Computer-Aided Design of Integrated Circuits: Hyperdimensional Computing Encoder Circuit

Ali Safa (ali.safa@imec.be)
Sergio Massaioli (sergio.massaioli@esat.kuleuven.be)
Francesco Lorenzelli (francesco.lorenzelli@imec.be)

November 23, 2022

1 Introduction

The CAD project will be divided into the following three parts:

1. Implementing a model of the HDC encoder circuit in *python* (3 sessions).
2. Setting up HDC training on the *Wisconsin Breast Cancer* (WBC) dataset (3 sessions).
3. Adding the *Nelder-Mead* circuit optimization routine (4 sessions).

Please refer to the set of slides presented during the lectures describing the project for the theoretical background required to complete these sessions. Please have a **copy of them with you** when you work as it will make things easier. At the end of the project, you will be asked to deliver a **brief report**, as well as your **working code** for further assessment.

2 Setting up you python workspace

You should have the following libraries installed: *numpy*, *matplotlib*, *sklearn*.

Please download the project files provided on Toledo. These will be your starting point for the project. The provided files contain:

1. *main_HDC.py* which is the main file where your code will run.
2. *HDC_library.py* where you will need to code the various functions called in *main_HDC.py*.
3. the folder *WISCONSIN* containing the dataset.

We expect you to have some basic notion of the *python* programming language. The notions used in this lab will be very basic and limit themselves to purely sequential programming. Therefore, those of you who have not covered *python* before will be able to learn alongside the project by asking the TAs and looking on-line.

Please also open a blank "playground" python file where you will be conducting *unit tests* for the function that you will write.

3 Part 1: HDC Encoder model

In this first set of sessions, we will code the HDC encoder circuit described in slide 12 of the project slide set.

3.1 Input encoding LUT

Complete the function *lookup_generate* in *HDC_library.py* for *mode == 1*. You will need to write a code that can initialize a random binary LUT following the rule described in the slides. This LUT will then be used in the function *encode_HDC_RFF* to encode the input vectors.

3.2 Generating binary weights

Complete the function *lookup_generate* in *HDC_library.py* for *mode == 0* in order to generate the binary weight matrix following the slides.

3.3 Encoding the input vectors

Complete the function *encode_HDC_RFF*. This function takes the LUT and the binary weight matrix generated by the previous steps in order to encode the WBC dataset and performs the XOR-ing with the weights. Still, this function does not yet perform the cyclic accumulation and thresholding.

3.3.1 UNIT TEST 1

Propose a way to test if the encoder steps implemented so far do correspond to the theory described in the project slides. **You will need to provide all unit test code and their outcome in the report.** Therefore, save a screen-shot of the result of the unit test somewhere alongside an explication that would clearly motivate why the unit test was considered successful.

3.4 Cyclic accumulation and thresholding

Now that the input LUT and binary weights have been coded and tested, write a code that makes the output of *encode_HDC_RFF* **cyclic** given an B_a-bit accumulator. Then, apply the threshold on the result of the cyclic accumulation to obtain the final HDC hypervector (HV) strictly composed of $-1, 0$ and $+1$ values.

Finally, add these steps to the function *evaluate_F_of_x* under the appropriated places indicated in the code. For info, *evaluate_F_of_x* is an all-encapsulating function which will be called during the Nelder-Mead circuit optimisation to compute the Nelder-Mead cost (i.e., F_of_x or C(s) in the slides) for each set of circuit parameter σ, threshold and γ (for training).

4 Part 2: Training the HDC encoder

4.1 Setting up the LS-SVM linear system of equation

We will now implement the linear system of slide 15 in the function *train_HDC_RFF*. First, let's build the matrix β such that the system of equation in slide 15 can be written as $\beta\bar{v} = \bar{L}$. Then, let's build the vector $\bar{L}$.

4.2 Solving the system of equation

Now, write the code in order to solve the system of equation and obtain the solution $\bar{v}$. Then, use $\bar{v}$ according to the theory given in the slides in order to find the HDC centroid (or prototype) for each class (there are two classes here).

4.3 Quantization

Quantize the HDC centroid (or prototype) into D_b bit and compute the amplification factor for the LS-SVM bias to compensate or the fact that the HDC centroid is now coded on D_b-bits (so its dynamic range has changed, therefore the dynamic range of the LS-SVM bias must change as well).

4.4 Computing the accuracy

Complete the function *compute_accuracy* in order to perform the HDC classificationand to return the accuracy of the system.

4.4.1 UNIT TEST 2

Propose a way to test if the HDC training implemented above performs as expected, under both the non-quantized (floating point HDC centroids) and quantized version. Again, save screenshots of the unit test output for your report alongside an explication that would clearly motivate why the unit test was considered successful.

5 Part 3: Nelder-Mead circuit and training optimization

5.1 Evaluating the Nelder-Mead cost

Make sure that the function *evaluate_F_of_x* has been completed during the previous sessions and is able to

1. perform the HDC encoding on both the training and test sets.
2. train the HDC system via the LS-SVM learning step.
3. report the Nelder-Mead cost between accuracy and HDC prototype sparsity given the trade off parameter *lambda_1* and *lambda_2* (where *lambda_1* is considered to be equal to 1 in the slides).

Go through the slides again and make sure that you understand the steps performed by the Nelder-Mead algorithm such that you will be able to code it in the following steps.

5.2 Implementing the Nelder-Mead optimization procedure

Now, go back to the main file *main_HDC.py* and complete the code for performing the Nelder-Mead optimization procedure.

5.2.1 UNIT TEST 3

Propose a way to test if the Nelder-Mead procedure was correctly coded and if everything done so far seems to work as expected **without** launching the full search over all possible trade off parameters *lambda_sp* (which takes a long time to run). Again, save screenshots of the unit test output for your report alongside an explication that would clearly motivate why the unit test was considered successful.

5.3 Running the complete CAD tool

Now, if you think everything is working as expected, run the file *main_HDC.py* and report the graphs obtained at the end. It should look closely to what is shown in slide 23 of the theory slides presented during the lectures.

5.4 *Optional*: Exploring the impact of the Nelder-Mead hyper-parameters

If there is time left, you can explore and document the impact of each Nelder-Mead optimization parameters alpha_simp, gamma_simp, rho_simp and sigma_simp on the result of the optimization procedure.

5.5 *Optional*: Exploring the impact of all other circuit parameters

There are many other hyper-parameter and HDC circuit parameter that we have considered fixed till now. You can also explore and document the behavior of the CAD tool when choosing other circuit parameter etc...