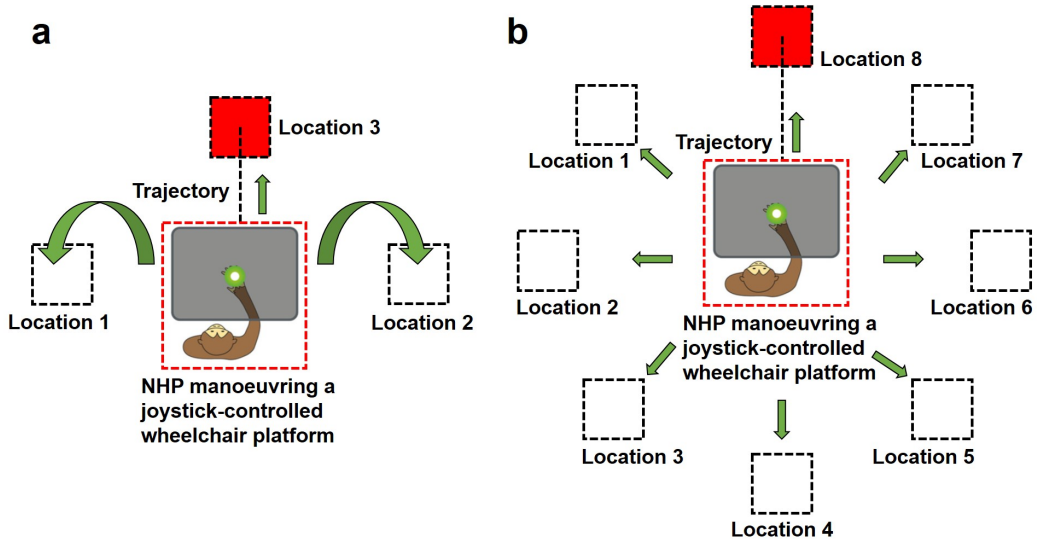

Figure 2. **(a)** This cartoon represents the general nature of the behavioural task performed in experiments 1, 2 and 3. In experiments 1 and 2, the nonhuman primate (NHP) is trained to maneuver a joystick-controlled wheelchair whose position updates every 100 *ms*. The wheelchair starts from a central position in every trial to reach the target over several time-steps, appearing in one of the three square-shaped locations [5]. However, in experiment 3, closed loop brain control was employed and decoded commands from a discrete classifier were used for controlling wheelchair motion. Wheelchair in both scenarios of brain and joystick control was driven at a frequency of 10 *Hz*. **(b)** represents a center-out task corresponding to experiment 4, wherein the NHP was trained to move a joystick-controlled cursor starting from a central position on a computer screen to one of the eight different square-shaped target locations. Cursor position was updated every 100 *ms* and the NHP reached the target over several time-steps.

2. Materials and methods

2.1. Signal acquisition and behavioural description

We have used two adult male cynomolgus macaques (Macaca fascicularis) to conduct our experiments. We will refer to them as non human primate (NHP)-A and NHP-B respectively. A titanium head post (Crist Instruments, MD, USA) was affixed to each subject's skull prior to implantation of microelectrode arrays. In NHP-A, 4 microelectrode arrays containing 16 electrodes each [5], and in NHP-B, 1 microelectrode array containing 100 electrodes were implanted in the hand/arm region of the left primary motor cortex respectively [7,36]. Threshold crossings from the sensed (recorded) neural signals were used as inputs [5,7,37]. Interested readers are referred to [5] to gain a deeper understanding of the experimental protocols used in this study. Raw datasets extracted from experiments 1 and 3 were already publicly available at: https://osf.io/dce96/. Through this work, we are releasing data from experiment 2 and 4 (including processed data for experiments 1 and 3) at: https://zenodo.org/records/15487063. A pictorial representation of the tasks can be seen in Figure 2 and a brief description is provided below.

2.1.1. Experiment 1

In this experiment (see Figure 2(a)), a robotic wheelchair bound NHP-A was controlling its motion through a three-directional spring-loaded joystick [5]. The experiment comprised of four tasks: (a) turning 90^o right, (b) moving forward by 2 *m*, (c) turning 90^o left, and (d) staying still for 5 seconds (*stop* task). A trainer holding a piece of reward served as a visual target and a trial was considered successful if NHP-A managed to reach the trainer within an elapsed time of 15 seconds while the wheelchair avatar was driven at a frequency of 10 Hz.

2.1.2. Experiment 2

In this experiment, NHP B was seated in a primate chair facing a computer screen and was trained to manipulate a three-directional joystick similar to experiment 1. This experiment consisted of three tasks, with a target being presented in a pseudo-random manner in one of the following locations—Top, right and left relative to the centre of the screen. The target was in the form of a red square of side 2 *cm* on a black background. Each trial began with a wheelchair avatar being displayed at the centre of the screen along with the target. A trial was considered successful if NHP-B managed to reach and stay in the target area for under a total elapsed time of 13 seconds. A juice reward was dispensed for every successful trial and the wheelchair avatar was driven at a frequency of 10 Hz.

2.1.3. Experiment 3

This experiment is similar to experiment 1 with the difference that the joystick was removed and closed-loop brain control was employed to drive the wheelchair driven at a frequency of 10 Hz. A Linear Discriminant Analysis (LDA) classifier based decoder was calibrated to achieve closed loop brain control [5].

2.1.4. Experiment 4

In this experiment (see Figure 2(b)), NHP-B was trained to perform the classical centre-out task [12,38,39] through joystick control with targets appearing in one of the 8 different locations. A cursor appeared at the centre of the screen at the beginning of every trial and a trial was considered successful if the NHP managed to manipulate the cursor to reach and stay in the target area for 2 seconds under a total elapsed time of 10 seconds. The cursor position was updated every 100 *ms* via joystick control.

2.2. Datasets and train-test split

We have performed offline analysis on four datasets acquired from two NHPs. In experiments 1 and 2, we have used a total of 8 days of recorded data with 3 sessions on each day. In experiments 3 and 4, we have used data recorded on 4 separate days with 3 sessions on each day. Test data is consistent across all algorithms and corresponds to the sessions 2 and 3 on recorded days, across all experiments. Training data corresponds to session 1 of each day for supervised daily *retrained* models, and session 1 of day 1 for supervised *fixed* models. RL algorithms do not employ explicit training data (see Figure 1(d)). They are randomly initialized at the beginning of a recorded day and are updated in a purely online manner on *test data*.

3. Preliminaries

In this text, k will be used to denote a scalar, and $[k]$ will be used to denote the set of integers $\{1,2,\cdots,k\}$. For two (row or column) vectors $\mathbf{v},\mathbf{w}\in\mathbb{R}^n$; $\mathbf{v}\cdot\mathbf{w}$ will be used to denote their usual inner product, and $||\mathbf{v}||$ will be used to denote the l_2 norm of $\mathbf{v}$. For a matrix $\mathbf{W}\in\mathbb{R}^{k\times n}$, denote by $\mathbf{W}_1,\mathbf{W}_2,\cdots,\mathbf{W}_k$ its rows, which are (row) vectors in $\mathbb{R}^n$. Thus, for two matrices $\mathbf{V}$ and $\mathbf{W}$, $\mathbf{V}\cdot\mathbf{W}$ denotes their inner product. For a matrix $\mathbf{W}$, we denote by $||\mathbf{W}||$ the Frobenius norm [40] of $\mathbf{W}$, which is also the usual l_2 norm of the vector form of $\mathbf{W}$. For two matrices $\mathbf{W}$ and $\mathbf{V}$ denote by $\mathbf{W}\otimes\mathbf{V}$ their Kronecker product [41].

3.1. Problem statement formulation

Mathematically, a general reinforcement learning (RL) problem can be abstracted as a Partially Observable Markov Decision Process (POMDP). POMDP is a generalization of MDP. For this work, we adopt the standard MDP formalism [42]. An MDP is defined by a tuple $\langle S,A,R,T,\gamma\rangle$, which consists of a set of states S, a set of actions A, a reward function $R(s,a)$, a transition function $T(s,a,s')=P(s'|s,a)$, and a discount factor $\gamma\in[0,1]$. In each state $s\in S$, the agent takes an action $a\in A$. Upon taking this action, the agent receives a reward $R(s,a)$ and reaches a new state s', determined from the probability distribution $P(s'|s,a)$. A policy $\pi(a|s)$ specifies for each state which action the agent will take.

The problem RL seeks to solve differs from the standard MDP because the state space S, transition $T(s,a,s')$ and reward functions $R(s,a)$ are unknown to the agent. The goal of the agent in an RL formulation is thus to both build an internal representation of the world and select actions that maximizes cumulative future reward. To do this, the agent interacts with an environment through a sequence of observations, and takes actions, and is given a reward in return. In this work, we have assumed a simplified discrete scenario where $t\in 1,2,3,\cdots,T$, denotes the discrete time steps. We further define $\mathbf{x}_t=\begin{bmatrix}r_1(t) & r_2(t) & \cdots & r_{\mathscr{D}-1}(t) & r_{\mathscr{D}}(t)\end{bmatrix}$; $\mathbf{x}_t\in\mathbb{R}^{\mathscr{D}}$ as the neural activity (*observations*) recorded from $\mathscr{D}$ input electrodes at each discrete step t (determined in a sliding window fashion with a step-size of $T_s=100$ *ms*, for the entire duration of the trial). Here, 'r_n' ($n=1,2,3,\cdots,\mathscr{D}$) denotes the number of spikes occurring in a backward looking window of time, $T_w=500$ *ms*, at each of the input $\mathscr{D}$ electrodes. The stride is chosen to match the ground truth frequency in the dataset while the bin-width is based on prior work using these datsets [5,36]. On each round, the learner is given an instance vector $\mathbf{x}_t\in\mathbb{R}^{\mathscr{D}}$ and is required to predict a label (*action*) out of a set of $\mathscr{C}$ pre-defined labels, which we denote by $[\mathscr{C}]=\{1,2,3,\cdots,\mathscr{C}\}$. We denote the predicted label by $\hat{y}_t$, whereas, the correct label associated with $\mathbf{x}_t$ is $y_t\in[\mathscr{C}]$. In case of an POMDP, learners (*agent*) do not use correct label information while learning; instead, they employ a binary scalar feedback for each step t. The feedback (reward) signal $r_t(\cdot)$ is given as $r_t=+1(c=c');-1(c\neq c')$, where c and c' are predicted and true labels respectively at the previous step, $t-1$. Overall, RL's goal is to map the actions from states of the environment (by following a sequence of observations) to get maximum outcome.

4. Reinforcement learning algorithms

4.1. Banditron

We consider a bandit setting where the identity of the true label is never exposed to the learner, instead the learner learns through a feedback $\mathbf{1}[\hat{y}_t \neq y_t]$, where $\mathbf{1}[\pi]$ is 1 if predicate π holds and 0 otherwise. In simpler terms, the learner will know whether it has predicted an incorrect label or not, without actually knowing the identity of the true label. The prediction of an online algorithm at each round t is determined by a hypothesis $h_t : \mathbb{R}^{\mathscr{D}} \rightarrow [\mathscr{C}]$, where h_t is taken from a class of hypotheses $\mathscr{H}$. Banditron specifically focuses on the class of margin based linear hypotheses for its prediction mechanism [32]. Formally, this hypothesis class is parameterized by weight matrices $\mathbf{W} \in \mathbb{R}^{\mathscr{C} \times \mathscr{D}}$ with $||\mathbf{W}|| \leq \mathscr{L}$, for some specified constant $\mathscr{L}$.

A good starting point of establishing the Banditron algorithm (and the class of linear hypotheses) is to model the learner as a perceptron adapted for multiclass prediction [43]. We denote by $\mathbf{W}^t$ the weight matrix used by perceptron at round t. Denoting the set of such matrices by $\mathscr{M}$. The learning starts with initializing the weight matrix as $\mathbf{W^1} = \mathbf{0}$. Given a matrix $\mathbf{W} \in \mathscr{M}$ with the rows $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \cdots, \mathbf{W}_{\mathscr{C}}$, the prediction associated with $\mathbf{W}$ for $\mathbf{x}$ at round t is

$$\hat{y}_t = \arg\max_{c \in [\mathscr{C}]} (\mathbf{W_c^t x_t}) \tag{1}$$

where, at each round Banditron picks $\hat{y}_t$ to be the best label according to the current weight matrix $\mathbf{W}^t$ as per (1). While ideally we would like to minimize the $0-1$ loss suffered by the learner (here, Banditron), for computational reasons it is preferable to consider convex loss functions [32]. Hence, performance bounds are commonly evaluated using the multiclass *hinge-loss function*. In particular, the hinge-loss of $\mathbf{W}^t$ on $(\mathbf{x}_t, y_t)$ is defined as follows:

$$\ell(\mathbf{W}^t; (\mathbf{x}_t, y_t)) = \max_{c \in [\mathscr{C}]/y_t} [1 - (\mathbf{W^t x_t})_{y_t} + (\mathbf{W_c^t x_t})]_+ \tag{2}$$

where, $[\cdot]_+ = \max\{\cdot, 0\}$ is the hinge function. The *hinge-loss* will be zero only if $(\mathbf{W^t x_t})_{y_t} - (\mathbf{W^t x_t})_c \geq 1$ for all $c \neq y_t$. If, $\hat{y}_t \neq y_t$ then $\ell(\mathbf{W^t}; (\mathbf{x}_t, y_t)) \geq 1$. Thus, the hinge-loss is a convex upper bound on the zero-one loss function, $\ell(\mathbf{W}^t; (\mathbf{x}_t, y_t)) \geq 1[\hat{y}_t \neq y_t]$.

Roughly speaking, it is difficult to learn while exploiting $\mathbf{W}^t$, *i.e.*, by predicting $\hat{y}_t$ while staying blind to the true label information y_t. Hence on some rounds, we let Banditron explore (with probability $1-\varepsilon$) and uniformly predict a random label from $[\mathscr{C}]$ according to (3). We denote this predicted label by $\tilde{y}_t$, and define the probability as

$$\mathscr{P}(c) = (1-\varepsilon)\mathbf{1}[c = \hat{y}_t] + \frac{\varepsilon}{\mathscr{C}} \tag{3}$$

On rounds in which we explore (so $\tilde{y}_t \neq \hat{y}_t$), if we additionally receive a positive feedback ($\tilde{y}_t = y_t$), then we indirectly obtain the full information regarding the identity of y_t, and we can therefore update our weight matrix $\mathbf{W}^t$ using this positive instance. Mathematically, it can be written as

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \widetilde{\mathbf{U}^t} \tag{4}$$

where, $\widetilde{\mathbf{U}^t} \in \mathbb{R}^{\mathscr{C} \times \mathscr{D}}$ is the update matrix, and is defined to be a function of the randomized prediction $\tilde{y}_t$

$$\widetilde{\mathbf{U}^{t}_{c,j}} = \mathbf{x}_{t,j}\left(\frac{\mathbf{1}[y_t = \tilde{y}_t]\mathbf{1}[\tilde{y}_t = c]}{\mathscr{P}(c)} - \mathbf{1}[\hat{y}_t = c]\right) \tag{5}$$

where, $c, j \in [\mathscr{C}], [\mathscr{D}]$. We emphasize that $\widetilde{\mathbf{U}^t}$ accesses the correct label y_t only through the indicator $\mathbf{1}[y_t = \tilde{y}_t]$ and thus the true label is never exposed to the algorithm, as per the bandit setting. The parameter ε controls the *exploration-exploitation* trade-off. For our analysis, we particularly focused on $\varepsilon \in \{0, 0.5\}$.

4.2. Banditron-RP

Banditron is inspired by the simple linear multiclass perceptron [43] and hence it is limited in its capability, and learns only from linearly separable patterns [32]. However, superior performance has been reported by non-linear methods over linear ones in iBMIs [7,17]. Thus, we have introduced non-linearity in the form of obtaining feature vector $\mathbf{f}_t \in \mathbb{R}^{\mathscr{C}}$ as a non-linear random projection (RP) of $\mathbf{x}_t$. Mathematically, it is expressed as $\mathbf{f}_t = g(\mathbf{W_{rand}}\mathbf{x}_t)$. $\mathbf{W_{rand}} \in \mathbb{R}^{\mathscr{C}\times\mathscr{D}}$ is the random projection weight matrix, whose weights are chosen from a standard uniform distribution in the interval $(0, 1)$. $g(\cdot)$ corresponds to the activation function, which in this case is the ReLU function. Learning proceeds in discrete rounds t similar to Banditron where $\mathbf{f}_t$ serves as an input. We refer to this variant of Banditron as Banditron-RP.

Addition of a fully connected feature extraction layer to Banditron is bound to drastically increase the overall power dissipation. Thus, we propose implementing the first layer of fixed random weights, $\mathbf{W_{rand}}$, in the form of single transistor based current multipliers reported in [14]. The custom designed chip - Spike-input Extreme Learning Machine (SELMA) [14] has the potential of directly being used to implement Banditron-RP.

4.3. Attention Gated Reinforcement Learning (AGREL)

The algorithm was originally defined as a biologically plausible network, which can attain advantages over previous algorithms by defining new plasticity parameters. The two signals that jointly determine plasticity are: (1) the global error signal δ and (2) an attention signal that feeds back from the output layer to the previous layers [44]. The model is developed from a perspective of a three-layered network (refer Figure 3), where error back propagation is used for training.

On each trial (round, t), a new input pattern $\mathbf{p}$ (observation $\mathbf{x}_t, t = \{1, 2, 3, \cdots, T\}$) is presented to the network's input layer $\mathbf{X}$ with $\mathscr{D}$ units (corresponding input activities, $r_i^{\mathbf{p}}, i = 1, 2, 3, \cdots, \mathscr{D}$) and the activity is propagated to the hidden layer $\mathbf{Y}$ with $\mathscr{M}$ hidden units through connection weights $\mathbf{V}_{ij}$, where $(j = 1, 2, 3, \cdots, \mathscr{M})$. The hidden layer $\mathbf{Y}$ uses sigmoidal non-linear functions. Hence, the output of the j^{th} node of the hidden layer is given as

$$\mathscr{Y}_j^{\mathbf{p}} = \frac{1}{1 + \exp\left(-\sum_{i=0}^{\mathscr{D}-1}\mathbf{V}_{ij}r_i^{\mathbf{p}}\right)} \tag{6}$$

where, $r_i^{\mathbf{p}}$ is the activity at the i_{th} node in the input layer (corresponding to the i_{th} electrode). The bias of the input and the hidden layer is defined as: $r_o, \mathscr{Y}_o = 1$. the number of output action classes (in the context, varies according to experiments) determines the number of output nodes. Each output unit $\mathscr{Z}_k, (k = 1, 2, 3, \cdots, \mathscr{C})$ points to a different action. A crucial feature of AGREL is that units in the output layer engage in a competition. On every trial (round, t), one output unit wins and gets activity 1, while the

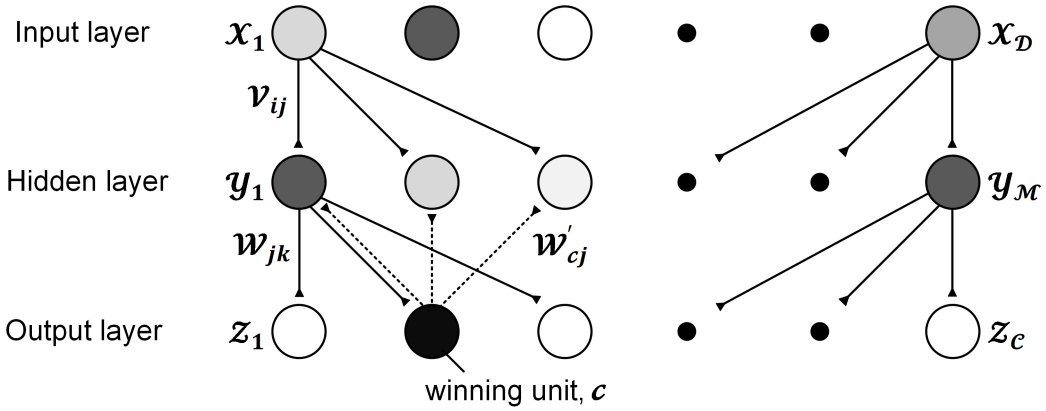

Figure 3. Three-layer Neural Network that is trained to perform a classification task. There are $\mathscr{D}$ units in the input layer, $\mathscr{M}$ in the second (hidden) layer, and $\mathscr{C}$ (depending on the number of actions available) in the output layer. Connections, $\mathbf{V}_{ij}$ propagate activity from the input layer to the hidden layer, and connections $\mathbf{W}_{jk}$ propagate the activity from the hidden layer to the output layer. The winning unit, c, feeds its activity back to the hidden layer through connections $\mathbf{W}'_{cj}$, represented as the dashed lines.

other output units get activity 0. AGREL uses the stochastic *softmax* rule to determine the probability of choosing unit k as the winning unit,

$$\mathscr{P}(\mathscr{Z}_k^{\mathbf{p}}=1)=\frac{\exp\left(\sum_{j=0}^{\mathscr{M}-1}\mathbf{W}_{jk}\mathscr{Y}_j^{\mathbf{p}}\right)}{\sum_{k'=1}^{\mathscr{C}}\exp\left(\sum_{j=0}^{\mathscr{M}-1}\mathbf{W}_{jk'}\mathscr{Y}_j^{\mathbf{p}}\right)} \tag{7}$$

If the network chooses the correct output unit for a particular stimulus (input pattern $\mathbf{p}$), it receives a reward $r=1$. No reward is given in the case of misclassification. The weights between the hidden units and output units, are updated according to (8), which depends mostly on the factor δ.

$$\Delta\mathbf{W}_{jk}=\beta\mathscr{Y}_j^{\mathbf{p}}\mathscr{Z}_k^{\mathbf{p}}f(\delta) \tag{8}$$

where, β is the learning rate. (8) implies that only connections onto the winning unit (with activity $\mathscr{Z}_k^{\mathbf{p}}=1$) are changed, because the other output units have activity 0 after the competition. On rewarded trials (where the network chooses the correct class), δ equals the difference between the amount of reward that was obtained and the amount that was expected for a particular stimulus, $\delta=r-E^{\mathbf{p}}(r)$. Here, $E^{\mathbf{p}}(r)$ is the average amount of reward that is expected with stimulus p, and this equals $\mathscr{P}(\mathscr{Z}_{c_{\mathbf{p}}}^{\mathbf{p}}=1)$, the probability that the correct output unit is chosen. On unrewarded trials, however the plasticity does not depend on $E^{\mathbf{p}}(r)$, and hence δ is set to -1. As a whole, δ is defined as

$$\delta=\begin{cases}1-\mathscr{P}(\mathscr{Z}_{c_{\mathbf{p}}}^{p}=1)\\ -1\end{cases} \tag{9}$$

Unexpected rewards are especially valuable in learning. δ however influences plasticity through an expansive function $f(\delta)$, which helps to fasten the learning process. $f(\delta)$ is defined such as, it takes large values if δ is close to 1, *i.e.*, when actions are rewarded unexpectedly:

$$f(\delta)=\begin{cases}\delta/(1-\delta); & \delta\geq 0\\ \delta; & \delta=-1\end{cases} \tag{10}$$

The weights $\mathbf{V}_{ij}$ between the input layer and the hidden layer are also modified as per (11), which depends on $f(\delta)$. However, here a second factor, which equals the feedback from the output layer at unit $\mathscr{Y}_j$, also influences plasticity

$$\Delta \mathbf{V}_{ij} = \beta r_i^{\mathbf{p}} \mathscr{Y}_j^{\mathbf{p}} f(\delta)(1-\mathscr{Y}_j^{\mathbf{p}}) \sum_{k=1}^{\mathscr{C}} \mathscr{Z}_k^{\mathbf{p}} \mathbf{W}'_{kj} \tag{11}$$

After the competition, only the winning output unit has activity 1, and the other output units have activity 0. Therefore, (11) simplifies to

$$\Delta \mathbf{V}_{ij} = \beta r_i^{p} \mathscr{Y}_j^{\mathbf{p}} f(\delta)[\mathbf{W}'_{cj}(1-\mathscr{Y}_j^{\mathbf{p}})] \tag{12}$$

where, $\mathbf{W}'_{cj}$ stands for feedback of the winning unit c. This feedback signal gates the plasticity of connections $\mathbf{V}_{ij}$ from the input layer to hidden unit j. Feedback thereby assigns credit to hidden units that are responsible for the choice of action [44].

4.4. HRL (Hebbian Reinforcement Learning)

HRL also takes in the form of AGREL, and is represented by a three-layered network, as depicted in Figure 3. The learning of HRL also uses the concept of error-backpropagation, similar to AGREL [45,46]. The output of the j_{th} node of the hidden layer is a probability of firing computed using a hyperbolic tangent function. Mathematically, it is written as

$$\mathscr{Y}_j^{\mathbf{p}} = \mathbf{tanh}(\begin{bmatrix}\mathbf{p} & b\end{bmatrix} * \mathbf{V}_{ij}) \tag{13}$$

where, $b \in \mathbb{R}^1$ is the bias term. HRL differs from AGREL in selecting the output node. The output of the k^{th} node in the output layer $\mathbf{Z}$ for each possible $\mathscr{C}$ actions, is given as

$$\mathscr{Z}_k = \mathbf{tanh}(\begin{bmatrix}\mathscr{S}(\mathscr{Y}_j^{\mathbf{p}}) & b\end{bmatrix} * \mathbf{W}_{jk}) \tag{14}$$

where, $\mathscr{S}(\cdot)$ is a sign function applied to the hidden layer outputs (positive values gets $+1$, negative value becomes -1). The output node with the highest action value among a total of C nodes is chosen to be the final output in the HRL scheme. The reward signal r_t is $+1$ if the previous action selection is correct, and -1 otherwise. The weights are initialized randomly from a standard normal distribution [28] and are updated at every iteration with learning rates μ_o, μ_H, using the reward signal r_t (defined earlier). It can be written as

$$\Delta \mathbf{V}_{ij} = \mu_H * r_t * \left(\begin{bmatrix}\mathbf{p} & b\end{bmatrix}^T * \left(\mathscr{S}(\mathscr{Y}_j^{\mathbf{p}}) - \mathscr{Y}_j^{\mathbf{p}}\right)\right) + \mu_H * (1-r_t) * \left(\begin{bmatrix}\mathbf{p} & b\end{bmatrix}^T * \left(1 - \mathscr{S}(\mathscr{Y}_j^{\mathbf{p}}) - \mathscr{Y}_j^{\mathbf{p}}\right)\right) \tag{15}$$

$$\Delta \mathbf{W}_{jk} = \mu_o * r_t * \left(\begin{bmatrix}\mathscr{Y}_j^{\mathbf{p}} & b\end{bmatrix}^T * (\mathscr{S}(\mathscr{Z}_k) - \mathscr{Z}_k)\right) + \mu_H * (1-r_t) * \left(\begin{bmatrix}\mathscr{Y}_j^{\mathbf{p}} & b\end{bmatrix}^T * (1 - \mathscr{S}(\mathscr{Z}_k) - \mathscr{Z}_k)\right) \tag{16}$$

4.5. Deep Q-learning

A policy $\pi : S \to \mathscr{P}(A)$ for the MDP maps any state $s \in S$ to a probability distribution $\pi(\cdot|s)$ over A. For policy π, the corresponding value function $V_\pi(s) : S \to \mathbb{R}$ is defined as the cumulative discounted reward obtained by when the actions are executed according to π, mathematically expressed above. On expanding that expression we get

$$V_\pi(s) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_i | S_o = s, A_i \sim \pi(\cdot|S_i)\right] \tag{17}$$

where, the next step is defined as $S_{i+1} \sim \mathscr{P}(\cdot|S_i, A_i)$. Similarly, the action-value function $Q^\pi : S \times A \to \mathbb{R}$ is defined as

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_i | S_o = s, A_0 = a\right] \tag{18}$$

where, $A_i \sim \pi(\cdot|S_i)$ and $S_{i+1} \sim \mathscr{P}(\cdot|S_i, A_i)$. Here, the set of actions A is same as the number of labels $[\mathscr{C}]$. For any given action-value function $Q : S \times A \to \mathbb{R}$, define the one-step greedy policy π_Q as any policy that selects the action with the largest Q-value, that is, for any $s \in S$, $\pi_Q(\cdot|s)$ satisfies

$$\pi_Q(c|s) = 0 \quad \text{if} \quad Q(s,c) \neq \max_{c' \in \mathscr{C}} Q(s,c') \tag{19}$$

Moreover, we define operator P^π by

$$(P^\pi Q)(s,c) = \mathbb{E}[Q(s',c')|s' \sim \mathscr{P}(\cdot|s,c)] \tag{20}$$

and define the Bellman operator T^π by $(T^\pi Q)(s,c) = r(s,c) + \gamma \cdot (P^\pi Q)(s,c)$, where $r(s,c)$ is the expected reward obtained at state s when taking action c. Then it can be verified that Q^π is the unique fixed point of T^π. The goal of reinforcement learning is to find the optimal policy, which achieves the largest cumulative reward. To characterize optimality, we define optimal action-value function Q^* as

$$Q^*(s',c') = \sup_{\pi \sim \mathscr{E}} Q^\pi(s,c) \tag{21}$$

where, the supremum is taken over all policies, and $\mathscr{E}$ is the environment dynamics. Based on Q^*, we define the optimal policy π^* as any policy that is greedy with respect to Q^*. it can be shown that $Q^* = Q^{\pi *}$. Finally, we define the *Bellman optimality* operator T via

$$(TQ^*)(s,c) = \mathop{\mathbb{E}}_{s' \sim \mathscr{E}}\left[r(s,c) + \gamma \max_{c' \in \mathscr{C}} Q(s',c') | s, c\right] \tag{22}$$

Here, $s' \sim \mathscr{P}(\cdot|s,c)$, which leads us to the Bellman optimality equation $TQ^* = Q^*$. In Deep Q-Learning algorithm, a deep neural network $Q_\theta : S \times A \to \mathbb{R}$ is used to approximate Q^*, where θ is the parameter. Authors' in [47] shows that two major steps are pivotal for the success of Deep Q-Learning (DQL).

First, DQL use the trick of experience replay [48]. Specifically, in each discrete round $(t = i)$, we store the transition $\langle S_i, A_i, R_i, S_{i+1}\rangle$ into the replay memory $\mathscr{M}$, and then sample a minibatch of independent samples from $\mathscr{M}$ to train the neural network via stochastic gradient descent. Since the trajectory of MDP has strong temporal correlation, the goal of experience replay is to obtain uncorrelated samples, which yields accurate gradient estimation for the stochastic optimization problem.

The second procedure is to use a target network Q_{θ^*} with parameter θ^*. Specifically, with independent samples $\langle s_i, c_i, r_i, s_i' \rangle_{i\in[t]}$ from replay memory, to update the parameter θ of the Q-network, we compute the target $y_i = r_i + \gamma \cdot \max_{c\in\mathscr{C}} Q_{\theta^*}(s_i', c)$, and update θ by the gradient of

$$\ell(\theta) = \mathop{\mathbb{E}}_{s,c\sim\rho(\cdot)} \left[\{y_i - Q_\theta(s,c;\theta_i)\}^2\right] \tag{23}$$

where, $\rho(s,c)$ is the exploration policy. The optimizers of the Q-network loss function can be computed using the gradient policy

$$Q(s,c;\theta) \leftarrow Q(s,c;\theta) + \alpha \nabla_\theta Q(s,c;\theta) \tag{24}$$

where, α is the learning rate. The parameter θ^* is updated once every t_{target} steps by letting $\theta^* = \theta$, and thus it is updated in a slower pace. For computational expedience, the parameters are updated after every time step; *i.e.*, with every new experience (input neural activity). We train on single sample updates, and hence this results in the following Q-learning update:

$$Q(s,c) \leftarrow Q(s,c) + \alpha(r + \gamma \max_{c'\in\mathscr{C}} Q(s',c') - Q(s,c)) \tag{25}$$

The procedure is an off-policy training method [49] that learns the policy based on $\hat{y}_i = \arg\max_{c\in[\mathscr{C}]}$ while using an exploration policy over $S \times A$, selected by an ε-greedy strategy. When the replay memory is large, experience replay is close to sampling independent transitions from an explorative policy. This reduces the variance of the $\nabla\ell(\theta)$, which is used to update θ. Thus, experience replay stabilizes the training of DQL, which benefits the algorithm in terms of computation.

To further understand how DQL works, let us replace the experience replay by sampling independent transitions from a fixed distribution $\sigma \in \mathscr{P}(S \times A)$. Under this setting, we have $\mathbb{E}(y_i|S_i,A_i) = (TQ_{\theta^*})(S_i,A_i)$, where Q_{θ^*} is the target network, which as we show as follows, is motivated from a statistical consideration. Let us first neglect the target network and set $\theta^* = \theta$. Using bias-variance decomposition, the expected value of $\ell(\theta)$ in (23) is

$$\mathbb{E}[\ell(\theta)] = ||Q_\theta - TQ_\theta||_\sigma^2 + \mathbb{E}\left\{[y_i - (TQ_\theta)(s_i,c)]^2\right\} \tag{26}$$

Here the first term is known as the mean-squared Bellman error (MSBE), and the second term is the variance of y_i, whereas $\ell(\theta)$ can be viewed as the empirical version of the MBSE, which has the bias $\mathbb{E}\left\{[y_i - (TQ_\theta)(s_i,c)]^2\right\}$ that also depends on θ. Thus, without the target network, minimizing $\ell(\theta)$ can be drastically different from minimizing the MBSE. Thus minimizing $\ell(\theta)$ is close to solving

$$\mathop{\text{minimize}}_{\theta\in\Theta} ||Q_\theta - TQ_{\theta^*}||_\sigma^2 \tag{27}$$

where, Θ is the parameter space. Note that in DQL we hold θ^* still and update θ for t_{target} steps. When t_{target} is sufficiently large and we neglect the fact that the objective in (27) is non-convex, we would update θ by the minimizer of (27) for fixed θ^*. Therefore, in the ideal case, DQL aims to solve the minimization problem (27) with θ^* fixed, and then update θ^* by the minimizer θ. Interestingly, this view of DQL offers a statistical interpretation of the target network. We have used a two hidden layer neural network implementation for Q_θ and Q_{θ^*} in the this study.

4.6. Hyperparameter optimization

We define the RL algorithm $\mathscr{A}$ having some hyperparameters $\lambda_1, \lambda_2, \lambda_3, \cdots, \lambda_n$ with respective domains $\Lambda_1, \Lambda_2, \cdots, \Lambda_n$ over some hyperparameter space $\Lambda = \Lambda_1, \times \cdots \times \Lambda_n$. For each hyperparameter setting $\lambda \in \Lambda$, we use $\mathscr{A}_\lambda$ to denote the learning algorithm $\mathscr{A}$ using this setting. We further use $\mathscr{L}(\mathscr{A}_\lambda, \mathscr{D}_{train}, \mathscr{D}_{valid})$ to denote the validation loss (e.g., misclassification rate) that $\mathscr{A}_\lambda$ achieves on data $\mathscr{D}_{valid}$ when trained on $\mathscr{D}_{train}$. The hyperparameter optimization problem then reduces to minimizing the blackbox function

$$f(\boldsymbol{\lambda}) = \frac{1}{k}\sum_{i=1}^{k} \mathscr{L}\left(\mathscr{A}_\lambda, \mathscr{D}_{train}^{(i)}, \mathscr{D}_{valid}^{(i)}\right) \tag{28}$$

Hyperparameters can be continuous, integer valued, or categorical. Foll [50], we can say, a hyperparameter λ_i is *conditional* on another hyperparameter λ_j if λ_i is only active if hyperparameter λ_j takes values from a given set $V_i(j) \subsetneq \Lambda_j$. These hyperparameters are optimized over a Bayesian averaging framework [50,51].

Bayesian Optimization [52] constructs a probabilistic model $\mathscr{M}$ of f based on point evaluations of f and any available prior information, and uses that model to select subsequent configurations $\boldsymbol{\lambda}$ to evaluate. In order to select its next hyperparameter configuration $\boldsymbol{\lambda}$ using model $\mathscr{M}$, Bayesian optimization uses an *acquisition function* $a_{\mathscr{M}} : \boldsymbol{\Lambda} \to \mathbb{R}$, which uses the predictive distribution of model $\mathscr{M}$ to quantify how useful knowledge about hyperparameter configurations $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ would be. This function is then maximized over $\boldsymbol{\Lambda}$ to select the most useful configuration $\boldsymbol{\lambda}$ to evaluate next. The most popular acquisition function is the expected improvement [52] over the best previously observed function value f_{min} attainable at a hyperparameter configuration $\boldsymbol{\lambda}$ (where expectations are taken over predictions with the current model $\mathscr{M}$):

$$\mathbb{E}_{\mathscr{M}}[I(\boldsymbol{\lambda})] = \int_{-\infty}^{f_{min}} \max\{f_{min} - f, 0\} \cdot p_{\mathscr{M}}(f|\boldsymbol{\lambda}) df \tag{29}$$

5. Results

5.1. Summary of models and hyperparameter tuning

Table 2 provides a summary of algorithms and their respective training paradigms analysed in this paper, whereas Table 3 captures the range of values and choices considered for arriving at optimal hyperparameters for different algorithms. For SVM, we have used bayesian hyperparameter optimization method *bayesopt* provided in Matlab *R*19*a* (MathWorks, Inc., Natick, Massachusetts, United States) to tune hyperparameters over 30 iterations in a five-fold cross-validation manner based on training set. LDA does not have any associated hyperparameters. Hyperparameters corresponding to the different RL algorithms are tuned on first recorded day's session 1 across all datasets. The code used for analysis is available in the following repository: https://github.com/aayushmanghosh/RL-Algorithms-for-iBMI-Applications.git.

Table 2. Summary of training paradigms and algorithms.

Training Modalities	Supervised Learning	Reinforcement Learning
Batch training, weight frozen	LDA_{fixed}	-
	$\text{LDA}_{retrain}$	-
	SVM_{fixed}	-
	$\text{SVM}_{retrain}$	-
Only online, update	-	Banditron, AGREL
	-	HRL, Deep Q-Learning
	-	Banditron-RP
Batch training, online update (BTOU*)	-	$\mathscr{X}$
	-	$\mathscr{Y}$

* We explicitly refer to the RL methods with batch training followed by online updates with the subscript $BTOU$, whereas the RL algorithms with only online update have no subscript. Here, $\mathscr{X}$ stands for $\text{AGREL}_{BTOU\_epoch\_xx}$, and $\mathscr{Y}$ for $\text{AGREL}_{BTOU\_transfer\_epoch\_xx}$. xx in the subscript stands for number of training data replications (also known as epochs).

Table 3. Hyperparameter tuning—parameter choices.

Hyperparameter	Associated Algorithm	Swept Range of values
Kernel	SVM	Linear, Gaussian, Polynomial
Number of hidden units	AGREL, HRL, DQL, Banditron-RP	$75, 100, \cdots, 200$
Explore-exploit trade-off (ε)	Banditron, AGREL, DQL, Banditron-RP	$0.0001, 0.001, \cdots, 0.1$
Discount factor (γ)	Deep Q-Learning	$0.1, 0.2, \cdots, 0.9$
Learning rates ($\alpha, \beta, \mu_o, \mu_H$)	AGREL, DQL, HRL	$0.001, 0.01, 0.1$

5.2. Decoding results – ideal feedback

We have reported classification accuracy as a measure of performance to compare decoding prowess of different techniques. Experiments 1, 2 and 3 represent a 4 class classification problem—*right*, *forward*, *stop* and *left*. Experiment 4 on the other hand has 8 distinguishable classes corresponding to the 8 different directions shown in Figure 2(b).

Results in Figure 4 show Banditron's average improvements of $6.01\% \pm 2.54\%$ over AGREL, $14.5\% \pm 1.83\%$ over HRL, $4.83\% \pm 1.83\%$ over DQL in experiment 1; $20.98\% \pm 14.95\%$ (AGREL), $14.97\% \pm 1.95\%$ (HRL), $10.07\% \pm 11\%$ (DQL) in experiment 2; $6.7\% \pm 6.66\%$ (AGREL), $16.62\% \pm 8.23\%$ (HRL), $9.12\% \pm 9.76\%$ (DQL) in experiment 3; and $7.44\% \pm 2.72\%$ (AGREL), $25.04\% \pm 4.35\%$ (HRL), $16.85\% \pm 2.53\%$ (DQL) in experiment 4, respectively. Statistical comparison of Banditron against AGREL, HRL and Q-learning yield p-values of 0.0078, 0.0078, 0.0078 in experiment 1 and 0.0391, 0.0078, 0.0391 in experiment 2 respectively for a Wilcoxon signed-rank test. This shows that the improvements afforded by Banditron are statistically significant ($p < 0.05$) over AGREL, HRL and Q-learning. We have not reported p-values for experiments 3 and 4 as the number of data points are too

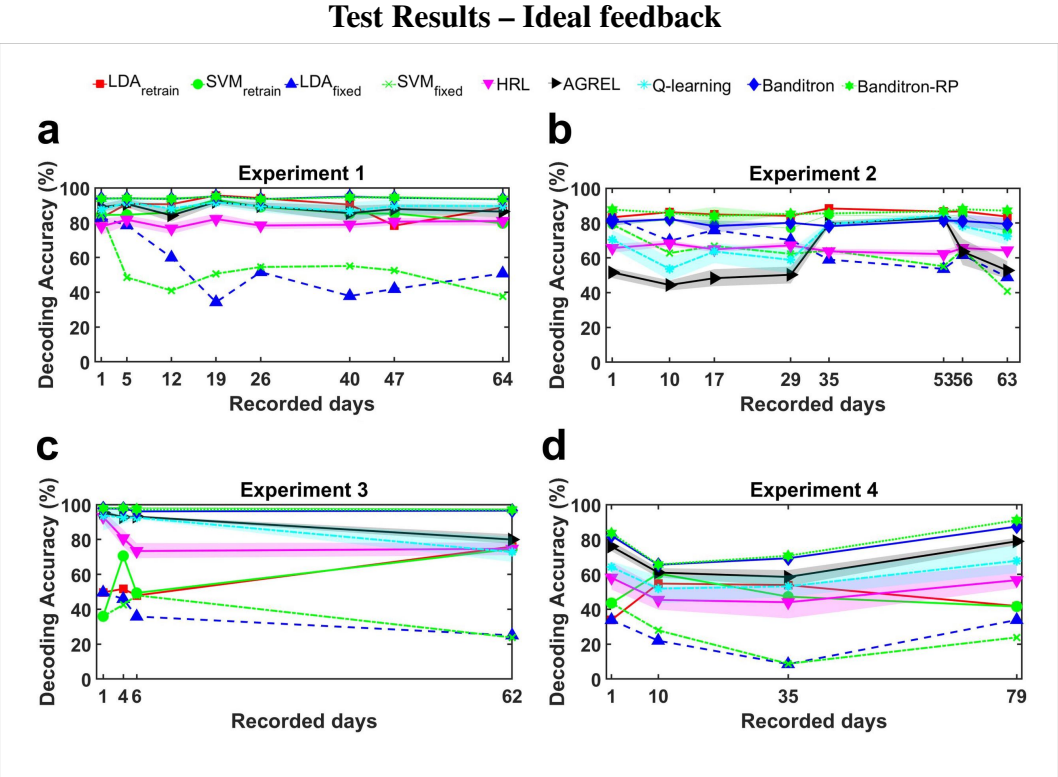

Figure 4. Decoding accuracy across four experiments has been reported for fixed and daily retrained supervised models (LDA, SVM) alongside RL algorithms (AGREL, HRL, Q-learning, Banditron and Banditron-RP). Shaded regions represent standard deviation of results across 20 iterations of random instantiations of probabilistic algorithms. Banditron and Banditron-RP significantly outperform the state of the art RL algorithms and *fixed* supervised models.

less [53]. Furthermore, non-linear version of Banditron, Banditron-RP yields average improvement of $\approx$ 6% and 2% over Banditron in experiments 2 and 4 while performing as good as Banditron in experiments 1 and 3 respectively.

Meanwhile, Banditron yields substantial average improvements over fixed models LDA_{fixed}, SVM_{fixed} of $39.4\% \pm 18.39\%$, $41.06\% \pm 14.09\%$ in experiment 1, $14.89\% \pm 11.66\%$, $18.19\% \pm 10.9\%$, in experiment 2, $61.02\% \pm 10.62\%$, $55.91\% \pm 20.31\%$, in experiment 3 and $51.51\% \pm 7.35\%$, $50.05\% \pm 13.86\%$ experiment 4 respectively. Changes in recording conditions resulting from electrode impedance changes, micro-motion of electrodes *etc.* lead to non-stationary distribution of input neural data across days [4,41]. This in turn leads to a drop in performance of fixed models – LDA_{fixed}, SVM_{fixed} due to the underlying shifts in data distribution [17,36].

In case of comparison to the daily retrained supervised models, Banditron yields superior performance with average improvements over $LDA_{retrain}$, $SVM_{retrain}$ of 5.21 ± 5.69, 8.19 ± 3.6 in experiment 1, 24.36 ± 7.17, 21.48 ± 4.89 in experiment 3 and 30 ± 19.65, 27.87 ± 18.16 in experiment 4 respectively. In experiment 2, Banditron performs almost as good as daily retrained classifiers with $\approx 94\%$ level of performance.

The above results are obtained for bin-width $T_w = 500$ ms following the choice of parameters in [5,36]. We have also obtained results by changing the bin-width to 250 and 750 ms. In general, the

results of all methods improve with increasing bin-width but the same trend is retrained – Banditron and Banditron-RP perform better than the other RL methods and is at par with the retrained methods.

Also, it can be seen that the performance of daily retrained models are worse in experiments 3 and 4 compared to experiments 1 and 2. We suspect two possible reasons for this related to the dataset for Experiments 3 and 4 – (a) it requires non-linear separation boundaries, (b) there is significant intra-day variability. To test out the hypothesis, we tested daily retrained versions of 2-layer ANNs (with 100 hidden layer neurons) which can create nonlinear separation boundaries. However, we found that the performance is still poor in Experiments 3 and 4. This led us to conclude that the drop in performance for experiments 3 and 4 is due to significant intra-day variability in these datasets.

5.3. Decoding results – erroneous and sparse feedback

In Figure 4, we have assumed the feedback at every time-step to be ideal corresponding to $+1$ for correct action and -1 otherwise along the lines of paradigm presented in [28,46,54,55]. In order to study the impact of quality of feedback signals, we have added additional controls of, (a) introducing error in feedback signals and (b) making feedback sparse.

Errors are introduced in the feedback signal across 10% and 20% of the total time-steps on each day across all datasets. The erroneous time-steps on each day are chosen via a Matlab *R*19*a* (The MathWorks, Inc., Natick, Massachusetts, United States) function *randi* which chooses the fraction of erroneous time-steps from total time-steps following a uniform discrete distribution. Introducing error involves changing the feedback signal at a given time-step with value $+1$ to -1 and *vice versa*. Furthermore, to study the impact of frequency of availability of feedback signal, we have reduced its occurrence to every second time-step (50% sparsity) and every fourth time-step (75% sparsity). Sparse feedback entails that the RL-decoder model is updated only when feedback is available. Figure 5 and *S*1 (in supplementary material) capture the results associated with erroneous and sparse feedback respectively.

When deliberately introducing error in feedback signal, Banditron ceases to yield the best performance among RL algorithms in all but one experiment, experiment 2. In experiment 1, Q-learning performs the best with $6.96\% \pm 3.93\%$, $12.97\% \pm 3.48\%$ average improvement over Banditron in 10%, 20% error cases. In experiments 3 and 4, AGREL performs the best with average improvements of $6.88\% \pm 10.67\%$, $9.4\% \pm 13.03\%$ over Banditron in 10%, 20% error cases respectively in experiment 3 and average improvements of $13.52\% \pm 7.48\%$, $19.55\% \pm 8.55\%$ in 10%, 20% error cases respectively in experiment 4. However, in experiment 2, Banditron still yields the best response over AGREL, HRL and Q-learning by $12.57\% \pm 15.69\%$, $12.12\% \pm 1.52\%$, $6.07\% \pm 12.13\%$ in 10% error case and $8.87\% \pm 14.5\%$, $11.72\% \pm 1.1\%$, $6.84\% \pm 11.62\%$ in 20% error case respectively. This goes to show that correct feedback is essential to Banditron yielding the best performance and can be considered as a limitation of this approach. Future work can try to combine aspects from other RL algorithms to improve this aspect. In case of introducing sparsity, Banditron still yields the overall best performance over other RL algorithms as seen in Figure *S*1 (in supplementary material). In case of 50% (75%) sparse feedback scenarios, Banditron achieves percentage improvements of at least $\approx$ 9% (11%), 11% (14%), 11% (13%), 6% (3%) in experiments 1, 2, 3 and 4 respectively over AGREL, HRL and Q-learning.

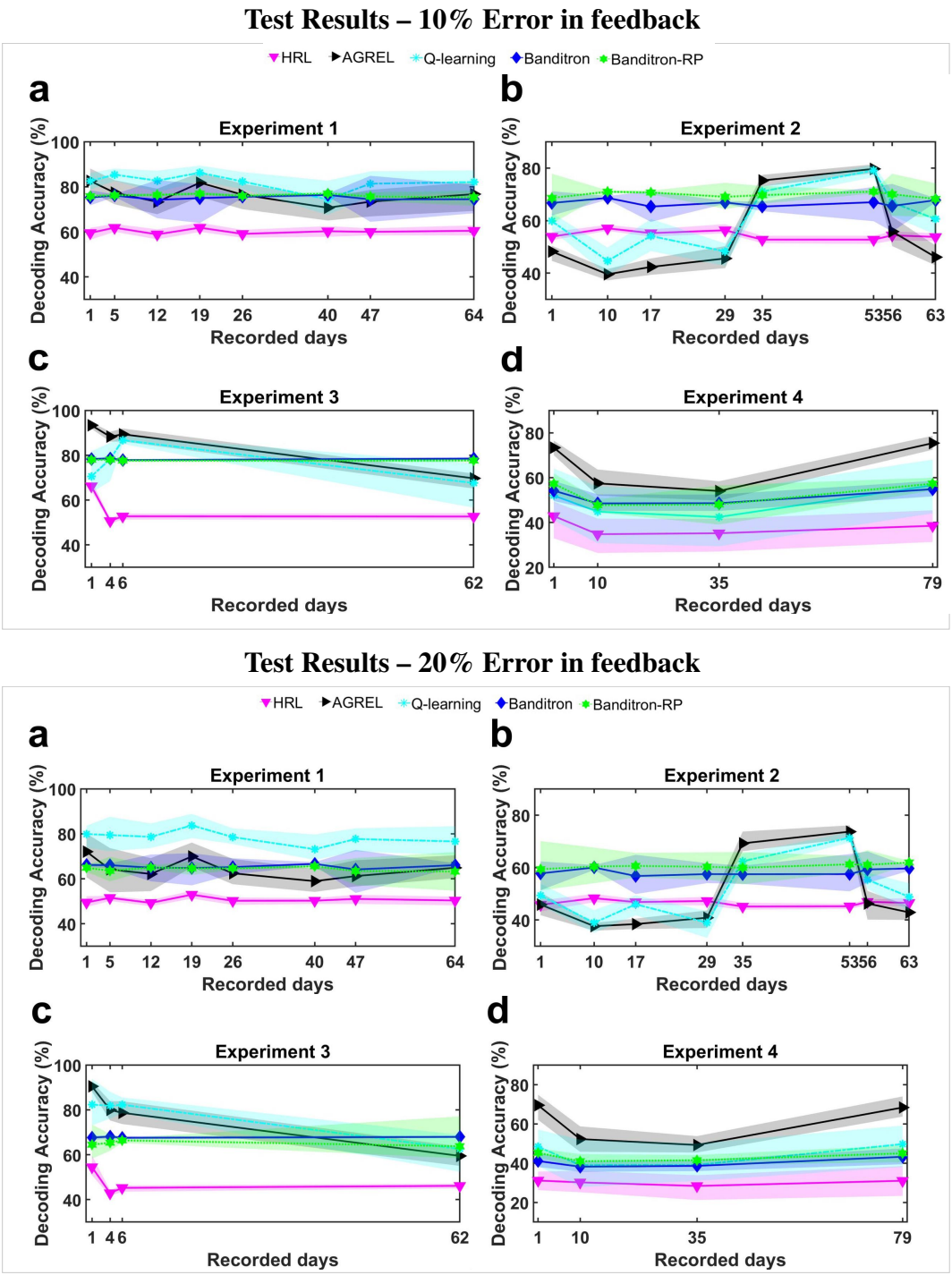

Figure 5. Decoding accuracy across four experiments has been reported for RL algorithms—AGREL, HRL, Banditron, Banditron-RP and Q-learning for 10% error in feedback in **(a)**, **(b)**, **(c)**, **(d)**—First half (top) and 20% error in feedback in **(a)**, **(b)**, **(c)**, **(d)**—Second half (bottom) respectively. Banditron and Banditron-RP are susceptible to erroneous feedback and cease to outperform state of the art RL algorithms in all but experiment 2. Shaded regions represent standard deviation of results across 20 iterations of random instantiation of weights of probabilistic algorithms. Q-learning performs the best in experiment 1 while AGREL performs the best in experiments 3 and 4.

5.4. Computational complexity

We consider an iBMI system with $\mathscr{D}$ inputs for a $\mathscr{C}$ option discrete control. The number of multiply-and-accumulate operations (MACs) required during prediction for single layer classifiers such as LDA, Banditron can be given as,

$$\text{MAC}_{predict\_linear} = \mathscr{D} \times \mathscr{C} \tag{30}$$

For single hidden layer neural network based approaches such as HRL, AGREL with $\mathscr{M}$ hidden nodes and a bias term in both input-hidden ($\mathbf{V}_{ij}$) and hidden-output weight ($\mathbf{W}_{jk}$) matrices, the number of MACs required during prediction is

$$\text{MAC}_{predict\_NN} = \mathscr{M}(\mathscr{D}+1) + \mathscr{C}(\mathscr{M}+1) \tag{31}$$

For Banditron-RP, we do not consider any bias term and accordingly, the number of MACs during prediction is

$$\text{MAC}_{predict\_Banditron-RP} = \mathscr{M} \times \mathscr{D} + \mathscr{D} \times \mathscr{C} \tag{32}$$

In Deep Q-learning we have used two-hidden layers with $\mathscr{M}_1$ and $\mathscr{M}_2$ nodes respectively and corresponding bias terms. Accordingly, the number of MACs required during prediction step is given as,

$$\text{MAC}_{predict\_Q} = \mathscr{M}_1(\mathscr{D}+1) + \mathscr{M}_2(\mathscr{M}_1+1) + \mathscr{C}(\mathscr{M}_2+1) \tag{33}$$

We have ignored the number of MACs required to implement the activation function for the sake of simplicity, since synaptic operations are known to dominate the computational complexity. RL algorithms are recursively updated and MACs expended by single-layer Banditron and multi-layer neural network systems—AGREL, HRL, Q-learning and Banditron-RP can be formulated as below,

$$\text{MAC}_{update\_AGREL} = 4 + \mathscr{M}(\mathscr{D}+6) \tag{34a}$$

$$\text{MAC}_{update\_HRL} = 5 + (\mathscr{M}+\mathscr{C}) + (\mathscr{M}+1)(\mathscr{D}+\mathscr{C}+2) \tag{34b}$$

$$\text{MAC}_{update\_Q} = \mathscr{M}_1(1+\mathscr{D}+\mathscr{M}_2) + 1 + \mathscr{M}_2(2+\mathscr{D}) \tag{34c}$$

$$\text{MAC}_{update\_Banditron-RP} = \mathscr{M} \tag{34d}$$

$$\text{MAC}_{update\_Banditron} = \mathscr{D} \tag{34e}$$

Do note that state of the art supervised schemes such as LDA, SVM are trained at the beginning of the day/session and are held fixed without being updated iteratively.

Consider a case of $\mathscr{D} = 64$ input electrode iBMI system driving a 4-option ($\mathscr{C} = 4$) discrete control at an operating frequency of 10 Hz, similar to the reported experiments in this paper. For neural networks based approaches HRL and AGREL, let us assume number of hidden nodes to be $\mathscr{M} = 80$.

With these assumptions, we can arrive at Table 4 comparing number of MACs memory requirement and estimate power dissipation across algorithms. Note that we have considered a linear version of SVM in Table 4. Furthermore, we consider weight resolution to be 16-bits across all algorithms. The average number of accumulation (AC) operations needed to sum spikes and create features are the same for all algorithms.

The values populated in Table 4 show that Banditron is at least an order of magnitude less computationally complex in terms of number of MACs and memory required than neural network based RL algorithms – AGREL, HRL and Q-learning. This low complexity feature fits well with the prospect of implementing low power iBMI decoders [7,9,20,56]. Furthermore, the number of MACs required by Banditron are in the same order of magnitude as LDA, SVM. The operations needed by Banditron-RP are slightly more than Banditron but the required number of updates are still orders of magnitude lesser than the other RL algorithms. Given that Banditron-RP performs better than Banditron in many cases, we feel it provides a good tradeoff.

Table 4. Computational complexity comparison[1].

Algorithms	MACs update	MACs prediction	Memory (kB)	AC
LDA, SVM	-	256	0.51	230
AGREL	5604	5524	11.1	230
HRL	5759	5524	11.1	230
DQL	16881	12004	23.3	230
Banditron	64	256	0.51	230
Banditron-RP	80	5440	10.8	230

* MAC = multiply and accumulate, Number of input channels, $\mathscr{D} = 64$, Number of hidden layer nodes, $\mathscr{M}, \mathscr{M}_1, \mathscr{M}_2 = 80$, Number of outputs, $\mathscr{C} = 4$, Weight resolution ($\mathbf{W}_{res}$) = 16 $bits$. Memory = (MACs prediction) $\times$ $\mathbf{W}_{res}$. AC = Accumulate spikes in data preprocessing.

6. Discussion

6.1. Comparison to state of the art RL algorithms

6.1.1. Effect of variability in neural data

As observed in Figure 4, the backpropagation (BP) based algorithms – AGREL, HRL, Q-learning perform poorly and erratically compared to Banditron, Banditron-RP. As an example of erratic performance, AGREL can be seen performing significantly better on days 35 and 53 in Figure 5(b) compared to the other days. We hypothesize that intra-day non-stationarity/variability is responsible for this phenomenon, wherein variability makes generalization difficult for BP-based algorithms operating in the purely online mode.

In order to validate our hypothesis, we first compute the mean value of minimum principal angles (MPAs) across trials in a sequential manner on all days of experiment 2. MPA is a parsimonious scalar metric used to evaluate the level of similarity between neural datasets [17]. Lower values of MPA correspond to higher levels of similarity and *vice versa*. Thus, the mean value of MPA across the sequence of trials serves as a metric of variability exhibited by the neural activity. In particular, we compute the MPA between the tuples: ($trial_1$, $trial_2$), followed by ($trial_2$, $trial_3$) until we reach the last trial tuple: ($trial_{last-1}$, $trial_{last}$). Thus, our variability metric, Var, for total trials, N_{trials}, can be given as,

$$Var = \frac{1}{N_{trials}-1} \sum_{i=1}^{N_{trials}-1} \text{MPA}(trial_i, trial_{i+1}) \tag{35}$$

Days 35 and 53 as seen in Figure 6 exhibit relatively lesser intra-day variability, thereby leading

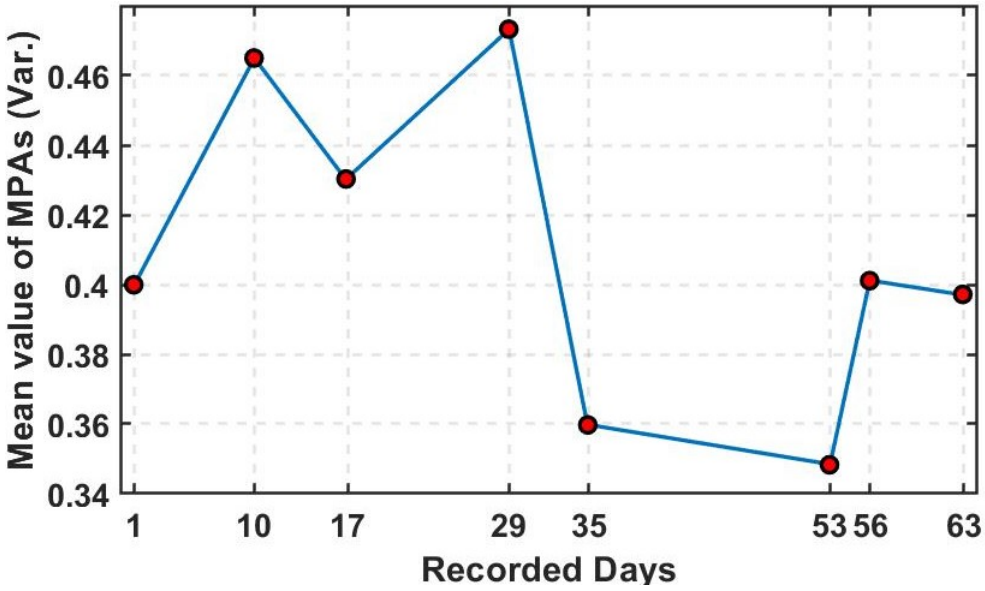

Figure 6. Shows intra-day variability (*Var.*) of firing rates of trials across days in experiment 2. Variability is captured in terms of the mean value of minimum principal angles (MPAs) computed between trials on each day. Days 35 and 53 have relatively low variability compared to the other days.

to improved performance for AGREL. In order to further validate the hypothesis about variability making generalization difficult for BP-based RL algorithms, we simulated two synthetic neural datasets, *Syn_Dir_4* and *Syn_Dir_8* along the lines of [57] using the Izhikevich model [58] for neural excitability. *Syn_Dir_4* and *Syn_Dir_8* correspond to the synthetic neural spike data generated for four-options (experiment 2) and eight-options (experiment 4) cursor control respectively. The detailed simulation methodology is reported in Section I of the supplementary material. We compared the decoding performance of RL algorithms against varying percentage of noisy neurons in Figure 7 for these datasets. All the algorithms were operated in the only online update mode. The inset figure in Figure 7 represents percentage of noise added along x-axis and variability referred to as Var. (mean value of minimum principal angles across trials sequence as shown in (35)) along the y-axis.

The general trend observed in Figure 7 is that the decoding performance of RL algorithms decreases with increase in variability. However, one must note that BP based RL algorithms such as HRL, AGREL and Q-learning suffer a relatively larger decline compared to Banditron, Banditron-RP. These results are qualitatively similar to the earlier presented results in Figure 4(b-top) and 6 wherein BP-based RL algorithms generalized relatively worse on days with more variability and vice-versa. Thus, the results in Figure 7 validate our previously stated hypothesis in a controlled simulation setting. We observe that non-stationarity in neural data makes it harder for BP-based RL algorithms to generalize in an online update mode. However, simple linear Banditron and its nonlinear projection variant Banditron-RP perform better due to its fewer trainable parameters. This underlines the strength of the neuromorphic online learning approach.

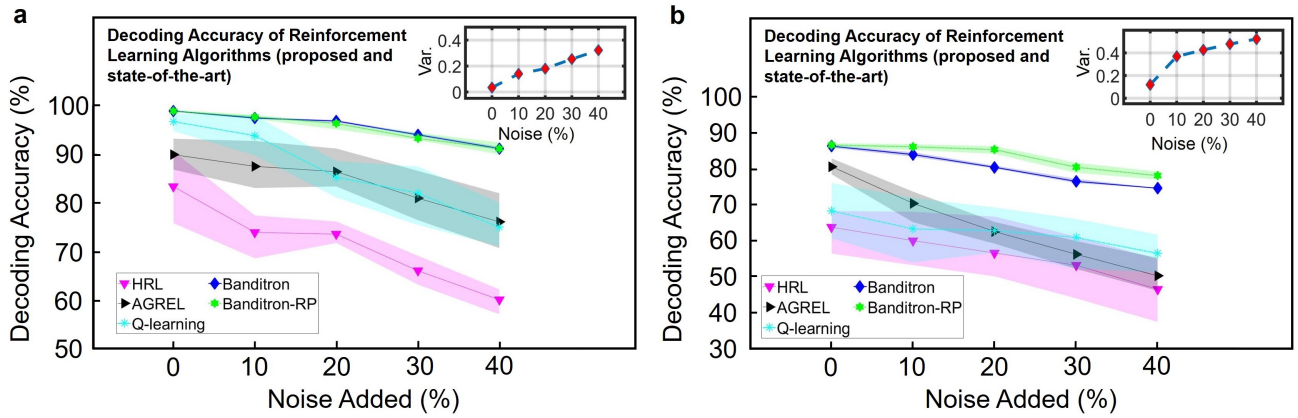

Figure 7. Shows decoder results for synthetic datasets **(a)** *Syn_Dir_4* and **(b)** *Syn_Dir_8* respectively across RL algorithms operated in the only online update mode. The figure in inset represents the Var. (35) on y-axis plotted against percentage added noise on x-axis. Variability increases as we add noise. Decoder accuracy is seen to be decreasing with increase in added noise, and at faster rate for BP-based RL algorithms.

6.1.2. Batch training mode

Neural Networks (NNs) are known to for their ability to approximate arbitrary functions following the universal approximation theorem. However, one must note that NN-based approaches trained by BP typically generalize well when trained over multiple replications of training data (also known as training epochs) [25,59]. Keeping this in mind, we consider an alternative more conventional batch-based setting involving a dedicated set of training data for a BP based algorithm. To carry out this analysis, we consider AGREL as the BP based algorithm and use experiment 2 dataset. We name the NN model as $AGREL_{BTOU\_epochs\_xx}$, where *BTOU* stands for batch training, online update, and *xx* stands for the number of training epochs. We train $AGREL_{BTOU\_epochs\_xx}$ on experiment 2's—session 1 of each day (similar to daily retrained supervised models) over 5, 10 and 20 training epochs (replications of training data).

Thereafter, we deploy it as a decoder on sessions 2 and 3 on each day. We let it update sequentially in an online fashion as the binary feedback is received. In this case, we observe $AGREL_{BTOU\_epochs\_xx}$ comfortably outperforming the online version of AGREL. Thus, having a dedicated training session comes with the prospect of improved decoding performance. However, this comes with the cost of significant amount of calibration time [10]. As mentioned earlier, the quantum of calibration time amounts to approximately one-third of the total experiment time in our reported experiments.

In Figure 8, we can see a saturation in performance of $AGREL_{BTOU\_epochs\_xx}$ after 10 epochs. Similar convergence can also be seen in supplementary Figure *S*2, where we show a plot of training (session 1) and validation accuracy (sessions 2 and 3) across number of epochs for day 1 of experiment 2. This shows that no further improvement is expected by increasing the number of epochs and we have trained the network optimally.

As observed in Figure 8, having a dedicated training session, leads to $AGREL_{BTOU\_epochs\_xx}$ performance to reach approximately the level of supervised learning algorithm ($LDA_{retrain}$), as we increase the epochs. This is in consonance with the universal approximation view, and we expect to

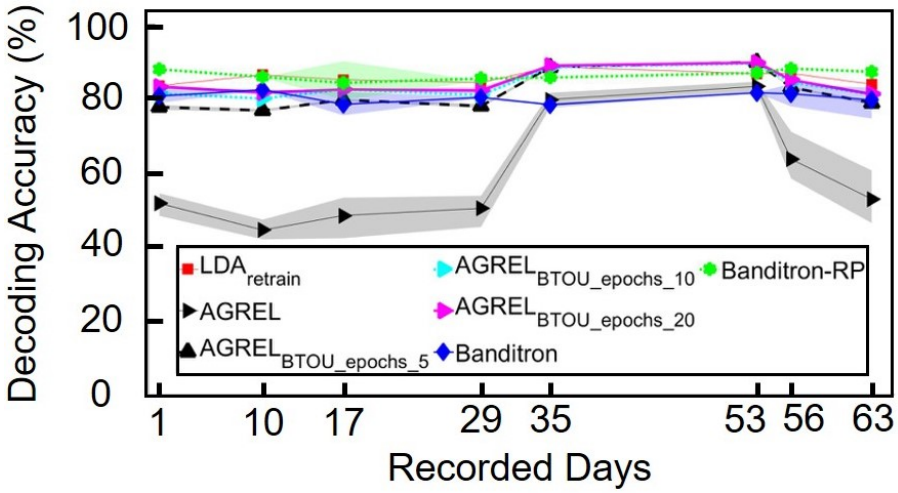

Figure 8. A more conventional batch-based train-test setup showing performance of an BP based RL algorithm, AGREL, when trained on a dedicated training set over 5, 10 and 20 training data replications (epochs) on experiment 2 dataset. The version employing a dedicated training set is referred to as $AGREL_{BTOU\_epochs\_xx}$, where xx stands for the number of epochs. $AGREL_{BTOU\_epochs\_xx}$ comfortably outperforms online AGREL while closely matching $LDA_{retrain}$'s results as we increase the number of epochs to 10 or higher.

achieve qualitatively similar results for other NN based approaches in other experiments as well. However, do note that the 2 layer version of Banditron with randomized neurons in the first layer (referred to as Banditron-RP) also achieves similar performance as $AGREL_{BTOU\_epochs\_10}$ while retaining the fully online version of training. Hence, this seems like a good choice to balance network learning capacity with online learning for iBMI applications.

A point to note here is that $AGREL_{BTOU\_epochs\_10}$ also outperforms online version of Banditron. This shows that in the event of a dedicated calibration procedure NN-based models such as AGREL outperform simple online linear model—Banditron. However, this superior performance comes at the cost of calibration times and computational complexity. If one were to implement training on chip, Banditron provides savings of at least 6 orders of magnitude in MACs and memory requirement compared to NN models. In addition to the assumptions listed in Table 4, we have assumed calibration over 10 epochs over recording time of 5 minutes with a time-step $T_s = 100ms$ and input firing rate resolution $FR_{res} = 8 - bits$.

6.1.3. Transfer learning

A common solution to the paucity of data is to train a NN on a related dataset and thereafter update only the last few layers following the paradigm of transfer learning popularly employed in both computer vision [60] and in biomedical applications [61]. The hope is that the model learns complex feature representations that can be easily deployed on similar tasks.

Following this approach, we performed a preliminary experiment on experiment 2 dataset with a single hidden layer $AGREL_{BTOU\_transfer\_epochs\_10}$ model trained on session 1 of each day over 10 epochs. The number of training data epochs was set based on the results observed in Figure *S*2 in supplementary material, wherein saturation in performance was observed at 10 epochs. Thereafter, we fixed the first layer

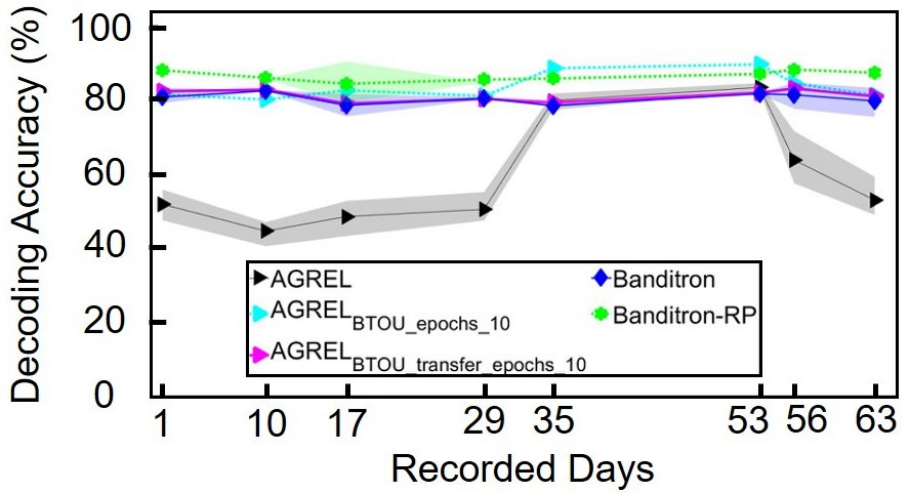

Figure 9. Applying transfer learning, while using an AGREL-trained NN as a feature extractor on experiment 2 dataset. We train the model $AGREL_{BTOU\_transfer\_epochs\_10}$ on 10 epochs using session 1 as training data. Subsequently, we fix the first layer weights and update the last layer weights using Banditron weight update rule. $AGREL_{BTOU\_transfer\_epochs\_10}$ does not yield improvements over simple Banditron.

weights and updated only the last layer weights for incoming samples from test sessions 2 and 3. We used the Banditron single layer rule to update the output layer weights. We observed however (Figure 9) that the learned features do not yield any improvement in decoding performance over random features used in Banditron-RP again pointing to variability in input data statistics.

In order to investigate the lack of improvement in performance, we visualize a plot of low dimensional projection of input firing rates and extracted features for training (session 1) and test data (sessions 2 and 3) across two days for experiment 2 in Figure *S*3 (supplementary material). The extracted feature embedding corresponds to the hidden layer of the aforementioned model – $AGREL_{BTOU\_transfer\_epochs\_10}$. Please note that for the purpose of ease of visualization, we showed only three clusters instead of the original four employed in the experiment. Visual inspection shows that the separability looks better for the extracted features on training set as seen in Figure S3a,b in supplementary material. However, separability looks somewhat similar on test data as seen in Figure S3c,d, in supplementary material. This, explains why the performance of $AGREL_{BTOU\_transfer\_epochs\_10}$ acting on extracted features is roughly equivalent to Banditron acting on input features.

6.2. Comparison to state of the art supervised learning paradigm

Compared to supervised retrained models, Banditron reports similar (experiment 2) or better (experiments 1, 3 and 4) levels of performance as $LDA_{retrain}$, $SVM_{retrain}$ (refer Figure 4). Supervised learning corresponds to the state of the art decoding methods employed in iBMIs. Thus, we have employed popular techniques—Linear Discriminant Analysis and Support Vector Machines to establish a baseline for the RL based results. One must note two stark differences between supervised learning and RL algorithms (see Figure 1a,b): (1) Supervised algorithms employ an explicit training phase, whereas RL algorithms do not have one, and (2) Supervised algorithm based models employed in iBMIs typically remain fixed once trained, whereas RL algorithms are constantly updated as they interact with

the environment. Thus, this comparison should be seen as a way to establish context in relation to state of the art techniques and not as a one-to-one comparison.

6.3. Towards autonomous iBMIs

The online continuous adaptive nature of the reported neuromorphic paradigm of iBMI operation coupled with possible biological sources of reward signal lead us to envision an autonomous iBMI system wherein no explicit calibration trials are required and the system simply learns from the feedback signal. Thus, this paradigm has the potential to drastically reduce the involvement of the neuro-engineer in enabling day to day usage of iBMI by patients.

However, this requires the presence of a valid biological reward/error signal. In this study, we are using binary evaluative feedback signal at every time-step along the lines of works [28,54]. The reason we have opted for this approach is that recent works have shown this kind of signal to be present in regions of the brain such as nucleus accumbens [29], primary motor cortex [30,46,55,62–65]. Apart from these regions, other candidate sources of obtaining this signal include EEG [66], ECoG [67] among others. We certainly feel that more work is needed in this area to fetch a reliable and stable feedback signal.

An alternative to sourcing the reward signal from a biological source is to derive it from the effector's trajectory as the agent attempts to move it towards a target [27]. In this approach [27], action chosen by the agent is rewarded if it moves towards the target and punished otherwise. This approach, while not useful for day to day autonomous usage by a patient, will still be useful in conducting iBMI experiments where the ground truth target is known from experimental design. We have reported open-loop results and it will be interesting to compare supervised approaches such as ReFIT [12] against RL-based approaches [27] in a closed-loop brain control setting.

7. Conclusion, limitations and future work

We have presented decoding results for different RL algorithms in a binary evaluative feedback scenario. Neuromorphic online learning based bandit algorithms—Banditron and Banditron-RP, for training a single layer of weights, are seen to be the best performing candidates among compared algorithms in an ideal feedback setting. Our results that shallow neural networks work better are also compatible with recent results in closed loop decoding [68]. Furthermore, they are seen to be relatively more robust to sparse feedback (see Figure *S*1 in supplementary material), *i.e.* when a feedback signal is not available at all time steps as is likely going to be the case in real BMI scenarios. However, one must point out that Banditron based learning system's superior performance is subject to accuracy of feedback. As seen in Figure 5, it is observed that Banditron's performance is not the best compared to other algorithms when feedback signals are erroneous. This implies a better strategy would be to estimate confidence in feedback signal first before updating weights using Banditron—this is an aspect for future in-depth study.

Moreover, as far as the power dissipation is concerned, Banditron has the potential to offer at least two orders of magnitude in savings compared to state of the art RL algorithms that employ batch learning and weight update for multiple layers. This has the potential to achieve a fully implantable solution and prolong the battery life of the iBMI.

Results presented in Section 4 show that the nature of adaptive learning of this paradigm is naturally robust against non-stationarities. This serves as a strong demonstration of the possible benefits of neuromorphic online learning for biomedical applications which was expected to be a prime use case for this technology [20,21]. The current open-loop decoding results hope to serve as a stepping stone towards closed-loop studies employing RL algorithms. Through the reported results we urge the community to advance along online, neuromorphic, RL lines of research instead of simply focussing on supervised learning paradigm. Following prior works, we have employed a feedback signal at every time-step. However an interesting question we wish to ask in future studies is that, "Can the iBMI system still perform well if the system gets delayed reward only at the end of a trial (attempted movement)?".

Acknowledgments

The research was partially sponsored by the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 11200922).

Authors' contribution

Aayushman Ghosh (A.G.): Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Visualization, Writing - original draft, Writing - review & editing. Shoeb Shaikh (S.S.): Conceptualization, Visualization, Methodology, Software, Validation, Investigation, Writing - original draft, Writing - review & editing. Biyan Zhou (B.Z.): Software, Investigation, Writing - review & editing. Pao-Sheng Vincent Sun (V.S.): Validation, Writing - review & editing. Camilo Libedinsky (C.L.): Conceptualization, Investigation, Data Curation, Resources, Writing - review & editing. Rosa Q. So (R.S.): Conceptualization, Investigation, Data Curation, Resources, Writing - review & editing. Arindam Basu (A.B.): Supervision, Funding acquisition, Writing - review & editing.

Conflicts of interests

The authors declare no conflict of interest.

References

[1] BS Armour, EA Courtney-Long, MH Fox, H Fredine, A Cahill. Prevalence and causes of paralysis—United States, 2013. *Am. J. Public Health* 2016, 106(10):1855–1857.

[2] Pandarinath C, Nuyujukian P, Blabe CH, Sorice BL, Saab J, *et al.* High performance communication by people with paralysis using an intracortical brain-computer interface. *eLife* 2017, 6:e18554.

[3] Nuyujukian P, Sanabria JA, Saab J, Pandarinath C, Jarosiewicz B, *et al.* Cortical control of a tablet computer by people with paralysis. *PloS One* 2018, 13(11):e0204566.

[4] Simeral J, Kim SP, Black M, Donoghue J, Hochberg L. Neural control of cursor trajectory and click by a human with tetraplegia 1000 days after implant of an intracortical microelectrode array. *J. Neural Eng.* 2011, 8(2):025027.

[5] Libedinsky C, So R, Xu Z, Kyar TK, Ho D, *et al.* Independent mobility achieved through a wireless brain-machine interface. *PLoS One* 2016, 11(11):e0165773.

[6] Benabid AL, Costecalde T, Eliseyev A, Charvet G, Verney A, *et al.* An exoskeleton controlled by an epidural wireless brain–machine interface in a tetraplegic patient: a proof-of-concept demonstration. *Lancet Neurol.* 2019, 18(12):1112–1122.

[7] Shaikh S, So R, Sibindi T, Libedinsky C, Basu A. Towards intelligent intracortical BMI (i^2 BMI): low-power neuromorphic decoders that outperform Kalman filters. *IEEE Trans. Biomed. Circuits Syst.* 2019, 13(6):1615–1624.

[8] Basu A, Yi C, Enyi Y. Big data management in neural implants: the neuromorphic approach. In *Emerging Technology and Architecture for Big-data Analytics*, 1st ed., Berlin: Springer, 2017, pp. 293–311.

[9] Boi F, Moraitis T, De Feo V, Diotalevi F, Bartolozzi C, *et al.* A bidirectional brain-machine interface featuring a neuromorphic hardware decoder. *Front. Neurosci.* 2016, 10:563.

[10] Brandman DM, Hosman T, Saab J, Burkhart MC, Shanahan BE, *et al.* Rapid calibration of an intracortical brain–computer interface for people with tetraplegia. *J. Neural Eng.* 2018, 15(2):026007.

[11] Glaser JI, Benjamin AS, Chowdhury RH, Perich MG, Miller LE, *et al.* Machine learning for neural decoding. *eNeuro* 2020, 7(4).

[12] Gilja V, Nuyujukian P, Chestek CA, Cunningham JP, Yu BM, *et al.* A high-performance neural prosthesis enabled by control algorithm design. *Nat. Neurosci.* 2012, 15(12):1752–1757.

[13] Schwemmer MA, Skomrock ND, Sederberg PB, Ting JE, Sharma G, *et al.* Meeting brain–computer interface user performance expectations using a deep neural network decoding framework. *Nat. Med.* 2018, 24(11):1669–1676.

[14] Chen Y, Yao E, Basu A. A 128-channel extreme learning machine-based neural decoder for brain machine interfaces. *IEEE Trans. Biomed. Circuits Syst.* 2015, 10(3):679–692.

[15] An H, Nason-Tomaszewski SR, Lim J, Kwon K, Willsey MS, *et al.* A power-efficient brain-machine interface system with a sub-mW feature extraction and decoding ASIC demonstrated in nonhuman primates. *IEEE Trans. Biomed. Circuits Syst.* 2022, 16(3):395–408.

[16] Shaeri M, Shin U, Yadav A, Caramellino R, Rainer G, *et al.* A 2.46-mm^2 miniaturized brain–machine interface (MiBMI) enabling 31-class brain-to-text decoding. *IEEE J. Solid-State Circuits* 2024, 59(11):3566–3579.

[17] Sussillo D, Stavisky SD, Kao JC, Ryu SI, Shenoy KV. Making brain–machine interfaces robust to future neural variability. *Nat. Commun.* 2016, 7(1):13749.

[18] Ma X, Rizzoglio F, Bodkin KL, Perreault E, Miller LE, *et al.* Using adversarial networks to extend brain computer interface decoding accuracy over time. *eLife* 2023, 12:e84296.

[19] Bose SK, Acharya J, Basu A. Is my neural network neuromorphic? Taxonomy, recent trends and future directions in neuromorphic engineering. In *2019 53rd Asilomar conference on signals, systems, and computers*, Pacific Grove, USA, November 3–6, 2019, pp. 1522–1527.

[20] Basu A, Acharya J, Karnik T, Liu H, Li H, *et al.* Low-power, adaptive neuromorphic systems: recent progress and future directions. *IEEE J. Emerging Sel. Top. Circuits Syst.* 2018, 8(1):6–27.

[21] Azghadi MR, Lammie C, Eshraghian JK, Payvand M, Donati E, *et al.* Hardware implementation of deep network accelerators towards healthcare and biomedical applications. *IEEE Trans. Biomed.*

Circuits Syst. 2020, 14(6):1138–1159.

[22] Yik J, Van den Berghe K, den Blanken D, Bouhadjar Y, Fabre M, *et al*. The neurobench framework for benchmarking neuromorphic computing algorithms and systems. *Nat. Commun.* 2025, 16(1):1545.

[23] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* 2016, 529(7587):484–489.

[24] DiGiovanna J, Mahmoudi B, Fortes J, Principe JC, Sanchez JC. Coadaptive brain–machine interface via reinforcement learning. *IEEE Trans. Biomed. Eng.* 2008, 56(1):54–64.

[25] Wang Y, Wang F, Xu K, Zhang Q, Zhang S, *et al.* Neural control of a tracking task via attention-gated reinforcement learning for brain-machine interfaces. *IEEE Trans. Neural Syst. Rehabil. Eng.* 2014, 23(3):458–467.

[26] Wang F, Wang Y, Xu K, Li H, Liao Y, *et al.* Quantized attention-gated kernel reinforcement learning for brain–machine interface decoding. *IEEE Trans. Neural Netw. Learn. Syst.* 2015, 28(4):873–886.

[27] Zhang X, Libedinsky C, So R, Principe JC, Wang Y. Clustering neural patterns in kernel reinforcement learning assists fast brain control in brain-machine interfaces. *IEEE Trans. Neural Syst. Rehabil. Eng.* 2019, 27(9):1684–1694.

[28] Pohlmeyer EA, Mahmoudi B, Geng S, Prins NW, Sanchez JC. Using reinforcement learning to provide stable brain-machine interface control despite neural input reorganization. *PloS One* 2014, 9(1):e87253.

[29] Prins NW, Sanchez JC, Prasad A. Feedback for reinforcement learning based brain–machine interfaces using confidence metrics. *J. Neural Eng.* 2017, 14(3):036016.

[30] Benyamini M, Nason SR, Chestek CA, Zacksenhouse M. Neural Correlates of error processing during grasping with invasive brain-machine interfaces. In *9th International IEEE/EMBS Conference on Neural Engineering (NER)*, New York: IEEE, 2019. pp. 215–218.

[31] Zhu B, Farivar M, Shoaran M. Resot: Resource-efficient oblique trees for neural signal classification. *IEEE Trans. Biomed. Circuits Syst.* 2020, 14(4):692–704.

[32] Kakade SM, Shalev-Shwartz S, Tewari A. Efficient bandit algorithms for online multiclass prediction. In *Proceedings of the 25th international conference on Machine learning*. 2008 pp. 440–447.

[33] Eliasmith C, Anderson CH. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*, Cambridge: MIT press, 2003.

[34] Schrauwen B, Verstraeten D, Van Campenhout J. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks*. 2007, pp. 471–482.

[35] Collinger JL, Boninger ML, Bruns TM, Curley K, Wang W, *et al.* Functional priorities, assistive technology, and brain-computer interfaces after spinal cord injury. *J. Rehabil. Res. Dev.* 2013, 50(2):145.

[36] Shaikh S, So R, Sibindi T, Libedinsky C, Basu A. Sparse Ensemble Machine Learning to improve robustness of long-term decoding in iBMIs. *IEEE Trans. Neural Syst. Rehabil. Eng.* 2019, 28(2):380–389.

[37] Quiroga RQ, Nadasdy Z, Ben-Shaul Y. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Comput.* 2004, 16(8):1661–1687.

[38] Shah S, Haghi B, Kellis S, Bashford L, Kramer D, *et al.* Decoding kinematics from human parietal cortex using neural networks. In *9th International IEEE/EMBS Conference on Neural Engineering (NER)*, New Youk: IEEE, 2019. pp. 1138–1141.

[39] Allahgholizadeh Haghi B, Kellis S, Shah S, Ashok M, Bashford L, *et al.* Deep multi-state dynamic recurrent neural networks operating on wavelet based neural features for robust brain machine interfaces. *Adv. Neural Inf. Process.Syst.* 2019, 32.

[40] Guo PC. A Frobenius norm regularization method for convolutional kernels to avoid unstable gradient problem. *arXiv* 2019, arXiv:1907.11235.

[41] Horn RA, Johnson CR. *Matrix analysis*, Landon: Cambridge University Press, 2012.

[42] Sutton RS, Barto AG, *et al. Reinforcement Learning: An Introduction*, vol. 1, Cambridge: MIT Press, 1998.

[43] Crammer K, Singer Y. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.* 2003, 3(Jan):951–991.

[44] Roelfsema PR, Ooyen Av. Attention-gated reinforcement learning of internal representations for classification. *Neural Comput.* 2005, 17(10):2176–2214.

[45] Mahmoudi B, Sanchez JC. A symbiotic brain-machine interface through value-based decision making. *PloS One* 2011, 6(3):e14760.

[46] Tarigoppula A, Rotella N, Francis JT. Properties of a temporal difference reinforcement learning brain machine interface driven by a simulated motor cortex. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, New York: IEEE, 2012 pp. 3284–3287.

[47] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, *et al.* Human-level control through deep reinforcement learning. *Nature* 2015, 518(7540):529–533.

[48] Thrun S, Littman ML. Reinforcement learning: An introduction. *AI Magazine* 2000, 21(1):103–103.

[49] Precup D, Sutton RS, Dasgupta S. Off-policy temporal-difference learning with function approximation. In *ICML*. 2001 pp. 417–424.

[50] Bergstra J, Bardenet R, Bengio Y, Kégl B. Algorithms for hyper-parameter optimization. *Adv. Neural Info. Proc. Syst.* 2011, 24.

[51] Swersky K, Duvenaud D, Snoek J, Hutter F, Osborne MA. Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. 2014, arXiv:1409.4011.

[52] Brochu E, Cora VM, De Freitas N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning 2010, .

[53] Hogg RV, Tanis EA, Zimmerman DL. *Probability and Statistical Inference*, vol. 993, New York: Macmillan, 1977.

[54] Mahmoudi B, Pohlmeyer EA, Prins NW, Geng S, Sanchez JC. Towards autonomous neuroprosthetic control using Hebbian reinforcement learning. *J. Neural Eng.* 2013, 10(6):066005.

[55] Marsh BT, Tarigoppula VSA, Chen C, Francis JT. Toward an autonomous brain machine interface: integrating sensorimotor reward modulation and reinforcement learning. *J. Neurosci.* 2015,

35(19):7374–7387.

[56] Sarpeshkar R. Universal principles for ultra low power and energy efficient design. *IEEE Trans. Circuits Syst. II* 2012, 59(4):193–198.

[57] Prins NW, Sanchez JC, Prasad A. A confidence metric for using neurobiological feedback in actor-critic reinforcement learning based brain-machine interfaces. *Front. Neurosci.* 2014, 8:111.

[58] Izhikevich EM. Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 2003, 14(6):1569–1572.

[59] Bengio Y. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade: 2nd ed.*, Berlin: Springer, 2012, pp. 437–478.

[60] Zhuang F, Qi Z, Duan K, Xi D, Zhu Y, *et al.* A comprehensive survey on transfer learning. *Proc. IEEE* 2020, 109(1):43–76.

[61] Acharya J, Basu A. Deep neural network for respiratory sound classification in wearable devices enabled by patient specific model tuning. *IEEE Trans. Biomed. Circuits Syst.* 2020, 14(3):535–544.

[62] Roesch MR, Olson CR. Neuronal activity related to anticipated reward in frontal cortex: does it represent value or reflect motivation? *Ann. N.Y. Acad. Sci.* 2007, 1121(1):431–446.

[63] Ramakrishnan A, Byun YW, Rand K, Pedersen CE, Lebedev MA, *et al.* Cortical neurons multiplex reward-related signals along with sensory and motor information. *Proc. Natl. Acad. Sci. USA* 2017, 114(24):E4841–E4850.

[64] Ramkumar P, Dekleva B, Cooler S, Miller L, Kording K. Premotor and motor cortices encode reward. *PloS One* 2016, 11(8):e0160851.

[65] McNiel DB, Choi JS, Hessburg JP, Francis JT. Reward value is encoded in primary somatosensory cortex and can be decoded from neural activity during performance of a psychophysical task. In *38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Orlando, Florida, USA16-20 August 2016, pp. 3064–3067.

[66] Kreilinger A, Neuper C, Müller-Putz GR. Error potential detection during continuous movement of an artificial arm controlled by brain–computer interface. *Med. Biol. Eng. Comput.* 2012, 50:223–230.

[67] Milekovic T, Ball T, Schulze-Bonhage A, Aertsen A, Mehring C. Error-related electrocorticographic activity in humans during continuous movements. *J. Neural Eng.* 2012, 9(2):026007.

[68] Willsey MS, Nason-Tomaszewski SR, Ensel SR, Temmar H, Mender MJ, *et al.* Real-time brain-machine interface in non-human primates achieves high-velocity prosthetic finger movements using a shallow feedforward neural network decoder. *Nat. Commun.* 2022, 13(1):6899.