CAD project 2022 : Optimization and training of a hyperdimensional computing (HDC) circuit

Prof. Georges Gielen
Ali Safa, Sergio Massaioli, Francesco Lorenzelli
ESAT-MICAS
Katholieke Universiteit Leuven
gielen@kuleuven.be

© Georges Gielen – KU Leuven

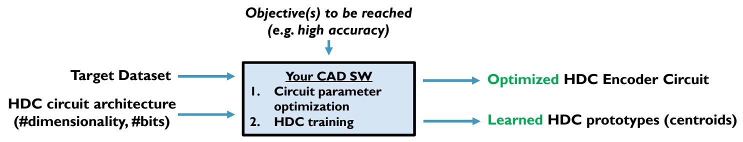
THE TEACHING ASSISTANTS

Ali.Safa@imec.be Sergio.Massaioli@esat.kuleuven.be Francesco.Lorenzelli@imec.be

GOAL: DESIGNING AN HDC CIRCUIT

WITH *CIRCUIT OPTIMIZATION* AND *HDC LEARNING*

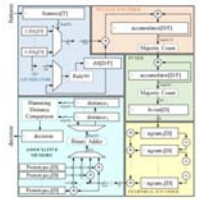
- HDC is a *state-of-the-art* method for performing simple Machine Learning tasks with very little hardware overhead (area-memory-energy efficiency).
- In a nutshell, the idea behind HDC is to:
  1. Encode input data into HW-efficient binary vectors.
  2. Classify these binary vectors via simple operations (such as taking a few inner products).
- In this project, we will write a CAD software in *python* for **automating** the design of an HDC circuit for **cancer detection** from biological signal measurements.


Objective(s) to be reached
(e.g. high accuracy)
Target Dataset
HDC circuit architecture (#dimensionality, #bits)
Your CAD SW
1. Circuit parameter optimization
2. HDC training
Optimized HDC Encoder Circuit
Learned HDC prototypes (centroids)

MOTIVATION

AUTOMATED HDC CIRCUIT DESIGN FOR BIO-SIGNAL CLASSIFICATION TASKS

- Classification problems can be found in numerous tasks.
- ML methods to train classification algorithms allow for high flexibility and accuracy.
- Very effective, but to reach a high accuracy is computationally intensive (energy, time, area).
- ➢ ***Optimal circuit implementation*** of such algorithm is needed for energy efficiency and speed.
- If the *circuit* represents one *implementation* of an *algorithm* it can be modelled and *optimized* before building it.
- **In this lab we will model and optimize such a circuit via the HDC framework.**



Recommended reading – HDC hardware implementation:

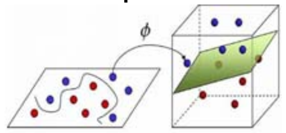
A. Menon, D. Sun, S. Sabouri, K. Lee, M. Aristio, H. Liew, J. Rabaey "A Highly Energy-Efficient Hyperdimensional Computing Processor for Biosignal Classification," in IEEE Transactions on Biomedical Circuits and Systems, vol. 16, no. 4, pp. 524-534, Aug. 2022

HDC ENCODER CIRCUIT AND THEORY

HDC CIRCUIT ARCHITECTURE 1

A.K.A. HEAVILY QUANTIZED RANDOM FOURIER FEATURE MAPPING

- In Machine Learning, we are usually interested in providing a nonlinear, high-dimensional mapping of input signals into **a feature space**.



- A popular mapping $\phi(\bar{x})$ is given by the *Random Fourier Feature (RFF)* map with $\boldsymbol{D}$ **dimension**:

$$\boldsymbol{\phi_{RFF}(\bar{x}) = [\cos(\sigma \times \bar{w}_1^T \bar{x} + b_1), \dots, \cos(\sigma \times \bar{w}_D^T \bar{x} + b_D)]}$$

where b_i are random biases between $[0, 2\pi[$, $\bar{\boldsymbol{w}}_{\boldsymbol{i}}$ are random weights drawn from a standard Normal distribution $N(0,1)$ and σ is a constant setting the nonlinearity degree (the higher, the more nonlinear).

HDC CIRCUIT ARCHITECTURE 2

QUANTIZING RFF

$$\boldsymbol{\phi_{RFF}(\bar{x}) = [\cos(\sigma \times \bar{w}_1^T \bar{x} + b_1), \dots, \cos(\sigma \times \bar{w}_D^T \bar{x} + b_D)]}$$

- We simply threshold the cosine values as $-1, 0, +1$ (ternary representation).

$$\boldsymbol{\phi_B(\bar{x})_i} = \begin{cases} \boldsymbol{+1} \text{ if } \boldsymbol{\cos(\sigma \times \bar{w}_i^T \bar{x} + b_i) > t} \\ \boldsymbol{0} \text{ if } \boldsymbol{|\cos(\sigma \times \bar{w}_i^T \bar{x} + b_i)| \le t} \\ \boldsymbol{-1} \text{ if } \boldsymbol{\cos(\sigma \times \bar{w}_i^T \bar{x} + b_i) < -t} \end{cases}$$

with threshold t and σ optimized automatically by our CAD SW ☺

HDC CIRCUIT ARCHITECTURE 3

GETTING RID OF COSINES

$$\boldsymbol{\phi_B(\bar{x})_i} = \begin{cases} \boldsymbol{+1} \text{ if } \boldsymbol{\cos(\sigma \times \bar{w}_i^T \bar{x} + b_i) > t} \\ \boldsymbol{0} \text{ if } \boldsymbol{|\cos(\sigma \times \bar{w}_i^T \bar{x} + b_i)| \le t} \\ \boldsymbol{-1} \text{ if } \boldsymbol{\cos(\sigma \times \bar{w}_i^T \bar{x} + b_i) < -t} \end{cases}$$

- To remove the cosines from the problem, we remark that the threshold t could be directly applied on the **argument** of the cosines.
- **Under the condition** that the argument wraps around e.g. 2π for exhibiting the cyclic behavior of the cosine.
- The argument $\bar{\boldsymbol{w}}_{\boldsymbol{i}}^{\boldsymbol{T}} \bar{\boldsymbol{x}}$ is an **inner product** and can be computed as a **recursive sum**:

$$\boldsymbol{Acc_k \leftarrow Acc_{k-1} + \sigma\, w_{i,k} x_k} \text{, executed for } k = 1, \dots, n \text{ steps}$$

HDC CIRCUIT ARCHITECTURE 4

ACCUMULATION WITH OVERFLOW

$$\boldsymbol{Acc_k} \leftarrow \boldsymbol{Acc_{k-1}} + \boldsymbol{\sigma\, w_{i,k} x_k}\text{, executed for } k = 1, \dots, n \text{ steps}$$

where $Acc_0 = 0$ and n is the dimensionality of $\bar{x}, \bar{w}$.

- The cyclic behavior (wrapping around) of the argument is obtained each time the accumulator Acc_k **under- or overflows.**
- The larger σ, the *sooner* the accumulator overflows for a fixed bit width.

  (One can also keep σ **fixed** to 1 and optimize the **accumulator bit width B_a.)**
- After the n steps, accumulation overflow terminates and $\boldsymbol{Acc_n}$ quantized as:

$$\boldsymbol{\phi_B(\bar{x})_i} = \begin{cases} \boldsymbol{+1} \text{ if } \boldsymbol{Acc_n - 2^{B_a-1} > t} \\ \boldsymbol{0} \text{ if } \boldsymbol{|Acc_n - 2^{B_a-1}| \leq t} \\ \boldsymbol{-1} \text{ if } \boldsymbol{Acc_n - 2^{B_a-1} < -t} \end{cases}$$

KU LEUVEN

HDC CIRCUIT ARCHITECTURE 5

BINARIZING THE INPUT $\bar{x}$

$$\boldsymbol{\phi_{RFF}(\bar{x})} = [\boldsymbol{\cos(\sigma \times \bar{w}_1^T \bar{x} + b_1)}, \dots, \boldsymbol{\cos(\sigma \times \bar{w}_D^T \bar{x} + b_D)}]$$

- We consider all entries in $\bar{x}$ to be B_{in}-bit integers (e.g., 8-bit in the labs).
- Therefore, the maximum value in $\bar{x}$ is 255 and there are 256 possible values.
- We can define a Look-up Table (LUT) which associates to each of the 256 possibilities, a **random** binary vector of dimensionality D

Input value	Associated HV
0	$[\boldsymbol{L_{0,1}}, \cdots, \boldsymbol{L_{0,D}}]$
⋮	⋮
255	$[\boldsymbol{L_{255,1}}, \cdots, \boldsymbol{L_{255,D}}]$

where $L_{i,j} \in \{-1, +1\}$.

- Each random binary vector L_i must represent its associated value i.
- **Therefore, we generate $\boldsymbol{L_i \sim B_{p_i}}$ with $\boldsymbol{p_i = \frac{i}{2^{B_{in}-1}}}$**
- Each element of $\bar{x}$ is encoded using the LUT to obtain a $n \times D$ matrix of $\{-1, +1\}$ noted $\boldsymbol{L}(\bar{x})$

KU LEUVEN

HDC CIRCUIT ARCHITECTURE 6

BINARIZING THE RFF WEIGHTS $\boldsymbol{w_{i,k}}$

$$\boldsymbol{\phi_{RFF}(\bar{x})} = [\boldsymbol{\cos(\sigma \times \bar{w}_1^T \bar{x} + b_1)}, \dots, \boldsymbol{\cos(\sigma \times \bar{w}_D^T \bar{x} + b_D)}]$$

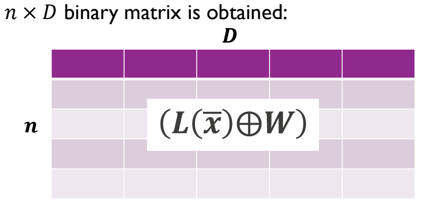
- We have already projected n-dimensional $\bar{\boldsymbol{x}}$ into a $n \times D$ binary representation using our LUT: $\bar{x} \rightarrow \boldsymbol{L}(\bar{x}) \in \{-1, +1\}$
- The weights $\bar{w}_i$ undergo a similar binarization where **each entry** in $\bar{w}_i$ is projected to a D-dimensional random binary vector drawn from a Bernoulli distribution B_p with probability $p = 0.5$ (**equiprobable** outcomes -1 and $+1$).
- Therefore, the matrix $\begin{bmatrix} \bar{w}_1^T \\ \vdots \\ \bar{w}_D^T \end{bmatrix}$ becomes $W \in \{-1, +1\}$ of size $n \times D$
- All inner products $i = 1, \dots, D$ can therefore be approximated as an element-wise multiplication (simple XOR $\oplus$) followed by an accumulation ("bundling"):
- $\boldsymbol{\bar{w}_i^T \bar{x}} \approx \left(\sum_{j=1}^{n} (\boldsymbol{L}(\bar{x}) \oplus \boldsymbol{W})_j\right)_i$ where j denotes the j^{th} row of the $n \times D$ matrix $(\boldsymbol{L}(\bar{x}) \oplus \boldsymbol{W})$

KU LEUVEN

HDC CIRCUIT ARCHITECTURE 7

PUTTING EVERYTHING TOGETHER

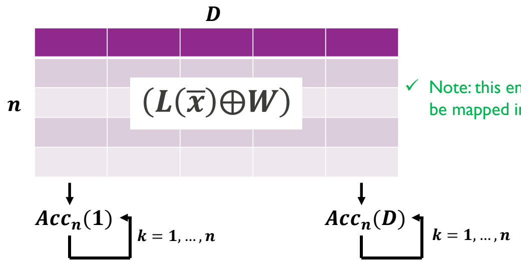
1. The input $\bar{x}$ is encoded into the $n \times D$ $L(\bar{x})$ using our LUT.
2. The random $n \times D$ binary weight matrix W is XOR-ed element-wise with $L(\bar{x})$: $(\boldsymbol{L}(\bar{x}) \oplus W)$. An $n \times D$ binary matrix is obtained:



KU LEUVEN

HDC CIRCUIT ARCHITECTURE 8

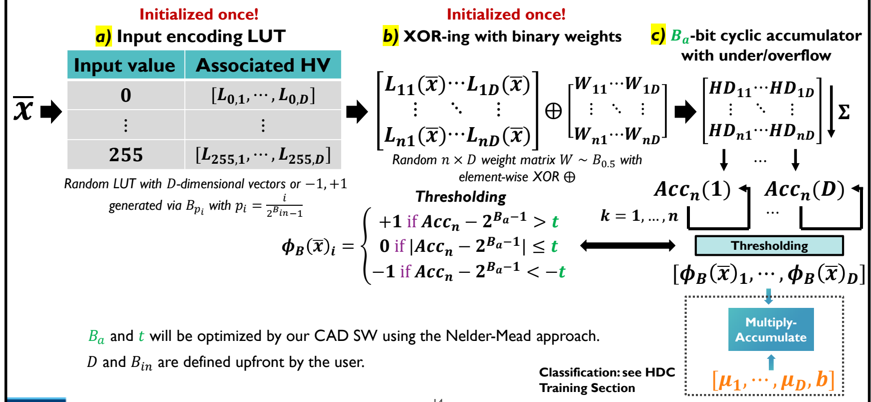
3. To compute the **inner product**, accumulation with overflow is done during n iterations (the accumulator are initialized at different random values to model the random biases b_i in RFF).



✓ Note: this encoder circuit can be mapped into e.g. an FPGA!

4. Finally, the result of each accumulator is thresholded to $\{-1, 0, +1\}$.

HDC CIRCUIT: THE RECIPE



Initialized once!
a) Input encoding LUT

Input value	Associated HV
0	$[L_{0,1}, \cdots, L_{0,D}]$
⋮	⋮
255	$[L_{255,1}, \cdots, L_{255,D}]$

Random LUT with D-dimensional vectors or $-1, +1$ generated via B_{p_i} with $p_i = \frac{i}{2^{B_{in}-1}}$

Initialized once!
b) XOR-ing with binary weights

$$\begin{bmatrix} L_{11}(\overline{x}) \cdots L_{1D}(\overline{x}) \\ \vdots \ \ddots \ \vdots \\ L_{n1}(\overline{x}) \cdots L_{nD}(\overline{x}) \end{bmatrix} \oplus \begin{bmatrix} W_{11} \cdots W_{1D} \\ \vdots \ \ddots \ \vdots \\ W_{n1} \cdots W_{nD} \end{bmatrix} \Rightarrow \begin{bmatrix} HD_{11} \cdots HD_{1D} \\ \vdots \ \ddots \ \vdots \\ HD_{n1} \cdots HD_{nD} \end{bmatrix} \Big\downarrow \Sigma$$

Random $n \times D$ weight matrix $W \sim B_{0.5}$ with element-wise XOR $\oplus$

c) B_a-bit cyclic accumulator with under/overflow

$\overline{x}$
$Acc_n(1)$
$Acc_n(D)$
$k = 1, \dots, n$
Thresholding
$[\phi_B(\overline{x})_1, \cdots, \phi_B(\overline{x})_D]$
Multiply-Accumulate
$[\mu_1, \cdots, \mu_D, b]$

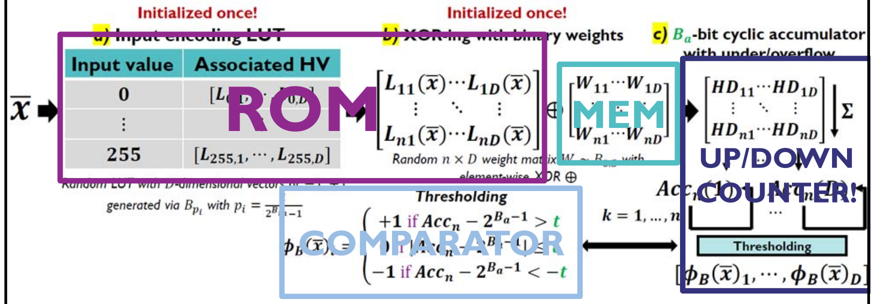
Thresholding

$$\phi_B(\overline{x})_i = \begin{cases} +1 \text{ if } Acc_n - 2^{B_a-1} > t \\ 0 \text{ if } |Acc_n - 2^{B_a-1}| \le t \\ -1 \text{ if } Acc_n - 2^{B_a-1} < -t \end{cases}$$

B_a and t will be optimized by our CAD SW using the Nelder-Mead approach.
D and B_{in} are defined upfront by the user.

Classification: see HDC Training Section

WHAT ABOUT HARDWARE IMPLEMENTATION?


Initialized once!
a) Input encoding LUT
Initialized once!
b) XOR-ing with binary weights
c) B_a-bit cyclic accumulator with under/overflow
Input value
Associated HV
0
255
$[L_{255,1}, \cdots, L_{255,D}]$
ROM
MEM
UP/DOWN COUNTER!
COMPARATOR
Thresholding
$k = 1, \dots, n$
$[\phi_B(\overline{x})_1, \cdots, \phi_B(\overline{x})_D]$

B_a and t will be optimized by our CAD SW using the Nelder-Mead approach.
D and B_{in} are defined upfront by the user.

HDC TRAINING

HDC TRAINING 1

VIA *LEAST-SQUARE SUPPORT VECTOR MACHINES* (LS-SVM)

- Given a dataset of N vectors $\bar{x}$ of dimension n together with the labels $Y \in \{-1,+1\}$, a **train-test split** can be done (e.g., 60% of data as training set, rest as test set, **randomly chosen**).
- Both the train and test vectors $\bar{x}$ can be encoded using our HDC encoder circuit to obtain the associated D-dimensional vectors $\phi(\bar{x}) \in \{-1,0,+1\}$
- An LS-SVM seeks a weight $\bar{\mu}$ and bias b such that:

$$\bar{\mu}, b = \arg\min_{\bar{\mu},b} \frac{1}{2}\bar{\mu}^T\bar{\mu} + \gamma\sum_{i=1}^{N}\xi_i^2 \quad \text{subject to:}$$
$$Y_i(\bar{\mu}^T\phi(\bar{x}_i) + b) = 1 - \xi_i \;\; \forall i = 1,\dots,N$$

✓ Intuition: **First term** is an L2 penalty for regularization against overfitting and second term is simply a **linear regression** with target values $-1,+1$ corresponding to class 1 or 2.

KU LEUVEN 17

HDC TRAINING 2

TRAINING VIA A LINEAR SYSTEM OF EQUATIONS

- By defining $\bar{\mu} = \sum_{i=1}^{N_{train}} \alpha_i \phi(\bar{x}_i)$, it can be shown that LS-SVM training reduces to:

$$\begin{bmatrix} 0 & \bar{Y}^T \\ \bar{Y} & \Omega + \gamma^{-1} I_N \end{bmatrix} \begin{bmatrix} b \\ \bar{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{1}_N \end{bmatrix}$$
$$\Omega_{ij} = Y_i Y_j \phi^T(\bar{x}_i)\phi(\bar{x}_j)$$

- This system can easily be solved in order to find b and the $\alpha_i, i = 1,\dots,N_{train}$
- Then $\bar{\mu}$ can be retrieved. $\bar{\mu}$ will be referred to as the "HDC prototype" or "centroid".
- The smaller γ, the less the system over-fits. γ will be also automatically optimized via Nelder-Mead search.

KU LEUVEN 18

HDC INFERENCE

USING THE TRAINED SYSTEM TO CLASSIFY DATA

- We can further quantize $\bar{\mu}, b$ to B_μ-bit (defined by the user upfront).
- For an incoming data point $\bar{x}$ in the test set, we first encode it via our HDC encoder $\phi(\bar{x})$.
- Then, we perform the inner product test: $\boldsymbol{\bar{\mu}^T \phi(\bar{x}) + b \geq 0}$?
- If this is verified, we infer the label "$+1$", else we give it the label "-1"
- We then check our inferred label against the test label and compute the accuracy of the system by averaging over all the data in the test set.

KU LEUVEN 19

NELDER-MEAD HDC CIRCUIT OPTIMIZATION

20

CIRCUIT HYPERPARAMETER OPTIMIZATION

NELDER-MEAD BASICS

- The Nelder-Mead method seeks to find a solution $\bar{s}$ which minimizes a cost $C(\bar{s})$.
- In our case $\bar{s} = [\gamma, t, \sigma]$ i.e., LS-SVM hyper-parameter, threshold and accumulation speed.
- In contrast to backprop etc... a precise knowledge of the function $f(\bar{s})$ is **not needed,** as long as it can be evaluated somehow.
- In our case, we will try to find solutions that are **both high-accuracy and sparse** (i.e., $\bar{\mu}$ contains lots of 0).
- Therefore, our cost is defined as $f(\bar{s}) = 1 - (\text{ACCURACY} + \lambda_1 \text{SPARSITY})$ where ACCURACY and SPARSITY are evaluated on the HDC system trained with the parameters in $\bar{s}$.
- λ_1 sets the importance of having a high sparsity.
- **We will sweep λ_1 in order to study the tradeoff between accuracy and sparsity.**

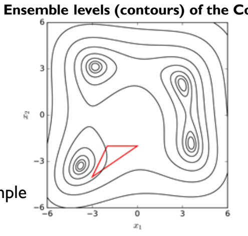
KU LEUVEN 21

NELDER-MEAD 1

INITIALIZING THE SIMPLEX

- A simplex is a bag of N_s hyperparameters $\bar{s}$ that are initialized randomly $\text{Simp} = \{\bar{s}_i, i = 1, \dots, N_s\}$ where N_s is chosen by the user.
- To each hyperparameter vector $\bar{s}$, an initial cost is associated, which gives us a "bag of costs" associated to each $\bar{s}_i$: $\text{Costs} = \{f(\bar{s}_i), i = 1, \dots N_s\}$
- The simplex is therefore roughly covering a possibly large portion of the hyperparameter-cost space.

Ensemble levels (contours) of the Cost



In this toy example, a simplex in red with $N_s = 3$ points converges to a possible solution of the cost minimization problem as Nelder-Mead iterates.

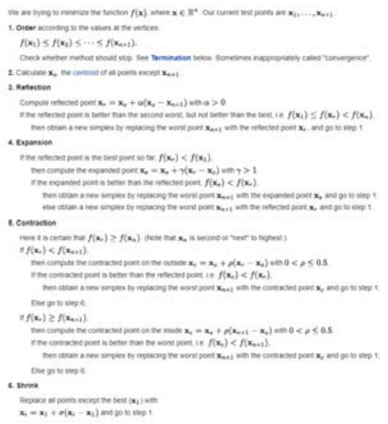
$\bar{s} = [x_1, x_2]$ in this toy example

KU LEUVEN 22

NELDER-MEAD 2

BEST EXPLANATION WE FOUND IS FROM WIKIPEDIA

- A heuristical method... Moves downhill if cost is better while trying to reach lower and lower costs.
- Tries to extrapolate the behavior of the cost function between the points in the simplex.
- In the algorithm shown here, $\alpha, \gamma, \rho, \sigma$ are user-defined parameters setting the speed of convergence.
- **The Nelder-Mead algorithm is straightforward to implement.**



We are trying to minimize the function $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$. Our current test points are $\mathbf{x}_1, \dots, \mathbf{x}_{n+1}$.

1. **Order** according to the values at the vertices:
   $f(\mathbf{x}_1) \le f(\mathbf{x}_2) \le \cdots \le f(\mathbf{x}_{n+1})$.
   Check whether method should stop. See Termination below. Sometimes inappropriately called "convergence".
2. Calculate $\mathbf{x}_o$, the centroid of all points except $\mathbf{x}_{n+1}$.
3. **Reflection**
   Compute reflected point $\mathbf{x}_r = \mathbf{x}_o + \alpha(\mathbf{x}_o - \mathbf{x}_{n+1})$ with $\alpha > 0$
   If the reflected point is better than the second worst, but not better than the best, i.e. $f(\mathbf{x}_1) \le f(\mathbf{x}_r) < f(\mathbf{x}_n)$,
   then obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the reflected point $\mathbf{x}_r$, and go to step 1.
4. **Expansion**
   If the reflected point is the best point so far, $f(\mathbf{x}_r) < f(\mathbf{x}_1)$,
   then compute the expanded point $\mathbf{x}_e = \mathbf{x}_o + \gamma(\mathbf{x}_r - \mathbf{x}_o)$ with $\gamma > 1$
   If the expanded point is better than the reflected point, $f(\mathbf{x}_e) < f(\mathbf{x}_r)$
   then obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the expanded point $\mathbf{x}_e$ and go to step 1;
   else obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the reflected point $\mathbf{x}_r$ and go to step 1.
5. **Contraction**
   Here it is certain that $f(\mathbf{x}_r) \ge f(\mathbf{x}_n)$. (Note that $\mathbf{x}_n$ is second or "next" to highest.)
   If $f(\mathbf{x}_r) < f(\mathbf{x}_{n+1})$,
   then compute the contracted point on the outside $\mathbf{x}_c = \mathbf{x}_o + \rho(\mathbf{x}_r - \mathbf{x}_o)$ with $0 < \rho \le 0.5$
   If the contracted point is better than the reflected point, i.e. $f(\mathbf{x}_c) < f(\mathbf{x}_r)$,
   then obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the contracted point $\mathbf{x}_c$ and go to step 1;
   Else go to step 6.
   If $f(\mathbf{x}_r) \ge f(\mathbf{x}_{n+1})$,
   then compute the contracted point on the inside $\mathbf{x}_c = \mathbf{x}_o + \rho(\mathbf{x}_{n+1} - \mathbf{x}_o)$ with $0 < \rho \le 0.5$
   If the contracted point is better than the worst point, i.e. $f(\mathbf{x}_c) < f(\mathbf{x}_{n+1})$,
   then obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the contracted point $\mathbf{x}_c$ and go to step 1;
   Else go to step 6.
6. **Shrink**
   Replace all points except the best ($\mathbf{x}_1$) with
   $\mathbf{x}_i = \mathbf{x}_1 + \sigma(\mathbf{x}_i - \mathbf{x}_1)$ and go to step 1.

KU LEUVEN 23

NELDER-MEAD 3

EARLY STOPPING AND TERMINATION

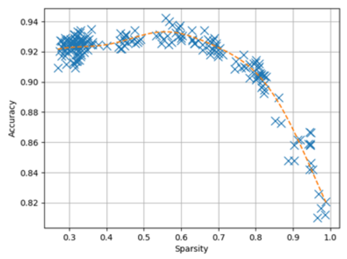
- The user defines a maximum number of iterations.
- Early stopping can be done to accelerate the code.
- As Nelder-Mead iterates, the N_s costs in the "bag of costs" $\text{Costs} = \{f(\bar{s}_i), i = 1, \dots N_s\}$ converge to lower and lower values.
- At some point, all costs in the "bag" are close to each other, indicating that a solution has been found.
- By checking the standard deviation of $\text{Costs} = \{f(\bar{s}_i), i = 1, \dots N_s\}$ against a threshold, the Nelder-Mead optimization is stopped when $std < t_{std}$

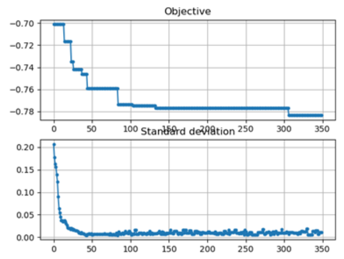
KU LEUVEN 24

RESULTS TO BE OBTAINED AT THE PROJECT'S END

WHAT YOU SHOULD GET AS OUTPUT

- **Graph 1:** Accuracy vs. Sparsity. **Each cross is a possible HDC circuit design.**
- **Graph 2:** Nelder-Mead iterations where the **objective** and the **standard deviation** of the simplex are minimized.





CONCLUSION

NOW IT'S YOUR TURN !!!

- Don't hesitate to try out your **custom ideas**...
  But beware that **time for the lab sessions is limited** !!!
- It is important that
  1. you produce and turn in a correct running code.
  2. your can critically reflect on the accuracy-sparsity tradeoff and explain **why** this matters.
- Plagiarism will **not** be accepted at all and will be checked upon (through software) !!!
  Plagiarism violations will be penalized by failing the project.

- Any questions?