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

Declaration

By submitting this project electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

In the preparation of this project, artificial intelligence (AI) tools were utilised solely as supportive instruments to enhance the quality and efficiency of the research process. The use of AI was limited to improving the clarity, structure, and grammatical quality of the written content. All technical content, experimental design, analysis, and conclusions presented are entirely the result of my own work, understanding, and interpretation. Any text generated through AI assistance were critically reviewed, revised, and restructured to ensure full alignment with my personal voice, intended meaning, and academic integrity.

With regard to coding, AI tools were employed to expedite routine programming tasks such as generating boilerplate code, refactoring existing code for improved readability, and identifying potential bugs. All AI-assisted code was carefully validated, modified where necessary, and verified to behave as intended. The use of AI in this context served only to improve workflow efficiency and did not influence the originality or integrity of the implemented methods or results.

Date: November 2025 .....

Abstract

Neural networks have become a cornerstone of modern artificial intelligence, but challenges remain in understanding their internal representations and efficiently generating new models. Weight space learning offers a framework for addressing these challenges by modelling the structure of neural network parameters and their relationship to datasets and performance. In this work, we propose a contrastive learning framework that jointly embeds neural network weights, the datasets they are trained on, and their resulting performance metrics into a shared embedding space. Using separate encoders for weights and datasets, along with a learned embedding for discretised performance bins, our method applies a contrastive objective to align these heterogeneous modalities.This unified embedding space enables interpretability—by revealing relationships between datasets and model behaviour—and conditional model sampling, approximating the joint distribution $P(W \mid \mathcal{D}, R)$. Experimental results highlight limitations in current weight encoding strategies, particularly linear PCA compression, which constrain generalisation. Nonetheless, the framework demonstrates the potential of contrastively aligning heterogeneous modalities, providing a foundation for future work aimed at improving embedding quality, expanding model diversity, and enhancing conditional model generation.

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]

Neural networks (NNs) have become a foundational technology in modern artificial intelligence, yet their widespread use has revealed important limitations. Among the most significant are their lack of explainability—the "black-box" effect—and the high computational cost of training. With platforms like Hugging Face and GitHub now hosting over a million pre-trained models [2], these challenges have drawn increasing attention. Motivated by this abundance of models, a new 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.

Discriminative weight space learning aims to extract meaningful information from pre-trained network weights. Embeddings of model weights are typically evaluated through tasks that predict meta-information about the original models, such as final performance or generalisation behaviour. These tasks demonstrate that weight embeddings capture informative aspects of the training process, providing both a measure of representation quality and practical predictive insights.

Generative weight space learning, by contrast, focuses on modelling the distribution of neural network weights $P(W \mid \dots)$ to enable the synthesis of new models. Early approaches [3, 4] employ autoencoders to learn compact latent representations of model zoos, facilitating the generation of novel weight configurations. Subsequent work [5], [6] has explored conditioning these distributions on additional information, such as the dataset a model was trained on

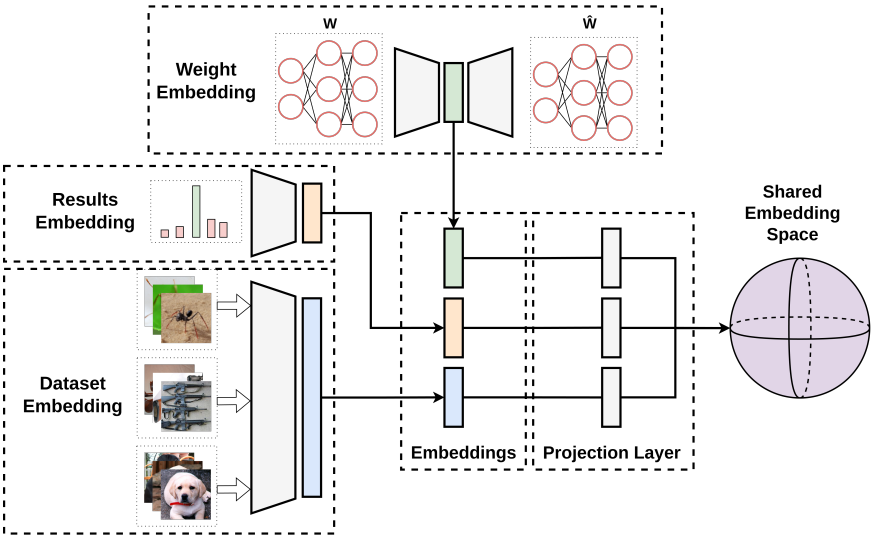


Figure 1.1: A Conditonal Model Sampling system, embedding a dataset, model weights and results into a shared embedding space

or its resulting performance, enabling the generation of weights tailored to specific tasks or behaviour patterns.

Although these approaches have advanced the field significantly, they typically focus on isolated aspects of weight learning. Current methods address either dataset-conditioned or performance-conditioned distributions, but rarely both simultaneously. This separation overlooks the fact that a model's weights are shaped by both the data it is trained on and the results it achieves. Capturing the joint relationship $P(W \mid \mathcal{D}, R)$ is therefore critical for fully understanding model behaviour and for generating models with specific, desired characteristics.

Contrastive learning has emerged as a powerful paradigm for learning unified representations across modalities, as demonstrated by models such as CLIP [7], which align vision and language in a shared representation space. A contrastive objective pulls related samples closer in representation space while pushing unrelated samples apart, creating a meaningful joint embedding space for heterogeneous data types. This property makes contrastive learning particularly suitable for modelling the complex distribution $P(W \mid \mathcal{D}, R)$, as it can embed multiple, fundamentally different data modalities into a common space without requiring them to share the same input dimensionality or structure.

In this report, we propose a contrastive learning framework to create a unified embedding space that jointly represents neural network weights , the datasets they were trained on, and their resulting performance characteristics. 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 [8], which encourages related triplets $(\mathcal{D}, W, R)$ to be close in the shared embedding space. Figure 1.1 depicts a high-level view of

the full embedding pipeline.

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 $P(W \mid \mathcal{D}, R)$.

The primary goal of this report is to demonstrate the viability of our methodology through successful conditional model sampling. While our hypothesis addresses both interpretability and conditional model generation, success in the latter provides strong evidence for the former. Specifically, the ability to accurately sample weights that achieve target performance metrics, conditioned on specific dataset and result targets, indicates that the unified embedding space has effectively captured the complex relationship between $(\mathcal{D}, R)$, and W. Accordingly, this report focuses on generating meaningful representations, evaluating their quality, and performing conditional model sampling to assess the effectiveness of the proposed methodology for generative weight space modelling.

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. Its 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 W of a neural network is formally defined as the high-dimensional Euclidean space $\mathbb{R}^N$ spanned by the N learnable parameters [9]—the weights and biases—of the specified architecture. A specific point $\mathbf{w} \in W$ represents a complete instantiation of the model, mapping input data $\mathbf{x}$ to output predictions $f(\mathbf{x}, \mathbf{w})$.

Weight space learning can be broadly divided into generative and discriminative tasks. Generative approaches aim to characterise the distribution of neural network weights, potentially conditioned on auxiliary information or reference models, $P(W \mid \dots)$, enabling the synthesis of novel weight configurations.

For instance, [3] employed an autoencoder to create compact representations of model zoos, with the decoder being used to generate new weights. Building on this concept, [4] proposed the Sequential Autoencoder for Neural Embeddings (SANE), which introduces weight tokenisation and sequential embedding to improve scalability for larger networks.

Other methods focus on conditioning the weight distribution on dataset or performance information. [5] employed a Vector Quantised Variational Autoencoder (VQ-VAE) to model $P(W \mid \mathcal{D})$, learning latent representations that allow generation of new weights for specific tasks. Similarly, [6] modelled $P(W \mid R)$ using a contrastive objective that minimises differences in both model outputs and weights between original and reconstructed networks, effectively embedding behavioural information into the embedding space.

Discriminative weight space learning, in contrast, uses pre-trained network weights—often aggregated in model zoos [10]—to infer meta-information about the original models [11].

Representations are typically assessed by training a simple predictor, such as an MLP, to map embeddings to model metrics, including final performance or generalisation gap. These tasks demonstrate that weight embeddings encode meaningful training information, providing both a benchmark for representation quality and practical predictive utility.

Discriminative and generative tasks alike operate on high-dimensional weight spaces, which can consist of millions of parameters in modern networks. To manage this complexity, latent representations are used to encode the essential information of the weights into a more compact form. Such representations encode a model's parameters into a compact embedding space through learned transformations—either linear or non-linear—that preserve essential information about the original weights. A key distinction among dimensionality reduction methods is whether they are reversible or non-reversible.

Reversibility is particularly important in weight space learning. While encoding weights into a latent representation (real $\rightarrow$ latent) is informative, the ability to reconstruct the original weights (real $\rightarrow$ latent $\rightarrow$ 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.

We next examine two fundamental approaches to creating such reversible representations: Principal Component Analysis (PCA), which provides linear but interpretable transformations, and autoencoders, which enable more expressive non-linear mappings. We begin with PCA as it establishes the baseline for reversible dimensionality reduction.

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 embedding 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, as illustrated in Figure 2.1. Their primary goal is to learn a compressed, informative latent representation of input data by training the system to reconstruct the original input after it passes through a lower-dimensional bottleneck layer [12]. The full architecture consists of an encoder and a decoder. Crucially, the layers within these components typically employ non-linear activation functions, enabling the model to perform complex, non-linear mappings from the high-dimensional input space to the embedding space. This capability is essential for capturing intricate, hierarchical structures within the data that linear dimensionality reduction techniques cannot access.

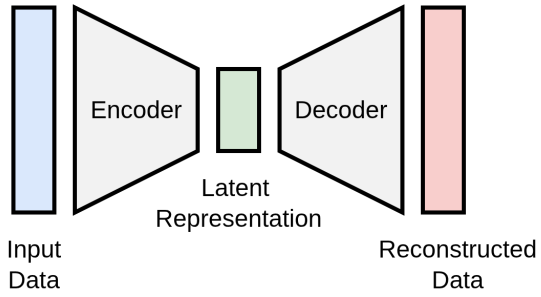


Figure 2.1: Autoencoder encoding input data into a latent representation and decoding it to reconstructed data

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 (embedding 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

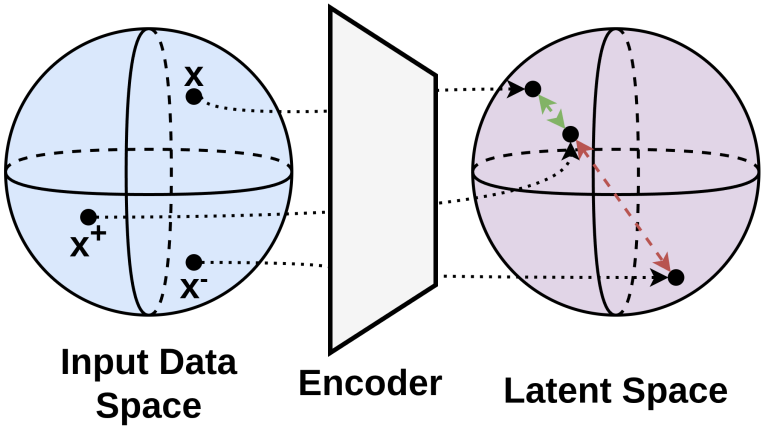


Figure 2.2: Contrastive encoder pulling similar data pairs (x, x^+) together in embedding space and pushing dissimilar pairs (x, x^-) apart [13].

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 embedding 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 embedding 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 [13]. 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.

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 [8]. This loss formulates the contrastive task as a classifi-

cation 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 embedding 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) [7]. 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 embedding space that captures structural and functional similarities between models. This approach was explored in [14], 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, 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.

The contrastive component $\mathcal{L}_{\text{c}}$ introduces additional structure in the embedding 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 2.4), 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 embedding space in which proximity correlates

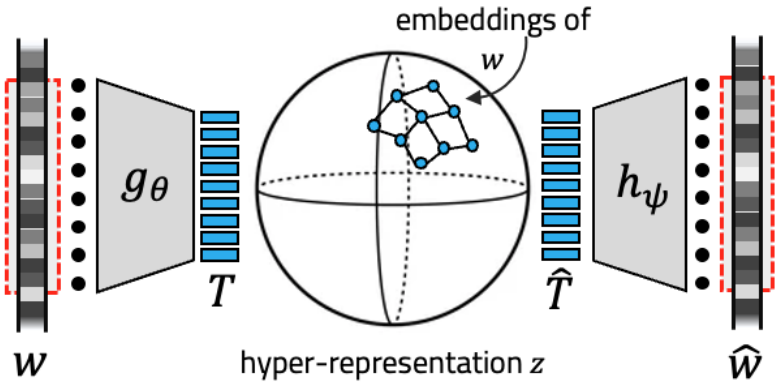


Figure 2.3: Sequential Autoencoder for Neural Embeddings generating a cloud of embeddings for a model weight [4]

with similarity in performance or behaviour.

The learned embedding 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. [4] 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 embedding 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 [4].

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 embedding space. This mandates three core encoding components followed by a decoding mechanism for generation, the summation of these componets is visualised in Figure 1.1:

1. **Weight Autoencoder ($W \rightarrow Z_W \rightarrow \hat{W}$):** To compress the high-dimensional weight tensor into a low-dimensional latent vector Z_W and reconstruct the functional weights.
2. **Dataset Encoder ($\mathcal{D} \rightarrow Z_{\mathcal{D}}$):** To map the dataset characteristics into a latent vector Z_D.
3. **Results Encoder ($R \rightarrow Z_R$):** To map scalar performance metrics into a latent vector Z_R.
4. **Shared Embedding/Alignment:** A mechanism to ensure the resultant latent vectors $(Z_W, Z_{\mathcal{D}}, Z_R)$ 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 autoencoder,dataset encoder , and results embedding. The weight autoencoder 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 embedding space.The model learns to align the dataset and result embeddings with their corresponding weight representations, forming a unified space that captures the relationships between data, model parameters,

and performance.

3.1. Model Zoo Generation

We constructed the model zoo to generate a diverse, reproducible, and analytically valuable dataset of model weights for comprehensive weight-space exploration.

The model zoo is composed of numerous models trained on tasks sampled from the same underlying distribution. For our study, this distribution is defined by 3-class classification problems drawn from ImageNet [15], a widely used benchmark known for its diversity and richness of visual features. Restricting tasks to three randomly selected classes strikes a balance between computational tractability and the creation of a large number of distinct training instances. This design ensures that the models learn from high-quality, real-world visual data while providing a flexible and varied task space through the many possible class combinations.

The architecture was consistently set to a ResNet-18 backbone [16] across the entire zoo population, with only the final fully connected layer adapted for the three-class classification task. This architectural standardisation ensures that any observed differences in weight-space properties are primarily attributable to the systematic variations in training parameters and the task distribution.

While early weight-space research often focused on smaller models, ResNet-18 introduces a more challenging architecture that remains compatible with modern approaches–enabling direct comparison to recent scalable methods such as [4], while exceeding the limitations of earlier techniques like [3].

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), adding structural diversity by allowing distinct optimization paths.

Table 3.1: Model Zoo Hyperparameter Sampling Distributions and Fixed Parameters

Hyperparameter	Distribution / Value	Type
Model Architecture	ResNet-18	Fixed
Problem Domain	Random subsets of 3 ImageNet classes	Sampled
Optimizer	Randomly selected: Adam or SGD	Sampled
Early Epoch Snapshot	Uniform, $U(5, 10)$	Sampled
Maximum Epoch	Truncated Normal, $N(100, 15)$, clipped to $[50, 150]$	Sampled

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, 1000, 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. Triplet Encoders

3.2.1. Weight Autoencoder

The weight autoencoder is implemented to learn a compact, functional latent representation for neural network parameters. 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, the model is trained on the training set $\mathcal{M}_{\text{train}}$, and performance is validated on a separate validation set $\mathcal{M}_{\text{val}}$, with 100 and 50 samples held out for testing final performance metrics.

An intuitive baseline is to flatten the entire ResNet18 weight tensor $\mathbf{W}_m$ into a single high-dimensional vector and train an autoencoder directly on it. However, ResNet18 contains over 11 million parameters, so even a single linear layer mapping this input to a modest hidden dimension quickly becomes impractical. For instance, mapping the flattened weights (~ 11.7M parameters) to a hidden dimension of 512 requires approximately 5.99 billion parameters. At 16-bit precision, this alone consumes nearly 12 GB of memory, demonstrating that a naive fully-connected approach is infeasible.

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 mode 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_PCA() ▷ Fit PCA
  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 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 3.1, 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

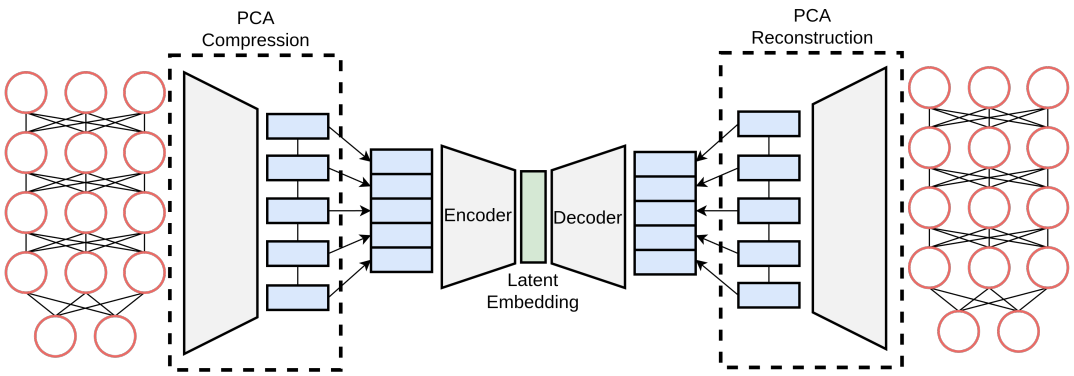


Figure 3.1: Two-stage compression of model weights: PCA reduces each parameter tensor, followed by an autoencoder producing a compact latent representation.

latent representation $\mathbf{z}_m \in \mathbb{R}^{D_{\text{latent}}}$:

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

where θ_e denotes the learnable parameters of the encoder. The decoder reconstructs the original coefficients as

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

with θ_d representing the learnable parameters of the decoder.

Training minimises the reconstruction loss of the PCA-compressed model weights, extending the standard mean-squared error loss in Equation 2.9:

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

where $\mathbf{C}_m$ denotes the PCA coefficient vector representing all weights of model m. This formulation encourages the autoencoder to capture the overall structure of each model in a compact embedding space while preserving the key variance across the full set of parameters.

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 combination of PCA and an autoencoder provides several advantages. Applying PCA first drastically reduces the dimensionality of each model's weight vector, lowering both computational and memory demands for training the autoencoder. This compression enables faster and more efficient training while still capturing the dominant variance patterns present in the model weights.

Despite these benefits, the approach has inherent limitations. PCA is a linear technique, which means it can only approximate the weight space along linear directions and may miss nonlinear dependencies between parameters. Additionally, it compresses based on the directions of variance observed in the training models, so models with weight configurations that vary along previously unrepresented directions may also be poorly encoded. Both factors can limit generalisation, particularly when the autoencoder encounters models with parameter distributions that differ substantially 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.

Each $\bar{z}_{c_i}$ represents the embedding of a specific class in the dataset, with c_1, c_2, c_3 consistently ordered according to the class indices in the classification layer. This consistent ordering preserves structure in the concatenated vector

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

which is then used as input to the shared encoding stage for alignment with the weight embeddings. By leveraging pretrained CLIP features, this approach provides a rich semantic

summarisation of each class, ensuring the final 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 embedding 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 embedding space. To achieve this, the dataset-level vector $\mathbf{z}_{\text{dataset}}$ from Section 3.2.2 and the results embedding $\mathbf{z}_{\text{results}}$ from Section 3.2.3 are first concatenated:

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

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

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

s 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}:

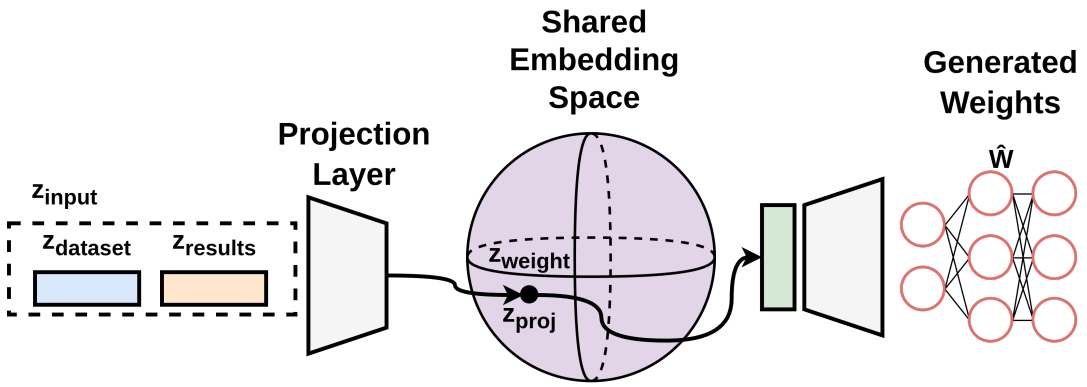


Figure 3.2: Conditional model sampling pipeline shown, from dataset and results embedding, to a shared embedding space, to a reconstructed model

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

This design ensures that the mapping from dataset and results space to the weight embedding 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.

The overall conditional model sampling process is illustrated in Figure 3.2. Given a dataset embedding $\mathbf{z}_{\text{dataset}}$ and results embedding $\mathbf{z}_{\text{results}}$, the shared encoder projects their concatenated representation into the latent weight space as $\mathbf{z}_{\text{proj}}$. Once aligned, this predicted latent vector can be passed through the trained weight decoder to reconstruct the full set of model parameters. In effect, this enables conditional generation of neural networks—sampling model weights that are consistent with a given dataset and desired performance profile.

Chapter 4

Results

4.1. Weight Autoencoder

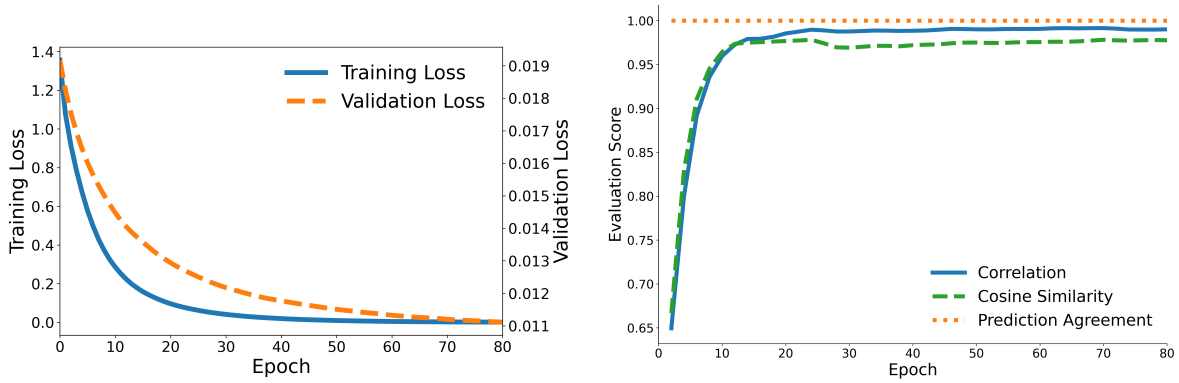
To evaluate the efficacy of the weight autoencoder, we examine the evolution of the mean squared error (MSE) during training. Figure 4.1a plots the MSE observed on both the training and validation sets following PCA projection of the model weights. The training error consistently decreases over epochs, eventually reaching a plateau, while the validation error follows a similar trajectory. This parallel behaviour suggests that the model is not overfitting to the training data and is capable of generalizing to unseen weight representations.

The use of MSE as the reconstruction metric confirms that the weight autoencoder is learning a meaningful mapping from weights to latent space and back. However, since the magnitude of the error depends on the scale of the input model weights, the absolute MSE value is not directly interpretable in terms of downstream model performance. Consequently, additional comparative evaluations are necessary to assess how well the learned representations preserve functionally relevant properties.

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.

These metrics show that the Weight Autoencoder fails to reconstruct models with enough



(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.

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%

fidelity to preserve the original decision boundaries. The large drop in output correlation between the reconstructed and real models suggests that, while a coarse representation of the ResNet18 weights is captured, critical fine-grained information required for functional accuracy is lost.

This degradation is likely caused by the initial PCA compression stage. Although PCA theoretically retains the principal directions of variance and thus the most informative components, it remains a linear approximation of a highly non-linear parameter space. Consequently, important non-linear relationships within the weights may have been discarded during projection. Given that the Autoencoder itself achieves low reconstruction loss and generalises well, as shown in Figure 4.1a, the poor reconstruction performance can reasonably be attributed to information loss introduced by PCA.

NOTE: When I tried to use a more expressive model the validaiotn loss of the autoencoder kept increasing for decreasing training loss, only the smallest possible model seemed to do well, no hidden layers. This indicates to me severe overfitting, despite the model being relatively small. This may be yet another indicaiton of the fact that the PCA compression did not retain sufficient information, as a more expressive encoder would have led to better results. Should I include a plot of this with my logic and conclusions outlined as above ?

NOTE: Also when I used a per param pca componets value of 64 and a more expresive model, reconstrcution and output performance metrics improved, though marginally. the NXT-loss

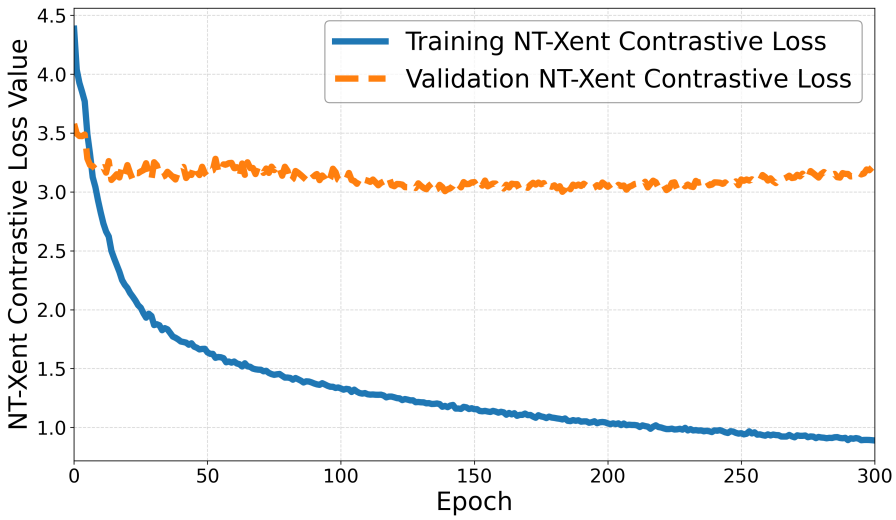


Figure 4.2: Training and validation losses for the shared encoder. The validation loss remains high, indicating a lack of generalisable signal due to information loss in the weight embeddings.

went up, but this is expected and the reconstruction is what matters. This may indicate that shifting to more compression in the encoder and less in the PCA may lead to better performance. should I include these plots and discussion as well?

4.2. Shared Encoder

The shared encoder was trained following the procedure described in Section 3.3, with both training and validation losses monitored throughout. Figure 4.2 shows that the training loss decreases rapidly at first and then decreases at a steady rate, while the validation loss decreases slightly before stabilising at a relatively high value of $\sim$3.1.

This behaviour suggests that the shared encoder is successfully fitting the training data but fails to generalise to unseen data. The persistently high validation loss indicates that the model cannot extract a meaningful or consistent signal from the validation set, likely due to information loss introduced earlier in the weight encoding process. As the weight representations lack sufficient structure or variance relevant to the corresponding $(\mathcal{D}, R)$ pairs, the shared encoder effectively trains on noise.

Interestingly, the validation loss does not worsen over time, which would typically occur during overfitting. This further supports the hypothesis that there is little to no informative signal to learn—changes to the shared encoder's weights have minimal effect on generalisation performance, implying that the training dynamics are dominated by random correlations rather than meaningful alignment.

Figure 4.3 provides further insight into this behaviour by visualising the cosine similarity between five weight embeddings and their corresponding projected embeddings z_{proj} within

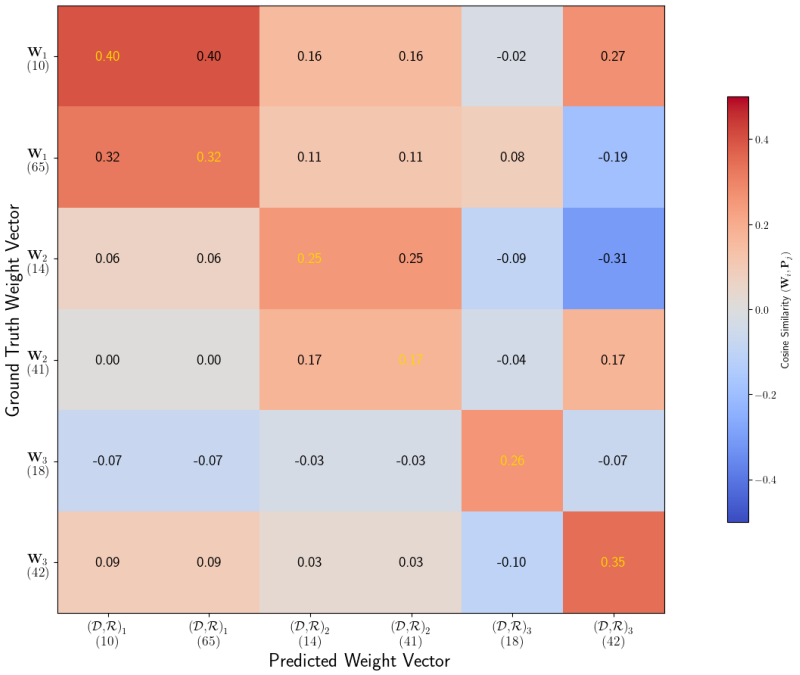


Figure 4.3: Cosine similarity heat map between five weight embeddings and their corresponding projected embeddings z_{proj}.

the training set. The x-axis represents $(\mathcal{D}, R)$ pairs at both early and late training epochs, while the y-axis shows the corresponding weight latent vectors from the weight autoencoder.

High cosine similarity values (red regions) are observed along the diagonal, showing that matching pairs of weights and $(\mathcal{D}, R)$ embeddings are aligned in the shared space. Negative pairs, shown as white or blue, exhibit low similarity as expected. Notably, the same $(\mathcal{D}, R)$ pairs across different epochs also display higher similarity, despite not being explicitly defined as positives. This suggests that the shared encoder partially retains the relational structure among triplets $(\mathcal{D}, R, W)$ across training stages, even in the absence of a strong global signal.

Chapter 5

Conclusion

This work set out to explore whether it is possible to learn a unified embedding space that jointly represents datasets, neural network weights, and performance outcomes. The study aimed to approximate the conditional distribution $P(W \mid \mathcal{D}, R)$ through contrastive alignment, enabling both interpretability and conditional model generation within the weight space.

A modular system was developed consisting of three components: a weight autoencoder for compressing model parameters, a dataset encoder based on pretrained CLIP features, and a shared encoder trained with the NT-Xent contrastive objective to align these representations. This framework was designed to test whether the learned embedding space could both capture meaningful structural relationships between datasets, results, and weights, and facilitate the conditional sampling of new model weights.

The results revealed that while the weight autoencoder successfully reconstructed latent representations of model parameters with low reconstruction loss, these reconstructions lacked sufficient detail to preserve the decision boundaries of the original networks. This is attributed primarily to the linear compression applied via PCA, which removed essential non-linear structure from the weight space. Consequently, the shared encoder lacked a strong, learnable signal linking the dataset–results pairs $(\mathcal{D}, R)$ to their corresponding weight embeddings, leading to weak generalisation and high validation loss. Despite this limitation, visual analyses of the shared embedding space suggested that some relational structure between modalities was maintained, indicating potential for further refinement.

The findings suggest that while the overall methodology is viable, effective joint modelling of $P(W \mid \mathcal{D}, R)$ requires more expressive weight encoders and non-linear compression methods capable of preserving fine-grained dependencies. Nonetheless, the results show that contrastive alignment across heterogeneous modalities is highly constrained, and the current approach is insufficient to capture the joint relationships needed for effective conditional weight modelling.

Future research directions are discussed in Section 5.1. In summary, potential improvements include exploring non-linear encoding methods to retain more structural information in weight embeddings, expanding the model zoo with a greater variety of architectures, and enhancing

dataset encoding strategies.

In conclusion, this study contributes an initial framework for contrastive, dataset-conditioned weight modelling. While preliminary, it highlights the promise of learning shared latent representations that unify models, data, and outcomes—an essential step toward interpretable and generative understanding of neural network weight spaces.

5.1. Future Work

Future research could focus on improving the weight autoencoder by exploring alternative architectures and compression strategies. One potential avenue is to reverse the PCA and autoencoder stages: first capture non-linear relationships in the weights, then apply linear compression, with a tunable hyperparameter controlling the relative contribution of each stage. This approach could mitigate information loss from linear compression while retaining effective dimensionality reduction.

Another direction is to implement the Sequential Autoencoder for Neural Embeddings (SANE), which may produce more representative weight embeddings. Expanding the model zoo with more models, or experimenting with smaller architectures, could help assess the feasibility and robustness of the approach before scaling to larger networks.

Enhancements to dataset encoding could also improve performance. Techniques from dataset distillation, such as generating a single representative image per class via gradient matching [17], may provide a more efficient and semantically meaningful summary of each dataset. This would reduce computational requirements while preserving the most relevant information for alignment with weight embeddings.

Additionally, introducing probabilistic conditioning mechanisms—such as variational embedding spaces or diffusion-based sampling—could improve the stability and controllability of conditional model generation. These methods would allow sampling from a structured embedding space, offering a principled way to model uncertainty and variability across datasets and performance targets.

Finally, increasing the size, diversity, and capacity of the model zoo could lead to improved results. It is possible that capturing the joint relationship between datasets, weights, and performance requires weight vectors sampled at multiple points during training, along with several thousand models and an effective weight encoder, before meaningful conditional model generation can be achieved.

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, 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

[4] 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.

[5] 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

[6] 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

[7] 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

[8] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. Daumé III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 1597–1607. [Online]. Available: https://proceedings.mlr.press/v119/chen20j.html

[9] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Lebanon and S. V. N. Vishwanathan, Eds., vol. 38. San Diego, California, USA: PMLR, 09–12 May 2015, pp. 192–204. [Online]. Available: https://proceedings.mlr.press/v38/choromanska15.html

[10] 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

[11] 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

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.cs.toronto.edu/~hinton/absps/science.pdf

[13] A. Torralba, P. Isola, and W. Freeman, *Foundations of Computer Vision*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2024. [Online]. Available: https://mitpress.mit.edu/9780262048972/foundations-of-computer-vision/

[14] 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

[15] J. Deng, K. Li, M. Do, H. Su, and L. Fei-Fei, "Construction and Analysis of a Large Scale Image Ontology." Vision Sciences Society, 2009.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:206594692

[17] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," 2020. [Online]. Available: https://arxiv.org/abs/1811.10959