Things to Know in Deep Learning these Days

The Lottery Ticket Hypothesis (LTH)

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks

Jonathan Frankle, Michael Carbin

Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. However, contemporary experience is that the sparse architectures produced by pruning are difficult to train from the start, which would similarly improve training performance.
We find that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on these results, we articulate the "lottery ticket hypothesis:" dense, randomly-initialized, feed-forward networks contain subnetworks ("winning tickets") that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.
We present an algorithm to identify winning tickets and a series of experiments that support the lottery ticket hypothesis and the importance of these fortuitous initializations. We consistently find winning tickets that are less than 10-20% of the size of several fully-connected and convolutional feed-forward architectures for MNIST and CIFAR10. Above this size, the winning tickets that we find learn faster than the original network and reach higher test accuracy.

ICLR 2019

https://arxiv.org/abs/1803.03635

Knowledge Evolution in NNs

Knowledge Evolution in Neural Networks

Ahmed Taha, Abhinav Shrivastava, Larry Davis

Deep learning relies on the availability of a large corpus of data (labeled or unlabeled). Thus, one challenging unsettled question is: how to train a deep network on a relatively small dataset? To tackle this question, we propose an evolution-inspired training approach to boost performance on relatively small datasets. The knowledge evolution (KE) approach splits a deep network into two hypotheses: the fit-hypothesis and the reset-hypothesis. We iteratively evolve the knowledge inside the fit-hypothesis by perturbing the reset-hypothesis for multiple generations. This approach not only boosts performance, but also learns a slim network with a smaller inference cost. KE integrates seamlessly with both vanilla and residual convolutional networks. KE reduces both overfitting and the burden for data collection.

We evaluate KE on various network architectures and loss functions. We evaluate KE using relatively small datasets (e.g., CUB-200) and randomly initialized deep networks. KE achieves an absolute 21% improvement margin on a state-of-the-art baseline. This performance improvement is accompanied by a relative 73% reduction in inference cost. KE achieves state-of-the-art results on classification and metric learning benchmarks. Code available at this http URL

CVPR 2021

https://arxiv.org/abs/2103.05152

Knowledge Evolution in NNs

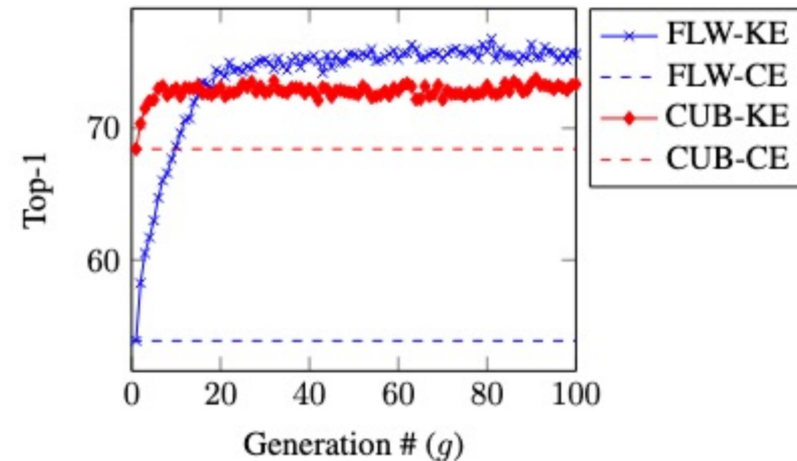
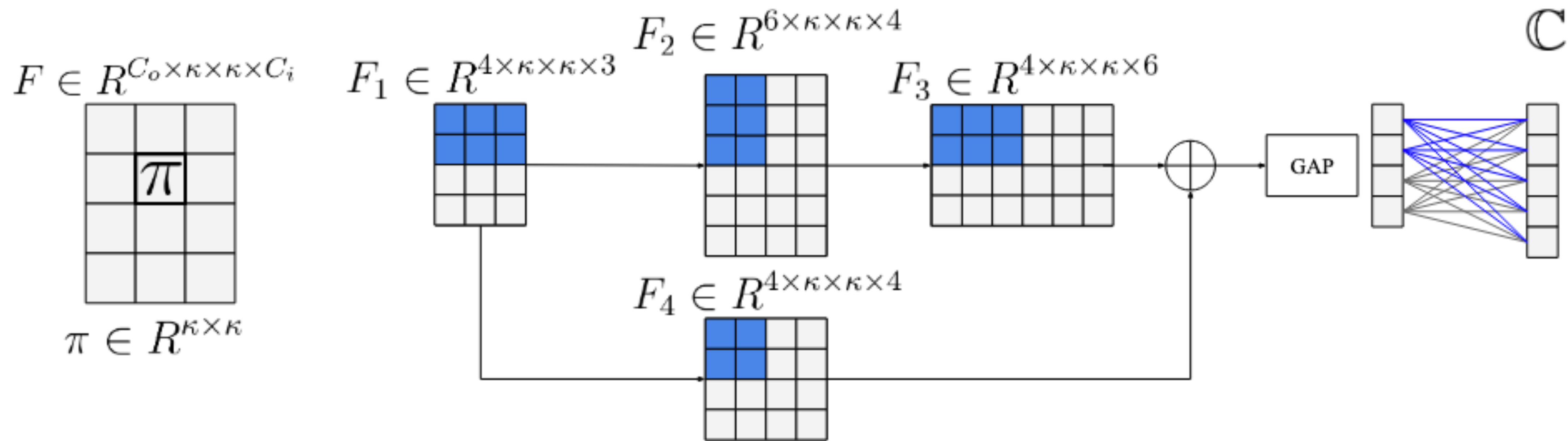


Figure 1. Classification performance on Flower-102 (FLW) and CUB-200 (CUB) datasets trained on a randomly initialized ResNet18. The horizontal dashed-lines denote a SOTA cross-entropy (CE) baseline [63]. The marked-curves show our approach (KE) performance across generations. The 100th generation KE-N_{100} achieves absolute 21% and 5% improvement margins over the Flower-102 and CUB-200 baselines, respectively.

CVPR 2021

https://arxiv.org/abs/2103.05152

Knowledge Evolution in NNs



The Knowledge evolution (KE) approach starts by *conceptually* splitting the deep network N into two exclusive hypotheses (subnetworks): the fit-hypothesis $H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$ as shown in Fig. 2. These hypothe-

CVPR 2021

https://arxiv.org/abs/2103.05152

Knowledge Evolution in NNs

$M_l \in \{0,1\}^{C_o \times \kappa \times \kappa \times C_i}$ $F_l \in R^{C_o \times \kappa \times \kappa \times C_i}$

1	1	0	0
1	1	0	0
0	0	0	0
0	0	0	0

π

$\pi \in R^{\kappa \times \kappa}$

Figure 3. The KELS technique for CNNs. Given a split-rate s_r and a convolutional filter F_l at a layer l, the binary split-mask M_l outlines the first $\lceil s_r \times C_i \rceil$ kernels inside the first $\lceil s_r \times C_o \rceil$ filters. In this example, $C_o = C_i = 4$ and $s_r = 0.5$. Through KELS, the binary mask M outlines the fit-hypothesis $H^{\triangle}$ such that it is a slim network inside a dense network. The slim network $H^{\triangle}$ is equivalent to a dense network with $(1 - s_r^2)$ sparsity.

CVPR 2021

https://arxiv.org/abs/2103.05152

Knowledge Evolution in NNs

The Knowledge evolution (KE) approach starts by *conceptually* splitting the deep network N into two exclusive hypotheses (subnetworks): the fit-hypothesis $H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$ as shown in Fig. 2. These hypothe-

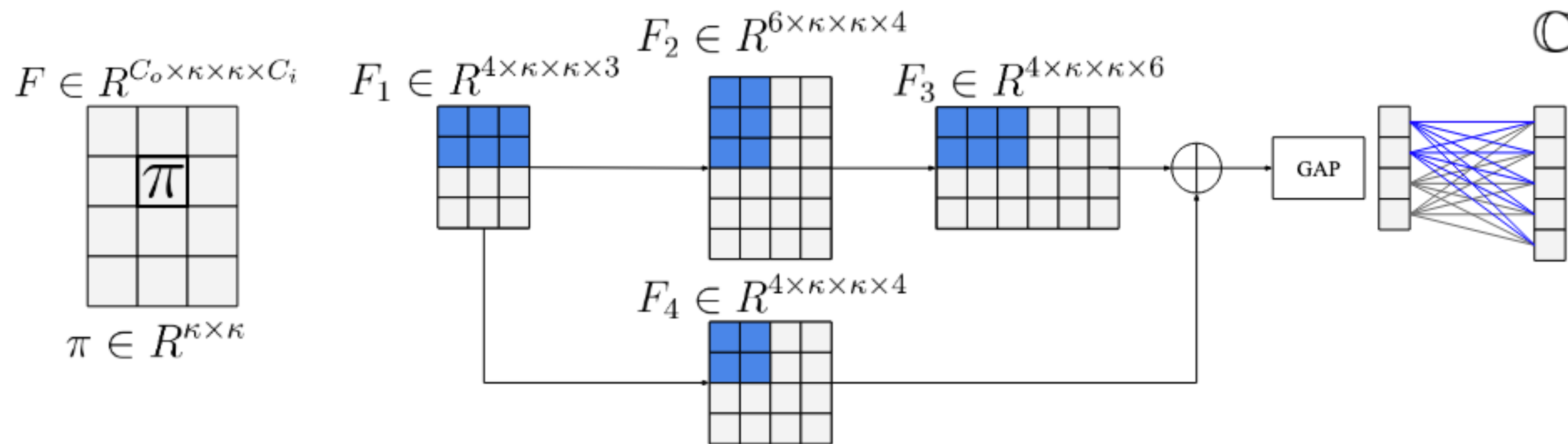
CVPR 2021

Table 6. The dimensions of the ResNet18 N versus its fit-hypothesis $H^{\triangle}$ with split-rate $s_r = 0.5$. The last table-section compares N and $H^{\triangle}$ through the number of operations and parameters (millions). The fit-hypothesis $H^{\triangle}$ is a slim independent network with 102 logits (Flower-102).

Layers	ResNet18 N	Fit-hypothesis $H^{\triangle}$
conv1	$64 \times 7 \times 7 \times 3$	$32 \times 7 \times 7 \times 3$
bn1	64	32
layer1.0.conv1	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.0.bn1	64	32
layer1.0.conv2	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.0.bn2	64	32
layer1.1.conv1	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.1.bn1	64	32
layer1.1.conv2	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.1.bn2	64	32
layer2.0.conv1	$128 \times 3 \times 3 \times 64$	$64 \times 3 \times 3 \times 32$
layer2.0.bn1	128	64
layer2.0.conv2	$128 \times 3 \times 3 \times 128$	$64 \times 3 \times 3 \times 64$
layer2.0.bn2	128	64
layer2.0.downsample.0	$128 \times 1 \times 1 \times 64$	$64 \times 1 \times 1 \times 32$
layer2.0.downsample.1	128	64
layer2.1.conv1	$128 \times 3 \times 3 \times 128$	$64 \times 3 \times 3 \times 64$
layer2.1.bn1	128	64
layer2.1.conv2	$128 \times 3 \times 3 \times 128$	$64 \times 3 \times 3 \times 64$
layer2.1.bn2	128	64
layer3.0.conv1	$256 \times 3 \times 3 \times 128$	$128 \times 3 \times 3 \times 64$
layer3.0.bn1	256	128
layer3.0.conv2	$256 \times 3 \times 3 \times 256$	$128 \times 3 \times 3 \times 128$
layer3.0.bn2	256	128
layer3.0.downsample.0	$256 \times 1 \times 1 \times 128$	$128 \times 1 \times 1 \times 64$
layer3.0.downsample.1	256	128
layer3.1.conv1	$256 \times 3 \times 3 \times 256$	$128 \times 3 \times 3 \times 128$
layer3.1.bn1	256	128
layer3.1.conv2	$256 \times 3 \times 3 \times 256$	$128 \times 3 \times 3 \times 128$
layer3.1.bn2	256	128
layer4.0.conv1	$512 \times 3 \times 3 \times 256$	$256 \times 3 \times 3 \times 128$
layer4.0.bn1	512	256
layer4.0.conv2	$512 \times 3 \times 3 \times 512$	$256 \times 3 \times 3 \times 256$
layer4.0.bn2	512	256
layer4.0.downsample.0	$512 \times 1 \times 1 \times 256$	$256 \times 1 \times 1 \times 128$
layer4.0.downsample.1	512	256
layer4.1.conv1	$512 \times 3 \times 3 \times 512$	$256 \times 3 \times 3 \times 256$
layer4.1.bn1	512	256
layer4.1.conv2	$512 \times 3 \times 3 \times 512$	$256 \times 3 \times 3 \times 256$
layer4.1.bn2	512	256
fc	102×512	102×256
#Ops (G-Ops)	3.63	0.96
#Parameters	22.44	5.64

Knowledge Evolution in NNs

various splitting techniques in Sec. 3.2. After outlining the hypotheses, the network N is initialized randomly, *i.e.*, both $H^{\triangle}$ and $H^{\triangledown}$ are initialized randomly. We train N for e epochs and refer to the trained network as the first generation N_1, where $H_1^{\triangle} = MN_1$ and $H_1^{\triangledown} = (1 - M)N_1$.



CVPR 2021

https://arxiv.org/abs/2103.05152

Knowledge Evolution in NNs

To learn a better network (the next generation), we (1) **re-initialize** the network N using $H_1^{\triangle}$, then (2) **re-train** N to learn N_2. First, the network N is **re-initialized** using the convolutional filters F and weights W in the fit-hypothesis $H_1^{\triangle}$ from N_1, while the rest of the network ($H^{\triangledown}$) is initialized randomly. Formally, we re-initialize each layer l, using Hadamard product, as follows

$$F_l = M_l F_l + (1 - M_l)\, F_l^r, \tag{1}$$

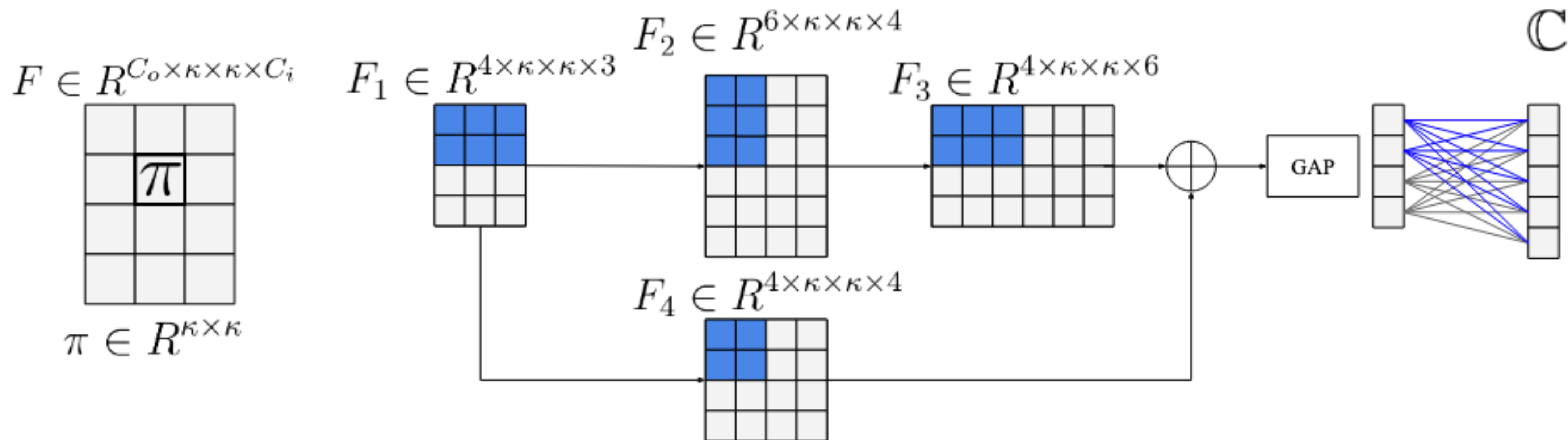
where F_l is a convolutional filter at layer l, M_l is the corresponding binary mask and F_l^r is a randomly initialized tensor. These three tensors (F_l, F_l^r, and M_l) have the same size ($\in R^{C_o \times \kappa \times \kappa \times C_i}$). F_l^r is initialized using the default initialization distribution. For example, PyTorch uses Kaiming uniform [19] for convolution layers.

CVPR 2021

https://arxiv.org/abs/2103.05152

Knowledge Evolution in NNs

To learn a better network (the next generation), we (1) **re-initialize** the network N using $H_1^{\triangle}$, then (2) **re-train** N to learn N_2. First, the network N is **re-initialized** using the



($\in R^{C_o\times\kappa\times\kappa\times C_i}$). F_l^r is initialized using the default initialization distribution. For example, PyTorch uses Kaiming uniform [19] for convolution layers.

CVPR 2021

https://arxiv.org/abs/2103.05152

WISE-FT

Robust fine-tuning of zero-shot models

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo-Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, Ludwig Schmidt

Large pre-trained models such as CLIP or ALIGN offer consistent accuracy across a range of data distributions when performing zero-shot inference (i.e., without fine-tuning on a specific dataset). Although existing fine-tuning methods substantially improve accuracy on a given target distribution, they often reduce robustness to distribution shifts. We address this tension by introducing a simple and effective method for improving robustness while fine-tuning: ensembling the weights of the zero-shot and fine-tuned models (WiSE-FT). Compared to standard fine-tuning, WiSE-FT provides large accuracy improvements under distribution shift, while preserving high accuracy on the target distribution. On ImageNet and five derived distribution shifts, WiSE-FT improves accuracy under distribution shift by 4 to 6 percentage points (pp) over prior work while increasing ImageNet accuracy by 1.6 pp. WiSE-FT achieves similarly large robustness gains (2 to 23 pp) on a diverse set of six further distribution shifts, and accuracy gains of 0.8 to 3.3 pp compared to standard fine-tuning on seven commonly used transfer learning datasets. These improvements come at no additional computational cost during fine-tuning or inference.

CVPR 2022

https://arxiv.org/abs/2109.01903

WISE-FT

Step 1: Standard fine-tuning. As in Section 2, we let $\mathcal{S}_{\text{ref}}^{\text{tr}}$ denote the dataset used for fine-tuning and g denote the image encoder used by CLIP. We are now explicit in writing $g(x, \mathbf{V}_{\text{enc}})$ where x is an input image and $\mathbf{V}_{\text{enc}}$ are the parameters of the encoder g. Standard fine-tuning considers the model $f(x, \theta) = g(x, \mathbf{V}_{\text{enc}})^\top \mathbf{W}_{\text{classifier}}$ where $\mathbf{W}_{\text{classifier}} \in \mathbb{R}^{d \times k}$ is the classification head and $\theta = [\mathbf{V}_{\text{enc}}, \mathbf{W}_{\text{classifier}}]$ are the parameters of f. We then solve $\arg\min_\theta \left\{ \sum_{(x_i, y_i) \in \mathcal{S}_{\text{ref}}^{\text{tr}}} \ell(f(x_i, \theta),\ y_i) + \lambda R(\theta) \right\}$ where ℓ is the cross-entropy loss and R is a regularization term (e.g., weight decay). We consider the two most common variants of fine-tuning: end-to-end, where all values of θ are modified, and fine-tuning only a linear classifier, where $\mathbf{V}_{\text{enc}}$ is fixed at the value learned during pre-training. Appendices D.2 and D.3 provide additional details.

Step 2: Weight-space ensembling. For a *mixing coefficient* $\alpha \in [0, 1]$, we consider the *weight-space ensemble* between the zero-shot model with parameters θ_0 and the model obtained via standard fine-tuning with parameters θ_1. The predictions of the weight-space ensemble wse are given by

$$\mathsf{wse}(x, \alpha) = f(x, (1 - \alpha) \cdot \theta_0 + \alpha \cdot \theta_1) \ , \tag{1}$$

i.e., we use the element-wise weighted average of the zero-shot and fined-tuned parameters. When fine-tuning only the linear classifier, weight-space ensembling is equivalent to the traditional output-space ensemble [20, 11, 26] $(1 - \alpha) \cdot f(x, \theta_0) + \alpha \cdot f(x, \theta_1)$ since Equation 1 decomposes as $(1 - \alpha) \cdot g(x, \mathbf{V}_{\text{enc}})^\top \mathbf{W}_{\text{zero-shot}} + \alpha \cdot g(x, \mathbf{V}_{\text{enc}})^\top \mathbf{W}_{\text{classifier}}$.

https://arxiv.org/abs/2109.01903

Git Re-basin

Git Re–Basin: Merging Models modulo Permutation Symmetries

Samuel K. Ainsworth, Jonathan Hayase, Siddhartha Srinivasa

The success of deep learning is thanks to our ability to solve certain massive non–convex optimization problems with relative ease. Despite non–convex optimization being NP–hard, simple algorithms –– often variants of stochastic gradient descent –– exhibit surprising effectiveness in fitting large neural networks in practice. We argue that neural network loss landscapes contain (nearly) a single basin, after accounting for all possible permutation symmetries of hidden units. We introduce three algorithms to permute the units of one model to bring them into alignment with units of a reference model. This transformation produces a functionally equivalent set of weights that lie in an approximately convex basin near the reference model. Experimentally, we demonstrate the single basin phenomenon across a variety of model architectures and datasets, including the first (to our knowledge) demonstration of zero–barrier linear mode connectivity between independently trained ResNet models on CIFAR–10 and CIFAR–100. Additionally, we identify intriguing phenomena relating model width and training time to mode connectivity across a variety of models and datasets. Finally, we discuss shortcomings of a single basin theory, including a counterexample to the linear mode connectivity hypothesis.

Subjects: **Machine Learning (cs.LG)**; Artificial Intelligence (cs.AI)
Cite as: arXiv:2209.04836 **[cs.LG]**
(or arXiv:2209.04836v1 **[cs.LG]** for this version)

Submission history

From: Samuel Ainsworth [view email]
[v1] Sun, 11 Sep 2022 10:44:27 UTC (1,268 KB)

https://arxiv.org/abs/2209.04836

Git Re-basin

1. Why does SGD thrive in the optimization of high-dimensional non-convex deep learning loss landscapes, despite being noticeably less robust in other non-convex optimization settings like policy learning (Ainsworth et al., 2021), trajectory optimization (Kelly, 2017), and recommender systems (Kang et al., 2016)?
2. Where are all the local minima? When linearly interpolating between initialization and final trained weights, why does the loss smoothly, monotonically decrease (Goodfellow & Vinyals, 2015; Frankle, 2020; Lucas et al., 2021; Vlaar & Frankle, 2021)?
3. How is it that two independently trained models with different random initializations and data batch orders inevitably achieve nearly identical performance? Furthermore, why do their training loss curves look identical?

https://arxiv.org/abs/2209.04836