Many thanks to all reviewers for their useful comments, which will improve the final version of the paper. We are very happy that the novelty, simplicity, effectiveness, significance, and generality of our method has been appreciated. We will address all the comments about the related work, the tables, and the details of the datasets in the final paper.

@R1: Motivation of loss function? @R2: Slightly different toy problems? @R3: Using max has already been proposed by other methods (e.g., DeepGO). @R3: Why MCLoss is better than BCELoss ($\mathcal{L}$)? While the max function has already been used, nobody so far has shown how to deploy it *effectively*, which is what the combination of MCM and MCLoss does. Experimentally, the ablation studies in Table 4 show that MCLoss allows for *consistent* higher performance than $\mathcal{L}$. In general, consider a class A with ancestors $A_1 \dots A_n$ in the hierarchy. The higher n the more likely it is that a neural network (NN) with MCM trained with $\mathcal{L}$ will remain stuck in bad local optima. Indeed, consider a datapoint such that $h_A > h_{A_1} \dots h_{A_n}$, $y_A = 0$, $y_{A_1} \dots y_{A_n} = 1$: then $\mathcal{L} = -\ln(1-h_A) - \sum_{i=1}^{n} \ln(h_A) = -\ln(1-h_A) - n\ln(h_A)$, while $\text{MCLoss} = -\ln(1-h_A) + \sum_{i=1}^{n} \text{MCLoss}_{A_i}$. Notice that, in the GO datasets, it is common to have $n > 10$, given that the hierarchies have 13 levels, and each class can have more than one parent. To visualize the negative impact of using $\mathcal{L}$ instead of MCLoss, consider the leftmost figure with 9 rectangles named $1 \dots 9$. Further, assume (i) we have classes $A_1 \dots A_9$, (ii) that a datapoint belongs to A_i if it belongs to the ith rectangle, and (iii) that A_5 (resp., A_3) is an ancestor (resp., descendant) of every class. Thus, all points in rectangle 3 belong to all classes, and if a datapoint belongs to a rectangle, then it also belongs to class A_5.

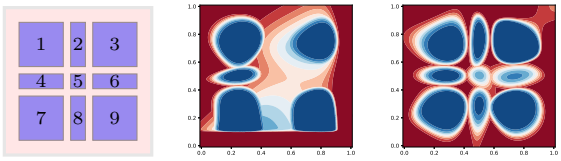

Figure 1: From left to right: (i) rectangles disposition, (ii) decision boundaries for A_5 of h + MCM trained with $\mathcal{L}$, and (iii) decision boundaries for A_5 of C-HMCNN(h).

Let h be a NN with a single hidden layer with 7 neurons. Then, the average $AU(\overline{PRC})$ (and std) over 10 runs for h + MCM trained with $\mathcal{L}$ is 0.938 (0.038), while h + MCM trained with MCLoss (C-HMCNN(h)) is 0.974 (0.007). Notice that not only h + MCM performs worse, but also, due to the convergence to bad local optima, the std obtained with h + MCM is 5 times higher than the one of C-HMCNN(h): the (min, median, max) $AU(\overline{PRC})$ for h + MCM are $(0.871, 0.945, 0.990)$ while for C-HMCNN(h) are $(0.964, 0.975, 0.990)$. The figure shows the decision boundaries of the 6th best perfoming networks for class A_5 (given the even number of runs, we could not take the exact median). We will include this example and a general discussion in the paper. Thanks also to R2 for the suggestion on how to create alternative synthetic examples: we will explore such direction in the future.

@R2: The need for this method is not fully demonstrated. @R2 It is hard to believe that NNs can outperform more classical methods on such datasets. As pointed out by R1, the paper deals with a problem with a lot of significance and relevance to real-world applications, while we achieved SOTA results on most benchmark datasets used in the literature of HMC. As shown by [2], who established the current SOTA on these datasets, NNs can beat more classical methods. In the paper, we compared ourselves with the current SOTA models (one is Clus-Ens: an ensemble of predictive clustering trees), which have already proved to beat famous tree-based and kernel-based methods.

@R2: What is the added value of using a NN here? NNs have shown the ability of performing very well in different scenarios. We wanted to create a model that was broadly applicable and easy to use, and hence NNs were our first choice. Notice though that h can be any model that outputs a probability for each class and can be trained with backpropagation.

@R2: Is there a danger to get stuck in bad local optima? Our results show our model is very stable. As stated at line 205 in the paper, the std over 10 runs is very small (in the range $[0.5 \times 10^{-3}, 2.6 \times 10^{-3}]$). This is not surprising: while, in theory, gradient descent can be performed only on differentiable functions, in practice, it performs well also on non-differentiable functions (e.g., on ReLU() and max()) leading to very good performances [1]. Considering the max() function, it is differentiable almost everywhere, and software implementations of NN training (e.g., Adam in Pytorch) usually return one of the one-sided derivatives. This may be heuristically justified by observing that gradient-based optimization on a digital computer is subject to numerical error anyway [1]. Hence, in practice, the non-differentiability of a small set of points does not affect the learning algorithm.

@R3: Comparative analysis of computation time? In addition to the inference time per batch in Table 4 (appendix), we will include a table with the training times of each model. Due to lack of space, here, we report the ones for CELLCYCLE FUN (in minutes, averaged over 10 runs): (i) C-HMCNN(h): ∼6m, (ii) Clus-Ens: ∼20m, and (iii) HMC-LMLP: ∼51m. Since we do not have the code, we could not measure the times for HMCN-R and HMCN-F.

@R4, R2 Plug in our method on top of [2] and/or CNNs? We still have to try this; we will do so in future work.

@R4: Did you optimize them over the validation sets? Yes, we optimized the hyperparameters on the validation sets. We used the test sets only to report the results. We will make it clearer in the appendix.

@R5: The results are different between tables, I would expect them to be the same. We conducted the ablation studies on the validation set (see caption of Table 3), while we report the results (after re-training on training+validation set) on the test set. We will specify this in the caption of Table 2.

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, pages 188–189. MIT Press, 2016. http://www.deeplearningbook.org.
[2] J. Wehrmann, R. Cerri, and R. C. Barros. Hierarchical multi-label classification networks. In *Proc. of ICML*, 2018.