We thank all the reviewers for their useful comments and positive feedback.

- *On the 'practicality':* it depends what 'practical' means. To be concrete, it would be **possible to implement the positive scheme** (architecture and initialization) to learn parties on a laptop. This would require driving down the constants as discussed in the appendix and implementing the scheme described in Example 1. It would not be a trivial task -and not a major addition we believe- but it would be feasible (this is proposed as a senior thesis project).

Parities give an interesting case as one can't learn parities efficiently with SQ algorithms. Re R3: This goes thus beyond improving on SQ; **one could simply not implement this with SQ.** Further one could not implement this either with small enough neural nets (we need at least n^2 edges and currently depth $\log(n)$). So this stresses the power of SGD-based deep learning. If allowed to be speculative, one may wonder whether NNs and SGD could eventually become the components of general computational systems, in which case some of the mechanisms highlighted in Section 2.4 may turn useful. Having said that, our goal was not to replace all other learning algorithms with DL (our general emulation would loose efficiency compared to tailored algorithms in most cases), but rather to show that **SGD-based deep learning has universality properties that are not granted to all learning paradigms**.

- *Further background on statistical query (SQ) algorithms:* Kearns introduced these to investigate which functions could be learned when having access to 'statistics' about the function data. One particular example of statistics being the gradient on the test loss (i.e., full GD). Kearns further introduced the key parameter of the precision noise (the exact population distribution being not accessible). He then shows that not all functions are poly-time learnable with poly-precision noise in the SQ framework, with parities as the prominent counter-example. This was after Minsky-Papert noticed the issue of the perceptron with parities. Our contribution shows that **parities are actually learnable** with **large enough** neural nets trained by **SGD**, with all relevant parameters being polynomial, including the precision noise (i.e., without 'cheating'). In fact, **SGD is not an SQ algorithm** (R2), because the queries are **stochastic**, i.e., one picks a random sample and not an average over the population distribution (SGD is what's sometimes called a 1-STAT oracle).

- *The vagueness around 'poly':* indeed we are in various place hand-wavy about 'poly' or 'inverse-poly'. We do not view this as a lack of rigor because **there is always only one way that is meaningful**, i.e., if the noise were to be polynomially large, it would be trivial to prove our negative results. However, we agree with the reviewers that this may not necessarily be obvious to the reader, so we will re-write the statements to be explicit on the poly terms. E.g.: if the batch-size is at least polynomially large (n^c with c large enough), the noise is at least inverse-polynomially large ($n^{-c'}$ with c' small enough), CP_∞ is at least inverse-polynomially small ($n^{-c''}$ with c'' large enough) and the NF is at most polynomially large ($n^{c'''}$ with c''' small enough), then no matter what the net initialization and architecture are, learning with accuracy $1/2 + \Omega_n(1)$ is not solvable. We will also put the exponents where we can, but some are tedious to get.

- *Regarding the precision:* it is important for this type of work to take into account the precision noise, i.e., one can achieve degenerate result if able to work with infinite precision/magnitude on the edge weights, as for SQ algorithms.

- *On the universality notion (R2):* yes the quantifiers in Theorem 1 are in the order mentioned. One is of course not given the function to be learned, but the class or distribution, and one can exploit this knowledge. Note that even in this setting, **SQ or small enough nets would not achieve such a universality.** Further, as stated in Remark 3, this implies as a direct corollary the result in the appendix that removes the knowledge of the class/algorithm and requires only a bound on the polynomial complexity: one can build a net that matches the best asymptotic performance of any algorithm that uses at most n^c time and n^c samples for any known c, **without knowing the algorithm in advance.** We will move this result up in the main part with the extra page (if accepted). Further, it is necessary to know c because given a neural net of size $O(n^{c'})$ we could just pick a function that requires a net of size $\Omega(n^{c'+1})$ to compute.

- *Some 'insight' on the GD failure (R2):* Conceptually, when one trains a neural net using gradient descent one is comparing the accuracy of the current net with the accuracies of the nets that would result from perturbing an edge weight slightly, and then changing the weights in the direction of improved accuracy. This commonly works in practice because most functions that one wants to learn are correlated with functions that the net is likely to compute. In such cases full gradients are typically beneficial. However, **this intuition breaks down for certain functions like random high-degree monomials:** functions that significantly correlate with random parities have vanishing probability. So, if we try to learn such a random function using GD to train a neural net, it is likely that neither the original net nor any of the nets resulting from shifting one edge's weight will be significantly correlated with the desired function, and the full gradients will be almost independent of the function. Instead one needs to extract more details about the function on individual samples in order to succeed with such functions, as the Gaussian elimination or SGD algorithms permit.

R2: No, GD with a small learning rate would not succeed in the considered setting. With an exponentially large learning rate we could have the component of the gradient that actually corresponded to the parity we were trying to learn be nonnegligable. We thus have to handle this technicality for the proof to hold, even though this is not a very relevant scenario for applications. **For smaller learning rates, the failure is less difficult to obtain** and is part of our result.