Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

A Contrastive Approach to Weight Space Learning

by

D.J. Swanevelder

MSc Machine Learning/ Artificial Intelligence

Department of Applied Mathematics

Faculty of Science

Stellenbosch University

Supervisor: Ruan van der Merwe

November 2025

Abstract

The fundamental goal is to have system which is capable of embedding a dataset $\mathcal{D}$, the trained neural weights W, and the corresponding results R (training and test) into a shared embedding latent space. The goal is to use the shared space, to sample from the space conditioned on a new unseen dataset and a specified results, and decode into real weights. The architecture is kept constant for simplicity.

$$p(W \mid R, \mathcal{D}, \text{Arch})$$

The way in which this will be implemented requires a few building blocks. - Synthetic dataset generator - Training pipeline and weight storage - Weight Embedding - Dataset Embedding - Shared embedding projection

The goal of the project is to get the simplest implementation of the following up and running as fast as possible, and the evaluate the performance, then iteratively update the part of the pipeline to achieve better results. The goal is a graph, with input results (desired results) on the x-axis, measured results on the y. If the graph is somewhat linear, project is a success.

Contents

Chapter 1

Introduction

> ”We don’t tell [computers] what to do, we give them examples... The problem is, sometimes we don’t understand how it figured it out.”
>
> – Jeff Dean, Head of Google AI [1]

The prevalence of neural networks (NNs) has established them as a foundational technology in modern artificial intelligence. However, as their use has expanded, so too has attention to their inherent mechanistic limitations. Two major drawbacks of NNs are their lack of explainability—the “black-box” effect—and the substantial computational cost of training. With platforms like Hugging Face and GitHub hosting over one million pre-trained models [2], interest in addressing these limitations has intensified. Motivated by this vast availability of models, a novel research direction—**weight space learning**—has emerged as a promising approach to better understand these limitations.

Formally, the weight space of a neural network refers to the set of all possible configurations of its parameters $W \in \mathbb{R}^n$, where n is the total number of trainable weights. Each point in this space represents a unique model with a distinct mapping from inputs to outputs. Weight space learning thus concerns learning representations or distributions over this space, capturing how variations in W relate to model behaviour and how variations in model training influence W. The field generally considers two types of tasks: discriminative and generative.

In discriminative applications, models use the weights of pre-trained networks—often collected into a model zoo [3]—as input to predict meta-information about the original models [4]. The quality of a weight space representation is typically evaluated by the performance of a simple multi-layer perceptron (MLP) in predicting such meta-information, conditioned only on the model’s weight embedding.

Common meta-information metrics include the model’s final performance and its generalisation gap (the difference between training and validation loss). [5] demonstrated that weight embeddings can encode key training characteristics, such as recovering the size of the dataset used for training. These discriminative tasks serve both to validate the quality of the derived weight representations and to provide practical predictive value.

In generative applications, researchers aim to model the underlying distribution of neural network weights W, conditioned on additional information or reference models, $P(W \mid \dots)$. Sampling from this distribution enables the generation of entirely new model weights.

[6] used an autoencoder with a bottleneck layer to generate hyper-representations of multiple model zoos. Building on this idea, [7] introduced the Sequential Autoencoder for Neural Embeddings (SANE), which improved scalability and enabled work on much larger models.

To model $P(W \mid \mathcal{D})$, [8] employed a Vector Quantised Variational Autoencoder (VQ-VAE), which incorporates dataset information when learning latent weight representations. Similarly, [9] modelled $P(W \mid R)$ by incorporating behavioural differences between reconstructed and original models into the embedding learning process.

While existing weight space learning methods have made significant progress, they typically address isolated aspects of the learning process. Current approaches model either $P(W \mid \mathcal{D})$ or $P(W \mid R)$, but rarely both simultaneously. This is a key limitation: in practice, a model's weights are influenced by both the data it was trained on and the results it achieved. Understanding the joint relationship $P(W \mid \mathcal{D}, R)$ is essential for explaining model behaviour and for generating models with desired characteristics.

Contrastive learning has emerged as a powerful paradigm for learning unified representations across modalities, as demonstrated by models such as CLIP [10], which bridge vision and language. A contrastive objective pulls related samples closer in representation space while pushing unrelated samples apart, forming a meaningful joint embedding space for heterogeneous data types. This property makes contrastive learning particularly suitable for modelling the complex distribution $P(W \mid \mathcal{D}, R)$ — encompassing visual datasets, high-dimensional weight tensors, and performance metrics — without requiring a shared native representation.

In this report, we develop a contrastive learning framework to create a unified embedding space that jointly represents neural network weights W, the datasets they were trained on $\mathcal{D}$, and their resulting performance characteristics R. Specifically, we construct two separate encoders—one for dataset embeddings using pre-trained CLIP features, and another for weight embeddings using an autoencoder architecture—alongside a binned result embedding table. These encoders are trained using the contrastive objective NT-Xent [11], which encourages related triplets $(\mathcal{D}, W, R)$ to be close in the shared latent space. Figure 1.1 depicts a high-level view of the full embedding pipeline.

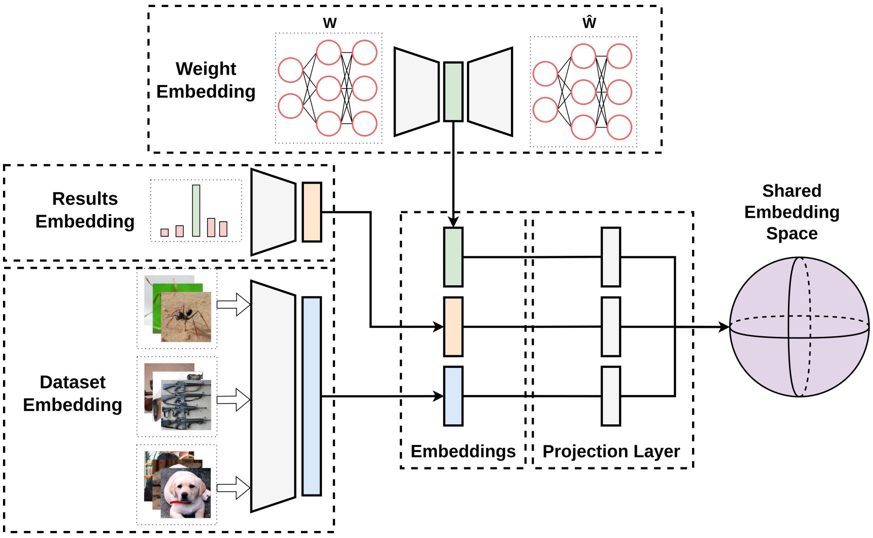


Figure 1.1: A Conditonal Model Sampling system, embedding a dataset, model weights and results into a shared embedding space

Our central **hypothesis** is that this unified representation space will enable two key capabilities:

- **Interpretability and Analysis:** Examining geometric relationships within the shared embedding space can reveal how dataset characteristics shape learned weights and model behaviour.
- **Conditional Model Sampling:** The learned distribution enables sampling of model weights conditioned on both dataset properties and target performance metrics–approximating $P(W \mid \mathcal{D}, R)$.

Primary Goal: The primary goal of this report is to demonstrate the viability of our methodology through the success of **Conditional Model Sampling.** While our hypothesis includes both Interpretability and Conditional Model Sampling, success in the latter strongly suggests a meaningful foundation for the former. Specifically, the ability to accurately sample weights that achieve target performance metrics conditioned on specific dataset ($\mathcal{D}$) and result (R) targets confirms that the unified embedding space has meaningfully captured the joint influence of $\mathcal{D}$ and R on the model weights W. We will therefore focus on generating meaningful representations, measuring their quality, and critically, performing various forms of conditional model sampling to determine the viability of this specific methodology for generative weight space representations.

Chapter 2

Background

2.1. Weight Space Learning

Weight space learning is a field within machine learning that focuses on understanding and leveraging the structure of the neural network weight space. The central aim is to model how network parameters are shaped by data, architecture, and training dynamics, and to capture these relationships within a learnable representation.

The weight space ($\mathcal{W}$) of a neural network is formally defined as the high-dimensional Euclidean space $\mathbb{R}^N$ spanned by the N total learnable parameters—the weights and biases—of the specified architecture. A specific point $\mathbf{w} \in \mathcal{W}$ represents a complete instantiation of the model, mapping input data $\mathbf{x}$ to output predictions $f(\mathbf{x}, \mathbf{w})$. Consequently, weight space learning concentrates on characterizing the intrinsic structure and topology of this landscape, aiming to identify low-dimensional subspaces or manifolds containing high-performing parameters, thus allowing the development of compact, learnable representations of network ensembles, parameter transfer methods, and novel subspace optimization techniques.

Due to the enormous scale and dimensionality of modern neural networks [], it is typically infeasible to operate directly on raw model weights. This challenge motivates one of the central subproblems of weight space learning: the discovery of low-dimensional representations of weight space.

Ideally from latent representations we should see smaller networks are performing the same, but not vice versa A *latent representation* of weight space provides a compact and structured encoding of a model's parameters. The transformations—linear or non-linear—that map weights into this latent space are learned to preserve the essential information required to reconstruct or analyse the original weights. Among the many dimensionality reduction techniques available, we distinguish between *reversible* and *non-reversible* methods.

Reversibility is of particular importance in weight space learning. While encoding weights into a latent representation (real → latent) is informative, the ability to reconstruct the original weights (real → latent → real) is far more valuable. This reversibility enables the synthesis of entirely new weight configurations, supporting generative applications such as zero-shot model creation and performance-guided model generation. Consequently, reversible latent representation methods are the most prevalent within weight space learning, especially for encoding and decoding neural network weights.

2.2. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a linear dimensionality reduction technique used to represent high-dimensional data in a lower-dimensional form while retaining as much variance as possible. It provides a compact latent representation that captures the most informative directions of variation in the data.

Consider a dataset $X \in \mathbb{R}^{n \times d}$ with n samples and d features, centred such that each feature has zero mean. The goal of PCA is to find a new coordinate system whose axes are linear transformations of the original dimensions, ordered by the amount of variance they capture. Orthogonality between these axes ensures that each captures unique, non-redundant information about the data.

Intuitively, PCA identifies the directions that best describe the “shape” or spread of the data cloud in feature space. Projecting the data onto the top k directions provides a compressed representation that preserves most of the information while discarding redundancy.

PCA seeks a linear projection matrix $W \in \mathbb{R}^{d \times k}$ that maps the data to a lower-dimensional space:

$$Z = XW \tag{2.1}$$

where $Z \in \mathbb{R}^{n \times k}$ represents the latent representation. Assuming X is already centered the sample covariance matrix simplifies to

$$\Sigma = \frac{1}{n-1} X^{\top} X,$$

and the total variance captured by the projection is

$$\text{Var}(Z) = \text{Tr}(W^{\top} \Sigma W),$$

where the trace operator $\text{Tr}(\cdot)$ sums the variances along all projected directions.

PCA thus maximises the variance of the projected data:

$$\max_{W} \text{Tr}(W^{\top} \Sigma W) \quad \text{subject to} \quad W^{\top} W = I_k.$$

The constraint enforces orthonormality among the new axes so that each captures distinct variance. Solving this optimisation leads to the eigenvalue problem:

$$\Sigma W = W\Lambda,$$

where the columns of W are the eigenvectors of Σ, and the diagonal entries of Λ are the corresponding eigenvalues that quantify the variance explained by each principal component. The top k eigenvectors define the optimal projection directions. The resulting embedding, Z is the *latent representation* of the data, capturing the dominant linear structure of the dataset.

The original data can be approximately reconstructed from the latent space using the transpose of the transformation matrix:

$$\hat{X} = ZW_k^\top \tag{2.2}$$

where $\hat{X} \in R^{n\times d}$ is the low-rank approximation of the centered data X.

PCA provides a linear method for generating lossy reversibility latent representations, however, it's linear nature limits its ability to capture non-linear relationships, thereby restricting its overall expressiveness.

2.3. Autoencoders

Autoencoders are neural network architectures designed for unsupervised representation learning. Their primary goal is to learn a compressed, informative latent representation of input data by training the entire system to reconstruct the original input after it has passed through a lower-dimensional bottleneck layer. The full architecture is characterized by an encoder and a decoder. Crucially, the layers within these components typically utilize non-linear activation functions, which enables the model to perform complex, non-linear projections from the high-dimensional input space to the latent space. This capability is essential for capturing intricate, hierarchical structures within the data that linear dimensionality reduction techniques cannot access.

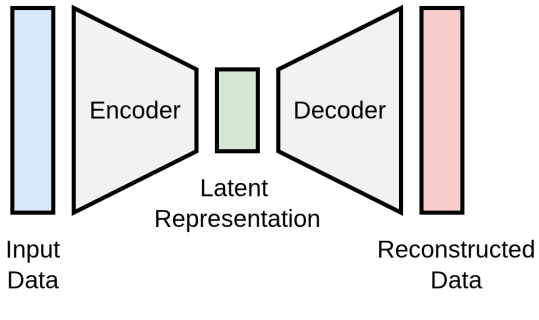


Figure 2.1: Autoencoder encoding input data into a latent representation and decoding to reconstructed data.

An autoencoder consists of two primary components: an *encoder* and a *decoder*. The encoder, parameterised by weights θ_e, maps an input $x \in \mathbb{R}^d$ to a latent representation $z \in \mathbb{R}^k$ through a sequence of nonlinear transformations:

$$z = f_{\text{enc}}(x; \theta_e). \tag{2.3}$$

The decoder, parameterised by θ_d, reconstructs the input from this latent representation:

$$\hat{x} = f_{\text{dec}}(z; \theta_d). \tag{2.4}$$

Together, the encoder and decoder form a composite function:

$$\hat{x} = f_{\text{dec}}(f_{\text{enc}}(x)). \tag{2.5}$$

The bottleneck layer (latent space) enforces a compression constraint, ensuring the model retains only the most salient features necessary for accurate reconstruction.

The network is trained to minimise the reconstruction error between the input x and its reconstruction $\hat{x}$. The most common loss function is the Mean Squared Error (MSE):

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \hat{x}_i\|^2. \tag{2.6}$$

By minimising this objective, the autoencoder learns weight configurations that best compress and reconstruct the data distribution.

The latent space z provides a nonlinear embedding that captures underlying structure within the data. Once trained, the encoder can be used independently for dimensionality reduction or feature extraction, while the decoder can serve as a generative mapping from the latent space back to the input domain.

2.4. Contrastive Learning

The overall goal of contrastive learning is to learn meaningful representations by comparing data points against one another, rather than relying on explicit labels []. The foundational assumption is that if a model is provided with sufficient examples of similar and dissimilar pairs, it can learn to represent data such that semantically similar samples lie close together in the embedding space, while dissimilar samples are placed farther apart. This framework enables unsupervised or self-supervised learning of latent representations that capture semantic structure purely through relative similarity.

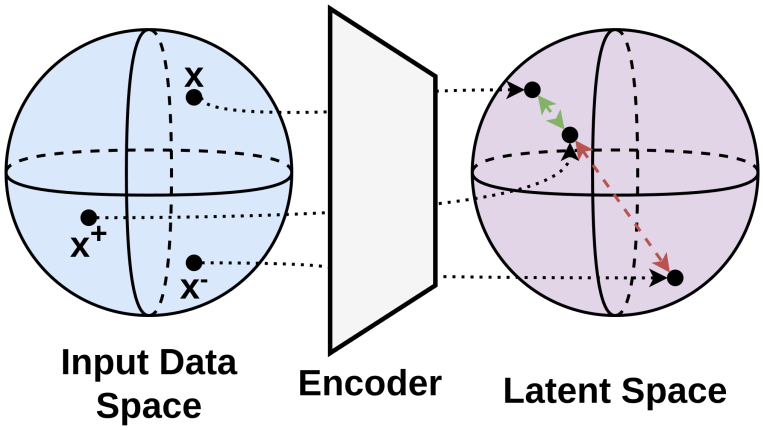


Figure 2.2: Contrastive encoder pulling similiar data pairs (x, x^+) together in latent space and pushing dissimilar pairs (x, x^-) apart.

2.4.1. NT-Xent Loss

A widely used objective for contrastive learning is the Normalised Temperature-Scaled Cross Entropy (NT-Xent) loss introduced in []. This loss formulates the contrastive task as a classification problem over positive and negative pairs within a batch. Given a batch of N samples, each with an augmented pair (x_i, x_i'), the loss for a positive pair (i, j) is defined as:

$$\mathcal{L}_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)} \tag{2.7}$$

where $\text{sim}(z_i, z_j)$ denotes the cosine similarity between the latent representations z_i and z_j, and τ is a temperature parameter that controls the sharpness of the distribution. The total batch loss is obtained by averaging over all positive pairs. This formulation encourages positive pairs (augmentations of the same sample) to have high similarity, while simultaneously pushing apart embeddings from different samples, shown visually in Figure 2.2, resulting in well-structured and discriminative latent representations.

2.4.2. Multimodal Embedding

A key advancement enabled by contrastive learning is the creation of multimodal embedding spaces, which align representations from distinct data types, such as images, text, and neural network weights, into a single, shared latent space. The importance of this lies in its ability to facilitate cross-modal understanding and retrieval. By representing different data types with vectors in the same space, models can compute the semantic similarity between, for example, an image and a piece of text directly.

An important application of contrastive learning using the NT-Xent objective is CLIP (Contrastive Language–Image Pretraining) []. CLIP jointly trains an image encoder and a text encoder to align visual and textual representations in a shared embedding space. During training, each image is paired with a corresponding caption, forming a positive pair, while all other image–text combinations in the batch act as negatives. The model optimises a symmetric contrastive objective, where each image predicts its matching caption and vice versa:

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2}(\mathcal{L}_{\text{image-to-text}} + \mathcal{L}_{\text{text-to-image}}).$$

The loss components, $\mathcal{L}_{\text{image-to-text}}$ and $\mathcal{L}_{\text{text-to-image}}$, are calculated using a contrastive objective. During training, a batch of N image-caption pairs is sampled. The paired image and text constitute the single positive pair against which the model optimizes. All other $N-1$ image-text combinations within that batch are treated as negative pairs. The overall objective $\mathcal{L}_{\text{CLIP}}$ is symmetric:

- $\mathcal{L}_{\text{image-to-text}}$ treats the images as anchor points, where the loss is calculated by having each image predict its correct matching caption from the N available captions in the batch.
- $\mathcal{L}_{\text{text-to-image}}$ treats the captions as anchor points, where the loss is calculated by having each caption predict its correct matching image from the N available images in the batch.

The total loss is the average of these two directional cross-entropy terms, ensuring that the embeddings are aligned symmetrically. Through this process, CLIP learns general-purpose visual and linguistic representations that are semantically aligned. Once trained, the model can perform zero-shot classification and other cross-modal tasks by measuring similarity between image and text embeddings without task-specific fine-tuning.

2.5. Autoencoder for Neural Embeddings

Applying the concept of an autoencoder to neural network weights has recently gained attention as a means to learn structured, low-dimensional representations of model parameters. Instead of encoding raw input data, the autoencoder learns to encode and reconstruct the weights of pre-trained neural networks, effectively embedding each model into a latent space that captures structural and functional similarities between models. This approach was explored in [12], where the goal was to build a continuous and interpretable representation space of neural weights.

Unlike conventional autoencoders that optimise a single reconstruction loss, the neural weight autoencoder is trained with a multi-objective loss function that combines reconstruction and contrastive terms:

$$\mathcal{L} = \beta\mathcal{L}_{\text{MSE}} + (1-\beta)\mathcal{L}_{\text{c}}, \tag{2.8}$$

where $\mathcal{L}_{\text{MSE}}$ represents the weight reconstruction loss and $\mathcal{L}_{\text{c}}$ is a contrastive loss. The reconstruction term encourages the decoder to accurately reproduce the original model weights,

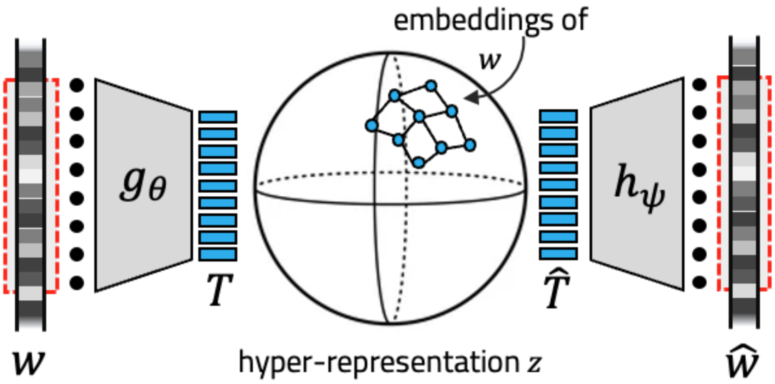


Figure 2.3: Sequential Autoencoder for Neural Embeddings generating a cloud of embeddings for a model weight [7]

expressed layer-wise as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{l=1}^{L} \|\hat{w}_i^{(l)} - w_i^{(l)}\|_2^2, \tag{2.9}$$

where $w_i^{(l)}$ and $\hat{w}_i^{(l)}$ are the original and reconstructed weights for the l-th layer of the i-th model in the collection (often called a *model zoo*).

The contrastive component $\mathcal{L}_c$ introduces additional structure in the latent space by applying data augmentations during training—such as weight permutation (which leverages inherent symmetries in neural networks) and random erasing—to ensure that semantically similar models are mapped closer together while dissimilar ones are pushed apart. In this work, the contrastive term follows the *NT-Xent* loss formulation (see Section **??**), which provides a more stable and expressive contrastive objective for modelling relationships in weight space.

This combination of reconstruction and contrastive learning encourages the encoder to capture not only numerical similarity in the weights but also functional and architectural relationships between models. The result is a structured latent space in which proximity correlates with similarity in performance or behaviour.

The learned latent space enables both discriminative and generative applications. The encoder can be used to extract informative embeddings that summarise a model's functional behaviour, useful for clustering, transfer learning, or model retrieval. Meanwhile, the decoder can generate entirely new sets of weights from latent vectors—supporting generative applications such as zero-shot model synthesis or interpolation between models. This dual capability makes the autoencoder framework particularly appealing for weight space learning, as it provides both interpretability and controllability.

A major challenge in encoding neural network weights lies in the enormous number of parameters, which quickly becomes infeasible to process directly. [7] introduced the Sequential

Autoencoder for Neural Embeddings (SANE) shown in Figure 2.3, which addresses this scalability problem by tokenising model weights rather than treating the entire parameter set as a single input.

Instead of encoding full weight tensors, SANE partitions them into smaller, semantically meaningful *tokens*—for example, by layer or even within layers—and encodes these sequentially. Each token is represented as a point in latent space, and the entire model is represented as a *cloud of latent tokens* rather than a single vector. This design assumes that sufficient information about the model's global behaviour is preserved through these local token representations, an assumption supported by empirical results in [7].

SANE achieves state-of-the-art performance in both discriminative and generative weight-space applications, demonstrating that this token-based decomposition maintains the essential structure of models while dramatically improving scalability. Furthermore, because the approach is sequential, it naturally accommodates heterogeneous architectures—normalising across different layer types and sizes—and allows the generation of new architectures by decoding variable-length token sequences.

Chapter 3

Methodology

Our hypothesis is that a unified representation space will enable conditional model sampling, allowing us to accurately approximate the conditional probability $P(W \mid \mathcal{D}, R)$. That is, we aim to sample model weights (W) conditioned on both specific dataset properties ($\mathcal{D}$) and target performance results (R).

To achieve this generative capability, our methodology requires a system that can project the three distinct data types—weights, datasets, and results—into a common, meaningful latent space. This mandates three core encoding components followed by a decoding mechanism for generation, the summation of these componets is visualised in Figure 3.1:

1. **Weight Encoder and Decoder ($\mathcal{W} \to Z_W \to \hat{\mathcal{W}}$):** To compress the high-dimensional weight tensor into a low-dimensional latent vector Z_W and reconstruct the functional weights.

2. **Dataset Encoder ($\mathcal{D} \to Z_D$):** To map the dataset characteristics into a latent vector Z_D.

3. **Results Encoder ($R \to Z_R$):** To map scalar performance metrics into a latent vector Z_R.

4. **Shared Embedding/Alignment:** A mechanism to ensure the resultant latent vectors (Z_W, Z_D, Z_R) are meaningfully aligned in a single, unified space Z.

In Section 3.1, we introduce a data generation approach that produces a diverse collection of $(\mathcal{D}, \mathcal{W}, \mathcal{R})$ triplets by systematically varying datasets, and optimisation parameters.

Section 3.2 presents the design of the full embedding model, including the weight encoder and decoder,dataset encoder , and results embdding. The weight encoder model transforms the high-dimensional weight tensors ($\mathcal{W}$) into compact latent representations. The dataset encoder maps each dataset ($\mathcal{D}$) into a semantic representation using pretrained CLIP features, while the results encoder transforms validation losses ($\mathcal{R}$) into fixed-size embeddings through a discretised lookup scheme.

Finally, Section 3.3 integrates these components within a shared latent space.The model learns to align the dataset and result embeddings with their corresponding weight representations,

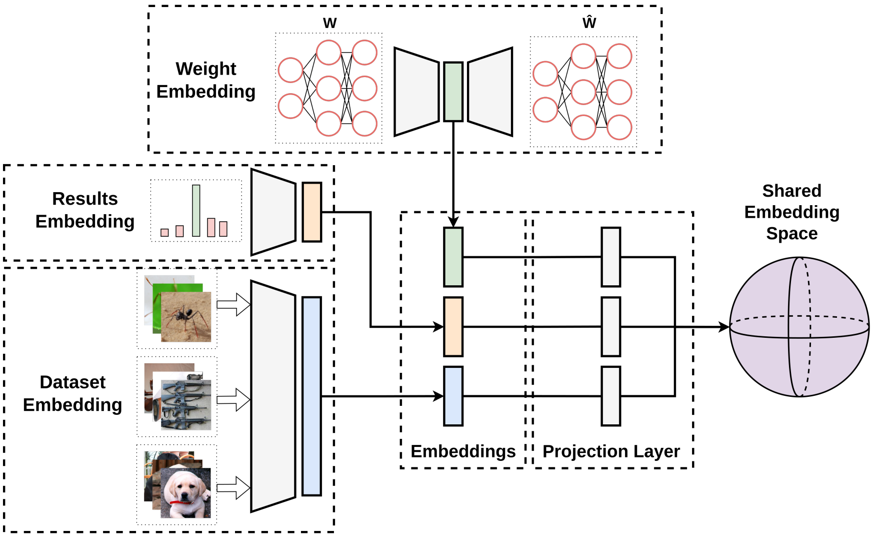


Figure 3.1: The process of embedding a dataset, model weigths and results into a shared embedding space.

forming a unified space that captures the relationships between data, model parameters, and performance.

3.1. Model Zoo Generation

The construction of the Model Zoo is fundamentally driven by the need to generate a diverse, reproducible, and analytically valuable dataset of model weights for comprehensive weight-space exploration. The methodological choices detailed below ensure the resulting zoo provides sufficient statistical power and representational richness for subsequent analyses.

The foundation of the Model Zoo is the classification task, which must be both representative and computationally manageable. Given that ImageNet is an established benchmark for representation learning in computer vision, its rich feature set was recognized as the appropriate basis for defining the task. To maintain a manageable level of complexity for large-scale generation and maximize the number of unique training instances, the core problem was defined as classifying randomly selected subsets of three ImageNet classes. This decision ensures the models are trained on high-fidelity, real-world visual features while simultaneously creating a highly flexible input-data space through numerous possible permutations.

The architecture was uniformly set to the ResNet-18 backbone across the entire zoo population, with only the final fully connected layer modified to accommodate the three-class classification task. By standardizing the architecture, any observed differences in the weight-space properties are primarily attributable to the systematic variations introduced through training parameters. While initial weight-space research focused on smaller models, ResNet-18 provides a sufficient architectural challenge necessary for direct comparability against modern methods, such as the scalability demonstrated in [citepaper] over previous methods like [citepaper].

To guarantee the Model Zoo accurately reflects the diversity of the high-dimensional weight space, variation is introduced at the start of each training run. This begins with the random selection of classification classes from ImageNet, with the primary mechanism for systematic variation being the hyperparameter sampling from the distributions detailed in Table 1. Furthermore, the optimizer is randomly selected to be either Adam or Stochastic Gradient Descent (SGD). This simple choice immediately adds substantial structural diversity by forcing models to traverse fundamentally distinct optimization paths across the loss landscape.

Table 3.1: Model Zoo Hyperparameter Sampling Distributions and Fixed Parameters

Hyperparameter	Distribution / Value
Problem Domain	Random subsets of 3 ImageNet classes
Model Architecture	ResNet-18 (Fixed)
Optimizer	Randomly selected: Adam or SGD
Early Epoch Snapshot	Uniform, $U(5, 10)$
Maximum Epoch	Truncated Normal, $N(\mu = 100, \sigma = 15)$ with bounds $[50, 150]$
Learning Rate (η)	$Insertspecific LR distribution$

To analyze the geometry of the weight space at different phases of optimization, two distinct weights snapshots are captured per training trajectory: an ’early epoch’ checkpoint and a ’max epoch’ checkpoint. The early epoch is sampled uniformly, specifically designed to capture models in low-performing, under-trained states. This intentional approach provides crucial data on the structure of the weight space’s nascent landscape and the optimization trajectory. Conversely, the maximum training epoch is sampled from a truncated Normal distribution. This range ensures the model has reached a point reflective of a converged, stable local loss minimum, thereby accurately representing the typical, high-performing outcomes of model training. Finally, the total sample size, X, is set to the maximum number computationally feasible given resource constraints. This practical constraint ensures the generation of a statistically representative number of unique training instances necessary for robust analysis of the weight space.

3.2. Model

3.2.1. Weight Encoder

The Weight Encoder is implemented to learn a compact, functional latent representation for neural network parameters obtained from the Model Zoo. The objective is to train an autoencoder that encodes the parameters of each model and reconstructs them with minimal deviation from the originals. This corresponds to minimising the reconstruction loss, $\mathcal{L}_{\text{MSE}}$, as defined in

Equation 2.9. To ensure generalisation, performance is validated on a separate validation set $\mathcal{M}_{\text{val}}$, restricted to ten distinct models due to limited availability of pre-trained networks.

An intuitive baseline would be to flatten the entire ResNet18 weight matrix $\mathbf{W}_m$ into a single high-dimensional vector and train an autoencoder directly on it. However, given that ResNet18 contains over 11 million parameters, this approach is computationally infeasible and would require an impractically large dataset for the model to generalise effectively [**?**].

Alternative methods such as sequential tokenisation of layers (discussed in Section 2.3) provide a more scalable representation but introduce additional architectural complexity. Instead, we adopt a simpler yet effective two-stage compression approach, beginning with a *Per-Named-Parameter PCA* transformation.

In the first stage, dimensionality reduction is applied separately to each named parameter tensor (for example, `conv1.weight` or `fc.bias`) across all models. This preserves the modular structure of the network while tailoring the compression to the statistical properties of each parameter type. The process for determining which compression method to apply is outlined in Algorithm 3.1.

Algorithm 3.1: Mode Selection During PCA Fitting

```
for each named parameter in Model Zoo do
    $V \leftarrow$ calculate_variance(parameter values across all models)
    $D \leftarrow$ calculate_dimension(parameter values across all models)
    if $V <$ VARIANCE_THRESHOLD then
        store_only_mean()                                  ▷ Retain only mean
    else if $D \leq$ DIMENSION_LIMIT then
        store_centered_weights()                           ▷ Retain centered weights
    else
        fit_IncrementalPCA()                               ▷ Fit IncrementalPCA
    end if
end for
```

Three compression modes are used:

- For high-variance, high-dimensional tensors, *IncrementalPCA* projects the centred weights onto a small number of principal components, retaining maximal variance in a compact coefficient representation.
- For small tensors (four dimensions or fewer), only centering is applied, as PCA provides negligible benefit.
- For low-variance tensors, only the mean value is stored, discarding coefficients entirely

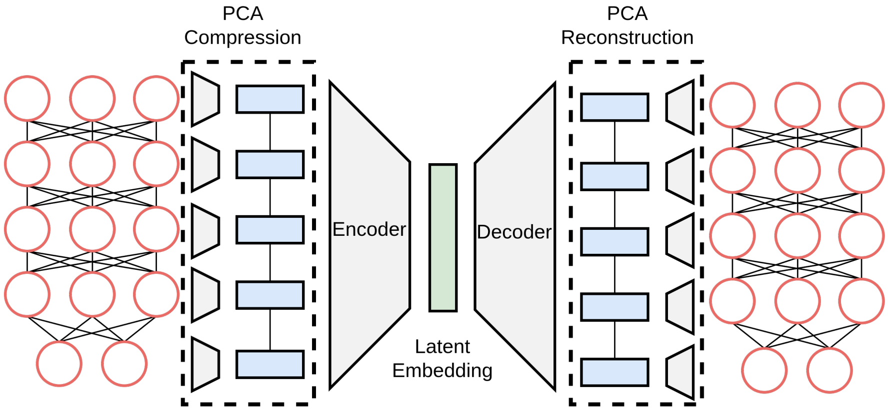


Figure 3.2: Two-stage compression of model weights: PCA reduces each parameter tensor, followed by an autoencoder producing a compact latent representation.

for maximal compression.

Following PCA, the resulting coefficient vectors obtained for each named parameter are concatenated to form a single reduced representation for each model. Each vector contains the retained PCA coefficients for one named parameter, where the number of retained components k_i varies depending on that parameter's variance structure. The concatenated coefficient vector, denoted $\mathbf{C}_m$, thus compactly represents all named parameters of model m in a unified format. This vector is normalised using dataset-wide statistics and then serves as input to the second compression step, a deep autoencoder. This two-stage compression process is illustrated in Figure 3.2, showing how PCA is applied per named parameter followed by a deep autoencoder producing a compact latent representation.

In the second stage, the autoencoder learns a lower-dimensional latent representation of the PCA-compressed vectors. The encoder maps the normalised coefficient vector $\mathbf{C}_m$ to a compact latent representation $\mathbf{z}_m \in \mathbb{R}^{D_{\text{latent}}}$:

$$\mathbf{z}_m = f_{\text{enc}}(\mathbf{C}_m; \theta_e),$$

and the decoder reconstructs it as

$$\hat{\mathbf{C}}_m = f_{\text{dec}}(\mathbf{z}_m; \theta_d).$$

Training minimises a per-named-parameter reconstruction objective, extending the standard mean-squared error loss in Equation 2.9 to operate across all parameters:

$$\mathcal{L}_{\text{AE}} = \frac{1}{|\mathcal{M}_{\text{train}}|} \sum_{m \in \mathcal{M}_{\text{train}}} \frac{1}{N_m} \sum_{i=1}^{N_m} \|\mathbf{c}_{i,m} - f_{\text{dec}}(f_{\text{enc}}(\mathbf{c}_{i,m}))\|_2^2, \tag{3.1}$$

where $\mathbf{c}_{i,m}$ denotes the PCA coefficient vector for the i-th named parameter of model m, and

N_m is the total number of named parameters for that model. This formulation ensures that the autoencoder learns to compress and reconstruct parameters in a balanced manner while maintaining structural variance across layers and modules.

After reconstruction, each parameter's coefficients $\hat{\mathbf{c}}_{i,m}$ are transformed back into their original tensor shapes using the stored PCA statistics from the fitting stage. Specifically, for each named parameter i, the inverse PCA transform restores the weight tensor by reprojecting the coefficients into the original feature space and re-adding the mean vector:

$$\hat{\mathbf{W}}_{i,m} = \mathbf{V}_i \hat{\mathbf{c}}_{i,m} + \boldsymbol{\mu}_i,$$

where $\mathbf{V}_i$ is the PCA component matrix and $\boldsymbol{\mu}_i$ is the mean of the original parameter distribution. If PCA was not applied (for small or low-variance tensors), the stored centring or mean operations are inverted accordingly.

The combined PCA and autoencoder encoding system offers several benefits. By applying PCA first, the dimensionality of each parameter tensor is significantly reduced, which lowers the computational and memory requirements for training the autoencoder. This high compression makes the system easier and faster to train while still capturing the dominant variance in the model weights. However, this approach also has inherent limitations. Since PCA compresses based on variance directions observed in the training models, unseen models with weights that are outliers or vary along unrepresented directions may be poorly encoded. Consequently, the system might not generalise well to models whose parameter distributions differ significantly from the training set.

3.2.2. Dataset Encoder

Following the discussion in Section 2.4 on contrastive learning and the CLIP framework, we adopt a pretrained CLIP visual encoder to extract semantic embeddings for the dataset classes. Each image x is passed through the CLIP encoder $f_{\text{CLIP}}(\cdot)$, producing a 512-dimensional latent representation z_x:

$$z_x = f_{\text{CLIP}}(x).$$

To obtain a class-level representation, we compute the embeddings for all images belonging to a given class and average them, yielding a single 512-dimensional vector $\bar{z}_c$ that summarises the class:

$$\bar{z}_c = \frac{1}{|X_c|} \sum_{x \in X_c} z_x,$$

where X_c denotes the set of images in class c.

Finally, the embeddings for the three classes are concatenated to form the final dataset-level representation:

$$\mathbf{z}_{\text{dataset}} = [\bar{z}_{c_1}; \bar{z}_{c_2}; \bar{z}_{c_3}] \in \mathbb{R}^{1536}.$$

This concatenated vector serves as the input to the shared encoding stage for alignment with the weight embeddings. This approach leverages the pretrained CLIP features to provide a rich semantic summarisation of each class, ensuring that the resulting dataset representation captures meaningful visual information relevant for model alignment.

3.2.3. Results Encoder

To capture the performance of each model, we use the validation loss recorded at the checkpoint corresponding to the saved weights. Instead of using the raw scalar loss directly, we quantize the range of validation losses observed across the Model Zoo into discrete bins. Each bin is associated with a learnable 512-dimensional embedding vector, allowing the model to map a validation loss to a rich latent representation:

$$\mathbf{r}_m = \text{Embedding}(\text{bin}(\mathcal{L}_{\text{val},m})),$$

where $\mathcal{L}_{\text{val},m}$ is the validation loss of model m, and $\mathbf{r}_m \in \mathbb{R}^{512}$ is the corresponding learned embedding.

This approach has several advantages. First, by representing the continuous range of validation losses with trainable embeddings, the system can learn a smooth latent space that captures nuanced performance differences between models. Second, it avoids the need to directly regress continuous loss values, which can be noisy and poorly scaled across diverse architectures or training schedules.

3.3. Shared Encoding

The shared encoding stage aligns the dataset and results embeddings with the weight latent space. To achieve this, the dataset-level vector $\mathbf{z}_{\text{dataset}}$ from Section **??** and the results embedding $\mathbf{z}_{\text{results}}$ from Section **??** are first concatenated:

$$\mathbf{z}_{\text{input}} = [\mathbf{z}_{\text{dataset}}; \mathbf{z}_{\text{results}}].$$

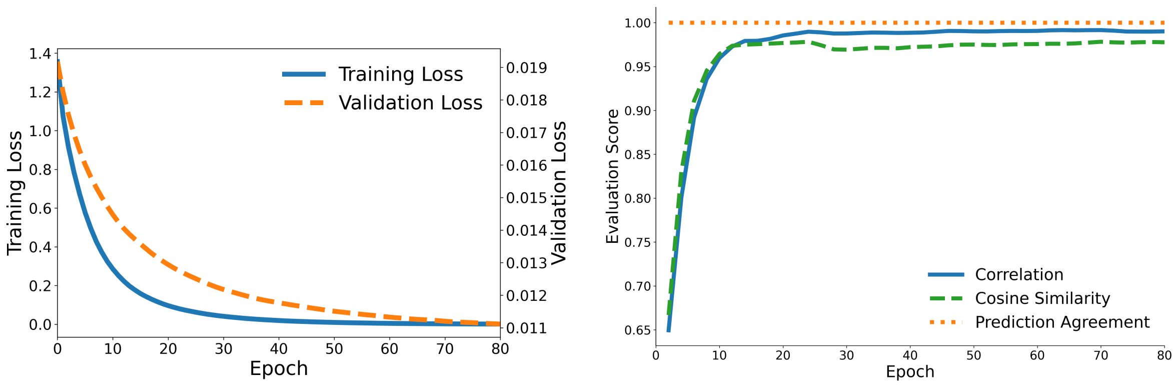
This combined vector is then passed through a multi-layer perceptron (MLP) to produce a predicted embedding $\hat{\mathbf{z}}_{\text{weight}}$ in the weight latent space:

$$\hat{\mathbf{z}}_{\text{weight}} = \text{MLP}(\mathbf{z}_{\text{input}}; \theta_{\text{MLP}}).$$

Training minimises the NT-Xent contrastive loss, as described in Section 2.4 and defined in Equation 2.8, between $\hat{\mathbf{z}}_{\text{weight}}$ and the corresponding weight embedding $\mathbf{z}_{\text{weight}}$. Backpropagation is used to update the MLP parameters θ_{MLP}:

$$\mathcal{L}_{\text{NT-Xent}}(\hat{\mathbf{z}}_{\text{weight}}, \mathbf{z}_{\text{weight}}) \to \min_{\theta_{\text{MLP}}}.$$

This design ensures that the mapping from dataset and results space to the weight latent space is fully differentiable and preserves the reversibility of the weight representation. Importantly, the projection is performed from dataset and results to weight space rather than the reverse: applying a non-linear transformation directly to the weight embedding would break invertibility, making it impossible to reconstruct the original weight vector from the shared representation. By contrast, pushing the dataset and results embeddings to the weight space allows the latent weight vector to remain consistent and decodable via the trained autoencoder.



(a) Training and Validation Loss

(b) Test Model Output Comparison

Figure 4.1: Performance curves of the Weight Autoencoder, showing loss progression and comparative output metrics.

Chapter 4

Results

4.1. Weight Autoencoder

PCA is initially applied to the training set of the Model Zoo, after which the autoencoder is subsequently trained.

To evaluate the efficacy of the Weight Autoencoder, the loss function evolution during training is analyzed. Figure 4.1a illustrates the loss function for the Weight Autoencoder following the PCA projection. The training loss consistently decreases over epochs, eventually reaching a plateau. The validation loss exhibits a similar trajectory, decreasing as training progresses. This consistent performance indicates that the model is not overfitting to the training data and demonstrates a capacity for generalization to unseen data. Specifically, this suggests that the Weight Autoencoder has successfully learned to encode and decode model representations beyond the training set, enabling the generation and reconstruction of effective latent representations for the designated task.

The MSE Loss, depicted in Figure 4.1a, confirms that the Weight Autoencoder is progressing in solving the reconstruction task. However, since the magnitude of the error is relative to the scale of the input model weights, it is not directly interpretable as a measure of preserved model performance; thus, a comparative evaluation is requisite.

Figure 4.1b presents the results from a comparative experiment: unseen models are encoded and reconstructed, and the reconstructed models are then utilized to classify a specific test dataset. The output of the original model is compared against the output of the reconstructed model. Throughout the training, the model agreement on synthetic images was 100%. Model agreement is defined as the percentage of test samples for which the classified label is identical between the original and reconstructed models. Correlation and cosine similarity were calculated on the pre-softmax outputs. It is evident that the Weight Autoencoder is capable of learning model representations with sufficient fidelity to reconstruct models exhibiting highly correlated outputs—specifically, $0.95 + \%$ correlation—compared to the original model's outputs.

The same model output comparison procedure is then executed, but using the data upon which the original model was initially trained. Table 4.1 summarizes these cross-dataset performance metrics.

Table 4.1: Model Reconstruction Performance on Training Data

Average Metric	Value
Cosine Similarity (Pre-Softmax Output)	-0.0323
Correlation (Pre-Softmax Output)	-0.08570
Predition Agreement	35.6905%

It is clear from these metrics that the Weight Autoencoder is unable to reconstruct models with sufficient quality to preserve the decision boundary of the original model. The significant discrepancy in output correlation between the synthetic and real data indicates that while a representation of the ResNet18 model is being learned, it lacks the necessary fine-grained resolution required for accurate reconstruction of functional models.

4.2. Shared Encoder

The NT-Xent loss is minized on a traing set of model weights, dataset and results, with a

4.2.1. Conditional Model Sampling

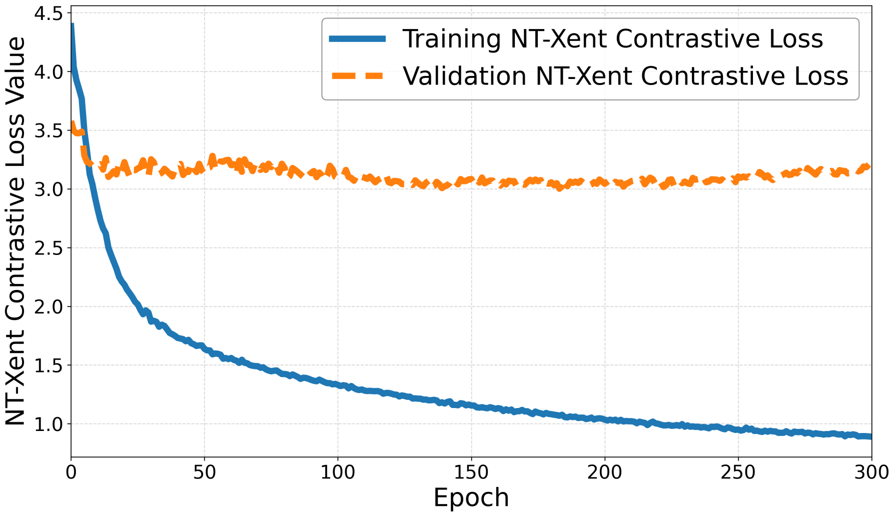

Training NT-Xent Contrastive Loss
Validation NT-Xent Contrastive Loss
NT-Xent Contrastive Loss Value
Epoch
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
50
100
150
200
250
300

Figure 4.2

Chapter 5

Summary

Bibliography

[1] J. Dean, "Keynote speech on the future of ai," https://www.google.com/search?q=https://events.google.com/io/, 2017, statement widely attributed to the keynote speech, highlighting the challenge of interpretability in deep learning systems.

[2] Hugging Face, "Open-source AI: Year in Review 2024," https://huggingface.co/spaces/huggingface/open-source-ai-year-in-review-2024, 2024, accessed: 14 October 2025.

[3] K. Schürholt, D. Taskiran, B. Knyazev, X. G. i Nieto, and D. Borth, "Model zoos: A dataset of diverse populations of neural network models," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. [Online]. Available: https://openreview.net/forum?id=MOCZI3h8Ye

[4] T. Unterthiner, D. Keysers, S. Gelly, O. Bousquet, and I. Tolstikhin, "Predicting neural network accuracy from weights," 2021. [Online]. Available: https://arxiv.org/abs/2002.11448

[5] M. Salama, J. Kahana, E. Horwitz, and Y. Hoshen, "Dataset size recovery from lora weights," 2024. [Online]. Available: https://arxiv.org/abs/2406.19395

[6] K. Schürholt, B. Knyazev, X. G. i Nieto, and D. Borth, "Hyper-representations as generative models: Sampling unseen neural network weights," 2022. [Online]. Available: https://arxiv.org/abs/2209.14733

[7] K. Schürholt, M. W. Mahoney, and D. Borth, "Towards scalable and versatile weight space learning," in *Proceedings of the 41st International Conference on Machine Learning (ICML)*. PMLR, 2024.

[8] B. Soro, B. Andreis, S. Chong, and S. J. Hwang, "Instruction-guided autoregressive neural network parameter generation," in *Workshop on Neural Network Weights as a New Data Modality*, 2025. [Online]. Available: https://openreview.net/forum?id=QutFK34ea1

[9] L. Meynent, I. Melev, K. Schürholt, G. Kauermann, and D. Borth, "Structure is not enough: Leveraging behavior for neural network weight reconstruction," in *Workshop on Neural Network Weights as a New Data Modality*, 2025. [Online]. Available: https://openreview.net/forum?id=APsHrpqO3W

[10] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, “Learning transferable visual models from natural language supervision,” in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[11] W. Ågren, “The nt-xent loss upper bound,” 2022. [Online]. Available: https://arxiv.org/abs/2205.03169

[12] K. Schürholt, B. Knyazev, X. Giró-i Nieto, and D. Borth, “Hyper-representations as generative models: Sampling unseen neural network weights,” in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 27 906–27 920. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/b2c4b7d34b3d96b9dc12f7bce424b7ae-Paper-Conference.pdf