We sincerely thank all reviewers for their valuable comments and we address individual questions below.

R1: This paper only compares to the 2 years ago's work (DARTS). We actually compare with AutoGrow in paragraph **Growing Wider and Deeper Networks**, around line 218. Due to the space limit, we move the result to appendix. Moreover, AutoGrow can only grow layers so it does not compare with many well known NAS baseline. We choose DARTS because it is still a very popular baseline in recent NAS papers, which gives new papers a fast, gradient based, weight-sharing NAS baseline with a low error rate to compare with.

R1: The paper only experiments on CIFAR-10/100 but not on Imagenet. At least it can be validated on NAS-Bench-101,201. Thanks for the suggestion, we have compared the firefly method with similar weight sharing baseline methods on NAS-Bench-201 in Table 1.

	RSPS	DARTSV1/2	ENAS	SETN	GDAS	Ours
Acc.	84.07	54.30	53.89	87.64	93.61	93.27

Table 1: Search Result on NAS-Bench-201

We also want to point out that firefly achieves very good results on continual learning (CL), outperforming the best known dynamic architecture baselines (CPG, DEN) in the CL literature.

R2: ENAS/DARTS are fairly different architecture search algorithms. I want more details on how random search is performed. Thanks. In the paper (Table 1 line 240), we reported the random search results from the DARTS paper. For a more comprehensive comparison, we add a detailed random search experiment here in Figure 1 which searches different numbers of random samples and evaluates the best one on the validation set.

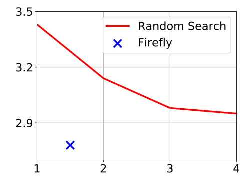

Figure 1: Err. v.s. search time (in GPU days).

R2: hyperparameters? Due to the complexity of the NAS problem, certain number of hyperparameters is unavoidable. Our method is already much simpler than alternatives such as RL-based methods and DARTS. We mainly have three key hyperparameters: how many new neurons we add, how many steps we train after adding these new neurons and how many candidate neurons we choose after training. We provided the exact numbers we used for each experiment in Appendix C.

R2: It would be nice to have an explicit related work section. We will move the related work to main content.

R3: Notations: 1) line 90 $f_t(x)$ should be the final output (a vector/scalar) of a net, but the sum is just an input to a neuron in the next layer. f_t here is a simple two-layer network that takes x and outputs a scalar; we followed the similar definition as in reference [10] in the original paper. **2) the use of ϵ and ε is ambiguous.** Thanks, we will make the usage consistent. **3) in Step Two (line 121-122), for a standard Taylor approximation, s_i should be just a Δ.** We used a different version of Taylor expansion here; note that our s_i is close to gradient $\nabla_{\xi_i} L(f_{[\tilde{\boldsymbol{\varepsilon}}_{\neg i},0],\tilde{\boldsymbol{\delta}}})$ when $\tilde{\varepsilon}_i$ is close to zero. **4) line 166-167, is $f_{1:t}(x)$ is a sequence of functions or just a function at step t?** $f_{1:t}$ is a single network at step t that combines all neurons grown from step 1 to t, and f_t is the subnetwork of $f_{1:t}$ selected by a binary mask only for task t. **5) "the candidate set of f_{t+1} should consist of" is confusing. Why a set is just a function?** Thanks, we will make it explicitly refer to the set of functions indexed by parameters $\boldsymbol{\varepsilon}, \boldsymbol{\delta}$.

R3: Clarifications: We will improve the clarity based on your suggestions. **1) line 116, when optimizing ε and δ, are neural network weights also updated?** No, the network $\boldsymbol{\theta}$ is fixed. We are only learning the perturbation $\varepsilon\delta$. **2) "measured by the gradient magnitude", magnitude of full-batch or a few mini-batches?** We use a few mini-batches for estimation. **3) clarify if L is training loss or validation loss.** L is the training loss throughout the paper. **4) clarify z in Line 124.** z is a dummy variable ranging from 1 to n; here $\{(2z-1)/2n\}_{1\leq z\leq n}$ is the discretization of the continuous range $[0,1]$. **5) Make the legend labels the same with those appeared in the text. In Fig. 3(a), a should-have simple baseline: add one neuron and randomly initialize new weights** Thanks, we will make them consistent. The suggested baseline is strictly worse than the Random(split) but we will include that as well. **6) In Figure 3(b), If the splitting and growing happen at the same time, the number of neurons (markers along x-axis) should have a gap larger than 1** We pick the single best neuron from splitting and growing, so they do not happen simultaneously. **7) In Line 207, clarify the depth of the net** For VGG-19, the depth is fixed to be 19. **8) In Figure 5/Line 249, cite and clarify baselines of EWC, DEN and RCL, where are not clarified/mentioned anywhere. Moreover, why other baselines are not curves but single dots** We will include the citations. We reported the numbers for the architectures from the original papers so they are only dots. **9) The x-axis with 20 tasks doesn't match the caption "on 10-way split"** We have corrected the typo.

R4: No discussion of time/space complexity. We discussed the space complexity on line 150. The time complexity per expansion is $\mathcal{O}(N+m)$, where N is the size of the sub-network we consider expanding and m is the number of new neuron candidates. Because per expansion, we only compute the gradient for each neuron, thus the complexity is linear. We will add more detailed discussion in the revision.

R4: The method seems complicated and difficult to implement It is in fact easier to implement Firefly than other NAS approaches because 1) firefly only requires gradient estimation, for which standard deep learning libraries have APIs; 2) the search space is smaller than conventional NAS methods because every expansion is restricted to be local. The core idea of Firefly is to train the local perturbations of a given architecture which leads to the steepest descent. The entire implementation of both growing/splitting is only a few hundred lines of python code. We have built our method in an API fashion and will open source the code.