Sheet 5: Hyperdimensional Computing

The concept of brain-inspired Hyperdimensional Computing (HDC) has attracted much attention in recent years. In addition to the discussed features in the theory session, it promises one-shot learning capabilities. This means the training set needs to be processed only once to get the class vectors, which starkly contrasts the iterative approach of gradient descent found in most classical machine learning algorithms. The foundation was laid by Kanerva in 2009 when he presented the concept of HDC and described how data structures could be constructed using the few core operations available [1].

In this task sheet, you will implement the operations needed for HDC and the encoding scheme for language recognition. For reference, you can check the slides or the paper by Rahimi et al. where the text-encoding algorithm has been proposed [2]. Start by implementing the necessary building blocks, i.e., the *binary* implementation of the HDC operations. We recommend the usage of packages like `numpy` to store the hypervectors in numpy arrays to speed up the operations on them. Using the core operations, implement the functions for the text encoding, i.e., constructing an N-gram and processing a longer piece of text. Build the structure around the encoding, e.g., functions for initializing the item memory, loading the training and test data, processing the training data and storing the class vectors with their labels, performing the inference of the test text, or calculating the inference accuracy.

We want to point out that frameworks available online (e.g., TorchHD) already implement the HDC algorithm with examples ready for many different applications, including language recognition. Using such frameworks is prohibited for this task sheet!

For training and testing, we provide the data sets in the initial material folder (`/space/chair-nas/teaching/bic_lab/initial_material/hdc/language_recognition_dataset`). Please do not copy the dataset to your home directory. Set the path(s) in your scripts to read the files directly from the NAS. To train the model, use all the text in the file in the respective language. You need to concatenate all lines into one long string for training. To test the inference accuracy, treat each line in a text file as one inference test sample (one line is one sentence). As we have 10 languages with 1000 test sentences each, we have 10 000 test samples to calculate the inference accuracy from. To calculate the accuracy, perform the inference for all test sentences and count the correctly inferred languages, then divide the number of correct inferences by the total number of test samples.

The most apparent hyperparameter to tune is the dimension of the used vectors. We want to explore the role of the dimension and see its effect on inference accuracy. For this, repeat the training and inference accuracy test with different vector dimensions in the following ranges: 100 to 1000 in steps of 100, and from 1000 to 10 000 in steps of 1000. The size of the N-gram should also be swept, with the values 3, 4 and 5. As a reference, the accuracy should be in the range of the values plotted in Figure S5.1.

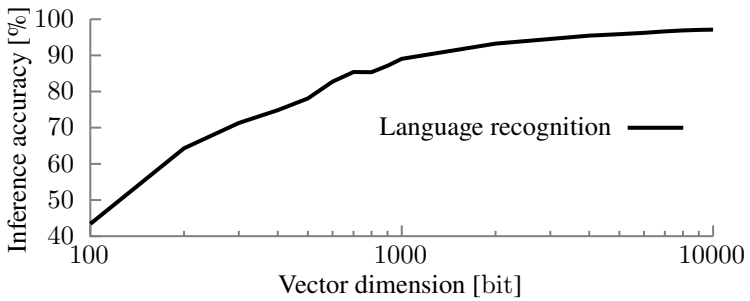

Figure S5.1: Inference accuracy over vector dimension using 4-grams.

Python provides the `multiprocessing` module to parallelize workloads. Ensure you understand the load you are causing and check again the "Load and Machine Utilization" section in the readme for a fair and respectful usage of the machines. With a naive implementation of the encoding algorithm where all N-gram vectors are stored (e.g., in a list) and the summation (bundling) and binarization are done at the end, you will run out of RAM and crash the PC for the larger dimensions. Think of a more efficient implementation that does not need to store all *individual* N-gram vectors (the keyword is accumulation).

Task Summary

- Implement the core HDC functions
- Implement the encoding scheme for language recognition
- Be able to train and test the inference accuracy for any given vector dimension
- Sweep the vector dimension from 100 to 1000 in steps of 100, and from 1000 to $10\,000$ in steps of 1000 and retrieve the inference accuracy respectively. Use different N-gram sizes with $N \in \{3; 4; 5\}$.

Results to Submit

- Plot the inference accuracy over the vector dimension with all three N-gram variations in one plot. The x axes should be the vector dimension on a logarithmic scale.
- Note that your code will be checked at the end of the lab to check if you used any prohibited frameworks. Attach your entire code (e.g., by using LibreOffice Writer) to the PDF with the results as an appendix.

References

[1] 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.

[2] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A Robust and Energy-Efficient Classifier Using Brain-Inspired Hyperdimensional Computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design - ISLPED '16*, San Francisco Airport, CA, USA: ACM Press, 2016, pp. 64–69.