TRAINING BINARIZED NEURAL NETWORKS WITH SIMULATED ANNEALING

Marco Degli Esposti
Matricola 3038432
MSc DSBA
Università Bocconi

Index

- Goals of the project
- Preprocessing
- Binarized Neural Networks
- Simulated Annealing
- Local Entropy
- Conclusions

MAIN GOALS

Implement

Implement a Simulated Annealing algorithm for BNN

Train

Train Binarized Neural Networks with Simulated Annealing

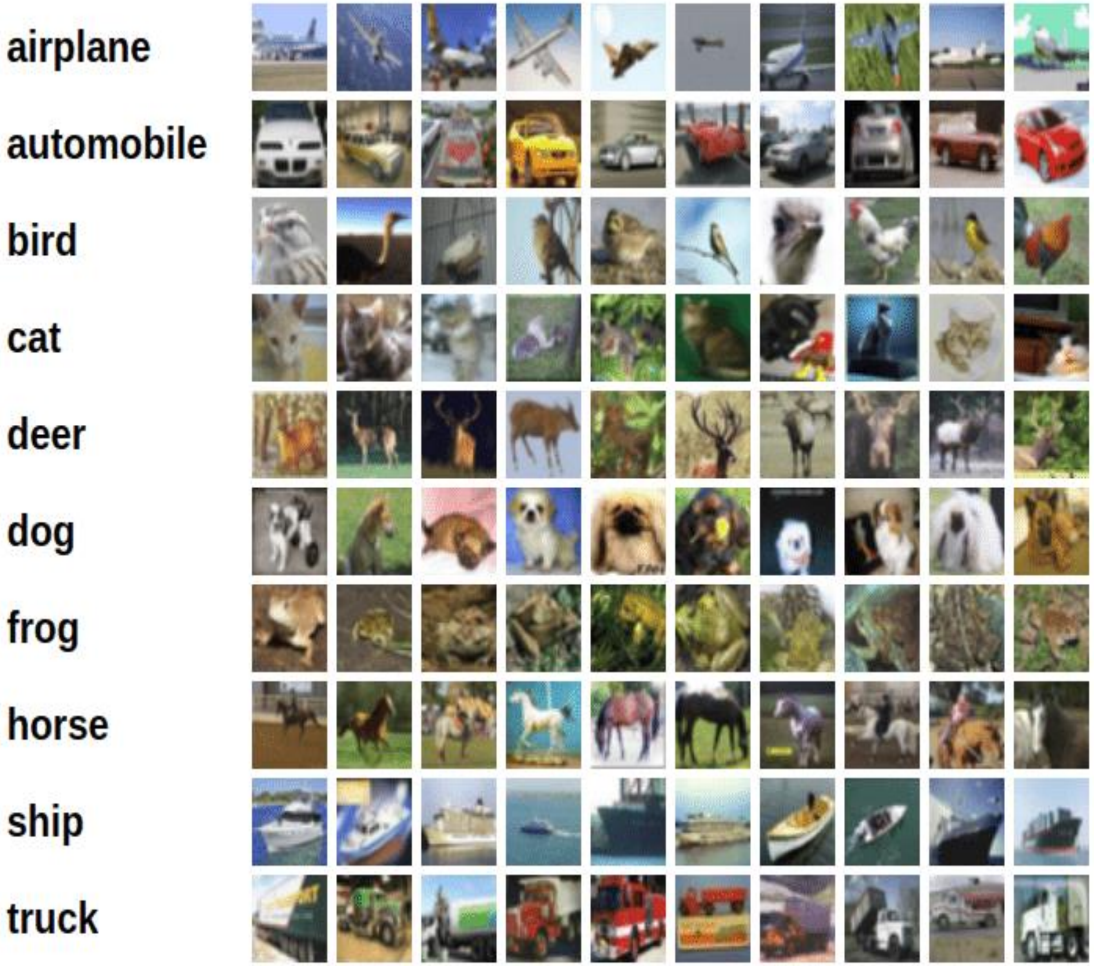
Compare and calculate

Compare results and capacities of different BNN and Calculate / Estimate local entropy

PRE-PROCESSING

Dataset

Cifar10



Selected classes

Dog

Deer

Transformations

- Standardization
- Dimensionality reduction with a convolutional layer
- Binarization
- Reshaping

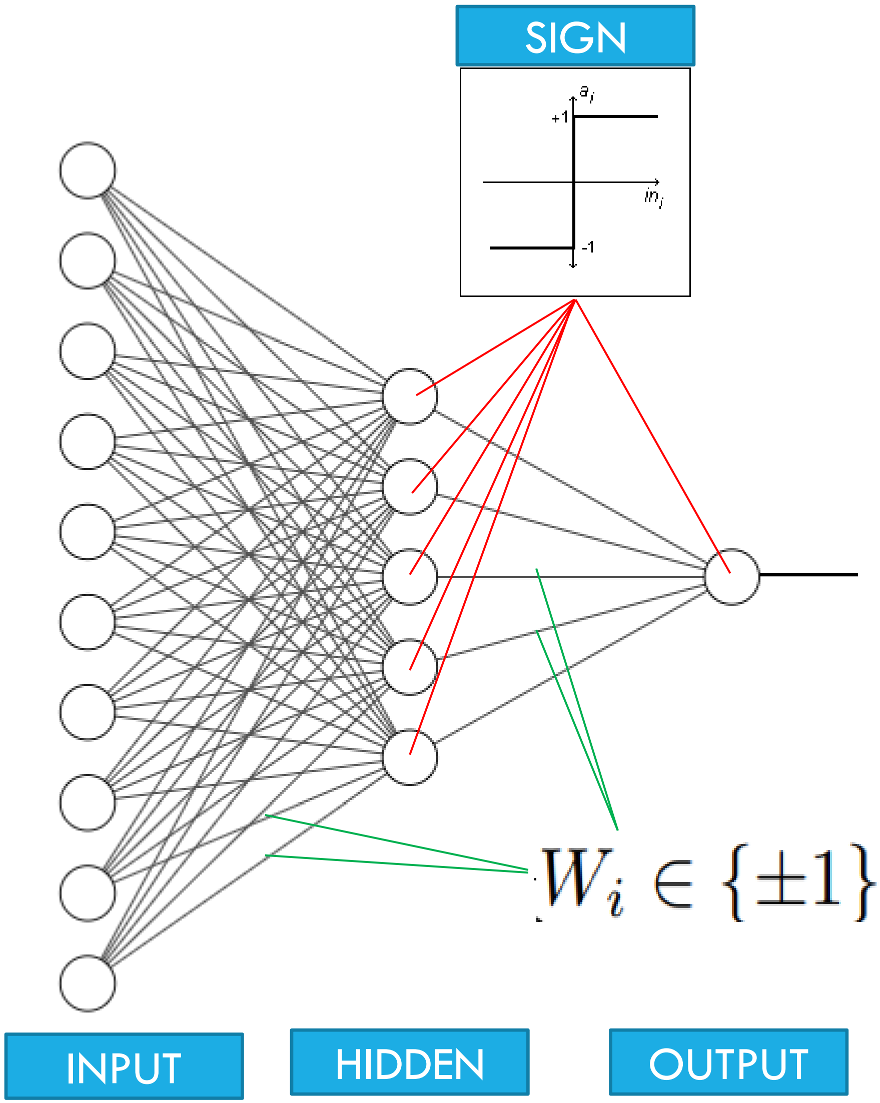
Binarized Neural Networks

Why Binarized Neural Networks?

Binarized Neural Networks (BNNs) are one solution that tries to reduce the memory and computational requirements of DNNs while still offering similar capabilities of full precision DNN models.

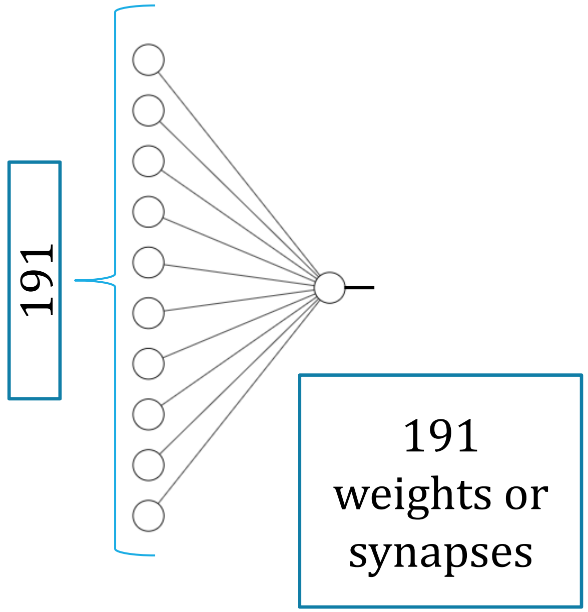
Which methodology

BNN methodology first proposed by Courbariaux et al. in 2016, where both weights and activations only use binary values, without bias.

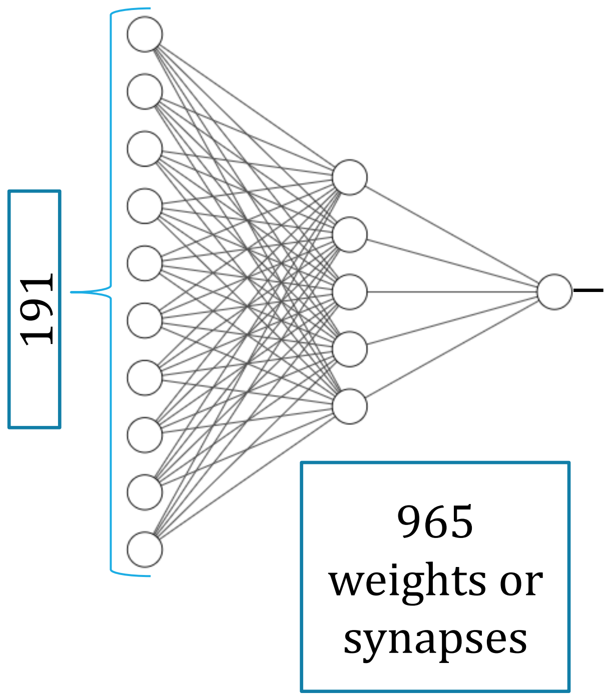


NEURAL ARCHITECTURES

Perceptron



Shallow 5x1

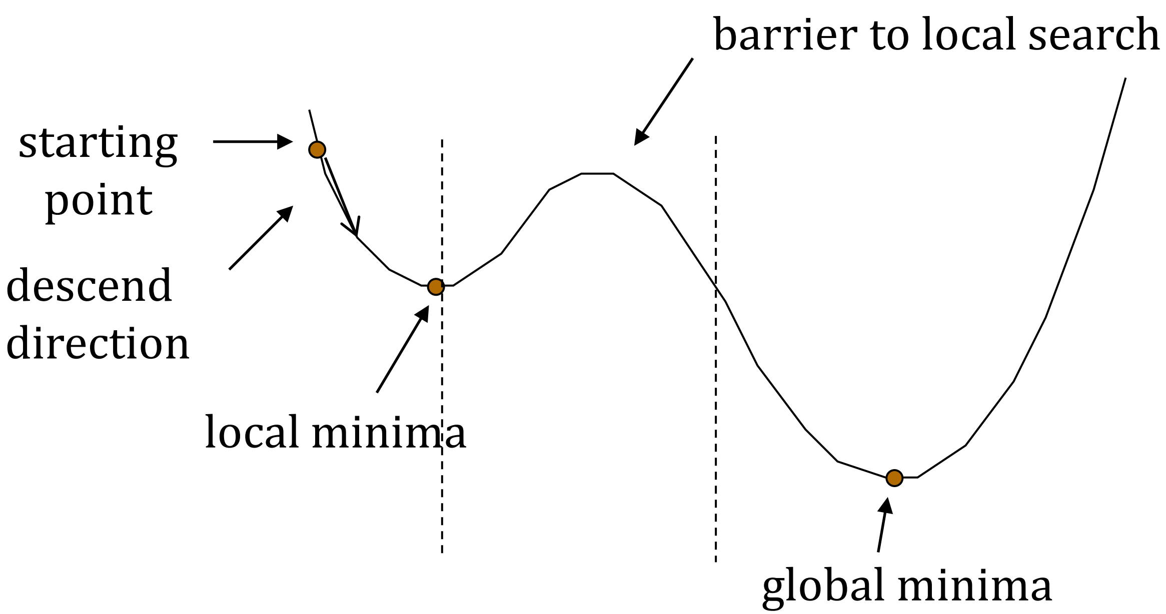


SIMULATED ANNEALING

Intuition

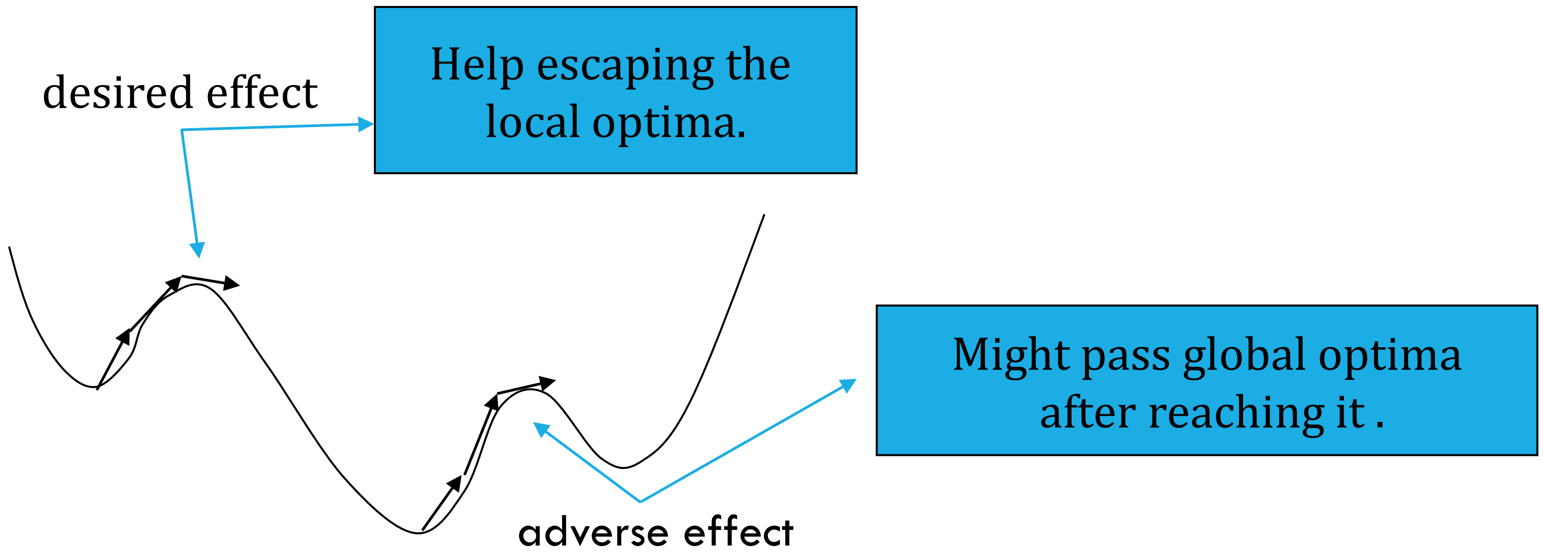
Why not use a steepest descend method?

Local search techniques, such as steepest descend method, are very good in finding local optima but difficulties arise when the global optima is different from the local optima. Since all the immediate neighboring points around a local optima is worse than it in the performance value, local search can not proceed once trapped in a local optima point. We need some mechanism that can help us escape the trap of local optima.



Intuition of Simulated Annealing

Origin of the algorithm	The annealing process of heated solids.
Intuition	By allowing **occasional ascent** in the search process, we might be able to escape the trap of local minima.



Control of Annealing Process

Define as Δ the performance change in the search direction and T as the temperature.
Es. Δ = $Loss_{t+1}$ - $Loss_t$

ACCEPTANCE OF A MOVE

$\Delta \leq 0 \quad ?$

YES

ACCEPT THE MOVE

NO

ACCEPT THE MOVE IF:
$\exp(-\Delta/T) > random[0,1)$

Cooling schedule

- T is the annealing temperature and controls the frequency of acceptance of ascending moves.
- We gradually reduce temperature T.
- At each temperature, search must proceed for a minimum number of steps, S and up to a maximum number of proposed moves.

Simulated Annealing pseudocode

```
Initialize(model, model2)
Select a Factor, a Temperature initial value, MNM, MNPM
list_Temperatures = [1.0 / (Factor**h) for h in range(5000)]
E = random_value
Err = random_value
contatore1 = 0
while Err > 0.0 and contatore1 != len(list_Temperatures)-1 :
        T = list_Temperatures[contatore1]
        contatore1 += 1
        moves = 0
        proposed_moves = 0
        while Err > 0.0 and moves < MNM and proposed_moves < MNPM:
                Err = 1-accuracy_score(ytrain, model(xtrain))
                E = - matthews_corrcoef(ytrain, model(xtrain))
                n = randrange(number_weights_network)
                proposed_moves += 1
                model2(weights) = copy.deepcopy(model(weights))
                flip(weight n in model2)
                Err1 = 1-accuracy_score(ytrain, model2(xtrain))
                E1 = - matthews_corrcoef(ytrain, model2(xtrain))
                Delta_E = (E1 - E)
                if Delta_E <= 0:
                        moves += 1
                        model(weights) = copy.deepcopy(model2(weights))
                elif np.exp(- Delta_E / T) > np.random.rand():
                        moves += 1
                        model(weights) = copy.deepcopy(model2(weights))
                E = - matthews_corrcoef(ytrain, model(xtrain))
                Err = 1-accuracy_score(ytrain, model(xtrain))
                if Err == 0.0:
                        break
```

Initialize two BNN of the same size

Temperature Initial Value

Initialized to random values > 0

- MNM: minimum number of moves for every value of T (1001)
- MNPM: maximum number of proposed moves for every value of T (3e5)
- Factor: factor decay of T

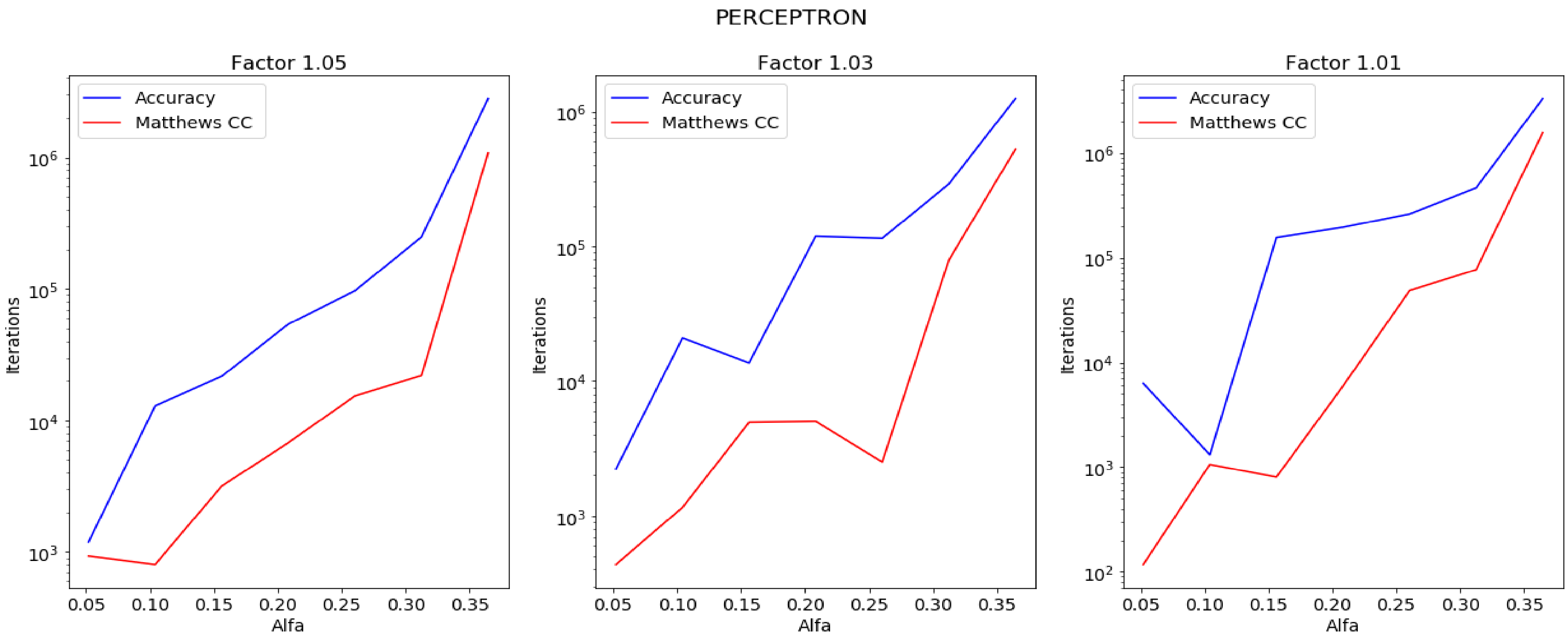
% of miss-classified samples, when 0 all samples are correctly classified

Our Loss function, in this example is Matthews Corr. Coeff. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html

Deleting these lines the algorithm becomes very greedy

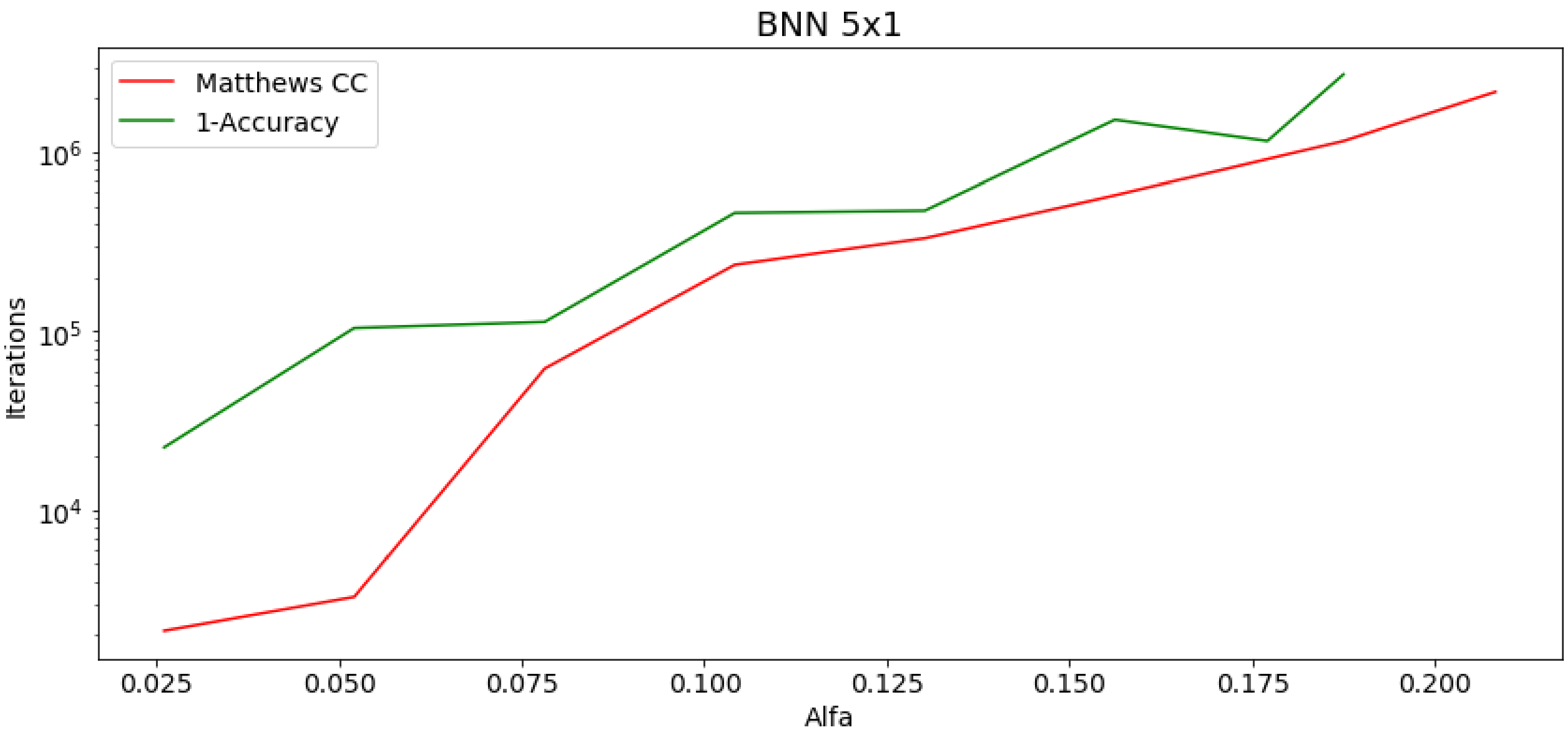
Perceptron capacity

- Define as α ratio between fitted patterns and total number of weights in the BNN. Es. for Perceptron $\alpha = \dfrac{Fitted\ patterns}{191}$
- Remember that Factor is the factor decay of the Temperature

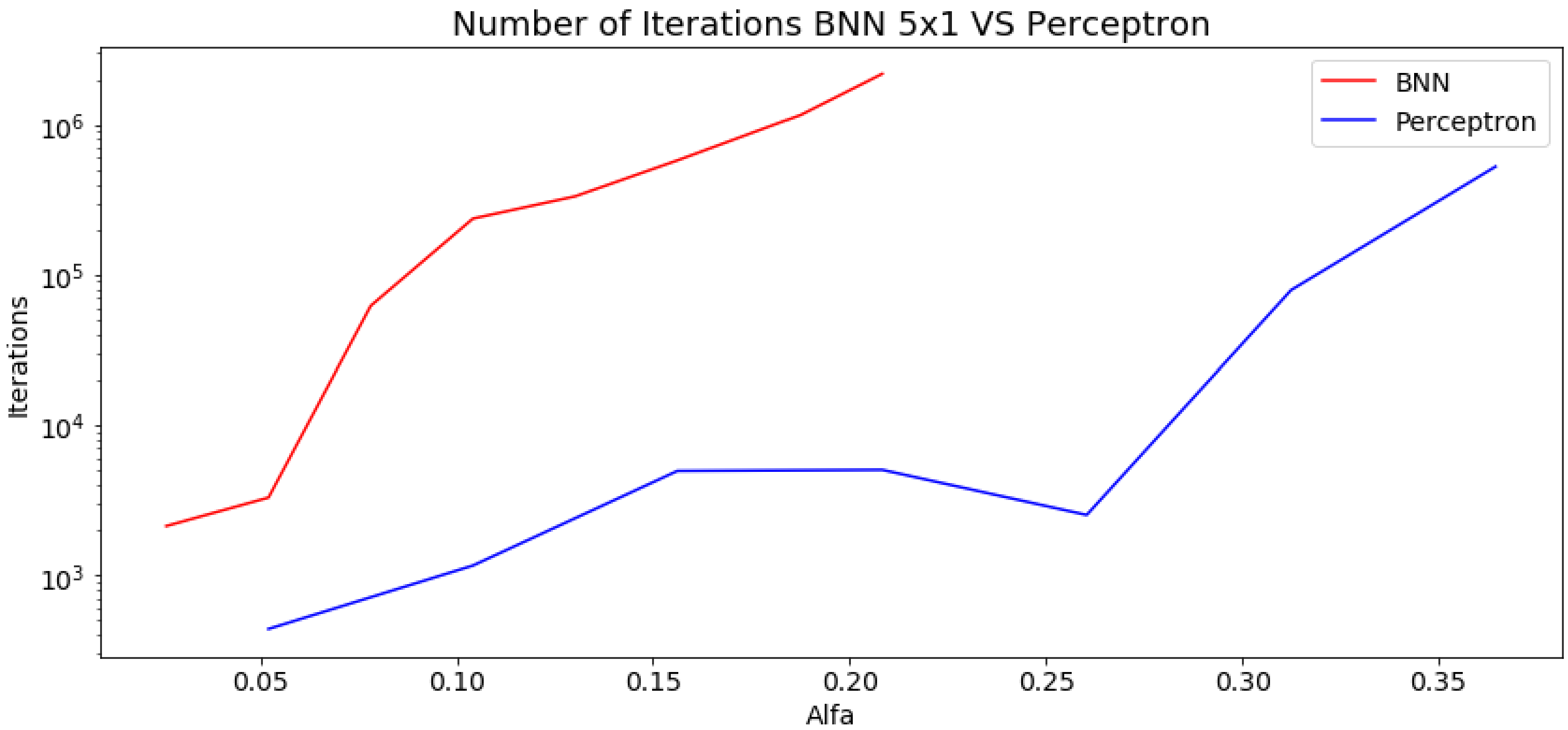


BNN 5x1 capacity

- Define as α ratio between fitted patterns and total number of weights in the BNN. Es. for BNN $\alpha = \frac{Fitted\ patterns}{960}$



Capacity BNN 5x1 vs Perceptron

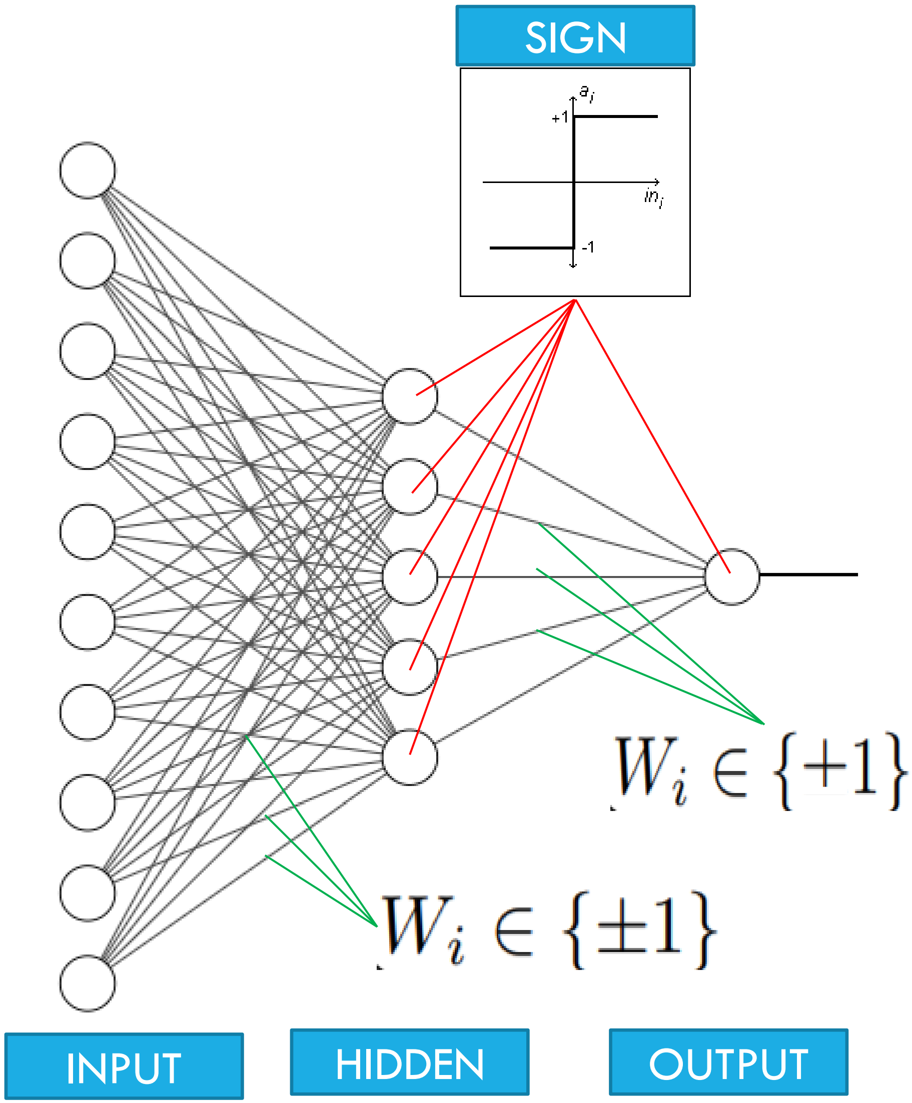


Perceptron has a higher capacity with respect to the number of weights and it also finds solutions in less iterations for a given α.

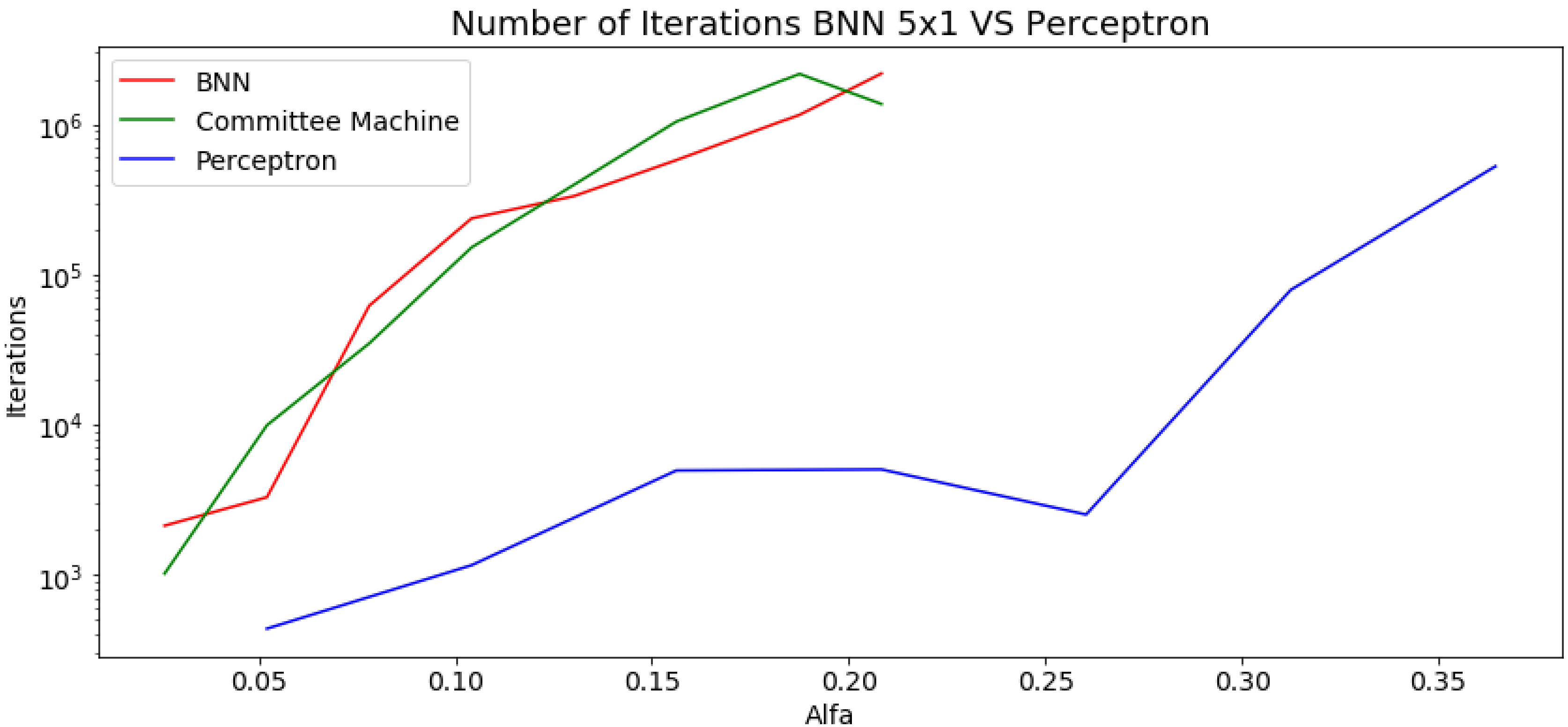
Why?

Committee machine

This is possibly the simplest version of a two-layers neural network where all the weights in the second layer are fixed to unity.

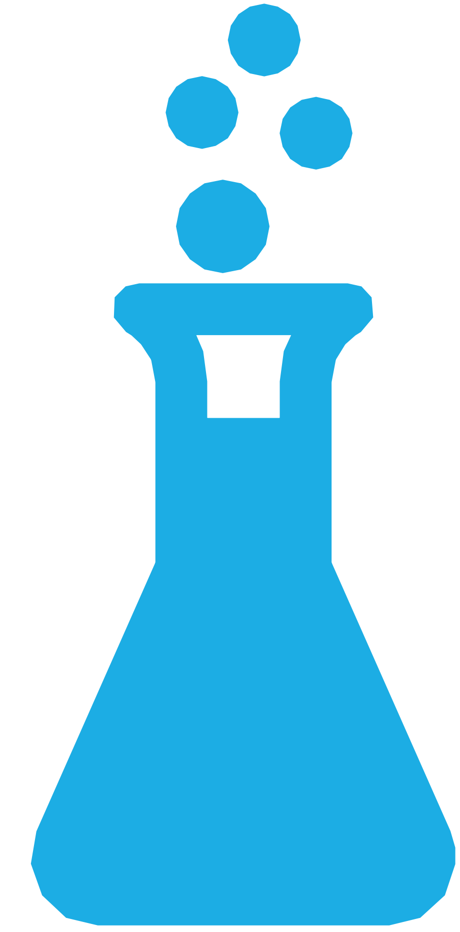


Capacity BNN 5x1 vs Perceptron vs CM



CM behaves very similarly to BNN and it does not explain why Perceptron has a higher capacity, this could be due to tuning or simply because the BNN has a higher number of patterns to fit

LOCAL ENTROPY



What is Local Entropy?

The **Local Entropy** is a function the number of solutions within a given radius from the reference solution

$$\mathcal{N}\left(\tilde{W}, d\right) = \sum_{\{W\}} \mathbb{X}_{\xi}\left(W\right) \delta\left(W \cdot \tilde{W}, N\left(1 - 2d\right)\right)$$

$\tilde{W}$ is the weight vector of the reference solution

$\mathbb{X}_{\xi}\left(W\right)$ is 1 if all patterns are correctly classified, 0 otherwise

$\delta\left(W \cdot \tilde{W}, N\left(1 - 2d\right)\right)$ Kronecker delta, 1 if the two quantities inside are equal, 0 otherwise

$\mathscr{E}\left(\tilde{W}\right) = -\log \mathcal{N}\left(\tilde{W}, d\right)$ **Local Entropy**

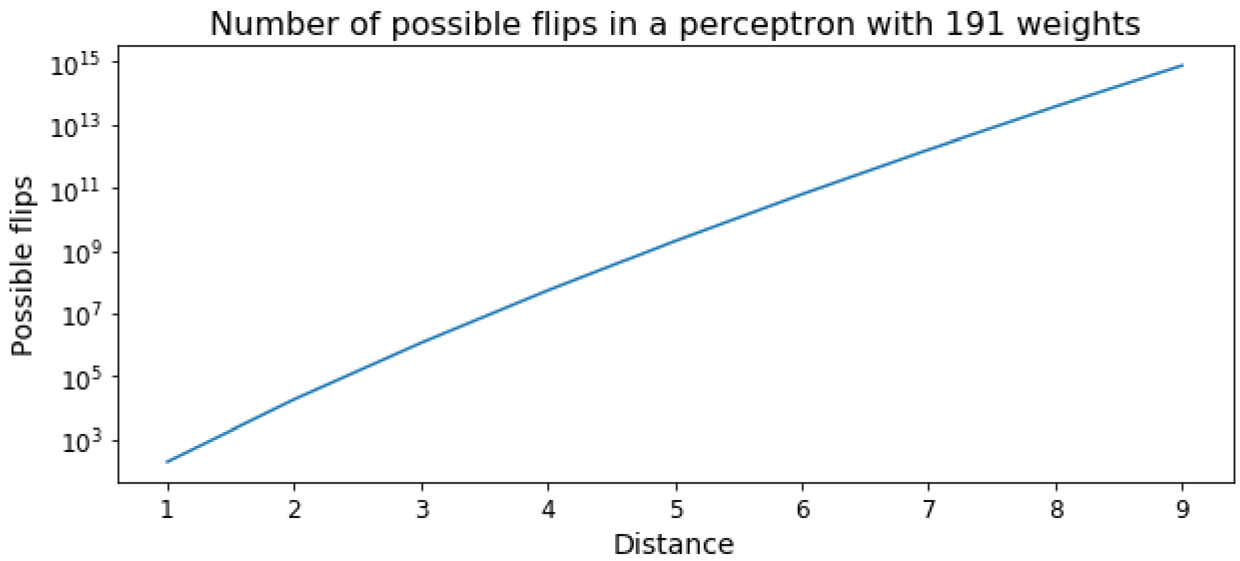
Why could be useful?

- It could be used in place of the Energy Function*
- It gives information on the structure of the minima found by the algorithm*
- Local entropy landscape is very different from the energy landscape

* Baldassi el al. 2016, see references

Drawback

Main drawback → Dimensionality → The n° of possible flips to check grows exponentially



NOTE THE LOG-SCALE IN THE Y AXIS IN THE CHART!

Local Entropy pseudocode

D is the radius

```
def Local_entropy(model,model2 ,d, factor, patterns, input_dim = 191):
    Initialize(model, model2)
    Calculate_number_weights(model, input_dim)
    assert accuracy_score(ytrain, model(xtrain)) == 1.0
    lista = [j for j in range(weights)]
    z = itertools.combinations(lista, d)
    LE = 0
    for k in range(int(comb(weights,d))):
        model2(weights) = copy.deepcopy(model(weights))
        perx = copy.deepcopy(next(z))
        for n in range(len(perx)):
            flip = perx[n]
            flip_weight(model2)
        if accuracy_score(ytrain, model2(xtrain)) == 1.0:
            LE += 1
    return LE
```

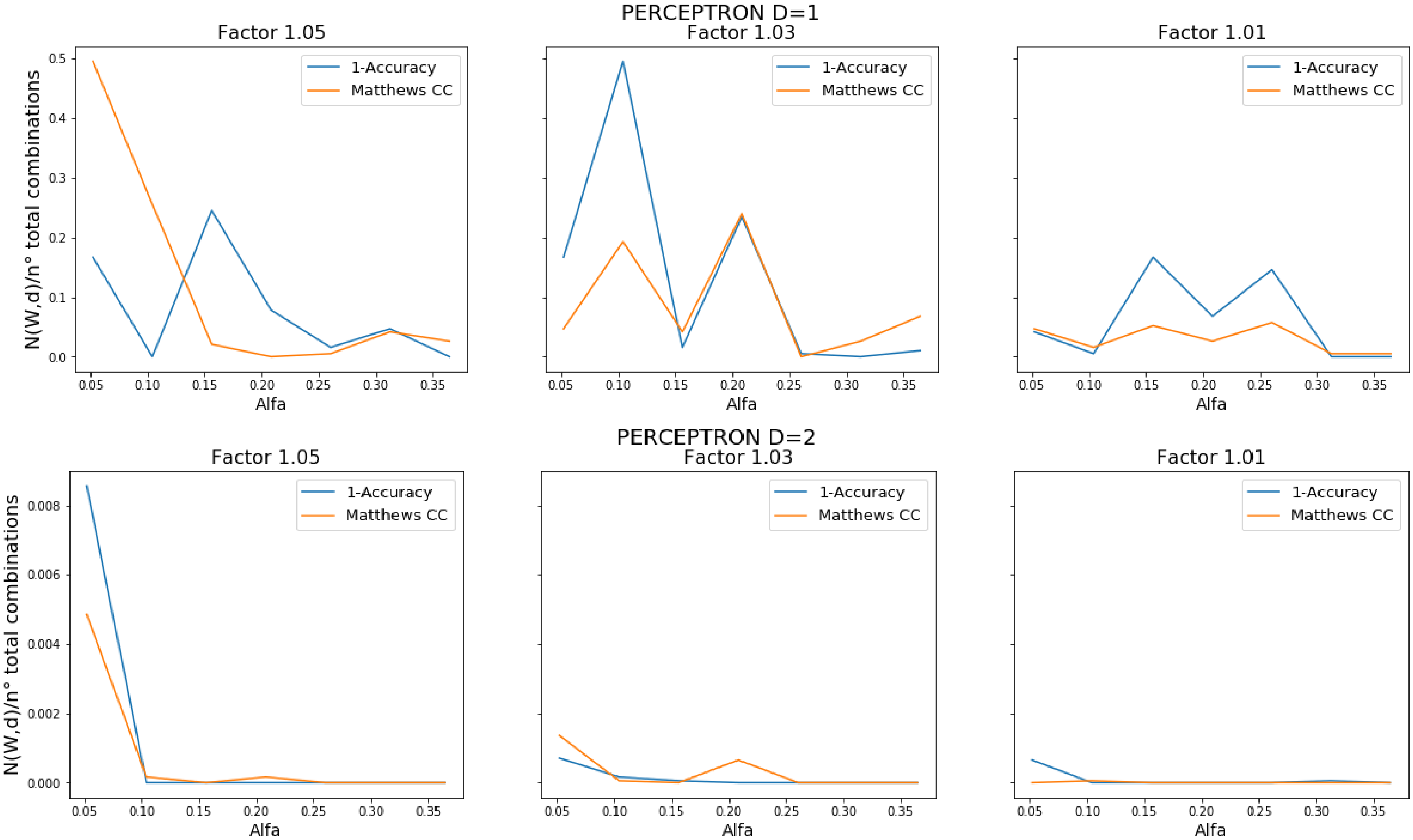
with weights of the reference solution

It creates a generator of tuples, in each tuple we have the position of the weights to flip

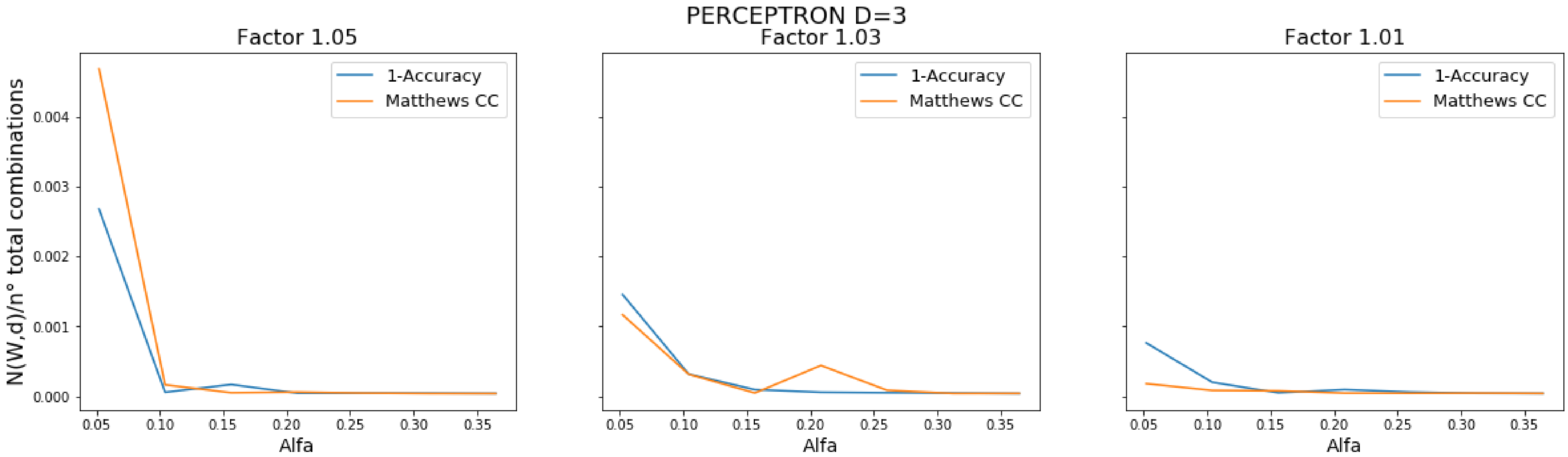
Flip the weights once at the time and then calculate the accuracy

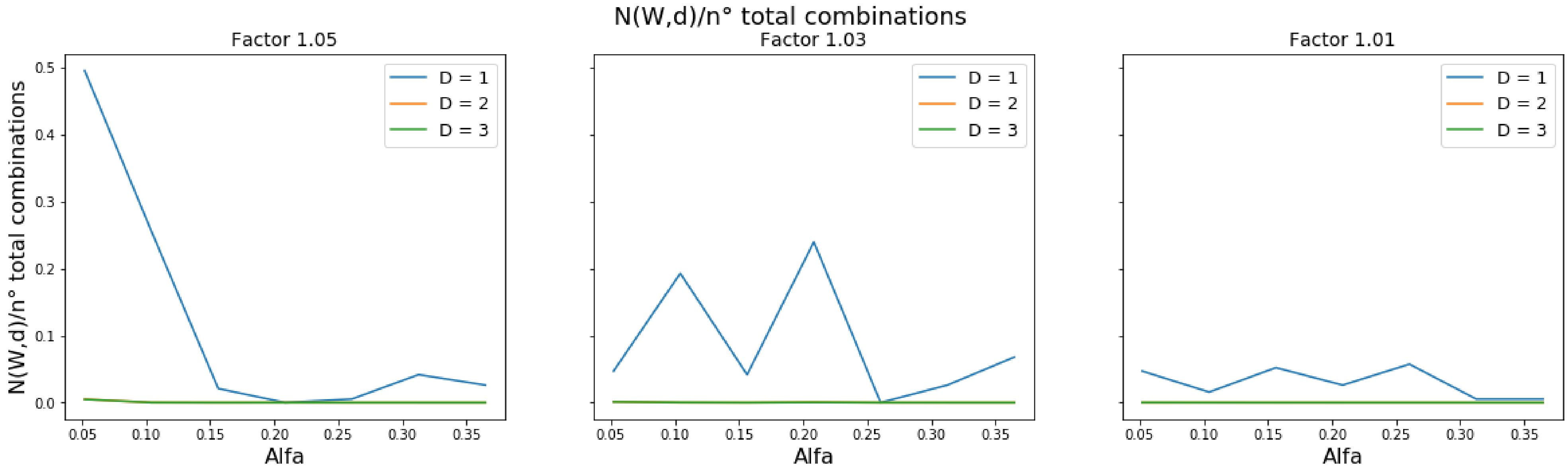
If the accuracy of the new array of weights is 1 increment the LE

Local Entropy Perceptron (1/2)

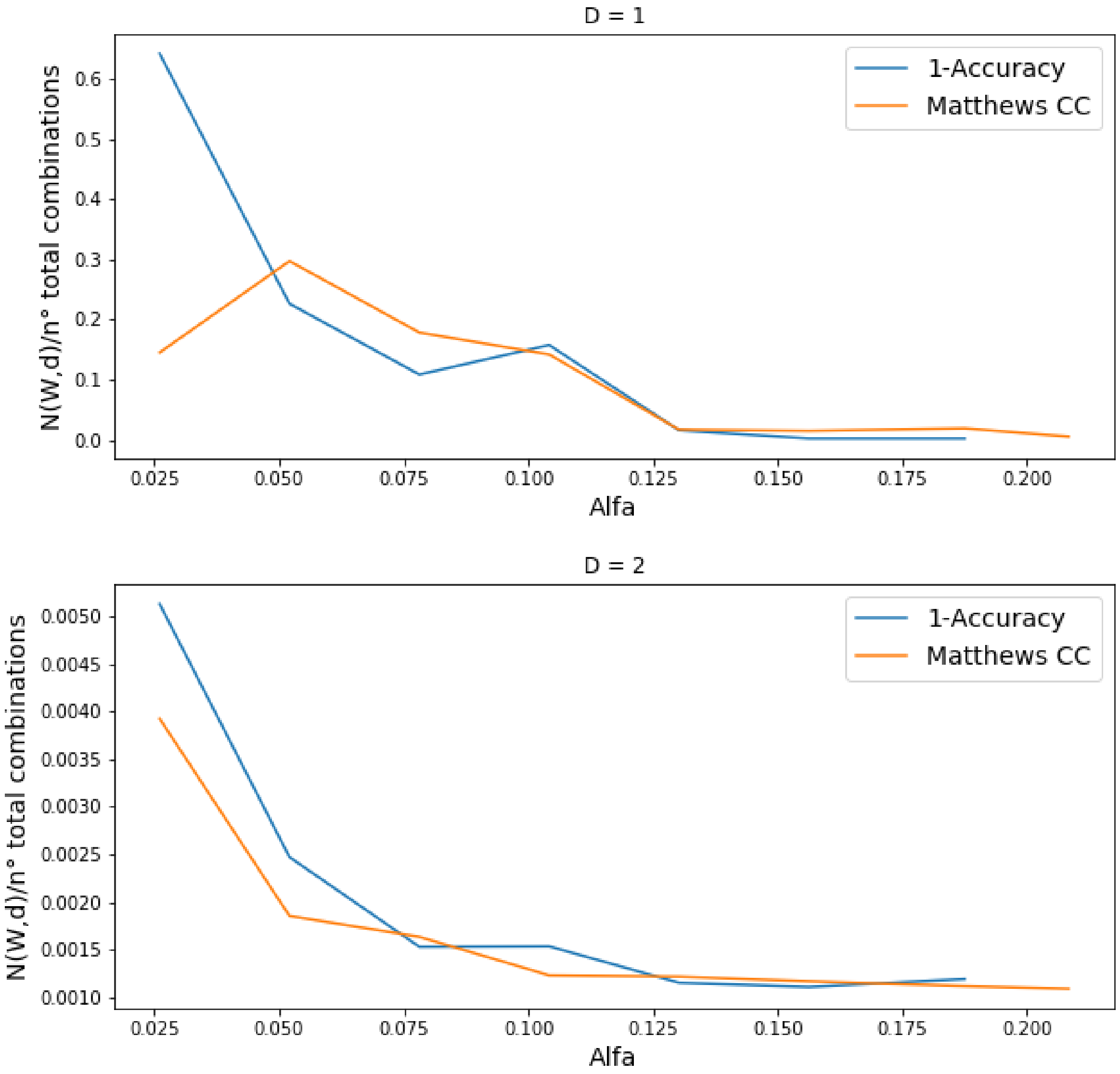


Local Entropy Perceptron (2/2)





Local Entropy BNN



Even in this framework N(W, D) at D=1 dominates N(W, D) at D=2.

Possible solution to dimensionality: MC

MAIN ASSUMPTION:
THE LOCAL ENTROPY IS UNIFORMLY DISTRIBUTED IN ALL SUBSAMPLES

DRAW A SAMPLE OF 10% FROM THE TOTAL NUMBER OF COMBINATIONS

↓

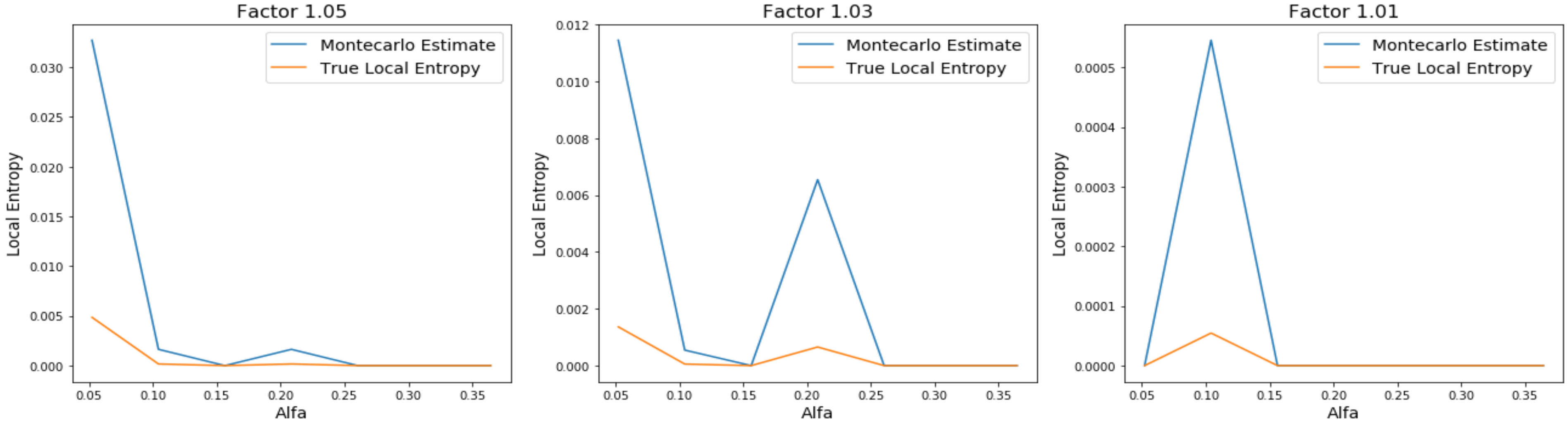
CALCULATE THE LOCAL ENTROPY IN THE SUBSET

↓

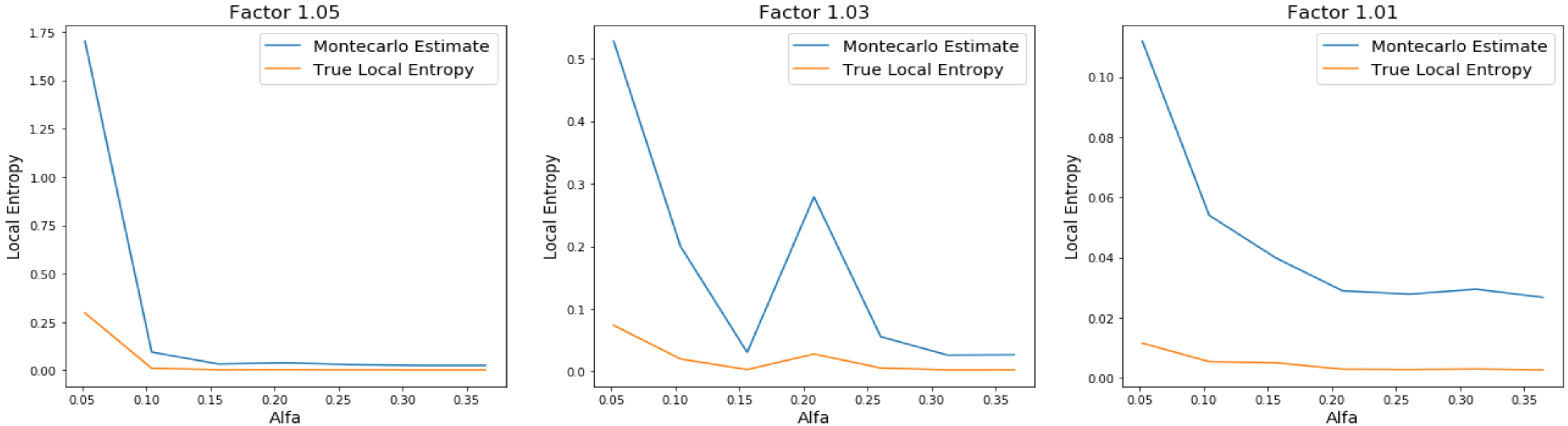
MULTIPLY BY 10 THE LOCAL ENTROPY FOUND AND OBTAIN THE ESTIMATE

Local Entropy Montecarlo

PERCEPTRON D=2

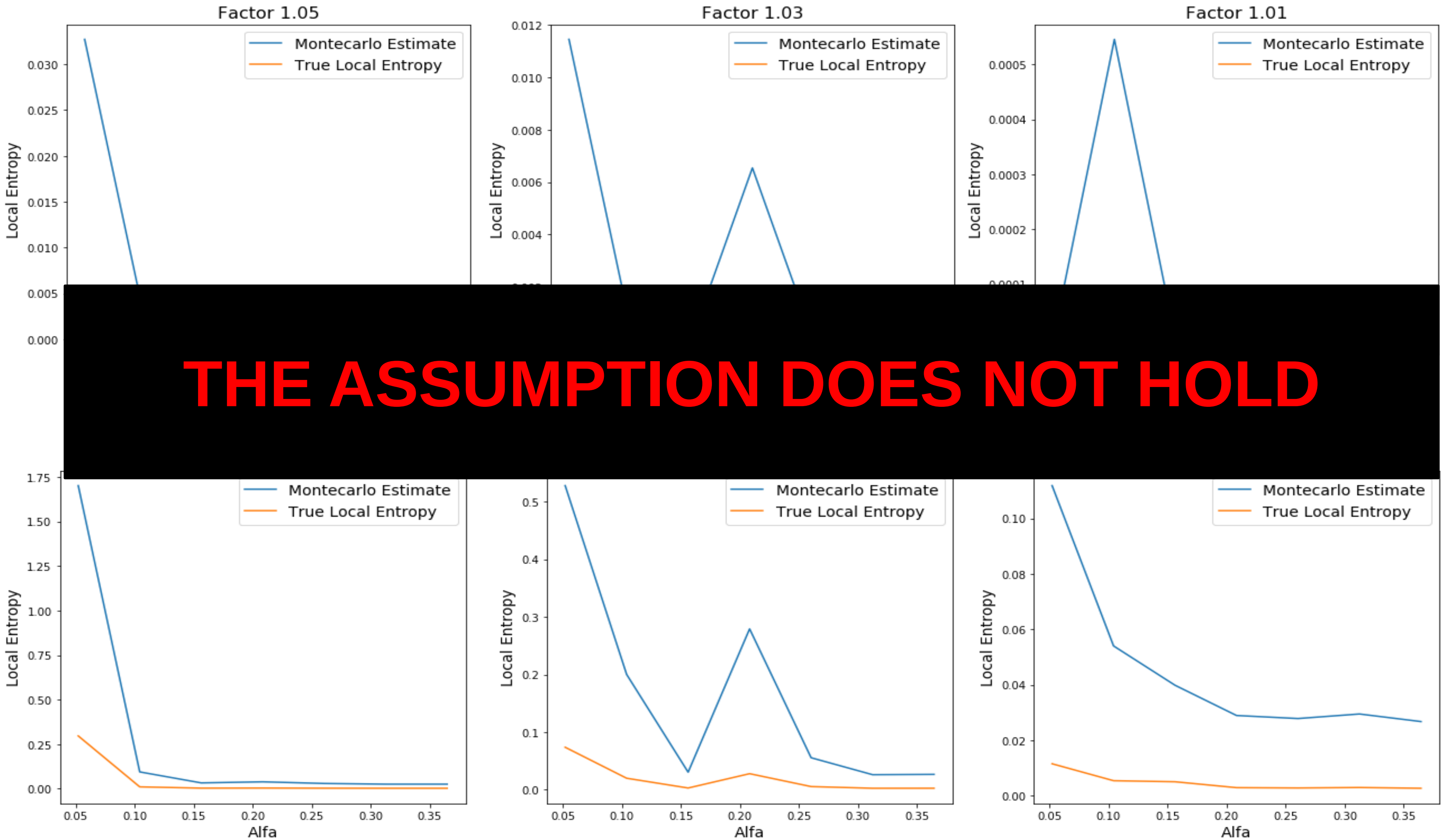


PERCEPTRON D=3



Local Entropy Montecarlo

PERCEPTRON D=2



THE ASSUMPTION DOES NOT HOLD

CONCLUSIONS

Contribution to the literature

Contributions
▪ Dataset with correlation among patterns and inside patterns (drop the restriction of random patterns)
▪ Simulated Annealing on BNN and CM (not only perceptron)
▪ Matthews CC as energy function/loss function
▪ SA with odd number of moves
▪ Capacity comparison among a perceptron, BNN and CM
▪ Local Entropy MC

Findings

Main Findings
▪ Matthews CC as energy function/loss function guarantees a convergence in a lower number of iteration w.r.t. 1-Acc
▪ Perceptron has a higher capacity for a fixed level of α w.r.t BNN and CM
▪ The SA algorithm seems to work better on perceptron in terms of capacity
▪ Local Entropy MC works very poorly in our framework

THANK YOU FOR THE ATTENTION!

References

Carlo Baldassi, Federica Gerace, Carlo Lucibello, Luca Saglietti, and Riccardo Zecchina. *Learning may need only few bits of synaptic precision*, 2016

Carlo Baldassi, Alessandro Ingrosso, Carlo Lucibello, Luca Saglietti, and Riccardo Zecchina. *Local entropy as a measure for sampling solutions in Constraint Satisfaction Problems*, 2016

Benjamin Aubin, Antoine Maillard, Jean Barbier, Florent Krzakala, Nicolas Macris and Lenka Zdeborová. *The committee machine: Computational to statistical gaps in learning a two-layers neural network*, 2019

Matthieu Courbariaux, Yoshua Bengio, Jean-Pierre David. *BinaryConnect: Training Deep Neural Networks with binary weights during propagations, 2016*

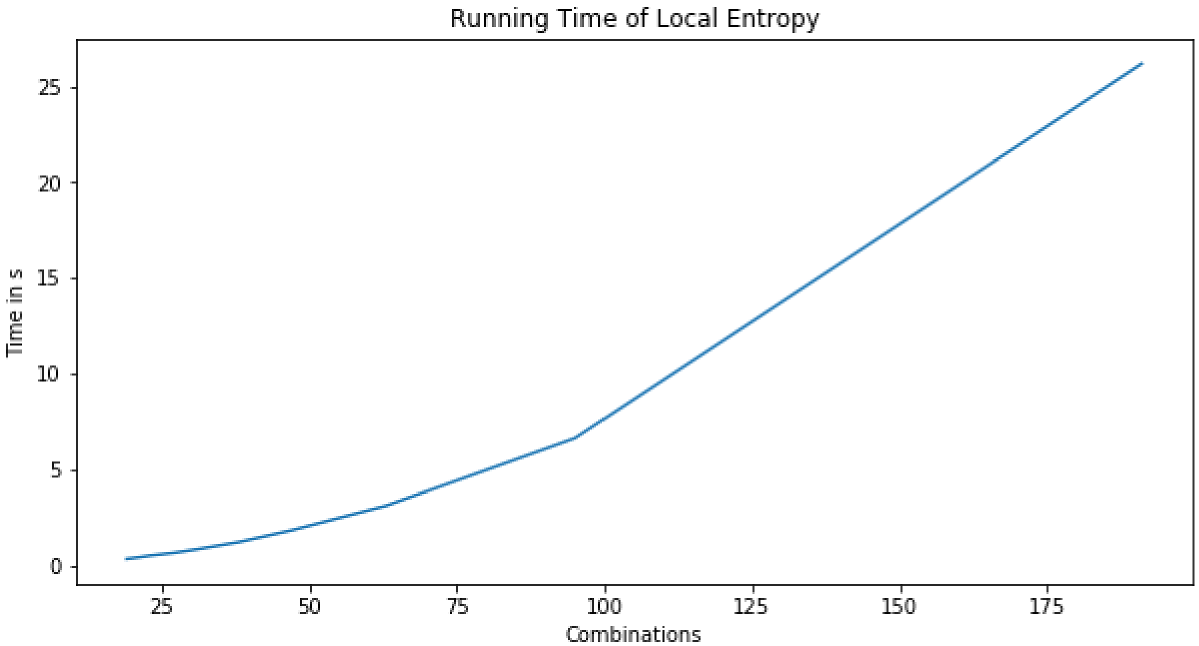
Alfredo Braunstein, Riccardo Zecchina. *Learning by message-passing in networks of discrete synapses,* 2015

Carlo Baldassi, Christian Borgs, Jennifer Chayes, Alessandro Ingrosso, Carlo Lucibello, Luca Saglietti, Riccardo Zecchina. *Unreasonable Effectiveness of Learning Neural Networks: From Accessible States and Robust Ensembles to Basic Algorithmic Schemes, 2016*

APPENDIX

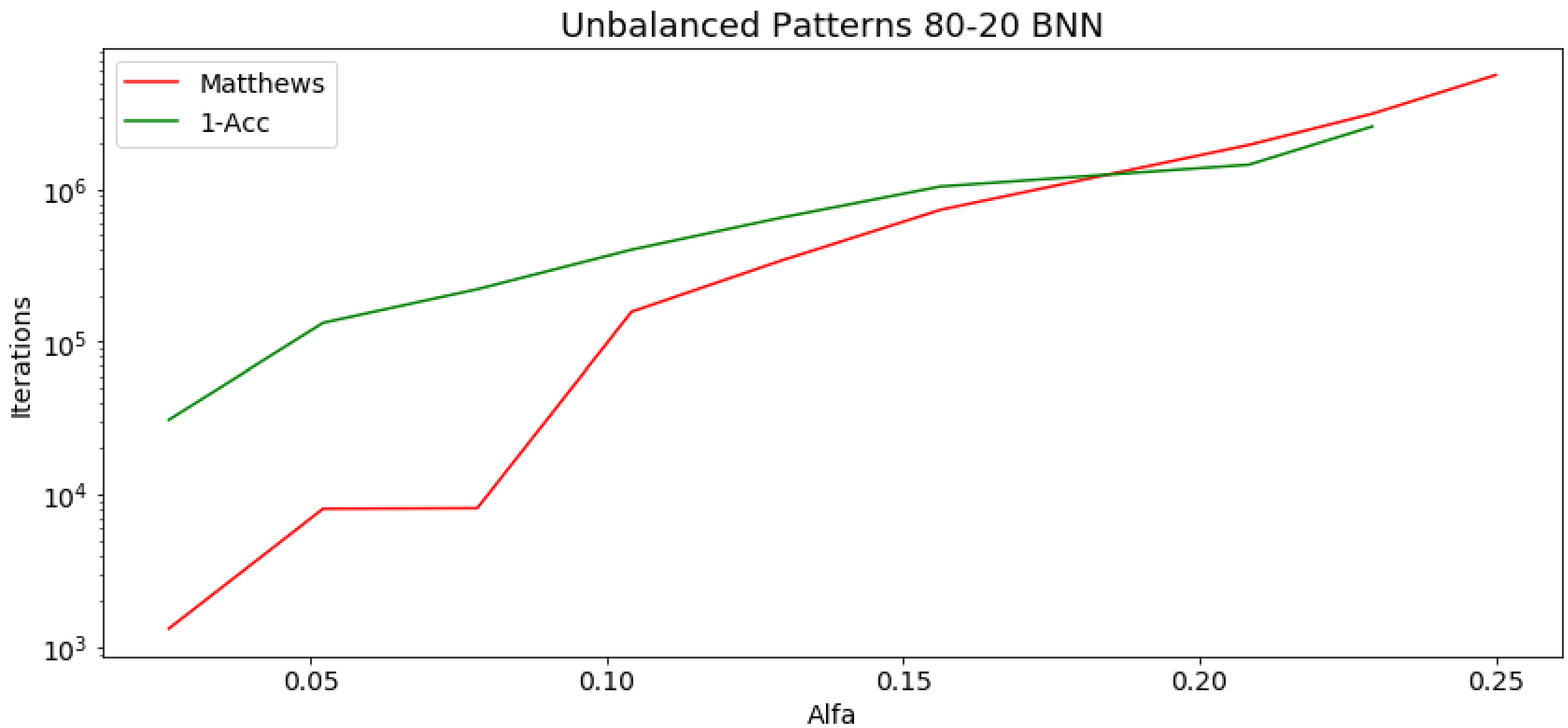
Running time of Local Entropy alg.

Running time of Local Entropy algorithm with N number of possible combinations



It runs approximately in linear time O(N)

Capacity results with unbalanced classes



Matthews CC allows to reach a higher alfa and usually reaches the solution in a lower number of iterations.

Matthews correlation coefficient formula

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

TP, TN = true positives, true negatives

FP, FN = false positives, false negatives

Test Accuracy BNN