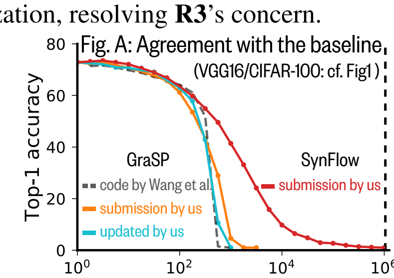
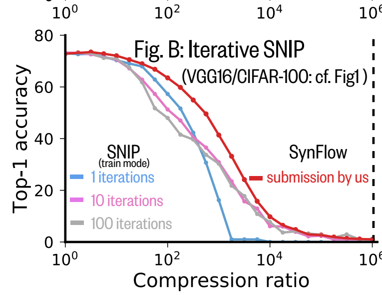
We thank all reviewers for their careful reviews and positive comments, including: (**R1**) "the method is very well-motivated with sound insights", (**R2**) "both the identification of layer collapse . . . and the SynFlow algorithm could be of interest to community", (**R3**) "the study on Iterative Magnitude Pruning is interesting and offers insights", (**R4**) "this work has a theoretical framework and solid experiments to support its arguments". We now address reviewer concerns:

Theorem 1 generalizes prior conservation laws. As we mention on L180/245, restricted versions of our conservation laws have been noted in the interpretability [39] and implicit regularization [45] literature. We will also cite [Liang et al.] as suggested by **R3**. Our theorem 1 generalizes these prior laws in three significant ways: (1) We do not limit ourselves to only gradients of activations [39] or the training loss [45], but consider any "scalar function of the output" L157. (2) Rather than proving conservation by layer only [Liang et al.], we prove conservation at the stronger neuron-level and generalize to any cut separating the input from the output (theorem 2). (3) We consider all incoming and outgoing parameters (weights and biases) allowing us to avoid the assumption that biases are zero ([45], [Liang et al.]) and understand how conservation applies to a "variety of neural network layers (e.g. dense, convolutional, batchnorm, pooling, residual)" L186. Most importantly, we are the first to connect conservation laws to network pruning and elucidate their significance in explaining a multitude of phenomena and in constructing a new pruning algorithm.

Theorem 1 holds even with biases, batch normalization, and residual connections. In Theorem 1, we consider θ^{in} to encompass *all* incoming parameters including the biases. We can understand **R3**'s confusion, so to clarify our proof, we will make the notation $z_j = \sum_k \theta^{\text{in}}_{jk}\phi(z_k)$ more explicit by designating a bias parameter $\theta^{in}_b = b_j$ and a neuron in each layer with the activation $\phi_b = 1$. We will further explain, mathematically and graphically, how our conservation laws generalize across modern architectures and at any point in training. For example, when considering a simple feedforward network with biases, then we get the non-trivial relationship: $\langle \frac{\partial \mathcal{L}}{\partial W^{[l]}}, W^{[l]} \rangle + \sum_{i=l}^{L} \langle \frac{\partial \mathcal{L}}{\partial b^{[i]}}, b^{[i]} \rangle = \langle \frac{\partial \mathcal{L}}{\partial y}, y \rangle$. Nonetheless, standard initialization schemes set biases to zero, thus the simpler version of our conservation law suffices to analyze the ResNet/VGGNet architectures we consider empirically at initialization, resolving **R3**'s concern.

Pruning in train mode and an implementation discrepancy in GraSP. We thank **R3** for noticing that the original implementations of SNIP and GraSP prune in train mode. To match the implementation exactly, we updated our code base and re-ran the results for both algorithms in figures 1 and 6. However the empirical conclusions with respect to SynFlow have not changed, as noted in the updated version of figure 1 on the right (Grasp - Fig. A, SNIP - Fig. B) We further communicated with the authors of GraSP to eliminate any implementation discrepancies in our GraSP submission code (orange line) and now our updated implementation (blue line) matches within error of the official baseline (dashed black line). As **R3** expected, GraSP now better aligns with SNIP (especially with Tiny-ImageNet where we had first reported poor GraSP performance). However, the assumption that GraSP should always align with SNIP is incorrect. A recent preprint [de Jorge, et al. 2020] independently reports that GraSP can perform much worse than SNIP at high compression ratios (SNIP: 51.3%, GraSP: 0.1% at 98% sparsity on VGG19/ImageNet) and the GraSP paper compared to SNIP at only 3 compression ratios. We sweep over 26 compression ratios representing one of the most thorough benchmarks of pruning algorithms at initialization.





Iterative magnitude, SNIP, GraSP pruning. Our theorem 3 states that iteration is a necessary ingredient for any pruning algorithm, elucidating the success of iterative magnitude pruning, as **R2** noted, and concurrent work on iterative versions of SNIP [Verdenius, et al. 2020], [de Jorge, et al. 2020]. We are currently comparing SynFlow to these much more computationally expensive methods. From initial experiments, iterative SNIP avoids layer collapse, but sacrifices its performance in low compression regimes underperforming SynFlow (Fig. B).

Computational cost of SynFlow. The computational cost of a pruning algorithm can be measured by the number of forward/backward passes (#iterations $\times$ #examples per iteration). We always run the data-agnostic SynFlow with 100 iterations, implying it takes 100 passes no matter the dataset. SNIP and GraSP each involve 1 iteration, but use 10 times the number of classes per iteration requiring 1000, 2000, and 10,000 passes for CIFAR-100, Tiny-ImageNet, and ImageNet respectively. Iterative versions of them will be multiplicatively more expensive. For example, iterative SNIP with 100 iterations on CIFAR-100 would require 100,000 passes, whereas SynFlow would only require 100!

SynFlow's data-agnostic property brings the fields of network pruning and initialization together. As noted by **R4**, SynFlow demonstrates the most impressive empirical improvement to other methods in the high compression regimes. However, even in the low compression regimes SynFlow does on par with other methods without even looking at the data, which we believe to be a major accomplishment. This striking capability which we have demonstrated theoretically and empirically in our work opens up a new direction of sparse initialization via network pruning.