HDVR: Hyperdimensional computing for audio classification tasks

Nikolaj Banke Jensen
Department of Computer Science and Technology
University of Cambridge
Cambridge, CB3 0FD
`nbj23@cam.ac.uk`

Abstract

This paper introduces Hype and HDVR as hyperdimensional computing (HDC) implementations targeting embedded devices. We explore the implications of such a framework on a common audio classification task using the ISOLET dataset and show that HDVR is capable of achieving 94% accuracy with only 160 seconds of preparation time. We demonstrate that single-pass training, high energy efficiency, and avoidance of floating-point operations are properties of HDC which makes the approach relevant to embedded devices.

1 Introduction

Audio classification is a common supervised machine learning task where appropriate labels are assigned to audio file inputs. This task is particularly interesting in cases where we want devices capable of reacting to human voice commands. For instance, smart speakers might use audio classification algorithms to detect a certain activation phrase (Amazon 2023, Feng 2023). In smart speakers, like Amazon Echo, a locally running algorithm is responsible for continuously listening for the activation phrase, and upon detection it will relay voice commands to more powerful systems in datacentres (Amazon 2023). Nevertheless, Arachchi & Samarasinghe (2023) has shown that customers' *perceived assurance* of their privacy is low when using these devices, reporting that users feel unsure of how their intimately collected data is being used. Arachchi & Samarasinghe (2023) particularly report that users feel uneasy about an internet-connected device "always listening".

This paper focuses on how machine learning can be deployed to embedded or IoT devices. This could be helpful in smart speakers where one could imagine a completely hardware-isolated system dedicated to listening to the activation phrase, and which, upon detection, would activate a more sophisticated, internet-connected system. One obstacle is that the task of continuously running audio classification then falls on a device which by design is intended to be simple. Moreover, audio classification tasks are often solved with neural networks (Feng 2023) which are data-centric and generally trade intensive resource use for versatility and performance across domains. In effect, neural networks are somewhat unsuitable for embedded devices due to their hardware requirements.

In this paper we will explore alternative solutions to neural networks for embedded devices, or devices which operate with extremely private data. To this end, we show how hyperdimensional computing can replace neural networks for edge computing. The main objective is to emphasize alternative solutions to neural networks, which are hardware-friendly, transparent, and highly accurate.

Hyperdimensional computing (HDC) is a branch of machine learning introduced by Plate (1995) and developed further by Kanerva (2009). While neural networks conceptually represent functions mapping between domains, HDC is a form of symbolic machine learning which manipulates vectors in high-dimensional spaces in a manner which closely represents how humans think (Kanerva 2009). Hyperdimensional computing has been used in machine learning tasks before (Ge & Parhi 2020, Kanerva 2009) due to its positive qualities: for instance, it requires little data for meaningful results, can be efficiently implemented in hardware, and has fully transparent decision-making (Kanerva 2009).

In this paper we show how hyperdimensional computing can deliver state of the art performance of 94% accuracy on the ISOLET dataset[1] for speech classification. We base our architecture on VoiceHD (Imani et al. 2017), but implement our architecture in C++ for increased compatibility with embedded devices. Concretely, we first introduce the reader to hyperdimensional computing and its uses in machine learning. We then describe Hype and HDVR – the main contributions of this paper – as improved architectures capable of handling binary, integer, and floating-point hypervectors for embedded devices. We also perform a comparative analysis of optimal parameter choices. Lastly, we argue that the proposed architecture is more scalable and hardware friendly than existing neural network solutions for audio classification.

2 Hyperdimensional computing

As the reader might not be familiar with hyperdimensional computing, we will here introduce some important notions used extensively in this paper. For brevity, we only cover the most essential notions here, but refer the reader to Kanerva (2009) for a thorough treatment and verification of the details.

Hypervectors Are arrays/strings typically of length 5,000 to 15,000. The type of individual elements can vary, but is often integers, floating-point numbers, bits (0 or 1), or polar bits (-1 or 1) Kanerva (2009). The benefit of operating in high-dimensional spaces is that two random vectors with i.i.d. components generally are dissimilar. Typically, similarity is measured using Hamming distance (for binary vectors) or cosine distance. Figure 1 shows the Hamming distance (where 0.5 is most dissimilar) is represented by a narrow probability distribution around dissimilar vectors.

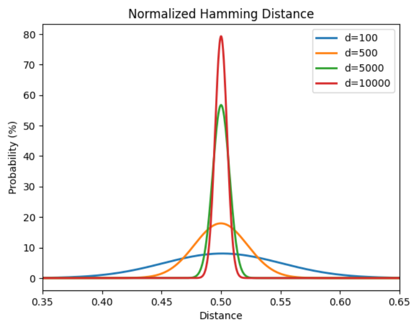

Figure 1: Normalized Hamming distance of binary vectors.

$$\cos(A, B) = \frac{A \cdot B}{|A||B|}$$

$$\text{Ham}(A, B) = \frac{1}{d}\sum_{i=}^{d} 1_{A_i \neq B_i}$$

This phenomenon is commonly referred to as the “curse of dimensionality” (Verleysen & François 2005), but it is helpful in HDC because it gives easy access to highly orthogonal vectors. These hypervectors can then be assigned “meaning”: for example, we could assign orthogonal hypervectors to different classes in a classification problem.

To meaningfully compute with hypervectors we have two types of operations: add and multiply. There are other operations – notably permutation Kanerva (2009) – but these are not directly relevant to algorithms discussed in this paper, so we omit the details.

$[+]$: Bundling An element-wise addition of hypervectors, denoted $X = [A + B + C]$, for hypervectors A, B, C (Ge & Parhi 2020, Kanerva 2009). The resulting hypervector X is “closer” (measured by distance, as above) to any of its constituent hypervectors than a randomly sampled hypervector. For an intuition, consider hypervectors A and B denoting “circle” and “blue” respectively, then $X = [A + B]$ denotes a “blue circle”, which is expected since the conceptual meaning of X also lies closer to both A and B than to a random hypervector. The bundle operation realizes the concept of a set.

[1]https://archive.ics.uci.edu/dataset/54/isolet

In polar, integer, and floating-point hypervectors bundling is regular arithmetic addition. In case of binary elements, bundling is a majority function (Ge & Parhi 2020, Schlegel et al. 2020) as indicated in (1). This is to avoid the resulting hypervector quickly becoming saturated with 1's.

$$\begin{array}{r} A = 1\ 1\ 1\ 1\ 0\ 0\ 0 \\ B = 0\ 0\ 1\ 0\ 0\ 1\ 0 \\ C = 0\ 1\ 1\ 1\ 1\ 0\ 0 \\ \hline [A + B + C] = 0\ 1\ 1\ 1\ 0\ 0\ 0 \end{array} \tag{1}$$

$\oplus$: Binding An element-wise multiplication of numbers, denoted $X = A \oplus B \oplus C$, for hypervectors A, B, C (Ge & Parhi 2020, Kanerva 2009). Unlike with bundling, the resulting vector X is (approximately) orthogonal to all its constituent hypervectors. The binding operation has an inverse operation, $\oslash$, which has the opposite effect: if $X = A \oplus B$, then $A \oslash X = B$ (Schlegel et al. 2020). For an intuition, consider A denotes a variable in a program and B denotes a value, then $X = A \oplus B$ denotes the assignment of B to A. Similarly, we can "retrieve" the value B by taking $A \oslash X = B$.

In the case of polar hypervectors (elements of -1 and 1), we have that $\oplus = \oslash = *$, where $*$ is regular arithmetic multiplication. In the case of binary hypervectors $\oplus = \oslash =$ `XOR`. Binding is not always its own inverse (Schlegel et al. 2020); integer hypervectors, for example, will have different binding and unbinding operations.

$$\begin{array}{r} A = 1\ 1\ 1\ 1\ 0\ 0\ 0 \\ B = 0\ 0\ 1\ 0\ 0\ 1\ 0 \\ \hline A \oplus B = 1\ 1\ 0\ 1\ 0\ 1\ 0 \end{array} \tag{2}$$

Associative memory We realize the concept of a "memory" as a collection of hypervectors which represent symbols to remember. If we have a hypervector Z, we can recall the meaning of Z by finding the hypervector in memory whose distance is the smallest from Z (Kanerva 2009, Schlegel et al. 2020). This is a powerful concept, because it allows memory to have a de-noising effect; if Z was the same hypervector as Y with added noise, then Z is likely still the closest hypervector to Y due to the sparseness of high-dimensional space. Thus, the curse of dimensionality becomes a tool for error-correcting memory. In HDC terms, this type of error-correcting memory is called an associative memory (Kanerva 2009).

2.1 Hyperdimensional classification

Due to entirely transparent decision-making and hardware-friendliness, HDC has been applied extensively to a variety of traditional machine learning tasks (Ge & Parhi 2020). Figure 2 shows a typical configuration for a classification problem.

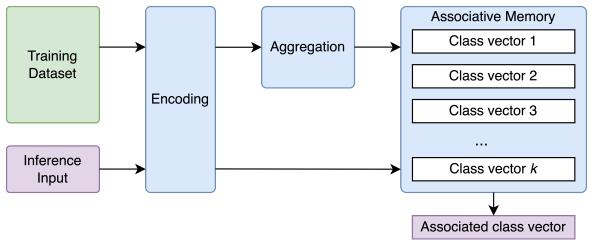

Figure 2: Typical HDC machine learning configuration.

Training data is collected as usual in machine learning. This is then encoded into hypervectors. The encoding process is domain-specific because data formats vary between domains. Thus, applying HDC classification to a particular domain involves manufacturing a suitable encoding algorithm (Ge & Parhi 2020). After encoding each training sample into a hypervector, the aggregation-phase constructs a single hypervector for each class. This is typically done by splitting all the hypervectors into sets of hypervectors based on their class, and then performing the "bundling" operation on each set (Ge & Parhi 2020). Provided a meaningful encoding method is used, the resulting hypervectors should be roughly similar to any of the hypervectors used to construct it (measured by distance), and it should be roughly dissimilar to any vector from another class.

Running inference on this model is simply the process of encoding the input, and then finding the hypervector in memory which is most similar (Kanerva 2009, Schlegel et al. 2020, Ge & Parhi 2020). The resulting class hypervector is associated with the correct class for the input, if the system works as intended. One of the benefits of HDC for machine learning tasks is that it requires only single-pass training (Kanerva 2009). That is, HDC models generally perform well after considering each dataset sample just once.

3 Related work

Imani et al. (2017) use the framework described in section 2.1 on the ISOLET dataset for audio classification. The ISOLET dataset (Cole & Fanty 1994) consists of recordings of 150 human subjects pronouncing each of the 26 letters in the English alphabet twice. Specifically, each datapoint is a voice recording represented as a vector of 617 elements corresponding to the amplitude of a particular frequency in the input data. Due to natural effects when recording audio, the dataset is relatively noisy (Cole & Fanty 1994). Imani et al. (2017) use 80% of the dataset for training, and the rest for testing.

The contribution of Imani et al. (2017) is VoiceHD, an HDC model which provides 11.9x higher energy efficiency and 4.6x faster training time compared to a reference deep neural network while achieving similar classification accuracy of ~93%. VoiceHD makes use of "retraining" to achieve ~93% accuracy – with just one training pass the accuracy is ~88%.

Retraining is a technique used to refine the associative memory after the initial construction of hypervectors. For every training example, the model is used to predict the appropriate class. If the model misclassifies a hypervector X as being closer to A than the correct hypervector B, then the memory is modified by setting $A = [A - X]$ and $B = [B + X]$. Note that the subtraction symbol implies bundling with the inverse of X. The process of retraining significantly improves accuracy from ~88% to ~93% after several iterations of the training dataset.

Furthermore, Imani et al. (2017) show that when using binary hypervectors a simple circuit of XOR-gates can implement the system. This circumvents the need for a CPU, and makes the system extremely fast and energy efficient.

This paper is intended as a replication-study of the contributions of Imani et al. (2017). Specifically, this paper implements a system similar to VoiceHD but extended to support not only binary hypervectors, but also integer, floating-point, and polar hypervectors.

As in Imani et al. (2017), HDC is often combined with a neural network architecture. Neural networks are statistical generalisations over input data (Ma & Jiao 2022), so while they perform well across many domains they lack the ability to decompose joint representations and reason about the distinct objects (Hersche et al. 2023). HDC on the other hand, requires domain-specific encoding algorithms (Ge & Parhi 2020) and suffers from exhaustive memory search problems (Hersche et al. 2023). In recent years, research has focused on combining the two approaches. For instance, the Raven's Progressive Matrices problem (Zhuo & Kankanhalli 2020) is infamously hard to solve for neural networks alone as it requires logical reasoning about distinct objects in an image. Hersche et al. (2023) have recently shown how HDC supplements a shallow neural network to achieve record accuracy of 87.7% on this problem, while cutting training time by two orders of magnitude. This shows HDC has an important role in future machine learning. In this paper, we focus on HDC itself and leave a neural network integration for future work.

4 Design and implementation

The contribution of this paper is two-fold: (1) development of Hype, a HDC C++ library for embedded devices, and (2) development of HDVR, an implementation of VoiceHD using Hype. We first describe Hype and HDVR at a high level, followed by notes on the implementation.

4.1 Design

The encoding process in HDVR is described in Figure 3.

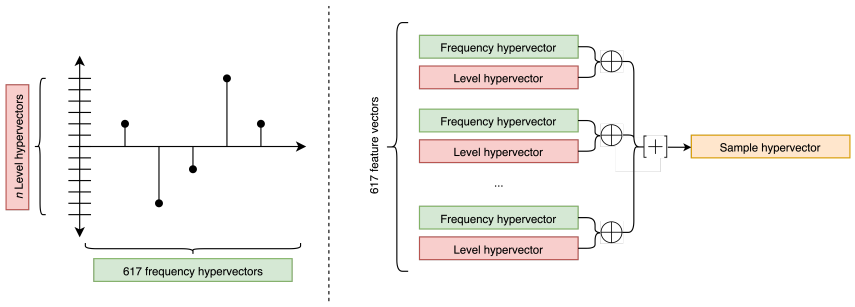

Figure 3: HDVR encoding.

HDVR is a model designed for the ISOLET dataset, just as VoiceHD. HDVR will use 80% of the dataset for training (6238 samples) and 20% for testing (1559 samples). As mentioned, each datapoint is a 617-dimensional vector of elements corresponding to amplitudes at the respective frequencies. To encode the frequency data into high-dimensional spaces, we construct two sets of hypervectors: First, 617 frequency hypervectors are constructed which each correspond to a particular frequency in the domain. Frequency hypervectors are random vectors with i.i.d. components. Secondly, n level hypervectors are constructed. Each level hypervector corresponds to a particular amplitude in the signal. While the frequency hypervectors are orthogonal (due to being randomly sampled), the level hypervectors are constructed more methodically.

The level hypervector L_1 is randomly sampled, and will be used to construct hypervectors $L_2, L_3, \ldots, L_n$. Let $c = \frac{d}{(n-1)}$, where $d = 10000$ and n is the number of partitions along the amplitude range. L_2 is constructed by inverting the first c elements of L_1. Subsequent level hypervectors are constructed as follows: $L_j = I_{(j-1)c}(L_1)$, where I_k is the inversion of the first k elements. The resulting set of level hypervectors are related to each other; hypervector L_1 is mostly similar to L_2, while being maximally dissimilar to $L_{n/2}$. Moreover, L_1 and L_n are inversely similar (that is, they are opposites).

The same set of frequency hypervectors and level hypervectors are used to encode every sample used by the model. To encode a sample from the dataset, first find the appropriate level hypervector (the hypervector associated most closely with the amplitude of the signal), and take the appropriate frequency hypervector. In effect, 617 pairs of frequency hypervectors and corresponding level hypervectors are formed. We use the binding operation to "assign" the level hypervectors to the frequency hypervectors. Finally, the bundling operation condenses the hypervectors into a single sample hypervector, which represents the dataset sample in hyperspace. Multiple sample hypervectors are bundled to create a single class vector which is inserted in the associative memory.

The hyperdimensional arithmetic supported by Hype is outlined in Table 1.

Type	Vector space	Bundling	Binding	Unbinding
1	$(0,1)^D$	Majority	Logical XOR ($\oplus$)	Logical XOR ($\oslash$)
2	$\mathbb{Z}^D$, $\mathbb{R}^D$, $(-1,1)^D$	Arith. add. ($+$)	Arith. mult. ($*$)	Arith. mult. ($*$)

Table 1: HDVR hyperdimensional arithmetic. D is the dimensionality.

4.2 Implementation

Although this paper focuses on audio classification, many HDC components are universal across domains. For example, hypervectors and associative memory models are recurring for most HDC use-cases. For this reason, the implementation is split into two parts: (1) an underlying library, Hype, with universal HDC components, and (2), HDVR, containing audio classification components such as the encoder described in section 4.1. Hype and HDVR are implemented in C++17 and have no external dependencies except for the standard library. This helps increase compatibility with embedded or IoT devices where other libraries may not be available.

Concrete implementations Hypervectors are specified using the interface `IVector`. Hype offers two implementations: `Vector` for all-purpose arithmetic hypervectors (type 2 in Table 1), and `BinaryVector` for binary hypervectors (type 1). Binary hypervectors are implemented using `std::bitset`, which minimizes memory footprint and allows for efficient bit-manipulation using C++ primitives such as '^' for XOR. This optimisation is an improvement in Hype over VoiceHD. Additionally, Hype also extends VoiceHD by allowing different "seeding strategies"; these determine how random vectors are sampled. Hype supports polar sampling, binary sampling and floating-point sampling for general hypervectors (type 2).

Memories are specified using the interface `Memory`. Concrete implementations include `AssociativeMemory`, which defines the necessary `find` function that is used to retrieve the closest matching hypervector. The `ContinuousItemMemory` and the `FrequencyChannelMemory` correspond to the collection of level and frequency hypervectors, respectively.

Saving and loading Hype contains infrastructure code which helps load datasets and report errors. Users provide raw datasets (such as ISOLET) in `csv` files, which are then encoded as part of the training process. However, encoding an entire training dataset is time consuming (see section 5), so Hype supports saving the encoded dataset under extension `.datmem`. Hype determines if encoding can be skipped at runtime depending on the file extension. This is another improvement over VoiceHD, which makes it easier to tweak models without encoding repeatedly. Moreover, when `BinaryVector` hypervectors are saved they are converted to hexadecimal digits, which further improves the on-disk storage consumption.

Furthermore, HDVR defines a `Model` class which holds the associative memory, as well as frequency and level memories. HDVR provides facilities for saving a trained set of memories using the `.mem` extension. This allows users to save and load pre-trained models, and is essential for deployment on other devices.

5 Evaluation

Data in this section is collected using the `main.cpp` program found in the project repository and compiled with CMake using AppleClang 15 with C++17 and optimisation flag `-O1`. All benchmarks are collected on a MacBook Pro 16" with an M1 Max processor (8 cores @ 3.2 GHz, 2 cores @ 2GHz), 32Gb RAM. Except for time-based benchmarks, all benchmarks are run only once, because once a set of frequency and level hypervectors are chosen the HDC training process is entirely deterministic with respect to the training data.

HDVR is capable of achieving 94% accuracy on the audio classification task using the ISOLET dataset. This accuracy is obtained when using $d = 5000$ dimensions, $n = 100$ levels, and integer hypervectors (initially seeded with polar bits). Moreover, the single-pass accuracy is 88.6%, and this increases to 94% after just 3 retraining epochs. This is comparable to the state-of-the-art single-task deep neural network model tested by Parameswaran & Weinberger (2010) on the same dataset, which achieves an accuracy of 94%.

5.1 Parameter analysis

When training neural networks, a set of hyperparameters – such as learning rate and batch size – must be selected (Yu & Zhu 2020). Oftentimes, a validation set taken from the combined dataset is used to tune the model between training epochs. Since HDVR is single-pass training, hyperparameters are essentially fixed from the start of training. Nevertheless, the hyperparameters used in HDVR still influence the performance greatly, as we show in this section. We will consider how the number of retraining epochs and dimensionality of hypervectors influence model performance.

As Figure 4 shows, the error (misclassification) rate expectedly drops as the number of retraining epochs increase. We also see that the accuracy is generally high, and that it quickly settles around 93.5% (for 10000 dimensions). We note that it only takes 3 retraining epochs for accuracy to settle around equilibrium, which is significantly fewer than most neural networks. Bringing down the number of complete training-set iterations is crucial for training models on the edge. Figure 4 also shows the effectiveness of the retraining mechanism in HDVR, although the model already has an initial accuracy of 88.6% after generating class hypervectors.

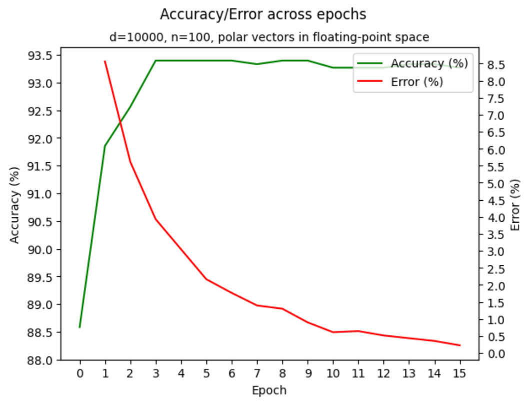

Figure 4: Accuracy of HDVR across epochs.

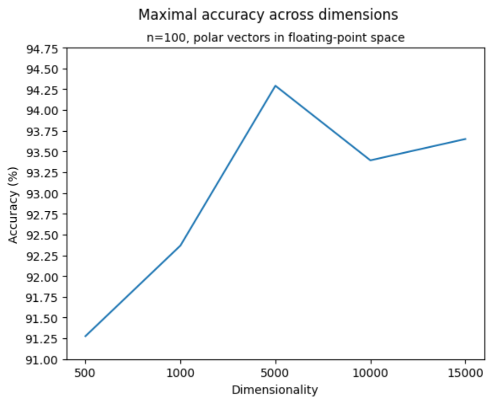

Figure 5: HDVR accuracy across dimensions.

In Figure 5 we show how dimensionality of the underlying hypervectors influence the accuracy of the model. Choosing a low-dimensional hypervector is beneficial for efficient computation because the dimensionality directly affects the number of operations required to compute. Moreover, relatively low-dimensional hypervectors have smaller memory footprints, which is important on embedded systems. Nevertheless, the sparseness of high-dimensional spaces makes it easier to produce consistent results (as vectors are more likely orthogonal) and it allows for more information to be stored in each hypervector before saturation (Kanerva 2009). Figure 5 shows that 5000 dimensions is sufficient to capture the required information in this audio classification task. It should be noted that the model is flexible in memory-constrained environments, where it still produces acceptable results (91% accuracy) at just 500 dimensions.

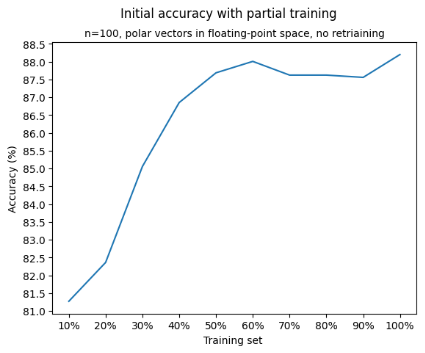

Figure 6: Accuracy (without retraining) on varying training set sizes.

Lastly, we consider how sensitive the model is to low amounts of data. Figure 6 shows the accuracy of the model as the amount of training data varies. Unsurprisingly, the model performs worse when less data is provided, but note that with just 10% of the initial data (roughly 624 samples over 26 classes), the model still achieves >80% accuracy. For many neural networks performance degrades quickly as the amount of data decreases (Markham & Rakes 1998). For HDC this effect is not nearly as severe.

5.2 Wall-clock performance

The VoiceHD implementation is provided by the original authors, written in Python (@moimani 2018). We compare time taken by individual tasks performed by each implementation, as indicated in Table 2. The benchmark uses polar integer hypervectors, with $d = 5000$ dimensionality, $n = 100$ levels, over 10 retraining epochs.

It is clear that HDVR is a slower implementation than VoiceHD. Amongst all tasks, encoding the dataset involves the most computation, and HDVR is 4.7x slower than VoiceHD on this task. On the

Mode	VoiceHD	HDVR
Encoding dataset	**28*s***	132*s*
Training	**25*s***	31*s*
Testing	**493*ms***	598*ms*
Single-sample inference	401*us*	**390*us***

Table 2: HDC wall-clock time performance. Average over 5 repetitions reported.

other hand, both implementations perform equally well on single test samples. It is likely that the Python interpreter is capable of recognising and optimising unnecessary copying of hypervectors during the encoding phase (Zehra et al. 2020), which might be one reason for the large difference. If that is the case, then HDVR could perform equally as well with further optimisations. It should be noted that the author went to great lengths to improve the performance of HDVR but the task is clearly better left to other programmers. Nevertheless, it is unlikely that a competitive neural network approach could deliver the same accuracy with under 3 minutes of training (Lemhadri et al. 2021).

5.3 Suitability for the edge

Many embedded devices do not have hardware support for floating-point arithmetic. For instance, the Arm Cortex M0+[2] processor is not equipped with an FPU. HDVR relies only on integer arithmetic in its most optimal configuration (see section 5), which makes it feasible to achieve high performance on even embedded devices. This is a clear benefit over neural networks which often require floating-point arithmetic. Furthermore, HDVR can be configured to run on binary arithmetic using XOR operations where our tests show that 88.6% accuracy is achievable (with no retraining). Since binary arithmetic can be implemented efficiently directly in silicon, the energy efficiency of HDVR approaches the theoretical maximum (Imani et al. 2017). Conversely, a neural network approach (without integer quantisation) would require software-simulated floating-point arithmetic on some embedded processors (Cornea et al. 2009), which might lead to performance penalties and energy cost overheads.

Furthermore, the HDC approach is useful for IoT devices because it achieves relatively high accuracy with very little data. This would allow, for example, smart sensors to collect and train on their own data, in real time, while almost immediately delivering good results. This would also eliminate costly pre-deployment data acquisition and neural network training.

One drawback to HDVR for embedded devices is the memory search problem (Schlegel et al. 2020, Hersche et al. 2023). Generally, comparing the distance between two hypervectors is $\mathcal{O}(d)$ (in dimensions), and searching for hypervectors in an arbitrarily large associative memory is $\mathcal{O}(k)$ (in class vectors) (Schlegel et al. 2020). Thus, searches can be especially costly as dimensionality increases.

6 Conclusion

We have introduced Hype and HDVR as hyperdimensional computing implementations. We have shown that HDVR, which is based on the VoiceHD architecture, is capable of achieving 94% accuracy on the ISOLET dataset for speech recognition, comparable to state of the art deep neural networks. To achieve this we use retraining of the associative memory as a technique to further enhance the single-pass accuracy of HDVR. Analysis of hyperparameters have shown that a dimensionality of 5000 is adequate for the voice recognition task with polar seeded integer hypervectors. We have also empirically demonstrated the usefulness of retraining and the resilience of HDC in situations with little data. While the wall-clock performance of Hype and HDVR is poor, we argue that with more emphasis on memory optimisations Hype could outperform other HDC libraries on both energy efficiency and accuracy.

The goal of this paper was to position hyperdimensional computing as an alternative to traditional neural networks on the edge. Through analysis of energy efficiency, accuracy, and hardware compatibility, we have demonstrated that this approach is not only viable, but comes with great advantages for embedded devices.

[2]https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m0-plus

References

Amazon, I. (2023), ‘Alexa, Echo Devices and Your Privacy’. Accessed: 10-01-2024.
URL: *https://www.amazon.co.uk/gp/help/customer/display.html?nodeId=GVP69FUJ48X9DK8V*

Arachchi, H. D. M. & Samarasinghe, G. D. (2023), ‘Impact of embedded AI mobile smart speech recognition on consumer attitudes towards AI and purchase intention across Generations X and Y’, *European Journal of Management Studies* .
URL: *https://doi.org/10.1108/EJMS-03-2023-0019*

Cole, R. & Fanty, M. (1994), ‘ISOLET’, UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C51G69.

Cornea, M., Harrison, J., Anderson, C., Tang, P. T. P., Schneider, E. & Gvozdev, E. (2009), ‘A Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic Using the Binary Encoding Format’, *IEEE Transactions on Computers* **58**(2), 148–162.

Feng, Y. (2023), ‘Intelligent speech recognition algorithm in multimedia visual interaction via BiLSTM and attention mechanism’, *Neural Computing and Applications* .
URL: *https://doi.org/10.1007/s00521-023-08959-2*

Ge, L. & Parhi, K. K. (2020), ‘Classification using Hyperdimensional Computing: A Review’, *CoRR* **abs/2004.11204**.
URL: *https://arxiv.org/abs/2004.11204*

Hersche, M., Zeqiri, M., Benini, L., Sebastian, A. & Rahimi, A. (2023), ‘A Neuro-vector-symbolic Architecture for Solving Raven’s Progressive Matrices’.

Imani, M., Kong, D., Rahimi, A. & Simunic, T. (2017), ‘VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition’, *2017 IEEE International Conference on Rebooting Computing (ICRC)* pp. 1–8.
URL: *https://api.semanticscholar.org/CorpusID:21351739*

Kanerva, P. (2009), ‘Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors’, *Cognitive Computation* **1**(2), 139–159.
URL: *https://doi.org/10.1007/s12559-009-9009-8*

Lemhadri, I., Ruan, F., Abraham, L. & Tibshirani, R. (2021), ‘LassoNet: A Neural Network with Feature Sparsity’.

Ma, D. & Jiao, X. (2022), ‘Hyperdimensional Computing vs. Neural Networks: Comparing Architecture and Learning Process’.

Markham, I. S. & Rakes, T. R. (1998), ‘The effect of sample size and variability of data on the comparative performance of artificial neural networks and regression’, *Computers & Operations Research* **25**(4), 251–263.
URL: *https://www.sciencedirect.com/science/article/pii/S0305054897000749*

@moimani (2018), ‘HD-Original-Encoding’. Accessed: 05-01-2024.
URL: *https://github.com/moimani/HD-IDHV*

Parameswaran, S. & Weinberger, K. Q. (2010), "Large Margin Multi-Task Metric Learning", *in* J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel & A. Culotta, eds, ‘Advances in Neural Information Processing Systems’, Vol. 23, Curran Associates, Inc.

Plate, T. (1995), ‘Holographic Reduced Representations’, *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* **6**, 623–41.

Schlegel, K., Neubert, P. & Protzel, P. (2020), ‘A comparison of Vector Symbolic Architectures’, *CoRR* **abs/2001.11797**.
URL: *https://arxiv.org/abs/2001.11797*

Verleysen, M. & François, D. (2005), "The Curse of Dimensionality in Data Mining and Time Series Prediction", *in* J. Cabestany, A. Prieto & F. Sandoval, eds, ‘"Computational Intelligence and Bioinspired Systems"’, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 758–770.

Yu, T. & Zhu, H. (2020), ‘Hyper-Parameter Optimization: A Review of Algorithms and Applications’.

Zehra, F., Javed, M., Khan, D. & Pasha, M. (2020), ‘Comparative Analysis of C++ and Python in Terms of Memory and Time’, *Preprints* .
URL: *https://doi.org/10.20944/preprints202012.0516.v1*

Zhuo, T. & Kankanhalli, M. (2020), ‘Solving Raven’s Progressive Matrices with Neural Networks’.