Incremental Learning in Multiple Limb Positions for Electromyography-Based Gesture Recognition using Hyperdimensional Computing

SUBMISSION DATE / POSTED DATE

20-09-2021 / 29-09-2021

CITATION

Zhou, Andy; Muller, Rikky; Rabaey, Jan (2021): Incremental Learning in Multiple Limb Positions for Electromyography-Based Gesture Recognition using Hyperdimensional Computing. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.16643257.v1

DOI

10.36227/techrxiv.16643257.v1

Incremental Learning in Multiple Limb Positions for Electromyography-Based Gesture Recognition using Hyperdimensional Computing

Andy Zhou, *Student Member, IEEE*, Rikky Muller *Senior Member, IEEE*, and Jan Rabaey, *Fellow, IEEE*

***Abstract*— Prosthetic control for rehabilitation, among many other applications, can leverage in-sensor hand gesture recognition in which lightweight machine learning models for classifying electromyogram (EMG) signals are embedded on miniature, low-power devices. While research efforts have demonstrated high accuracy in controlled settings, these methods have yet to make a significant commercial or clinical impact due to the wide variety of scenarios and situational contexts that are faced during everyday use. Typical static models suffer from the effects of EMG signal variation caused by changing contexts in which they are deployed. Here, we propose an incremental learning algorithm using hyperdimensional (HD) computing that can efficiently learn gesture patterns performed in new limb positions, a context-change which normally significantly degrades classification accuracy. A prototype-based learning algorithm, HD computing enables memory- and computation-efficient incorporation of new training examples into the model, while preserving information about already learned contexts. We present various configurations of the incremental HD classifier, allowing system designers to trade classification performance for implementation efficiency as measured through memory footprint. Incremental learning experiments with data from 5 subjects show that HD computing can achieve similar accuracies as incrementally trained SVM and LDA classifiers while requiring a fraction of the memory allocation.**

***Index Terms*— Hand gesture recognition, hyperdimensional computing, incremental learning**

I. Introduction

FOR users of upper-limb prostheses, hand gesture recognition has the potential to enable higher levels of dexterity over existing control strategies while remaining unobtrusive [1]. Advances in machine learning (ML) algorithms have enabled a larger number of gestures to be classified using electromyographic (EMG) signals from muscle activity, controlling more numerous degrees of freedom in an attempt to restore natural movement. Various types of algorithms have been employed, including support vector machines (SVM [2]), linear discriminant analysis (LDA [3]), and artificial neural networks [4]. These gesture recognition capabilities can be embedded on-board wearable/implantable devices and prostheses, eliminating the need for separate computation devices and inefficient wireless streaming of raw EMG data [5], [6].

Still, a lack of robustness due to variation in EMG signals remains one of the largest issues leading to abandonment of smart prosthetics [7]. Over time, deployment conditions become different than the ones in which a classifier is trained, leading to accuracy degradation [8]. One common problem is the variation of upper limb position [9], [10]. Users must be able to expect accurate classification with their limb in various postures, but this causes recording electrodes to shift and affects gesture performance as well as underlying muscle activity [11]. Experiments have shown that gesture classifiers trained in one position fail to perform in others with acceptable accuracy due to the shifts in data distribution [11]–[13].

To address this, training datasets are expanded to include EMG signals collected in a larger variety of limb positions [11], [12] along with other sensing modalities like accelerometers and gyroscopes [13]–[15]. These can be used to train larger, more complex models [11], [13], [16] or multiple position-specific models [11], [14], [15]. As more training data is required, it becomes burdensome to the user and impractical to collect it all at once for batch training. Instead, classifiers which can be incrementally trained over several sessions may be more suited for this application. When untrained limb position contexts are encountered smaller incremental training datasets may be collected to update the existing model.

Several works have demonstrated and advocated for adaptable ML models and incremental learning to address EMG variation. A general framework consists of selectively updating the training dataset with new examples as new information is made available (Figure 1a) [17]. Incremental training for gesture recognition with such a framework has been demonstrated using both SVM [18], [19] and LDA [10], [17], [20]–[22] classifiers. Much focus has been on how well models adapt to gradual changes in the EMG over prolonged use, but when the incrementally learned contexts consist of multiple limb positions that all continue to be relevant, more attention must be paid to how well models also retain past information. Incrementally adapting a model to a new position must enable

This work was supported by DARPA DSO Virtual Intelligence Processing (VIP) program under Grant No. HR00111990072. The authors thank Weill Neurohub and sponsors of Berkeley Wireless Research Center. Research in this work involved human subjects. All experiments were performed in strict compliance with IRB guidelines and were approved by the Committee for Protection of Human Subjects at University of California, Berkeley (Protocol title: "FlexEMG Continuous Learning Limb Position Study"; Protocol no.: 2020-12-13906). Informed consent was received from all subjects.

A. Zhou, R. Muller, and J. Rabaey are with the University of California, Berkeley, Berkeley, California, USA. R. Muller is also with the Chan-Zuckerberg Biohub, San Francisco, California, USA.

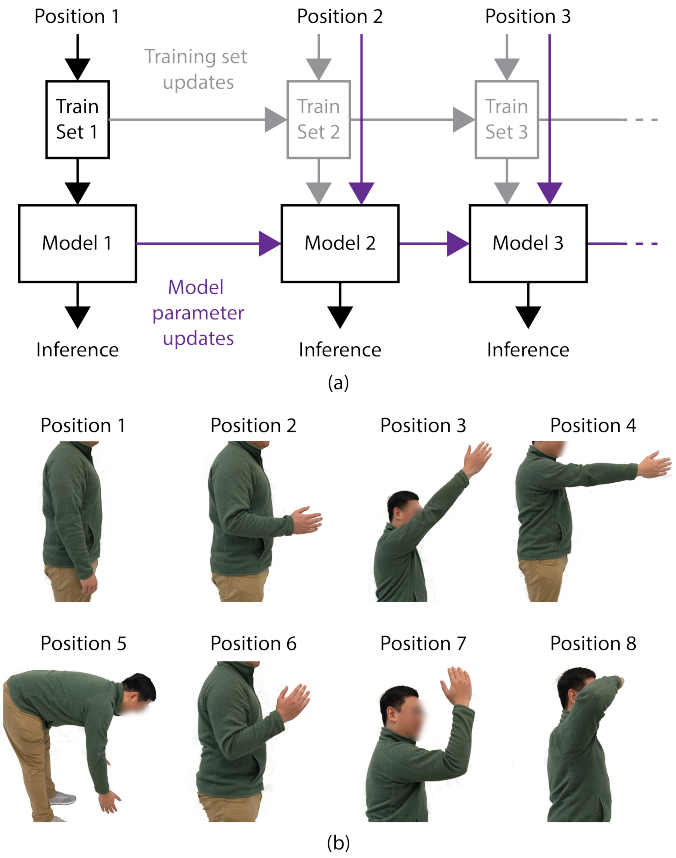

Fig. 1. Framework for incremental learning in multiple limb positions. (a) Diagram with different methods for updating model parameters at each step. An accumulated training set for updating models may be kept as new data is collected (gray path), or new data can be directly incorporated into the model by updating model parameters (purple path). (b) 8 limb positions considered in this study.

performance in the new context while also preserving accuracy when used in previous contexts.

Additionally, there has been a lack of emphasis on implementing incremental learning methods with computational resource constraints in mind. Because the framework grows a dataset that must be retained for subsequent training sessions, the memory footprint for storing training examples must be considered for efficient in-sensor implementations. While model parameters condense the information contained in training examples for use during inference, some portion of the original training dataset must usually be retained for training or updating the model (Figure 1a, gray). Online SVM training [23], [24] requires previous examples to potentially be promoted as support vectors, and models trained with stochastic gradient descent require that data from different contexts be shuffled. It is therefore beneficial to find a model whose parameters capture the necessary information for incremental training but is also able to incorporate new data regardless of the order in which it arrives (Figure 1a, purple).

Hyperdimensional (HD) computing, a computing paradigm using random, high-dimensional hypervectors (HVs) as units of representation, satisfies these needs [25]. For classification, input features are first projected as HVs, distributing the information across of 1000's of elements consisting of ± 1. The HD classifier then operates as a nearest-prototype classifier, where prototypes are calculated by superimposing training HVs using element-wise addition or majority vote. The process of learning a new example consists entirely of accumulating it into the appropriate prototype, compressing training examples into smaller model parameters with no dependence on the order in which data is presented. This makes HD classifiers inherently amenable to incremental learning.

HD computing has been applied in many biosignal classification tasks [26]–[28], and a fully embedded architecture capable of training, inference, and a single incremental update was demonstrated for EMG-based hand gesture recognition [6]. That study showed how two different limb positions could be learned in separate training sessions through continued accumulation of training examples in real time. A more recent pilot study with eight limb positions showed how sensor-fusion with an accelerometer could improve classification without drastically increasing model size [13]. However, there has been little work analyzing methods for maintaining prototypes in an incrementally trained HD classifier and their impact on classification accuracy and memory footprint.

In this work, we present a range of HD classifier configurations for incrementally learning hand gestures in multiple limb positions. We investigate choices for how accelerometer signals should be encoded to provide limb position context information, how new training HVs should be superimposed into existing prototypes, and also how many prototypes per class are allowed. To evaluate these choices, we collected a new dataset from 5 able-bodied subjects performing 13 hand gestures in 8 limb positions (Figure 1b), and we created an incremental training scenario in which new limb position contexts were learned one at a time in random order. We compared the configurations based on classification accuracy and memory footprint, and we implemented incrementally trained SVM and LDA models as traditional ML comparisons.

A key result we demonstrate is that performing a majority vote computation for incremental superposition in two stages not only reduces the memory required to store intermediate values between incremental steps, but it also improves the accuracy across all seen contexts. We then proceed to show how various configurations trade off classification accuracy and memory footprint, ranging from the smallest model with 85.71% accuracy and 264 kb of required memory to the largest with 97.15% accuracy and 1.125 Mb of memory footprint. Finally, by comparing with traditional ML models, we show that HD computing can achieve comparable classification accuracies while requiring an order of magnitude smaller memory allocation, making it ideal for implementation on resource-constrained devices.

II. Methods

A. Dataset collection and feature extraction

We collected data from 5 subjects (2 male and 3 female) performing 13 gestures in 8 limb positions while measuring their forearm EMG and wrist-mounted accelerometer signals, expanding on a pilot study with a single subject [13]. Data were acquired using a custom wearable EMG device for

wireless biosignal acquisition [6], [29], with a 64-channel electrode array wrapped around the largest section of the forearm and an accelerometer unit fixed a few inches above the wrist and over the ulna. For this study, the array was fabricated using a standard commercial flexible printed circuit board process, rather than using the custom screen-printed array shown in prior work [6]. The 13 gesture classes included 12 single degree-of-freedom movements of each digit along with the rest class (no movement). Eight different limb positions were chosen based on literature to mimic ones that may be encountered in everyday use (Fig. 1b) [15].

Each gesture and limb position combination was performed three times. Each repetition began in the default limb position (Position 0) and the rest gesture, followed by transitioning to the directed limb position and directed gesture, holding the gesture for four seconds, and then returning to the default position and rest gesture. For analysis, only the central four-second steady-state period of the gesture performance was used. Each period was divided into 50 ms windows for feature extraction of the mean absolute value (MAV) of each of the 64 EMG channels and the average x-, y-, and z-axis acceleration.

B. Hyperdimensional computing for gesture recognition

Classification using HD computing involves projecting features into bipolar HVs ($\{+1,-1\}^D, D = 10,000$), with key building blocks including a random item memory (IM) representing categorical information, a continuous item memory (CIM) representing ranges of quantized values, and algebraic operations to manipulate and combine HVs from these memories [25]. These operations include element-wise addition ($+$) and multiplication ($*$) between HVs, permutation (ρ) of the elements within an HV, and scalar multiplication between an HV and a scalar. The result of adding bipolar HVs is passed through a sign function ($\sigma(x) = +1$ if $x > 0$, $\sigma(x) = -1$ if $x < 0$) with random tiebreaks to produce superimposed bipolar HVs with element-wise majority vote. For this study, we implemented a projection architecture that has been successful in EMG classification [6], [13], [26]. We represented each electrode with a random, orthogonal HV stored in an IM. First, spatial encoding was performed by computing a bipolarized weighted sum of these HVs using the per-channel feature values as weights. Temporal information was then encoded using a permute and multiply operation across $N_{temporal} = 5$ consecutive spatial HVs. Each encoded EMG HV represented 250 ms of data, with an overlap of 200 ms.

As shown previously, orthogonalizing HVs from different contexts enables better classification performance when superimposing them into a single prototype [13]. We implemented the two types of context encoding for orthogonalization described in that work. For the first, accelerometer features were quantized and represented by a CIM for each axis, and a single context HV was produced by multiplying the retrieved HVs. The number of quantization levels and distance between neighboring CIM HVs were optimized for each subject individually. For the second encoding, random, orthogonal context HVs were assigned to each limb position context. Prior to training or inference, context encoding was performed by multiplying an EMG HV either type of context HV.

Classification of query HVs was performed through nearest neighbor search among class prototypes. Prototypes were calculated by superimposing training examples from each class through element-wise majority, and similarity between prototypes and queries was measured using Hamming distance.

C. Incremental learning with HD classifiers

We considered a scenario in which we incrementally train on a sequence of m datasets $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_m$ where each set consists of $\mathcal{S}_i = \{(x_{i,j}, y_{i,j})\}$ with $x_{i,j} \in X$ and $y_{i,j} \in \{1, 2, ..., k\}, 1 \leq j \leq n_i$ being training examples and labels, X being either the feature space or HD space, k being the number of classes, and n_i being the size of each dataset $\mathcal{S}_i$. Each incremental dataset corresponds to a context, modeling a scenario in which new limb positions are incrementally learned over time. We represent the initial classifier trained on $\mathcal{S}_1$ as CLF_1, and subsequent updated classifiers as $CLF_i = f_{train}(\mathcal{S}_i, CLF_{i-1})$ where updated classifiers can only access the latest dataset and previously trained classifier parameters.

For each new incremental training dataset, we can compute candidate prototypes representing that specific context. Therefore, we use the terms "candidate prototype" and "context-specific prototype" interchangeably. For HD computing, where incremental training and batch training can differ are:

1) How the candidate prototypes are incorporated into the model's prototype memory, and
2) What representations of the stored prototypes (or previous training examples) are available

We can either superimpose a candidate prototype with an existing prototype, or, if our memory budget permits, we can store it separately such that multiple prototypes represent the same gesture. We first describe various algorithmic design choices for incrementally superimposing candidates with stored prototypes, beginning with a method that produces an identical model as with batch training, and introducing configurations with increasing optimization and approximation. We then describe the overall algorithm with the ability to either superimpose candidates or store them separately for a trade-off between memory footprint and classification performance.

1) Incremental prototype superposition: If all training examples are retained in memory as model "parameters", then each incremental step can consist of batch training with all data seen until this point, $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup ... \cup \mathcal{S}_i$ (Figure 1a). This storage would require $D \times n$ bits in memory, where $n = \sum_i n_i$ is the combined size of all incremental datasets. However, we ultimately only require the class centroids, and thus we only need to store their accumulated sums from each class (Fig. 2a). We will refer to this method as *example accumulation*, and, assuming class-balanced training examples, this reduces the required memory footprint to be $D \times k \times \left(\lfloor \log_2\left(\frac{n}{k} + 1\right)\rfloor + 1\right)$ bits, where $\lfloor \log_2(N+1)\rfloor + 1$ is the memory required to accurately sum N bipolar bits. For our dataset with 240 examples per class per context ($n = 240 \times m \times k = 24,960$), this amounts to a reduction from $10,000 \times 24,960 \approx 238$ Mb required to store all training examples down to $10,000 \times 13 \times 11 \approx 1.36$ Mb to only store their sums.

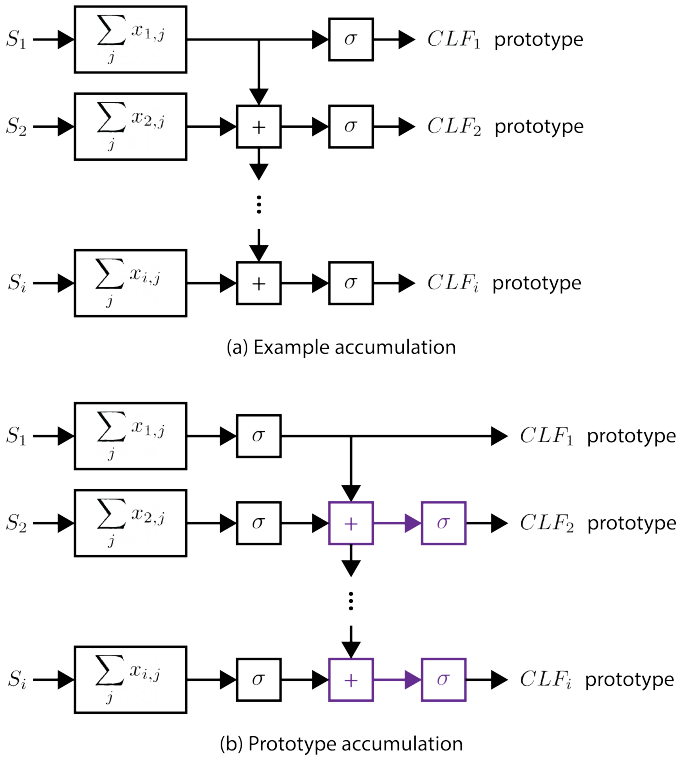

Fig. 2. HD classifier incremental superposition configurations. Rows of operations in (a) and (b) show updating a prototype during an incremental training step, with S_i being the ith incremental training set, $x_{i,j}$ being training example HVs, σ being the bipolarization function, and CLF_i being the classifier after i steps. Vertical arrows indicate values that must be preserved for the next incremental training iteration. (a) *Example accumulation* of all previously seen training examples. (b) *Prototype accumulation* from each dataset requires a lower bit-width representation in memory. Prototypes are accumulated using integer addition (shown) or through bit-wise merge (replacing the purple blocks).

The first optimization we can make is to only store and accumulate bipolarized candidate prototypes at each step (Fig. 2b). *Prototype accumulation* requires fewer bits per element than example accumulation, with overall memory footprint being $D \times k \times (\lfloor \log_2(m+1) \rfloor + 1)$. In our case with 13 gestures and 8 limb positions, we now only require $10,000 \times 13 \times 4 \approx 508$ kb of memory.

The benefit of prototype accumulation is two-fold: not only does it require less memory than example accumulation, but it also potentially improves classification performance. Example accumulation from multiple contexts with differing class cluster spread creates prototypes with unbalanced representation of the different contexts. By cluster spread, we refer to the average similarity between HVs belonging to the same class. In superposition using majority vote, classes with low spread overwhelm more spread-out classes, leading to a prototype that is more similar to the tighter class's centroid and distant from the noisier class's centroid. This results in especially bad performance when encoding limb position information with orthogonal context HVs, as the prototype may end up completely orthogonal to certain contexts it is supposed to represent. On the other hand, prototype accumulation ensures even context representation, as only a single HV from each context is included in the sum.

Beyond this, we can approximate prototype accumulation to further reduce model parameter memory size. Rather than storing integer values, we instead only need 1 bit of memory per prototype element if we use bit-wise merge to approximate accumulation (Fig. 2b). The ith candidate prototype can be merged into the stored prototype by randomly replacing $1/i$ of the stored bits with bits from the candidate. This "learning rate" of $1/i$ ensures that all seen candidate prototypes are approximately equally represented [30]. While this results in noisier prototypes, we reduce the parameter memory for our scenario to $10,000 \times 13 \approx 127$ kb. A comparative summary of these configurations is given in Table I.

2) HD computing incremental learning algorithm: The alternative to superimposing a candidate prototype is inserting it as a *separate* prototype representing the same class. Interference caused by different contexts in superposition is eliminated, and the method is akin to using an ensemble of classifiers. Whether a candidate prototype is kept separate must be consistent across all gesture classes, requiring an additional $D \times k$ bits of parameter memory. This decision can be based on a memory budget for the number of separated contexts allowed, m_{sep}. Before the budget is reached, the number of prototypes per class grows with the number of incremental training steps. After meeting the budget, one context will be selected as the target context for future superposition, either at random or based on its age. In subsequent steps, either new prototypes or an existing separate prototype will be superimposed onto the target prototype. The overall HD incremental learning algorithm is described in Algorithm 1.

Algorithm 1: HD classifier incremental training

Input: Memory budget of m_{sep} separate prototypes per class and one superimposed prototype representing m_{sup} contexts per class; incremental superposition configuration; sequence of incremental training datasets $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_m$
Result: Incrementally trained HD classification model

```
for S_i, i ∈ 1, 2, ..., m_sep + m_sup do
    for g ∈ 1, 2, ..., k do
        Compute candidate prototype for x_{i,j}, y_{i,j} ∈ S_i ∀ y_{i,j} == g
    end
    if i ≤ m_sep then
        Store candidate prototypes as new AM prototypes
    else
        Randomly select between new context and stored separate contexts
        if i = m_sep + 1 then
            Set aside selected context prototypes as target prototypes
        else
            Accumulate selected context prototypes onto target prototypes
              using incremental superposition configuration
        end
    end
end
```

3) Memory allocation: In addition to parameter memory, a scratch memory is needed to store intermediate values during feature projection to HVs and incremental training operations. Random, orthogonal IM HVs can be generated sequentially using a rule 90 cellular automaton with hardwired seeds and connections [30], [31]. For spatial encoding, a single 32-bit floating point accumulator can be used to generate the weighted sum for each, independent element. D bits are required for storage of the bipolarized spatial HV. For temporal encoding, we must store the previous $N_{temporal} - 1$ projected spatial HVs as well, totaling a memory requirement

Configuration	Incremental superposition operations	Parameter memory calculation	Est. parameter memory[a]
Example accumulation	$\sigma\left(\sum_{l=1}^{i-1}\left(\sum_{j=1}^{n_l} x_{l,j}\right)+\sum_{j=1}^{n_i} x_{i,j}\right)$	$D\times k\times\left(\lfloor\log_2\left(\frac{n}{k}+1\right)\rfloor+1\right)$	1.36 Mb
Prototype accumulation	$\sigma\left(\sum_{l=1}^{i-1}\sigma\left(\sum_{j=1}^{n_l} x_{l,j}\right)+\sigma\left(\sum_{j=1}^{n_i} x_{i,j}\right)\right)$	$D\times k\times(\lfloor\log_2(m+1)\rfloor+1)$	508 kb
Prototype merge	$\text{merge}\left(\text{merge}_{l=1}^{i-1}\left(\sigma\left(\sum_{j=1}^{n_l} x_{l,j}\right)\right),\sigma\left(\sum_{j=1}^{n_i} x_{i,j}\right)\right)$	$D\times k$	127 kb

[a] $D=10,000; k=13; n=24,960; m=8; 1\text{ Mb}=1024\text{ kb}=1024^2\text{ b}$

TABLE I
INCREMENTAL HV SUPERPOSITION CONFIGURATIONS

of $D\times N_{temporal}+32$ bits for projection of EMG data.

For projection and classification of accelerometer data, we can also generate CIMs based on hardwired logic gates and seeds from the IM generator [30]. A single D-bit memory is required for storage and sequential binding of the three encoded accelerometer axes. If we wish to perform orthogonal encoding, we can also implement a small HD model for context classification, with a reduced HV dimension employing only a subset of the bits from accelerometer encoded context HVs. We found that a dimension $D_{accel}=316$ was sufficient for a context classification accuracy of 97.36%, within 0.5% of the accuracy using full 10,000-D HVs.

For computation of candidate prototypes, projected HVs can be accumulated using a D-dimensional array of 8-bit saturating counters for 240 training examples per class in each incremental dataset. A summary of the memory footprints for various HD incremental learning configurations is given in Table II. In building a classifier, an encoding method (columns) must be chosen for context information, and an incremental superposition method (rows) must be chosen to update superimposed prototypes. A list of hyperparameters is given, along with values that have been implemented for hand gesture recognition. In particular, selection of m_{sep}, the budget for separate prototypes per class, plays a large role in both accuracy and memory footprint.

D. Comparison methods

As comparisons to our HD classifier, we incrementally trained two traditional ML algorithms—support vector machine (SVM) and linear discriminant analysis (LDA)—commonly used for EMG-based gesture recognition. We used the last $N_{temporal}=5$ MAV features for each channel along with the 3 accelerometer axis features to build 323-element feature vectors. In addition, the entire dataset was Z-scored when using SVM. We chose to compare models based on accuracy and memory footprint only, as computation time or complexity comparisons would depend on the exact implementations of each algorithm. For gesture recognition, classification throughput is low (e.g., 20 classifications per second) and data windows are long (e.g., 250 ms), so algorithm latency is a relatively small portion of the overall system latency. Incorporating hardware re-use and time multiplexing can make it especially difficult to make fair comparisons.

We chose a linear SVM classifier which required no kernel function evaluation that produced high classification accuracy results with our dataset. We tuned the regularization parameter C using 5-fold cross-validation in a batch setting for each subject. The optimal value $C=0.1$ offered the best classification performance while reducing the number of support vectors, thus reducing memory footprint. Although storing individual support vectors is not required for SVM inference, it is still needed for updating incrementally trained SVM models.

We first implemented a linear classifier trained using stochastic gradient descent (SGD) as a way to fit parameters for a linear SVM. This method did not require storage of previous training examples, instead updating model weights through a few iterations of SGD over each new incremental training dataset. Within each context, data from all gesture classes were made available at one time to allow for shuffling, necessitating a large scratch memory during each training step.

Next, we explored methods for augmenting and curating datasets for retraining. After each incremental step, we reserved the model support vectors to augment the next incremental training dataset [32]. Each new dataset could consist of either all gestures or a single gesture in a new context. Fewer total steps were required when updating with all gestures, but this relied on a larger scratch memory for temporarily storing the data. Because the overall training effort and the resulting number of support vectors grow with the size of the training dataset [24], we also explored methods to subsample from the new datasets. The first method subsampled training examples which produced errors when tested on the existing model, based on the assumption that these examples were most likely to become new support vectors [33]. We further reduced the number of examples by also randomly subsampling a percentage of the training sets, trading off generalization for reduced memory.

We then trained an ensemble of SVM models, with one model for each context. We still enabled gesture-by-gesture training for each context-specific model, as well as training set reduction by randomly subsampling. Memory footprints for the incrementally trained SVM models were measured as the number of 323×32-bit floating point support vectors retained the next step. We also estimated the training scratch memory as the size of the subsampled training dataset used. This underestimates the memory requirement, as we didn't account for data standardization or additional memory for solving the SVM quadratic programming problem and computing coefficients for candidate support vectors.

For the LDA classifier, we first considered online updates to compute model parameters, which consisted of 323×32-bit mean feature vectors for each class and an overall $323\times 323\times 32$-bit covariance matrix. Model parameters could be updated on a sample-by-sample basis in an exact manner, requiring only the previous parameters and scratch memory to compute the update to the covariance matrix [34]. We also compared

Superposition configuration	Encoding configuration: None	Encoding configuration: Accelerometer	Encoding configuration: Orthogonal
Example accum.	$D \times N_{temporal} + 32$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\lfloor \mathbf{log_2} \left(\frac{\mathbf{n_m m_{sup}}}{\mathbf{k}} + \mathbf{1}\right) \rfloor + \mathbf{1} + \mathbf{m_{sep}}\right)$	$D \times N_{temporal} + 32$ $+\mathbf{D}$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\lfloor \mathbf{log_2} \left(\frac{\mathbf{n_m m_{sup}}}{\mathbf{k}} + \mathbf{1}\right) \rfloor + \mathbf{1} + \mathbf{m_{sep}}\right)$	$D \times N_{temporal} + 32$ $+\mathbf{D} + \mathbf{D_{accel}} \times \mathbf{m}$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\lfloor \mathbf{log_2} \left(\frac{\mathbf{n_m m_{sup}}}{\mathbf{k}} + \mathbf{1}\right) \rfloor + \mathbf{1} + \mathbf{m_{sep}}\right)$
Prototype accum.	$D \times N_{temporal} + 32$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\lfloor \mathbf{log_2} \left(\mathbf{m_{sup}} + \mathbf{1}\right) \rfloor + \mathbf{1} + \mathbf{m_{sep}}\right)$	$D \times N_{temporal} + 32$ $+\mathbf{D}$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\lfloor \mathbf{log_2} \left(\mathbf{m_{sup}} + \mathbf{1}\right) \rfloor + \mathbf{1} + \mathbf{m_{sep}}\right)$	$D \times N_{temporal} + 32$ $+\mathbf{D} + \mathbf{D_{accel}} \times \mathbf{m}$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\lfloor \mathbf{log_2} \left(\mathbf{m_{sup}} + \mathbf{1}\right) \rfloor + \mathbf{1} + \mathbf{m_{sep}}\right)$
Prototype merge	$D \times N_{temporal} + 32$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\mathbf{1} + \mathbf{m_{sep}}\right)$	$D \times N_{temporal} + 32$ $+\mathbf{D}$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\mathbf{1} + \mathbf{m_{sep}}\right)$	$D \times N_{temporal} + 32$ $+\mathbf{D} + \mathbf{D_{accel}} \times \mathbf{m}$ $+D \times \left(\lfloor \log_2 \left(\frac{n}{mk} + 1\right) \rfloor + 1\right)$ $+\mathbf{D} \times \mathbf{k} \times \left(\mathbf{1} + \mathbf{m_{sep}}\right)$

D = HV dimension $= 10,000$; $N_{temporal}$ = N-gram length $= 5$
n = total number of training examples $= 24,960$; m = total number of contexts/incremental steps $= 8$; k = total number of classes $= 13$
n_m = number of training examples per context/incremental step $= \frac{n}{m} = 3,120$
D_{accel} = HV dimension for accelerometer context classification $= 316$
m_{sup} = number of contexts expected for superposition; m_{sep} = number of contexts allowed to be stored separately
$m_{sup} + m_{sep} = m = 8$

TABLE II
MEMORY FOOTPRINT OF VARIOUS HDC INCREMENTAL LEARNING CONFIGURATIONS

using an ensemble of context-specific LDA classifiers, which requires the same incremental training effort, but m-times the parameter storage memory.

III. EXPERIMENTS & RESULTS

A. Comparison of incremental superposition methods

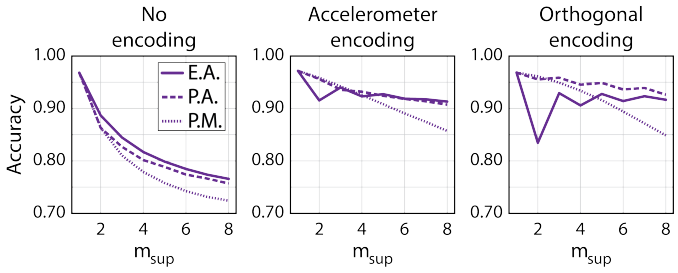

Fig. 3. Accuracy for increasing number of contexts in superposition using HD incremental superposition configurations. E.A. = example accumulation; P.A. = prototype accumulation; P.M. = prototype merge.

Fig. 3 demonstrates the ability of HD classifiers to learn different numbers of limb position contexts through incremental superposition. For each value of m_{sup}, a model with one prototype per class was incrementally trained using all possible combinations of m_{sup} context-specific datasets. As a reference, we were able to achieve 98.95% accuracy with SVM and 93.78% accuracy with LDA when batch training and testing in all 8 limb positions. Accelerometer encoding and orthogonal encoding greatly improved classification accuracy for larger m_{sup}, with the best performance achieved using orthogonal encoding and prototype accumulation. This configuration leveraged both the reduced interference from orthogonalizing HVs of different contexts, as well as the more balanced representation of two-stage bipolarization. The effect of unbalanced representation can be clearly seen in the example accumulation results for accelerometer and orthogonal encoding of $m_{sup} = 2$ contexts, where it was much more likely for a tighter cluster from one context to "overwhelm" a noisier cluster from another. This effect was reduced as more contexts were superimposed, resulting in increasing performance with orthogonal encoding for higher m_{sup}.

There was also a slight dependence of performance on the parity of m_{sup}, with odd values outperforming even values when using context encoding. By orthogonalizing different contexts, the likelihood of ties in superposition was higher when summing an even number of contexts. Ties should be broken at random, but slight differences in spread caused tie-breaking to skew towards one context, reducing representation of others. Skewed tie-breaking is less likely to occur when summing an odd number of contexts.

Finally, prototype merge is shown to be a useful approximation of prototype accumulation only for small values of m_{sup}. Beyond $m_{sup} = 3$ or 4, the limited capacity of merging orthogonal HVs caused accuracy to degrade more.

B. Incremental HD classifier performance

For incremental learning performance evaluation, we first divided the gesture recognition dataset into 5 folds for leave-one-out cross-validation. We further split training and testing datasets into context-specific datasets. We generated a random

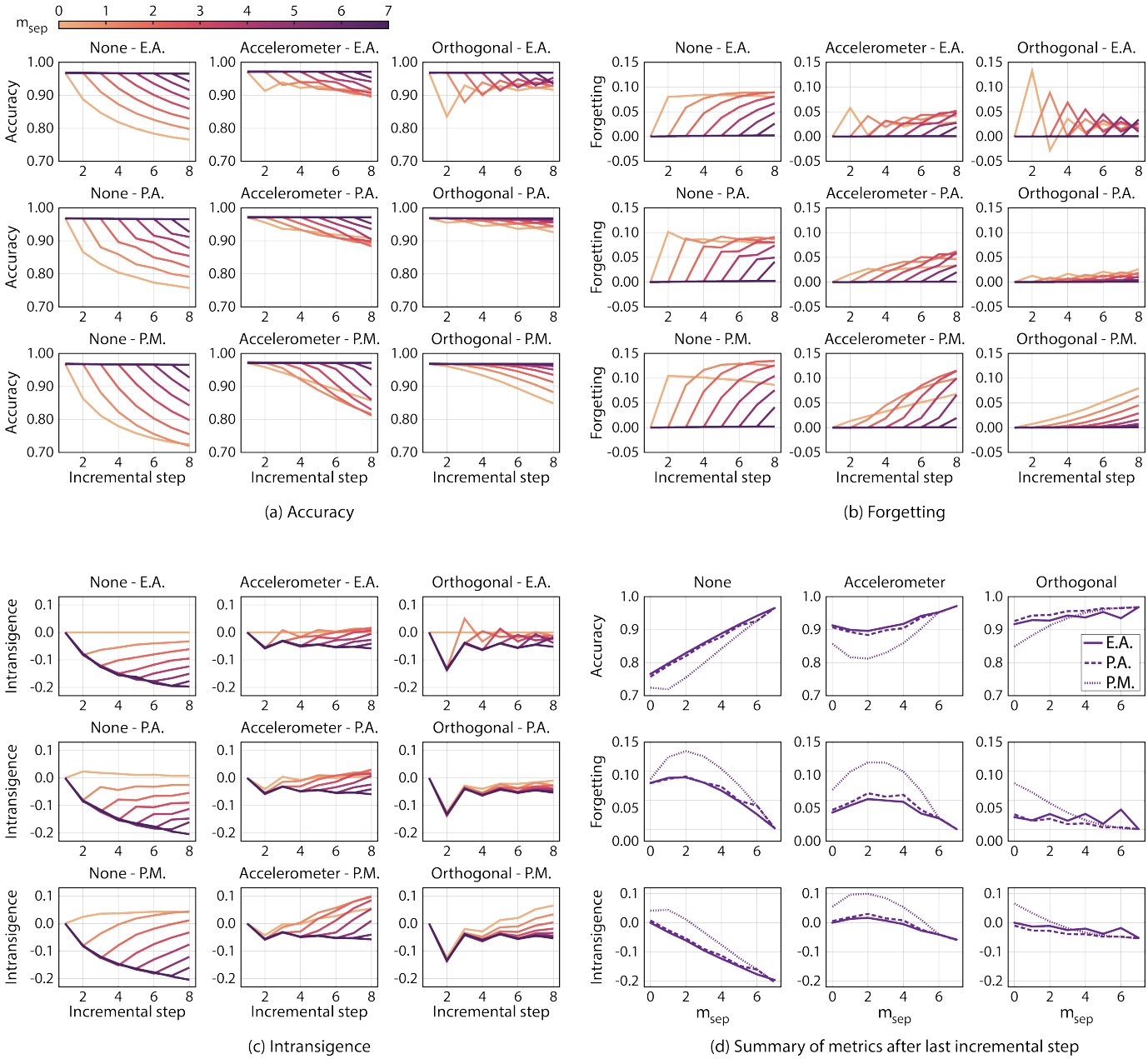

Fig. 4. Incremental learning performance using HD computing as measured in (a) accuracy, (b) forgetting, and (c) intransigence with respect to a batch trained HD classifier with $\boldsymbol{m_{sep}} = \mathbf{0}$. Within the $\mathbf{3} \times \mathbf{3}$ grids, each row represents a different incremental superposition configuration (E.A. = example accumulation; P.A. = prototype accumulation; P.M. = prototype merge) and each column represents a different context encoding method. Colors represent the value of $\boldsymbol{m_{sep}}$. (d) Summary of results after the last incremental step with varying $\boldsymbol{m_{sep}}$. Each row represents a different metric, each column represents a different context encoding method, and line styles represent the incremental superposition configuration.

ordering for the contexts and provided each context-specific training dataset to the incremental learning algorithm one at a time. After each step, the model was tested on all testing datasets corresponding to the seen contexts, and this was continued until all contexts were learned. This process was repeated for each subject individually, and at least 128 random context permutations were generated for each experiment.

We measured three metrics during the incremental training process, the first being average classification accuracy. We computed context-specific classification accuracy $a_{i,j}$ corresponding to the accuracy of the jth seen context after i training steps. The average classification accuracy was then computed as $A_i = \frac{1}{i}\sum_{j=1}^{i} a_{i,j}$.

While accuracy reflected the overall performance, we could better analyze model deficiencies by computing forgetting and intransigence [35]. Forgetting quantifies how much the incremental learning algorithm loses what it has learned in the past due to overwriting or interfering with relevant parameters. We calculated forgetting for the jth seen context after the ith step as $f_{i,j} = \max_{l\in\{1,2,\ldots,i-1\}} a_{l,j} - a_{i,j}$. The first term was the best classification accuracy ever achieved by the model, and the second was the accuracy with the updated model. More positive values of $f_{i,j}$ were worse and indicated more forgetting, while negative values indicated that the model had

improved for that context after incrementally training with other contexts. Average forgetting for each step was then computed as $F_i = \frac{1}{i-1}\sum_{j=1}^{i-1} f_{i,j}$.

Intransigence, on the other hand, measures the model's inability to incrementally learn a new context, compared to if the model was batch trained. This was computed as $I_i = a_i^* - a_{i,i}$, where a_i^* was the batch trained model accuracy using the same data. Like forgetting, poorer intransigence was represented by greater positive values. For the incremental HD classifier, we measured intransigence against a batch-trained HD model with example accumulation and $m_{sep} = 0$.

Fig. 4 shows metrics measured after each incremental step for various configurations of our HD classifier. In each 3×3 grid of Figs. 4a-c, each configuration is represented by one plot with different colored lines for varying m_{sep}. Metrics were averaged over all subjects, all possible context permutations, and all possible groupings of superimposed and separate contexts. Each step is shown as incrementally learning an entire new limb position context, although the training process was able to incorporate new context data one gesture at a time.

Average classification accuracy (Fig. 4a) degraded as more contexts were stored in superposition. This is expected from the superposition analysis, with the curves representing $m_{sep} = 0$ in Fig. 4a being identical to those in Fig. 3. Higher m_{sep} budgets traded off memory footprint for better performance. However, storing superimposed prototypes together with separate prototypes also introduced some negative effects, as shown in the top row of Fig. 4d. Here, we plot the metrics achieved by each configuration after completing all 8 incremental steps, demonstrating that there was at first a slight accuracy degradation when increasing m_{sep} with accelerometer encoding, contrary to intuition. This was due to certain pairs of contexts producing more similar accelerometer encoded context vectors (e.g., positions 0 and 4). With smaller, non-zero m_{sep}, there was an increased chance that one of the pair was stored separately while the other was superimposed with a large number of more distant contexts. Examples from the superimposed context could be most similar to the separate paired context prototypes, causing misclassification.

The causes of accuracy degradation over incremental training steps can be seen in the average forgetting, plotted in Fig. 4b. Compared to $m_{sep} = 0$, small non-zero values for m_{sep} caused worse forgetting since superimposed contexts were randomly selected. In these cases, there was a greater likelihood that prototypes for a context were at one point stored separately, achieving maximum within-context accuracy, and then subsequently superimposed, causing some accuracy degradation. Along with the paired context effect explained above, this caused a peak in average forgetting after all incremental steps to fall near $m_{sep} = 2$ for no encoding and accelerometer encoding, as shown in the middle row of Fig. 4d. As we increased m_{sep} to allow for separate prototypes for all contexts, the forgetting dropped to near 0. For orthogonal encoding, there was no peak, but rather a dependence on the parity of m_{sep} when using example accumulation.

Finally, plotting intransigence shows that incrementally trained HD classifiers were for the most part more capable of incorporating new contexts than the batch trained model due to their ability to store new prototypes separately (Fig. 4c). This effect was much more pronounced when not using any context encoding. When constrained to superimpose new contexts, models exhibited higher intransigence with the number of incremental training steps. However, using prototype accumulation with orthogonal encoding resulted in negative intransigence even for $m_{sep} = 0$, suggesting that bipolarization in two stages should also be leveraged during batch training for improved performance.

C. Incremental SVM and LDA performance

Fig. 5 shows the performance of SVM and LDA in our incremental learning experiments. The SVM trained using SGD (Fig. 5a) experienced large amounts of accuracy degradation, with a classification accuracy under 70% after all incremental training steps. This was almost entirely due to forgetting, a well known issue with gradient descent-trained classifiers.

Methods of dataset augmentation for incrementally update an SVM classifier (Fig. 5b-e) produced results with similar dependencies on the percentage of training data used, with almost all error due to intransigence. Incrementally augmenting the dataset for each context (Fig. 5b) in one step produced more consistent accuracy early on than incrementally augmenting gesture-by-gesture (Fig. 5d). However, the performance difference was marginal after all steps. Augmenting the dataset with only misclassified examples (Fig. 5c, e) made a larger impact when incrementally training on entire contexts at a time than gesture-by-gesture.

Finally, the performance of the LDA classifier (Fig. 5f) showed that with accurate mean and covariance updates, classification accuracy fell gracefully due to forgetting.

D. Comparison of memory footprint

Fig. 6a shows memory footprint vs. classification accuracy for incremental HD classifier configurations after all incremental steps. As expected, superposition using example accumulation required the largest memory footprint to store higher-precision accumulations. Each configuration employing example accumulation was bested by a smaller configuration using prototype accumulation or merge for equivalent or better accuracy. Between prototype accumulation and merge configurations, accuracy depended mostly on the context encoding method. The Pareto set consisted almost entirely of orthogonal encoding configurations, except for the smallest, single-prototype configuration using prototype merge and accelerometer encoding. At the other extreme, all of the configurations performed equally well when $m_{sep} = m$.

It is worth noting that while memory footprint comparison is a good baseline, it does not paint the full picture. For example, of the configurations sized approximately 600 kb, the most accurate used prototype merge with $m_{sep} = 3$ for a total of 4 prototypes per class. Slightly below this in accuracy was the model using prototype accumulation with $m_{sep} = 0$. Despite having the same memory footprint, inference for this configuration would be much quicker, requiring $1/4$ the number of comparisons with a query. Since a comparison consists of $D = 10,000$ XORs followed by a popcount of D

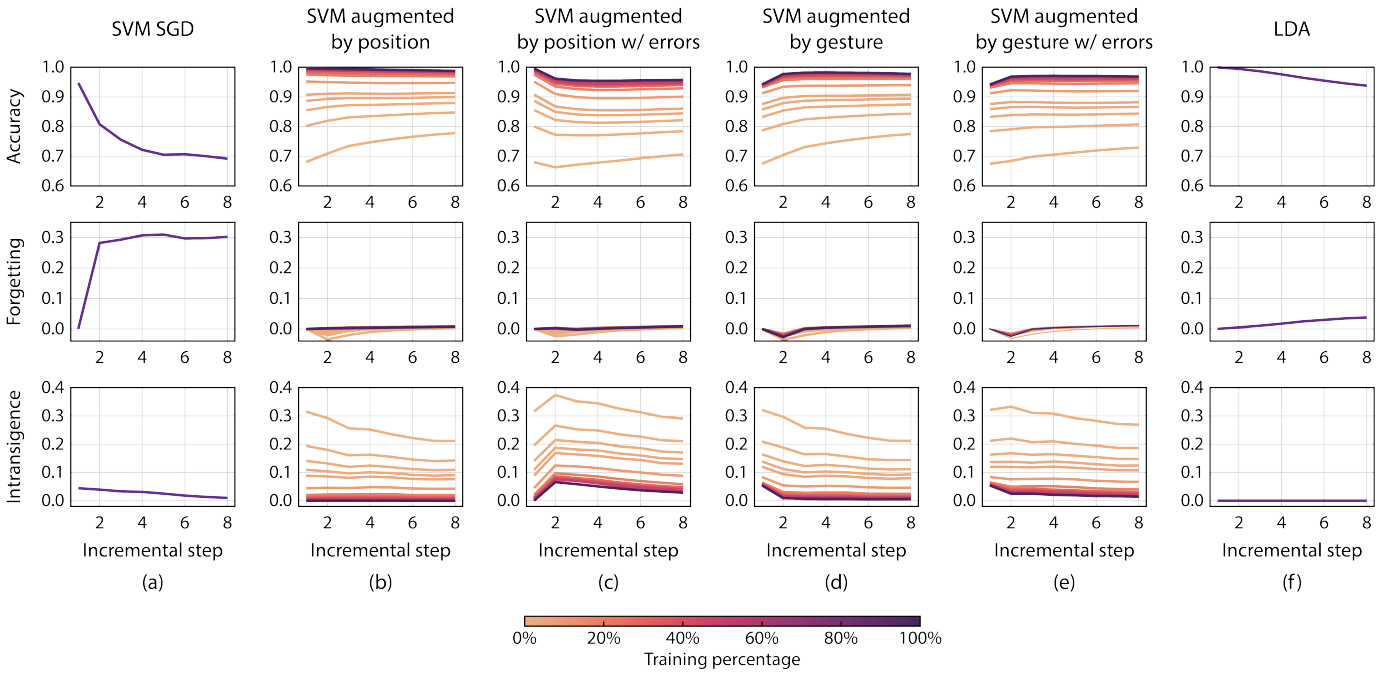

Fig. 5. Performance comparison with SVM and LDA classifiers. Each column shows accuracy, forgetting, and intransigence metrics for (a) SVM updated through stochastic gradient descent (SVM SGD), (b) SVM with data augmentation using entire context-specific datasets, (c) SVM with data augmentation using misclassified examples from entire context-specific datasets, (d) SVM with data augmentation using gesture-by-gesture datasets, (e) SVM with data augmentation using misclassified examples from gesture-by-gesture datasets, and (f) incrementally trained LDA. For the SVM with augmented datasets, different training percentages were used and plotted in different colors.

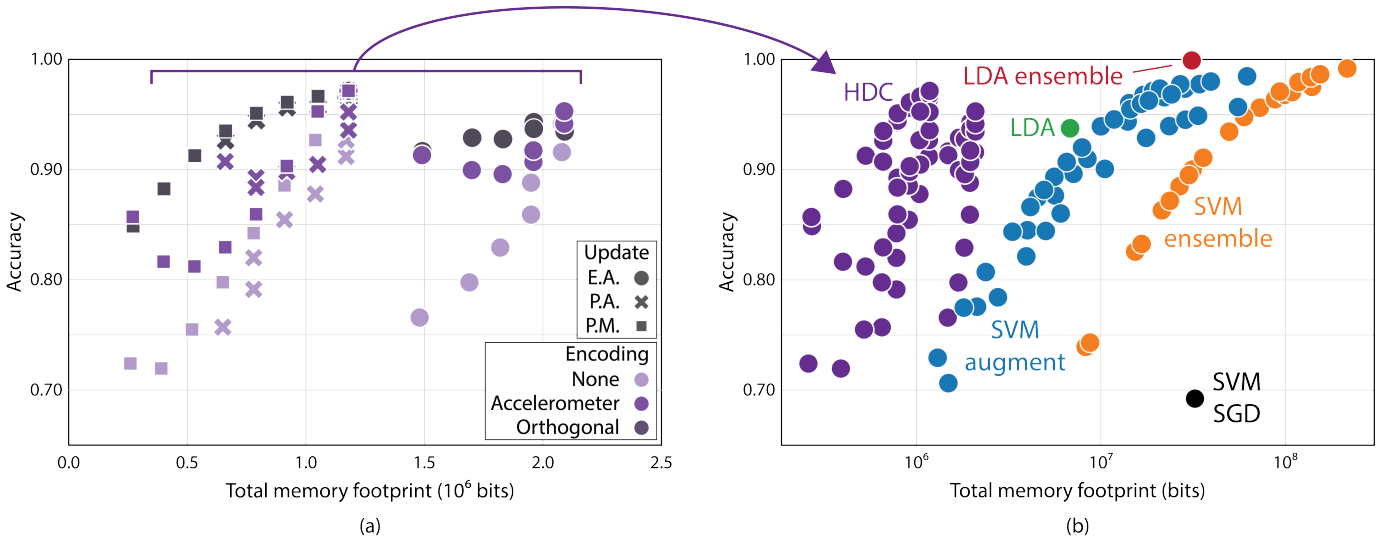

Fig. 6. Memory footprint for implementing various incremental HD classifier configurations and incrementally-trained traditional ML methods. (a) Differentiation between different HD classifier configurations. Total memory includes both parameter storage and scratch memory for training. Shapes represent the different incremental superposition configurations (E.A. = example accumulation; P.A. = prototype accumulation; P.M. = prototype merge), and colors represent the different encoding methods. (b) Comparison of HD configurations with traditional models. HDC = incremental HD; LDA ensemble = ensemble of context-specific LDA; LDA = incrementally updated LDA; SVM ensemble = ensemble of context-specific SVM; SVM augment = incrementally retrained SVM using dataset augmentation; SVM SGD = SVM trained with stochastic gradient descent.

bits, this could be a substantial saving in inference computation without much sacrifice in terms of accuracy.

Finally, we compare the incremental HD classifier configurations with traditional ML models in Fig. 6b. The best HD configurations were able to achieve similar accuracies while occupying over an order of magnitude smaller footprints.

The SVM trained using SGD was unable to produce acceptable results while also requiring a large memory footprint. Performance of SVMs incrementally trained through dataset augmentation was extremely dependent on the number of support vectors, which varied with training percentage. Even the smallest SVM models were larger than the best performing HD classifiers. Training an ensemble of context-specific SVMs improved the top-end classification accuracy but resulted in the largest of models we tested.

We were able to achieve the best overall accuracy using

an ensemble of context-specific LDA classifiers with a total memory footprint more than 26× the size of the most accurate HD model. Incrementally updating a single LDA classifier required only 3× more memory but had poorer classification performance. Our comparison shows that incremental HD classifiers were considerably more memory efficient than traditional methods and made up a large portion of the Pareto set when considering both memory footprint and accuracy.

IV. Conclusion

In this paper, we presented various configurations of incrementally trained HD classifiers for gesture recognition in different limb positions. We offer a range of context encoding and incremental HV superposition methods, and we demonstrate how increasing the allowance for multiple prototypes can improve accuracy. Selections will depend on platform memory allowances in which the model is to be embedded. Our experiments show that the best HD classifiers can be updated in multiple limb position contexts while maintaining a competitive classification accuracy of 97.15%. Compared to traditional ML models like SVM and LDA, which can require an order of magnitude more memory for similar performance, HD classifiers are far better suited for implementation on resource-constrained, wearable devices for rehabilitation and human-machine interface.

References

[1] D. Farina *et al.*, "The Extraction of Neural Information from the Surface EMG for the Control of Upper-Limb Prostheses: Emerging Avenues and Challenges," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 22, pp. 797–809, jul 2014.

[2] M. A. Oskoei and H. Hu, "Support vector machine-based classification scheme for myoelectric control applied to upper limb," *IEEE Trans. Biomed. Eng.*, 2008.

[3] K. Englehart and B. Hudgins, "A robust, real-time control scheme for multifunction myoelectric control," *IEEE Trans. Biomed. Eng.*, vol. 50, pp. 848–854, jul 2003.

[4] W. Geng *et al.*, "Gesture recognition by instantaneous surface EMG images," *Sci. Rep.*, 2016.

[5] S. Benatti *et al.*, "Online Learning and Classification of EMG-Based Gestures on a Parallel Ultra-Low Power Platform Using Hyperdimensional Computing," *IEEE Trans. Biomed. Circuits Syst.*, 2019.

[6] A. Moin *et al.*, "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition," *Nat. Electron.*, vol. 4, pp. 54–63, jan 2021.

[7] Ning Jiang *et al.*, "Myoelectric Control of Artificial Limbs—Is There a Need to Change Focus? [In the Spotlight]," *IEEE Signal Process. Mag.*, vol. 29, pp. 152–150, sep 2012.

[8] P. Kaufmann, K. Englehart, and M. Platzner, "Fluctuating emg signals: Investigating long-term effects of pattern matching algorithms," in *2010 Annu. Int. Conf. IEEE Eng. Med. Biol.* IEEE, aug 2010, pp. 6357–6360.

[9] C. Castellini *et al.*, "Proceedings of the first workshop on peripheral machine interfaces: Going beyond traditional surface electromyography," 2014.

[10] Y. Gu *et al.*, "Robust EMG pattern recognition in the presence of confounding factors: features, classifiers and adaptive learning," *Expert Syst. Appl.*, 2018.

[11] A. Fougner *et al.*, "Resolving the limb position effect in myoelectric pattern recognition," *IEEE Trans. Neural Syst. Rehabil. Eng.*, 2011.

[12] A. Radmand, E. Scheme, and K. Englehart, "A characterization of the effect of limb position on EMG features to guide the development of effective prosthetic control schemes," in *2014 36th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.* IEEE, aug 2014, pp. 662–667.

[13] A. Zhou, R. Muller, and J. Rabaey, "Memory-Efficient, Limb Position-Aware Hand Gesture Recognition using Hyperdimensional Computing," in *Proc. TinyML Res. Symp. (TinyML Res. Symp. '21)*, New York, NY, USA, 2021.

[14] Y. Geng, P. Zhou, and G. Li, "Toward attenuating the impact of arm positions on electromyography pattern-recognition based motion classification in transradial amputees," *J. Neuroeng. Rehabil.*, vol. 9, p. 74, 2012.

[15] E. Scheme *et al.*, "Examining the adverse effects of limb position on pattern recognition based myoelectric control," in *2010 Annu. Int. Conf. IEEE Eng. Med. Biol.* IEEE, aug 2010, pp. 6337–6340.

[16] W. Shahzad *et al.*, "Enhanced Performance for Multi-Forearm Movement Decoding Using Hybrid IMU–sEMG Interface," *Front. Neurorobot.*, vol. 13, jul 2019.

[17] J. W. Sensinger, B. A. Lock, and T. A. Kuiken, "Adaptive Pattern Recognition of Myoelectric Signals: Exploration of Conceptual Framework and Practical Algorithms," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 17, pp. 270–278, jun 2009.

[18] D. Yang *et al.*, "Adaptive learning of multi-finger motion recognition based on support vector machine," in *2013 IEEE Int. Conf. Robot. Biomimetics*. IEEE, dec 2013, pp. 2231–2238.

[19] Q. Huang *et al.*, "A Novel Unsupervised Adaptive Learning Method for Long-Term Electromyography (EMG) Pattern Recognition," *Sensors*, vol. 17, p. 1370, jun 2017.

[20] M. M. Vidovic *et al.*, "Improving the Robustness of Myoelectric Pattern Recognition for Upper Limb Prostheses by Covariate Shift Adaptation," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 24, pp. 961–970, sep 2016.

[21] J. Liu *et al.*, "Reduced Daily Recalibration of Myoelectric Prosthesis Classifiers Based on Domain Adaptation," *IEEE J. Biomed. Heal. Informatics*, vol. 20, pp. 166–176, jan 2016.

[22] X. Chen, D. Zhang, and X. Zhu, "Application of a self-enhancing classification method to electromyography pattern recognition for multifunctional prosthesis control," *J. Neuroeng. Rehabil.*, vol. 10, p. 44, 2013.

[23] C. Diehl and G. Cauwenberghs, "Svm incremental learning, adaptation and optimization," in *Proc. Int. Jt. Conf. Neural Networks, 2003.*, vol. 4. IEEE, 2003, pp. 2685–2690.

[24] A. Bordes *et al.*, "Fast kernel classifiers with online and active learning," *J. Mach. Learn. Res.*, vol. 6, 2005.

[25] A. Rahimi *et al.*, "Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals," *Proc. IEEE*, vol. 107, pp. 123–143, jan 2019.

[26] A. Moin *et al.*, "An EMG Gesture Recognition System with Flexible High-Density Sensors and Brain-Inspired High-Dimensional Classifier," in *2018 IEEE Int. Symp. Circuits Syst.*, vol. 2018-May. IEEE, apr 2018, pp. 1–5.

[27] A. Rahimi *et al.*, "Hyperdimensional Computing for Blind and One-Shot Classification of EEG Error-Related Potentials," *Mob. Networks Appl.*, vol. 25, pp. 1958–1969, oct 2020.

[28] A. Burrello *et al.*, "One-shot Learning for iEEG Seizure Detection Using End-to-end Binary Operations: Local Binary Patterns with Hyperdimensional Computing," in *2018 IEEE Biomed. Circuits Syst. Conf.* IEEE, oct 2018, pp. 1–4.

[29] A. Zhou *et al.*, "A wireless and artefact-free 128-channel neuromodulation device for closed-loop stimulation and recording in non-human primates," *Nat. Biomed. Eng.*, vol. 3, pp. 15–26, jan 2019.

[30] M. Schmuck, L. Benini, and A. Rahimi, "Hardware Optimizations of Dense Binary Hyperdimensional Computing: Rematerialization of Hypervectors, Binarized Bundling, and Combinational Associative Memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, pp. 1–25, dec 2019.

[31] A. Menon *et al.*, "Efficient emotion recognition using hyperdimensional computing with combinatorial channel encoding and cellular automata," 2021.

[32] N. A. Syed, H. Liu, and K. K. Sung, "Handling concept drifts in incremental learning with support vector machines," in *Proc. fifth ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '99*. New York, New York, USA: ACM Press, 1999, pp. 317–321.

[33] C. Domeniconi and D. Gunopulos, "Incremental support vector machine construction," in *Proc. 2001 IEEE Int. Conf. Data Min.* IEEE Comput. Soc, 2001, pp. 589–592.

[34] X. Chen, D. Zhang, and X. Zhu, "Application of a self-enhancing classification method to electromyography pattern recognition for multifunctional prosthesis control," *J. Neuroeng. Rehabil.*, vol. 10, p. 44, 2013.

[35] A. Chaudhry *et al.*, "Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 2018, vol. 11215 LNCS, pp. 556–572.