Towards Scalable and Versatile Weight Space Learning

Konstantin Schürholt [1 2] **Michael W. Mahoney** [2 3 4] **Damian Borth** [1]

Abstract

Learning representations of well-trained neural network models holds the promise to provide an understanding of the inner workings of those models. However, previous work has either faced limitations when processing larger networks or was task-specific to either discriminative or generative tasks. This paper introduces the SANE approach to weight-space learning. SANE overcomes previous limitations by learning task-agnostic representations of neural networks that are scalable to larger models of varying architectures and that show capabilities beyond a single task. Our method extends the idea of *hyper-representations* towards sequential processing of subsets of neural network weights, thus allowing one to embed larger neural networks as a set of tokens into the learned representation space. SANE reveals global model information from layer-wise embeddings, and it can sequentially generate unseen neural network models, which was unattainable with previous *hyper-representation* learning methods. Extensive empirical evaluation demonstrates that SANE matches or exceeds state-of-the-art performance on several weight representation learning benchmarks, particularly in initialization for new tasks and larger ResNet architectures.

1. Introduction

The exploration of the "weight space" of neural network (NN) models, i.e., the high-dimensional space spanned by the model parameters of a population of trained NNs, allows us to gain insights into the inner workings of those models.

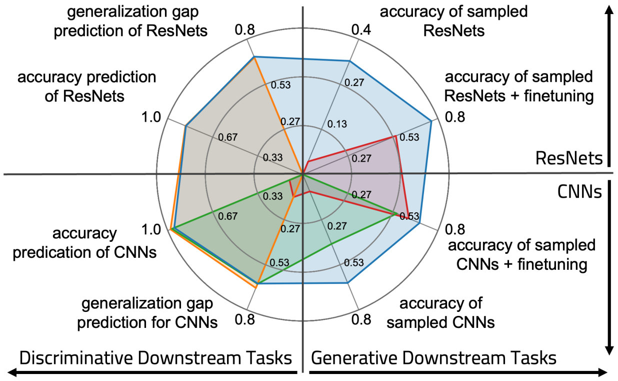

Figure 1. Aggregated results of 56 experiments showing **(left:)** four discriminative downstream tasks in R^2, and (**right:**) four generative downstream tasks in accuracy, each evaluated on (**bottom:**) CNNs model zoos trained on 4 datasets (NMIST, SVHN, CIFAR-10, STL) and (**top:**) ResNet18 model zoos trained on three datasets (CIFAR-10, CIFAR-100, Tiny-ImageNet). The colors indicate performance of **Red:** raw NN weights, **Orange:** weight statistics from Unterthiner et al. (2020), **Green:** trained *hyper-representations* from Schürholt et al. (2021; 2022a), and **Blue:** SANE (ours). While some methods perform well on specific tasks, or are restricted by the size of the underlying models, SANE can deliver excellent performance on all tasks and model sizes.

In the discriminative context, previous works aim to link weight space properties to properties such as model quality, generalization gap, or hyperparameters, using either the margin distribution (Yak et al., 2019; Jiang et al., 2019), graph topology features (Corneanu et al., 2020), or eigenvalue decompositions of weight matrices (Martin & Mahoney, 2019b; 2020; 2021; Martin et al., 2021). Some works learn classifiers to map between statistics of weights and model properties (Eilertsen et al., 2020; Unterthiner et al., 2020), or learn lower-dimensional manifolds to infer NN model properties (Schürholt et al., 2021).

In the generative context, methods have been proposed to generate model weights using (Graph) HyperNetworks (Ha et al., 2016; Zhang et al., 2019; Knyazev et al., 2021), Bayesian HyperNetworks (Deutsch, 2018), HyperGANs (Ratzlaff & Fuxin, 2019), and HyperTransformers (Zhmoginov et al., 2022). These approaches have been used for tasks such as neural architecture search, model compression, ensembling, transfer learning, and meta-learning. They have in common that they derive their learning signal from the underlying (typically image) dataset. In contrast to these

[1] AIML Lab, University of St.Gallen, St. Gallen, Switzerland [2] International Computer Science Institute, Berkeley, CA, USA [3] Lawrence Berkeley National Laboratory, Berkeley, CA, USA [4] Department of Statistics, University of California at Berkeley, CA, USA. Correspondence to: Konstantin Schürholt <konstantin.schuerholt@unisg.ch>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

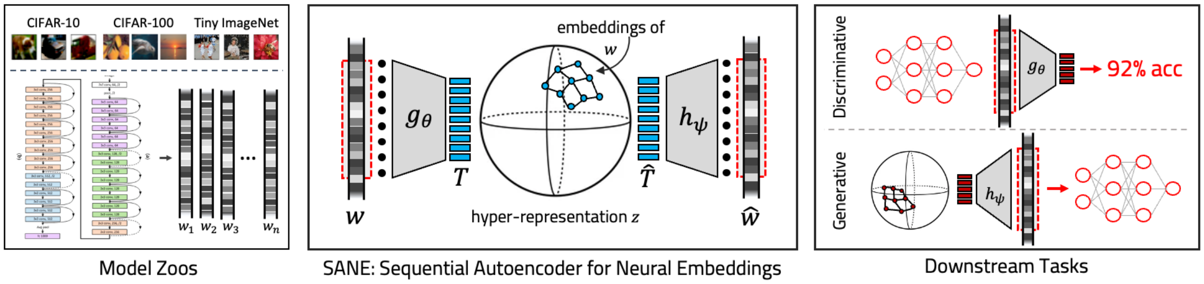

Figure 2. Given model zoos trained on different classification tasks, we extract and sequentialize the model weights. SANE trains *hyper-representations* on weights subsequences, i.e., individual layers. SANE can be used for multiple downstream tasks, either using the encoder for discriminative tasks such as the prediction of model accuracy, or the decoder for generative tasks such as sampling of new models.

methods, so-called *hyper-representations* (Schürholt et al., 2022a) learn a lower-dimensional representation directly from the weight space without the need to have access to data, e.g., the image dataset, to sample unseen NN models from that latent representation.

In this paper, we present Sequential Autoencoder for Neural Embeddings(SANE), an approach to learn task-agnostic representations of NN weight spaces capable of embedding individual NN models into a latent space to perform the above-mentioned discriminative or generative downstream tasks. Our approach builds upon the idea of *hyper-representations* (Schürholt et al., 2021; 2022a), which learn a lower-dimensional representation z from a population of NN models. This is accomplished by auto-encoding their flattened weight vectors w_i through a transformer architecture, with the bottleneck acting as a lower-dimensional embedding z_i of each NN model. While the *hyper-representation* method promises to be useful for discriminative and generative tasks, until now, separate *hyper-representations* had to be trained specifically for either discriminative or generative tasks. Additionally, existing approaches have a major shortcoming: the underlying encoder-decoder model has to embed the entire flattened weight vectors w_i at once into the learned lower-dimensional representation z. This drastically limits the size of NNs that can be embedded. SANE addresses these limitations by decomposing the entire weight vector w_i into layers or smaller subsets, and then sequentially processes them. Instead of encoding the entire NN model by one embedding, SANE encodes a potentially very large NN as multiple embeddings. The change from processing the entire flattened weight vector to subsets of weights is motivated by Martin & Mahoney (2019a; 2021), who showed that global model information is preserved in the layer-wise components of NNs. An illustration of our approach can be found in Fig. 2.

To evaluate SANE , we analyze how NN embeddings encoded by SANE behave in comparison to Martin & Mahoney (2019a; 2021) quality measures. We show that some of these weight matrix quality metrics show similar characteristics as the embeddings produced by SANE . This holds not only for held-out NN models of the model zoo used for training SANE but also for NN models of out-of-distribution model zoos with different architectures and training data. Further, we demonstrate that SANE can learn *hyper-representations* of much larger NN models, and so it makes them applicable to real-world problems. In particular, the models in the ResNet model zoo used for training are three orders of magnitudes larger than all model zoos used for *hyper-representation* learning in previous works. While previous *hyper-representation* learning methods were structurally constrained to encode the entire NN model at once, SANE is scalable by applying its sequential approach to encode layers or subsets of weights into hyper-representation embeddings. While we demonstrate scaling up to ResNet-101 models, SANE is not fundamentally limited to that size. Finally, we evaluate SANE on both discriminative and generative downstream tasks. For discriminate tasks, we evaluate on six model zoos by linear-probing for properties of the underlying NN models. For generative tasks, we evaluate on seven model zoos by sampling targeted model weights for initialization and transfer learning.

We provide an aggregated overview of our results in Fig. 1. On very small CNN models (evaluated on MNIST, SVHN, CIFAR-10, and STL, which we include for comparison with prior work), SANE performs as well as previous state-of-the-art (SOTA) in discriminative tasks. In generative downstream tasks, SANE outperforms SOTA by 25% in accuracy for initialization on the same task and 17% in accuracy for finetuning to new tasks. On larger models such as ResNets (evaluated on CIFAR-10, CIFAR-100, Tiny-ImageNet, which were beyond the capabilities of prior work), we show results comparable to baselines for discriminative downstream tasks, and we report outperformance to baselines for generative downstream tasks by 31% for initialization and 28% for finetuning to new tasks. Additionally, we show that SANE can sample targeted models by prompting with different architectures than it used for training. These sampled models can outperform models trained from scratch on the prompted architecture. Code is available at github.com/HSG-AIML/SANE.

2. Methods

Hyper-representations learn an encoder-decoder model on the weights of NNs (Schürholt et al., 2021):

$$\mathbf{z} = g_\theta(\mathbf{W}) \tag{1}$$

$$\widehat{\mathbf{W}} = h_\psi(\mathbf{z}), \tag{2}$$

where g_θ is the encoder which maps the flattened weights $\mathbf{W}$ to embeddings $\mathbf{z}$, and h_ψ decodes back to reconstructed weights $\widehat{\mathbf{W}}$. Even though previous work realized both encoder and decoder with transformer backbones, the weight vector had to be of fixed size, and models are represented in a global embedding space (Schürholt et al., 2021; 2022a). *Hyper-representations* are trained with a reconstruction loss $\mathcal{L}_{rec} = \|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2$ and contrastive guidance loss $\mathcal{L}_c = NTXent(p_\phi(\mathbf{z_i}), p_\phi(\mathbf{z_j}))$, where p_ϕ is a projection head. Schürholt et al. (2021) proposed weight permutation, noise, and masking as augmentations to generate views i, j of the same model.

Existing *hyper-representation* methods have two major limitations: i) using the full weight vector to compute global model embeddings becomes infeasible for larger models; and ii) they can only embed models that share the architecture with the original model zoo. Our SANE method addresses both of these limitations. To make models more digestible for pretraining and inference, we propose to express models as sequences of token vectors. To address i), SANE learns per-token embeddings, which are trained on subsequences of the full base model sequence. This way, the memory and compute load are decoupled from the base model size. By decoupling the tokenization from the representation learning, we also address ii). The models in the model zoo set can have varying architectures, as long as they are expressed as a sequence with the same token-vector size. The transformer backbone and per-token embeddings also allow changes to the length of the sequence during or after training. Below, we provide technical details on SANE . We first provide details on pretraining SANE , computing model embeddings, and sampling models; and we then introduce additional *aligning*, *haloing*, and *bn-conditioning* methods to stabilize training and inference.

SANE: Sequential Autoencoder for Neural Embeddings To tokenize weights, we reshape the weights $\mathbf{W}_{raw} \in \mathbb{R}^{c_{out} \times c_1 \times \dots \times c_{in}}$ to 2d matrices $\mathbf{W} \in \mathbb{R}^{c_{out} \times c_r}$, where c_{out} are the outgoing channels, and where c_r the remaining, flattened dimensions. We then slice the weights row-wise, along the outgoing channel. Using global token size d_t, we split the slices into multiple parts if $c_r > d_t$ and zero-pad to fill up to d_t. For weights $\mathbf{W}_l$ of layer l, this gives us tokens $\mathbf{T}_l \in \mathbb{R}^{n_l \times d_t}$, where $n_l = c_{out,l}\, \text{ceil}(\frac{c_r}{d_t})$. Since all tokens $\mathbf{T}_l$ share the same token size, the tokens of layer $l = 1, ..., L$ can be concatenated to get the model token sequence $\mathbf{T} \in \mathbb{R}^{N \times d_t}$. To indicate the position of a token, we use a 3-dimensional position $\mathbf{P}_n = [n, l, k]$, where $n \in [1, N]$ indicates the global position in the sequence, $l \in [1, L]$ indicate the layer index, and $k \in [1, K(l)]$ is the position of the token within the layer.

Out of the full token sequence $\mathbf{T}$ and positions $\mathbf{P} \in \mathbb{N}^{N \times 3}$, we take a random consecutive sub-sequence $\mathbf{T}_{s,n} = \mathbf{T}_{n,\dots,n+ws}$ with positions $\mathbf{P}_{s,n} = \mathbf{P}_{n,\dots,n+ws}$ of length ws. We call these sub-sequences windows and the length of the sub-sequence the window size ws.

For SANE on windows of tokens, we extend Eqs. 1 and 2 to encode and decode token windows as

$$\mathbf{z}_{s,n} = g_\theta(\mathbf{T}_{s,n}, \mathbf{P}_{s,n}) \tag{3}$$

$$\widehat{\mathbf{T}}_{s,n} = h_\psi(\mathbf{z}_{s,n}, \mathbf{P}_{s,n}), \tag{4}$$

where $\mathbf{z}_{s,n} \in \mathbb{R}^{ws \times d_z}$ is the per-token latent representation of the window. In contrast to *hyper-representations* Eqs. 1 and 2 which operate on the full flattened weights of a model, SANE encodes sub-sequences of tokenized models. For simplicity, we apply linear mapping to and from the bottleneck, to reduce tokens from d_t to d_z.

We adapt the composite training loss of *hyper-representations*, $\mathcal{L} = (1 - \gamma)\mathcal{L}_{rec} + \gamma\mathcal{L}_c$, for sequences as:

$$\mathcal{L}_{rec} = \|\mathbf{M}_{s,n} \odot \left(\mathbf{T}_{s,n} - \widehat{\mathbf{T}}_{s,n}\right)\|_2^2 \tag{5}$$

$$\mathcal{L}_c = NTXent(p_\phi(\mathbf{z_{s,n,i}}), p_\phi(\mathbf{z_{s,n,j}})). \tag{6}$$

Here, the mask $\mathbf{M}_{s,n}$ indicates signal with 1 and padding with 0, to ensure that the loss is only computed on actual weights. The contrastive guidance loss uses the augmented views i, j and projection head p_ϕ.

The pretraining procedure is detailed in Algorithm 1. We preprocess model weights by standardizing weights per layer and aligning all models to a reference model; see *Model Alignment* below. As in previous work (Schürholt et al., 2021; Peebles et al., 2022), the encoder and decoder are realized as transformer blocks. Training on the full sequence would memory-limit the base-model size by its sequence length. Training the encoder and decoder on windows instead of the full model sequence decouples the memory requirement from the base model's full sequence length. The window size can be used to balance GPU memory load

Algorithm 1 SANE pretraining

```
Input: population of models
i: standardize models weights
ii: align models to one common reference model
iii: tokenize models to tokens $\mathbf{T}$, positions $\mathbf{P}$, masks $\mathbf{M}$
iv: draw $k$ windows per model: $\mathbf{T}_{s,n}$, $\mathbf{P}_{s,n}$, $\mathbf{M}_{s,n}$
v: train on $\mathcal{L}_{train}$ until convergence of $\mathcal{L}_{val}$
```

and the amount of context information. Notably, since we disentangle the tokenization from the representation learning model, SANE also allows us to embed sequences of models with varying architectures, as long as their token size is the same. To prevent potential overfitting to specific window positions, we propose to sample windows from each model sequence multiple times randomly.

Computing SANE Model Embeddings. SANE can be used to analyze models in embedding space, e.g., by using embeddings as features to predict properties such as accuracy or to identify other model quality metrics. In contrast to *hyper-representations*, SANE can embed different model sizes and architectures in the same embedding space. To embed any model, we begin by preprocessing weights by standardizing per layer and aligning models to a pre-defined reference model (see *Model Alignment* below). Subsequently, the preprocessed models are tokenized as described above. For short model sequences, the embedding sequences can be directly computed as $\mathbf{z} = g_\theta(\mathbf{T}, \mathbf{P})$. For larger models, the token sequences are too long to embed as one. We therefore employ *haloing* (see below) to encode the entire sequence as coherent subsequences. Algorithm 2 summarizes the embedding computation. To compare different models in embedding space, we aggregate the sequences of token embeddings. To that end, we understand the token sequence of one model to form a surface in embedding space and choose to represent that surface by its center of gravity. That is, we take the mean of all tokens along the embedding dimension as $\bar{\mathbf{z}} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{z}_n)$. That results in one vector in embedding space per model. Of course, one could use other aggregation methods with SANE .

Algorithm 2 SANE model embedding computation

```
Input: population of models
i: preprocessing: standardize and align model weights
ii: tokenize models: $\mathbf{T}$, positions $\mathbf{P}$, property $y$
iii: split $\mathbf{T}$, $\mathbf{P}$ to consecutive chunks $\mathbf{T}_{hs,n}, \mathbf{P}_{hs,n}$
iv: compute embeddings $\mathbf{z}_{hs,n} = g_\theta(\mathbf{T}_{hs,n}, \mathbf{P}_{hs,n})$
v: stitch model embeddings $\mathbf{z}$ together from chunks $\mathbf{z}_{hs,n}$
```

Sampling Models with Few Prompt Examples. Sampling models from SANE promises to transfer knowledge from existing populations to new models with different architectures. Given pretrained encoders g_θ and decoder h_ψ, the challenge is to identify the distribution $\mathcal{P}$ in latent space which contains the targeted properties. To approximate that distribution, previous work used a large number of well-trained models (Peebles et al., 2022; Schürholt et al., 2022a). However, increasing the size of the sampled models makes generating a large number of high-performance models exceedingly expensive. Instead of using expensive high-performance models to model $\mathcal{P}$ directly, we propose to find a rough estimate of $\mathcal{P}$, sample broadly, and refine $\mathcal{P}$ using the signal from the sampled models. Using E prompt examples $\mathbf{W}^e$ we compute the token sequence $\mathbf{T}^e$ and corresponding embedding sequence $\mathbf{z}^e = g_\theta(\mathbf{T}^e, \mathbf{P})$. Following previous work, we model the distribution $\mathcal{P}$ with a Kernel Density Estimation (KDE) per token as $\mathcal{P}_{e\in E}(\mathbf{z}_n^e)$ (Schürholt et al., 2022a). We then draw k new token samples as:

$$\mathbf{z}_n^k \sim \mathcal{P}_{e\in E}(\mathbf{z}_n^e). \tag{7}$$

We reconstruct the sampled embeddings to weight tokens $\mathbf{T}^k = h_\psi(\mathbf{z}^k, \mathbf{P})$ and then weights $\mathbf{W}^k$. Sampling tokens can be done cheaply, decoding and evaluating the weights using some performance metric involves only forward passes and is likewise cheap. Therefore, one can draw a large amount of samples and keep only the top m models, according to the performance metric. We call this method *subsampling*. The process can be refined iteratively, by re-using the embeddings $\mathbf{z}^k$ of the best models as new prompt examples, to adjust the sampling distribution to best fit the needs of the performance metric. We call this sampling method *bootstrapped*. By only requiring a rough version of $\mathcal{P}$ and refining with the target signal, our sampling strategy reduces requirements on prompt examples such that only very few and slightly trained prompt examples are necessary. The overall sampling method is outlined in Algorithm 3. It makes use of *model alignment*, *haloing*, and *batch-norm conditioning* which are detailed below. In addition to the compute efficiency, these sampling methods learn the distribution of targeted models in embedding space. Further, they are not bound to the distribution of prompt examples, but instead they can find the distribution that best satisfies the target performance metric, independent of the prompt examples.

Algorithm 3 Sampling models with SANE

```
Input: model prompt examples $\mathbf{W}^e$
i: tokenize prompt examples: tokens $\mathbf{T}^e$, positions $\mathbf{P}^e$
ii: embed prompt examples $\mathbf{z}^e$ following Alg. 2
for $i_{boot} = 1$ to bootstrap iterations do
  iii: draw $k$ samples $\mathbf{z}_n^k \sim \mathcal{P}_{e\in E}(\mathbf{z}_n^e)$
  iv: decode to tokens $\mathbf{T}^{\mathbf{k}} = h_\psi(\mathbf{z}^k)$
  v: apply batch-norm conditioning
  vi: compute target metric and keep best $m$ models
  if bootstrap iterations $> 1$ then
    vii: $\mathbf{z}^e = \mathbf{z}^k for\ k \in m$
  end if
end for
```

Growing sample model size poses several additional challenges, three of which we address with the following methods. We evaluate these methods in Appendix A.

Model Alignment. Symmetries in the weight space of NN complicate representation learning of the weights. The number of symmetries grows fast with model size (Bishop, 2006). To make representation learning easier, we reduced all training models to a unique, canonical basis of a reference model. With reference model A we align model B

by finding the permutation $\pi = argmin_{\pi} \|\text{vec}(\Theta(A)) - \text{vec}(\Theta(B))\|^2$, where $\Theta(A)$ are the parameters of model A (Ainsworth et al., 2022). We fix the same reference model across all dataset splits and use the last epoch of each model to determine the permutation for that model.

Haloing. The sequential decomposition of SANE decouples the pretraining sequence length from downstream task sequence lengths. Since the memory load at inference is considerably lower, the sequences at inference can be longer. However, full model sequences may still not fit in memory and may have to be processed in slices. To ensure consistency between the slices, we add context around the content windows. With added context halo before and after the content window, we get $\mathbf{T}_{hs,n} = \mathbf{T}_{n-h,..,n,...,n+ws,n+ws+h}$. Similar to approaches in computer vision (Vaswani et al., 2021), this context halo is added for the pass through encoder and decoder, but disregarded after.

Batch-Norm Conditioning. In most current NN models, some parameters like batch-norm weights are updated during forward passes instead of with gradients. Since that makes them structurally different, we exclude these parameters from representation learning and sampling with SANE . Nonetheless, these parameters need to be instantiated for sampled models to work well. For model sampling methods, we therefore propose to condition batch-norm parameters by performing a few forward passes with some target data. Importantly, this process does not update the learned weights of the model. It serves to align the batch norm statistics with the model's weights.

3. Training SANE

We pretrain SANE following Alg. 1 on several populations of trained NN models, from the model zoo dataset (Schürholt et al., 2022c). We use zoos of small models to compare with previous work, as well as zoos containing larger ResNet-18 models. All zoos are split into training, validation, and test splits $70 : 15 : 15$.

- **Smaller CNN zoos.** The MNIST and SVHN zoos contain LeNet-style models with 3 convolution and 2 dense layers and only $\sim 2.5k$ parameters. The slightly larger CIFAR-10 and STL-10 zoos use the same architecture with wider layers and $\sim 12k$ parameters.
- **Larger ResNet zoos.** We also use the CIFAR-10, CIFAR-100, and Tiny-Imagenet zoos containing ResNet-18 models (Schürholt et al., 2022c) with $\sim 12M$ parameters to evaluate scalability to large models.

Pretraining. We train SANE using Alg. 1. As augmentations, we use noise and permutation. The permutation is computed relative to the aligned model. For contrastive learning, the aligned model serves as one view, and a permuted version as the second view.

Implementation Details. To maintain diversity within each batch, we select only a single window from each model. Loading, preprocessing, and augmenting the entire sample, only to use ca. 1% of it, is infeasible. To address this, we leverage FFCV (Leclerc et al., 2023) to compile datasets consisting of sliced and permuted windows of models. Each model is super-sampled for approximately full coverage within the training set, considering the ratio of window length to sequence length. For the ResNet zoos, we include 140 models per zoo, a number that remains manageable in terms of memory and storage. We train for 50 epochs using a OneCycle learning rate scheduler (Smith & Topin, 2018). Seeds are recorded to ensure reproducibility. We build SANE in PyTorch (Paszke et al., 2019), using automatic mixed precision and flash attention (Dao et al., 2022) to enhance performance. We use ray.tune (Liaw et al., 2018) for hyperparameter optimization.

4. Embedding Analysis

In this section, we analyze the embeddings of SANE and compare to the weight-analysis methods *WeightWatcher* (WW) (Martin et al., 2021). We focus on three aspects: i) global relation between accuracy and embeddings; ii) the trend of embeddings over layer index, as in (Martin et al., 2021); and iii) the identification of training phases as in (Martin & Mahoney, 2019b; 2021).

To analyze weights, we focus on two WW metrics which in previous work reveal model performance as well as internal model composition (*correlation flow*); the log spectral norm $\log(\|\mathbf{W}\|^2_\infty)$ and weighted α, the coefficient of the power law fitted to the empirical spectral density (Martin et al., 2021). These two metrics describe different aspects of the eigenvalue distribution. To get a similar signal on the internal dependency of weight matrices, we compute per-layer scalars $\hat{z}_l$ as the spread of the tokens of one layer in hyper-representation space, i.e., their standard deviation:

$$\hat{z}_l = std_t(\mathbf{z}^t_m) \tag{8}$$

$$\mathbf{z}^t_m = g(\mathbf{W}^t_m), \tag{9}$$

where g is the hyper-rep encoder, $\mathbf{z}^t_m$ are the stacked tokens t of layer m, and $\mathbf{W}^t_m$ is the weight-slice t of layer m.

To compare WW metrics to SANE , we pretrain SANE on a Tiny-Imagenet ResNet-18 zoo and compute the two metrics on ResNets and VGGs of different sizes trained on ImageNet from pytorchcv (Sémery, 2024). On both ResNets in Figure 3, 9 and VGGs in Figure 8, the WW metrics and our embeddings show similar global trends. On ResNets, our embeddings and WW have low values at early layers and a sharp increase at the end. However, our embeddings add an additional step for intermediate layers, which may indicate that SANE is sensitive to a higher degree of variation in these layers which previous work found by comparing activations (Kornblith et al., 2019).

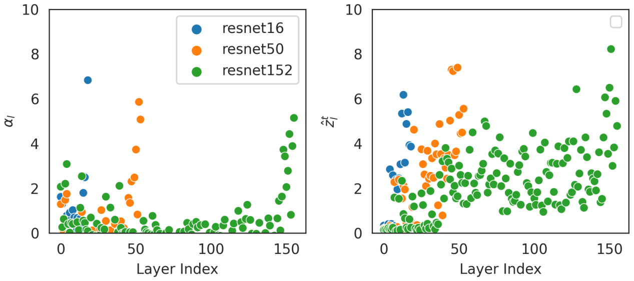

Figure 3. Comparison between WeightWatcher (WW) features (left) and `SANE` (right). Features over layer index for ResNets from pytorchcv of different sizes.

In a second experiment, we aggregate the layer-wise embeddings $\hat{z}_l$ to evaluate relations to model accuracy in Figures 4, 10 and 11, similar to previous work (Martin et al., 2021). On models from pytorchcv and the Tiny-ImagNet model zoo from (Schürholt et al., 2022c), the WW features and `SANE` embeddings both show strong correlations to model accuracy. However, while the WW metrics are negatively correlated to accuracy, our embeddings are positively correlated to accuracy. The reason for that may lie in the additional 'step' in Figure 3. That is, larger models with more layers generally have higher performance. As Figure 3 shows, more layers add very small values reducing the global average for WW metrics. For our embeddings, deeper models have more layers with higher $\hat{z}_l$ values, due to the afore-mentioned step. This increases the global model average with growing model size. Lastly, we compare the eigenvalue spectrum to embeddings. Previous work identified distinct shapes at different training phases or with varying training hyperparameters (Martin & Mahoney, 2019b; 2021). While we can replicate the distributions of the eigenvalues, the distributions of our embeddings only show the change from early phases of training to the heavy-tailed distribution; see Figure 7.

In summary, our embedding analysis indicates that `SANE` represents several aspects of model quality (globally and on a layer level) that have been established previously.

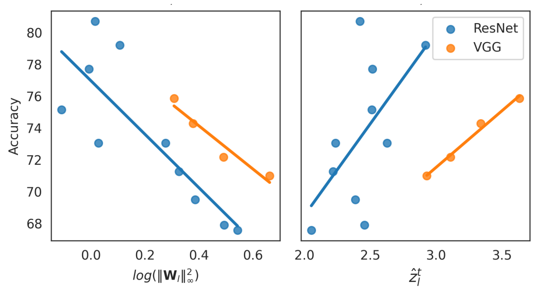

Figure 4. Comparison between WeightWatcher features (left) and `SANE` (right). Accuracy over model features for ResNets and VGGs from pytorchcv of different sizes. Although `SANE` is pre-trained in a self-supervised fashion, it preserves the linear relation of a globally-aggregated embedding to model accuracy.

5. Empirical Performance

In this section, we describe the general performance `SANE` .

5.1. Predicting Model Properties

We evaluate `SANE` for discriminative downstream tasks as a proxy for encoded model qualities. Specifically, we investigate whether `SANE` matches the predictive performance of *hyper-representations* on small CNN models (Table 1) and whether similar performance can be achieved on ResNet-18 models (Table 2). To that end, we compute model embeddings $\bar{\mathbf{z}}$ as outlined in Alg. 2, and we compare against flattened weights W and weight statistics $s(W)$. Following the experimental setup of (Eilertsen et al., 2020; Unterthiner et al., 2020; Schürholt et al., 2021), we compute embeddings using the three methods and linear probe for test accuracy (`Acc`), epoch (`Ep`), and generalization gap (`Ggap`). We again use trained models from the modelzoo repository (Schürholt et al., 2022c), with the same train, test, val splits as above.

Table 1. Property prediction on populations of small CNNs used in previous work (Schürholt et al., 2021). We report the regression R^2 on the test set prediction test accuracy `Acc.`, epoch `Ep.` and generalization gap `Ggap` for linear probing with model weights W, model weights statistics $s(W)$ or `SANE` embeddings as inputs.

	MNIST			SVHN			CIFAR-10 (CNN)		
	W	S(W)	SANE	W	S(W)	SANE	W	S(W)	SANE
ACC.	0.965	**0.987**	0.978	0.910	0.985	**0.991**	-7.580	**0.965**	0.885
EP.	0.953	**0.974**	0.958	0.833	**0.953**	0.930	0.636	**0.923**	0.771
GGAP	0.246	0.393	**0.402**	0.479	0.711	**0.760**	0.324	**0.909**	0.772

`SANE` matches baselines on small models. The results of linear probing on small CNNs in Tables 1 and 9 confirm the performance of W (low) and $s(W)$ (very high) of previous work. `SANE` embeddings show comparably high performance to the $s(W)$ and previous *hyper-representations*. Additional experiments in Appendix D.1 compare to previous work and confirm these findings. Sequential decomposition and representation learning as well as using the center of gravity does not significantly reduce the information contained in `SANE` embeddings.

Table 2. Property prediction on ResNet-18 model zoos of (Schürholt et al., 2022c). We report the regression R^2 on the test set prediction test accuracy `Acc.`, epoch `Ep.` and generalization gap `Ggap` for linear probing with model weights statistics $s(W)$ or `SANE` embeddings as inputs.

	CIFAR-10		CIFAR-100		TINY-IMAGENET	
	S(W)	SANE	S(W)	SANE	S(W)	SANE
ACC.	0.880	0.879	0.923	0.922	0.802	0.795
EP.	0.999	0.999	0.999	0.992	0.999	0.980
GGAP	0.490	0.512	0.882	0.879	0.704	0.699

SANE performance prediction scales to ResNets. Both $s(W)$ and SANE embeddings show similarly high performance on populations of ResNet-18s; see Table 2. On ResNet-18s, using the full weights W for linear probing is infeasible due to the size of the flattened weights. SANE matches the high performance of $s(W)$. The results show that sequential *hyper-representations* are capable of scaling to ResNet-18 models. Further, the aggregation even of long sequences (ca. 50k tokens) embedded in SANE preserves meaningful information on model performance, which indicates the feasibility of applications like model diagnostics or targeted sampling.

5.2. Generating Models

We evaluate SANE for the generative downstream tasks i.e., for sampling model weights. We generate weights following Alg. 3 and test them in fine-tuning, transfer learning, and how they generalize to new tasks and architectures. In the following paragraphs, we begin with experiments on small CNN models from the modelzoo repository to compare with previous work (Tables 3, 13). Subsequently, we evaluate SANE for sampling ResNet-18 models for finetuning and transfer learning (Tables 4, 14). Lastly, we evaluate sampling for new tasks and new architectures using only few prompt examples (Figure 5 and Tables 5, 15, 16, 17).

We pretrain SANE on models from the first half of the training epochs with Alg. 1, and keep the remaining epochs (26-50) as holdout to compare against, following the experimental setup of (Schürholt et al., 2022a). We sample using Alg. 3 and use models from the last epoch in the pretraining set (epoch 25) as prompt examples. We denote subsampling with SANE$_{SUB}$ and iteratively updating the distribution $\mathcal{P}$ as SANE$_{BOOT}$. To evaluate the impact of the sampling method, we also combine SANE with the $KDE30$ sampling approach that uses high-quality prompt examples (Schürholt et al., 2022a). We further evaluate sampling without prompt examples by bootstrapping off of a Gaussian prior $\mathcal{P}$, denoted as SANE$_{GAUSS}$. We compare against training from scratch, as well as fine-tuning from the prompt examples.

Sampling High-Performing CNNs Zero-Shot. We begin with finetuning and transfer learning experiments on small CNNs from the modelzoo dataset to validate that the sequential decomposition for pretraining and sampling does not hurt performance. The results of these experiments show dramatically improved performance zero-shot for fine-tuning and transfer learning over previous *hyper-representations*; see Tables 3 and 13. At epoch 0, SANE improves over previous *hyper-representations* S_{KDE30} by almost 20%. The effect becomes smaller during fine-tuning. Nonetheless, SANE consistently outperforms training from scratch with a higher epoch budget, often by several percentage points. This demonstrates on small CNNs that sequential pretraining and sampling of SANE improves performance, particularly zero shot. This indicates the potential for scenarios with little labelled data.

Table 3. Model generation on CNN model populations fine-tuned on the same task. We compare training from scratch with S_{KDE30} from (Schürholt et al., 2022a), SANE combined with the $KDE30$ sampling method, and our SANE subsampled. Each of the sampled populations is fine-tuned over 25 epochs.

Ep.	Method	MNIST	SVHN	CIFAR-10	STL
0	tr. fr. scratch	~10 /%	~10 /%	~10 /%	~10 /%
	S_{KDE30}	68.6±6.7	54.5±5.9	*n/a*	*n/a*
	SANE$_{KDE30}$	84.8±0.8	70.7±1.4	56.3±0.5	39.2±0.8
	SANE$_{SUB}$	**86.7±0.8**	**72.3±1.6**	**57.9±0.2**	**43.5±1.0**
	SANE$_{GAUSS}$	20.8±0.1	21.6±0.5	19.3±0.2	17.5±1.5
1	tr. fr. scratch	20.6±1.6	19.4±0.6	37.2±1.4	21.3±1.6
	S_{KDE30}	83.7±1.3	69.9±1.6	*n/a*	*n/a*
	SANE$_{KDE30}$	85.5±0.8	71.3±1.4	58.2±0.2	43.5±0.7
	SANE$_{SUB}$	**87.5±0.6**	**73.3±1.4**	**59.1±0.3**	**44.3±1.0**
	SANE$_{GAUSS}$	61.3±3.1	24.1±4.4	27.2±0.3	22.4±1.0
5	tr. fr. scratch	36.7±5.2	23.5±4.7	48.5±1.0	31.6±4.2
	S_{KDE30}	**92.4±0.7**	57.3±12.4	*n/a*	*n/a*
	SANE$_{KDE30}$	87.5±0.7	72.2±1.2	58.8±0.4	45.2±0.6
	SANE$_{SUB}$	89.0±0.4	**73.6±1.5**	**59.6±0.3**	**45.3±0.9**
	SANE$_{GAUSS}$	83.4±0.8	35.6±8.9	43.3±0.3	34.2±0.7
25	tr. fr. scratch	83.3±2.6	66.7±8.5	57.2±0.8	44.0±1.0
	S_{KDE30}	**93.0±0.7**	74.2±1.4	*n/a*	*n/a*
	SANE$_{KDE30}$	92.0±0.3	74.7±0.8	60.2±0.6	**48.4±0.5**
	SANE$_{SUB}$	92.3±0.4	**75.1±1.0**	**61.2±0.1**	48.0±0.4
	SANE$_{GAUSS}$	**94.2±0.4**	54.2±17.6	52.2±0.6	43.5±0.5
50	tr. fr. scratch	91.1±2.6	70.7±8.8	61.5±0.7	47.4±0.9

SANE Sequential Sampling Scales to ResNets. To evaluate how well sampling with SANE scales to larger models, we continue with experiments on ResNet-18s. The results of these experiments Tables 4 and 14 show that despite the long sequences, the sampled ResNet models perform well above random initialization. For example, sampled ResNet-18s achieve 68.1% on CIFAR-10 without any fine-tuning (Table 4). These models are at least three orders of magnitude larger than previous models used for *hyper-representation* learning (Schürholt et al., 2022a), rendering it computationally infeasible for the approach presented in (Schürholt et al., 2022a) to be evaluated against. As before, the performance difference to random initialization becomes smaller during fine-tuning. Similar to our experiments on CNNs, sampled ResNet-18s achieve competitive performance or even outperform training from scratch with a considerably smaller computational budget.[1] Transferred to a new task, sampled models outperform training from scratch and match fine-tuning from prompt examples (Table 14). Interestingly, subsampling and bootstrapping appear to

[1] The base population is trained with a one-cycle learning rate scheduler. To avoid any bias, we adopt the same scheduler but train for only 10 epochs, which affects direct comparability.

Table 4. Model generation on ResNet-18 model populations fine-tuned on the same task. We compare sampled models at different epochs with models trained from scratch.

Epoch	Method	CIFAR-10	CIFAR-100	Tiny-Imagenet
0	tr. fr. scratch	∼10 /%	∼1 /%	∼0.5 /%
	SANE$_{KDE30}$	64.8±2.0	19.8±2.5	8.4±0.9
	SANE$_{SUB}$	68.1±0.7	19.8±1.3	11.1±0.5
	SANE$_{BOOT}$	**68.6±1.2**	**20.4±1.3**	**11.7±0.5**
1	tr. fr. scratch	43.7±1.3	17.5±0.7	13.8±0.8
	SANE$_{KDE30}$	82.4±0.9	59.0±1.3	46.7±0.8
	SANE$_{SUB}$	**83.6±1.5**	**60.8±0.8**	**47.4±1.0**
	SANE$_{BOOT}$	82.8±1.4	60.2±0.5	47.2±0.8
5	tr. fr. scratch	64.4±2.9	36.5±2.0	31.1±1.6
	SANE$_{KDE30}$	**85.9±0.6**	56.2±1.7	45.6±1.4
	SANE$_{SUB}$	85.4±1.3	**56.7±1.6**	45.7±0.8
	SANE$_{BOOT}$	85.4±0.7	56.4±1.2	**49.1±1.7**
10	tr. fr. scratch	76.5±2.7	49.0±2.0	39.9±2.2
	SANE$_{KDE30}$	91.4±0.1	**72.9±0.2**	**64.2±0.3**
	SANE$_{SUB}$	**91.6±0.2**	**72.9±0.1**	64.0±0.2
	SANE$_{BOOT}$	91.6±0.2	72.8±0.1	64.1±0.2
25	tr. fr. scratch	85.5±1.5	56.5±2.0	43.3±1.9
50	tr. fr. scratch	92.14±0.2	70.7±0.4	57.3±0.6
60	tr. fr. scratch	*n/a*	74.2±0.3	63.9±0.5

work well when there is a useful signal to start with, i.e., on easier tasks that are similar to the pretraining distribution. This suggests that the sampling distributions are not ideal, and may require a better fit, more samples, or iterative adjustment to fit new datasets zero-shot. Nonetheless, even the relatively naive sampling methods can successfully sample competitive models, even at the scale of ResNet-sized architectures. This shows that our sequential sampling works even for long sequences of tokens.

Subsampling Improves Performance. Previous work requires high-quality prompt examples to target specific properties (Schürholt et al., 2022a). Our sampling methods drop these requirements and use prompt examples only to model a prior. We therefore compare SANE with S_{KDE30} from (Schürholt et al., 2022a) to SANE . Further, we compare the $KDE30$ sampling method with our subsampling approach on SANE . On datasets where published results are available, using $KDE30$ with SANE improves performance over previously published results with S_{KDE30}; see Table 3 for MNIST and SVHN results, e.g., epoch 0. We credit this to the better reconstruction quality of pre-training with SANE . Further, our sampling methods improve performance over S_{KDE30}. We compare SANE + S_{KDE30} with SANE + subsampling and SANE + bootstrapping, e.g., in Table 4 on CIFAR-10 at epoch 0 from 64.8% to 68.1%, or on Tiny Imagenet from 8.4% to 11.1%. Using bootstrapping to adjust $\mathcal{P}$ iteratively further improves the sampled models slightly. It even allows to replace prompt examples with a Gaussian prior $\mathcal{P}$. The results of SANE$_{GAUSS}$ show high performance after fine-tuning, even the highest overall on MNIST. These results show that our sampling methods not only drop requirements for the prompt examples but even improve the performance of the sampled models.

Few-Shot Model Sampling Transfers to New Tasks and Architectures. Lastly, we explore whether sampling models using SANE generalizes beyond the original task and architecture with very few prompt examples. Such transfers are out of reach of previous *hyper-representations*, which are bound to a fixed number of weights. SANE , on the other hand, represents models of different sizes or architectures simply as sequences of different lengths, which can vary between pretraining and sampling. Since we use the prompt examples only to roughly model the sampling distribution, we need only a few (1-5) prompt examples which are trained for only a few epochs (1-5). That way, sampling for new architectures and/or tasks can become very efficient. We test that idea in three experiments: (i) *changing the tasks* between pretraining and prompt-examples from CIFAR-100 to Tiny-Imagenet (Table 5); (ii) *changing the architecture* between pretraining and prompt-examples from ResNet-18 to ResNet-34 (Table 15); and (iii) *changing both task and architecture* from ResNet-18 on CIFAR-100 to ResNet-34 on Tiny-Imagenet (Figure 5 and Table 16).

In all three experiments, using target prompt examples improves over random initialization as well as previous transfer experiments. This indicates that SANE representations contain useful information even for new architectures or tasks. The sampled models outperform the prompt examples and training from scratch, considerably in earlier epochs, and preserve a performance advantage throughout fine-tining.

Sampling for a new task (Table 5), the sampled models outperform the prompt examples after just two epochs of fine-tuning, which indicates that transfer-learning using SANE is an efficient alternative. Sampling from ResNet-18 to ResNet-34 for the same task (Table 15) shows likewise improved performance over training from scratch, which indicates that the learned representation generalizes to larger architectures as well. Sampling for new tasks and different architecture (Figure 5 and Table 16) combines the previous

Table 5. Sampling ResNet-18 models for Tiny-Imagenet. SANE was pretrained on CIFAR-100, 15 samples are drawn using subsampling, and 5 prompt examples are taken from the Tiny-Imagenet ResNet-18 zoo at epoch 25 with a mean accuracy of 43%.

ResNet-18 CIFAR100 to TinyImagnet		
Ep.	Method	Acc TI
0	tr. fr. scratch	0.5±0.0
	SANE	0.6±0.0
1	tr. fr. scratch	10.4±2.2
	SANE	**39.4±1.5**
2	tr. fr. scratch	28.5±0.9
	SANE	**61.0±0.2**
2	SANE ensamble	64.0

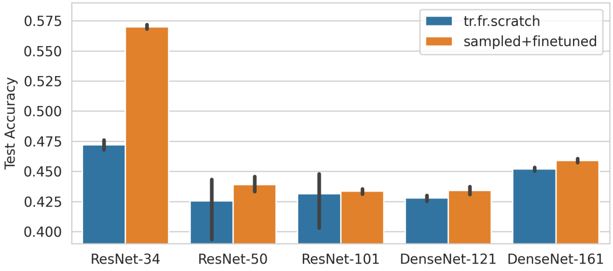

Figure 5. Comparison between sampled models and random initialization trained for 5 epochs Tiny-Imagenet. Different architectures are sampled from SANE pretrained on a ResNet-18 CIFAR-100 zoo. Although both models and tasks are changed, sampled models perform better.

experiments and confirms their results. Sampled models outperform training from scratch by a considerable margin. Figure 5 indicates that with increasing distance from the pretraining architecture for SANE , the performance gain of sampled models decreases, e.g., with increasing ResNet size. Additionally, since sampling models using SANE is cheap and lends itself to ensembling, we investigate the diversity of sampled models in Appendix E.1. Taken together, the experiments show that SANE learns representations that can generalize beyond the pretraining task and architecture, and can efficiently be sampled for both new tasks and architectures.

Limitations

In this paper, we pretrain SANE on homogeneous zoos with one architecture. This simplifies alignment for pre-training, but more importantly it simplifies evaluation for model generation. Since SANE can train on varying architectures and model sizes, the model population requirement for pre-training is significantly relaxed. A sufficient number of models are available on public model hubs. Further, our sampling method requires access to prompt examples, to have an informed prior from which to sample. For small models, bootstrapping from a Gaussian finds the targeted distribution; see SANE$_{GAUSS}$ in Table 3. For large models with correspondingly long sequences, that approach is too expensive, which is why we rely on prompt examples. Lastly, in this paper, we perform experiments only on computer vision tasks. This is a choice to simplify the experiment setup.

6. Related Work

Representation learning in the space of NN weights has become a growing field recently. Several methods with different approaches to deal with weight spaces have been proposed to predict model properties such as accuracy (Unterthiner et al., 2020; Eilertsen et al., 2020; Andreis et al., 2023; Zhang et al., 2023) or to learn the encoded concepts (Ashkenazi et al., 2022; De Luigi et al., 2023). Other work investigates the structure of trained weights on a fundamental level, using their eigen or singular value decompositions to identify training phases or predict properties (Martin & Mahoney, 2019b; 2020; Martin et al., 2021; Martin & Mahoney, 2021; Yang et al., 2022; Meller & Berkouk, 2023). Taking an optimization perspective, other work has investigated the uniqueness of the basis of trained NNs (Ainsworth et al., 2022; Brown et al., 2023). Other work identifies subspaces of weights that are relevant, which motivates our work (Benton et al., 2021; Lucas et al., 2021; Wortsman et al., 2021; Fort & Jastrzebski, 2019). The mode connectivity of trained models has been investigated to improve understanding of how to train models (Draxler et al., 2018; Nguyen, 2019; Frankle et al., 2019).

A different line of work trains models to generate weights for target models, such as HyperNetworks (Ha et al., 2016; Nguyen et al., 2019; Zhang et al., 2019; Knyazev et al., 2021; 2023; Kofinas et al., 2024), with a recurrent backbone (Wang et al., 2023) as learned initialization (Dauphin & Schoenholz, 2019) or for meta learning (Finn et al., 2017; Zhmoginov et al., 2022; Nava et al., 2022). While the last category uses data to get learning signals, another line of work learns representations of the weights directly. Hyper-Representations train an encoder-decoder architecture using reconstruction of the weights, with contrastive guidance, and has been proposed to predict model properties (Schürholt et al., 2021) or generate new models (Schürholt et al., 2022b;a). While previous work was limited to small models of fixed length, this paper proposes methods to decouple the representation learner size from the base model. Related approaches use convolutional auto-encoders (Berardi et al., 2022) or diffusion on the weights (Peebles et al., 2022).

7. Conclusion

In this work, we propose SANE , a method to learn task-agnostic representations of Neural Network models. SANE decouples model tokenization from *hyper-representation* learning and can scale to much larger neural network models and generalize to models of different architectures. We analyze SANE embeddings and find they reveal model quality metrics. Empirical evaluations show that i) SANE embeddings contain information on model quality both globally and on a layer level, ii) SANE embeddings are predictive of model performance, and iii) sampling models with SANE achieves higher performance and generalizes to larger models and new architectures. Further, we propose sampling methods that reduce quality and quantity requirements for prompt examples and allow targeting new model distributions.

Acknowledgements

KS and DB would like to acknowledge the Google Research Scholar Award and the Swiss National Science Foundation for partial funding of this work. We are thankful to Erik Vee at Google Research for the insightful discussions, Léo Meynent and Joelle Hanna for editorial support, and Harald Rotter, Kurt Städler, and Martin Eigenmann for the support with the computational infrastructure which made this project possible. MWM would like to acknowledge the NSF, DOE, and IARPA for partial support of this work.

Impact Statement

This paper introduces a novel weight representation learning method designed to enhance the performance and scalability of machine learning models across various applications. As a fundamental approach, it serves as a foundation for future advancements in the field of machine learning. SANE holds potential for use in both academic research and industry applications and thus inherits all their benefits but also risks for adverse applications of machine learning. Its versatility and scalability make SANE a valuable tool that may also offer insights into model interpretability.

References

Ainsworth, S. K., Hayase, J., and Srinivasa, S. Git Re-Basin: Merging Models modulo Permutation Symmetries, September 2022.

Andreis, B., Bedionita, S., and Hwang, S. J. Set-based Neural Network Encoding, May 2023.

Ashkenazi, M., Rimon, Z., Vainshtein, R., Levi, S., Richardson, E., Mintz, P., and Treister, E. NeRN – Learning Neural Representations for Neural Networks, December 2022.

Benton, G. W., Maddox, W. J., Lotfi, S., and Wilson, A. G. Loss Surface Simplexes for Mode Connecting Volumes and Fast Ensembling. In *PMLR*, 2021.

Berardi, G., De Luigi, L., Salti, S., and Di Stefano, L. Learning the Space of Deep Models, June 2022.

Bishop, C. M. *Pattern Recognition and Machine Learning*. springer, 2006.

Brown, D., Vyas, N., and Bansal, Y. On Privileged and Convergent Bases in Neural Network Representations, July 2023.

Corneanu, C. A., Escalera, S., and Martinez, A. M. Computing the Testing Error Without a Testing Set. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2020.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, June 2022.

Dauphin, Y. N. and Schoenholz, S. MetaInit: Initializing learning by learning to initialize. In *Neural Information Processing Systems*, 2019.

De Luigi, L., Cardace, A., Spezialetti, R., Ramirez, P. Z., Salti, S., and Di Stefano, L. Deep Learning on Implicit Neural Representations of Shapes, February 2023.

Deutsch, L. Generating Neural Networks with Neural Networks. April 2018.

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. Essentially No Barriers in Neural Network Energy Landscape. In *International Conference on Machine Learning*, March 2018.

Eilertsen, G., Jönsson, D., Ropinski, T., Unger, J., and Ynnerman, A. Classifying the classifier: Dissecting the weight space of neural networks. February 2020.

Finn, C., Abbeel, P., and Levine, S. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR, July 2017.

Fort, S. and Jastrzebski, S. Large Scale Structure of Neural Network Loss Landscapes. June 2019.

Frankle, J., Dziugaite, G., Roy, D. M., and Carbin, M. Linear Mode Connectivity and the Lottery Ticket Hypothesis. December 2019.

Ha, D., Dai, A., and Le, Q. V. HyperNetworks, 2016.

Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. Predicting the Generalization Gap in Deep Networks with Margin Distributions. June 2019.

Knyazev, B., Drozdzal, M., Taylor, G. W., and Romero-Soriano, A. Parameter Prediction for Unseen Deep Architectures. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Knyazev, B., Hwang, D., and Lacoste-Julien, S. Can We Scale Transformers to Predict Parameters of Diverse ImageNet Models? In *arXiv.Org*, March 2023.

Kofinas, M., Knyazev, B., Zhang, Y., Chen, Y., Burghouts, G. J., Gavves, E., Snoek, C. G. M., and Zhang, D. W. Graph neural networks for learning equivariant representations of neural networks. In *International Conference on Learning Representations (ICLR)*, 2024.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. Similarity of Neural Network Representations Revisited. May 2019.

Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H., and Madry, A. FFCV: Accelerating Training by Removing Data Bottleneck. In *Computer Vision and Pattern Recognition (CVPR)*, 2023.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. Tune: A Research Platform for Distributed Model Selection and Training. July 2018.

Lucas, J. R., Bae, J., Zhang, M. R., Fort, S., Zemel, R., and Grosse, R. B. On Monotonic Linear Interpolation of Neural Network Parameters. In *International Conference on Machine Learning*. PMLR, July 2021.

Martin, C. H. and Mahoney, M. W. Rethinking generalization requires revisiting old ideas: Statistical mechanics approaches and complex learning behavior, February 2019a.

Martin, C. H. and Mahoney, M. W. Traditional and Heavy-Tailed Self Regularization in Neural Network Models. January 2019b.

Martin, C. H. and Mahoney, M. W. Heavy-tailed Universality predicts trends in test accuracies for very large pre-trained deep neural networks. In *Proceedings of the 20th SIAM International Conference on Data Mining*, 2020.

Martin, C. H. and Mahoney, M. W. Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *The Journal of Machine Learning Research*, 22(1), January 2021.

Martin, C. H., Peng, T. S., and Mahoney, M. W. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12(1), July 2021.

Meller, D. and Berkouk, N. Singular Value Representation: A New Graph Perspective On Neural Networks, February 2023.

Nava, E., Kobayashi, S., Yin, Y., Katzschmann, R. K., and Grewe, B. F. Meta-Learning via Classifier(-free) Diffusion Guidance. October 2022.

Nguyen, P., Tran, T., Gupta, S., Rana, S., and Dam, H.-C. HyperVAE: A Minimum Description Length Variational Hyper-Encoding Network. 2019.

Nguyen, Q. N. On Connected Sublevel Sets in Deep Learning. In *International Conference on Machine Learning*, January 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 2019.

Peebles, W., Radosavovic, I., Brooks, T., Efros, A. A., and Malik, J. Learning to Learn with Generative Models of Neural Network Checkpoints, September 2022.

Ratzlaff, N. and Fuxin, L. HyperGAN: A Generative Model for Diverse, Performant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019.

Schürholt, K., Kostadinov, D., and Borth, D. Self-Supervised Representation Learning on Neural Network Weights for Model Characteristic Prediction. In *Conference on Neural Information Processing Systems (NeurIPS)*, volume 35, 2021.

Schürholt, K., Knyazev, B., Giró-i-Nieto, X., and Borth, D. Hyper-Representations as Generative Models: Sampling Unseen Neural Network Weights. In *Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS)*, September 2022a.

Schürholt, K., Knyazev, B., Giró-i-Nieto, X., and Borth, D. Hyper-Representations for Pre-Training and Transfer Learning. In *First Workshop of Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022b.

Schürholt, K., Taskiran, D., Knyazev, B., Giró-i-Nieto, X., and Borth, D. Model Zoos: A Dataset of Diverse Populations of Neural Network Models. In *Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, September 2022c.

Sémery, O. Osmr/imgclsmob, January 2024.

Smith, L. N. and Topin, N. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates, May 2018.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., and Tolstikhin, I. Predicting Neural Network Accuracy from Weights. February 2020.

Vaswani, A., Ramachandran, P., Srinivas, A., Parmar, N., Hechtman, B., and Shlens, J. Scaling Local Self-Attention for Parameter Efficient Visual Backbones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Wang, J., Chen, Y., Yu, S. X., Cheung, B., and LeCun, Y. Compact and Optimal Deep Learning with Recurrent Parameter Generators. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, January 2023.

Wortsman, M., Horton, M. C., Guestrin, C., Farhadi, A., and Rastegari, M. Learning Neural Network Subspaces. In *International Conference on Machine Learning*. PMLR, July 2021.

Yak, S., Gonzalvo, J., and Mazzawi, H. Towards Task and Architecture-Independent Generalization Gap Predictors. June 2019.

Yang, Y., Theisen, R., Hodgkinson, L., Gonzalez, J. E., Ramchandran, K., Martin, C. H., and Mahoney, M. W. Evaluating natural language processing models with generalization metrics that do not need access to any training or testing data. Technical Report Preprint: arXiv:2202.02842, 2022.

Zhang, C., Ren, M., and Urtasun, R. Graph HyperNetworks for Neural Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2019.

Zhang, D. W., Kofinas, M., Zhang, Y., Chen, Y., Burghouts, G. J., and Snoek, C. G. M. Neural Networks Are Graphs!Graph Neural Networks for Equivariant Processing of Neural Networks. July 2023.

Zhmoginov, A., Sandler, M., and Vladymyrov, M. HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning. In *International Conference on Machine Learning (ICML)*, January 2022.

A. Ablation Studies

In this section, we perform ablation studies to assess the effectiveness of the methods proposed above: model alignment to simplify the learning task; inference window size to improve inference quality; haloing and batch-norm conditioning to increase sample quality.

Table 6. Impact of alignment ablation and permutation on reconstruction loss.

Sample Permutations			$\mathcal{L}_{rec}$	
Aligned	View 1	View 2	Train	Test
No	Perm.	Perm.	0.304	0.167
Yes	Perm.	Perm.	0.148	0.082
Yes	Align	Perm.	0.107	0.082
Yes	Align	Align	0.072	0.082

Impact of Model Alignment. Model alignment intuitively reduces training complexity by mapping all models to the same subspace. To evaluate its impact, we conduct training experiments with the same configuration on datasets with and without aligned models. In the dataset with aligned models, we use either the aligned form or 5 random permutations for the two views for both reconstruction and contrastive learning. As shown in Table 6, the results show two effects. First, alignment through git re-basin simplifies the learning task and contributes to improved generalization, both training and test losses are reduced by more than 50%. Second, anchoring at least one of the views to the aligned form does further reduce the training loss, but does not improve generalization.

Window Size Ablation. The sequential decomposition of SANE allows one to pretrain not on the full model sequence, but on subsequences. The choice of the length of the subsequence, the window size, is a critical parameter that balances computational load and context. We used a window of 256 for pretraining for most of our experiments.

Here, we study the influence of the window size on reconstruction error, exploring values ranging from 32 to 2048. Our experiments did not reveal substantial impact of smaller windows on pretraining loss or sampling performance. This seems to suggest that a window size as large as 2048 may still be insufficient on ResNets to capture enough context. Alternatively, it may suggest that the underlying assumption that context matters may not entirely hold up.

However, we did observe an important impact on the relationship between training and inference window sizes. During inference, memory load is significantly lower. Inference allows much larger window sizes, up to the entire length of the ResNet sequence. However, departing from the training window size appears to introduce interference, which affects the reconstruction error (Figure 6).

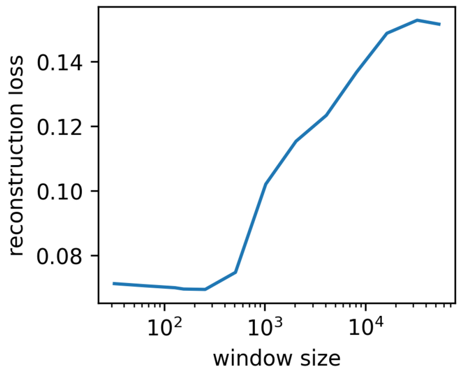

Figure 6. SANE reconstruction loss over number of tokens within a window. The loss is lowest around the training windowsize of 256 tokens, longer sequences up to the full model sequence length of 50k tokens cause interference and double the reconstruction error.

Halo and batch-norm conditioning. Haloing and batch-norm conditioning aim at reducing noise in model sampling; see Section 2. To assess their impact on sampling performance, we conduct an in-domain experiment using SANE trained on CIFAR-10 ResNet-18s, using prompt examples from the train set and fine-tuning on CIFAR-10. We compare with *naïve* sampling without haloing and batch-norm conditioning. The results in Table 7 show the significant improvements achieved by both haloing and batch-norm conditioning. From random guessing of naïve sampling, combining both improves to around 65%. Since both methods aim at reducing noise for zero-shot sampling, their effect is largest then and diminishes somewhat during finetuning. Both methods not only improve zero-shot sampling per se but make the sampled models provide enough signal to facilitate subsampling or bootstrapping strategies.

Table 7. Ablation of batch-norm conditioning and haloing.

Ep.	Method	CIFAR-10
0	rand init	∼10 /%
	naïve	10±0.0
	Haloed	14.5±6.3
	BN-cond	60.8±2.2
	Haloed+BN-cond	**64.8±2.1**
5	rand init	64.4±2.9
	naïve	90.8±0.2
	Haloed	90.9±0.1
	BN-cond	90.7±0.2
	Haloed+BN-cond	90.9±0.2

B. SANE Architecture Details

In Table 8, we provide additional information on the training hyper-parameters for SANE on populations of small CNNs as well as ResNet18s. These values are the stable mean across all experiments, exact values can vary from population to population. Full experiment configurations are documented in the code.

Table 8. Architecture Details for SANE

Hyper-Parameter	CNNs	ResNet-18
tokensize	289	288
sequence lenght	∼50	∼50k
window size	32	256, 512
d_model	1024	2048
latent_dim	128	128
transformer layers	4	8
transformer heads	4,8	4,8

C. SANE Embedding Analysis - Additional Results

This section contains additional results on SANE embedding analysis, in comparison with previous weight matrix analysis. In Figure 7, we compare the eigenvalue distribution for different models with SANE embeddings. Replicating the experiment setup from (Martin & Mahoney, 2019b; 2021), we train MiniAlexNet models on CIFAR-10 varying only the batch size. With a smaller batch size and longer training duration, the eigenvalue distribution transitions from random with very few spikes, over a bulk with many spikes, to heavy-tailed. The embeddings of SANE appear to also become more heavy-tailed, but do not seem to pick up on the change from few to many spikes. The results are suggestive, pointing to obvious follow-up work.

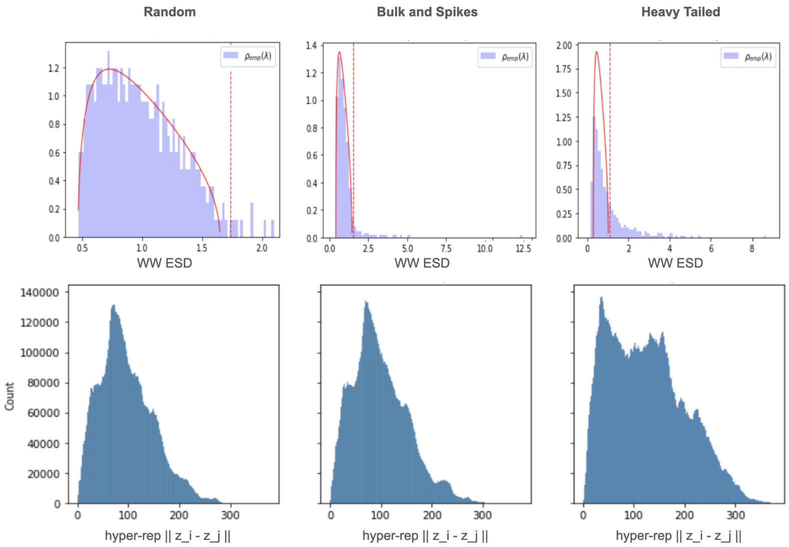

Figure 7. Comparison between WeightWatcher features (top) and SANE (bottom). Martin & Mahoney (2019b) identify different phases in the eigenvalue spectrum of trained weight matrices. We replicate the experiment setup and find ESDs similar to *random* (top left), *bulk and spikes* (top middle) and *heavy-tailed* (top right). We compare these against pairwise distances of SANE embeddings of the same layer. While the distributions have a different shape, it appears to become more heavy-tailed going from *random* to *heavy tailed*.

Figures 8 and 9 compare SANE with different WeightWatcher metrics on VGGs from pytorchcv (Sémery, 2024) and the ResNet-18 zoo from the modelzoo dataset (Schürholt et al., 2022c).

D. Model Property Prediction - Additional Results

In this section, we provide additional details for Section 5.1. Table 9 shows full results for populations of small CNNs.

Table 9. Property prediction on populations of small CNNs.

	MNIST			SVHN			CIFAR-10 (CNN)			STL		
	W	$s(W)$	SANE	W	$s(W)$	SANE	W	$s(W)$	SANE	W	$s(W)$	SANE
ACC	0.965	**0.987**	0.978	0.910	0.985	**0.991**	-7.580	**0.965**	0.885	-18.818	**0.919**	0.305
Epoch	0.953	**0.974**	0.958	0.833	**0.953**	0.930	0.636	**0.923**	0.771	-1.926	**0.977**	0.344
Ggap	0.246	0.393	**0.402**	0.479	0.711	**0.760**	0.324	**0.909**	0.772	-0.617	**0.858**	0.307

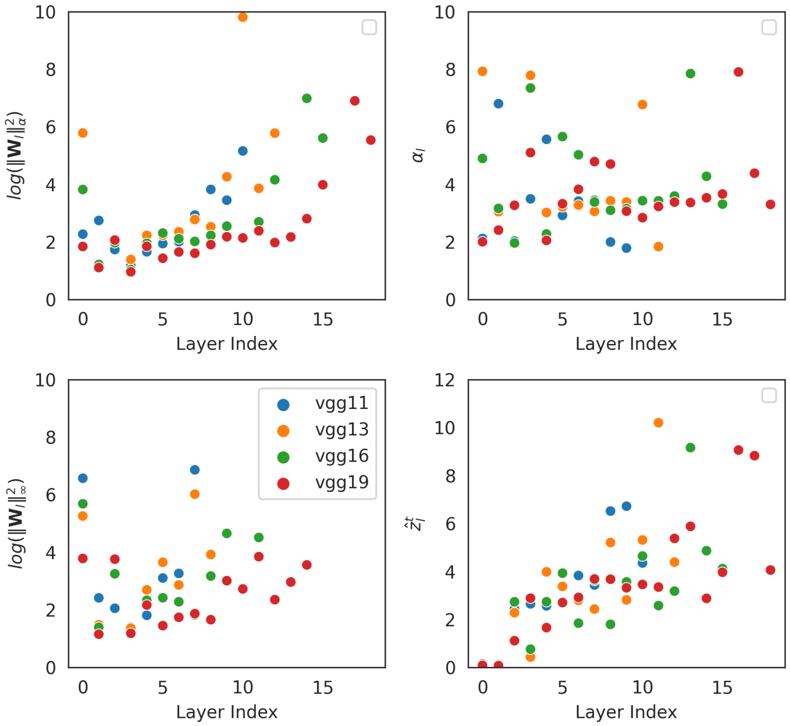

Figure 8. Comparison between different WeightWatcher (WW) features (left) and SANE (right). Features over layer index for VGGs from pytorchcv of different sizes. SANE shows similar trends to WW, low values at early layers and a sharp increase at the end.

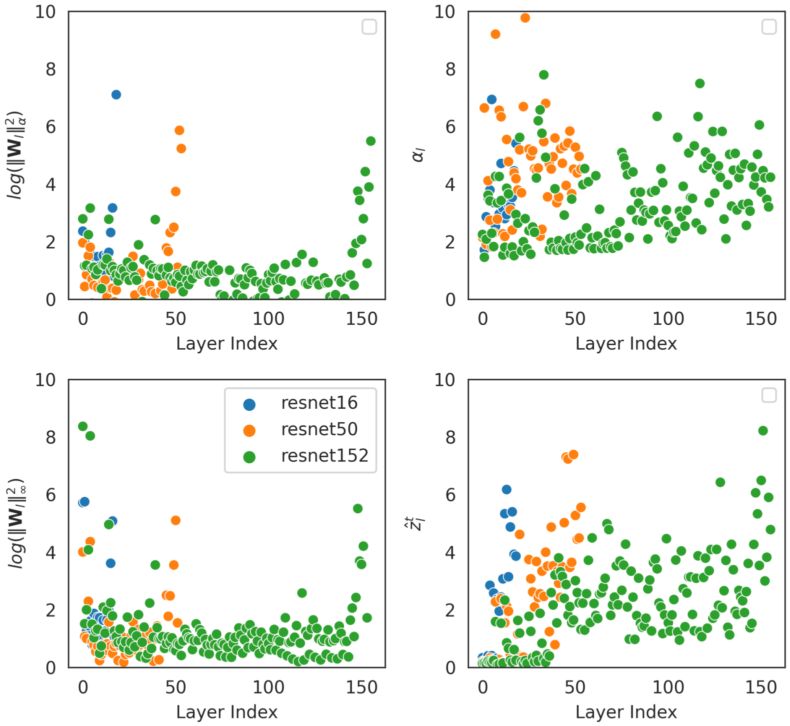

Figure 9. Comparison between different WeightWatcher (WW) features (left) and SANE (right). Features over layer index for Resnets from pytorchcv of different sizes. SANE shows similar trends to WW, low values at early layers and a sharp increase at the end.

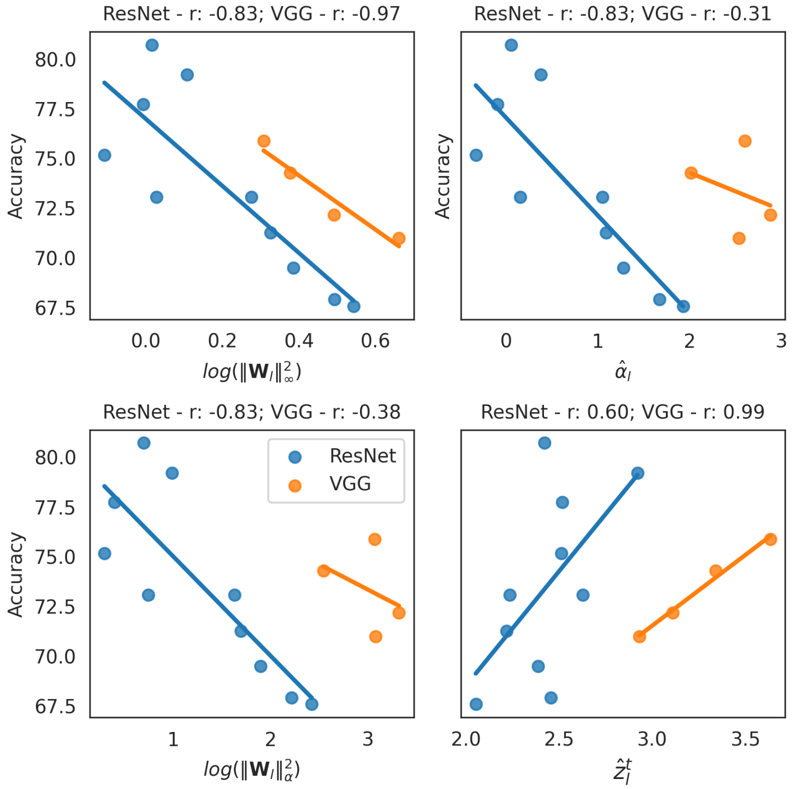

Figure 10. Comparison between WeightWatcher features (left) and SANE (right). Accuracy over model features for Resnets and VGGs from pytorchcv of different sizes. SANE shows similar trends to WW, low values at early layers and a sharp increase at the end.

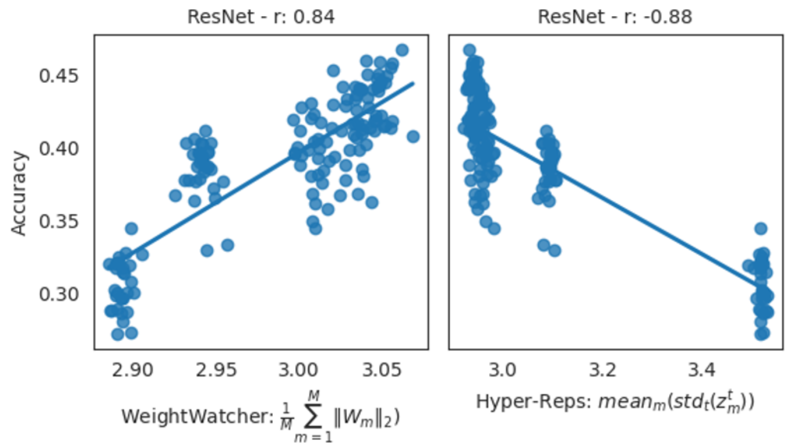

Figure 11. Comparison between WeightWatcher features (left) and SANE (right). Accuracy over model features for ResNets from the ResNet model zoo. Although SANE is pretrained in a self-supervised fashion, it preserves the linear relation of a globally-aggregated embedding to model accuracy.

D.1. Comparison to Previous Work

Here, we compare SANE with previous work to disseminate the information contained in model embeddings. The experiment setup in this paper is designed around the ResNets, and therefore it uses sparse epochs for computational efficiency. For consistency, we use the same setup for the CNN zoos as well. The exact numbers are therefore not directly comparable to Schürholt et al. (2021). To provide as much context as possible, we approach the comparison from two angles:

(1) **Direct comparison to the published results:** to contextualize, we use the (deterministic) results of weight statistics $s(W)$ to adjust for the differences in setup. We mark the results for $s(W)$ from Schürholt et al. (2021) as $s(W)_{pp}$ and compare to their $E_{c+}D$ where possible.

(2) **Approximation of the effect of global embeddings:** previous work used global model embeddings, which we approximate by using the full model embedding sequence. We therefore compare SANE + aggregated tokens (as proposed in the submission) to SANE + full model sequence (similar to Schürholt et al. (2021)).

The results in the Tables below allow the following conclusions:

(1) **SANE matches the performance of previous work:** The only data available for direct comparison is the MNIST zoo. Here, both in direct comparison and in relation to s(W) cross-relating our results with published numbers, SANE matches published performance of $E_c + D$. On other zoos, $E_c + D$ had similar performance to $s(W)$. We likewise find SANE embeddings to have similar performance to $s(W)$ in our experiments.

(2) **SANE + full sequence improves downstream task performance over the SANE + aggregated sequence:** That indicates that SANE + full sequence contains more information for model prediction. However, both Schürholt et al. (2021) and SANE with full sequence have the disadvantage that they do not scale. With growing models, the representation learner of Schürholt et al. (2021) and the input to the linear probe of SANE + full sequence grow accordingly. SANE + aggregated sequence does lose some information on small models, but scales gracefully to large models and remains competitive.

Table 10. Property Prediction comparison to previous work on the MNIST-CNN model zoo. We compare our linear probing results from weights W, layer-wise quintiles $s(W)$, embeddings from SANE either aggregated into one embedding or using the full sequence to results previously published in Schürholt et al. (2021). We mark their results for $s(W)$ as $s(W)_{pp}$. Since the experimental setup is not the same, the numbers of $s(W)$ do not match.

	W	$s(W)$	SANE aggregated	SANE full sequence	$s(W)_{pp}$	$E_{c+}D$
ACC	0.965	**0.987**	0.978	**0.987**	0.977	0.973
Epoch	0.953	0.974	0.958	**0.975**	0.987	0.989
Ggap	0.246	0.393	0.402	**0.461**	0.662	0.667

Table 11. Property Prediction comparison to previous work on the SVHN-CNN model zoo. We compare our linear probing results from weights W, layer-wise quintiles $s(W)$, to embeddings from SANE either aggregated into one embedding or using the full sequence. For this zoo, previous results are not available.

	W	$s(W)$	SANE aggregated	SANE full sequence	$s(W)_{pp}$	$E_{c+}D$
ACC	0.910	**0.985**	0.991	**0.993**	*n/a*	*n/a*
Epoch	0.833	**0.953**	0.930	**0.943**	*n/a*	*n/a*
Ggap	0.479	0.711	0.760	**0.77**	*n/a*	*n/a*

Table 12. Property Prediction comparison to previous work on the CIFAR-CNN(m) model zoo. We compare our linear probing results from weights W, layer-wise quintiles $s(W)$, to embeddings from SANE either aggregated into one embedding or using the full sequence. For this zoo, previous results are not available.

	W	$s(W)$	SANE aggregated	SANE full sequence	$s(W)_{pp}$	$E_{c+}D$
ACC	-7.580	**0.965**	0.885	**0.947**	*n/a*	*n/a*
Epoch	0.636	**0.923**	0.771	**0.879**	*n/a*	*n/a*
Ggap	0.324	**0.909**	0.772	**0.811**	*n/a*	*n/a*

E. Model Generation - Additional Results

This section contains additional results from model sampling experiments, extending Section 5.2. In Table 13, we show results on small CNNs transferring to a new task. Similarly, Table 14 shows results on ResNet-18 models for task transfers.

Lastly, Tables 15, 16 and 17 contain additional results for transferring from ResNet-18 CIFAR-100 to ResNet34 and/or Tiny-Imagenet.

Table 13. Model generation on CNN model populations transfer learned on a new task. We compare sampled models at different epochs with models trained from scratch and models fine-tuned from the anchor samples.

Method	SVHN to MNIST			CIFAR-10 to STL-10		
	Epoch 0	Epoch 1	Epoch 25	Epoch 0	Epoch 1	Epoch 25
tr.fr.scratch	∼10 /%	20.6+-1.6	83.3+-2.6	∼10 /%	21.3+-1.6	44.0+-1.0
pretrained	29.1+-7.2	84.1+-2.6	94.2+-0.7	16.2+-2.3	24.8+-0.8	49.0+-0.9
S_{KDE30}	31.8+-5.6	86.9+-1.4	95.5+-0.4	n/a	n/a	n/a
SANE $_{KDE30}$	**40.2+-4.8**	86.7+-1.6	94.8+-0.4	15.5+-2.3	24.9+-1.6	49.2+-0.5
SANE $_{SUB}$.	37.9+-2.8	**88.2+-0.5**	**95.6+-0.3**	**17.4+-1.4**	**25.6+-1.7**	**49.8+-0.6**

E.1. Diversity of sampled models

An interesting question is whether sampling SANE generates versions of the same model. To test that, we evaluate the diversity of samples generated with only a few few-shot examples by combining the models to ensembles. The improvements of the ensembles over the individual models demonstrate their diversity. This indicates that given very few, early-stage prompt examples, sampling hyper-representations improves learning speed and performance in otherwise equal settings. Additionally, we conducted experiments with varying numbers of prompt examples, revealing that increasing the number of prompt examples enhances both performance and diversity. Nonetheless, even a single prompt example trained for just 2 epochs contains sufficient information to generate model samples that surpass those derived from random initialization; see Table 17.

Table 14. Model generation on ResNet-18 model populations transferred to a new task. We compare sampled models at different transfer learning epochs with models trained from scratch and models fine-tuned from the same anchor samples.

Epoch	Method	CIFAR-10 to CIFAR-100	CIFAR-100 to Tiny-Imagenet	Tiny-Imagenet to CIFAR-100
0	tr. fr. scratch	∼1 /%	∼0.5 /%	∼1 /%
	Finetuned	1.0+-0.3	0.5+-0.0	1.1+-0.2
	SANE $_{KDE30}$	1.0+-0.3	0.5+-0.1	1.0+-0.2
	SANE $_{SUB}$	1.0+-0.3	0.6+-0.0	1.1+-0.2
	SANE $_{BOOT}$	1.1+-0.2	0.5+-0.0	0.9+-0.2
1	tr. fr. scratch	17.5+-0.7	13.8+-0.8	17.5+-0.7
	Finetuned	27.5+-1.3	25.7+-0.5	51.7+-0.5
	SANE $_{KDE30}$	26.8+-1.4	21.5+-0.9	40.2+-1.0
	SANE $_{SUB}$	26.4+-1.9	21.5+-1.0	40.63+-1.3
	SANE $_{BOOT}$	25.7_01.9	21.7+-1.0	40.9+-0.8
5	tr. fr. scratch	36.5+-2.0	31.1+-1.6	36.5+-2.0
	Finetuned	45.7+-1.0	36.3+-2.5	52.6+-1.3
	SANE $_{KDE30}$	44.5+-2.0	36.3+-1.2	47.2+-3.3
	SANE $_{SUB}$	45.6+-1.2	35.8+-1.4	49.8+-2.3
	SANE $_{BOOT}$	43.3+-2.4	**37.3+2.0**	50.2+-3.4
15	tr. fr. scratch	53.3+-2.0	38.5+-1.9	53.3+-2.0
	Finetuned	71.9+-0.1	63.4+-0.2	**73.9+-0.3**
	SANE $_{KDE30}$	71.8+-0.3	**63.6+-0.2**	73.4+-0.2
	SANE $_{SUB}$	72.0+-0.2	**63.6+-0.3**	73.5+-0.2
	SANE $_{BOOT}$	71.9+-0.3	63.4+-0.1	73.7+-0.3
25	tr. fr. scratch	56.5+-2.0	43.3+-1.9	56.5+-2.0
50	tr. fr. scratch	70.7+-0.4	57.3+-0.6	70.7+-0.4
60	tr. fr. scratch	74.2+-0.3	63.9+-0.5	74.2+-0.3

Table 15. Few-shot model generation for a new task: Sampling ResNet-34 models for CIFAR-100. SANE was pretrained on CIFAR-100 ResNet-18s, 5 samples are drawn using subsampling. To get prompt examples, we train 3 ResNet-34 models on CIFAR-100 for 2 epochs to a mean accuracy of 26 %.

CIFAR100 ResNet-18 to ResNet-34			
Ep.	Method	5 Epochs	15 Epochs
0	tr. fr. Scratch	1.0±0.1	1.0±0.1
	SANE	**1.6±0.3**	**1.6±0.3**
1	tr. fr. Scratch	12.4±1.0	12.9±0.8
	SANE	**16.8±0.7**	**23.1±0.3**
5	tr. fr. Scratch	49.5±0.6	36.2±1.7
	SANE	**51.9±0.6**	**37.8±1.4**
15	tr. fr. scratch		68.8±0.4
	SANE		**69.3±0.3**
	SANE Ens.	53.5	71.3

Table 16. Few-shot model generation for a new task and architecture: SANE trained on CIFAR-100 ResNet-18s used to generate ResNet-34s for Tiny-Imagenet. 5 samples are drawn using subsampling. To get prompt examples, we train 3 ResNet-34 models on Tiny-Imagenet for 2 epochs to a mean accuracy of 28.5 %.

ResNet-18 CIFAR100 to ResNet-34 Tiny-Imagenet			
Ep.	Method	5 epochs	15 epochs
0	tr. fr. Scratch	0.5±0.0	0.5±0.0
	SANE	0.5±0.1	0.6±0.2
1	tr. fr. Scratch	10.5±1.4	11.9±1.9
	SANE	**13.3±0.5**	**18.5±0.7**
5	tr. fr. Scratch	47.2±0.7	31.1±1.7
	SANE	**50.6±0.3**	**31.6±0.6**
15	tr. fr. Scratch		61.9±0.3
	SANE		**62.7±0.3**
	SANE Ens.	52	65.1

Table 17. Sampling ResNet-34 models for CIFAR-100. SANE was pretrained on CIFAR-100 ResNet-18s, 5 samples are drawn using subsampling. To get prompt examples, we train a single ResNet-34 model on CIFAR-100 for 2 epochs to an accuracy of 26 %.

CIFAR100 ResNet-18 to ResNet-34			
Epoch	Method	5 Epochs	15 Epochs
0	tr. fr. Scratch	1.0+-0.1	1.0+-0.1
	SANE	**1.5+-0.2**	**1.6+-0.1**
1	tr. fr. Scratch	12.4+-1.0	12.9+-0.8
	SANE	**16.9+-0.7**	**19.4+-0.2**
5	tr. fr. Scratch	49.5+-0.6	36.2+-1.7
	SANE	**51.5+-0.3**	**38.6+-1.6**
15	tr. fr. scratch		68.8+-0.4
	SANE		**69.1+-0.1**
Ensemble	SANE	51.8	70.2