VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition

Mohsen Imani‡, Deqian Kong‡, Abbas Rahimi*, and Tajana Rosing‡
‡Computer Science and Engineering, UC San Diego, La Jolla, CA 92093, USA
*Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, CA 94720, USA
{moimani, dekong, tajana}@ucsd.edu, abbas@eecs.berkeley.edu

***Abstract*—In this paper, we propose VoiceHD, a novel speech recognition technique based on brain-inspired hyperdimensional (HD) computing. VoiceHD maps preprocessed voice signals in the frequency domain to random hypervectors and combines them to compute a hypervector (as *learned* patterns) representing each class. During inference, VoiceHD similarly computes a query hypervector; the classification task is done by checking the similarity of the query hypervector with all learned hypervectors and finding a class with the highest similarity. We further extend VoiceHD to VoiceHD+NN that uses a neural network with a single small hidden layer to improve the similarity measures. This neural network is a small block directly operating on the similarity outputs of VoiceHD to slightly improve the classification accuracy. We evaluate efficiency of VoiceHD and VoiceHD+NN compared to a deep neural network with three large hidden layers over Isolet spoken letter dataset. Our benchmarking results on CPU show that VoiceHD and VoiceHD+NN provide 11.9× and 8.5× higher energy efficiency, 5.3× and 4.0× faster testing time, and 4.6× and 2.9× faster training time compared to the deep neural network, while providing marginally better classification accuracy.**

I. Introduction

The majority of Internet of Things (IoT) devices today tend to think of speech recognition being within the cloud. This is mainly because of limited resources and battery capacity of these small devices which make them unable to run complex speech recognition algorithms. To enable efficient *on-device* learning major algorithmic and architectural improvements are required.

Neural Networks (NN), in particular deep NN, have been widely used for speech recognition [1], [2]. However, such deep networks are computationally expensive especially in the training mode with iterative gradient decent. Among brain-inspired computing paradigms, hyperdimensional (HD) computing [3] is founded on the mathematical properties of high-dimensional spaces and offers a promising new avenue to enable efficient on-chip learning [4], [5]. HD computing builds based on a well-defined set of arithmetic operations and provides a complete computational paradigm for learning problems. Examples include analogy-based reasoning [6], language recognition [7], [8], [9], text classification [10], spoken word classification [11], biosignal electromyography processing [12], brain-computer interfaces [13], and prediction from multimodal sensor fusion [14], [15]. In this work, we further extend the application of HD computing for fast and efficient speech recognition.

We propose VoiceHD, a hardware-friendly and efficient speech recognition technique using HD computing. VoiceHD maps input voice signals in the frequency domain to hypervectors. It assigns each frequency bin to a channel with a unique identification that is mapped to a random hypervector. Depending on distribution of values over the frequency bins, an encoder generates a prototype hypervector representing the class of interest. At the end of training mode, a set of such prototype hypervectors is produced and will be stored as *learned* patterns. VoiceHD uses the same encoder during the test mode: it generates a query hypervector from unseen input voice and then classifies it to one of the learned patterns with highest similarity.

We further extend VoiceHD to VoiceHD+NN that uses a neural network with a single small hidden layer to improve the resolution of similarity measures. We evaluate efficiency of VoiceHD and VoiceHD+NN over Isolet [16], a popular speech recognition dataset which targets classifying the spoken letter of 150 persons among 26 English alphabets. Our results show that during test on CPU, VoiceHD and VoiceHD+NN provide 11.9× and 8.5× higher energy efficiency, and 5.3× and 4.0× faster execution compared to a pure deep neural network with three large hidden layers, while providing marginally better accuracy. Moreover, VoiceHD and VoiceHD+NN are trained 4.6× and 2.9× faster than the deep neural network on CPU.

II. Background and Related Work

A. Speech Recognition

Neural networks (NNs) combine with hidden Markov models have a long history in speech recognition [17], [18]. Deep neural networks have also been recently used for highly accurate speech recognition [1], [2]. Prior work tried to accelerate NNs on GPUs [19], [20], FPGAs [21], [22], and ASICs [23], [24]. However, NNs are still expensive in train due to costly iterative back-propagation using gradient descent. Moreover, NN inference works based on the costly matrix multiplication, which requires the floating point precision for accurate recognition.

Although the accuracy is an important factor, the energy efficiency of classifier is becoming more important by going toward IoT domain. It is crucial to have a low-cost classification technique which can efficiently train and test on embedded devices, while providing good accuracy. This local data processing: (i) improves the computation efficiency by

978-1-5386-1553-9/17/$31.00 ©2017 IEEE

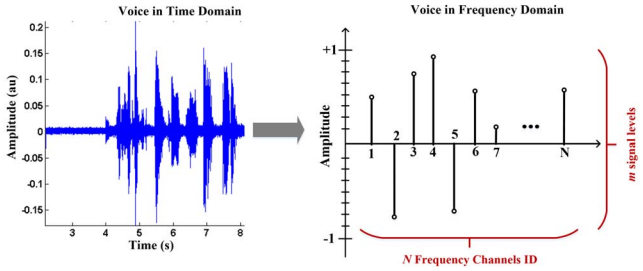


Fig. 1. Example of voice signal in time and frequency domain.

addressing data movement issue, (ii) provides higher network bandwidth and (iii) makes the computation more secure by avoiding to send the data to cloud to process. Thus, in this work we propose a novel classification technique with HD computing, which supports one-shot training, along with fast and efficient test operation. Our design uses basic linear algebra and simple Hamming distance operations for efficient classification.

B. HD Computing

HD computing operates with high-dimensional vectors, aka hypervectors. Hypervectors are holographic and (pseudo)random with i.i.d. components. A hypervector contains all the information combined and spread across all its components in a full holistic representation so that no component is more responsible to store any piece of information than another. These unique features make a hypervector robust against errors in its components. Hypervectors can be manipulated with arithmetic operations, such as *binding* that forms a new hypervector which associates two hypervectors, and *bundling* that combines several hypervectors into a single composite hypervector. The reasoning in HD computing is based on similarity between the hypervectors. This similarity is measured by a distance metric.

HD computing is able to operate with analog inputs as well, examples include biosignal electromyography processing for hand gesture recognition [12] and electroencephalogram classification for brain-computer interfaces [13]. These applications use HD computing to encode spatial and temporal analog signals. In contrast, in this paper we focus on mapping voice signals in frequency domain into HD space for speech recognition.

III. Proposed VoiceHD

In this section, we propose VoiceHD, a novel design for energy-efficient and fast speech recognition. VoiceHD encodes voices to HD space and then classifies them by the similarity check. VoiceHD has two main blocks: an encoder and an associative memory. In train mode, VoiceHD only uses the encoder and write into associative memory, whereas the test mode uses both encoder and associative memory.

A. VoiceHD Encoder

In this section, we describe the proposed encoding scheme which maps input data to HD space. We focus on the Isolet dataset [16] with the goal of recognizing the voices among 26 letter of English alphabets. There are well-defined pre-processing steps for voice such as Mel-frequency cepstral coefficients (MFCCs) [25], which extracts and maps a raw voice information into the frequency domain. Figure 1 shows a sample of voice signal from Isolet dataset, with 617 input frequency bins. The signal amplitude in each frequency bin varies from -1 to +1. The goal of VoiceHD is to encode this voice signal in the frequency domain to a single hypervector with D dimensions. The encoder outputs a voice hypervector with D binary (0,1) components.

To this end, our encoding considers the impact of each frequency bin and its signal amplitude on the final voice hypervector. Our encoder assigns a unique channel ID to each frequency bin by generating a random hypervector called ID hypervector. Hence, two frequency bins will have orthogonal ID hypervectors: $\delta(ID_i,\ ID_j) > 5000$ for $D = 10,000$ and $i \neq j$; where the δ measures the Hamming distance between the hypervectors. For example, in Isolet dataset each input voice has 617 input frequency bins, therefore requires $N = 617$ IDs, i.e., $ID_1, \ldots, ID_{617}$.

As Figure 2 shows, in each input channel the signal amplitude changes from -1 to +1. For mapping this continuous range of values, we use the concept of continuous item memory proposed in [12], [13]. The continuous item memory provides fast and efficient lookup actions to map a continuous scalar value to hypervectors. Its implementation is indeed hardware-friendly when the range of scalar values is known; otherwise more general approach, e.g., in [11] can be used to map even a continuous-valued multivariate input to hypervectors by at higher hardware costs.

The continuous item memory linearly divides the full range of amplitude values ([-1,+1] to M levels and assigns a mixture of correlated and uncorrelated hypervectors to each level (L_1, L_2 ,..., L_{M-1}, L_M], where L_i shows the hypervector of level i^{th}. In contrast to hypervectors of IDs, there is a correlation between the hypervectors of neighbour levels. For example, L_1 has a high positive correlation with L_2, almost no correlation with $L_{M/2}$ (i.e., quasi-orthogonal), and high negative correlation with L_M. To this end, we first randomly generate L_1 hypervector with D dimension for the fist level. Then, we then flipped $D/(M-1)$ bits of L_1 to generate another hypervector for the next level (i.e, L_2). This procedure continues until generating L_M hypervector from L_{M-1} hypervector.

To encode the voice signal, our encoder (shown in Figure 2) looks at each frequency bin and multiplies its channel ID hypervector to its corresponding level hypervector. The level hypervector is chosen among the M hypervectors based on the amplitude of the signal in the frequency bin. The encoder applies these multiplications over all N frequency bins and then bundle these bound hypervectors together using the bundling operation. The following equation shows how the N channel IDs and corresponding level hypervectors generate a single voice hypervector for a given voice signal example in i^{th} class (S^i):

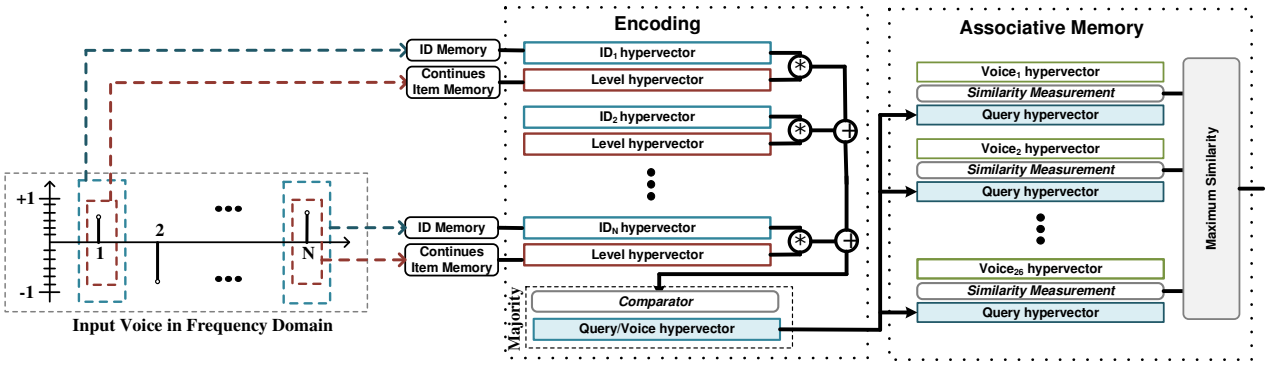


Fig. 2. The overview of VoiceHD architecture.

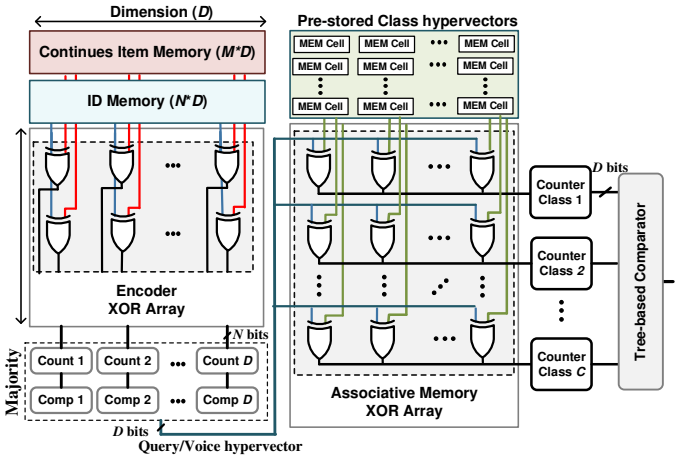


Fig. 3. Simple digital hardware for VoiceHD.

$$S_1^i = [L_m * ID_1 + \ldots + L_m * ID_N] \quad m \in [1, M]$$

We should note that the binding operation is a component-wise multiplication (e.g., XOR), while bundling operation is a component-wise *majority* function and is denoted as $[hypervector]$. The *majority* function counts the number of 1's in each dimension of hypervector, and outputs a 1 in that dimension if more than half of elements are 1. Similarly, the output value will set to 0, if more than half of elements are 0. Next, we generate a hypervector as the class prototype by bundling all voice hypervectors which belong to the same class. Our encoding uses the same *majority* function to bundle the voice hypervectors to a class hypervector. The following equation shows how K voice hypervectors are bundled to generate i^{th} class hypervector (C_i):

$$C_i = [S_1^i + S_2^i + \ldots + S_K^i]$$

B. Associative Memory

A class hypervector can be written to a row of associative memory as the learned patterns (see Figure 2). Once the last class is learned, it will be the end of training mode. In test mode, an unseen input voice is encoded to a hypervector using the same encoder for training. This query hypervector is compared to all stored hypervectors in the associative memory and a class with highest similarity will be returned.

Algorithm 1 Proposed VoiceHD encoding and associative memory

1: ***VoiceHD Encoding:***
2: **for** $i = 1 \ldots Alphabet$ **do**
3: **for** $k = 1 \ldots N$ **do**
4: $S_k^i \leftarrow [S_{k-1}^i + hv_k^i * ID_k^i]$
5: **end for**
6: $C_i \leftarrow [C_i + S^i]$
7: ***end for*** *//Save trained model* $(C_1, \ldots, C_{Alphabet})$
8: ***VoiceHD Associative Memory:***
9: ***for*** $I_j = I_1 \ldots I_{test}$ ***do***
10: ***for*** $i = 1 \ldots Alphabet$ ***do***
11: $d_i \leftarrow Distance(C_i, I_j)$ *//similarity check to all classes*
12: ***end for***
13: $out_j \leftarrow min(d_1, \ldots, d_{Alphabet})$ *//class with max similarity*
14: ***end for***

Algorithm 1 shows the required steps of VoiceHD in the encoder and the associative memory. After assigning hypervectors to N channels and M vector levels, the encoding starts multiplying the ID hypervector with the corresponding level hypervector in the channel. Encoder combines these bounded hypervectors by applying the majority function to generate the voice hypervectors. The algorithm applies the same *majority* function to all generated voice hypervectors in each class to get a hypervector of the corresponding class (line 6). The classification applies over the testing dataset, by similarly mapping the input signals to HD space and calculating the distance function between the query and the class hypervectors (line 11). Finally, the classification is done by looking for a class with the maximum similarity (line 13).

C. Digital Hardware Design for VoiceHD

In this section, we propose a simple digital hardware for VoiceHD. We aim to illustrate its plausibility for an efficient dedicated hardware implementation by describing the operations the are requires in details. Our hardware design is consist of the encoder and the associative memory as shown in Figure 3. Our design uses two memory blocks: ID memory to

store N channel ID hypervectors, and continuous item memory to store M level hypervectors. The encoder fetches the corresponding hypervectors from each of these memories. These hypervectors are then multiplied together. As Figure 3 shows, the multiplication between IDs and Ls is performed using XOR gate array. To bundle these hypervectors across all the channels, the majority function is applied using counter and comparator in each dimension. The counter accumulates the number of 1s in each dimension, and a comparator compares the result with a threshold value. For majority function, this threshold sets to be the half of the hypervectors summing up together.

During the testing, the input signal is passed through the same encoder and is mapped to a query hypervector. The query hypervector is then compared with all stored class hypervectors using an XOR array. The XOR array measures the Hamming distance as similarity metric between the query hypervector to all classes hypervectors. For every dimension of hypervector, the XOR array sets a one in that dimension in case of mismatch between the query and stored hypervector, otherwise the XOR array sets a zero. The number of 1s at each XOR row shows the Hamming distance of the query hypervector to stored class. At the second stage, a counter adds the number of mismatches at each row of XOR array. Finally, a comparator block, implemented in a tree structure, finds a class which has the minimum Hamming distance to query hypervector.

D. Parameters

We tested the proposed VoiceHD design over Isolet dataset, where 150 subjects spoke the name of each letter of the alphabet twice.

Figure 4 shows the impact of splitting the voice signal to different number of levels, M, on VoiceHD classification accuracy. Our evaluation shows that the M should be large enough to differentiate the patterns of voice amplitudes. In terms of energy consumption, large M increases the size of continuous item memory and results in slightly higher encoding energy. Our result shows that using $M = 10$ results in the maximum accuracy of 88.4%. Note that using large M than 10 will degrades the encoding efficiency with no impact on the classification accuracy.

E. Retraining: Associative Memory Updates

VoiceHD has 26 class hypervectors each representing one letter of English alphabet. In the training mode, each output class is generated by combining 2,300 voice signals. Adding large number of vectors to a single hypervector results in a capacity issue and loss of accuracy. For instance, the proposed design by adding all vectors together results in 88% classification accuracy. In order to avoid this issue, we *retrain* the associative memory by incremental updates that improves the classification accuracy. The goal of such retraining is to reduce the misclassification rate of VoiceHD by modifying the trained class hypervectors. After training the VoiceHD, our design tests the classification accuracy over the training dataset. For any trained data which classifies wrongly, our design modifies the class hypervectors to avoid such misclassification in future. The changes are categorized in two: (i) subtracting the query hypervector from the class hypervector which it wrongly classified with, and (ii) adding the query hypervector to an actual class that signal belongs to it. This update by addition and subtraction results in generating more distinguishable class hypervectors.

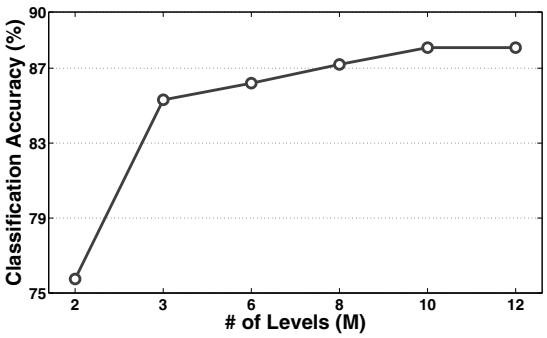


Fig. 4. The impact of the number of levels (M) on the recognition accuracy of the proposed encoding.

Considering an example that the query hypervector (Q) belongs to j^{th} class (C_j), but it wrongly matches with the hypervector of i^{th} class (C_i). Our retraining method makes the following modifications to the class hypervectors:

$$C_i = C_i - Q$$

$$C_j = C_j + Q$$

These updates continues over all the training examples only for one iteration. Our evaluation shows that the proposed associative memory update significantly improves the classification accuracy to 93.8% which is comparable with the state-of-the-art learning techniques to classify the same dataset. For example, the K-nearest neighbor (KNN) algorithm using Euclidean distance achieves 91.4% accuracy and a multi-task deep NN (with 48 hidden layers) provides 95.9% accuracy [26], [27], [28]. The size of this deep NN is indeed large compared to the VoiceHD using a single stage of the encoder and associative memory. Further, our proposed VoiceHD has the following advantages over the prior classification techniques:

- In contrast to the most of learning algorithms which require iterative training, VoiceHD supports fast and efficient one-shot learning (i.e., it requires few examples to learn).
- HD computing requires bit-wise computations over hypervectors in both train and test modes. These operations are fast and efficient as compared to floating-point operations that state-of-the-art classification algorithms use.

IV. VoiceHD+NN: VoiceHD Plus a Simple NN

Our observation over VoiceHD shows that it works well in getting the general information from the input voice signals

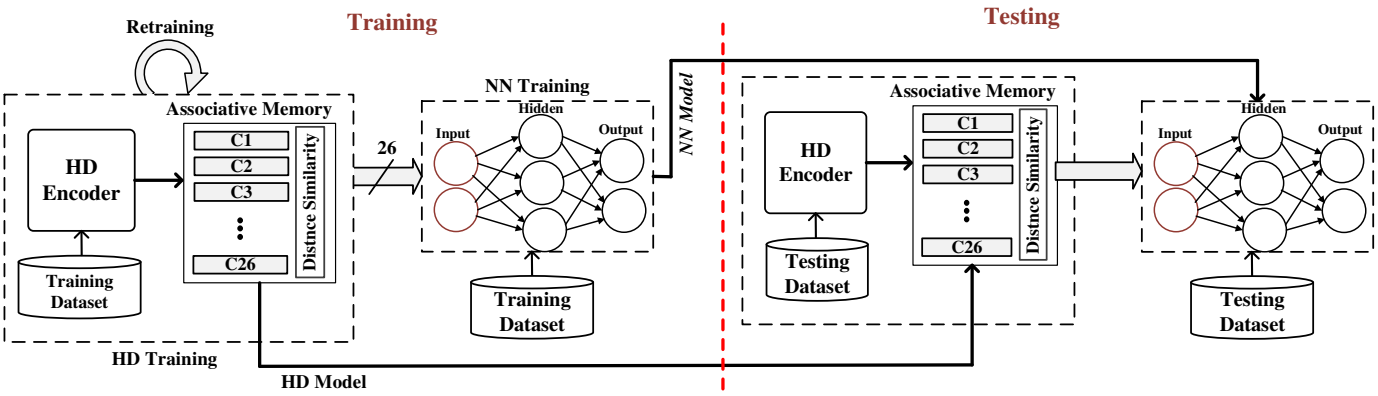


Fig. 5. VoiceHD+NN flow for training and testing.

with substantially higher efficiency. For example, over the Isolet dataset, the HD can properly categorize most of cases in the letter of alphabet. However, it is hard for VoiceHD with single stage to recognize the hard tasks, e.g., distinguishing T from C, or M from N. To further improve the VoiceHD classification accuracy, in this section we propose VoiceHD+NN to combine VoiceHD with a simple NN stage at the end. To do so, we train the NN block that works on the VoiceHD similarity outputs. This NN accepts 26 output classes of VoiceHD as inputs (layer with 26 neurons) and uses a hidden layer (with 50 neurons) and an output layer (with another 26 neurons) to improve classification.

A. VoiceHD+NN Flow

Figure 5 shows VoiceHD+NN flow for training and testing. The VoiceHD+NN training has three steps. First, we train VoiceHD by single time passing through the training dataset. The output of this training is 26 class hypervectors, each representing a class of voice signal. In the second step, we retrain VoiceHD as we earlier described by passing through the training dataset once again. Finally, we use this retrained model to adapt NN weights. NN training is done by using gradient descent. However, due to the small network size this training is relatively fast as compared to training large and deep NN. During the test mode, VoiceHD uses the same encoding block as the training phase to compute a query hypervector. The query hypervector is compared with all the class hypervectors in the associative memory. Based on the similarity measurement in the associative memory, each class will get a Hamming distance between 0 and D. Finally, NN with the trained weights operates on these distance values over all the classes to decide about the best output class. Coupling VoiceHD with this NN stage improves its accuracy from 93.8% to 95.3% as shown in Figure 6(a).

V. Evaluations and Experimental Results

A. Experimental Setup

We first describe the functionality of the proposed VoiceHD and VoiceHD+NN using Matlab implementation. We use Tiny-dnn [29] to implement NN architecture in C++. We then compare the power, execution time, and accuracy of these designs running on CPU cores. We use Intel Core i7 processor with 16 GB memory (4-core, 2.8GHz) to train and test. For measurement of the CPU power, we use Hioki 3334 power meter. To estimate the cost of digital design, we also use a standard cell-based flow to design dedicated hardware for VoiceHD and VoiceHD+NN. We describe the proposed designs using RTL System-Verilog.

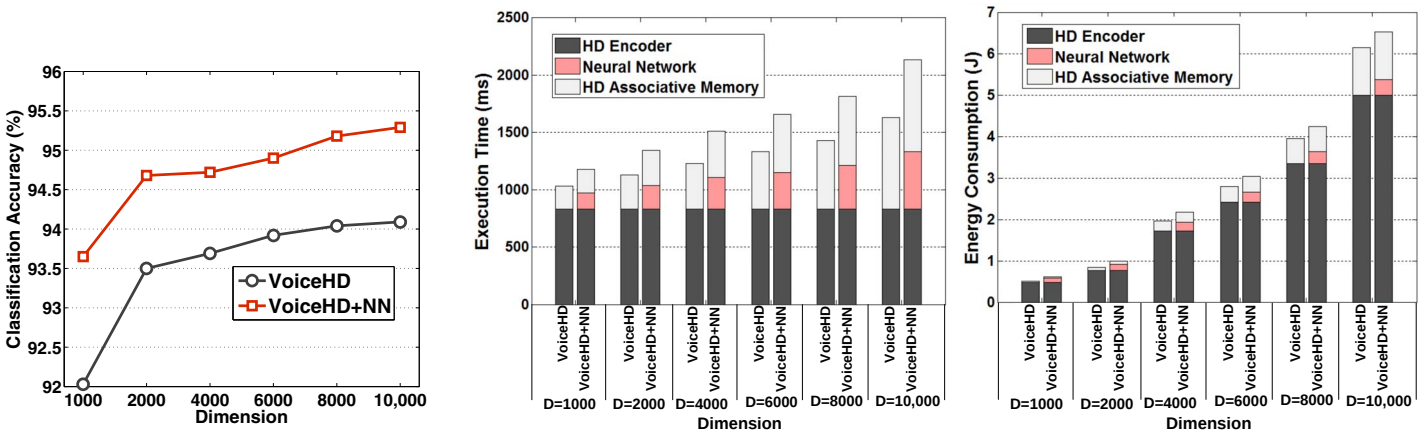
TABLE I
Energy consumption and execution time of NN, VoiceHD, VoiceHD +NN during training and testing on CPU.

		NN	VoiceHD	VoiceHD +NN
Training	Execution Time/Dataset	17min	3.7min	5.9min
Testing	Energy Consumption/Query	454mJ	38mJ	53mJ
	Execution Time/Query	4.61ms	0.87ms	1.14ms

To assess the efficiency of proposed design, we apply the application to the speech recognition of 26 English letter of alphabets. The training and testing data sets are taken from the Isolet dataset [16]. This dataset consists of 150 subjects pronounce each letter of the alphabet twice. The speakers are grouped into sets of 30 speakers each, and are referred to as Isolet1, Isolet2, Isolet3, Isolet4, and Isolet5. The training is performed on Isolet1+2+3+4, and testing is done on Isolet 5. We compare the efficiency of our HD-based designs with the well-known NN designs in the following. Although there is a deep NN [26] with low error rate of 4.1% over the Isolet dataset, its topology uses 48 hidden layers which makes its efficiency unreasonable to compare with the single stage VoiceHD and tiny VoiceHD+NN. Instead, we use another relatively smaller NN with three hidden layers (L^1 : 617, L^2 : 1024, L^3 : 1024, L^4 : 512, L^5 : 26) which provides 6.4% error rate [26], [27], [28]–we refer to it as NN in the rest of paper.

B. Training Efficiency

We compare the energy efficiency and speedup of HD-based designs and the NN during the training mode. Table I lists the training execution time of VoiceHD, VoiceHD+NN and NN on CPU. The result shows that VoiceHD provides significantly higher efficiency than NN during training. This efficiency



(a) Accuracy (b) Execution Time (c) Energy Consumption

Fig. 6. The accuracy, energy consumption, and execution time of the synthesized HD-based designs with scaled dimensionality.

comes from (i) one-shot learning capability of the HD and (ii) using simple element-wise operations for computation rather than using costly floating-point operations. This efficiency is more obvious in the dedicated digital design, as the CPU cores do not use dedicated hardware for fast element-wise HD operations. Our evaluation shows that VoiceHD and VoiceHD+NN provide 4.6× and 2.9× faster training time compared to NN.

C. Testing Efficiency

Table I lists the average energy consumption and execution times of VoiceHD, VoiceHD+NN and NN for a voice query over the test dataset. For VoiceHD and VoiceHD+NN, both encoder and associative memory contribute to VoiceHD energy and execution time. However, the encoder is the dominant part since it operates on $N = 617$ input channels as the frequency bins, while the associative memory has only 26 classes. Comparing VoiceHD and VoiceHD+NN indicates that the VoiceHD provides the maximum efficiency due to using pure HD operations. In VoiceHD+NN design, its NN stage increases the energy consumption by performing the costly matrix multiplication using floating-point numbers. Although the VoiceHD+NN consists of both HD and NN layers, it still provides higher efficiency compared to the pure NN with fpur hidden layers. Our evaluation shows that the VoiceHD and VoiceHD+NN provide 11.9× and 8.5× higher energy efficiency, and 5.3× and 4.0× speedup compared to NN, while providing the similar test accuracy.

D. Scalability

Here, we compare the efficiency of synthesized VoiceHD and VoiceHD+NN using hypervectors with smaller dimensions (D). Figure 6(a) shows the classification accuracy when the dimension of hypervectors is scaled from 10,000 to 1,000. The result shows that the HD-based designs provide a graceful degradation with the reduced dimensions. In comparison with VoiceHD, VoiceHD+NN performs slightly better at the lower dimensions: for example, decreasing the dimension

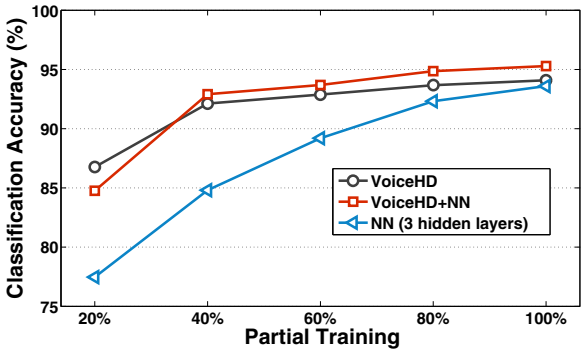


Fig. 7. Classification accuracy of VoiceHD, VoiceHD+NN and NN in partial training.

from 10,000 to 1,000 reduces the VoiceHD and VoiceHD+NN classification accuracy only by 2.1% and 1.6%, respectively. This is due to the fixed size of NN stage that can compensate the misclassifications due to low dimensionality in VoiceHD.

Figure 6(b),(c) show the energy consumption and execution time as well. As shown, the dimension has direct impact on the energy and execution time of both designs. In the encoder, the large dimension increases the size of component-wise operations. In the associative memory, the higher dimension increases the size of XOR array. In addition, the bit-width of the counter (and, comparator) blocks increases linearly (and, logarithmically) with the hypervector dimension D. This results in higher energy consumption and slows down the computation of the associative memory in large dimension. Considering the overall efficiency, increasing the hypervector dimension from 1,000 to 10,000 increases the energy consumption of VoiceHD and VoiceHD+NN by 10.0× and 5.4×, and results in 1.4× and 1.2× higher execution time respectively.

E. Partial Training

We compare the learnability of VoiceHD, VoiceHD+NN, and NN in terms of their ability to infer from a smaller training dataset. For NN, the momentum is set to 0.1, the learning rate is set to 0.001, and the batch size to 10 [30]. Dropout [31] with drop rate of 0.5 is applied to the hidden layers to avoid over-fitting. The activation functions are set to rectified linear unit clamped at 6. A softmax function is also applied to the output layer. Figure 7 shows the accuracy of VoiceHD, VoiceHD+NN and NN designs when the have been trained only with a portion of training dataset. The figure indicates that the HD-based designs show significantly faster learning rate compared to NN. For example, VoiceHD and VoiceHD+NN can still maintain high accuracy of 91.7% and 92.9% when they use only 40% of examples in the training dataset, while the NN accuracy sharply drops by shrinking the training examples below 80%. When training with only 20% of the examples, both HD-based designs show classification accuracy above 85% while NN shows an accuracy lower than 80%. Moreover, it is clear that by using the full set of training examples, the tiny VoiceHD+NN surpasses the accuracy of NN with three hidden layers (95.3% versus 93.6%).

VI. Conclusion

In this paper, we propose VoiceHD, a novel design based on HD computing for fast, efficient and highly accurate speech recognition. VoiceHD is design to significantly reduce the cost of training and testing for speech recognition. VoiceHD encodes a voice signals in the frequency domain to random hypervectors and combines them to generate a prototype hypervector representing each class. VoiceHD+NN further couples a small neural network (with a total number of 102 neurons) at the last stage of VoiceHD to improve the resolution of similarity measures and decision. We evaluate efficiency of VoiceHD and VoiceHD+NN compared to a deep neural network (with a total number of 3,203 neurons) over Isolet spoken letter dataset. Using the full set of training examples, VoiceHD and VoiceHD+NN show classification accuracy of 93.8% and 95.3% which are marginally better than the deep neural network (i.e., 93.6%), in additions to 11.9× and 8.5× higher energy efficiency. Moreover during testing, VoiceHD and VoiceHD+NN achieve 5.3× and 4.0× faster execution on CPU compared to the deep neural network; similarly during training 4.6× and 2.9× faster executions are achieved.

Acknowledgment

This work was supported by NSF grants 1730158 and 1527034.

References

[1] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649, IEEE, 2013.

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, pp. 173–182, 2016.

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

[4] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, pp. 2508–2521, Sept 2017.

[5] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

[6] P. Kanerva, "What we mean when we say "what's the dollar of mexico?": Prototypes and mapping in concept space," in *AAAI Fall Symposium: Quantum Informatics for Cognitive, Social, and Semantic Processes*, pp. 2–6, 2010.

[7] A. Joshi, J. Halseth, and P. Kanerva, "Language geometry using random indexing," *Quantum Interaction 2016 Conference Proceedings*, In press.

[8] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy efficient classifier using brain-inspired hyperdimensional computing," in *Low Power Electronics and Design (ISLPED), 2016 IEEE/ACM International Symposium on*, August 2016.

[9] M. Imani, A. Rahimi, J. Hwang, T. Rosing, and J. M. Rabaey, "Low power sparse hyperdimensional encoder for language recognition," in *IEEE Design and Test*, IEEE, 2017.

[10] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth*.

[11] O. J. Räsänen, "Generating hyperdimensional distributed representations from continuous-valued multivariate sensory input," in *Proceedings of the 37th Annual Meeting of the Cognitive Science Society, CogSci 2015, Pasadena, California, USA, July 22-25, 2015*, 2015.

[12] A. Rahimi, P. K. L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *IEEE International Conference on Rebooting Computing (ICRC 2016)*, October 2016.

[13] A. Rahimi, P. Kanerva, J.d.R. Millán, and J. M. Rabaey, "Hyperdimensional computing for noninvasive brain–computer interfaces: Blind and one-shot classification of eeg error-related potentials," in *10th EAI International Conference on Bio-inspired Information and Communications Technologies (BICT)*, 2017.

[14] O. Räsänen and S. Kakouros, "Modeling dependencies in multiple parallel data streams with hyperdimensional computing," *IEEE Signal Processing Letters*, vol. 21, no. 7, pp. 899–903, 2014.

[15] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2015.

[16] UCI machine learning repository. http://archive.ics.uci.edu/ml/datasets/ISOLET.

[17] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pp. 273–278, IEEE, 2013.

[18] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 4960–4964, IEEE, 2016.

[19] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, p. 1237, Barcelona, Spain, 2011.

[20] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3642–3649, IEEE, 2012.

[21] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 32–37, IEEE, 2009.

[22] J.-Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.-J. Yoo, "A 201.4 gops 496 mw real-time multi-object recognition processor with bio-inspired neural perception engine," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 32–45, 2010.

[23] R. St Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 505–516, 2014.
[24] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE Computer Society, 2014.
[25] B. Logan *et al.*, "Mel frequency cepstral coefficients for music modeling.," in *ISMIR*, 2000.
[26] S. Parameswaran and K. Q. Weinberger, "Large margin multi task metric learning," in *Advances in neural information processing systems*.
[27] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, pp. 1473–1480, 2006.
[28] G. Chechik, U. Shalit, V. Sharma, and S. Bengio, "An online algorithm for large scale image similarity learning," in *Advances in Neural Information Processing Systems*, pp. 306–314, 2009.
[29] tiny-dnn. https://github.com/tiny-dnn/tiny-dnn.
[30] Sutskever *et al.*, "On the importance of initialization and momentum in deep learning.," *ICML (3)*, vol. 28, pp. 1139–1147, 2013.
[31] Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.