We would like to thank all the valuable and constructive feedback from the reviewers. Please see the following for the response, and we will make corresponding modifications in the revised manuscript.

[Reviewer # 1 and # 2] Remarks on Motivation/Comparison with Dropout. We would like to point out that AdaReg does not explicitly enforce the weight matrices to be positively/negatively correlated. Instead, we capture the correlations in weight matrices/gradients by defining a prior with tunable covariance matrices. From an optimization perspective, as the reviewer pointed out, we use a preconditioning matrix to change the curvature and reduce the condition number of the optimization problem. We choose the learned precision matrices Ω_c, Ω_r (not covariance matrices) as our preconditioning matrices. As shown in line 139-140, AdaReg encourages the covariance of the effective optimization variable $vec(W') = (\Omega_c^{1/2} \otimes \Omega_r^{1/2})vec(W)$ to be identity, which means the rows/columns of W' are encouraged to be uncorrelated. We note that W is the original weight matrix and W' can be viewed as the transformed weight matrix in the preconditioned system. As a comparison, Dropout prevents co-adaptation by randomly inhibiting the activations. Therefore, our method is orthogonal to but not contradictory with Dropout. To verify the claim, in Fig. 6 in the Appendix, we provided experiments to combine Dropout and AdaReg. The result shows further improvement ($\sim 2\%$) compared with using only AdaReg.

[Reviewer #1] Remarks on Statistical Strength. We use the metaphor of *statistical strength* to refer that, by taking into account the correlations between data/gradients, we improve the effective sample size. From an optimization viewpoint, reducing the number of hidden features will not help optimization since the condition number can still be very large. To address the raised concerns, we performed additional experiments using only 15 hidden units in the last fully connected layer (the original implementation has 50 hidden units) on MNIST with batch size 256. {Regularizing_Type/Hidden_Dim} with {L2/50}, {L2/15}, {AdaReg/50}, and {AdaReg/15} are 97.53%, 96.85%, 98.27%, and 98.26%. We see that when reducing the number of hidden units, L2 incurs performance drop while AdaReg maintains similar result.

[Reviewer # 1] Remarks on Line 46. In linear regression (see [Chapter 1, 1]), in terms of the expected mean-squared-error, the estimator of the hyperparameters obtained by empirical Bayes strictly dominates the one obtained by MLE. Inspired by this result, we explored hyperparameter learning by empirical Bayes. The hyperparameters here correspond to the "prior over weight matrix W", and the empirical Bayes framework corresponds to learning parameters of the prior from the correlations in data.

[Reviewer # 1] Remarks on Batch Size. On the MNIST dataset, for most of the methods except AdaReg and BatchNorm, we do observe that smaller batch size leads to better generalizations. The result is consistent with the observations in [2], who empirically argue that a smaller batch size corresponds to a flatter minima in convergence. We note that AdaReg is insensitive to the choice of the batch size (consistent in both MNIST and CIFAR10). From an optimization perspective, we conjecture that the preconditioning matrices (precision matrices) found by AdaReg change the curvature of the loss landscape, and it converts sharp minima to a flatter minima in the preconditioned system.

[Reviewer # 2 and # 3] Computational Overhead and Oscillation in Fig. 3. In Algorithm 1, the computation overhead lies in InvThreshold operation. First, as shown in line 189-192, InvThreshold's time complexity scales sub-quadratically in terms of number of parameters in the layer. Second, in practice, due to the nature of block coordinate descent, we perform the InvThreshold at every outer loop, where each outer loop contains an inner loop of several ($n = 25/50$) epochs for updating network parameters via first-order optimization. Hence the computational overhead due to InvThreshold (line 3 and 4 in Algorithm 1) is almost negligible compared with that of updating network's parameters (line 2 in Algorithm 1). On the MNIST dataset with batch size 256, for 300 epochs, CNN takes 3103 seconds to finish and our CNN-AdaReg takes 3172 seconds to complete training.

As explained in our block coordinate descent algorithm, we update covariance matrices in the outer loop which contains $n = 25/50$ inner loops for updating the model parameters. The outerloop-innerloop update causes the oscillation in the optimization trajectories in Fig. 3. We will clarfify this in the revised paper.

[Reviewer # 2] Remarks on the Experiments. We conducted additional experiments with AdaReg on deep residual net (18 layers) for CIFAR-10. This model contains Batch Normalization and Weight Decay as in the original paper. The results are as follows: The performance before applying AdaReg is 93.02% while applying AdaReg gives us 95.04%. In all of our experiments, the standard regularization techniques (L2, BN, DeCov, and Dropout) are applied to all the layers. We have also provided experiments of applying AdaReg to all the layers in Fig. 7 in Appendix, where we do not observe much difference as comparing to applying it only on the last layer.

We also provide additional experiments of applying L2+BN. On MNIST with batch size 256, L2 only reaches 97.53%, L2+BN reaches 97.72%, and AdaReg reaches 98.27%. Although we do see improvements for combining L2 and BN as comparing to L2 only, AdaReg still performs the best.

[1] Large-scale Inference: empirical Bayes methods for estimation, testing and prediction. Efron, Bradley, 2012.
[2] On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. Keskar et al., ICLR 2017.