Supplementary Material of
Finding trainable sparse networks through Neural Tangent Transfer

Tianlin Liu and Friedemann Zenke

Appendix A Proof of Proposition 1

Here we prove Proposition 1 in the main text.

Proposition 1. *Consider a linear NTT teacher model $f(\boldsymbol{x}, \boldsymbol{a}(0)) = \boldsymbol{a}(0)^\top \boldsymbol{x}$, where $\boldsymbol{x}$ and $\boldsymbol{a}(0) \in \mathbb{R}^d$ are model input and initial parameters. Suppose that we are given a linear NTT student model $g_{\boldsymbol{m}}(\boldsymbol{x}, \widetilde{\boldsymbol{a}}(0)) = (\boldsymbol{m} \odot \widetilde{\boldsymbol{a}}(0))^\top \boldsymbol{x}$ whose initial parameters $\widetilde{\boldsymbol{a}}(0)$ are NTT-optimal in the sense that $J_{\boldsymbol{a}(0)}(\boldsymbol{m} \odot \widetilde{\boldsymbol{a}}(0)) = 0$. Then upon continuous-time and quadratic-loss based gradient descent training, the dense and sparse models' outputs evolve in the same way:*

$$f(\boldsymbol{x}, \boldsymbol{a}(t)) = g_{\boldsymbol{m}}(\boldsymbol{x}, \widetilde{\boldsymbol{a}}(t)),$$

for all training inputs $\boldsymbol{x}$ and time steps $t \geq 0$.

Proof of Proposition 1. Let $\{\boldsymbol{x}_i, y_i\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ be a training dataset of n input-target pairs. We first consider the dense linear model $f(\boldsymbol{x}, \boldsymbol{a}) = \boldsymbol{a}^\top \boldsymbol{x}$. We use the shorthand notation $\boldsymbol{X} = [\boldsymbol{x}_i]_{i \in [n]} \in \mathbb{R}^{d \times n}$ as the column-wise concatenation of training inputs, $f(\boldsymbol{X}, \boldsymbol{a}) = \boldsymbol{X}^\top \boldsymbol{a} \in \mathbb{R}^n$ as the vector whose entries are outputs of the dense linear model, and $\boldsymbol{y} = (y_i)_{i \in [n]} \in \mathbb{R}^n$ as the corresponding targets. As a special case of [Arora et al., 2019, Lemma 3.1], namely when the model is assumed to be linear, the model output follows the evolution

$$\frac{\mathrm{d}f(\boldsymbol{X}, \boldsymbol{a}(t))}{\mathrm{d}t} = -\boldsymbol{H}\Big[f\Big(\boldsymbol{X}, \boldsymbol{a}(t)\Big) - \boldsymbol{y}\Big], \tag{1}$$

where $\boldsymbol{H} = \boldsymbol{X}^\top \boldsymbol{X} \in \mathbb{R}^{n \times n}$. The solution of the linear ODE (1) is given by

$$f\Big(\boldsymbol{X}, \boldsymbol{a}(t)\Big) = e^{-t\boldsymbol{H}} f\Big(\boldsymbol{X}, \boldsymbol{a}(0)\Big) + \left[\boldsymbol{I} - e^{-t\boldsymbol{H}}\right] \boldsymbol{y}. \tag{2}$$

We now turn to the case of the sparse linear model $g_{\boldsymbol{m}}(\widetilde{\boldsymbol{a}}, \boldsymbol{x}) = (\boldsymbol{m} \odot \widetilde{\boldsymbol{a}})^\top \boldsymbol{x}$. For convenience, we write $g_{\boldsymbol{m}}\Big(\boldsymbol{X}, \boldsymbol{a}(t)\Big) = \boldsymbol{X}^\top \mathrm{diag}(\boldsymbol{m}) \boldsymbol{a} \in \mathbb{R}^n$ as a vector whose entries are outputs of the sparse linear model, where diag$(\cdot)$ transforms the d-dimensional vector $\boldsymbol{m}$ into a d-by-d diagonal matrix. Similar to the case of the dense model, the sparse model's output dynamics are characterized by the linear ODE

$$\frac{\mathrm{d}g_{\boldsymbol{m}}(\boldsymbol{X}, \boldsymbol{a}(t))}{\mathrm{d}t} = -\widetilde{\boldsymbol{H}}\Big[g_{\boldsymbol{m}}\Big(\boldsymbol{X}, \boldsymbol{a}(t)\Big) - \boldsymbol{y}\Big], \tag{3}$$

where $\widetilde{\boldsymbol{H}} = (\mathrm{diag}(\boldsymbol{m})\boldsymbol{X})^\top \mathrm{diag}(\boldsymbol{m})\boldsymbol{X} \in \mathbb{R}^{n\times n}$. Solving the linear ODE in Eqn. (3), we get

$$g_{\boldsymbol{m}}\Big(\boldsymbol{X}, \boldsymbol{a}(t)\Big) = e^{-t\widetilde{\boldsymbol{H}}} g_{\boldsymbol{m}}\Big(\boldsymbol{X}, \widetilde{\boldsymbol{a}}(0)\Big) + [\boldsymbol{I} - e^{-t\widetilde{\boldsymbol{H}}}]\boldsymbol{y}. \tag{4}$$

Note that the NTT objective (Eqn. (14)) achieves 0 only if $\boldsymbol{H} = \widetilde{\boldsymbol{H}}$ and $f\Big(\boldsymbol{X}, \boldsymbol{a}(0)\Big) = g_{\boldsymbol{m}}\Big(\boldsymbol{X}, \widetilde{\boldsymbol{a}}(0)\Big)$. Comparing Eqn. (2) and Eqn. (4), we see

$$f(\boldsymbol{x}, \boldsymbol{a}(t)) = g_{\boldsymbol{m}}(\boldsymbol{x}, \widetilde{\boldsymbol{a}}(t)),$$

for all $\boldsymbol{x} \in \{\boldsymbol{x}_i\}_{i=1}^n$ and timesteps $t \geq 0$ as claimed. □

Appendix B Details of experiment setup

We proceed to introduce the details of experiments reported in the main text, including the model architecture, optimization hyper-parameters, and baseline methods. All experiments were performed using JAX [Bradbury et al., 2018] and the neural-tangents library [Novak et al., 2020].

B.1 Neural network architecture

For most of the MNIST and Fashion MNIST classification tasks, we use the standard Lenet-300-100 MLP and Lenet-5-caffe CNN architecture together with Relu activations for hidden layers and softmax cross-entropy loss on logit outputs. One exception is the toy example reported in Sec. 4.1, where we used 2 linear output neurons to perform regression.

For the CIFAR-10 and SVHN datasets, we used a CNN model consisting of 4 convolution layers followed by 2 feedforward layers with dropout (Table 1). This can be considered as a slightly scaled-up version of the CNN from the Keras tutorial[1], in which the only modification we made was to double the number of filters in each convolutional layer.

Operation	Filter size	# Filters	Stride	Dropout rate	Activation
3x32x32 input	–	–	–	–	–
Conv	3 × 3	64	1 × 1	–	ReLu
Conv	3 × 3	64	1 × 1	–	ReLu
MaxPool	–	–	2 × 2	0.25	–
Conv	3 × 3	128	1 × 1	–	ReLu
Conv	3 × 3	128	1 × 1	–	ReLu
MaxPool	–	–	2 × 2	0.25	–
FC	–	512	–	0.5	ReLu
FC	–	10	–	–	Softmax

Table 1: The Conv-4 architecture used for the CIFAR-10 and SVHN tasks. The dropout rate is defined to be the fraction of the input units to drop.

[1] `https://keras.io/examples/cifar10_cnn/`

B.2 NTT hyperparameters

In Table 2 we summarize the hyperparameters used in the NTT optimization stage. For each experiment, we initialized NTT teachers using the Glorot initialization scheme [Glorot & Bengio, 2010]; we then perform perform gradient-based optimization using the Adam optimizer [Kingma & Ba, 2015] with various learning rates and batch sizes (see Table 2).

Task	Model	Epoch number	Batch size	Learning rate η	Mask update frequency	Strength parameter γ^2	weight-decay constant β
Visualization	Lenet-300-100	5000 (full-batch)	500	1e-03	100 iters	1e-5	0
	Lenet-5-caffe	5000 (full-batch)	500	5e-04	100 iters	1e-6	0
MNIST	Lenet-300-100	20	64	5e-04	100 iters	1e-3	1e-4
	Lenet-5-caffe	20	64	5e-04	100 iters	1e-3	1e-5
Fashion-MNIST	Lenet-300-100	20	64	5e-04	100 iters	1e-3	1e-4
	Lenet-5-caffe	20	64	5e-04	100 iters	1e-3	1e-5
CIFAR-10	Conv-4 CNN (see Table 1)	20	32	5e-04	100 iters	1e-3	1e-8
SVHN	Conv-4 CNN (see Table 1)	20	32	5e-04	100 iters	1e-3	1e-8

Table 2: Hyperparameters used for NTT in this paper.

The toy example in Section 4.1 of the main text deserves some additional comments. For this task, we used 500 images from the MNIST dataset, containing 250 images of each digit 0 and 1. We performed 5000 iterations of full-batch gradient descent for this task; for this reason, 5000 is also the total number of epochs.

B.3 Supervised learning hyperparameters

Regarding the supervised learning experiments, we spared 10% of the training data for model validation purposes and only used 90% for model training. We used the Adam optimizer with learning rate 1e-3, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for all supervised learning tasks except for the visualization task in Sec. 4.1, in which the stochastic gradient descent optimizer with learning rate 0.01 was used. In addition, all experiments, except for the visualization task, used a minibatch-size of 64. For MNIST and Fashion MNIST experiments, we performed optimization for 50 epochs. On CIFAR-10, we trained for 600 epochs.

B.4 Baseline pruning methods

In this subsection, we first recap the SNIP pruning method [Lee et al., 2019] and introduce two straightforward extensions of it, Layerwise-SNIP and Logit-SNIP, which were used as baselines for NTT. Finally, we point out some technicalities of random pruning baselines.

SNIP and Layerwise-SNIP Recall that SNIP [Lee et al., 2019] assigns each neural network parameter θ a sensitivity score $S(\theta)$ defined as

$$S(\theta) = \left| \theta \cdot \frac{\partial L_{\boldsymbol{\theta}}}{\partial \theta} \right| ,$$

where $L_{\boldsymbol{\theta}} = \sum_{i=1}^{n_B} L(f(\boldsymbol{x}_i, \boldsymbol{\theta}), \boldsymbol{y}_i)$ is the loss evaluated over a batch of n_B number of input-output data pairs $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{n_B}$ and $\boldsymbol{\theta}$ is the vector of randomly initialized parameters. Lee et al. [2019]

proposed to remove neural network parameters with lowest sensitivity scores. That is, in its original formulation, SNIP is a global pruning method. To be consistent with [Lee et al., 2019], we reserve the terminology SNIP to only be used in global pruning context. A straightforward way to turn SNIP into a layerwise pruning method is to remove a fixed fraction of the parameters having the lowest sensitivity scores from each layer. We call this extension Layerwise-SNIP.

Logit-SNIP and layerwise Logit-SNIP The SNIP and Layerwise-SNIP methods described above depend on labels. Here we provide a label-free extension: We modify the sensitivity score $S(\theta)$ into a logit-based sensitivity score $\tilde{S}(\theta)$ defined as

$$\tilde{S}(\theta) = \left| \theta \cdot \frac{\partial Z_{\boldsymbol{\theta}}}{\partial \theta} \right| ,$$

where $Z_{\boldsymbol{\theta}} = \sum_{i=1}^{n_B} \|f(\boldsymbol{x}_i, \boldsymbol{\theta})\|_2^2$. We can perform either global or layerwise pruning in reference to the scores $\tilde{S}(\theta)$. We refer the global pruning criteria as Logit-SNIP and layerwise criteria as layerwise Logit-SNIP. When the context is clear, we may use the terminology Logit-SNIP to refer to either its layerwise or global variant.

Random pruning In the main text we have explained two ways to randomly sample sparse neural networks. Note that for these random methods, their global pruning variant is equivalent to their respective layerwise variant: In either formulation, each weight parameter receives an identical chance to be removed and therefore the expected fraction of pruned parameters for each layer is the same. In each run, we randomly remove such expected fraction of parameters from each layer.

Appendix C Additional experiments and results

C.1 Visualizing network output evolution in CNNs

We repeated the experiment introduced in Sec. 4.1 of the main text using a Lenet-5-like architecture with two linear output neurons (Fig. 1). In good agreement with the MLP results, we found that the NTT teacher and student follow a similar output evolution during supervised training.

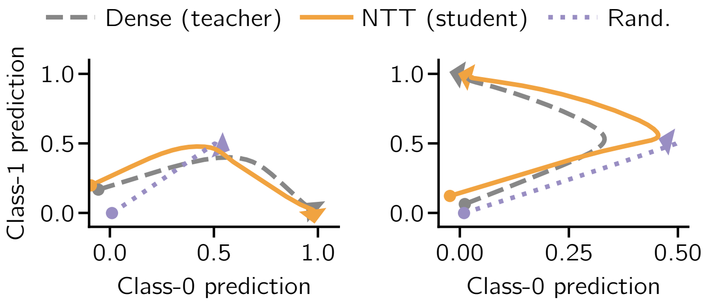
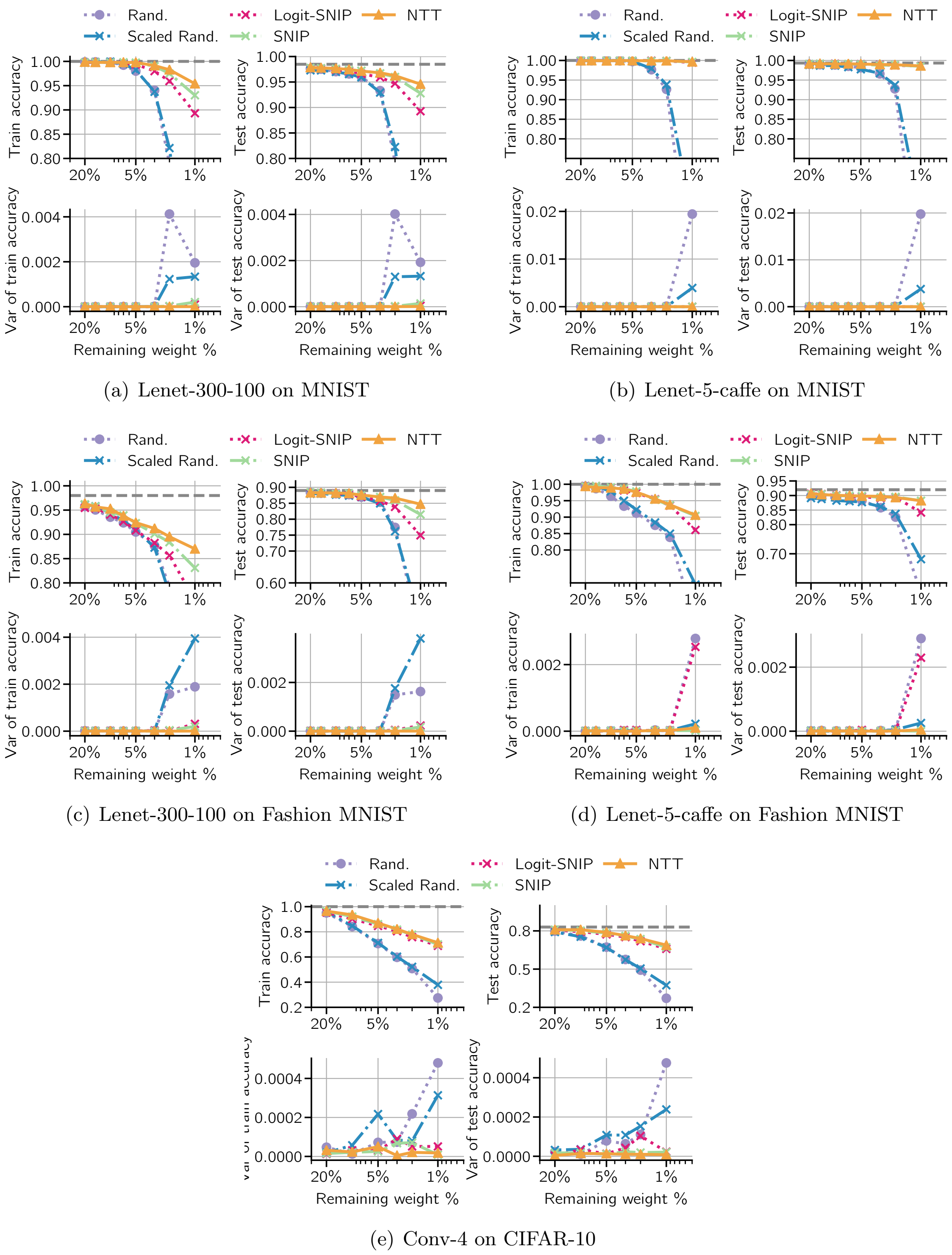


Figure 1: **A CNN NTT student's output evolution closely follows its dense teacher's.**

C.2 Experiments on global pruning

In the main text, we have focused on layerwise pruning. Here we provide experimental results on global pruning methods using various datasets. The overall experiment procedure and hyperparameters used in this set global pruning experiments are identical to the settings in the layerwise experiments, except at one place: During NTT, instead of initializing the binary mask based on weight magnitudes as outlined in Sec 3.3, we found that the NTT optimization process converges slightly faster if we use the Logit-SNIP produced mask as the initialization. Since Logit-SNIP masks do not depend on labels, the NTT procedure remains label-independent.



(a) Lenet-300-100 on MNIST

(b) Lenet-5-caffe on MNIST

(c) Lenet-300-100 on Fashion MNIST

(d) Lenet-5-caffe on Fashion MNIST

(e) Conv-4 on CIFAR-10

Figure 2: Supervised performance of NTT and baseline methods under global pruning.

References

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. On exact computation with an infinitely wide neural net. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8139–8148. Curran Associates, Inc., 2019.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., and Wanderman-Milne, S. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Lee, N., Ajanthan, T., and Torr, P. H. S. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J., and Schoenholz, S. S. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020. URL `https://github.com/google/neural-tangents`.