Neuroscience 299: Computing with High-Dimensional Vectors
Assignment 11: Hardware implementations
Due November 17, 1pm

Reminder: Please do *either* the writing assignment *or* the programming assignment. Expected length for the writing assignment is approximately 250-500 words, but there is no strict minimum or maximum.

Writing assignment:

Address the following questions, citing sources and references as appropriate:

- Why does it seem promising to combine HD computing/VSA and unconventional hardware?
- What are the main advantages of using HD computing/VSA from the unconventional hardware point of view?
- What do you think are the most promising application domains?

Programming assignment:

In this programming assignment, we will simulate the non-deterministic behavior present in phase-change memory (PCM) devices operating at low signal-to-noise ratio conditions. First, to set up the context, please read the "Robust High-dimensional Memory-augmented Neural Networks" article.

In essence, this paper suggests combining neural networks with ideas from HD computing/VSA. A neural network (referred to as the *controller*) is trained to take an image as an input and return a d-dimensional HD vector describing the image as an output. Once the controller is trained, it can operate in the context of a few-shot learning task, where for a given m-way (number of classes) n-shot problem (number of training samples per class) the explicit memory is composed of a key memory for storing and comparing the HD vectors produced by the controller (support vectors), and a value memory for storing labels, that are jointly referred to as a key-value memory. Note that the size of the key memory is [d x mn]. We will call such an organization of the key-value memory localist, since the HD vectors in the corresponding parts of the key-value memory can be identified with a particular sample of the training data.

In this assignment, we will only work with the inference phase, i.e., we will not modify the controller architecture and its training process. As a starting point, you are provided with the 512-dimensional HD vectors of the test data obtained as the output of a trained controller. The HD vectors are provided in a file called "DataLab.mat". Please download it using the following link: https://www.dropbox.com/s/1vlz3rlqsuxwufs/DataLab.mat?dl=0

As an alternative to the localist organization of the key-value, we will explore a distributed organization. The main feature of this distributed organization is that it allows decoupling the dimension of the key memory from mn by introducing a free parameter r for controlling the redundancy, such that the dimension becomes [d x r] instead of [d x mn]. In the inference phase, this new parameter r can add or remove redundancy in the representations of the key memory, on demand, without any retraining of the controller neural network. This results in a fully distributed version of the key memory, which is obtained by the linear superposition of the outer products between the support vectors and randomized distributed representations of their corresponding class labels.

You are provided with the Jupyter notebook (see course website) that includes the code to support loading the data and to implement the functionality of the original localist key-value memory in full. The notebook, however, is missing a few steps (search for ***#ToDo*** comments in the notebook), which are related to implementation of the distributed key-value memory.

In order to explore the role of the redundancy introduced by the distributed key-value memory, we will simulate the noise present in the PCM device with an empirically measured model. The noise model of the PCM devices is implemented in the function "NoisePCM". To get a better grasp on the model, please read the section "PCM model and simulations" in "Robust High-dimensional Memory-augmented Neural Networks," as well as Supplementary Table 1 and Supplementary Note 6: Spatial Variability on PCM Crossbar in the corresponding Supplementary Information of the article.

Once you implement all the ***ToDo***s, please perform the following experiments and answer the following questions:

1. What type of key-value memory (binary or bipolar) seems to be more robust against noise? Why is this the case? What is the number of the PCM devices required by each type for a given *r*?
2. How does the value of *r* (*n* in the code) affect robustness of the distributed key-value memory?
3. How do the distributed and localist key-value memories compare to each other when *r*=*mn* (nWay*nShot in the code)? If one seems to be better than the other, why could that be the case?
4. In the implementation, we used orthogonal random vectors in the key memory of the distributed variant. What would be the effect of skipping the orthogonalization step, i.e., using random vectors, which might have some correlations between them? For which values of *r* (*n* in the code), it is more visible?