Generalized Key-Value Memory to Flexibly Adjust Redundancy in Memory-Augmented Networks

Denis Kleyko, *Member, IEEE*, Geethan Karunaratne, Jan M. Rabaey, *Life Fellow, IEEE*, Abu Sebastian, *Senior Member, IEEE*, and Abbas Rahimi

***Abstract*— Memory-augmented neural networks enhance a neural network with an external key-value (KV) memory whose complexity is typically dominated by the number of support vectors in the key memory. We propose a generalized KV memory that decouples its dimension from the number of support vectors by introducing a free parameter that can arbitrarily add or remove redundancy to the key memory representation. In effect, it provides an additional degree of freedom to flexibly control the tradeoff between robustness and the resources required to store and compute the generalized KV memory. This is particularly useful for realizing the key memory on in-memory computing hardware where it exploits nonideal, but extremely efficient nonvolatile memory devices for dense storage and computation. Experimental results show that adapting this parameter on demand effectively mitigates up to 44% nonidealities, at equal accuracy and number of devices, without any need for neural network retraining.**

***Index Terms*— Hyperdimensional computing, in-memory computing, key-value (KV) memory, linear distributed memories with associations, memory-augmented neural networks (MANNs), nonvolatile memory (NVM), phase-change memory (PCM), vector symbolic architectures.**

I. Introduction

The idea of using memory for the neural networks has been widely used since the formulation of long short-term memory [1]. Recent approaches to memory-augmented neural networks (MANNs) incorporate an *explicit* memory into the neural networks as an end-to-end differentiable module [2]–[5]. These MANNs are typically applied in knowledge-based reasoning [2]–[4], sequential prediction [5], unsupervised learning [6], [7], and few-shot learning tasks [8], [9]. All these MANN models commonly expand the explicit memory to be able to handle various tasks and datasets with an increased complexity. For instance, the size of explicit memory grows linearly with the number of available samples and classes in the few-shot learning tasks [8], [9], or with the total number of training samples in the unsupervised learning [6], [7].

In the supervised learning tasks, the explicit memory is composed of a key memory for storing and comparing learned patterns, and a value memory for storing labels, that are jointly referred to as a key-value (KV) memory [4]. The entries in the key memory are not accessed by stating a hard address, but by comparing a query with all the entries, forming soft read and write operations, which involve every individual memory entry. These extremely memory intensive operations cause a bottleneck when implemented in conventional von Neumann architectures (e.g., CPUs and GPUs), especially for tasks demanding a large number of memory entries.

To address the aforementioned bottleneck, one viable option is to implement the KV memory with emerging nonvolatile memory (NVM) devices that offer dense storage as well as in-memory computing capability to efficiently execute the comparison operations at constant time. For instance, in [10] the NVM devices have been arranged as a ternary content addressable memory to perform comparisons inside the key memory. This structure, however, cannot support widely used metrics such as cosine similarity. Furthermore, practical in-memory computing is challenging due to low computational precision resulting from various sources of nonidealities in the NVM devices such as intrinsic randomness, noise, and variability [11]. A recent methodology [12] addresses these issues by enhancing the key memory representations with robust properties of hyperdimensional computing [13], such that the representations can be readily transformed to low-precision (i.e., bipolar or binary) vectors in the key memory while exhibiting robustness against the nonidealities in the NVM-based in-memory computing hardware.

In the few-shot learning tasks, for a given m-way n-shot problem, the methodology in [12] sets the size of the key memory to $[d \times mn]$ where m is the number of classes in the problem, n denotes the number of training samples per class, and d is the dimensionality of support vectors. Specifically, the controller neural network in [12] assigns a d-dimensional support vector to every training sample such that the support vectors for different classes are quasi-orthogonal. The dimensionality of support vector is typically set to thousands [13] to generate a holographic distributed representation that is extremely robust against nonidealities in in-memory computing [14]. Therefore, the dimensionality of support vectors could be changed to maintain a desired accuracy in the presence of the nonidealities for a fixed m-way n-shot problem. However, every time d is changed, the controller neural network should be retrained. This is impractical as retraining the controller neural network is a costly process. Therefore, it is important to consider alternative approaches for regulating the robustness of the KV memory. To address this limitation, we propose a generalization of the KV memory that can be dynamically adapted in the inference phase to deal with any amount of nonidealities.

Thus, the generalization of the KV memory is the main contribution of this brief. The proposed generalization decouples the

Manuscript received August 6, 2021; revised December 17, 2021; accepted March 10, 2022. The work of Denis Kleyko was supported in part by the European Union's Horizon 2020 Programme through the Marie Skłodowska-Curie Individual Fellowship under Grant 839179, in part by the Defense Advanced Research Projects Agency's (DARPA's) Artificial Intelligence Exploration (AIE) HyDDENN Project Program, and in part by the Air Force Office of Scientific Research (AFOSR) under Grant FA9550-19-1-0241. The work of Geethan Karunaratne and Abu Sebastian was supported in part by the European Research Council (ERC) through the European Unions Horizon 2020 Research and Innovation Program under Grant 682675. The work of Jan M. Rabaey was supported in part by the DARPA's AIE HyDDENN Project Program. *(Corresponding author: Denis Kleyko.)*

Denis Kleyko is with the Redwood Center for Theoretical Neuroscience, University of California, Berkeley, Berkeley, CA 94720 USA, and also with the Intelligent Systems Laboratory, Research Institutes of Sweden, 16440 Kista, Sweden (e-mail: denis.kleyko@ri.se).

Geethan Karunaratne, Abu Sebastian, and Abbas Rahimi are with IBM Research Zurich, 8803 Rüschlikon, Switzerland (e-mail: kar@zurich.ibm.com; ase@zurich.ibm.com; abr@zurich.ibm.com).

Jan M. Rabaey is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA 94720 USA (e-mail: jan_rabaey@berkeley.edu).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3159445.

Digital Object Identifier 10.1109/TNNLS.2022.3159445

dimension of the key memory from mn by introducing a free parameter r for controlling the redundancy such that the dimension becomes $[d \times r]$ instead of $[d \times mn]$. During the inference phase, this new parameter r can add or remove redundancy in the representations of the key memory, on demand, without any retraining of the controller neural network. This results in a fully distributed version of the key memory, which is obtained, by the linear superposition of the outer products between the support vectors and randomized distributed representations of their corresponding class labels. The empirical investigation of the generalized KV memory demonstrates its flexibility and robustness against noise and NVM nonidealities, e.g., it maintains the noiseless accuracy (obtained in software) of the original KV memory when exposed up to 44% NVM device nonidealities by adjusting r such that it demands no more NVM devices than the original KV memory.

The rest of this brief is structured as follows. Section II provides an overview of the original MANN architecture. The organization of the proposed generalized KV memory is presented in Section III. Section IV presents performance evaluation of the generalized KV memory. Section V concludes the brief.

II. MANNs Overview

The MANN architectures combine neural networks with an explicit memory [2]–[9], [12]. Such an approach exploits meta-learning for performing few-shot learning tasks [8], [9], [12]. The trained neural network can produce representations of new previously unseen data, which are then written to the explicit memory, so that the memory can be used to, e.g., classify new queries with only a few examples per each class. The distinctive feature of the MANN architecture in [12] is that the neural network is guided to produce support vectors in the form of d-dimensional vectors with the properties suitable for hyperdimensional computing [13], [15] (also known as vector symbolic architectures [16], [17]).[1] The architecture shows that the support vectors produced by the trained neural network can be directed toward robust bipolar or binary representations. It was shown to solve the Omniglot [20] problems, as large as 100-way five-shot using the in-memory computing hardware. Specifically, this architecture allows implementation of the binary key memory on 256 000 ($5 \times 100 \times 512$) noisy phase-change memory (PCM) devices, performing highly efficient analog in-memory computation, with less than 2.7% accuracy drop compared to the 32-bit real-valued memory in software for the largest problem ever-tried on Omniglot [12].

As follows from the above, conceptually the architecture can be divided into two parts: the controller (i.e., the neural network) and the explicit memory. This brief is devoted to the organization of the explicit memory during the inference phase.[2] The explicit memory is split into two parts: the key memory and the value memory (hence, the KV memory). In the original formulation, the key memory (denoted as $\mathbf{K}$) stores mn d-dimensional[3] support vectors produced by the controller for a given m-way n-shot problem so $\mathbf{K} \in [d \times mn]$. For the inference phase with PCM devices, every d-dimensional support vector is quantized to a binary or bipolar vector. The value memory (denoted as $\mathbf{V}$) stores one-hot encodings of the class labels corresponding to the support vectors in $\mathbf{K}$, so $\mathbf{V} \in [m \times mn]$. Here, it is important to emphasize that despite the fact that the KV memory contains the holographic distributed representations (i.e., the support vectors), the organization of the memory is local because the vectors $\mathbf{K}_i$ and $\mathbf{V}_i$ in the corresponding parts of the KV memory can be identified with a particular sample of the training data.

During the inference phase, the query vector $\mathbf{q} \in [d \times 1]$ is used as an input to the key memory where the main step is to compute the similarity between $\mathbf{q}$ and the support vectors $\mathbf{K}_i$ using the dot product (denoted as $\boldsymbol{\alpha}$) as the similarity measure

$$\boldsymbol{\alpha} = \mathbf{K}^\top \mathbf{q} \tag{1}$$

where α_i contains the similarity between the query and ith support vector. Note that when the support vectors in the key memory are normalized to the same norm, the dot products in $\boldsymbol{\alpha}$ are proportional to the corresponding cosine similarities.

Next, the dot products can be modified with some sharpening function [denoted as $\sigma(\cdot)$]

$$\boldsymbol{\gamma} = \sigma(\boldsymbol{\alpha}). \tag{2}$$

The sharpened similarity scores are used to compute the accumulated scores for each class as

$$\mathbf{s} = \mathbf{V}\boldsymbol{\gamma} \tag{3}$$

where s_j contains the score for jth class ($1 \leq j \leq m$). The prediction is chosen to be the class with the highest accumulated score: $\arg\max_j s_j$.

It is worth noting that the above inference procedure is a special case of the k-nearest neighbor classifier with distance-weighted voting where $k = mn$ and the weight for ith training sample (i.e., neighbor) corresponds to γ_i. This observation suggests that it is worth exploring a *fully distributed* organization of the KV memory where there is no *local* correspondence between the entries of the key memory and the support vectors. Such a fully distributed organization can be achieved using, e.g., a linear distributed memory with the outer product learning rule [21], [22]. The distributed organization of the KV memory allows achieving similar functionality for the inference phase while providing an additional degree of freedom, which plays an important role in controlling the tradeoff between the robustness of the key memory and the resources required to store it.

III. Generalized KV Memory

As highlighted in [12], an important advantage of the architecture is that the key memory can be mapped to PCM devices for analog in-memory computing, which has shown to significantly improve the energy efficiency of the inference procedure compared to a digital design. Note, however, that in the original formulation, the dimension of the key memory is $\mathbf{K} \in [d \times mn]$, i.e., assuming that d is fixed, the dimension of the key memory is determined by the number of support vectors in a given m-way n-shot problem. This dependency makes the key memory rigid in the sense that there is no possibility to control the dimension of the key memory other than changing d, which demands retraining the controller neural network, once m and n are fixed. Therefore, it is important to consider a generalization of the KV memory allowing to decouple the dimension of the key memory from m and n by using a free parameter r so the dimension becomes $[d \times r]$ instead of $[d \times mn]$.

The decoupling is achieved by changing the organization of the KV memory from the local one to a fully distributed one using the principles of forming context-dependent associations in linear distributed associative memories [21], [23], [24]. To reformulate the KV memory in terms of the context-dependent associations, we need to first define a new value memory ${}^{(d)}\mathbf{L} \in [r \times m]$, where ${}^{(d)}\mathbf{L}_j$ is an r-dimensional vector representing the label of jth class ($1 \leq j \leq m$). We will discuss the options for choosing ${}^{(d)}\mathbf{L}_j$ in Section IV. Once the class labels' representations are defined, the real-valued support

[1] Consult [18], [19] for a comprehensive survey of hyperdimensional computing/vector symbolic architectures.

[2] It is important to note that the proposed approach does not require any modification of the training of the controller described in [12].

[3] Following [12], d is set to 512 for the experiments in this brief.

vectors $\mathbf{K}_i$ and their corresponding ${}^{(d)}\mathbf{L}_j$ are used to create the distributed version of the key memory (denoted as ${}^{(d)}\mathbf{K}$) using the outer product learning rule

$$ {}^{(d)}\mathbf{K} = \sum_{i=1}^{mn} {}^{(d)}\mathbf{L}_{c(i)}\mathbf{K}_i^\top \tag{4} $$

where $c(i)$ denotes the class index of ith support vector. Thus, the distributed version of the key memory is the linear superposition of the outer products of the support vectors and the representations of their class labels,[4] therefore, ${}^{(d)}\mathbf{K} \in [r \times d]$. Since the distributed version of the key memory in (4) does not strictly depend on mn, we refer to ${}^{(d)}\mathbf{K}$ and ${}^{(d)}\mathbf{L}$ as the generalized KV memory.

The inference procedure with the generalized KV memory is very similar to the original one. For a given query, $\mathbf{q}$, the dot product between ${}^{(d)}\mathbf{K}$ and $\mathbf{q}$, which is computed as

$$ \boldsymbol{\gamma} = \boldsymbol{\alpha} = {}^{(d)}\mathbf{K}\mathbf{q} \tag{5} $$

produces an r-dimensional vector, which is the weighted superposition of class labels' representations. This vector $\boldsymbol{\alpha}$ can be seen as a sharpened vector $\boldsymbol{\gamma}$ assuming that an identity function is used as the sharpening function. Finally, the value memory is used to measure the scores of each class as

$$ \mathbf{s} = {}^{(d)}\mathbf{L}^\top \boldsymbol{\gamma}. \tag{6} $$

As in the original KV memory, the result is an m-dimensional vector where the jth component contains the accumulated score for the corresponding class so the prediction is chosen as before: $\arg\max_j s_j$.

Similar to the original key memory $\mathbf{K}$, ${}^{(d)}\mathbf{K}$ can be bipolarized using the componentwise sign$(\cdot)$ function

$$ {}^{(d)}\hat{\mathbf{K}} = \text{sign}({}^{(d)}\mathbf{K}). \tag{7} $$

Obviously, the bipolar version ${}^{(d)}\hat{\mathbf{K}}$ can be transformed to the binary version.

IV. Evaluation of the Generalized KV Memory

Here, we discuss when it would be beneficial to use the generalized key-memory instead of the original KV memory. We suggest the following two modes.

1) For compression when there is a little or no noise. For the problems where $n > 1$, the generalized KV memory effectively removes the redundancy and can achieve the classification accuracy on a par to the original KV memory for $r \ll mn$.
2) For increasing robustness at very low signal-to-noise ratio (SNR) conditions. The generalized KV memory flexibly increases the redundancy by allocating more resources to the key memory so that $r > mn$ that results in improved robustness to noise and nonidealities compared to the original KV memory, which does not have a mechanism to regulate the redundancy of the key memory.

Both the aforementioned modes are related to each other since usually it is necessary to tradeoff between the achieved accuracy and required resources in the presence of noise and nonidealities but to make the points clear, we isolate the two. In the following, we present the results of three sets of experiments to illustrate these modes. The first mode is investigated in Section IV-A while the second one is studied in Sections IV-B and IV-C.

We use the data from [12] in the form of 512-dimensional real-valued vectors obtained from the trained controller using the images in the test set of the Omniglot dataset. It includes 659 classes with 20 samples per each class. These data were used to perform the experiments below.

Recall that the first step in the generalized KV memory is to populate the value memory ${}^{(d)}\mathbf{L} \in [r \times m]$. There are several options to do so. The most obvious choice is to generate ${}^{(d)}\mathbf{L}$ randomly so that each class label is represented by a random vector ${}^{(d)}\mathbf{L}_j$. This, however, is not the best choice since the random vectors are only approximately orthogonal, hence, there will be some crosstalk noise between different ${}^{(d)}\mathbf{L}_j$ in ${}^{(d)}\mathbf{K}$, which would negatively affect the classification accuracy and robustness to noise and nonidealities. Therefore, in the experiments below we use random orthogonal matrices to form the value memory ${}^{(d)}\mathbf{L}$ when $r \geq m$, or whitened random matrices when $r < m$. In the former case, an orthogonal matrix can be formed by applying the QR decomposition to a random matrix generated from the standard normal distribution. The fact that values of ${}^{(d)}\mathbf{L}$ are real-valued should not be discouraging as it is meant to be implemented in software. In case if ${}^{(d)}\mathbf{L}$ is also desired to be binary/bipolar, one could use, e.g., Walsh codes to form ${}^{(d)}\mathbf{L}$.

A. Compression in Noiseless Condition

In the first experiment, we evaluate the tradeoff between the classification accuracy of the generalized KV memory and its dimension, which is controlled by r. Fig. 1 presents the average classification accuracy against r for two problems: 20-way five-shot (left panel) and 100-way five-shot (right panel). The markers depict the baselines obtained with the original KV memories[5] Recall, that the dimension of the original key memory is mn, i.e., 100 and 500 for the considered problems, respectively. The dashed lines correspond to the results of the generalized KV memories. For both problems, to provide the intuition for the dimensions of the original key memories, r was limited to mn. It is clear that for both problems, the accuracy of the generalized KV memories approached the accuracy of the original KV memories with values of r being smaller than mn. For example, for the 20-way five-shot problem, the generalized KV memory reached 95% of the accuracy of the original KV memory by using r of 10, 12, and 14 for real, bipolar, and binary key memory, respectively; these correspond to mn/r ratio, hence, memory saving of 10.0×, 8.3×, and 7.1×. For the 100-way five-shot problem, the corresponding values of r were 60, 70, and 80, with memory savings of 8.3×, 7.1×, and 6.3×, respectively. These results demonstrate the possibility of compressing the KV memory without sacrificing most of the classification accuracy.

B. Robustness in the Presence of White Noise

The previous experiment did not take into account the fact that a hardware implementation of the key memory might return a noisy version of $\boldsymbol{\gamma}$ as the result of computing the dot products. Therefore, in the second experiment, we measure the classification accuracy against the white noise added to $\boldsymbol{\gamma}$. The experiment is conducted with 20-way five-shot problem using all three variants of the key memory: real, bipolar, and binary. The dimension of the original key memory was fixed to $nm = 100$, while for the generalized KV memory flexibly sets four different values with $r \in \{50, 100, 150, 200\}$ without any need to retrain the controller.

[4]While here we do not go into the details of comparing the computational costs of the two considered approaches to the organization of the KV memory, it is worth noting that both of them incur certain computation costs when it comes to adding new support vectors to the key memory. In the case of the original KV memory, the cost is in the allocation of space for a new support vector and writing the new vector to the key memory; while in the case of the generalized KV memory, the cost is in computing the outer product between the support vector and its label vector as well as in incrementing the key memory with the outer product result.

[5]All the experimental results for the original KV memory were obtained using the identity function as the sharpening function in (3).

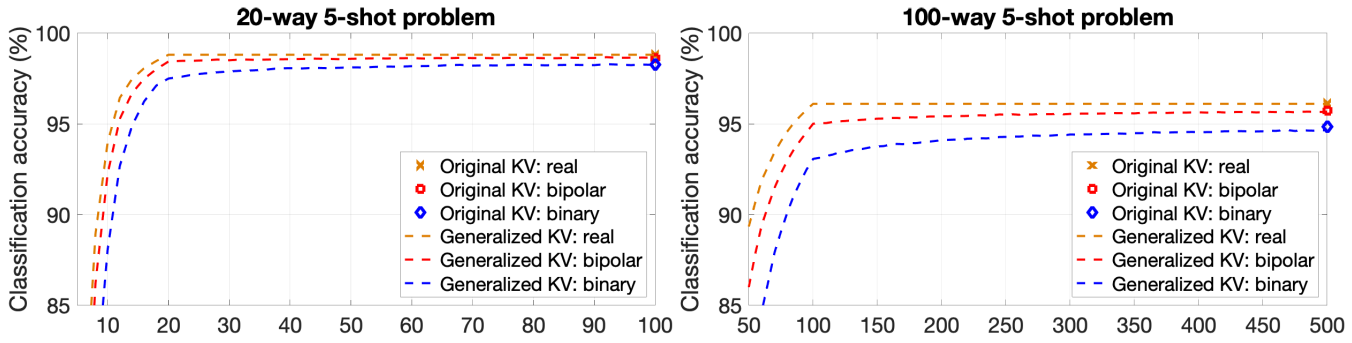

Fig. 1. Average classification accuracy against the dimensionality of representations of the class labels (r). The results are shown for the real, bipolar, and binary variants of the key memory and query vectors, in software without any noise. The results are averaged over 1000 problems randomly chosen from the test data.

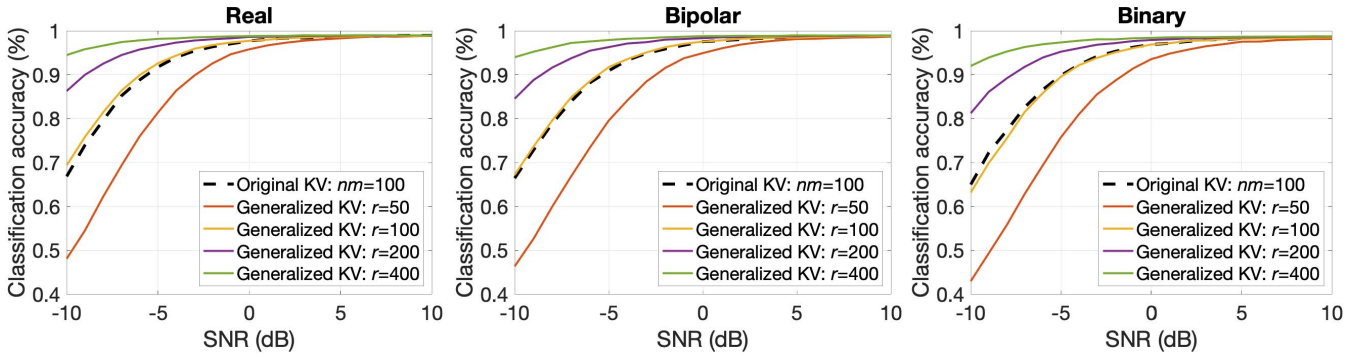

Fig. 2. Average classification accuracy against SNR of $\boldsymbol{\alpha}$. The 20-way five-shot problem is used with fixed $nm = 100$ while $r < nm$ or $r \geq nm$. Panels correspond to real, bipolar, and binary variants of the key memory and query vectors in software. The results are averaged over 100 problems randomly chosen from the test data.

Fig. 2 presents the results. As expected, the low SNR values reduced the classification accuracy. At the same time, it is clear that the dimension of the key memory has an effect on the robustness to noise. The generalized KV memory with the lowest r ($r = 50$) demonstrated the worst performance for the low SNR values. When $r = mn$, both memory types performed very close to each other independent of the SNR values. However, since the generalized KV memory can control r, it can be set to $r > mn$. Thus, the generalized KV memory can be flexibly switched to the second mode with a larger amount of redundancy to provide robustness to extremely low SNR values. For example, $r = 400$ exhibits a very graceful accuracy degradation (e.g., an accuracy of $>90\%$ at SNR $= -10$ dB). These results provide the evidence that subject to sufficient dimensionality of r, the generalized KV memory can be designed to operate at very low SNR.

C. Robustness in the Presence of PCM Nonidealities

It is important to note that the noise and nonidealities present in the PCM devices are not well-described by white noise. It has been shown that there are three major components to PCM noise. These include 1) a programming noise component which is modeled as multiplicative Gaussian noise; 2) drift noise component which models $1/f$ noise as a Gaussian random exponent with respect to time; and 3) read noise component which is modeled as additive Gaussian noise (see Supplementary notes in [12]). Of these, the programming noise variability can be directly controlled by employing an iterative programming scheme whereas drift and noise components are controlled by external conditions such as temperature or the internal conductance state, over which there is less controllability. Therefore, we perform the last set of experiments using the model of a temporal evolution of conductance $G(t)$ of a single PCM device [12]

$$G(t) = \mathcal{N}\left(0, \tilde{G}_r^2\right) + \left(G_0 \cdot \mathcal{N}\left(1, \tilde{G}_p^2\right)\right) \cdot t^{-\nu \cdot \mathcal{N}(1, \tilde{\nu}^2))} \tag{8}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes a normal distribution; t is the time since programming (assumed to be 20 s), G_0 is the mean conductance at $t = 1$ s (measured: $G_0 = 22.8 \times 10^{-6}$ S), ν is the mean drift exponent (measured: $\nu = 0.0598$); $\tilde{G}_r^2$, $\tilde{G}_p^2$, and $\tilde{\nu}^2$ represent the variation in additive read noise (measured: $\tilde{G}_r = 0.496 \times 10^{-6}$ S), conductance variation (programming noise; measured: $\tilde{G}_p = 31.7\%$), and drift variation (measured: $\tilde{\nu} = 9.07\%$), respectively. Please refer to "PCM model and simulations" subsection in "Method" section in [12] for the additional details of the model. We perform the experiments by varying the relative conductance variation (i.e., $\tilde{G}_p$) while keeping all other parameters fixed.

Figs. 3 and 4 present the average classification accuracy against the conductance variation in the PCM model for 20-way five-shot and 100-way five-shot problems, respectively.[6] The dimension of the original key memories are fixed to nm, while for the generalized KV memories different values of r are used: {50, 100, 150, 200} and {250, 500, 1000, 2000}, respectively. Note that the PCM implementation of the bipolar variants requires two devices per dimension.

The results for the original KV memory are consistent with the ones reported in [12] in the sense that the bipolar variant is more robust against the conductance variations. Also, similar to the results in Fig. 2, for both problems the generalized KV memory with

[6]The diligent readers are kindly referred to the Supplementary Material that provides an additional experimental evaluation to further justify the proposed approach.

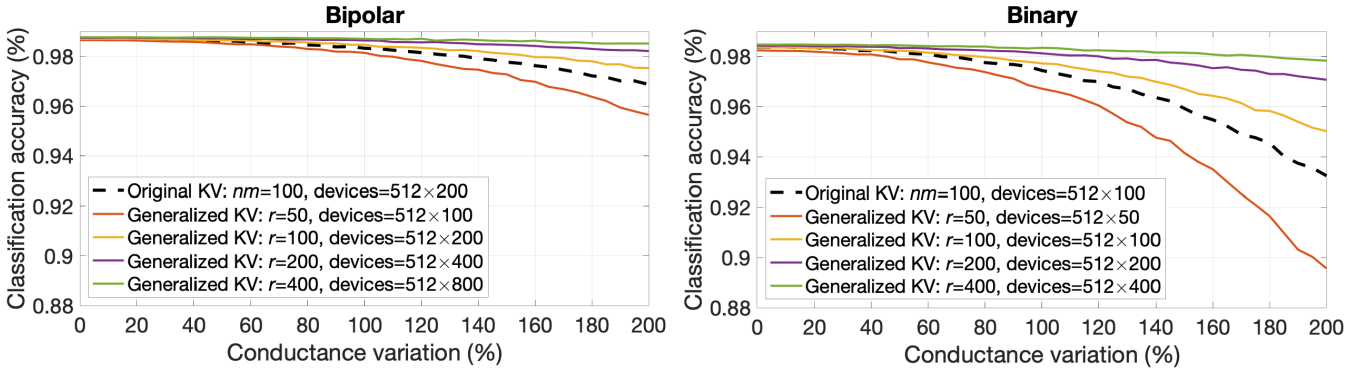

Fig. 3. Average classification accuracy against the conductance variations in the PCM model for 20-way five-shot problem. Panels correspond to the bipolar and the binary variants of the key memory mapped to PCM devices. The results were averaged over 1000 problems randomly chosen from the test data.

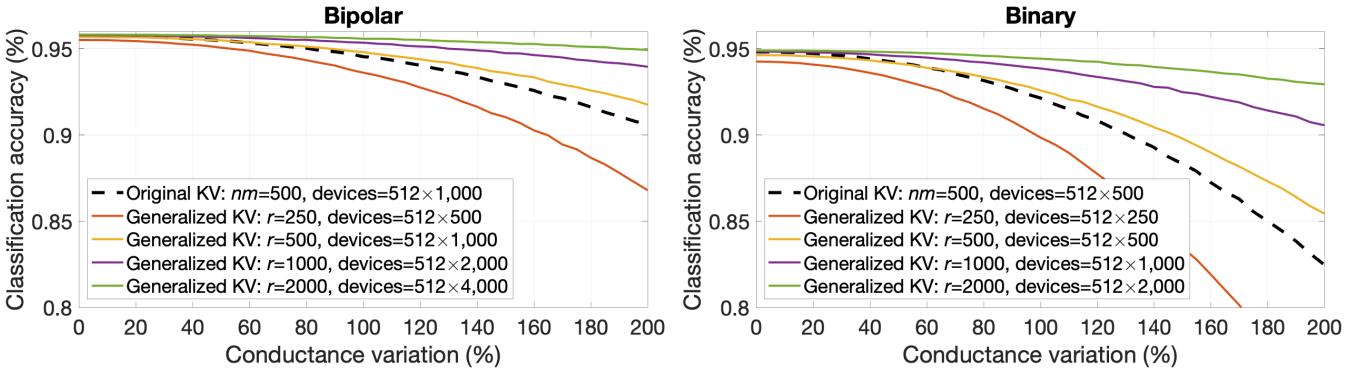

Fig. 4. Average classification accuracy against the conductance variations in the PCM model for 100-way five-shot problems. Panels correspond to the bipolar and the binary variants of the key memory mapped to PCM devices. The results were averaged over 1000 problems randomly chosen from the test data.

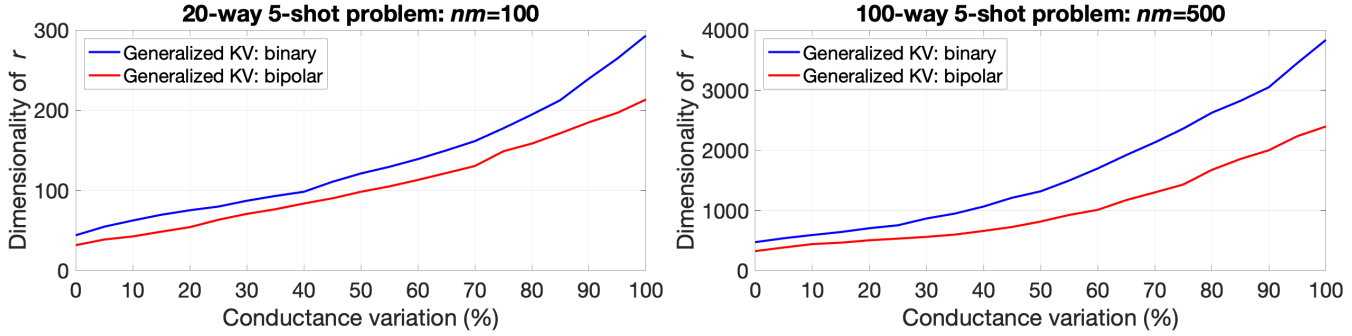

Fig. 5. Average dimensionality of r against the conductance variation in the PCM model required to maintain iso-accuracies of the corresponding variants of the original noiseless KV memory. Panels correspond to 20-way five-shot and 100-way five-shot problems, respectively. The results are averaged over 1000 problems randomly chosen from the test data.

the lowest r ($r = 50$ and $r = 250$, respectively) consistently demonstrates the lowest accuracy amongst all depicted. Importantly, the generalized KV memory performs better than the original one for high conductance variations (>80%) when $r = nm$ (i.e., having the same number of devices). This can be attributed to the fact that the local organization is more brittle to errors than the distributed one, which is a well-known advantage of distributed representations [25]. Finally, the robustness to the conductance variation can be increased further by increasing r, and naturally, the largest values of r ($r = 400$ and $r = 2000$, respectively) demonstrate the least accuracy degradation even at the very high conductance variation. In particular, for $\tilde{G}_p = 200\%$ compared to $\tilde{G}_p = 0\%$ the accuracy decreases by only 0.26% ($r = 400$) & 0.90% ($r = 2000$) and 0.64% ($r = 400$) & 1.97% ($r = 2000$) for the bipolar and binary variants, respectively.

Iso-Accuracy Generalized KV Memory at Conductance Variations: In the previous experiment, the generalized KV memory improves the robustness to conductance variations by increasing r. Hence, the next step is to investigate whether the generalized KV memory can achieve the iso-accuracy of the original KV memory without any noise and nonidealities as in Fig. 1. The average values of r providing the iso-accuracy for a range of conductance variations are depicted in Fig. 5.

Considering the 20-way five-shot problem with the binary vectors, the generalized KV memory can maintain the original accuracy of its noiseless software variant when experiencing up to 44% variations in

the PCM hardware, yet using the same number of devices ($r = mn$). Since in the hardware implementation, the "set" conductance was fixed at 38 μS, 44% conductance variation translates into a standard deviation of the "set" conductance distribution of 16 μS. For the larger amount of variations, increased r can guarantee the iso-accuracy; as expected, for the bipolar KV memories r grows slower than that of the binary ones. Similar trend is observed for the large 100-way five-shot problem. As shown, it is possible to achieve the iso-accuracy even for extremely high conductance variations, which confirms the general nature of the proposed KV memory, which allows trading-off additional hardware resources for the robustness of the classification accuracy.

V. Conclusion

This brief presented the generalized KV memory for MANNs. The proposed approach is based on the two key ideas. First, the fact that it is not necessary to use one-hot encodings for the value memory; randomized distributed representations can be used instead. Second, the organization of the key memory can be changed from the local one (storing individual support vectors) to the distributed one with the outer product learning rule.

The empirical evaluation demonstrated the flexibility of the generalized KV memory. In the noiseless conditions, it achieves the classification accuracy on a par with the original KV memory using a fraction of dimensions required by the original key memory. Alternatively, for very harsh conditions the generalized KV memory can easily adjust the memory redundancy to tolerate extreme amounts of noise and nonidealities, which is an attractive feature for implementing the KV memory on emerging computational NVM devices.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[2] A. Graves, G. Wayne, and I. Danihelka, "Neural Turing machines," 2014, *arXiv:1410.5401*.

[3] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[4] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 2440–2448.

[5] S. Sukhbaatar, A. szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 2440–2448.

[6] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 3733–3742.

[7] Z. Wu, A. A. Efros, and S. X. Yu, "Improving generalization via scalable neighborhood component analysis," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 685–701.

[8] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1–9.

[9] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 3637–3645.

[10] K. Ni *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nat. Electron.*, vol. 2, no. 11, pp. 521–529, 2019.

[11] A. Sebastian, M. L. Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnol.*, vol. 15, no. 7, pp. 529–544, 2020.

[12] G. Karunaratne *et al.*, "Robust high-dimensional memory-augmented neural networks," *Nature Commun.*, vol. 12, no. 1, pp. 1–12, Dec. 2021.

[13] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognit. Comput.*, vol. 1, no. 2, pp. 139–159, Oct. 2009.

[14] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electron.*, vol. 3, no. 6, pp. 327–337, Jun. 2020.

[15] A. Rahimi *et al.*, "High-dimensional computing as a nanoscalable paradigm," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 9, pp. 2508–2521, Sep. 2017.

[16] R. W. Gayler, "Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience," in *Proc. Joint Int. Conf. Cognit. Sci. (ICCS/ASCS)*, Dec. 2003, pp. 133–138.

[17] D. Kleyko *et al.*, "Vector symbolic architectures as a computing framework for nanoscale hardware," 2021, *arXiv:2106.05268*.

[18] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures—Part I: Models and data transformations," 2021, *arXiv:2111.06077*.

[19] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures—Part II: Applications, cognitive models, and challenges," 2021, *arXiv:2112.15424*.

[20] B. M. Lake, R. Salakhutdinov, and J. B. Tenebaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[21] E. Mizraji, "Context-dependent associations in linear distributed memories," *Bull. Math. Biol.*, vol. 51, no. 2, pp. 195–205, 1989.

[22] E. P. Frady and F. T. Sommer, "Robust computation with rhythmic spike patterns," *Proc. Nat. Acad. Sci. USA*, vol. 116, no. 36, pp. 18050–18059, Sep. 2019.

[23] A. A. Frolov, D. Husek, and D. A. Rachkovskij, "Time of searching for similar binary vectors in associative memory," *Cybern. Syst. Anal.*, vol. 42, no. 5, pp. 615–623, 2006.

[24] V. I. Gritsenko, D. A. Rachkovskij, A. A. Frolov, R. Gayler, D. Kleyko, and E. Osipov, "Neural distributed autoassociative memories: A survey," *Cybern. Comput. Eng.*, vol. 2, no. 188, pp. 5–35, 2017.

[25] E. P. Frady, D. Kleyko, and F. T. Sommer, "A theory of sequence indexing and working memory in recurrent neural networks," *Neural Comput.*, vol. 30, no. 6, pp. 1449–1513, 2018.