[Reviewer #3] Thank you for your valuable comments and suggestions.

"Training efficiency comparison": 1) Because splitting GD and pruning methods work in a very different fashion, it is difficult to draw a concrete timing comparison. However, generally speaking, the pruning methods need to start with a large network whose training cost is high, and is hence very costly when the goal is to learn small-size networks (for e.g., energy efficiency). In comparison, our splitting strategy only requires to incrementally train small-size networks and is hence more preferred for learning light-weight neural architectures. Our experiments show that splitting method can also obtain higher accuracy with the same network size compared with prunning methods; this is in sharp contrast with the common believe that it is critical to take advantage of the knowledge of the pre-trained large networks.

2) With the gradient descent approximation and auto-differentiation trick shown in Appendix C.3, we can now implement the eigen-calculation of the splitting matrices very efficiently, with a time complexity comparable to the regular back-propagation on the same network (see more details in Appendix C.3). We will release our implementation.

3) Because the number of steps of splitting is linear (not exponential) to the size of the final network, our approach is *much faster* than the existing neural architecture search (NAS) methods that require random or brute force search in an exponentially large model space. We will add more discussion on the time efficiency in the revision.

[Reviewer #5] Thanks for your valuable time and comments. We do respectfully disagree with your rejection based on concerns on computational cost and convergence, with SGD on over-parametrized networks as the baseline.

Our work is designed for learning small network structures: Although SGD on over-parameterized networks have good empirical and theoretical properties, they yields large and redundant networks that are slow and costly in the inference phase. The goal of our algorithm is to learn small network architectures that are fast and energy efficient in the testing phase, for resource constrained settings such as mobile and IoT. Therefore, our method should be compared with existing methods of the same purpose, including (1) pruning-based methods, which obtain small networks by trimming or compressing a pre-trained large network, and (2) neural architecture search, which searches for good architectures starting from scratch by random or black-box optimization (such as evolutionary strategy or policy gradient).

Our method is fast and practical: As we elaborate in the response to Reviewer 3, our method has significant advantage over pruning-based and NAS methods. We believe that our method yields one of the fastest strategy for efficient neural structure optimization. *We will release our code to demonstrate this after acceptance.* Further comments:

i) We use regular mini-batch gradient descent to train networks with fixed structures. The convergence can be quickly checked by averaging over a large number of min-batches (and it just takes one epoch of gradient computation even if we average over the whole dataset). In fact, because the effect of splitting and parametric updates are decoupled as shown in Eq(4), it is fine to split even when SGD is not strictly converged.

ii) The eigen-computation is fast. First, the cost exact eigen-computation is $O(nd^3)$, where n is the number of neurons to be split and d the parameters of each neuron. This is not huge for modern machines because d is small compared with the overall parameter size. For example, a 3×3 Conv-filter with 64 input channels has $d = 64 \times 3 \times 3 = 576$. In addition, because we only need to calculate the minimum eigenvalue/eigenvector, we can use the gradient descent approximation we described in Appendix C.3 to significantly speedup the computation. Our current implementation allows us to calculate the eigens of all the neurons jointly (without actually expending the matrix $S(\theta)$) with a cost roughly equivalent to back-propagating on the same network. We will release our implementation.

iii) For dealing with dynamic growing weight matrices, deep learning frameworks such as Pytorch support dynamic computational graph, which allows us to implement our method easily in practice. See the "rethinking-network-pruning" repository by github/Eric-mingjie for an example. Once again, we will release our implementation.

Convergence Guarantees and other: Our convergence guarantee is better or at least as good as (parametric) SGD because we can improve the loss of SGD by splitting. Note that all our experiments are examples when parametric SGD has been stuck at local optimal, but splitting allows us to further decrease the loss (by escaping a "functional" saddle point). In particular, see the "cliff points" in the Fig 1(d). The normalization in $G(\theta)/||G(\theta)||$ is not substantial and impacts the step size, which we do not discuss in the work. "Non-asympotitic" refers the point that works like Mei etal is based on asymptotic of infinite number of neurons, while our method does not. We will use the term "mean field" to avoid confusion. Further clarification will be provided.

[Reviewer #6] Thank you for your valuable comments and suggestions. Our work provides a new framework of neural structure optimization which is both practical and theoretically substantial. We optimistically believe that many new practical and theoretical approaches can be developed based on our work. Please see our reply to Reviewer #3/#5 for more discussion on practical efficiency. We will release the code.