NANO LETTERS
Cite This: Nano Lett. 2018, 18, 4447−4453
Letter
pubs.acs.org/NanoLett

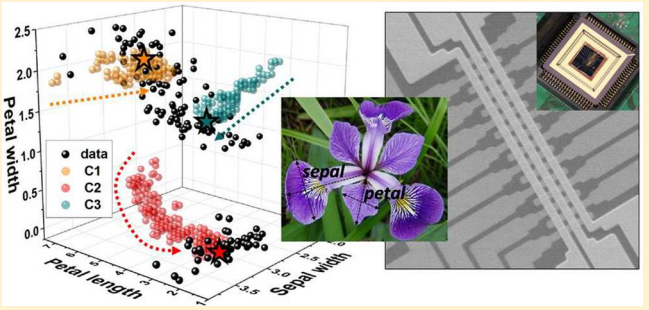
K-means Data Clustering with Memristor Networks

YeonJoo Jeong,[†,§] Jihang Lee,[†,‡,§] John Moon,[†] Jong Hoon Shin,[†] and Wei D. Lu*[,†]

[†]Department of Electrical Engineering and Computer Science and [‡]Department of Materials Science and Engineering, University of Michigan, Ann Arbor, Michigan 48109, United States

Supporting Information

ABSTRACT: Memristor-based neuromorphic networks have been actively studied as a promising candidate to overcome the von-Neumann bottleneck in future computing applications. Several recent studies have demonstrated memristor network's capability to perform supervised as well as unsupervised learning, where features inherent in the input are identified and analyzed by comparing with features stored in the memristor network. However, even though in some cases the stored feature vectors can be normalized so that the winning neurons can be directly found by the (input) vector−(stored) vector dot-products, in many other cases, normalization of the feature vectors is not trivial or practically feasible, and calculation of the actual Euclidean distance between the input vector and the stored vector is required. Here we report experimental implementation of memristor crossbar hardware systems that can allow direct comparison of the Euclidean distances without normalizing the weights. The experimental system enables unsupervised *K*-means clustering algorithm through online learning, and produces high classification accuracy (93.3%) for the standard IRIS data set. The approaches and devices can be used in other unsupervised learning systems, and significantly broaden the range of problems a memristor-based network can solve.



KEYWORDS: *Unsupervised learning, Euclidean distance, neuromorphic computing, analog switching, RRAM, Ta_2O_5*

Neuromorphic computing systems based on emerging devices such as memristors have attracted great interest, especially following recent advances of experimental demonstrations at the network-level aimed for practical tasks.[1−7] As a hardware system that offers co-location of memory and logic and highly parallel processing capability, memristor-based networks enable efficient implementation of machine learning algorithms.[8−12] Among the approaches, unsupervised learning is of growing importance since it relies on an unlabeled training data set,[13] which is far cheaper to obtain than those required by supervised learning algorithms.[14−16] During training, unsupervised learning rules rely on indicators of similarity between the input feature vectors and the learned feature vectors (dictionary elements), e.g., distance between these vectors in Euclid space, to identify the dictionary element that best matches the input and subsequently adjust the weights accordingly.[17−19] If the dictionary elements are normalized, finding the shortest Euclidean distance is equivalent to finding the smallest dot-product between the input vector and the dictionary element vector, which can be readily obtained in a memristor crossbar.[1,2,4] However, when the dictionary elements cannot be normalized during learning, finding the shortest Euclidean distance is no longer trivial, and can cause significant overhead in hardware implementation.

In this study, we propose and experimentally demonstrate an approach that directly compares the shortest Euclidean distance in a memristor crossbar hardware system, without normalizing the weights. As a test case, we use this approach to implement the *K*-means algorithm,[17] one of the most widely used unsupervised methods in cluster analysis,[20,21] experimentally in a memristor crossbar array. The network successfully performs cluster analysis through online training for different kinds of inputs regardless of the initial centroid locations. When used to analyze the standard IRIS data set,[22] a 93.3% classification accuracy was obtained experimentally in the hardware system based on a Ta_2O_{5-x} memristor array, comparable with software-based results in ideal conditions.

Mapping *K*-means onto Memristor Array Network. The *K*-means[17] algorithm aims to partition a set of vector inputs into *K* clusters through exploratory data analysis, as schematically shown in Figure 1a (for $K = 3$). From the random initial centroid locations, the network evolves as input data points are assigned to different clusters based on the distances between the input data point and the different centroid locations, followed by updating the centroid locations of the clusters. With a relatively simple form, *K*-means provides a comparable solution to more complex approaches such as autoencoders[23] for preclustering of unlabeled data set through online training, and reduces the original input space into disjoint smaller subspaces for subsequent use of fine clustering algorithms or data classification through another supervised layer.[24,25]

Received: April 16, 2018
Revised: May 29, 2018
Published: June 7, 2018

DOI: 10.1021/acs.nanolett.8b01526
Nano Lett. 2018, 18, 4447−4453

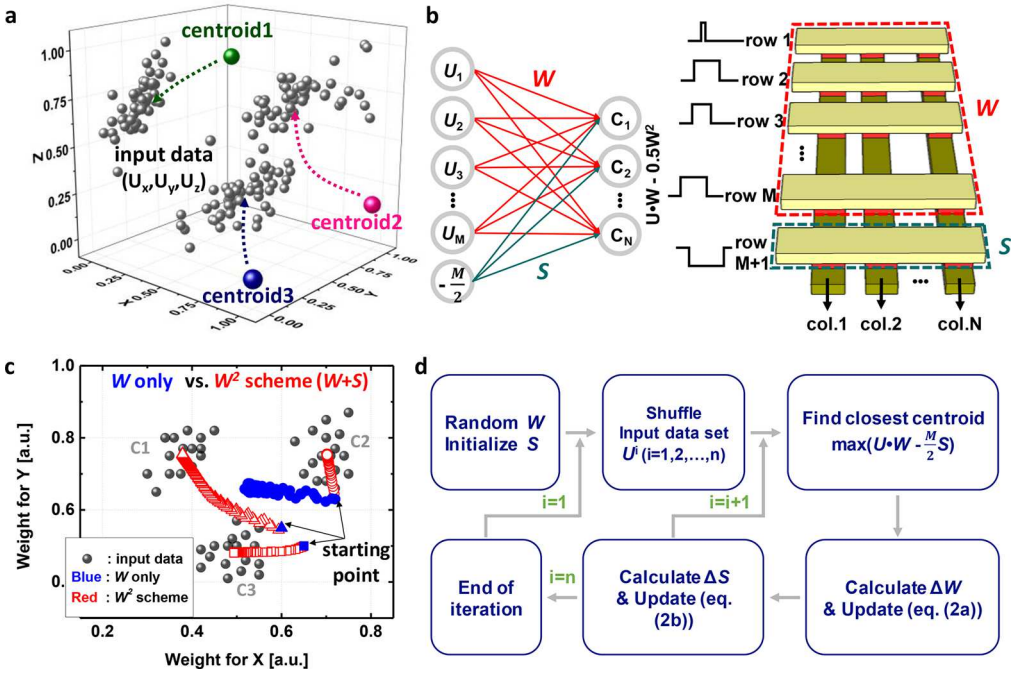


Figure 1. Experimental setup of *K*-means implementation. (a) Schematic of the *K*-means algorithm showing the evolution of the *K* centroid locations during online learning. (b) Mapping the proposed algorithm onto the memristor array. The coordinates of a centroid is stored as the memristor conductance values in the corresponding column in the *W*-matrix. The W^2 information for the centroid is stored by the memristor conductance in the *S* matrix in the same column. The input values are coded as pulses with different widths and are applied to the rows of the expanded matrix. The accumulated charges at the columns' outputs allow direct comparison of the Euclidean distances. (c) Simulation showing the proposed W^2 scheme can lead to correct centroid evolutions, whereas without the *S* matrix (blue circle symbol) only one centroid was trained. (d) Flowchart of the online learning algorithm. Both the search and update operations are implemented in memristor hardware.

However, there are two challenges for experimentally implementing algorithms such as *K*-means in memristor-based systems. The first challenge is obtaining the Euclidean distance directly in hardware. Memristor arrays can readily implement vector−vector dot-product operations.[1,2,26,27] However, when the weights are not normalized, the feature vector that produces the largest dot-product with the input vector is not necessarily the one having the shortest distance to the input, as will be discussed in detail below. The second challenge is related to the fact that *K*-means is not aimed at minimizing an error/cost. In previously demonstrated systems, the operation of the memristor network is to minimize a cost function based on the learning rule: either the output label error for supervised learning[1,3] or the cost function for representing the input in certain unsupervised learning cases.[2] As a result, the algorithm relies on a known reference (either the output label or the original input) to calculate the error/cost. The reference in turn provides a feedback mechanism that helps the network converge near the targeted solution, even in the presence of sizable device variations.[28,29] This feedback mechanism, however, is missing in many unsupervised learning and arithmetic operation processes, including the *K*-means approach, since such algorithms do not explicitly aim to minimize an error against a reference, and thus hardware demonstration will post more stringent requirements on the device performance.

Here we address these challenges and experimentally implement the *K*-means algorithm in memristor crossbar array-based hardware as a test case. In the implementation, locations of the *K* centroids are directly mapped as weights in the memristor array in column-wise fashion, i.e., W_{mn} at cross-point (m, n) in the memristor array corresponds to the *m*th coordinate value of the *n*th centroid. The weights are in turn represented by the state variable (w) in the device model (eqs S1 and S2 in the Supporting Information), which is linearly mapped onto the device conductance. With the input vector set *U* and a randomly generated initial weight matrix *W*, the *K*-means algorithm iteratively performs two successive operations to assign the inputs to the appropriate clusters. The first process, the search step, is to find the nearest centroid for a given input and assign the input to the associated cluster. It is then followed by the second process, the update (learning) step, which updates the selected centroid coordinates due to changes in the cluster composition. The crossbar structure as shown in Figure 1b is extremely efficient in implementing vector-matrix multiplication operation, which is one of the core operations[1−4] in neuromorphic systems, through simple Ohm's raw and Kirchhoff's law, i.e., the current (or charge) collected at each output neuron $(I_n = \sum_j V_j \cdot W_{jn})$ represents the dot-product of the input voltage vector and the stored conductance vector. In general, vector−vector dot-product operation provides a good indication of the similarity between the input vector and the stored vector, and thus can be viewed as performing a pattern-matching operation and is commonly used in machine-learning algorithms, particularly if the stored feature vectors can be readily normalized. However, algorithms such as *K*-means rely on finding the exact Euclidean distances, not just the similarity between the vectors, to decide the winning neurons and perform the updates.

The Euclidean distance of the input vector *U* and the weight vector W_n for the *n*th centroid is determined by eq 1

$$\| U - W_n \|^2 = U^2 - 2U \cdot W_n + W_n^2 \quad (1)$$

DOI: 10.1021/acs.nanolett.8b01526
Nano Lett. 2018, 18, 4447−4453

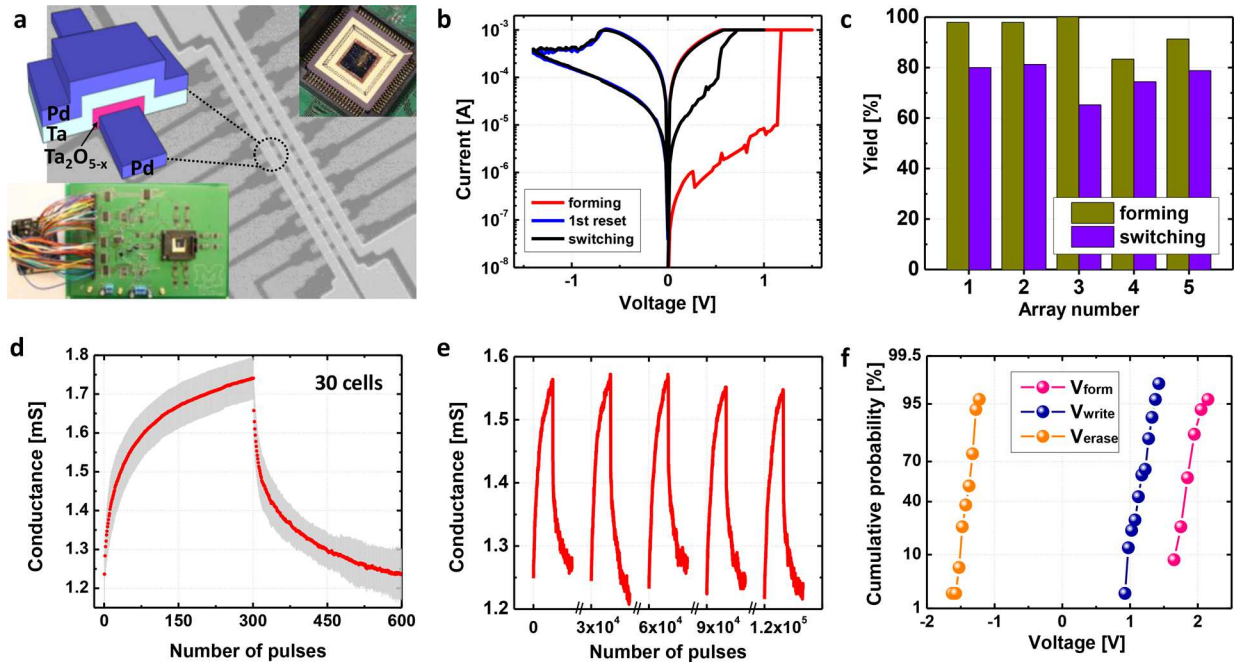


Figure 2. Ta_2O_{5-x} memristor crossbar characteristics. (a) Scanning electron micrograph (SEM) image of a fabricated crossbar array used in this study. Upper left inset: schematic of the device structure. Lower left inset: Photo of the test board. The memristor array is wire-bonded (upper right inset) and integrated into the board system. (b) DC $I-V$ curves during the forming cycle and the subsequent switching cycle. The low forming voltage is critical to enable all devices in the crossbar to be properly formed without damaging the devices that have been formed earlier. (c) Device yields from 5 different arrays, showing the percentage of successful forming and the percentage of fully functional devices after all devices in the crossbar have gone through the forming process. (d) Analog conductance update characteristics obtained from 30 devices in the array, showing reliable incremental conductance changes and tight device distribution. Red curve: average conductance during the measurement obtained from the 30 devices. The devices were programmed by 300 consecutive write pulses (1.15 ± 0.1 V, 1 μs), followed by 300 erase pulses (-1.4 ± 0.1 V, 1 μs). (e) Cycling curves showing reliable analog conductance updates can be maintained after 1.2×10^5 write/erase pulses. (f) Distribution of the forming, write and erase voltages, obtained from the 16×3 array. We used a 4×3 array having narrow voltage range in the K-means experiments.

Beside the dot-product term $U \cdot W_n$, two other terms, U^2 and W_n^2, are required to obtain the Euclidean distance $\|U - W_n\|$. The U^2 term depends on the input only and thus is a constant for all centroids and will not affect the comparison when determining the winning neuron. However, the $W_n^2 = \sum_j W_{jn}^2$ term, representing the L2 norm of the weight vector associated with centroid n, can in general be different for each centroid. Only in certain conditions will the weight vector be naturally normalized (e.g., when the weight update follows the Oja's rule[29]). Numerically normalizing the weight vectors is expensive, and if not performed, the dot-product $U \cdot W_n$ will not in general correctly represent the Euclidean distance due to differences in the W_n^2 term.

Below we show that Euclidean distance comparisons can still be obtained in memristor crossbar-based implementations. We note that if the array matrix is expanded to include one additional row representing the W^2 term (called the W^2 scheme approach for convenience), results from eq 1 can still be obtained in the memristor array during a single read operation using the same vector-matrix multiplication approach. Here, the new matrix consists of both the original W matrix (sized $M \times N$) and a new S matrix (sized $1 \times N$) (Figure 1b). The S matrix in this case is a single row and stores the average value of the squared weight $\left(\langle W_n^2 \rangle = \sum_j \left(\frac{W_{jn}^2}{M}\right)\right)$ for the nth centroid. The key is then to obtain the desired output using this expanded matrix and to allow the S matrix to be updated correctly during unsupervised online learning.

During Euclidean distance calculation, the input vector will now include the original input U (applied to the rows of the original W matrix), and an additional, constant element $-M/2$ that will be applied to the S matrix, as shown in Figure 1b. Here M is the number of rows in the W matrix. The choice of the input and the S matrix is to ensure the values of the elements in the S matrix are in the same range as the values in the W matrix, so that the same type of memristor devices can be used for the expanded matrix. Experimentally, the elements in the input vector are implemented with voltage pulses having a fixed amplitude (V_{read} = 0.3 V in our study) and variable widths proportional to the desired input values. The input voltage pulses are applied to the network to perform the vector-matrix multiplication, and the charge Q_n accumulated at each output will represent the distance between the original input vector U and the weight vector W_n since $Q_n = \sum_j \left(U_j \cdot W_{jn} - \frac{W_{jn}^2}{2}\right)$ differs from the exact expression (1) only by a $(-)$ sign and a constant U^2. As a result, the higher the output charge obtained from a particular column in the memristor array, the shorter the distance is between the input U and the centroid represented by the column.

After identifying the nearest centroid, the second step is to update both the weights associated with the centroid (representing the coordinates of the centroid location) in the original W matrix and the S matrix representing the $\langle W^2 \rangle$ term. We have developed a learning rule that can be readily implemented in the memristor hardware, shown in eq 2 and eqs S3–S6 in the Supporting Information.

DOI: 10.1021/acs.nanolett.8b01526
Nano Lett. 2018, 18, 4447–4453

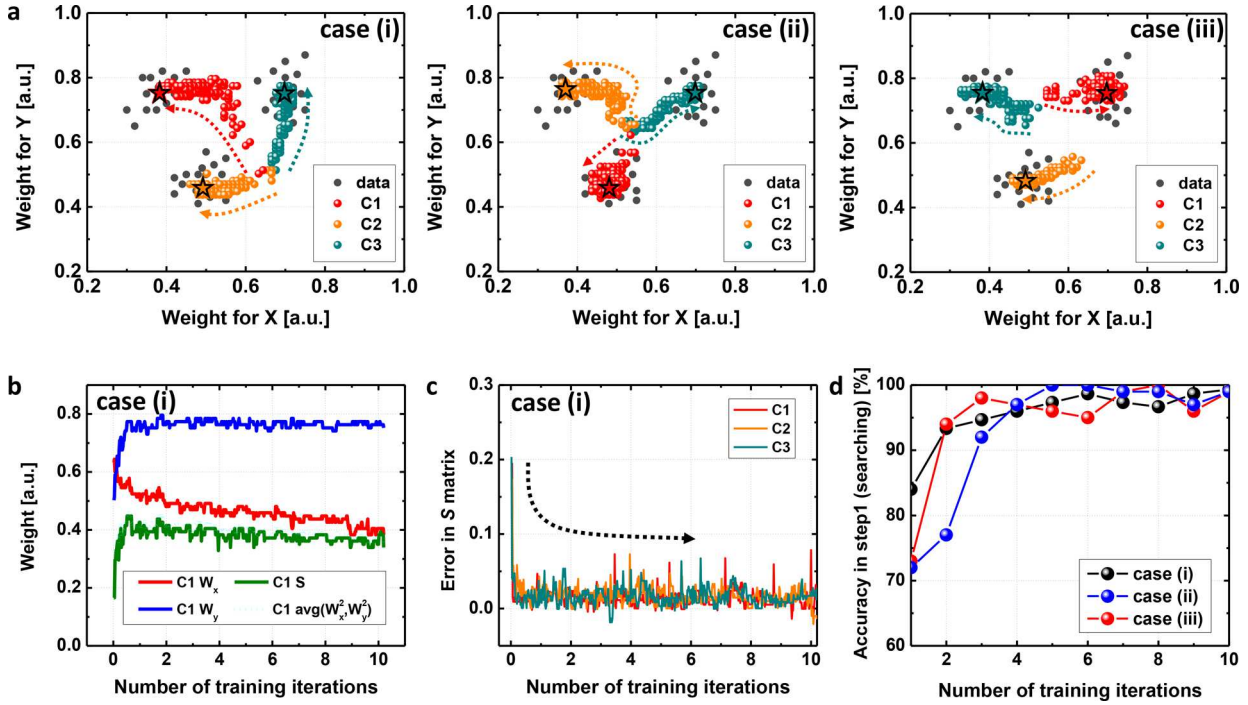


Figure 3. Experimental implementation of *K*-means using the memristor crossbar. (a) Evolution of the three centroid locations during training, for three cases with different initial configurations: (i) all three centroids were located at the same position outside the data set; (ii) all three centroids were located at the same position inside the data set; (iii) randomly assigned initial positions. The final locations of the centroids are represented by the stars. (b) Evolution of the *W* elements W_x, and W_y (representing the centroid coordinates) for centroid 1 in case I, along with the centroid's *S* element, and a reference showing the calculated $\langle W^2 \rangle$ during training. Each iteration includes 50 training operations for a given input data set. (c) Difference between the stored *S* element and the calculated $\langle W^2 \rangle$ for the three centroids, showing a rapid decrease after only a few training steps. (d) Success rate of correctly finding the nearest centroid using the memristor network, as a function of the iteration number during training. The success rate becomes >95% after 4 iterations.

$$\Delta W_n = \eta \cdot (\text{input} - W_n) \tag{2a}$$

$$\Delta S_n = \sum_j \left(\frac{W_{jn}^2}{M} \right) - S_n \tag{2b}$$

Here η is a constant and represents the learning rate for the *W* matrix. Note that an update is only performed for the column corresponding to the nearest centroid, and both the *W* matrix and the *S* matrix are updated simultaneously. By repeating the iterative training process with the search step and the update step as described in Figure 1d, the network stabilizes and learns the centroids of the *K* clusters.

To verify the proposed method, we first performed simulations based on a realistic device model (eqs S1 and S2 in the Supporting Information). As shown in Figure 1c, without the *S* matrix and simply comparing the dot-products between the input vector and the centroid weight vectors, the network could not correctly identify the nearest centroid, and the centroid that happens to have a high initial weight vector length by chance is always picked and updated due to its larger output of the dot-product. Two of the three centroids never got trained as a result. By contrast, with the same starting conditions, the proposed W^2 scheme correctly identifies the nearest centroid and properly updates the centroid locations, leading to successful clustering of the data after 30 iterations.

Note that from a mathematical point of view, the added *S* matrix can be considered as a bias term in neural networks. Implementation of the bias term in memristor networks has been reported previously[1,30] However, there are important differences. The *S* matrix has specific physical meaning, W^2, and requires a different training algorithm than the weight update, contrary to generic bias terms added to the network output. As such, direct comparison of the Euclidean distances can be obtained in our proposed approach, enabling the experimental implementations of algorithms such as *K*-means clustering in memristor arrays for the first time.

Analog Memristor Array Implementation. Experimental implementation is based on an optimized Ta_2O_{5-x} memristor structure having low forming voltage and reliable analog conductance modulations. A scanning electron microscopy (SEM) image of the as-fabricated 16 × 3 crossbar array is shown in Figure 2a, along with a schematic showing the device structure consisting of a resistive Ta_2O_{5-x} layer and a scavenging Ta layer (detailed fabrication methods can be found in Supporting Information). Before reliable switching, the as-fabricated devices generally need to undergo a forming step. However, in a passive crossbar, the high voltage used during forming of one device can damage the already-formed, half-selected devices sharing the same word-line or bit-line electrode, even with a protective scheme (Figure S4). A thin oxide layer can reduce the forming voltage, but can also reduce device yield due to stuck-at-1 (SA1) issues during switching (Figure S3a). We addressed this issue by using a thin oxide (3.5 nm) deposited with extremely low sputtering power and rate (30W, 1.1 Å/min). The low deposition rate allows better control of the oxide film quality and stoichiometry, and helps mitigating the SA1 problem and allows both low forming voltage and high array yield, as shown in Figure 2b,c (additional discussions are in the Supporting Information). In addition, a

DOI: 10.1021/acs.nanolett.8b01526
Nano Lett. 2018, 18, 4447−4453

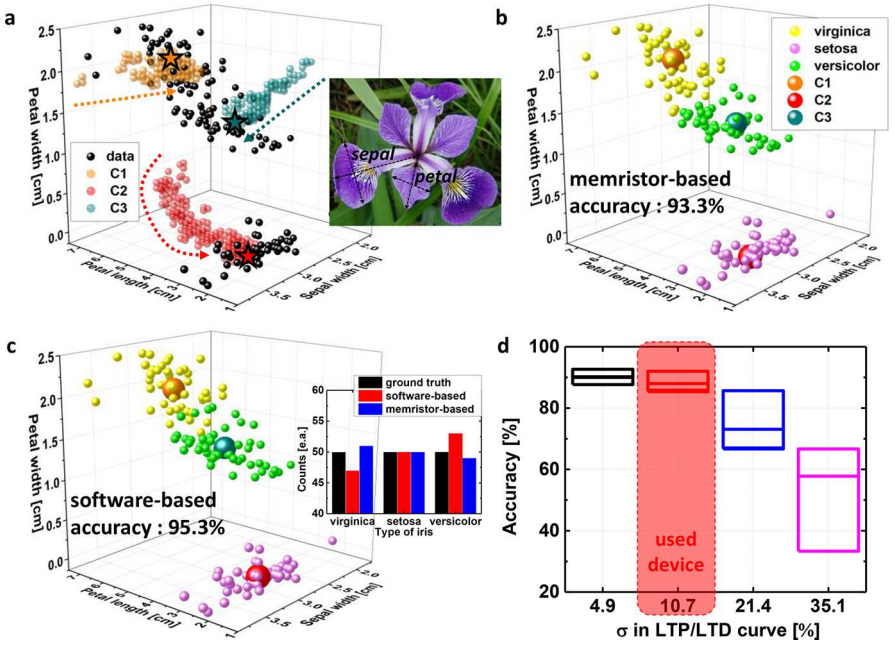


Figure 4. *K*-means analysis of the IRIS data set. (a) Evolution of the centroid locations during online training with the unlabeled 3D IRIS data set. Inset: features in the IRIS data set, including the length and the width of the sepal and the petal. In this work, only the three features (the sepal width, the petal width, and the petal length) that produce the highest accuracy were used. (b) Clustering results using the memristor-based network after training. The final positions of the three centroids are represented by large circles, and the data points that are partitioned into the three clusters depending on the nearest means are represented by small circles with different colors. (c) Results from *K*-means analysis obtained from software. Inset: Comparison of results obtained from software- and memristor-based methods for the 3 types of flowers, with reference to the ground truth. (d) Simulation results showing the effects on *K*-means analysis accuracy as a function of device variation. High device uniformity achieved in this study is critical to obtain the desired accuracy.

Ta electrode is used to form an oxygen-deficient layer above the Ta_2O_{5-x} switching layer. The Ta electrode was deposited under a low power (100W, 0.5 Å/s) condition to minimize the defect creation in the Ta_2O_{5-x} switching layer (e.g., due to Ta atom injection during deposition). These measures improved the quality of the Ta_2O_{5-x} layer, so that switching will be driven by electrically controlled oxygen vacancy exchange between the Ta_2O_{5-x} switching layer and the oxygen deficient layer adjacent to the Ta electrode, rather than less controllable intrinsic vacancies in the as-deposited oxide.[31,32]

With these optimizations, improved analog switching behaviors with high yield can be obtained from the memristor array. The long-term potentiation/depression (LTP/LTD) curves of 30 cells from a total of 600 consecutive write and erase pulses are shown in Figure 2d, highlighting gradual and uniform switching behavior (Figure S3). As shown in Figure 2e, reliable analog switching behaviors can be maintained after 1.2 × 10^5 programming cycles. Figure 2f shows the distribution of the forming, write and erase voltage values measured from the 16 × 3 array, showing tight voltage range (σ = 0.1 V) for V_{write} and V_{erase}. These devices were used for the *K*-means analysis.

Experimental Implementation of the *K*-means Algorithm. The *K*-means algorithm was experimentally implemented using the memristor array and a custom-built test board (Figure 2a and S4). The test board allows arbitrary pulse signals to be sent to and electronic current collected from either individual devices or multiple devices in multiple rows and columns simultaneously in parallel.

A simple two-dimensional (2D) data set was first used to test the system. Specifically, 50 2D data points that can be explicitly partitioned into three clusters, i.e., K = 3, were manually generated in the study to verify the operation of the memristor network. Figure 3a shows the evolution of the learned centroid positions obtained from the memristor-based *K*-means system, using online unsupervised training for three different initial weights conditions. During training, the closest centroid to an input is first determined from the memristor array using the W^2 approach, followed by updates of the W and S elements for the selected centroid based on eq 2. For example in case (i), the three centroids were initially placed at the same location, i.e., having nearly identical W vectors in the memristor matrix (with the small differences due to device variations among the different columns). During training, centroid 2 moved in the left direction toward one group, while centroid 3 moved in the up direction. Centroid 1 moved in the upper left direction with relatively large movements at the beginning of training, and turned left toward the group of data farthest from the initial point. After training, the three centroids settled to the centers of the respective clusters, and every point in the data set can be properly assigned to one of the three clusters. Other initial centroid locations, such as having all centroids located in the center of the data set (case (ii)), and having the centroids randomly distributed (case (iii)) have also been tested, and the memristor network can successfully perform *K*-means clustering in all cases (Figure 3a and Figure S5 in the Supporting Information) using our experimental setup, demonstrating the reliability of the proposed training algorithm and the hardware implementation using physical memristor crossbar arrays, even with non-normalized weights.

We note that the reliability of the *K*-means algorithm implementation depends critically on how accurately the W and S elements are modulated during training, since errors in these elements directly affect the obtained Euclidean distance and the subsequent identification of the winning centroid. The

DOI: 10.1021/acs.nanolett.8b01526
Nano Lett. 2018, 18, 4447−4453

experimentally obtained evolution of the W and S elements during training are shown in Figure 3b. One can see that as a W vector (e.g., W_x and W_y of centroid 1 in case (i) shown in Figure 3b) is updated when a new data point is added to the cluster, the corresponding S element is properly adjusted based on the proposed algorithm, and the learned S elements in the memristor network following the proposed eq 2b are indeed in good agreement with the calculated value of $\langle W^2 \rangle$. Specifically, Figure 3c shows the difference between the stored S values and the calculated $\langle W^2 \rangle$ value (numerically calculated from the measured values of the W matrix) in case (i). The error is large in the beginning because the S elements were not initialized according to the W matrix values. Importantly, the error drops rapidly with only a few training steps and remains very low, due to the excellent incremental conductance modulation capability of the physical device as shown in Figure 2d. As a result, the system can correctly find the nearest centroid with over 95% success rate after 4 iterations as shown in Figure 3d, verifying that the proposed W^2 scheme can be used to efficiently calculate distances between vectors without compute-intensive normalization processes.

Analysis of the IRIS Data Set. Based on the successful analysis of the test data set, we applied the memristor-based hardware system to perform K-means analysis of the IRIS flower data set,[22] a widely used data set in machine learning. The IRIS data set includes data from four features such as the length and the width of the sepal and the petal (inset of Figure 4a), measured from 150 samples from each of the three iris flower species: *setosa*, *virginica*, and *versicolor*. The goal is to successfully separate the three species based on these measured data. Figure 4a shows the evolution of the three centroids (representing the three species) obtained from the memristor system during training. Following standard practice, only three features (the sepal width, the petal width, and the petal length) that produce the highest classification accuracy were used in the analysis. Even though the boundary between *virginica* and *versicolor* is inherently complex in the IRIS data set (Figure S6a), the three centroids were updated properly in the memristor-system with initially randomized W and S matrices, and led to a final configuration that enabled proper clustering of the unlabeled data. The final cluster analysis obtained from the trained memristor-based network is shown in Figure 4b and S6, corresponding to a classification accuracy of 93.3%. This experimental result is comparable to the result (95.3%) obtained from a software-based method (Figure 4c), demonstrating the feasibility of the experimental memristor network system with the proposed W^2 scheme for data-intensive cluster analysis.

Since K-means analysis is based solely on the Euclidean distances between the input and the dictionary vectors, and does not rely on minimizing an output label error or a cost function which can provide a feedback mechanism to help network stabilization,[1−3] more accurate vector-matrix multiplication and weight modulation operations are required. In this case, effects such as cycle-to-cycle and device-to-device variations can significantly affect the accuracy of the clustering analysis during experimental implementation. Importantly, the memristor devices used in this work with improved switching uniformity has a device variation of ∼10% (characterized by the standard deviation during LTP/LTD measurements), as shown in Figure 2d, which enabled us to achieve the high clustering accuracy observed experimentally. To verify the effect of device variations, detailed simulations that incorporate device variation effects (Figure S1) were performed. The accuracy of K-means analysis was found to significantly degrade when device-to-device variation is larger than 20% (Figure 4d and S1), confirming our hypothesis and experimental findings. The high clustering accuracy obtained experimentally demonstrates the potential of memristor-based networks for neuromorphic applications as well as arithmetic applications that require high accuracy.

Finally, we note that exact Euclidean distances are not obtained in our implementation due to the missing U^2 term in the output. If the complete Euclidean distances are needed,[21] more changes to the implementations are required. One possible approach based on the general idea proposed in this study, which can calculate the (approximate) complete Euclidean distances is discussed in Figure S8 in the Supporting Information. By extending the simple dot-product operations between the input and the weights using bias-like terms as discussed in this study, more complex and elaborate algorithms may also be implemented.

Conclusions. In this work, we experimentally demonstrated memristor-based neural networks for K-means clustering analysis. A W^2 scheme was proposed to allow accurate calculation of the Euclidean distance, which is an essential operation in many machine learning algorithms, through direct vector-matrix multiplication in memristor networks with non-normalized weights. Without a cost function that provides a feedback mechanism, the K-means algorithm poses stricter requirements on compute accuracy and device variability compared with other neuromorphic computing algorithms. With an expanded weight matrix, properly designed training rules, and improved device properties, we show K-means clustering can be reliably implemented using memristor networks. The standard IRIS data set was successfully processed through unsupervised, online learning with high accuracy (93.3%). With continued device and algorithm optimizations, the results obtained here can pave the way toward practical memristor network hardware for more broad applications beyond neuromorphic systems.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acs.nanolett.8b01526.

Device model, updating rules, training the memristor array, device fabrication, additional RS characteristics of the memristor, electrical forming process, array measurements setup using a test board, results of K-means analysis on 2D data, results of K-means analysis on the IRIS data set, device linearity data, and possible approach to obtain the approximate complete Euclidean distance (PDF)

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: wluee@eecs.umich.edu.

ORCID

Wei D. Lu: 0000-0003-4731-1976

Author Contributions

§Y.J.J. and J.L. contributed equally. The manuscript was written through contributions of all authors. All authors have given approval to the final version of the manuscript.

DOI: 10.1021/acs.nanolett.8b01526
Nano Lett. 2018, 18, 4447−4453

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

The authors would like to thank Dr. M. Zidan and F. Cai for helpful discussions. This work was supported in part by the National Science Foundation (NSF) through grants ECCS-1708700 and CCF-1617315.

■ REFERENCES

(1) Prezioso, M.; Merrikh-Bayat, F.; Hoskins, B. D.; Adam, G. C.; Likharev, K. K.; Strukov, D. B. *Nature* **2015**, *521* (7550), 61−64.
(2) Sheridan, P. M.; Cai, F.; Du, C.; Ma, W.; Zhang, Z.; Lu, W. D. *Nat. Nanotechnol.* **2017**, *12* (8), 784−789.
(3) Yao, P.; Wu, H.; Gao, B.; Eryilmaz, S. B.; Huang, X.; Zhang, W.; Zhang, Q.; Deng, N.; Shi, L.; Wong, H.-S. P.; Qian, H. *Nat. Commun.* **2017**, *8*, 15199.
(4) Choi, S.; Shin, J. H.; Lee, J.; Sheridan, P.; Lu, W. D. *Nano Lett.* **2017**, *17* (5), 3113−3118.
(5) Strukov, D. B.; Snider, G. S.; Stewart, D. R.; Williams, R. S. *Nature* **2008**, *453* (7191), 80−83.
(6) Waser, R.; Aono, M. *Nat. Mater.* **2007**, *6* (11), 833−840.
(7) Jo, S. H.; Chang, T.; Ebong, I.; Bhadviya, B. B.; Mazumder, P.; Lu, W. *Nano Lett.* **2010**, *10* (4), 1297−1301.
(8) Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; Hassabis, D. *Nature* **2016**, *529* (7587), 484−489.
(9) Merolla, P. A.; Arthur, J. V.; Alvarez-Icaza, R.; Cassidy, A. S.; Sawada, J.; Akopyan, F.; Jackson, B. L.; Imam, N.; Guo, C.; Nakamura, Y.; Brezzo, B.; Vo, I.; Esser, S. K.; Appuswamy, R.; Taba, B.; Amir, A.; Flickner, M. D.; Risk, W. P.; Manohar, R.; Modha, D. S. *Science* **2014**, *345* (6197), 668−673.
(10) Indiveri, G.; Liu, S.-C. *Proc. IEEE* **2015**, *103* (8), 1379−1397.
(11) Li, C.; Hu, M.; Li, Y.; Jiang, H.; Ge, N.; Montgomery, E.; Zhang, J.; Song, W.; Davila, N.; Graves, C. E.; Li, Z.; Strachan, J. P.; Lin, P.; Wang, Z.; Barnell, M.; Wu, Q.; Williams, R. S.; Yang, J. J.; Xia, Q. *Nat. Electron.* **2018**, *1* (1), 52−59.
(12) Wang, Z.; Joshi, S.; Savel'ev, S.; Song, W.; Midya, R.; Li, Y.; Rao, M.; Yan, P.; Asapu, S.; Zhuo, Y.; Jiang, H.; Lin, P.; Li, C.; Yoon, J. H.; Upadhyay, N. K.; Zhang, J.; Hu, M.; Strachan, J. P.; Barnell, M.; Wu, Q.; Wu, H.; Williams, R. S.; Xia, Q.; Yang, J. J. *Nat. Electron.* **2018**, *1* (2), 137−145.
(13) Hinton, G. E.; Sejnowski, T. J. Unsupervised learning: Foundations of neural computation.*Computers & Mathematics with Applications*; MIT Press: Cambridge, MA, 199938
(14) Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems* **2014**, *3*, 2672−2680.
(15) Hinton, G. E.; Osindero, S.; Teh, Y.-W. *Neural Comput.* **2006**, *18* (7), 1527−1554.
(16) Hinton, G. E. *Science* **2006**, *313* (5786), 504−507.
(17) MacQueen, J. *Proc. 5th Berkeley Symp.* **1967**, *1*, 281−297.
(18) Kohonen, T. *Neurocomputing* **1998**, *21* (1−3), 1−6.
(19) Filippone, M.; Camastra, F.; Masulli, F.; Rovetta, S. *Pattern Recogn.* **2008**, *41* (1), 176−190.
(20) Jiang, Y.; Kang, J.; Wang, X. *Sci. Rep.* **2017**, *7*, 45233.
(21) Jain, A. K.; Murty, M. N.; Flynn, P. J. *ACM Comput. Surv.* **1999**, *31* (3), 264−323.
(22) Bache, K.; Lichman, M. UCI Machine Learning Repository; University of California, Irvine, School of Information and Computer Sciences, 2013; URL: http://archive.ics.uci.edu/ml.
(23) Bengio, Y. *Foundation and Trends in Machine Learning* **2009**, *2* (1), 1−127.
(24) Lin, D.; Wu, X.*Meeting of the Association for Computational Linguistics*, Morristown, NJ, USA, 2009; Vol. *2*, p 1030.
(25) Coates, A.; Ng, A. Y. In *Neural Networks: Tricks of the Trade*; Springer: Berlin, Heidelberg, 2012; LNCS 7700, pp 561−580.
(26) Yang, J. J.; Strukov, D. B.; Stewart, D. R. *Nat. Nanotechnol.* **2013**, *8* (1), 13−24.
(27) Merced-Grafals, E. J.; Davila, N.; Ge, N.; Williams, R. S.; Strachan, J. P. *Nanotechnology* **2016**, *27* (36), 365202.
(28) Yu, S.; Gao, B.; Fang, Z.; Yu, H.; Kang, J.; Wong, H.-S. P. *Adv. Mater.* **2013**, *25* (12), 1774−1779.
(29) Sheridan, P. M.; Du, C.; Lu, W. D. *IEEE T. Neur. Net. and Lear* **2016**, *27* (11), 2327−2336.
(30) Bayat, F. M.; Prezioso, M.; Chakrabarti, B.; Kataeva, I.; Strukov, D. *arXiv preprint arXiv*:1712.01253 (2017).
(31) Baek, G. H.; Lee, A. R.; Kim, T. Y.; Im, H. S.; Hong, J. P. *Appl. Phys. Lett.* **2016**, *109* (14), 143502.
(32) Gao, B.; Chen, B.; Zhang, F.; Liu, L.; Liu, X.; Kang, J.; Yu, H.; Yu, B. *IEEE Trans. Electron Devices* **2013**, *60* (4), 1379−1383.

DOI: 10.1021/acs.nanolett.8b01526
Nano Lett. 2018, 18, 4447−4453