Chapter 1

Background

1.1. 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.

At its core, weight space learning seeks to construct *meta-models*—models that learn from other models. Unlike standard machine learning models that capture patterns in data, meta-models capture patterns in the *weights* of networks trained on that data. In this way, the goal shifts from learning a direct input–output mapping to learning the structure that governs how such mappings are formed.

Due to the enormous scale and dimensionality of modern neural networks [], it is typically infeasible to operate directly on raw model weights. Furthermore, redundancy is well known to exist in neural networks; smaller architectures can often achieve comparable performance to larger ones []. These challenges motivate one of the central subproblems of weight space learning: the discovery of low-dimensional representations of weight space.

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, a useful distinction can be made between *reversible* and *non-reversible* methods.

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

This chapter proceeds as follows. Section 1.2 introduces Principal Component Analysis (PCA), a linear and probabilistic approach that provides a simple yet effective reversible dimensionality reduction method. Section 1.3 discusses reversible, non-linear approaches, focusing on autoregressive encoder architectures and presenting the Sequential Autoencoder for Neural Embeddings (SANE) []. Finally, Section 1.4 explores a non-reversible, modality-heterogeneous technique, contrastive learning, including a description of the NT-Xent loss and its implementation in CLIP [].

1.2. 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. This section outlines the motivation behind PCA, its mathematical formulation, and its relevance and limitations as a latent representation method.

Problem Setup and Intuition. 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.

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

$$Z = XW,$$

where $Z \in \mathbb{R}^{n \times k}$ represents the latent representation. The sample covariance matrix of X is

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

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.

Latent Representation and Limitations. The resulting embedding,

$$Z = XW_k,$$

is the *latent representation* of the data, capturing the dominant linear structure of the dataset. PCA provides a reversible mapping that probabilistically preserves the greatest possible amount of variance under a linear projection.

However, its linear nature limits its ability to capture nonlinear relationships when the data lie on curved manifolds. Furthermore, PCA does not adapt to unseen data that deviate from the original subspace—it yields a fixed, non-learnable transformation.

Incremental PCA. In practice [cite], the covariance matrix Σ can be prohibitively large for high-dimensional data, such as neural network weights. Incremental PCA (IPCA) addresses this by processing data in smaller batches and updating the principal components iteratively. Rather than recomputing the full covariance, it approximates it using partial updates and truncated singular value decompositions (SVD). This approach makes IPCA suitable for large-scale or streaming datasets while maintaining results comparable to standard PCA. However, it remains an approximation and inherits PCA's linear and non-generalisable nature.

1.3. Autoencoders

Autoencoders are neural network architectures designed for unsupervised representation learning. Their goal is to learn a compressed, informative latent representation of input data by training the network to reconstruct the original input after passing it through a lower-dimensional bottleneck. An Autoencoder's architecture is charactersized by a layer, with less than nodes than the dimension of the input data, then expanding the the size of the original data.

Architecture and Operation. 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:

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

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

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

Together, the encoder and decoder form a composite function:

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

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

Training Objective. 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):

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

This loss is automatically differentiated and optimised through backpropagation using gradient descent. By minimising this objective, the autoencoder learns weight configurations that best compress and reconstruct the data distribution.

The objective can be extended to include additional terms depending on the desired properties of the latent space. For example, a *contrastive loss* component can be incorporated to encourage semantic separation between latent representations, improving feature discriminability. Likewise, *regularisation* terms—such as L_1 or L_2 penalties, or sparsity constraints—can be added to improve generalisation and prevent overfitting by controlling the magnitude or distribution of learned weights.

Latent Representation and Flexibility. 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.

Autoencoders are highly flexible in architecture — they can be shallow for simple data or deep to capture hierarchical structure. Nonlinear activation functions such as ReLU, tanh, or sigmoid increase expressivity, allowing the model to represent complex, nonlinear manifolds within the data distribution.

1.3.1. 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 [1], where the goal was to build a continuous and interpretable representation space of neural weights.

Training Objective. 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:

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

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:

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

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 $\mathcal{L}_{\text{c}}$ 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 *NXTne* loss formulation (see Section 1.4.1), 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.

Interpretation and Utility. 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.

Sequential Autoencoder for Neural Embeddings. A major challenge in encoding neural network weights lies in the enormous number of parameters, which quickly becomes infeasible to process directly. [2] introduced the Sequential Autoencoder for Neural Embeddings (SANE), 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 [2].

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.

This flexibility marks an important step toward scalable and architecture-agnostic representation learning in weight space.

1.4. 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.

1.4.1. 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:

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

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, resulting in well-structured and discriminative latent representations.

1.4.2. CLIP

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:

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

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.

Bibliography

[1] 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

[2] 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.