We thank the reviewers for their detailed reviews and constructive feedback. We will address your questions and concerns below (due to space constraints, we focus on the main concerns):

Generalization (R1, R3). The key insight from the generalization bound is that the generalization gap depends on the distance of weights to the initialization. In the paper, we have shown that for GD and NGD, the distance they move are of the same order $\sqrt{n}$ though they take very different paths. Moreover, we showed in Appendix F that GD and NGD converge to the same min-norm least-square solution in the infinite width limit. The point we intended to make is not that NGD always generalizes as well as GD, but that our provable generalization bound for NGD is as good as the known bounds for GD. It is not known how tight any of these bounds are. We will clarify this point in the final version.

Real data experiments (R1, R2, R3). Numerous other papers have compared NGD and SGD on modern neural net benchmarks in terms of both convergence and generalization (e.g. see the series of papers on K-FAC). Here are some additional experiments. Specifically, for regression on MNIST, we generated the data in the same way as in Figure 1 of the paper, but using 5000 training examples. For classification, we used the standard training-test split. For fair comparison, we removed all bells and whistles (including batch norm, data augmentation, weight decay). For GD, we don't include the momentum since previous theory papers only discussed plain gradient descent. We tuned the learning rate for GD using standard grid search. For MNIST, we used a two-layer MLP (one hidden layer) with 6000 hidden units. For CIFAR-10, we used a VGG-style network with 5 conv layers, and the filter count for each layer is $[32, 64, 128, 256, 256]$.

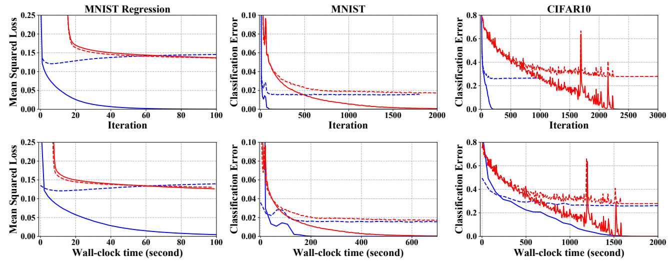

Figure 1: Red lines are GD while blue lines are NGD (Hessian-free). Solid lines are training curves while dashed lines are testing curves. NGD **converges faster** than GD and also **generalizes well**.

Stable Jacobian condition (R3). We have numerical results in our submission. In Figure 1 (page 6), we verified the stable Jacobian condition on MNIST with 100 training samples. In particular, we showed that for the over-parameterized network (in the second row), natural gradient descent matches output space gradient descent well (even with a large learning rate), indicating that the Jacobian is stable enough for the output space path to be nearly linear.

Removing simplifying assumptions (R1). The assumption of two-layer networks with a fixed second layer simplifies the proofs. We believe we can remove these assumptions using the techniques of Du et al., [2018b]. Since almost all of our analysis is architecture-agnostic, one only needs to check the conditions. Condition 2 was essentially verified for multi-layer and non-fixed-second-layer networks by Du et al., [2018a]. Intuitively, the conditions just require the network to behave like a linearized one, and we'd expect this to hold for wide networks of any depth.

Memory and computation costs of NGD and K-FAC (R2). Most of our paper analyzes an idealized version of NGD which practical algorithms like Hessian-free optimization and K-FAC are trying to approximate; hence, it's not intended to be a practical training procedure. A naïve implementation of exact NGD requires $\mathcal{O}(m^2)$ space to store the Fisher matrix and $\mathcal{O}(m^3)$ time to invert it (where m is the number of parameters). Expressing the pseudoinverse in terms of the Gram matrix as we do (see equation (3) in the paper) makes the costs $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$, respectively, which is much smaller for overparameterized networks. We note that this is equivalent to preconditioning the output-space gradient $\mathbf{u} - \mathbf{y}$ with the Gram matrix, which suggests a *new* way for running natural gradient descent.

K-FAC requires much less memory and computation than exact NGD — in practice, a small constant factor overhead compared with GD — and has been applied to large modern networks such as ImageNet classifiers [Ba et al., 2018, Osawa et al., 2018] and large transformers. Specifically, we only need to store and invert small matrices which has roughly the same shape as weight matrices in each layer. See Martens and Grosse [2015] for detailed discussion.

Novelty of the proof techniques (R2). While we borrowed much high-level structure from Du et al.'s analysis, several aspects of our analysis are novel. First, we significantly improve the bound by *bounding the distance of the whole weight vector*, giving the bound $\Omega(n^4)$. By contrast, the bound in Du et.al., [2018b] and Wu et al., [2019] is $\Omega(n^6)$. Second, we introduced two modular conditions, making our proofs much clearer and more general than Du et al., [2018b]. Third, we extend the results to general loss functions in Theorem 2. Lastly, we give an explicit bound for λ_0.

Why does a larger step size imply faster convergence? (R2) A larger step size doesn't imply faster convergence in general, but it does in the context of Theorem 3 and the analogous result for GD, since the convergence rate is given in terms of the step size (see lines 244-247 for short discussion). Hence, a larger bound on the step size (in the condition of the Theorem) implies faster convergence. We'll clarify this in the revised version.

Proofs for Thm 4 (R2). Because the proof for K-FAC is simliar to that of NGD, we skipped a step in the proof of Thm 4. We should have included a version of Lemma 4 (see lines 513-521), and will do so in the revision.