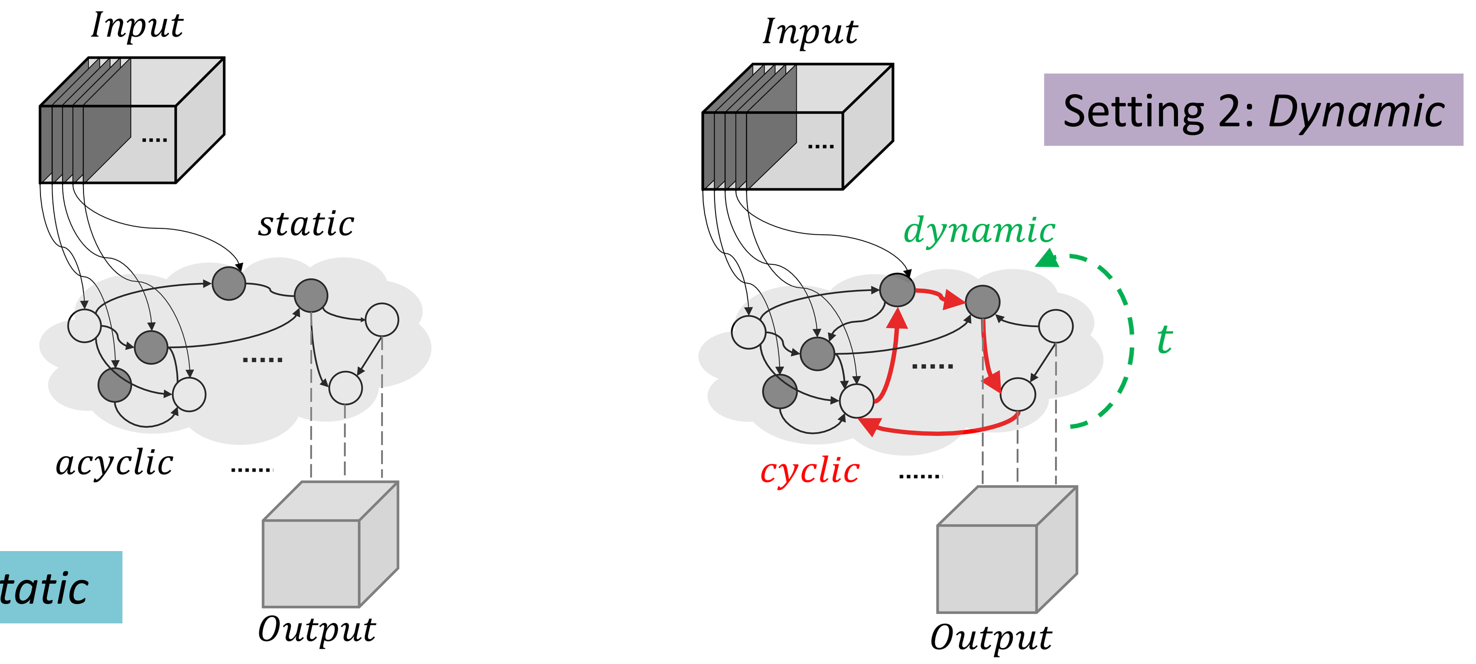
Discovering Neural Wirings

Mitchell Wortsman[1,2], Ali Farhadi[1,2,3], Mohammad Rastegari[1,3]

[1] PRIOR @ Allen Institute for AI   [2] University of Washington   [3] XNOR.AI

MOTIVATION

- Traditionally, the connectivity patterns of neural networks are manually defined or largely constrained (even with methods of Neural Architecture Search (NAS) [1]).
- We allow for a much **larger space of possible networks** by relaxing the typical notion of layers and enabling channels to form connections independent of each other.
- The **wiring of our network is not fixed** during training – as we learn the network parameters we also learn the connectivity.



CONTRIBUTIONS

- We present an algorithm, **Discovering Neural Wirings (DNW),** to efficiently learn the connectivity of a neural network.
- By learning the connectivity of MoileNetV1 (x0.25) [2] we **boost the accuracy on ImageNet by 10%.**
- We demonstrate that DNW can also be used to **effectively train sparse neural networks in a single training run**: We only ever train with 10% of weights (in the forward pass) and only lose 2.5% accuracy on ImageNet compared to our dense baseline.

KEY TAKEAWAYS

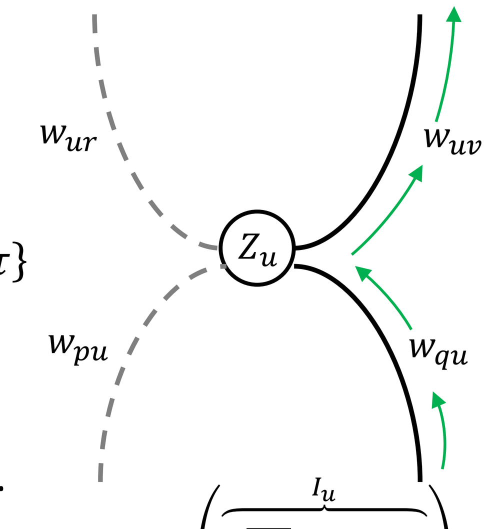
- It is possible to realize the benefits of overparameterization during training, even when the resulting model is sparse.
- As NAS becomes more fine grained, finding a good architecture is akin to finding a sparse subnetwork of the complete graph.

Algorithm

Forward

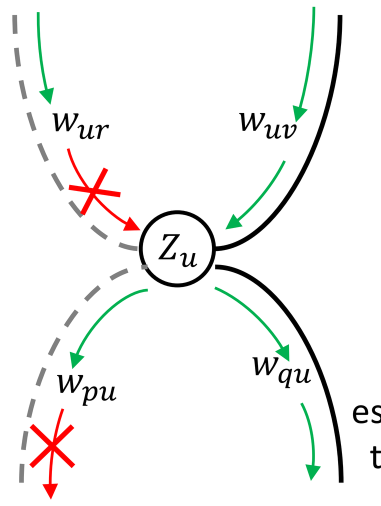
$\mathcal{E} = \{w_{uv} : |w_{uv}| \geq \tau\}$

Take the k edges with the highest magnitude weights.



$$Z_u = f_{\theta_u}\left(\sum_{(q,u)\in\mathcal{E}}^{I_u} w_{qu} Z_q\right)$$

Backward



$$\frac{\partial \mathcal{L}}{\partial Z_u} = \sum_{(u,v)\in\mathcal{E}} \frac{\partial \mathcal{L}}{\partial I_v} w_{uv}$$

Update ***all*** edges in the graph via

$$w_{uv} \leftarrow w_{uv} + \left\langle Z_u, -\alpha \frac{\partial \mathcal{L}}{\partial I_v} \right\rangle$$

(*i.e.* using a straight-through estimator [3]) and update the rest of the network parameters normally.

Setting 1: *Static*

- $G = (V, \mathcal{E})$ is directed acyclic graph where w_{uv} is the weight of edge (u, v).
- Let Z_u denote the state of node u, and let I_u denote the input to node u.
- The state of a fixed set of input nodes are equal to some function g of the input data. For all other nodes, the state Z_u is computed as

$$Z_u = f_{\theta_u}\left(\sum_{(q,u)\in\mathcal{E}}^{I_u} w_{qu} Z_q\right)$$

Setting 2: *Dynamic*

- $G = (V, \mathcal{E})$ is directed possibly cyclic graph where w_{uv} is the weight of edge (u, v).
- Let $Z_u(t)$ denote the state of node u at time t, and let $I_u(t)$ denote the input to node u at time t.
- The initialization $Z_u(0)$ of a fixed set of input nodes are equal to some function g of the input data. For all other nodes, $Z_u(0) = 0$. The state $Z_u(t)$ is computed as

$$Z_u(t) = f_{\theta_u}\left(\sum_{(q,u)\in\mathcal{E}}^{I_u(t-1)} w_{qu} Z_q(t-1)\right) \quad \text{when } t \in \{0, 1, \dots, T-1\}$$

$$\frac{dZ_u(t)}{dt} = f_{\theta_u}\left(\sum_{(q,u)\in\mathcal{E}}^{I_u(t)} w_{qu} Z_q(t)\right) \quad \text{when } t \in [0, T] \text{ as in [4]}$$

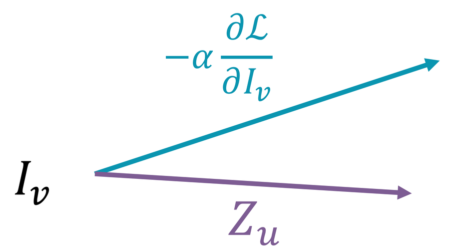
- We consider the case where the input and output of each node is a two-dimensional matrix, commonly referred to as a channel.
- The state of a fixed set of output nodes are taken to be the output of the network.
- We let f_{θ_u} at each non-output node be a batch-norm, (2 parameters), ReLU, 3x3 convolution (9 parameters) triplet.
- The goal is to learn a good graph with exactly k edges.

Intuition

- If the gradient is pushing I_v in a direction which aligns with Z_u, then we strengthen the magnitude of the weight w_{uv}.
- If this alignment happens consistently then $|w_{uv}|$ will be eventually be strong enough to enter the edge set $\mathcal{E}$.
- If (u, v) enters $\mathcal{E}$, another edge will be removed. We show that when this swapping does occur, it is beneficial (i.e. it decreases the loss on the mini-batch under certain conditions).
- If the edge is already in the graph, the update rule is no different than standard SGD.

Update Rule

$$w_{uv} \leftarrow w_{uv} + \left\langle Z_u, -\alpha \frac{\partial \mathcal{L}}{\partial I_v} \right\rangle$$



Results

By learning the connectivity of MobileNetV1 [3], we boost the ImageNet accuracy by ~10% at ~41M FLOPS.

Model	Params	FLOPs	Accuracy
MobileNetV1 (×0.25)	0.5M	41M	50.6%
X-4 MobileNetV1	—	> 50M	54.0%
MobileNetV2 (×0.15)*	—	39M	44.9%
MobileNetV2 (×0.4)**	—	43M	56.6%
DenseNet (×0.5)*	—	42M	41.1%
Xception (×0.5)*	—	40M	55.1%
ShuffleNetV1 (×0.5, $g = 3$)	—	38M	56.8%
ShuffleNetV2 (×0.5)	1.4M	41M	60.3%
MobileNetV1-Random Graph(×0.225)	1.2M	55.7M	53.3%
MobileNetV1-DNW-Small (×0.15)	0.24M	22.1M	50.3%
MobileNetV1-DNW-Small (×0.225)	0.4M	41.2M	59.9%
MobileNetV1-DNW(×0.225)	1.1M	42.1M	60.9%
MnasNet-search1	1.9M	65M	64.9%
MobileNetV1-DNW(×0.3)	1.3M	66.7M	65.0%
MobileNetV1 (×0.5)	1.3M	149M	63.7%
MobileNetV2 (×0.6)*	—	141M	66.6%
MobileNetV2 (×0.75)***	—	145M	67.9%
DenseNet (×1)*	—	142M	54.8%
Xception (×1)*	—	145M	65.9%
ShuffleNetV1 (×1, $g = 3$)	—	140M	67.4%
ShuffleNetV2 (×1)	2.3M	146M	69.4%
MobileNetV1-Random Graph(×0.49)	1.8M	170M	64.1%
MobileNetV1-DNW(×0.49)	1.8M	154M	70.4%

Comparing the Static vs. Dynamic Setting with a tiny (41k parameter) classifier on CIFAR-10.

Model	Accuracy
Static (Random Graph)	$76.1 \pm 0.5\%$
Static (DNW)	$80.9 \pm 0.6\%$
Discrete Time (Random Graph)	$77.3 \pm 0.7\%$
Discrete Time (DNW)	$82.3 \pm 0.6\%$
Continuous (Random Graph)	$78.5 \pm 1.2\%$
Continuous (DNW)	$83.1 \pm 0.3\%$

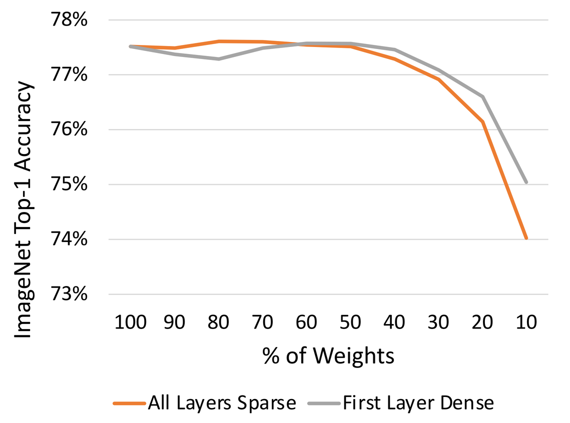
Contrasting with other methods of Discovering Wirings with a fixed structure and number of edges on CIFAR-10.

Model	Accuracy
MobileNetV1 (×0.25)	$86.3 \pm 0.2\%$
MobileNetV1-Random Graph(×0.225)	$87.2 \pm 0.1\%$
No Update Rule	$86.7 \pm 0.5\%$
L1 + Anneal	$84.3 \pm 0.6\%$
Targeted Dropout $\rho = 0.95$	$89.2 \pm 0.4\%$
Lottery Ticket (one-shot)	$87.9 \pm 0.3\%$
MobileNetV1-DNW(×0.225)	$89.7 \pm 0.2\%$

Sparse Neural Network Learning

Method	Weights (%)	Top-1 Accuracy	Top-5 Accuracy
Sparse Networks from Scratch	10%	72.9%	91.5%
Ours - All Layers Sparse	10%	74.0%	92.0%
Ours - First Layer Dense	10%	75.0%	92.5%
Sparse Networks from Scratch	20%	74.9%	92.5%
Ours - All Layers Sparse	20%	76.2%	93.0%
Ours - First Layer Dense	20%	76.6%	93.4%
Sparse Networks from Scratch	30%	75.9%	92.9%
Ours - All Layers Sparse	30%	76.9%	93.4%
Ours - First Layer Dense	30%	77.1%	93.5%
Sparse Networks from Scratch	100%	77.0%	93.5%
Ours - Dense Baseline	100%	77.5%	93.7%

We apply our algorithm to the task of training a sparse ResNet-50. The sparsity is maintained throughout training, as motivated by Sparse Networks From Scratch [5].



[1] Barret Zoph and Quoc V. Le. **Neural Architecture Search with Reinforcement Learning.** In *ICLR*, 2016.
[2] Andrew G. Howard et al. **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.** In *ArXiv*, 2017.
[3] Yoshua Bengio et al. **Estimating or propagating gradients through stochastic neurons for conditional computation.** ArXiv, 2013.
[4] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt and David Duvenaud. **Neural Ordinary Differential Equations.** In *NeurIPS*, 2018.
[5] Tim Dettmers and Luke Zettlemoyer. **Sparse Networks from Scratch: Faster Training without Losing Performance.** In *ArXiv*, 2019.