OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Fraunhofer AISEC

On Homotopy of Similar Neural Networks using Non-linear Transformations

Ostbayerische Technische Hochschule Regensburg
Faculty of Computer Science and Mathematics

MASTER THESIS

Bettina Zieger

Student ID:	3401509
Study Programme:	Master Mathematics
Deadline:	31. July 2025
Supervisor:	Prof. Dr. Stefan Körkel
Secondary Supervisor:	Prof. Dr. Wolfgang Lauf
External Supervisor:	M. Sc. Daniel Kowatsch, Fraunhofer AISEC

Contents

1 Introduction

Neural networks are widely used in research, industry, and a broad range of application domains, including anomaly detection, image analysis, pattern recognition, and natural language processing [27]. Their popularity stems from their ability to approximate complex, non-linear functions [1], which enables them to model intricate patterns in high-dimensional data.

Given this flexibility, a key question in modern deep learning research is how different neural networks represent information internally, and how similar these representations are across different architectures, training procedures, or random initializations. Understanding these similarities is important not only for model interpretability but also for transfer learning, model selection, and robustness analysis.

To this end, various similarity measures have been proposed in the literature [20]. We can distinguish, for example, between *representational measure similarity*, where we calculate the difference in the activation of intermediate layers and the *functional measure similarity*, where we calculate the difference in the models output data.

Related Work. Most existing methods for comparing neural networks are based on linear transformations, such as permutations, orthogonal mappings, or affine projections [22, 35, 25]. These methods aim to identify structural correspondences between learned representations but are limited in expressiveness due to their linear nature.

A recent contribution by Chan et al. [2] introduces a homotopy-based framework for comparing language encoders via affine transformations. They formalize two types of similarity:

- *Intrinsic similarity*, based on how well one model's representation can be transformed into another's, and
- *Extrinsic similarity*, based on how well the output behavior can be matched via task-specific heads.

While this affine homotopy framework is elegant and mathematically grounded, it is inherently limited to linear transformations. Given the non-linear nature of modern neural networks, this restricts the ability to fully capture complex functional relationships between models.

Motivation and Research Questions. This thesis extends a framework based on homotopy by incorporating *nonlinear* transformations subject to a 1-Lipschitz constraint.

Our goal is to explore whether this generalization leads to more expressive and informative measures of similarity between language encoders.

Specifically, we address the following research questions:

- How are intrinsic and extrinsic homotopy defined for nonlinear transformations?
- Do non-linear transformations yield more accurate or meaningful similarity measures than affine ones?
- Is there a consistent relationship between intrinsic and extrinsic homotopy in the non-linear setting?
- Do similar encoders also show similar performance and training behavior?

By investigating these questions, we aim to contribute to a deeper understanding of model similarity and functional alignment in modern neural architectures.

To address the research questions, we begin in Chapter 2 by formally introducing neural networks. We define key components such as activation functions, loss functions, and training procedures, and provide an overview of common architectures.

Building on this foundation, Chapter 3 focuses on Natural Language Processing. We introduce the concept of language encoders and present an overview of state-of-the-art transformer models and their applications.

Chapter 4 introduces mathematical basics such as norms, distances, and homotopy.

Chapter 5 addresses the topic of similarity between neural networks. We distinguish between representational and functional similarity measures.

In Chapter 6, we describe the existing affine homotopy approach proposed by Chan et al. [2], and explain and explain how it measures similarity between language encoders.

Chapter 7 presents our novel extension of this framework, in which we use non-linear transformations to obtain a more expressive and flexible notion of model similarity.

In Chapter 8, we describe the practical implementation of this extended framework, including details on model design, training procedure, and evaluation metrics.

To validate our approach, Chapter 9 presents a series of experiments using three language encoders with identical architecture to BERT-BASE [7]. We evaluate the effectiveness of our method across multiple tasks and discuss the experimental findings.

Finally, Chapter 10 concludes the thesis by summarizing the main contributions and insights, identify current limitations, and suggest directions for future research.

The language model ChatGPT (OpenAI) was used to support the linguistic revision of individual text passages in order to make formulations more precise and improve the style.

2 Mathematical Formalizations of Neural Networks

The technique of Machine Learning (ML), which improves the performance of a system by using computational methods, is used to learn from the data. We denote the process of using ML algorithms to build models from data as training or learning. By feeding the learning algorithm with experience data, we obtain a model that can make predictions for new observations [37]. Inspired by the human brain, artificial neural networks are developed and have gained a lot of popularity in different fields like Imaging, Generative Artificial Intelligence and Natural Language Processing (NLP).

To provide the necessary foundations for this work, we begin with an overview of fundamental concepts in machine learning in Section 2.1. This sets the stage for Section 2.2, where we introduce neural networks as a central model class, together with commonly used activation and loss functions. Building on these components, we then formalize the training process and explain how neural networks learn from data. Finally, in Section 2.3, we present different types of neural networks that are relevant for the analysis in later chapters.

2.1 Overview of Machine Learning

Depending on how ML methods perform the prediction task, there are three main types of ML. We distinguish between supervised, unsupervised, and reinforcement learning, which are introduced in the following:

1. **Supervised Learning** uses a training set that contains input data along with the corresponding outputs. These known outputs provide the information needed for learning, and are commonly referred to as labels [18]. Depending on whether the outputs are discrete or continuous, the task is called a **classification** or **regression problem**, respectively [37].

2. **Unsupervised Learning**: uses input data without any known output values. The goal is to discover hidden patterns or structures in the data without relying on predefined target information. A common example of unsupervised learning is *clustering*, where the model groups similar data points based on their features [16].

3. **Self-Supervised Learning** is a subcategory of unsupervised learning in which the system generates its own supervisory signal from the input data [17]. Instead of relying on human-annotated labels, the model formulates auxiliary tasks, called *pretext tasks*, whose targets can be derived automatically. A prominent example is *masked language modeling*, where certain tokens in a sentence are masked and the model is trained to predict the missing parts. This training paradigm plays a central role in transformer-based language models and will be revisited in later chapters.
4. **Reinforcement Learning** ist a paradigm of ML in which an *agent* learns to make decisions by interacting with an *environment*. At each time step, the agent observes the current state of the environment, takes an action, and receives a *reward* as feedback. The reward is computed based on the new state of the environment and reflects the quality of the action taken [37, 10]. Over time, the agent adapts its behavior in order to maximize the cumulative reward and solve the task at hand.

2.2 Neural Networks

Neural networks, as outlined in this chapter's structure, are central to modern machine learning. Before discussing specific architectures or training procedures, we first provide a structured overview of neural networks as function approximators. This includes their basic components—neurons and layers—and the way they are composed into deep architectures. The goal is to provide an intuitive understanding of how they operate as function approximators by passing data through layers of interconnected units. This intuition is supported by the *Universal Approximation Theorem*, which states that even a feedforward network with a single hidden layyer can napproximate any continuous function on a compact subset of $\mathbb{R}^n$. After this conceptual overview, we discuss common activation and loss functions, formalize the training process, and conclude the chapter with an overview of different neural network architectures.

In supervised learning, we are given a dataset $\mathcal{D}$ of N input-output pairs

$$\mathcal{D} := \{(x_i, y_i)\}_{i=1}^{N} \subset \mathbb{R}^{D^{(0)}} \times \mathbb{R}^{D^{(L)}},$$

where each input $x_i \in \mathbb{R}^{D^{(0)}}$ represents a data point with $D^{(0)}$ features, and each $y_i \in \mathbb{R}^{D^{(L)}}$ is the corresponding target or label. A neural network defines a function

$$f : \mathbb{R}^{D^{(0)}} \to \mathbb{R}^{D^{(L)}}$$

that aims to approximate the relationship between inputs and outputs.

This function f is represented, as in Cotterell et al. [5], as a composition of layer-wise transformations:

$$f = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}.$$

Each function $f^{(\ell)} : \mathbb{R}^{D^{(\ell-1)}} \to \mathbb{R}^{D^{(\ell)}}$, for $\ell = 1, \ldots, L$, represents a single layer of the network, where $D^{(\ell)}$ denotes the number of neurons in layer ℓ. Each layer consists of

a linear transformation followed by a non-linear activation function. The architecture of the network is defined by the number of layers, their types and their respective with, i.e, the dimension $\mathbb{R}^{D^{(0)}}, ..., \mathbb{R}^{D^{(L)}}$

The final layer $f^{(L)}$ produces the network's output, so that the entire network is defined as a composition:

$$f(x) := f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(x).$$

The total number of layers L is called the *depth* of the network.

Given an input $x_i \in \mathbb{R}^{D^{(0)}}$, the output of the network is the prediction

$$\hat{y}_i := f(x_i), \quad i = 1, \ldots, N.$$

A neural network is composed of the following types of layers, this is illustrated in Figure 2.1:

- **Input layer** represents the feature vector $x \in \mathbb{R}^{D^{(0)}}$ and serves as the entry point of the network. It does not perform any computation itself.
- **Hidden layers** are responsible for transforming representations through combinations of parameterized operations and nonlinearities, enabling the network to learn complex functions from data.
- **Output layer** produces the final prediction $f(x) \in \mathbb{R}^{D^{(L)}}$

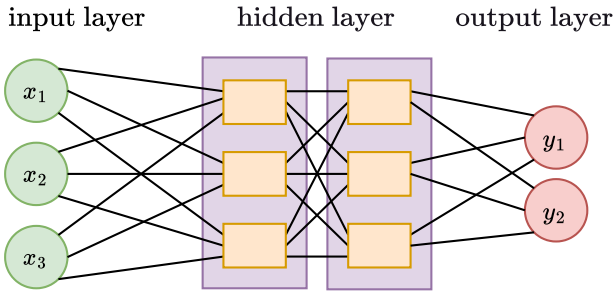


Figure 2.1: Illustration of a neural network. Adapted from [37].

To understand the functional building blocks of neural networks, we now examine the internal structure of a single layer. As illustrated in Figure 2.2, a layer consists of multiple neurons that receive the same input vector $x \in \mathbb{R}^{D^{(0)}}$, but apply different parameters. If the layer comprises $D^{(L)}$ neurons, that is, it produces an output vector of dimension $D^{(L)}$, its parameters consist of a weight matrix $W \in \mathbb{R}^{D^{(L)} \times D^{(0)}}$ and a bias vector $b \in \mathbb{R}^{D^{(L)}}$.

The output of the layer is a vector $y \in \mathbb{R}^{D^{(L)}}$, computed as

$$z := Wx + b, \quad y := \sigma(z),$$

where σ denotes the activation function, which is applied element-wise.

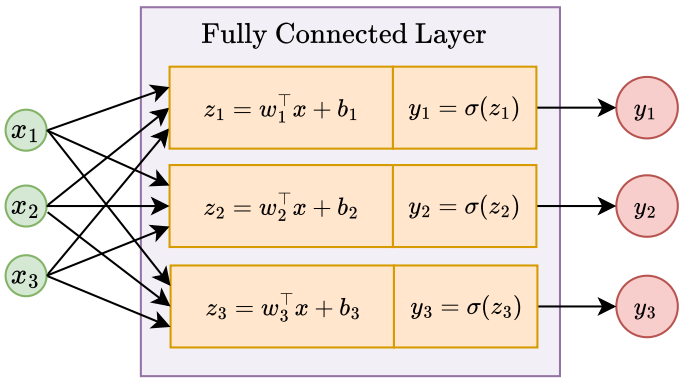


Figure 2.2: Illustration of a neural networks layer

At the core of each layer is the artificial neuron. It performs a simple tow-step operation:

1. Compute an affine transormation of the input,
2. Apply a nonliear activation function.

2.2.1 Activation Functions

In the previous section (Section 2.2), we saw that a model a fully connected layer computes outputs via the quation:

$$y = \sigma(Wx + b),$$

where $W \in \mathbb{R}^{m \times n}$ is the weight matrix, $b \in \mathbb{R}^m$ is a bias vector, and σ is a non-linear activation function.

If we don't consider the activation function, we only have an affine transformation $Wx + b$, resulting in a purely linear model.While such models are useful in simple settings (e.g. linear regression), they are limited in ther expressive power. A composition of linear functions is still linear, which means that even deep networks without non-linearities cannot represent complex patterns or decision boundaries.

The inclusion of non-linear activation functions, particularly continuous sigmoidal functions endows neural networks with the expressive capacity to approximate a wide range of functions. In fact, the Universal Approximation Theorem states that a feedforward neural network with a single hidden layer and a continuous sigmoidal activation function can uniformly approximate any continuous function defined on a compact domain, such as the unit hypercube $[0, 1]^n$ [6].

Activation functions define the function space that a neural network can represent. Their properties influence not only the expressive capacity of the network, but also its trainability, optimization dynamics, and numerical stability.

In the following, we discuss several commonly used activation functions as presented in Stevens et al. [1], including Heaviside's threshold function, the Rectified Linear Unit (ReLU), the sigmoid function, and the hyperbolic tangent. These functions are illustrated in Figure 2.3.

(a) **Heaviside function**

- is a historically significant choice of activation function
- $\sigma(x) = H(x) := \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$

(b) **Rectified Linear Units (RELU)**

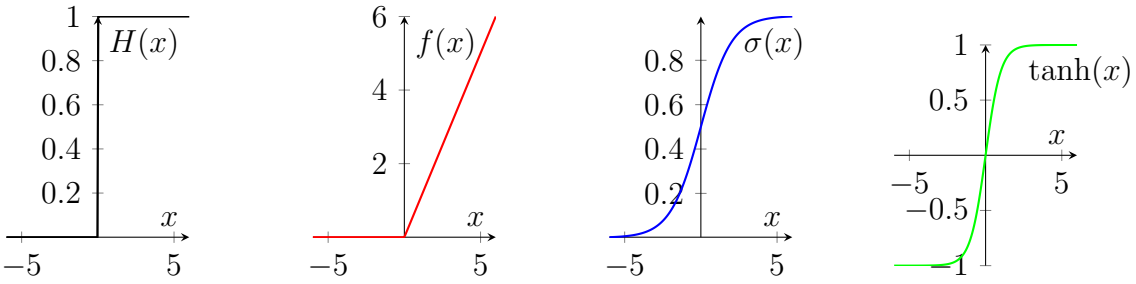
- one of the most used activation function in modern neural networks
- $\sigma(x) = \text{ReLU}(x) := \max\{0, x\}$

(c) **Sigmoid**

- commonly used activation function in early neural networks
- reason that activation functions are often labelled with σ
- $\sigma(x) = \text{Sigmoid}(x) := \frac{1}{1+e^{-x}}$

(d) **Hyperbolic tangent**

- similar to the sigmoid
- $\sigma(x) = \tanh(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}}$



(a) Heaviside **(b)** ReLU **(c)** Sigmoid **(d)** Tanh

Figure 2.3: Activation functions

There exists some common multivariate functions as Softmax and Maxpool, too. His multivariate functions operate on vectors and tensor and are commonly used to transform a vector of real-values scores into a probability distribution over classes.

(a) **Softmax**

- For $\lambda > 0$, the scaled softmax is defined as

$$\mathrm{softmax}_\lambda : \mathbb{R}^n \to [0,1]^n, \quad \mathrm{softmax}_\lambda(x)_i = \frac{e^{\lambda x_i}}{\sum_{j=1}^{n} e^{\lambda x_j}}, \quad \forall i \in \{1,\dots,n\}.$$

(b) **Maxpool operation**

- The maxpool function is defined as

$$\mathrm{maxpool} : \mathbb{R}^n \to \mathbb{R}^m, \qquad \mathrm{maxpool}(x)_i = \max_{j \in I_i} x_j \quad \text{for all } i \in \{1,\dots,m\},$$

  where $I_i \subset \{1,\dots,n\}$ defines the pooling region.

2.2.2 Training Process

After introducing layers and non-linear activation functions, we now have all required components for a neural network to approximate complex functions. However, in order to fit an actually given task such as mapping inputs to outputs, we shall need to determine appropriate values for the networks weights and biases. This is where we use the trainings process. The goal of training process is to adjust the parameters of a neural network that explains the best relationship between the given inputs and outputs. This is achieved by computing the error between the networks output and the target values using a loss function $\mathcal{L}$ and then propagating this error backwards through the network to update the parameters, this process is known as backpropagation. During backpropagation, the network applies the chain rule to pass the error from the output layer back to the earlier layers, adjusting weights and biases along the way in order to minimize the loss.

A loss function $\mathcal{L}$ measures the discrepancy between the network's predicted output and the true target value. Its purpose is to guide the training process by quantifying how wrong the network's predictions are.

It measures the loss of the model and will be minimized during the training process. Common choices of loss functions, as presented in [37], include:

(a) **Squared Error Loss**

- Measures the squared difference between predicted and target values.
- Often used in regression settings, but can also be applied to compare continuous representations in higher-dimensional spaces.

- Defined as

$$\mathcal{L}_{\text{SquareErr}}(y, \hat{y}) = \sum_{i=1}^{n} \|y_i - \hat{y}_i\|^2.$$

(b) **Cross Entropy Loss**

- Used for classification tasks involving discrete probability distributions.
- Let $p = (p_1, \ldots, p_k)$ and $q = (q_1, \ldots, q_k)$ be discrete probability distributions over k classes.
- Defined as

$$\mathcal{L}_{\text{CrossEnt}}(p, q) = -\sum_{i=1}^{k} p_i \ln q_i.$$

(c) **Huber Loss**

- Combines properties of squared error loss and absolute error loss..
- Less sensitive to outliers than squared error loss.
- Introduced by Peter J. Huber in his foundational work on robust statistics [12]
- Defined for a threshold $\delta > 0$ as

$$\mathcal{L}_{\text{Huber}}(y, \hat{y}) = \sum_{i=1}^{n} \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta, \\ \delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right) & \text{otherwise.} \end{cases}$$

We now formalize the training task. Let $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^{n}$ be a dataset of input-output pairs, where x_i are the inputs and y_i the corresponding labels. Let $f(x, \theta)$ be a neural network parameterized by θ that maps inputs to outputs. We define the set of all trainable parameters, i.e., the weight matrices and bias vectors of each layer $l = 1, \ldots, L$, as

$$\theta := \{(W^{(l)}, b^{(l)}) \mid W^{(l)} \in \mathbb{R}^{m_l \times n_l},\ b^{(l)} \in \mathbb{R}^{m_l}\}.$$

Following Smets [31], we define the sample-wise loss as

$$\tilde{f}_i(\theta) := \mathcal{L}(y_i, f(x_i, \theta)) \quad \text{for } i = 1, \ldots, n,$$

and define the overall training objective as minimizing the average loss across the dataset:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \tilde{f}_i(\theta).$$

To solve this optimization problem, gradient-based methods such as *stochastic gradient descent* (SGD) are commonly used. These methods update the model parameters iteratively in the direction that reduces the loss.

Concretely, the parameters θ are updated via the rule

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \left(\frac{1}{n} \sum_{i=1}^{n} \tilde{f}_i(\theta) \right),$$

where $\eta > 0$ is the learning rate that controls the step size. The required gradients $\nabla_\theta \mathcal{L}(y_i, f(x_i, \theta))$ are efficiently computed using *backpropagation*, which applies the chain rule to propagate derivatives through the network layers. In classical *gradient descent*, the full gradient over the entire dataset is used in each step. This becomes computationally expensive for large datasets. In contrast, *stochastic gradient descent (SGD)* approximates this full gradient by computing it on small random subsets of the data, called mini-batches. This introduces some noise into the updates but allows for faster and more scalable training.

One widely used variant of SGD is the *Adam optimizer* [19], which adaptively adjusts the learning rate for each parameter based on estimates of the first and second moments of the gradients. Adam combines the benefits of two earlier methods, and often leads to faster convergence and more stable training, especially in deep networks.

The following code snippet 2.1 shows the training routine of a neural network used to learn a nonlinear transformation for intrinsic homotopy, which will be discussed in detail in Chapter 7.

Code 2.1: Code snippet showing how to train a neural network

```
import torch
import torch.nn as nn
from torch.nn.utils import spectral_norm
from torch.utils.data import DataLoader, TensorDataset

class NonLinearNetwork(nn.Module):
        """A simple non-linear neural network with:
        - input layer, one hidden layer (with ReLU activation),
           output layer, spectral normalization for training
           stability"""
        def __init__(self, in_dim, out_dim, hidden_dim=128):
                super().__init__()
                self.net = nn.Sequential(
                spectral_norm(nn.Linear(in_dim, hidden_dim),
                   n_power_iterations=20),
                nn.ReLU(),
                nn.Linear(hidden_dim, out_dim)
                )

        def forward(self, x):
                return self.net(x)

def train_neural_network(X_train, Y_train, device, epochs=100, lr=1e
   -3):
        """Trains simple neural network to learn mapping X to Y."""
```

```
    model = NonLinearNetwork(X_train.shape[-1], Y_train.shape
        [-1]).to(device) # Create the network and move it to the
        CPU or GPU
    opt = torch.optim.Adam(model.parameters(), lr=lr) # Define
        the optimizer (Adam adjusts the weights during training)
    # Mean Squared Error (MSE) measures how close the network's
        output is to the target
    loss_fn = nn.MSELoss(reduction='none')  # Sample-wise loss

    loader = DataLoader(
                    TensorDataset(torch.tensor(X_train, dtype=
                        torch.float32),
                            torch.tensor(Y_train, dtype=torch.
                                float32)),
                            batch_size=128, shuffle=True
                            )

    # Loop over the training process for the number of epochs
    for epoch in range(epochs):
            model.train()
            total_loss, max_loss = 0.0, 0.0

            for x_batch, y_batch in loader:
                    x_batch, y_batch = x_batch.to(device),
                        y_batch.to(device)
                            opt.zero_grad() # Reset gradients
                                from previous step
                    output = model(x_batch) # Run input through
                        the network
                    loss = loss_fn(output, y_batch) # Compute
                        squared error
                    sample_loss = loss.mean(dim=1)
                    batch_loss = sample_loss.mean()
                    batch_loss.backward()    # Backpropagation:
                        compute gradients
                    opt.step()      # Update model weights

                    total_loss += batch_loss.item()
                    max_loss = max(max_loss, sample_loss.max().
                        item())

            print(f"Epoch {epoch+1}/{epochs} | Avg Loss: {
                total_loss:.4f} | Max Sample Loss: {max_loss:.4f}"
                )

    return model
```

2.3 Types of Neural Networks

In the previous sections, we discussed artificial neural networks in general. There exist various types of neural networks, distinguished by their architecture and typical use cases. The following list gives a brief overview of some common types, as presented by Goyal et al. [15]:

(a) **Feedforward Neural Networks (FNNs)** are the simplest type of neural network. Data flows in one direction—from the input layer, through one or more hidden layers, to the output layer—without cycles or loops.

(b) **Convolutional Neural Networks (CNNs)** are commonly used for image and handwriting recognition. They apply convolutional filters to local regions of the input to extract features, which are then used to build hierarchical representations.

(c) **Recurrent Neural Networks (RNNs)** are designed to process sequential data. Each time step receives an input and the output of the previous step, allowing the network to retain a form of memory. This makes RNNs particularly suitable for time series, language, and other temporally dependent data.

(d) **Transformers** [32] are modern neural network architectures that rely entirely on attention mechanisms rather than recurrence or convolution. They allow parallel processing of sequences and have achieved state-of-the-art performance in natural language processing tasks such as translation, summarization, and question answering.

3 Natural Language Processing

In this thesis, we focus on the similarity analysis of language encoders, which are central components of modern NLP systems. Language encoders transform raw text into structured vector representations that can be processed by machine learning models. Since our approach is based on comparing and transforming these encoders, they form a foundational element of this work.

Natural Language Processing (NLP) enables machines to interpret, process, and generate human language. Unlike structured data, natural languages present a wide variety of problems that vary from language to language. Structuring or extracting meaningful information from free text represents a great solution, if it is done in the right manner. Historically, NLP relied on rule-based systems that analyzed grammatical structure using syntactic parsers and manually designed algorithms. These approaches were often limited in scalability and adaptability across different languages and contexts. Previously, computer scientists broke a language into its grammatical forms, such as parts of speech, phrases, etc., using complex algorithms. With the rise of deep learning, data-driven methods have largely replaced rule-based systems. Modern NLP models can learn complex patterns directly from text corpora and perform tasks such as machine translation, text summarization, question answering, and speech recognition with remarkable accuracy [15]. This chapter provides a technical introduction to the foundations of NLP. Section 3.1 introduces the concept of language encoders, which transform text into structured vector representations. Section 3.2 then presents transformer-based models, the current state-of-the-art architecture in NLP.

3.1 Language Modeling

In this section, we introduce the mathematical building blocks of modern language models. Intuitively, a *language encoder* is a function that transforms text into numbers. This is necessary because computers cannot directly work with text, they need numerical representations.

To do so, we use Cotterells et al. publication [5] as baseline and define the important tools in language modelling if not referenced differently as they did.

Definition 3.1 (Language Encoder)**.** *A **language encoder** is a function*

$$h : \Sigma^* \to V$$

that maps text strings to vectors of real value. Here, Σ^* denotes the set of all possible sequences of characters from a finite alphabet Σ, and $V = \mathbb{R}^D$ is a D-dimensional real vector space.

For example, the sentence “language is powerful” might be mapped to a 768-dimensional vector such as $(0.1, -0.3, \ldots, 0.2)$.

We define:

- $\mathcal{E}_V := V^{\Sigma^*}$: the set of all such encoders.
- $\mathcal{E}_b := \{h \in \mathcal{E}_V \mid h(\Sigma^*)$ is bounded$\}$: the subset of encoders that do not produce arbitrarily large vectors.

A **language model** is a probability distribution over all possible texts. It allows us to answer questions like: “How likely is this sentence?” or “What word should come next?”

Definition 3.2 (Language Model)**.** *A **language model** is a probability distribution*

$$p_{\mathrm{LM}} : \Sigma^* \to [0, 1]$$

that assigns probabilities to strings over the alphabet Σ.

A common type of language model is the *autoregressive model*, which predicts the next word in a sentence one step at a time. This process stops when a special end-of-sequence symbol (`EOS`) is generated.

Definition 3.3 (Autoregressive Language Model)**.** *Given a sequence $y = (y_1, \ldots, y_T)$, an autoregressive model defines:*

$$p_h^{\mathrm{LM}}(y) = p_h(EOS \mid y) \cdot \prod_{t=1}^{T} p_h(y_t \mid y_{<t})$$

The conditional probabilities are parametrized using a language encoder h as:

$$p_h(y_t \mid y_{<t}) = \mathrm{softmax}(Eh(y_{<t}))_{y_t}$$

where $E \in \mathbb{R}^{(|\Sigma|+1)\times D}$ is a learned embedding matrix.

This formulation is used in models like GPT.

An alternative training strategy is *masked language modeling*, where some words in a sentence are replaced with a special token like `[MASK]` and the model must guess them.

Definition 3.4 (Masked Language Model)**.** *Let $y = (y_1, \ldots, y_T)$ be a sequence of tokens from a vocabulary $\mathcal{V}$, and let $t \in \{1, \ldots, T\}$ be a randomly selected position to mask. Let $y_{\setminus t} := (y_1, \ldots, y_{t-1}, \texttt{[MASK]}, y_{t+1}, \ldots, y_T)$ denote the input sequence with the t-th token replaced by a special* `[MASK]` *token. A **masked language model (MLM)** estimates the conditional probability*

$$p_h(y_t \mid y_{\setminus t}) = \mathrm{softmax}(E\, h(y_{\setminus t}))_{y_t},$$

where:

- *$h \colon \mathcal{V}^T \to \mathbb{R}^D$ is a language encoder that maps the masked sequence $y_{\setminus t}$ to a hidden representation;*
- *$E \in \mathbb{R}^{|\mathcal{V}| \times D}$ is a learned projection from the hidden space to vocabulary logits;*
- *The softmax is applied on the vector $z := E\, h(y_{\setminus t}) \in \mathbb{R}^{|\mathcal{V}|}$:*

$$\mathrm{softmax}(z)_v := \frac{\exp(z_v)}{\sum_{v' \in \mathcal{V}} \exp(z_{v'})} \quad \textit{for } v \in \mathcal{V}.$$

Example:

```
The students [MASK] to learn about language models.
```

The model should assign high probabilities to words like *want* or *like*. This task encourages the model to understand both the left and right context of a word.

3.2 Transformer Models

Transformer models have become the foundation of modern NLP due to their ability to capture complex relationships between words — even when those words are far apart in a sentence. Unlike older models such as Recurrent Neural Networks (RNN), which process text sequentially and struggle with long-range dependencies, Transformers process entire sequences in parallel. Their core innovation is the **self-attention** mechanism, which allows the model to weigh the relevance of each word with respect to others in the sequence.

A Transformer is a deep neural network architecture introduced by Vaswani et al. [32]. It has since become the basis of widely-used models such as BERT and GPT, which are trained on large corpora in a self-supervised manner, that is, without labeled data.

Transformers are primarily composed of an encoder and a decoder block, as illustrated in Figure 3.1. The encoder processes the input sequence and generates contextual representations. The decoder then uses these representations, along with previously generated tokens, to produce the output sequence. This architecture enables applications such as machine translation, summarization, and text generation.

A key strength of Transformer models is their self-attention mechanism, which enables them to weigh different parts of the input when making predictions.

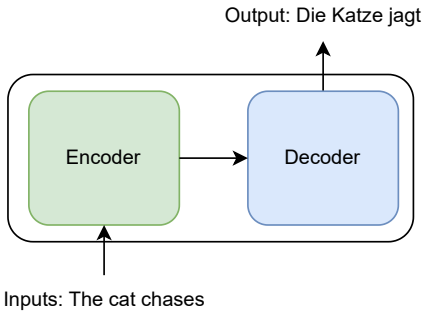


Figure 3.1: Encoder-decoder structure of a Transformer.

For example, in the sentence: *"The cat chased the mouse because she was hungry."* a Transformer can learn that "she" refers to "the cat" by attending to all words in the sequence at once.

Two of the most influential Transformer-based language models are **BERT** and **GPT**, which differ in their architecture and training objective.

BERT – Bidirectional Encoder Representations from Transformers BERT [7] is an *encoder-only* model. It processes the entire input sentence at once using bidirectional self-attention and is designed to learn contextual representations from unlabeled text through a masked language modeling objective.

Unlike earlier models [28, 29], BERT's attention layers can access all tokens in the sequence during training. This allows it to capture both left and right context simultaneously. Models of this type are referred to as *auto-encoding models*. Thanks to its pre-training on large corpora and its architecture, BERT can be fine-tuned with minimal task-specific modifications for a wide range of NLP tasks, including sentence classification, named entity recognition, and extractive question answering.

GPT – Generative Pre-Training GPT [29] is a *decoder-only* model. It uses unidirectional (left-to-right) attention and is trained to predict the next word in a sentence, making it well suited for text generation tasks.

GPT follows a two-stage training approach:

- **Unsupervised pre-training** on raw text to learn general-purpose language representations.
- **Supervised fine-tuning** on task-specific labeled data.

Because GPT can only attend to preceding tokens at each position, it is called an *auto-regressive model*.

4 Preliminaries

In order to formally develop the concepts of intrinsic and extrinsic homotopy for language encoders, we first need some basic mathematical notions. The aim of this chapter is to provide a theoretical framework in which encoders can be compared with respect to their similarity using transformations. To this end, we first introduce *hemi-metric spaces* in Section 4.1 , which enable an asymmetric distance function between language encoders. This structure allows us to measure similarities without necessarily requiring symmetry or the triangle inequality, properties that are often not given in practical comparison scenarios. Based on this, we define the norms and distances in Section 4.2 in the space of language encoders. These measures are crucial for mathematically quantifying differences between encoders and analyzing transformations. A central concept, the so-called *Specialisation Quasi-Ordering*, which is a special form of preordering derived from the hemi-metric structure, is presented in Section 4.3. It describes whether one encoder can be completely transformed into another one. Finally, we motivate *homotopy* in Section 4.4 to measure similarity.

4.1 Hemi-Metric Spaces

In the following, we will use the definition of metrices and a hemi-metric as tom Dieck [8] defined. Further, we consider Metric Spaces (X, d), that is, a set X and a metric d operation on X. A metric on a set X is a mapping $d : X \times X \to [0, \infty[$ that satisfies the known axioms:

M1 $d(x, y) > 0$ and $d(x, y) = 0 \Leftrightarrow x = y$

M2 $d(x, y) = d(y, x) \quad \forall x, y \in X$

M3 $d(x, z) \leq d(x, y) + d(y, z) \quad \forall x, y, z \in X$

If there is a map $d : X \to \overline{\mathbb{R}}_+$, that means that the metric can be equal to ∞ and we denote it as an **extended metric**. A *hemi-metric* or *Lawvere metric* [24] is a generalized distance function that relaxes the symmetry requirement of a metric.

Definition 4.1 (Hemi-metric / Lawvere metric)**.** *Let X be a set. A function $d\colon X \times X \to \overline{\mathbb{R}}_+$ is called a* hemi-metric *(or* Lawvere metric *[24]) if it satisfies the following two axioms:*

H1 $d(x, x) = 0$ *for all* $x \in X$*,*

H2 $d(x, z) \leq d(x, y) + d(y, z)$ *for all* $x, y, z \in X$*.*

Note that symmetry is not required, i.e., it may hold that $d(x, y) \neq d(y, x)$. *If the codomain includes* ∞*, we call* d *an* extended hemi-metric*.*

This relaxation allows us to define an asymmetric notion of distance between representations. In particular, we can formalize how far a language encoder h is from the set of all possible transformations of another encoder g. To this end, the hemi-metric is lifted from individual elements $x \in X$ to subsets of X.

Definition 4.2 (Hausdorff-Hoare map)**.** *Let* (X, d) *be a hemi-metric space, and let* $E, E' \subset X$ *be non-empty subsets. The* Hausdorff-Hoare map *is the function*

$$d^{\mathcal{H}}(E, E') := \sup_{x \in E} \inf_{y \in E'} d(x, y),$$

that defines a hemi-metric on $\mathcal{P}(X) \setminus \{\emptyset\}$*.*

If E is considered as a singleton set $\{x\}$, we will write $d^{\mathcal{H}}(x, E')$ instead of $d^{\mathcal{H}}(\{x\}, E') := \inf_{g \in E'} d(x, y)$

If a set X does not itself carry a hemi-metric, but can be embedded into the power set of a second space Y that does, we can induce a hemi-metric on X via this embedding. Specifically, let $\varphi : X \to \mathcal{P}(Y) \setminus \{\emptyset\}$ be an embedding of X into non-empty subsets of Y. Then a hemi-metric on X can be defined by

$$d_X(x, x') := d^{\mathcal{H}}(\varphi(x), \varphi(x')),$$

where $d^{\mathcal{H}}$ is the Hausdorff–Hoare map induced by the hemi-metric on Y.

4.2 Norm and Distances

As is well known, every norm on a real vector space induces a metric via $d(x, y) := \|x - y\|$. Since all norms on a finite-dimensional $\mathbb{R}$-vector space V are equivalent [23], we will work with a fixed norm $|\cdot|_V$ on V in the following.

We further define the extended norm

$$\|h\|_\infty := \sup_{y \in \Sigma^*} |h(y)|_V,$$

and the induced metric

$$d_\infty(h, g) := \|h - g\|_\infty$$

on the space $\mathcal{E}_V$, which consists of all functions $h : \Sigma^* \to V$ with finite $\|\cdot\|_\infty$-norm.

Equipped with this metric, $(\mathcal{E}_V, d_\infty)$ forms a complete metric space, meaning that every Cauchy sequence in $\mathcal{E}_V$ converges to a limit within the space. This follows from the fact that V is a finite-dimensional normed space and hence complete. This property will later be used to argue the existence of certain infima in $\mathcal{E}_V$.

For the subordinate matrix norm we write $\|\cdot\|_V : GL(V) \to \mathbb{R}_+$, where $GL(V)$ denotes the set of invertible $D \times D$ matrices. It is defined as

$$\|W\|_V = \sup_{v \in V \setminus \{0\}} \frac{|Wv|_V}{|v|_V}.$$

We consider V as an affine space and set $\mathsf{Aff}(V)$ for the group of affine transformations of V, by abuse of language, if we forget about the special role that the zero vector has.

A map $v \mapsto Wv + b$ is called an affine transformation ψ on V for $W \in GL(V)$ invertible and $b \in V$. The transformation consists of a linear part $\psi_{lin} := W$ and a translation part $\psi_t : v \mapsto v + b$. Further, $\mathcal{T} \subset \mathsf{Aff}(V)$ denotes the subgroup of translation. There is also a natural left action on $\mathsf{Aff}(V)$ on $\mathcal{E}_V$. I.e. $\mathsf{Aff}(V) \times \mathcal{E}_V \to \mathcal{E}_V, h \mapsto \psi \circ h$, such that there exists an identity element e on $\mathsf{Aff}(V)$, where $e \cdot h = h \quad \forall h \in \mathcal{E}_V$ and $\psi_1 \cdot (\psi_2 \cdot h) = (\psi_1 \psi_2) \cdot h \quad \forall \psi_1, \psi_2 \in \mathsf{Aff}(V)$ and $h \in \mathcal{E}_V$.

4.3 Specialization Quasi-ordering and Pre-ordering

In this section, we formalize the notion of specialization quasi-ordering and show that it induces a preorder. This structure will be used to define various homotopies on language encoders in later sections. We follow the exposition of Goubault-Larrecq in [14].

We begin with a brief reminder of the concept of a topological space. A *topological space* is a set X equipped with a collection of subsets, called *open sets*, that satisfies the following properties:

- The empty set $\emptyset$ and the full set X are open.
- Arbitrary unions of open sets are open.
- Finite intersections of open sets are open.

This collection of open sets defines the topology on X.

Definition 4.3 (Base)**.** *Let X be a topological space, and $\mathcal{B}$ be a family of open subsets of X. If every open set is a union of elements of $\mathcal{B}$, $\mathcal{B}$ is a* base *of the topology.*

Lemma 4.4 (Characterization of a Basis)**.** *A family $\mathcal{B}$ of subsets of a topological space X is a basis of the topology if and only if for every open set $U \subseteq X$ and every point $x \in U$, there exists an element $V \in \mathcal{B}$ such that $x \in V \subseteq U$.*

Proof. Let U be open and $x \in U$. By assumption, there exists $V_x \in \mathcal{B}$ with $x \in V_x \subseteq U$. Using the Axiom of Choice, pick such a V_x for each $x \in U$, and write $U = \bigcup_{x \in U} V_x$.

Suppose $\mathcal{B}$ is a basis, and let $U = \bigcup_{i \in I} V_i$ with $V_i \in \mathcal{B}$. For any $x \in U$, there exists some $i \in I$ such that $x \in V_i$. Let $V := V_i$, then $x \in V \subseteq U$. □

Definition 4.5 (Specialization Quasi-ordering)**.** *Let X be a topological space. The specialization quasi-ordering on X is defined by:*

$$x \gtrsim y \quad \textit{if and only if} \quad \textit{every open set containing } x \textit{ also contains } y.$$

This quasi-ordering induces a preorder on X, as stated in the following lemma.

Lemma 4.6 (Preorder)**.** *The specialization quasi-ordering is a preorder; that is, it is reflexive and transitive.*

Proof. Let $\gtrsim$ denote the specialization quasi-ordering.

Reflexivity: For any $x \in X$, every open set containing x trivially contains x itself, so $x \gtrsim x$.

Transitivity: Suppose $x \gtrsim y$ and $y \gtrsim z$. Let U be an open set containing x. Then, by definition of $\gtrsim$, $y \in U$, and hence $z \in U$. Therefore, $x \gtrsim z$. □

Now consider a hemi-metric d on a set X. We define open balls by

$$B(x, \varepsilon) := \{z \in X \mid d(x, z) < \varepsilon\},$$

where $x \in X$ is the center and $\varepsilon \in \mathbb{R}^+$ the radius. The topology generated by such open balls is referred to as the *open ball topology*.

Lemma 4.7 (Open Balls Form a Base)**.** *Let (X, d) be a hemi-metric space. Then, for every open set U containing $x \in U \subset X$, there exists $\varepsilon > 0$ such that*

$$B(x, \varepsilon) \subset U.$$

Proof. Let U be an open set containing $x \in X$. By the definition of the topology induced by d, there exists a finite collection of open balls $B(x_i, \varepsilon_i)$, $1 \leq i \leq n$, such that

$$x \in \bigcap_{i=1}^{n} B(x_i, \varepsilon_i) \subset U.$$

We show that there exists an open ball $B(x, \varepsilon)$ centered at x such that

$$B(x, \varepsilon) \subseteq \bigcap_{i=1}^{n} B(x_i, \varepsilon_i) \subset U.$$

Define

$$\varepsilon := \min_{1 \leq i \leq n} \left(\varepsilon_i - d(x_i, x)\right).$$

Since $x \in B(x_i, \varepsilon_i)$ for each i, we have $d(x_i, x) < \varepsilon_i$, and thus $\varepsilon > 0$.

Now take any $y \in B(x, \varepsilon)$, i.e., $d(x, y) < \varepsilon$. By the (possibly asymmetric) triangle inequality for hemi-metrics, we get

$$d(x_i, y) \leq d(x_i, x) + d(x, y) < d(x_i, x) + \varepsilon \leq \varepsilon_i,$$

so $y \in B(x_i, \varepsilon_i)$ for all i. Hence,

$$B(x, \varepsilon) \subseteq \bigcap_{i=1}^{n} B(x_i, \varepsilon_i) \subset U.$$

Therefore, for every open set U and every point $x \in U$, there exists an open ball $B(x, \varepsilon) \subseteq U$. By Lemma 4.4, the collection of open balls forms a basis for the open ball topology. □

4.4 Using Homotopy to Measure Similarity

Within this section, we follow the notion of homotopy as introduced by Knudson [21]. Let X and Y be topological spaces.

A function $f : X \to Y$ is called *continuous* if the preimage $f^{-1}(V) \subseteq X$ is open for every open set $V \subseteq Y$.

Let A be a subspace of X, then (X, A) is called a pair. Further, if we have a continuous map $f : X \to Y$ that maps a subspace $A \subset X$ into the subspace $B \subset Y$, we write $f : (X, A) \to (Y, B)$.

Definition 4.8 (Homotopy)**.** *Let $f, g \colon (X, A) \to (Y, B)$ be two continuous maps that agree on $A \subseteq X$. A continuous map*

$$H \colon X \times I \to Y, \quad \textit{with } I = [0, 1],$$

is called a homotopy from f to g relative to A *if the following properties hold:*

1. $H(x, 0) = f(x)$ *for all* $x \in X$*,*
2. $H(x, 1) = g(x)$ *for all* $x \in X$*,*
3. $H(a, t) = f(a)$ *for all* $a \in A$ *and* $t \in [0, 1]$*.*

In this case, we write $f \simeq g$ rel A. The interval $I = [0, 1]$ is chosen without loss of generality, as any closed interval can be rescaled to this standard domain.

If there exists a homotopy between two continuous maps $f, g \colon X \to Y$, then f and g are said to be **homotopic**.

A homotopy H can be interpreted in two equivalent ways:

- For each point $x \in X$, the map $t \mapsto H(x, t)$ defines a path in Y from $f(x)$ to $g(x)$, which depends continuously on x.

- For each $t \in [0,1]$, the map $x \mapsto H(x,t)$ defines a continuous map $H_t \colon X \to Y$, and the family $(H_t)_{t\in[0,1]}$ varies continuously with t.

In this sense, a homotopy is a **continuous deformation** of f into g.

Consider the map $F : \mathbb{R} \times [0,1] \to \mathbb{R}$ given by $F(x,t) = x^3 + tx + 1$. Since this is continuous, it is a homotopy from $f_{-1}(x) = x^3 - x + 1$ and $f_1(x) = x^3 + x + 1$ [21].

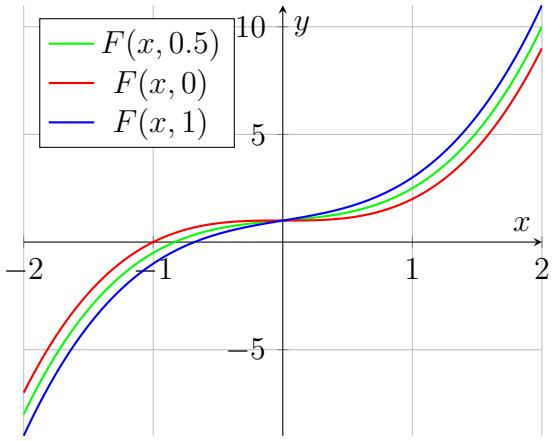


Figure 4.1: Plot of the homotopy function $F(x,t) = x^3 + tx + 1$ for different values of t.

Let $\mathcal{E}_V$ denote a set of encoders, i.e., functions from a set X into a vector space V. Define $\mathcal{C}_{V,W} := \{\varphi \colon V \to W \mid \varphi \text{ is continuous}\}$. Let $S \subseteq \mathcal{C}_{V,W}$ be a specified class of admissible transformations between representation spaces, such as the set of all continuous, smooth, or affine maps.

Two encoders $f, g \in \mathcal{E}_V$ are called *S-homotopic* if there exists a transformation $\psi \in S$ such that f can be approximated by $\psi \circ g$.

We refer to this notion as *S-homotopy*, and study it as a framework to compare encoders via transformations in S. Accordingly, we define the set

$$E_h := \{f \in \mathcal{E}_V \mid \exists\, \psi \in S \text{ such that } f \approx \psi \circ h\}\,,$$

which collects all encoders to which h can be transformed via S-mappings.

A special case is obtained by taking $S = \mathrm{Aff}(V)$, the set of affine maps on V, which corresponds to the affine encoder framework studied in the work of Chan et al. [3].

5 Similarity of Neural Networks

As mentioned in the introduction (see Section 1), understanding whether neural network models are similar is crucial in many areas of research and application. This chapter is primarily based on the work by Klabunde et al. [20], with additional references cited where appropriate. Since there are various perspectives on what it means for models to be similar, we focus on two central and state-of-the-art viewpoints, illustrated in Figure 5.1. These are:

- Representational similarity, and
- Functional similarity

We examine the key perspectives of *representational similarity measures* and *functional similarity measures* in more detail in Sections 5.1 and 5.2, respectively.

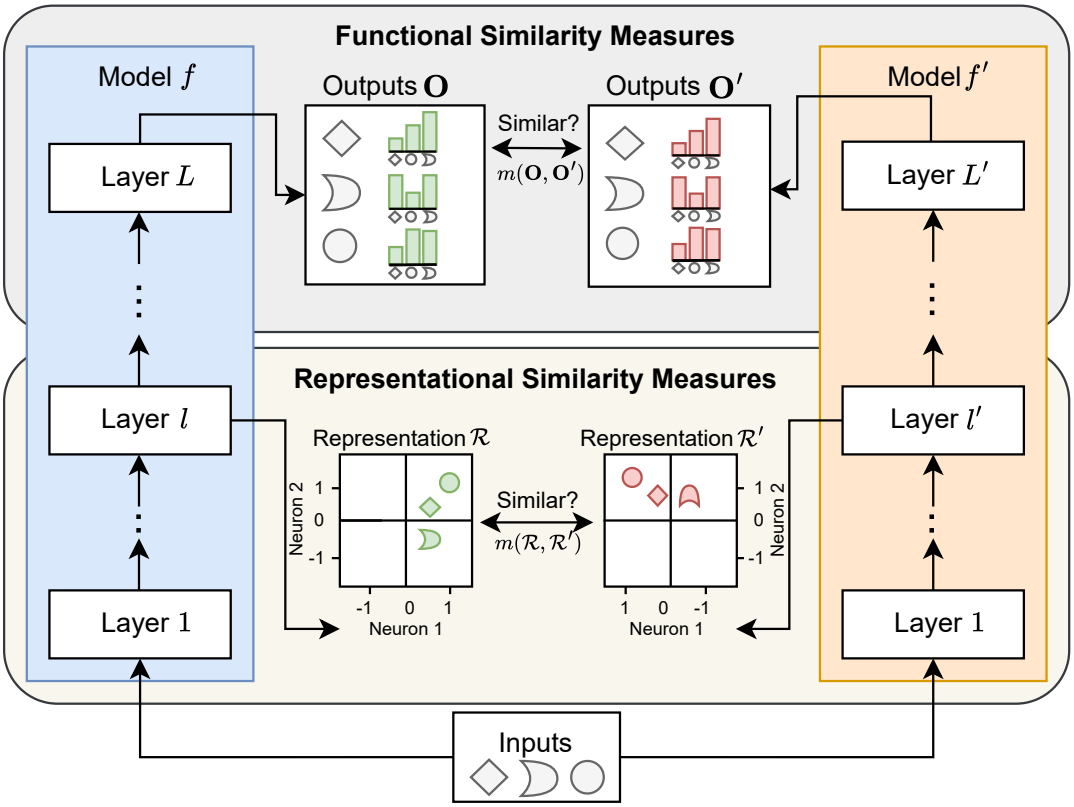


Figure 5.1: Visualization of representational and functional similarity [20].

5.1 Representational Similarity Measure

In order to compare neural networks, representational similarity methods typically assess how similar the internal representations of a fixed set of inputs are across different models or layers. We will focus in the following on comparing activations at specific layers, but alternative approaches may also consider other components, such as weights or gradients.

Therefore, we consider a representation of the model f at layer l:

$$R := R^{(l)} = (f^{(l)} \circ f^{(l-1)} \circ ... \circ f^{(1)}) \in \mathbb{R}^{N \times D}.$$

Here, N denotes the (possibly infinite) number of datapoints, and D the number of neurons in a single layer.

Definition 5.1 (Representational similarity measures)**.** *A representational similarity measure is a map*

$$m : \mathbb{R}^{N \times D} \times \mathbb{R}^{N \times D} \longrightarrow \mathbb{R},$$

that assigns a similarity score $m(R, R')$ to a pair of representations R, R' of different models f, f' that use the same inputs $\mathcal{X}$.

Nevertheless, we can consider two representation matrices $R, R' \in \mathbb{R}^{N \times D}$ to be equivalent, even if they are not identical element-wise. This is particularly relevant when comparing language encoders whose outputs may differ due to arbitrary linear transformations, such as sign flips (e.g., $R = -R'$) or rotations. Such transformations do not necessarily alter the underlying structure or information content of the representations. Hence, comparing representations up to such equivalence classes allows us to focus on their functional behavior rather than their raw numerical values.

The specific type of transformation used to define equivalence depends on the context and intended application. For instance, some analyses may permit any orthogonal transformation, while others restrict equivalence to affine or even task-informed transformations.

By using terms of bijective mappings (transformations), we can formalize the notions of equivalence as follows:

$$\psi : \mathbb{R}^{N \times D} \to \mathbb{R}^{N \times D}$$
$$\psi(R) = R'.$$

Definition 5.2 (Equivalence of Representations)**.** *Let $R, R' \in \mathbb{R}^{N \times D}$ be two representations and $\mathcal{T} := \mathcal{T}(N, D)$ a group. We denote R and R' as equivalent with respect to a group $\mathcal{T}$, written as $R \sim_{\mathcal{T}} R'$ if there is a $\psi \in \mathcal{T}$, such that $\psi(R) = R'$*

The group of transformations under which representations should be considered equivalent is of crucial importance in practice, since equivalent representations should be indistinguishable for the chosen similarity measure. In contrast, non-equivalent representations must be distinguishable for a similarity measure. Formally, this means that a measure must be invariant to exactly those groups of transformations under which the underlying representations are equivalent.

Definition 5.3 (Invariant Representational Similarity Measure)**.** *Let $\mathcal{T}(N, D)$ be a group of transformations acting on representation matrices in $\mathbb{R}^{N\times D}$. A similarity measure $m : \mathbb{R}^{N\times D} \times \mathbb{R}^{N\times D} \to \mathbb{R}$ is called* invariant under *$\mathcal{T}(N, D)$ if for all $R, R' \in \mathbb{R}^{N\times D}$ and all $\varphi, \varphi' \in \mathcal{T}(N, D)$, it holds that*

$$m(R, R') = m(\varphi(R), \varphi'(R')).$$

In particular, if $R \sim_{\mathcal{T}} R'$, then $m(R, R) = m(R, R')$.

Using this definition, we will later examine whether the similarity measures considered satisfy the axioms of a distance metric. In the context of representational similarity, however, one of these axioms—namely the *identity of indiscernibles*, can be relaxed. Rather than requiring that $m(R, R') = 0$ if and only if $R = R'$, it is sufficient to require that

$$m(R, R') = 0 \iff R \sim_{\mathcal{T}} R',$$

where $\mathcal{T}$ is a group of transformations under which the measure m is invariant [35]. This allows the similarity measure to capture equivalence of representations up to transformations that preserve their functional or structural meaning.

5.2 Functional Similarity Measure

If we measure similarity by the output behavior of neural networks, we refer to *functional similarity measures*. To formalize this, let f be a neural network trained on a set of inputs $X = \{x_1, \ldots, x_N\}$ to perform a classification task with C output classes. Evaluated on a subset $Y \subseteq X$, the network produces an output matrix

$$\mathbf{O} := f(Y) \in \mathbb{R}^{N\times C},$$

where N denotes the (possibly infinite) number of datapoints and C the number of classes.

Definition 5.4 (Functional similarity measure)**.** *A functional similarity measure is a map*

$$m : \mathbb{R}^{N\times D} \times \mathbb{R}^{N\times D} \longrightarrow \mathbb{R},$$

that assigns a similarity score $m(\mathbf{O}, \mathbf{O}')$ to a pair of outputs $\mathbf{O}, \mathbf{O}'$ of different models f, f' that use the same inputs X.

It is assumed that the compared outputs $\mathbf{O}, \mathbf{O}' \in \mathbb{R}^{N\times C}$ are class-aligned, meaning that the columns of both matrices correspond to the same set of classes in the same order.

We induce an equivalence relation based on that similarity in the following.

Definition 5.5 (Equivalence of Functions)**.** *Let X be a fixed input set, f, f' two neural networks and $m(f(Y), f'(Y)) \in \mathbb{R}$ the functional similarity score.*

We denote f and f' as equivalent $f \sim_Y f'$, if $m(f(Y), f'(Y)) \geq \tau$, where the user defined similarity threshold $\tau \to 0$. In the following, we will write for simplicity $f = f'$ if both neural networks produce the same output as both of them are an element of the same equivalence class.

6 Similarity Analysis using Affine Transformations

In this chapter, we present the framework developed by Chan et al. [2], which introduces two notions of homotopy to compare language encoders in a principled way.

These two notions, *intrinsic* and *extrinsic homotopy*, capture distinct perspectives on model similarity. Intrinsic homotopy focuses on representational similarity: it compares models solely based on their internal hidden representations, typically using the final encoder layer, and does not rely on any downstream task.

In contrast, extrinsic homotopy assesses models in a task-specific context. Here, a classification head is trained on top of each encoder, and similarity is measured via alignment of outputs or task performance.

At the core of the intrinsic and extrinsic perspective lies a *specialization quasi-ordering* induced by a hemi-metric. This preorder expresses directional approximability between encoders and serves as the formal backbone for defining intrinsic and extrinsic homotopy. By separating intrinsic and extrinsic views, this framework enables a comprehensive analysis of model similarity from both structural and functional perspectives.

Based on the Prelimiaries in Chapter 4 a general recipe for defining hemi-metric spaces on function spaces can be defined.

Definition 6.1 (Hemi-metric on Function Spaces)**.** *Let X be a set, and let (Y, d) be a hemi-metric space. Furthermore, let*

$$S : X \to \mathcal{P}(Y) \setminus \{\emptyset\}, \quad x \mapsto E_x$$

be a mapping that assigns to each $x \in X$ a non-empty subset $E_x \subseteq Y$.

We define a hemi-metric $d_S^{\mathcal{H}}$ on X by setting

$$d_S^{\mathcal{H}}(x, y) := d^{\mathcal{H}}(E_x, E_y) := \sup_{y_1 \in E_y} \inf_{x_1 \in E_x} d(x_1, y_1),$$

which is sometimes called the Hausdorff hemi-distance from E_x to E_y.

Optionally, we can symmetrize this to define an extended pseudo-metric by

$$d_S^{sym}(x, y) := \max\left(d_S^{\mathcal{H}}(x, y),\ d_S^{\mathcal{H}}(y, x)\right).$$

We now define a notion of alignment between two encoders via a class of affine transformations. Let $S \subset \mathrm{Aff}(V)$ be a set of affine maps acting on a vector space V. For two encoders $h, g : X \to V$, we define the S-alignment from g to h by

$$d_S(h, g) := d_\infty^{\mathcal{H}}(h, S(g)) := \inf_{\psi \in S} \|h - \psi \circ g\|_\infty,$$

where $S(g) := \{\psi \circ g \mid \psi \in S\}$.

This captures how well the encoder g can be transformed into h via elements of S. Note that $d_S(h, g)$ is generally asymmetric: it measures the alignment from g to h, but not necessarily the other way around.

We further define the S-norm of an encoder h as

$$\|h\|_S := d_S(0_{\mathcal{E}_V}, h),$$

which measures how well h can be approximated by an S-transformation of the zero encoder.

This concept of affine alignment can be seen as a special case of the construction in Definition 6.1, where we set $X = Y = \mathcal{E}_V$, and use the uniform convergence hemi-metric from Subsection 4.2.

6.1 Intrinsic Homotopy

Chan et al. [2] developed the notion of *intrinsic homotopy*, which aims to compare language encoders independently of any downstream task. The key idea is to define similarity intrinsically, based solely on the internal representations produced by the models. This is formalized using the structure of a hemi-metric space and a directional relation derived from it.

A central component of this framework is the *specialization quasi-ordering* $\gtrsim_d$, introduced on a hemi-metric space (X, d) as in Definition 4.5. It captures whether one encoder can approximate another via a structure-preserving transformation. Since such transformations may exist in only one direction, the relation is defined as a *preorder*, i.e., it is reflexive and transitive but not necessarily symmetric.

This asymmetric relation provides a principled way to compare encoders in terms of representational similarity. If mutual approximability holds, i.e., both $h \gtrsim_d g$ and $g \gtrsim_d h$, the encoders are said to be *exactly intrinsically affinely homotopic*.

In practice, intrinsic homotopy is based on the output of the final hidden layer, which aggregates semantic information and serves as the default input to downstream classifiers. Accordingly, this notion aligns closely with the concept of representational similarity discussed in Section 5.1.

We begin this chapter by examining how the specialization preorder arises from the hemi-metric structure and serves as the foundation for intrinsic homotopy.

Lemma 6.2. *Let (X, d) be a hemi-metric space, and let $\gtrsim_d$ denote the specialization quasi-ordering with respect to the open ball topology. Then, for all $x, y \in X$,*

$$x \gtrsim_d y \quad \iff \quad d(x, y) = 0.$$

Proof. „⇒“Suppose $x \gtrsim_d y$. Then, for all $\varepsilon > 0$, we have $y \in B(x, \varepsilon)$, which implies $d(x, y) < \varepsilon$. Since this holds for all $\varepsilon > 0$, we must have $d(x, y) = 0$.

„⇐“Now suppose $d(x, y) = 0$. Let U be any open set containing x. By Lemma 4.7, there exists $\varepsilon > 0$ such that $B(x, \varepsilon) \subseteq U$. Since $d(x, y) = 0 < \varepsilon$, it follows that $y \in B(x, \varepsilon) \subseteq U$, so $x \gtrsim_d y$. □

We consider two encoders f, g in the real vector space of language encoders $\mathcal{E}_V := V^{\Sigma^*}$, and define an affine intrinsic pre-order $\gtrsim_{\mathrm{Aff}}$ on this space.

Definition 6.3 (Exact Intrinsic Affine Homotopy)**.** *Two encoders $h, g \in \mathcal{E}_V$ are said to be exactly intrinsically affinely homotopic, written $h \simeq_{\mathrm{Aff}} g$, if both $g \gtrsim_{\mathrm{Aff}} h$ and $h \gtrsim_{\mathrm{Aff}} g$ hold.*

In the original formulation, two encoders $h, g \in \mathcal{E}_V$ are defined to be exactly intrinsically affinely homotopic if

$$d_{\mathrm{Aff}(V)}(h, g) = 0 \quad \text{and} \quad \mathrm{rank}(h) = \mathrm{rank}(g).$$

Chan et al. [2] showed that this condition is equivalent to the ladder-based formulation presented above, which is more convenient for our purpose as it avoids the need to explicitly define rank.

6.2 Extrinsic Homotopy

Compared to intrinsic homotopy, which evaluates language encoders based on their internal structure, *extrinsic homotopy* focuses on their observable behavior in downstream tasks. It captures *functional similarity*, as discussed in Section 5.2, by assessing how similarly two encoders perform when composed with task-specific classifiers.

Formally, two encoders h and g are considered extrinsically homotopic if they yield similar performance across all downstream tasks in which they could serve as feature extractors. To make this precise, each encoder is composed with affine classifiers, yielding complete prediction pipelines. This setup allows us to define a hemi-metric that quantifies behavioral divergence between encoders across tasks.

If this distance is zero, the encoders are said to be *exactly extrinsically homotopic*. In practice, this notion can be approximated using a finite set of input strings and optimization techniques such as gradient descent. Finally, we explore how extrinsic and intrinsic homotopy relate, and how functional similarity can complement representational alignment.

Let Σ be a finite alphabet and let Σ^* denote the set of all finite strings over Σ, i.e., the Kleene closure of Σ. Let V be a vector space of intermediate representations, and let $W := \mathbb{R}^C$ denote the output space for a classification task with C classes.

An affine map $f : V \to W$ is given by a linear map $A \in \mathcal{L}(V, W)$ and a bias vector $b \in W$, such that

$$f(v) = A \cdot v + b \quad \text{for all } v \in V.$$

We denote the set of all affine maps from V to W by $\mathrm{Aff}(V, W)$.

We define the following sets of encoder functions:

- $\mathcal{E}_V := \mathrm{Map}(\Sigma^*, V)$ – encoders mapping strings to vector representations,
- $\mathcal{E}_W := \mathrm{Map}(\Sigma^*, W)$ – classifiers mapping strings to class scores,
- $\mathcal{E}_{\Delta^{C-1}} := \mathrm{Map}(\Sigma^*, \Delta^{C-1})$ – classifiers returning probability distributions over C classes.

Given an encoder $h \in \mathcal{E}_V$, we define the set of affine classifiers based on h as

$$\mathrm{Aff}_{V,W}(h) := \{\psi \circ h \mid \psi \in \mathrm{Aff}(V, W)\} \subset \mathcal{E}_W.$$

Applying the softmax function with inverse temperature $\lambda > 0$ yields the associated family of log-linear classifiers:

$$\mathcal{V}_C(h) := \{\mathrm{softmax}_\lambda \circ f \mid f \in \mathrm{Aff}_{V,W}(h)\} \subset \mathcal{E}_{\Delta^{C-1}}.$$

Based on this construction, we define a hemi-metric that quantifies the difference between two encoders $h, g \in \mathcal{E}_V$ in terms of their downstream classification behavior:

$$d^{\mathcal{H}}_{\mathcal{V}_C}(h, g) := \sup_{\psi \in \mathcal{V}_C(h)} \inf_{\varphi \in \mathcal{V}_C(g)} \|\psi \circ h - \varphi \circ g\|_\infty.$$

This metric captures how differently two encoders behave when composed with affine classification heads and applied to the same string input. It forms the basis for the definition of *extrinsic homotopy*, as introduced in the following section.

Definition 6.4 (Exact Extrinsic Homotopy)**.** *Two encoders $g, h \in \mathcal{E}_V$ are called* exactly extrinsically homotopic *if*

$$d^{\mathcal{H}}_{\mathcal{V}(V,\Delta)}(h, g) = 0.$$

Furthermore, Chan et al. [2] defined an extrinsic affine preorder and showed, that affine instrinsic preorder is contained in the extrinsic preorder. This connects the internal structure of encoders to their external behavior.

In the practical concern two language encoders are not considered over the entire Σ^*, only over a finite set of strings $X = \{x_i\}_{i=1}^N$. Then, the encoders are considered as matrices and the notion of similarity is approximated by optimizing over the affine maps, for example by using gradient descent.

7 Improving Similarity Analysis using Nonlinear Transformations

In this chapter we formalize the mathematical methods of our novel approach. We focus on continuous non-linear transformations instead of affine ones, as presented by Chan et al. [2]. The non-linear transformations are represented by a neural network in practice, as it is one of the most used state-of-the-art approaches for non-linear transformations. We will formalize the mathematical methods to analyse the influence of continuous non-linear transformations on the approach presented in the chapter 6. First, we will modify the definition of intrinsic and extrinsic homotopy. Therefore, we extend the notion of homotopy by considering the set of continuous maps, which includes both linear and non-linear functions, as long as they are continuous. Furthermore, we define an approach, where the similarity will be measured by the models performance.

7.1 Intrinsic Homotopy on Non-linear Transformations

Inspired by the framework of Chan et al. [2], we consider a space of language encoders $\mathcal{E}_V$, viewed as a subset of the space of continuous functions from sequences Σ^* to a finite-dimensional vector space $V \subset \mathbb{R}^d$, where Σ denotes a finite alphabet and Σ^* its Kleene closure.

Equipped with the uniform norm

$$\|h\|_\infty := \sup_{x \in \Sigma^*} |h(x)|_V,$$

the space $(\mathcal{E}_V, \|\cdot\|_\infty)$ becomes a complete metric space with distance $d_\infty(h, g) := \|h - g\|_\infty$, due to the completeness of V.

The goal of this section is to define an intrinsic, nonlinear notion of homotopy between language encoders. While Chan et al. [2] define such a structure using affine maps, we extend this approach to a more expressive class of transformations: namely, general nonlinear maps. In principle, we are interested in the full range of nonlinear structure-preserving transformations. However, in order to ensure that the induced distance satisfies the triangle inequality, and thus defines a valid hemi-metric, we restrict ourselves to 1-Lipschitz continuous functions. This technical constraint is essential in the proof of Proposition 7.1, where we rely on the fact that 1-Lipschitz functions are closed under composition and do not increase the ℓ^∞-distance.

Intuitively, we say that an encoder h is intrinsically related to another encoder g if there exists a well-behaved transformation that maps g closely to h. To formalize this idea, we define an asymmetric notion of distance $d_{\mathcal{F}_{\text{Lip}_1}}(h, g)$, which quantifies how well h can be approximated by transformed versions of g under a class of admissible maps. The smaller this distance, the more intrinsically related the models are. If the distance is zero, we consider the two encoders intrinsically similar.

We now make this precise using the notion of a hemi-metric as introduced in Definition 6.1.

Let $\mathcal{F} \subset \mathcal{C}(V, V)$ denote a class of admissible transformations, where $V \subset \mathbb{R}^d$ is a finite-dimensional vector space. Each transformation $\psi \in \mathcal{F}$ maps encoder outputs from V into V, and induces a new encoder $\psi \circ g \colon \Sigma^* \to V$.

In the following, we will focus on the case where $\mathcal{F} = \mathcal{F}_{\text{Lip}_1}$, the set of 1-Lipschitz continuous functions from V to itself.

Then, for any pair of encoders $h, g \in \mathcal{E}_V$, we define the hemi-metric, as in Definition 6.1

$$d_{\mathcal{F}_{\text{Lip}_1}}(h, g) := \inf_{\psi \in \mathcal{F}_{\text{Lip}_1}} \|h - \psi \circ g\|_\infty,$$

which measures how well h can be approximated by transformed versions of g. This construction induces a preorder $h \gtrsim_{\text{Intr}} g$ on the space of encoders.

Definition 7.1 (Intrinsic Relation)**.** *Let $h, g \in \mathcal{E}_V$, and let $\mathcal{F}_{\text{Lip}_1} \subset \mathcal{C}(V, V)$ be a class of admissible transformations. We say that h is intrinsically related to g, written*

$$h \gtrsim_{\text{Intr}} g,$$

if the induced distance satisfies

$$d_{\mathcal{F}_{\text{Lip}_1}}(h, g) = \inf_{\psi \in \mathcal{F}_{\text{Lip}_1}} \|h - \psi \circ g\|_\infty = 0.$$

As we need to restrict ourselves to 1-Lipschitz continuous functions, we will introduce the Lipschitz Continuty in the following.

Definition 7.2 (Lipschitz Continuity)**.** *A function $\psi \colon V \to W$ between two normed vector spaces is called* Lipschitz continuous *[9] if there exists a constant $L \geq 0$ such that for all $v_1, v_2 \in V$, we have*

$$\|\psi(v_1) - \psi(v_2)\|_W \leq L \cdot \|v_1 - v_2\|_V.$$

The smallest such constant L is called the Lipschitz constant *of ψ. If $L \leq 1$, we say that ψ is* 1-Lipschitz*.*

Proposition 7.3. *Let $\mathcal{F}_{\text{Lip}_1} \subset \mathcal{C}(V, V)$ denote the set of 1-Lipschitz continuous functions from V to W, where V and W are finite-dimensional normed vector spaces over $\mathbb{R}$. Then $\mathcal{F}_{\text{Lip}_1}$ is a closed subset of the complete metric space $(\mathcal{C}(V, V), d_{\mathcal{F}_{\text{Lip}_1}})$, and hence itself complete.*

Proof. Let $(\psi_n) \subset \mathcal{F}_{\text{Lip}_1}$ be a sequence that converges uniformly to some function $\psi \in \mathcal{C}(V, V)$ with respect to the supremum norm

$$\|\psi_n - \psi\|_\infty := \sup_{x \in V} \|\psi_n(x) - \psi(x)\|_W.$$

We show that ψ is also 1-Lipschitz: for any $x, y \in V$,

$$\|\psi(x) - \psi(y)\|_W \leq \|\psi(x) - \psi_n(x)\|_W + \|\psi_n(x) - \psi_n(y)\|_W + \|\psi_n(y) - \psi(y)\|_W.$$

Using the triangle inequality and the fact that $\psi_n \in \mathcal{F}_{\text{Lip}_1}$, we get

$$\|\psi(x) - \psi(y)\|_W \leq 2\|\psi - \psi_n\|_\infty + \|x - y\|_V.$$

Taking the limit $n \to \infty$, the first term vanishes and we obtain

$$\|\psi(x) - \psi(y)\|_W \leq \|x - y\|_V,$$

so ψ is 1-Lipschitz and thus $\psi \in \mathcal{F}_{\text{Lip}_1}$. Therefore, $\mathcal{F}_{\text{Lip}_1}$ is closed.

Since $(\mathcal{C}(V, V), d_\infty)$ is a complete metric space (by completeness of W [34]), it follows that $\mathcal{F}_{\text{Lip}_1}$ is complete as a closed subset of a complete space. □

To formalize how well an encoder h can be approximated by another encoder g via a transformation from a given function class, we define the *intrinsic approximation distance*

$$d_{\mathcal{F}_{\text{Lip}_1}}(h, g) := \inf_{\psi \in \mathcal{F}_{\text{Lip}_1}} \|h - \psi \circ g\|_\infty,$$

where $\mathcal{F}_{\text{Lip}_1} \subset \mathcal{C}(V, V)$ denotes a set of admissible transformations – for instance, the class of 1-Lipschitz continuous maps. This defines a *hemi-metric* on the space of encoders $\mathcal{E}_V$, as it may fail to be symmetric.

The *specialization quasi-ordering* induced by this hemi-metric is given by

$$h \gtrsim_{\mathcal{F}_{\text{Lip}_1}} g \quad \Leftrightarrow \quad d_{\mathcal{F}_{\text{Lip}_1}}(h, g) = 0.$$

Intuitively, this means that h can be approximated arbitrarily well by applying some transformation $\psi \in \mathcal{F}_{\text{Lip}_1}$ to g. That is, for any error tolerance $\varepsilon > 0$, there exists a transformation $\psi \in \mathcal{F}_{\text{Lip}_1}$ such that $\|h - \psi \circ g\|_\infty < \varepsilon$. The relation $\gtrsim_{\mathcal{F}_{\text{Lip}_1}}$ is thus a *preorder*: it is reflexive and transitive, but not necessarily antisymmetric.

Proposition 7.4 (Intrinsic Preorder)**.** *Let $\mathcal{F}_{\text{Lip}_1} \subset \mathcal{C}(V, V)$ denote the class of 1-Lipschitz continuous functions from V to itself, and define the map*

$$d_{\mathcal{F}_{\text{Lip}_1}} : \mathcal{E}_V \times \mathcal{E}_V \to \mathbb{R}, \quad (h, g) \mapsto \inf_{\psi \in \mathcal{F}_{\text{Lip}_1}} \|h - \psi \circ g\|_\infty.$$

Then $d_{\mathcal{F}_{\text{Lip}_1}}$ is a hemi-metric, and the induced specialization quasi-ordering

$$h \gtrsim_{\mathcal{F}_{\text{Lip}_1}} g \quad \Longleftrightarrow \quad d_{\mathcal{F}_{\text{Lip}_1}}(h, g) = 0$$

defines a preorder on $\mathcal{E}_V$, by Lemma 6.2.

Proof. We verify that $d_{\mathcal{F}_{\mathrm{Lip}_1}}$ is a hemi-metric on $\mathcal{E}_V$, as required by the axioms of Definition 4.1.

H1: For any $h \in \mathcal{E}_V$, we have $\mathrm{id} \in \mathcal{F}_{\mathrm{Lip}_1}$, so

$$d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, h) \leq \|h - \mathrm{id} \circ h\|_\infty = 0.$$

Since the infimum is non-negative, it follows that $d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, h) = 0$.

H2: Let $h, g, f \in \mathcal{E}_V$. We want to show:

$$d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, f) \leq d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + d_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f).$$

Since $\mathcal{F}_{\mathsf{Lip}_1} \subset \mathcal{C}(V, V)$ is the set of 1-Lipschitz continuous maps acting on the complete metric space $(\mathcal{E}_V, d_\infty)$, we can approximate the infimum arbitrarily well.

That is, for any $\varepsilon > 0$, there exist alignments:

- $\psi_g \in \mathcal{F}_{\mathsf{Lip}_1}$ such that

$$\|h - \psi_g(g)\|_\infty \leq d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + \varepsilon,$$

- $\psi_f \in \mathcal{F}_{\mathsf{Lip}_1}$ such that

$$\|g - \psi_f(f)\|_\infty \leq d_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f) + \varepsilon.$$

Such approximations exist because the infimum in $d_{\mathcal{F}_{\mathrm{Lip}_1}}$ is taken over a set of continuous functions, and $\mathcal{E}_V$ is complete. In particular, minimizing sequences $(\psi_n) \subset \mathcal{F}_{\mathsf{Lip}_1}$ with $\|h - \psi_n(g)\|_\infty \to d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g)$ converge (up to subsequences) to a map in $\mathcal{F}_{\mathsf{Lip}_1}$, so approximations up to ε are always available.

Since $\mathcal{F}_{\mathsf{Lip}_1}$ is closed under composition (composition of 1-Lipschitz maps is again 1-Lipschitz), we can define

$$\psi := \psi_g \circ \psi_f \in \mathcal{F}_{\mathsf{Lip}_1}.$$

Then

$$d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, f) \leq \|h - \psi(f)\|_\infty = \|h - \psi_g(\psi_f(f))\|_\infty.$$

We estimate this norm using the triangle inequality:

$$\|h - \psi_g(\psi_f(f))\|_\infty \leq \|h - \psi_g(g)\|_\infty + \|\psi_g(g) - \psi_g(\psi_f(f))\|_\infty.$$

The first term is bounded by $d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + \varepsilon$ by construction. To bound the second term, we use the fact that ψ_g is 1-Lipschitz, i.e., has Lipschitz constant $L = 1$:

$$\|\psi_g(g) - \psi_g(\psi_f(f))\|_\infty \leq L \cdot \|g - \psi_f(f)\|_\infty \leq d_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f) + \varepsilon.$$

Putting it all together:

$$d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, f) \leq \|h - \psi_g(g)\|_\infty + \|g - \psi_f(f)\|_\infty \leq d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + d_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f) + 2\varepsilon.$$

Since $\varepsilon > 0$ was arbitrary, the triangle inequality follows:

$$d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, f) \leq d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + d_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f).$$

Note that the triangle inequality in this form relies on the fact that all transformation functions in $\mathcal{F}_{\text{Lip}_1}$ have Lipschitz constant $L = 1$. If we allowed larger constants $L > 1$, the second term

$$\|\psi_g(g) - \psi_g(\psi_f(f))\|_\infty \leq L \cdot \|g - \psi_f(f)\|_\infty$$

would scale the second distance term by L, and we would only obtain the weaker inequality

$$d_{\mathcal{F}_{\text{Lip}_1}}(h, f) \leq d_{\mathcal{F}_{\text{Lip}_1}}(h, g) + L \cdot d_{\mathcal{F}_{\text{Lip}_1}}(g, f),$$

which does not satisfy the standard triangle inequality unless $L = 1$. Hence, the restriction to 1-Lipschitz maps is essential to ensure that $d_{\mathcal{F}_{\text{Lip}_1}}$ defines a hemi-metric.

Consequently, $d_{\mathcal{F}_{\text{Lip}_1}}$ satisfies the axioms of a hemi-metric on $\mathcal{E}_V$ in the sense of Definition 4.1.

To complete the proof, we verify that the relation

$$h \gtrsim_{\text{Intr}} g \quad \Longleftrightarrow \quad d_{\mathcal{F}_{\text{Lip}_1}}(h, g) = 0$$

is a preorder on $\mathcal{E}_V$:

- Reflexivity: For all $h \in \mathcal{E}_V$, we have $d_{\mathcal{F}_{\text{Lip}_1}}(h, h) = 0$ by axiom (H1), so $h \gtrsim_{\text{Intr}} h$.
- Transitivity: Let $h \gtrsim_{\text{Intr}} g$ and $g \gtrsim_{\text{Intr}} f$, i.e., $d_{\mathcal{F}_{\text{Lip}_1}}(h, g) = 0$ and $d_{\mathcal{F}_{\text{Lip}_1}}(g, f) = 0$. Then by the triangle inequality:

  $$d_{\mathcal{F}_{\text{Lip}_1}}(h, f) \leq d_{\mathcal{F}_{\text{Lip}_1}}(h, g) + d_{\mathcal{F}_{\text{Lip}_1}}(g, f) = 0,$$

  hence $d_{\mathcal{F}_{\text{Lip}_1}}(h, f) = 0$, so $h \gtrsim_{\text{Intr}} f$.

Thus, $\gtrsim_{\text{Intr}}$ is reflexive and transitive, i.e., a preorder. □

Definition 7.5 (Exact Intrinsic Non-Linear Homotopy)**.** *Let $h, g \in \mathcal{E}_V$, and let $\mathcal{F}_{Lip_1} \subset \mathcal{C}(V, V)$ denote the class of admissible 1-Lipschitz continuous transformations. We say that h and g are* exactly intrinsically non-linearly homotopic*, written*

$$h \gtrsim_{\mathcal{F}_{Lip_1}} g,$$

if both directional distances vanish:

$$d_{\mathcal{F}_{Lip_1}}(h, g) := \inf_{\psi \in \mathcal{F}_{Lip_1}} \|h - \psi \circ g\|_\infty = 0 \quad \textit{and} \quad d_{\mathcal{F}_{Lip_1}}(g, h) := \inf_{\phi \in \mathcal{F}_{Lip_1}} \|g - \phi \circ h\|_\infty = 0.$$

That is, each encoder can be approximated arbitrarily well (in the ℓ^∞-norm) by applying admissible transformations to the other.

7.2 Extrinsic Homotopy on Non-linear Transformations

In this section, we extend the notion of homotopy to incorporate extrinsic structure, focusing not only on the encoder representations themselves, but also on their behavior when composed with downstream classifiers.

Whereas intrinsic homotopy compares encoder functions directly via transformations in function space, extrinsic homotopy evaluates the functional output of composite systems, such as encoder–classifier pipelines.

The key idea is to assess whether one encoder can emulate another by applying suitable post-processing transformations e.g., neural classifiers, followed by normalization.

To formalize this, let $V \subset \mathbb{R}^d$ denote the output space of an encoder and $W = \mathbb{R}^C$ the target space of classifier outputs (e.g., logits). We fix a class of admissible transformations $\mathcal{C}_{V,W} \subset \mathcal{C}(V, W)$, consisting of continuous, potentially non-linear maps that preserve relevant structure (e.g., Lipschitz continuity or smoothness). These transformations model classifiers applied to encoder outputs.

Let Σ be a finite alphabet and let Σ^* denote the set of all finite strings over Σ, i.e., the Kleene closure of Σ. Let V be a vector space of intermediate representations, and let $W := \mathbb{R}^C$ denote the output space for a classification task with C classes.

We fix a class of admissible transformations $\mathcal{C}_{V,W} \subset \mathcal{C}(V, W)$, consisting of continuous, potentially nonlinear functions (e.g., neural networks) that map encoder representations into classifier logits. These functions model downstream classifiers.

We define the following sets of functions:

- $\mathcal{E}_V := \mathrm{Map}(\Sigma^*, V)$ – encoders mapping strings to vector representations,
- $\mathcal{E}_W := \mathrm{Map}(\Sigma^*, W)$ – classifiers mapping strings to class scores (logits),
- $\mathcal{E}_{\Delta^{C-1}} := \mathrm{Map}(\Sigma^*, \Delta^{C-1})$ – classifiers returning class probability distributions.

Given an encoder $h \in \mathcal{E}_V$, we define the set of classifier functions applied to h as

$$\mathcal{C}_{V,W}(h) := \{\psi \circ h \mid \psi \in \mathcal{C}_{V,W}\} \subset \mathcal{E}_W.$$

Applying the softmax function with inverse temperature $\lambda > 0$ yields the associated family of nonlinear logit-to-probability classifiers:

$$\mathcal{V}_C(h) := \{\mathrm{softmax}_\lambda \circ f \mid f \in \mathcal{C}_{V,W}(h)\} \subset \mathcal{E}_{\Delta^{C-1}}.$$

Here, $\mathrm{softmax}_\lambda : W \to \Delta^{C-1}$ is defined as

$$\mathrm{softmax}_\lambda(x)_i := \frac{\exp(\lambda x_i)}{\sum_{j=1}^{C} \exp(\lambda x_j)} \quad \text{for } x \in \mathbb{R}^C,\ i = 1, \ldots, C.$$

This framework allows us to compare encoders extrinsically by measuring how closely the outputs of their composed pipelines align. Following Definition 6.1, we define the extrinsic hemi-metric as:

$$d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(h,g) := \sup_{\psi\in\mathcal{C}_{V,W}} \inf_{\varphi\in\mathcal{C}_{V,W}} \|\psi\circ h - \varphi\circ g\|_\infty,$$

which quantifies how well the output of encoder g can be aligned to that of h through admissible post-transformations in $\mathcal{C}_{V,W}$.

Definition 7.6 (Norm-induced Distance Functions)**.** *Using the hemi-metric framework introduced in Definition 6.1, we define two norm-induced distance functions on the encoder space $\mathcal{E}_V$:*

$$\begin{aligned} d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(h,g) &:= d^{\mathcal{H}}_{\infty,W}\left(\mathcal{C}_{V,W}(h), \mathcal{C}_{V,W}(g)\right), \\ d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h,g) &:= d^{\mathcal{H}}_{\infty,\Delta^{C-1}}\left(\mathcal{V}_C(h), \mathcal{V}_C(g)\right). \end{aligned}$$

The first distance $d^{\mathcal{H}}_{\mathcal{C}_{V,W}}$ compares the representations of h and g after all admissible nonlinear transformations, and quantifies how closely g can approximate h via such transformations.

The second distance $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}$ measures how differently two encoders behave under downstream classification tasks, modeled via post-composition with a classifier followed by softmax normalization.

Both distances capture extrinsic discrepancies: the former in terms of raw feature transformations, the latter in terms of final classification behavior.

Proposition 7.7 (Triangle Inequality for $d^{\mathcal{H}}_{\mathcal{C}_{V,W}}$)**.** *Let $h, g, f \in \mathcal{E}_V$. Then the following triangle inequality holds:*

$$d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(h,f) \le d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(h,g) + d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(g,f).$$

Proof. We prove the inequality using the definition of $d^{\mathcal{H}}_{\mathcal{C}_{V,W}}$ and the triangle inequality of the ∞-norm.

Let $\epsilon > 0$ be arbitrary. Fix any $\psi_h \in \mathcal{C}_{V,W}$.

By the definition of the infimum, there exists $\psi_g \in \mathcal{C}_{V,W}$ such that:

$$d_{\infty,W}(\psi_h\circ h, \psi_g\circ g) \le \inf_{\tilde{\psi}_g\in\mathcal{C}_{V,W}} d_{\infty,W}(\psi_h\circ h, \tilde{\psi}_g\circ g) + \epsilon.$$

Likewise, for this fixed ψ_g, there exists $\psi_f \in \mathcal{C}_{V,W}$ such that:

$$d_{\infty,W}(\psi_g\circ g, \psi_f\circ f) \le \inf_{\tilde{\psi}_f\in\mathcal{C}_{V,W}} d_{\infty,W}(\psi_g\circ g, \tilde{\psi}_f\circ f) + \epsilon.$$

Applying the triangle inequality of $d_{\infty,W}$, we obtain:

$$\begin{aligned} d_{\infty,W}(\psi_h \circ h, \psi_f \circ f) &\leq d_{\infty,W}(\psi_h \circ h, \psi_g \circ g) + d_{\infty,W}(\psi_g \circ g, \psi_f \circ f) \\ &\leq \inf_{\tilde{\psi}_g \in \mathcal{C}_{V,W}} d_{\infty,W}(\psi_h \circ h, \tilde{\psi}_g \circ g) + \inf_{\tilde{\psi}_f \in \mathcal{C}_{V,W}} d_{\infty,W}(\psi_g \circ g, \tilde{\psi}_f \circ f) + 2\epsilon. \end{aligned}$$

Now take the infimum over ψ_f on the left-hand side and then the supremum over all $\psi_h \in \mathcal{C}_{V,W}$ to obtain:

$$\begin{aligned} &d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(h, f) = \sup_{\psi_h \in \mathcal{C}_{V,W}} \inf_{\psi_f \in \mathcal{C}_{V,W}} d_{\infty,W}(\psi_h \circ h, \psi_f \circ f) \\ &\leq \sup_{\psi_h \in \mathcal{C}_{V,W}} \left(\inf_{\tilde{\psi}_g \in \mathcal{C}_{V,W}} d_{\infty,W}(\psi_h \circ h, \tilde{\psi}_g \circ g) + \inf_{\tilde{\psi}_f \in \mathcal{C}_{V,W}} d_{\infty,W}(\tilde{\psi}_g \circ g, \tilde{\psi}_f \circ f) \right) + 2\epsilon \\ &\leq \sup_{\psi_h \in \mathcal{C}_{V,W}} \inf_{\tilde{\psi}_g \in \mathcal{C}_{V,W}} d_{\infty,W}(\psi_h \circ h, \tilde{\psi}_g \circ g) + \sup_{\tilde{\psi}_g \in \mathcal{C}_{V,W}} \inf_{\tilde{\psi}_f \in \mathcal{C}_{V,W}} d_{\infty,W}(\tilde{\psi}_g \circ g, \tilde{\psi}_f \circ f) + 2\epsilon \\ &= d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(h, g) + d^{\mathcal{H}}_{\mathcal{C}_{V,W}}(g, f) + 2\epsilon. \end{aligned}$$

Since $\epsilon > 0$ was arbitrary, the result follows. □

Proposition 7.8. *The temperature-scaled softmax function*

$$\mathrm{softmax}_\lambda(z)_i := \frac{e^{\lambda z_i}}{\sum_{j=1}^n e^{\lambda z_j}}$$

is Lipschitz continuous with respect to the ℓ^∞ norm with Lipschitz constant at most $\frac{\lambda}{2}$.

Proof. The softmax function is continuously differentiable, and its Jacobian matrix $J(z) \in \mathbb{R}^{n \times n}$ has entries

$$J_{ij}(z) = \lambda \cdot s_i \cdot (\delta_{ij} - s_j), \quad \text{where } s_i = \mathrm{softmax}_\lambda(z)_i.$$

The row sums of the absolute values are

$$\sum_{j=1}^n |J_{ij}(z)| = \lambda \cdot s_i \left(|1 - s_i| + \sum_{j \neq i} s_j \right) = 2\lambda \cdot s_i(1 - s_i).$$

The product $s_i(1 - s_i)$ is maximized at $s_i = \frac{1}{2}$, hence

$$\sum_j |J_{ij}(z)| \leq \frac{\lambda}{2} \quad \text{for all } i.$$

Therefore, the operator norm satisfies $\|J(z)\|_\infty \leq \frac{\lambda}{2}$, and the function $\mathrm{softmax}_\lambda$ is $\frac{\lambda}{2}$ -Lipschitz with respect to the ℓ^∞ norm. □

Definition 7.9 (Extrinsic Relation)**.** *Let $h, g \in \mathcal{E}_V$. We define the extrinsic relation as:*

$$h \gtrsim_{\mathrm{Ext}} g \quad \Leftrightarrow \quad d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) = 0.$$

Proposition 7.10 (Extrinsic Preorder)**.** *Let $\mathcal{F}_{\mathrm{Lip}_1} \subset \mathcal{C}(V, W)$ denote the class of 1-Lipschitz continuous functions from V to W. Define the extrinsic approximation distance by*

$$d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) := \sup_{\psi \in \mathcal{F}_{\mathrm{Lip}_1}} \inf_{\varphi \in \mathcal{F}_{\mathrm{Lip}_1}} \|\psi \circ h - \varphi \circ g\|_\infty,$$

for $h, g \in \mathcal{E}_V$, where $\mathcal{E}_V \subset \mathrm{Map}(\Sigma^*, V)$ is the space of encoders into V.

Then $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}$ is a hemi-metric on $\mathcal{E}_V$, and the induced relation

$$h \gtrsim_{\mathrm{Ext}} g \quad \Longleftrightarrow \quad d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) = 0$$

defines a preorder on $\mathcal{E}_V$.

Proof. We show first, that $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}$ is a hemi-metric as defined in Definition 4.1.

H1: $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}} \geq 0$.

Since $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}$ is a metric, the Hausdorff-Hoare-map inherits non-negativity

$d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, h) = 0 \quad \forall h \in \mathcal{E}_V$.

For $h = g$ we have $\mathcal{C}_{V,W}(h) = \mathcal{V}(V, \Delta)(g)$ and the infimum becomes zero.

H2: Let $h, g, f \in \mathcal{E}_V$. We show that

$$d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, f) \leq d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f).$$

To do so, consider the following setup: Let $h \in \mathcal{E}_V$ be a language encoder $h : \Sigma^* \to V$, and let $\psi \in \mathcal{C}_{V,W}$ be a transformation $\psi : V \to W$. We define the composed map

$$p_\psi := \mathrm{softmax}_\lambda(\psi \circ h) : \Sigma^* \to \Delta^{C-1},$$

which can be interpreted as assigning probability distributions over C classes. This yields the compositional structure

$$\Sigma^* \xrightarrow{h} V \xrightarrow{\psi} W \xrightarrow{\mathrm{softmax}_\lambda} \Delta^{C-1}.$$

Since Proposition 7.8 shows, $\mathrm{softmax}_\lambda \colon \mathbb{R}^N \to \Delta^{N-1}$ is differentiable with bounded Jacobian entries, it is Lipschitz continuous with respect to the ℓ^∞-norm. In particular, there exists a constant $L > 0$ such that for all $x, y \in \mathbb{R}^N$,

$$\|\mathrm{softmax}_\lambda(x) - \mathrm{softmax}_\lambda(y)\|_\infty \leq L \cdot \|x - y\|_\infty.$$

This Lipschitz continuity extends to function composition: for any $\psi \in \mathcal{C}_{V,W}$, the composed function $\mathrm{softmax}_\lambda(\psi) \in \mathcal{C}_{V,\Delta^{C-1}}$ satisfies

$$\|\mathrm{softmax}_\lambda(\psi(h)) - \mathrm{softmax}_\lambda(\psi(g))\|_\infty \leq L \cdot \|\psi(h) - \psi(g)\|_\infty.$$

In particular, the Hausdorff–Hoare distances satisfy

$$d^{\mathcal{H}}_{\infty,\Delta^{C-1}}(\mathrm{softmax}_\lambda(\mathcal{C}_{V,W}(h)), \mathrm{softmax}_\lambda(\mathcal{C}_{V,W}(g))) \leq L \cdot d^{\mathcal{H}}_{\infty,W}(\mathcal{C}_{V,W}(h), \mathcal{C}_{V,W}(g)),$$

where the Hausdorff–Hoare distance is computed with respect to the supremums norm over V and the ℓ^∞-norm on the codomain.

As a result, the triangle inequality for $d^{\mathcal{H}}_{\mathcal{C}_{V,W}}$ implies the triangle inequality for the softmax-composed version,

$$d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, f) \leq d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f),$$

provided that $L = 1$. If $L > 1$, the inequality remains valid after rescaling the distance function accordingly.

Hence, $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}$ is a hemi-metric.

We show that this relation $\gtrsim_{\mathrm{Ext}}$ is a preorder:

Reflexivity: For all $h \in \mathcal{E}_V$, $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, h) = 0$, so $h \gtrsim_{\mathrm{Ext}} h$.

Transitivity: If $h \gtrsim_{\mathrm{Ext}} g$ and $g \gtrsim_{\mathrm{Ext}} f$, then

$$d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, f) \leq d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) + d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(g, f) = 0 + 0 = 0,$$

so $h \gtrsim_{\mathrm{Ext}} f$.

Based on Lemma 6.2, the relation $\gtrsim_{\mathrm{Ext}}$ defined by

$$h \gtrsim_{\mathrm{Ext}} g \quad \Longleftrightarrow \quad d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) = 0$$

is a preorder on the encoder space $\mathcal{E}_V$. This coincides with our definition of the extrinsic approximation relation $\gtrsim_{\mathrm{Ext}}$, as both are defined by the vanishing of the approximation distance $d_{\mathcal{F}_{\mathrm{Lip}_1}}$.

□

Definition 7.11 (Exact Extrinsic Nonlinear Homotopy)**.** *An encoder $h \in \mathcal{E}_V$ is said to be* exactly extrinsically non-linearly homotopic to $g \in \mathcal{E}_V$*, written*

$$h \gtrsim_{\textit{Ext}} g,$$

if

$$d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) = 0.$$

Note that this relation is generally not symmetric; it only expresses that h can be approximated arbitrarily well by applying admissible transformations to g, not necessarily the other way around. This directional behavior reflects the asymmetric notion of homotopy proposed in [2], where similarity is understood in terms of functional approximation from one model to another.

7.3 Performance-Based Similarity Measure

As explained in Chapter 5, functional similarity measures compare the outputs $\mathbf{O}, \mathbf{O}' \in \mathbb{R}^{N \times C}$, where each element $\mathbf{O}_{i,c}$ denotes the predicted probability or score for class c on input x_i. The predicted class label is given by

$$\hat{y}_i = \arg\max_c \mathbf{O}_{i,c}.$$

In this work, we focus exclusively on *classification agreement* as a performance-based similarity measure. This captures how often two models assign the same predicted label to the same input, independent of confidence scores or loss values.

Let $\mathbf{O}^{(h)}$ and $\mathbf{O}^{(g)}$ be the class score matrices of models h and g, respectively. Their predicted labels are defined as

$$\hat{y}_i^{(h)} = \arg\max_c \mathbf{O}_{i,c}^{(h)}, \quad \hat{y}_i^{(g)} = \arg\max_c \mathbf{O}_{i,c}^{(g)}.$$

The agreement score is then computed as

$$\text{Agreement}(h, g) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\left[\hat{y}_i^{(h)} = \hat{y}_i^{(g)}\right],$$

where $\mathbb{1}$ denotes the indicator function.

This simple yet interpretable metric reflects functional alignment in terms of model decisions and is well suited for classification tasks.

8 Practical Realization and Implementation

This chapter discusses the practical implementation of the theoretically formalized approaches from Chapter 7. Since we are dealing with a non-linear and non-convex bi-level structure, it is not possible to solve the optimization with classical methods. Instead, an iterative approach is employed: the lower-level problem is addressed via a neural network, while the upper-level problem is handled through implicit optimization and adaptation of the training process.

To ensure computational efficiency, especially in view of the large number of datasets, random seeds, and model comparisons, all core components of the experimental pipeline for intrinsic and extrinsic homotopy were implemented in `Python` and parallelized using Python's `multiprocessing` library. In particular, this parallelization was applied to the computation of similarity measures, the training of transformation networks, and the evaluation across tasks and model configurations.

In the following, we give an overview about the implementation that focuses on how models are loaded, processed, and evaluated for intrinsic, extrinsic and performance-based similarity.

8.1 Implementation of Loading the Models

Transformer-based language models such as `BERT`, `RoBERTa`, or `ELECTRA` process text by passing token embeddings through multiple layers of self-attention and feedforward operations. At each layer, the model computes intermediate representations for each input token, referred to as *hidden states*, which encode increasingly abstract and contextualized information.

Formally, let $x = (x_1, \ldots, x_n)$ be a tokenized input sequence of length n. For a given pretrained language model g, the hidden states at layer ℓ are denoted by

$$h^{(g,\ell)}(x) = \left(h_1^{(g,\ell)}, \ldots, h_n^{(g,\ell)}\right) \in \mathbb{R}^{n \times d},$$

where $h_i^{(g,\ell)} \in \mathbb{R}^d$ is the representation of token x_i, and d is the hidden size of the model.

To obtain a fixed-size vector representation per input and layer, we compute the mean over all tokens. This aggregation yields a fixed-size representation that enables model comparisons and downstream processing independent of input length.

$$h^{(g,\ell)} := \frac{1}{n} \sum_{i=1}^{n} h_i^{(g,\ell)} \in \mathbb{R}^d.$$

For notational convenience, we write $h^{(g)}(x)$ when the layer ℓ is fixed or clear from context.

To compute both extrinsic and intrinsic homotopy distances between different language encoders, we learn a nonlinear transformation between their internal representations.

Concretely, we train neural networks that map the hidden representations of one language model to those of another. Let $h \in \mathbb{R}^d$ denote a mean-aggregated hidden representation extracted from a pretrained language encoder for a given input from a General Language Understanding Evaluation (GLUE) task, which is described in detail in Section 9.1. Let $\psi : \mathbb{R}^d \to \mathbb{R}^{d'}$ be a neural network that learns to transform these representations into the feature space of another encoder.

Unless stated otherwise, we extract hidden states from the final Transformer layer, as it typically contains the most task-relevant and semantically rich information.

During training, we treat the hidden states of the source model as inputs and the corresponding hidden states of the target model (for the same textual input) as targets. That is, we train ψ such that

$$\psi(h^{(g)}(x)) \approx h^{(f)}(x),$$

where $h^{(g)}(x)$ and $h^{(f)}(x)$ denote the mean-aggregated hidden states produced by models g and f, respectively, for the same input x.

This approach enables a learned alignment between the internal representation spaces of two encoders. Once trained, the transformation ψ allows us to compare model representations across the GLUE tasks in terms of homotopy distances. The composition $\psi \circ h^{(g)}$ thus maps raw input text into the representation space of model f via a nonlinear transformation that approximates $h^{(f)}$. This composition forms the foundation for the extrinsic and intrinsic homotopy metrics introduced in Sections 7.1 and 7.2.

The following implementation extracts and stores hidden states from `MultiBERTs`, `RoBERTa` and `ELECTRA` models.

The `MultiBERTs` models differ only in their random initialization and are therefore processed individually. All models are publicly available via Hugging Face and pretrained. They are subsequently fine-tuned on classification tasks from the GLUE benchmark.

To analyze the internal behavior of these models, we first extract the hidden states from all transformer layers for each input sample across several GLUE tasks. This involves loading the respective model (`RoBERTa`, `ELECTRA`, or `MultiBERTs`), preparing the task-specific input data, and performing a forward pass to compute the hidden representations.

For each layer, the outputs are averaged across the token dimension to yield a fixed-size vector for each input. These representations, together with their associated labels, are saved for the training, validation, and test splits of the GLUE tasks. Although the classification heads of the models are customized with additional non-linear layers to increase expressiveness, our analysis focuses on the pre-classification activations, i.e., the internal representations learned by the transformer encoder itself.

The general procedure used for extracting and saving the hidden states is summarized in Algorithm 1.

Algorithm 1 Extraction of Hidden States and Labels for GLUE Tasks

procedure EXTRACTHIDDENSTATES(model, $\mathcal{L}$)
  $H, Y \leftarrow$ empty list
  for batch $\in \mathcal{L}$ **do**
    Move batch to device (CPU/GPU)
    $y \leftarrow$ batch.labels
    batch.labels $\leftarrow \mathbf{0}$ ▷ Dummy labels for forward pass
    $\hat{h} \leftarrow$ model(batch, output_hidden_states $=$ True)
    $h \leftarrow$ mean_pool($\hat{h}$.hidden_states) ▷ Mean pooling over tokens (per layer)
    H.append(h), Y.append(y)
  end for
  $H \leftarrow$ concatenate(H, axis $= 0$) ▷ Stack over batch dimension
  $Y \leftarrow$ concatenate(Y)
  return H, Y
end procedure

All transformer models used in this work consist of 12 layers in the encoder stack, followed by a task-specific classification head (the 13$^{\text{th}}$ layer). These models are fully fine-tuned and downloaded in their task-specific form via Hugging Face, meaning that both the encoder and the classification head are adapted to the downstream GLUE task.

After extracting the hidden representations for all training, validation, and test samples, we serialize the resulting tensors using PyTorch's built-in `torch.save` function. This enables efficient reuse of the representations in downstream tasks such as classification or similarity analysis.

The extraction and serialization process is repeated for each combination of model type (e.g., `BERT`, `RoBERTa`), random seed (in the case of `MultiBERTs`), and GLUE task (e.g., SST-2, MNLI).

The corresponding script is fully configurable via command line arguments and supports features such as caching, adjustable batch sizes, and dynamic model selection.

8.2 Practical Approximation of Intrinsic Homotopy

In Chapter 7, we extended the concept of intrinsic homotopy to a family of non-linear models and transformations. In this context, we consider the distance map between two language encoders $g, h \in \mathcal{E}_V$ defined in Definition 7.1 as

$$d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) = \inf_{\psi \in \mathcal{F}_{\mathrm{Lip}_1}} \|h - \psi(g)\|_\infty,$$

where $\mathcal{F}_{\mathrm{Lip}_1} \subset \mathcal{C}_{V,W}$ denotes a subset of continuous functions, restricted in our case to non-linear neural networks.

Building on the theoretical considerations in Section 7.1 and following the affine setting analyzed by Chan et al. [2], we constrain our analysis to *non-linear 1-Lipschitz continuous neural networks*. This restriction ensures functional comparability and regularity of the learned transformations. The transformation $\psi \in \mathcal{F}_{\mathrm{Lip}_1}$ is thus implemented as a neural network with architectural constraints that tries to enforce the Lipschitz property.

This setup allows us to compare encoder representations $g(x)$ and $h(x)$ by asking whether one can be mapped into the other in a structure-preserving way. If such a mapping exists in both directions (i.e., $h \approx \psi(g)$ and $g \approx \psi^{-1}(h)$), we refer to g and h as *exactly intrinsically affinely homotopic*.

Importantly, this notion of similarity is defined independently of any particular downstream task. Therefore, we analyze the final hidden layer representation of each language encoder, as it captures the model's abstract understanding before task-specific heads are applied.

Model Architecture. To estimate the intrinsic homotopy distance, we train transformation models that map representations from one encoder space into another. For each pair of encoders g and h, we learn a mapping ψ such that

$$\psi(g(x)) \approx h(x).$$

We consider two types of transformation models:

- a linear mapping (`LinearMap`), replicating the affine baseline from Chan et al. [2] and
- a non-linear neural network (`NonLinearNetwork`) composed of fully-connected (dense) layers with ReLU activations. Each layer is equipped with spectral normalization to ensure a controlled Lipschitz constant (i.e., ≤ 1).

This architectural flexibility enables controlled comparisons between linear and non-linear function classes under Lipschitz constraints.

Data Handling and Model Input. Each transformation model operates on precomputed hidden state representations $g(x)$ and $h(x)$, extracted from the final encoder layer for each input sample. These representations are stored on disk and loaded via `DataLoader` objects during training and evaluation. Training is supervised using the Huber loss.

Training Procedure. Models are trained using the Adam optimizer, as shown in Listing 2.1 with early stopping based on validation loss. Training is performed on the same predefined training split that was previously used for extracting hidden representations, with the corresponding validation split used to monitor convergence. We explore multiple learning rates and record the corresponding loss trajectories. To ensure reproducibility across random seeds and tasks, all training runs are logged using process-specific log files.

Evaluation. As defined in Section 7, the intrinsic homotopy distance is the infimum of the ℓ^∞-norm between $h(x)$ and $\psi(g(x))$ over all admissible transformations $\psi \in \mathcal{F}_{\mathrm{Lip}_1} \subset \mathcal{C}_{V,W}$. After training, the learned transformation ψ^* approximates this infimum within the class of nonlinear 1-Lipschitz neural networks.

To evaluate the quality of this approximation, we compute the maximum deviation between the transformed and target representations over a held-out test set:

$$\max_{x \in \mathcal{D}_{\text{test}}} \|h(x) - \psi^*(g(x))\|_\infty.$$

This value serves as a practical estimate of the intrinsic homotopy distance.

In addition, we compute the Pearson and Spearman correlation coefficient between the transformed and target representations over the test set. These metrics capture whether the transformation ψ^* preserves linear or monotonic structure between the input and output representations:

- The *Pearson correlation coefficient* measures the linear correlation between components of $h(x)$ and $\psi^*(g(x))$.
- The *Spearman rank correlation* captures the strength of any monotonic (not necessarily linear) relationship between the two vectors.

High correlation values indicate that ψ^* not only minimizes pointwise distances but also aligns structural trends in the representation space.

Formally, the *Pearson correlation coefficient* between the target and predicted representations is defined as

$$\rho_{\text{Pearson}} = \frac{\mathrm{Cov}(h(x), \psi^*(g(x)))}{\sigma_{h(x)}\, \sigma_{\psi^*(g(x))}},$$

where $\mathrm{Cov}(\cdot,\cdot)$ denotes the empirical covariance and σ the standard deviation computed over the test set.

The *Spearman rank correlation* is defined analogously, but computed on the rank-transformed data:

$$\rho_{\text{Spearman}} = \rho_{\text{Pearson}}(\text{rank}(h(x)), \text{rank}(\psi^*(g(x)))).$$

Code 8.1: Computation of $\|h(x) - \psi(g(x))\|_\infty$ in PyTorch

```
with torch.no_grad():
    for g_batch, h_batch in test_dataloader:
        output = model(g_batch)
        diff = h_batch - output
        dist = torch.max(torch.abs(diff)) # ||h - \psi(g)||_\infty
        all_dist.append(dist.item())
```

The maximum deviation across all batches is then used as an estimate for the distance:

```
overall_max = max(all_dist)
```

This value corresponds to the right-hand side of Definition 7.1, and serves as an empirical approximation to the intrinsic homotopy distance $d_{\mathcal{F}_{\text{Lip}_1}}(g, h)$.

In addition, we compute the following metrics to enable a quantitative comparison between linear and non-linear transformations (see Section 9.2 for detailed analysis):

- the Spearman rank correlation between predicted and target representations,
- the Pearson correlation between input and output representations (averaged over all features),
- an upper bound on the Lipschitz constant via the product of spectral norms of the network's weight matrices,

All hyperparameters, including learning rate, number of epochs, batch size, and network type, are configurable via command-line arguments. Results are stored in structured directories, and summary metrics are exported in JSON format for subsequent analysis.

8.3 Practical Approximation of Extrinsic Homotopy

In Chapter 7, we extended the concept of extrinsic homotopy to a family of non-linear models and transformations. In this setting, we aim to quantify the dissimilarity between two language encoders $g, h \in \mathcal{E}_V$ by approximating the Hausdorff–Hoare distance:

$$d^{\mathcal{H}}_{\mathcal{F}_{\text{Lip}_1}}(g, h) = \sup_{\psi \in \mathcal{C}_{V,W}} \inf_{\varphi \in \mathcal{C}_{V,W}} \|\psi \circ h - \varphi \circ g\|_\infty.$$

Following Section 7.2, we restrict the hypothesis space $\mathcal{C}_{V,W}$ to the subclass $\mathcal{F}_{\text{Lip}_1} \subset \mathcal{C}_{V,W}$ of non-linear 1-Lipschitz continuous neural networks. This choice, motivated by

prior work in the affine case [2], ensures bounded sensitivity and comparability across models.

Since the resulting minimax problem is non-linear and non-convex, no global optimum can be guaranteed in practice. Inspired by adversarial training strategies in generative models [13], we approximate the solution via alternating optimization.

Adversarial Optimization Scheme. Let $\psi, \varphi \in \mathcal{F}_{\mathsf{Lip}_1}$ denote two neural networks modeling the transformations of the source and target encoders, respectively. We employ the following alternating training strategy:

- In the *inner loop*, we fix ψ and train φ to minimize the discrepancy $\|\psi(h(x)) - \varphi(g(x))\|_\infty$.
- In the *outer loop*, we fix ψ and update φ via gradient ascent to increase the discrepancy to the current approximation $\psi(g(x))$.

This approach mirrors the generator–discriminator dynamics in GAN training, where alternating updates push the system toward an equilibrium. An overview is given in Algorithm 2.

Algorithm 2 Alternating Training of Neural Networks ψ and φ

Initialize neural networks ψ, φ, and their optimizers
for each training epoch **do**
  for each batch index i, and inputs x_g, x_h **do**
    if $i \bmod 2 = 0$ **then** ▷ Even batch: train ψ
      $\hat{y}_\psi \leftarrow \psi(x_g)$
      $y_\varphi \leftarrow \varphi(x_h)$.detach()
      $\mathcal{L}_\psi \leftarrow \mathsf{MSE}(\hat{y}_\psi, y_\varphi)$
      Perform gradient **ascent** on $-\mathcal{L}_\psi$
    else ▷ Odd batch: train φ
      $\hat{y}_\varphi \leftarrow \varphi(x_h)$
      $y_\psi \leftarrow \psi(x_g)$.detach()
      $\mathcal{L}_\varphi \leftarrow \mathsf{MSE}(\hat{y}_\varphi, y_\psi)$
      Perform gradient **descent** on $\mathcal{L}_\varphi$
    end if
  end for
end for

Model Architecture and Training. Both ψ and φ are implemented as fully connected neural networks with spectral normalization applied to all linear layers to enforce 1-Lipschitz continuity. Training is conducted using the Adam optimizer with early stopping based on validation loss. Alternating updates are controlled via mini-batch indices: even-numbered batches update ψ, odd-numbered batches update φ. To avoid gradient leakage between updates, we detach the target model's output using `.detach()` during each step.

Evaluation. After training, the extrinsic homotopy distance is approximated by the maximum deviation between the transformed outputs:

$$d^{\mathcal{H}}_{\mathcal{F}_{\mathsf{Lip}_1}}(g,h) \approx \max_{x \in \mathcal{D}_{\text{test}}} \|\psi(h(x)) - \varphi(g(x))\|_\infty.$$

This value is computed over the test set using `torch.max(torch.abs(...))` (see Listing 8.1). It serves as a practical estimate of the extrinsic homotopy distance under the considered model class $\mathcal{F}_{\mathsf{Lip}_1}$.

Additional Metrics. As in the previous subsection, we provide a more comprehensive view of representational alignment and to enable comparisons between extrinsic homotopy similarity and performance similarity in the following experimet in Section 9.3, we additionally compute:

- Spearman rank correlation between $\psi(h(x))$ and $\varphi(g(x))$,
- Pearson correlation (averaged over dimensions),
- Upper bound on the Lipschitz constant via the product of spectral norms.

8.4 Estimation of the Lipschitz Constant

In the proofs of the triangle inequality in Chapter 7, we required the Lipschitz continuity of the neural networks involved. Therefore, it is necessary to confirm that the trained neural networks are indeed Lipschitz continuous.

Since neural networks are defined as compositions of multiple layers that map inputs to outputs, we cannot evaluate their Lipschitz continuity analytically in general, but only approximate or bound it. In the following, we consider both upper and lower bounds on the Lipschitz constant, based on the work by Geuchen et al. [11].

In Section 2.2, we defined neural networks as compositions of the form

$$\psi = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(0)},$$

where each layer is given by $f^{(i)}(x) = \sigma(W_i x + b_i)$, and the activation functions σ are assumed to be Lipschitz continuous. In particular, ReLU networks are Lipschitz continuous as compositions of Lipschitz continuous functions.

Let $\psi \in \mathcal{C}_{V,W}$ be a neural network that maps between two language encoders, and let $x, y \in V$. Denote the Lipschitz constant of each layer $f^{(i)}$ by L_i. By recursively applying the definition of Lipschitz continuity, we obtain the following upper bound:

$$\|\psi(x) - \psi(y)\| \leq L_L \cdot L_{L-1} \cdots L_0 \cdot \|x - y\|.$$

Hence, the product of the individual layer constants provides an upper bound on the global Lipschitz constant of the network.

To obtain a lower bound, we consider the basic definition of the Lipschitz constant L for a function $\psi \in \mathcal{C}_{V,W}$:

$$\|\psi(x) - \psi(y)\| \leq L\|x - y\| \quad \Longleftrightarrow \quad \frac{\|\psi(x) - \psi(y)\|}{\|x - y\|} \leq L.$$

Taking the supremum over all $x \neq y$ yields:

$$\sup_{x \neq y} \frac{\|\psi(x) - \psi(y)\|}{\|x - y\|} \leq L.$$

If ψ is differentiable, the local Lipschitz behavior around a point x is governed by the Jacobian $J_\psi(x)$. The operator norm of the Jacobian gives a local estimate of the Lipschitz constant:

$$\lim_{\|h\| \to 0} \frac{\|\psi(x + h) - \psi(x)\|}{\|h\|} = \|J_\psi(x)\|.$$

In the case of the Euclidean norm, the induced operator norm is the spectral norm, defined as the largest singular value of the Jacobian:

$$\|J_\psi(x)\|_2 = \sup_{\|v\|=1} \|J_\psi(x)v\|_2.$$

Consequently,

$$\sup_x \|J_\psi(x)\|_2 \leq L,$$

which provides a lower bound on the global Lipschitz constant L.

In practice, computing a tight lower bound on the Lipschitz constant via the spectral norm of the Jacobian over the entire domain is computationally expensive, especially in high-dimensional settings.

An alternative approach for estimating the Lipschitz constant is presented by Wood et al. [36]. The key idea is to approximate the supremum of all slopes

$$\frac{\|g(x) - g(y)\|}{\|x - y\|},$$

for distinct points x, y in the domain of the function g, where g denotes the trained neural network. The quantity

$$\frac{\|g(X) - g(Y)\|}{\|X - Y\|}$$

can be interpreted as a random variable when X and Y are sampled uniformly from the domain. This induces a cumulative distribution function F, which we refer to as the *slope distribution* of g. The Lipschitz constant is then equal to the supremum of the support of F.

To approximate this distribution, one repeatedly samples pairs (X, Y) from the domain and computes the corresponding slopes. By accumulating these values and computing their maximum over many repetitions, an empirical estimate of the Lipschitz constant is obtained.

However, in our setting, this method turned out to be computationally expensive and yielded unrealistically small estimates. This is due to the fact that the model's predictions tend to be relatively flat across large regions of the input space, which causes the majority of sampled slopes to underestimate the true Lipschitz constant. As a result, these empirical estimates were not informative, and we therefore rely solely on the analytically computable upper bound as a conservative estimate.

As such, we restrict ourselves to estimate an upper bound based on the product of the layer wise Lipschitz constant, which is easy to compute and provides a guaranteed over-approximation of the true Lipschitz constant. However, this upper bound can be quite loose in practice and should be interpreted as a worst-case estimate rather than a tight characterization of the network's sensitivity.

8.5 Implementation of Performance-Based Functional-Similarity Measure

To analyze the performance-based similarity measure (see Section 7.3), we evaluate how well different language encoders support nonlinear classification tasks. The goal is to assess how informative the internal representations of each model are for downstream prediction.

To this end, we train a neural classifier on top of the hidden state representations produced by each encoder. The architecture of the classifier is shown in Listing 2.1. It takes as input the fixed-size hidden representations as represented in Algorithm 1 (obtained via mean pooling or another aggregation strategy) and the corresponding class labels.

These hidden representations and labels are obtained by running the input data through the respective language encoder. During training, we record the predicted class outputs and compare them to the ground-truth labels using standard classification metrics, such as accuracy and Spearman rank correlation.

Once the classifier is trained for each model, we evaluate its performance on a held-out test set. By comparing the classification performance across models on the same task, we obtain a coarse-grained similarity signal: if two models achieve similar performance, their internal representations may be considered functionally similar from a task-oriented perspective.

To refine this view, we additionally compare the actual predictions made by the classifiers. Specifically, we compute:

- the percentage of agreeing predictions between two models on the test set, and
- the Spearman rank correlation between their prediction confidence scores.

These metrics allow us to assess not only how well each model performs individually, but also how similarly they behave in terms of decision-making. High prediction alignment, beyond accuracy, can indicate that two models encode similar decision boundaries or representational structures.

This comparison forms the basis for our performance-based similarity metric (see Section 7.3), which integrates both task-level accuracy and prediction-level agreement.

9 Practical Experiments on Nonlinear Transformation

This chapter is structured around three successive experimental components, each designed to examine a specific aspect of the research questions.

We begin by outlining the experimental setup, including the selected models, datasets, and training details for the transformation networks.

The subsequent sections are organized as follows:

- In the intrinsic homotopy experiment, we compare the quality of non-linear transformations against the original affine approach.
- In the extrinsic homotopy experiment, we investigate how extrinsic similarity aligns with predictive agreement between models, and whether good transformations can be learned when model agreement is high.
- Finally, we conduct a comparative analysis of intrinsic and extrinsic distances to assess whether structural and behavioral similarity are consistently related in both the affine and non-linear settings.

Together, these experiments provide a comprehensive empirical evaluation of the extended homotopy framework introduced in this thesis.

9.1 Experimental Setup

This section describes the experimental setup used to evaluate homotopy-based similarity between language encoders. We first introduce the selected models. Subsequently, we describe the datasets and training objectives used for learning the transformations, and provide an overview of the computational effort required for training.

Models and Representations. To investigate homotopic relationships between language models, we evaluate three families of transformer-based encoders: `ELECTRA` [4], `RoBERTa` [26], and the 25 variants of `MultiBERTs` [30]. For each model, we extract contextualized representations from all 12 transformer layers using pretrained weights

available on HuggingFace.[1] Each hidden state is a 768-dimensional vector per input token. To obtain fixed-size input representations for the transformations, we apply mean pooling across tokens.

Datasets. We use six classification tasks from the General Language Understanding Evaluation (GLUE) benchmark [33], a standard suite for evaluating language understanding. These tasks span diverse linguistic phenomena and vary in size and difficulty. The selected tasks are:

- **Stanford Sentiment Treebank** `(SST-2)`: A binary sentiment classification task in which the model must predict whether a given sentence expresses positive or negative sentiment.
- **Microsoft Research Paraphrase Corpus** `(MRPC)`: A binary classification task to determine whether two given sentences are semantically equivalent (i.e., paraphrases of each other).
- **Recognizing Textual Entailment** `(RTE)`: A task where the model must decide whether a given premise entails a hypothesis.
- **Corpus of Linguistic Acceptability** `(CoLA)`: A grammatical acceptability task in which the model predicts whether a given English sentence is grammatically correct.
- **Multi-Genre Natural Language Inference** `(MNLI)`:

  A Three-way classification task to determine whether a hypothesis is entailed by, contradicts, or is neutral with respect to a given premise.
- **Quora Question Pairs** `(QQP)`: A binary paraphrase detection task on question pairs from Quora, asking whether two questions have the same meaning.

Following best practices in empirical research, we report the specific splits used for training, validation, and testing to facilitate reproducibility.

Table 9.1: GLUE datasets used in the experiments.

Task	Train	Validation	Test	License
SST-2	67,349	872	1,821	CC0: Public Domain
MRPC	3,668	408	1,725	Apache License 2.0
RTE	2,490	277	3,000	CC BY 3.0
CoLA	8,551	1,043	1,063	CC BY-SA 4.0
MNLI	392,702	9,815	9,796	GPL
QQP	363,846	40,430	390,965	Custom (non-commercial)

[1] `https://huggingface.co/`

Training Details Each transformation model was trained independently for a specific encoder pair and task. For each configuration, we ran experiments across multiple learning rates to select the best-performing model. Early stopping was applied to prevent overfitting and reduce unnecessary computation: training was terminated if the validation loss did not improve for 20 consecutive epochs.

Table 9.2: Training hyperparameters used across all experiments.

Parameter	Value
Optimizer	Adam [19]
Learning Rates	{1e-5, 1e-4, 1e-3, 1e-2, 1e-1}
Batch Size	8
Epochs	Up to 100
Early Stopping	Patience = 20 epochs (validation loss)
Loss Functions	MSELoss, HuberLoss (task-dependent)

Hardware and Runtime. All experiments were conducted on NVIDIA DGX A100 systems. All computations were executed inside a dedicated Docker container to ensure consistent software environments across runs. The hidden state extraction was performed on a single GPU and took approximately 6 hours in total. The training time for the transformation models varied depending on the task:

- Nonlinear intrinsic transformation models (with spectral normalization) took up to **37 hours** due to increased model complexity.
- Extrinsic transformation models, trained adversarially, completed in approximately **7 hours**.
- Classification models for functional similarity required roughly **30 minutes** per run.

9.2 Experiment on Intrinsic Homotopy

To assess the effectiveness of our extended framework, we begin by replicating the intrinsic homotopy experiment from Chan et al. [3]. Our goal is to directly compare the representational alignment achieved by affine and non-linear transformations.

We aimed to reproduce the results reported in the original work as closely as possible. In a first attempt, we applied the architecture and loss computation as described in the original paper, and used the train data for training and test data for evaluation. However, this setting did not yield meaningful approximation of the target representations. In a second attempt, we trained the transformations on the training set, again using the original architecture and per-sample maximum loss. This setup led to a close reproduction of the results reported in the paper: for most tasks, we observed very similar approximation distances on the training set. The only deviations were slightly lower

distances for `SST-2` and slightly higher values for `RTE`, as measured by the median. Figure 9.1 confirms that our reimplementation closely matches the reported training performance

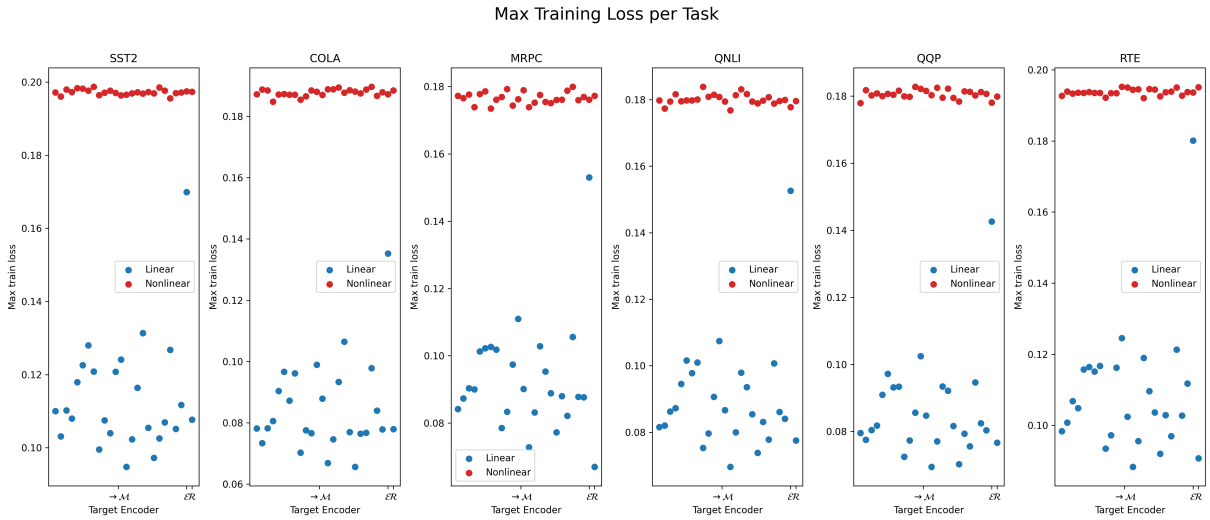


Figure 9.1: Intrinsic approximation distances on the training set for affine and non-linear transformations under the original training scheme.

To ensure a rigorous and transparent evaluation protocol for the subsequent experiments, we now consistently train on the training split, use early stopping based on validation loss, and report final results on a held-out test set.

After verifying that our implementation reproduces the reported approximation behavior on the training set, we next examine the intrinsic homotopy distances across a range of learning rates for both affine and non-linear transformations.

In Section 6.1, we formalized the concept of intrinsic homotopy using linear mappings $\psi \in S \subset \mathrm{Aff}(V)$, and in Section 7.1 extended this framework to the nonlinear case $\psi \in \mathcal{F}_{\mathrm{Lip}_1} \subset \mathcal{C}_{V,W}$, where $\mathcal{F}_{\mathrm{Lip}_1} \subset \mathcal{C}$ consists of 1-Lipschitz continuous neural networks.

To empirically compare the representational alignment induced by these transformation classes, we compute the intrinsic homotopy distances $d_{\mathrm{Aff}} := d_{\mathsf{Aff}}(h, g)$ and $d_{\mathcal{F}_{\mathrm{Lip}_1}} := d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g)$ for each pair of encoders $h, g \in \mathcal{E}_V$. The mean of all distances is visualized for each learningrate and task in Figure 9.2.

In addition to the homotopy distances, we report Spearman's ρ and Pearson's r correlations between predicted and target representations for both transformation regimes in Table 9.3. These metrics serve as complementary indicators of representational alignment: while the homotopy distance measures global approximation quality, the correlation coefficients quantify local relational consistency between vector pairs.

Spearman's ρ captures the rank-order correlation between the transformed representation $\psi(g)$ and the target h. As a rank-based measure, it is robust to monotonic nonlinear distortions and reflects whether the relative structure between examples is preserved.

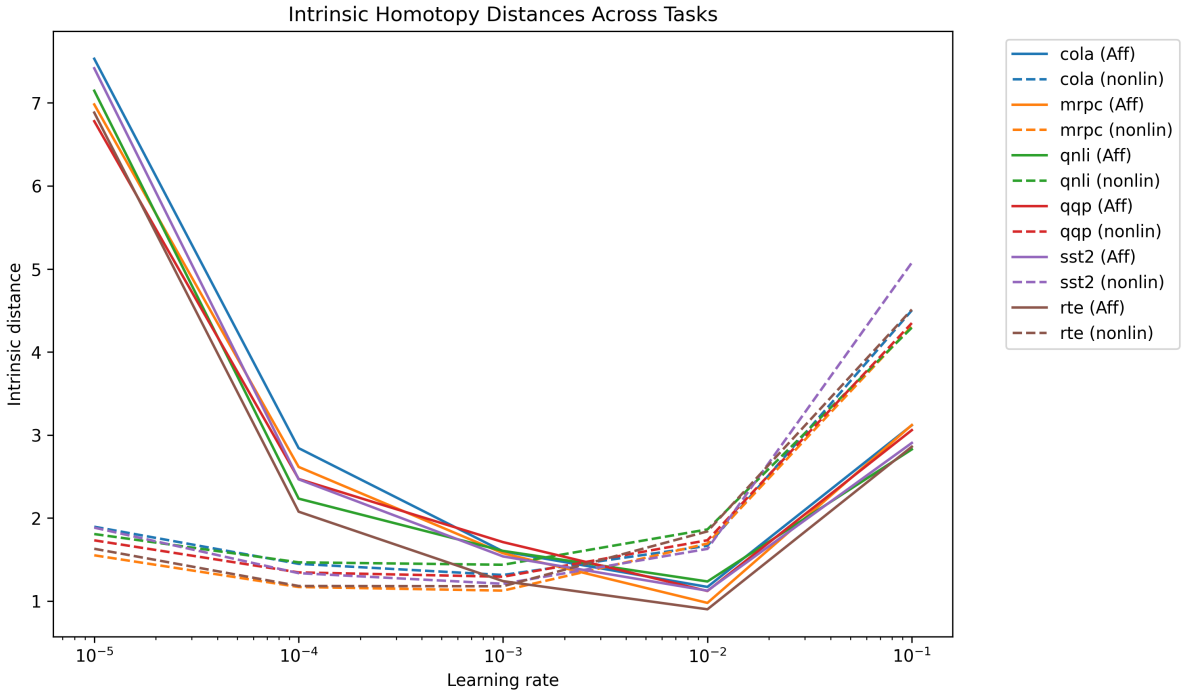


Figure 9.2: Intrinsic Distances across tasks and learning rates for linear and non-linear models

In contrast, Pearson's r quantifies the linear correlation between vectors. To assess whether a meaningful linear mapping between model representations exists, we compute $r(g, h)$, the Pearson correlation between the hidden states of the source and target models. Consistently low values of $r(g, h)$ across all tasks indicate that no strong global linear relationship exists between g and h, implying that affine transformations are fundamentally limited in their ability to approximate one encoder from another.

While we include Pearson's r for interpretability, we do not use it as a basis for comparing model performance across tasks.

Considering Figure 9.2 and Table 9.3, we observe that non-linear models consistently achieve lower intrinsic distances than their affine counterparts, i.e.,

$$d_{\mathcal{F}_{\mathrm{Lip}_1}} < d_{\mathrm{Aff}},$$

especially at moderate learning rates such as $\mathrm{lr} = 10^{-3}$. This indicates that the class of 1-Lipschitz neural networks $\mathcal{F}_{\mathrm{Lip}_1} \subset \mathcal{C}_{V,W}$ offers greater expressivity and better captures the structure-preserving mappings between encoder representations.

Interestingly, Spearman correlation ρ tends to be higher for affine transformations, suggesting that they preserve the rank ordering of feature dimensions more faithfully than their non-linear counterparts. This highlights a trade-off: non-linear models achieve tighter approximations in terms of distance, but may introduce distortions in the internal feature geometry.

This highlights a trade-off: non-linear models achieve tighter approximations in terms of distance, but may introduce distortions in the internal feature geometry.

Table 9.3: Correlation scores ρ, r across tasks and learning rates for linear and non-linear models.

	lr	QNLI	CoLA	MRPC	QQP	RTE	SST-2
ρ_{Aff}	$1 \cdot 10^{-5}$	0.44	0.29	0.47	0.44	0.44	0.32
	$1 \cdot 10^{-4}$	0.48	0.33	0.51	0.48	0.47	0.35
	$1 \cdot 10^{-3}$	0.47	0.33	0.51	0.47	0.47	0.35
	$1 \cdot 10^{-2}$	0.33	0.23	0.36	0.34	0.32	0.23
	$1 \cdot 10^{-1}$	0.089	0.058	0.089	0.094	0.080	0.054
$\rho_{\mathcal{C}}$	$1 \cdot 10^{-5}$	0.089	0.11	0.12	0.11	0.11	0.10
	$1 \cdot 10^{-4}$	0.079	0.13	0.18	0.10	0.12	0.11
	$1 \cdot 10^{-3}$	0.026	0.079	0.0054	0.066	0.021	0.071
	$1 \cdot 10^{-2}$	0.014	0.061	0.0037	0.043	0.012	0.049
	$1 \cdot 10^{-1}$	–0.008	–0.018	–0.0046	–0.018	–0.0051	–0.023
r		$5 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$3 \cdot 10^{-6}$	$5 \cdot 10^{-6}$

As discussed earlier, the Pearson correlation between source and target representations remains close to zero across all tasks, underscoring the absence of any strong linear alignment between encoder representations. This further explains the limited performance of affine models and their inability to approximate a transformation across the models.

At high learning rates (e.g., lr $= 10^{-1}$), performance degrades in both model classes, with non-linear models exhibiting greater instability due to their higher complexity and non-convex loss landscape. Affine models remain more robust under such conditions but are fundamentally limited in their capacity to approximate complex transformations.

Now that we have gained a general understanding of the model behavior, we turn to the core concept of intrinsic homotopy.

To this end, we analyze two sets of heatmaps: one based on affine transformations (Figure 9.3) and one on non-linear transformations (Figure 9.4). Model pairs classified as intrinsically homotopic, i.e., with intrinsic distance below threshold, are highlighted with circles in both plots.

Since the MRPC task yielded the lowest overall approximation distances across learning rates, we focus our comparative analysis on this benchmark and restrict attention to the learning rates 10^{-5}, 10^{-4}, 10^{-3}, and 10^{-2}.

Recall that intrinsic homotopy is defined by the relation

$$h \gtrsim_{\text{Intr}} g \quad \Leftrightarrow \quad d_{\mathcal{F}_{\text{Lip}_1}}(h, g) = 0,$$

where $d_{\mathcal{F}_{\text{Lip}_1}}$ denotes the intrinsic distance.

In practice, however, the models operate on 768-dimensional hidden representations, and due to numerical and optimization limitations, exact zeros are generally not achiev-

able. We therefore treat all distances below a threshold of 1 as effectively zero and consider such model pairs to be homotopic.

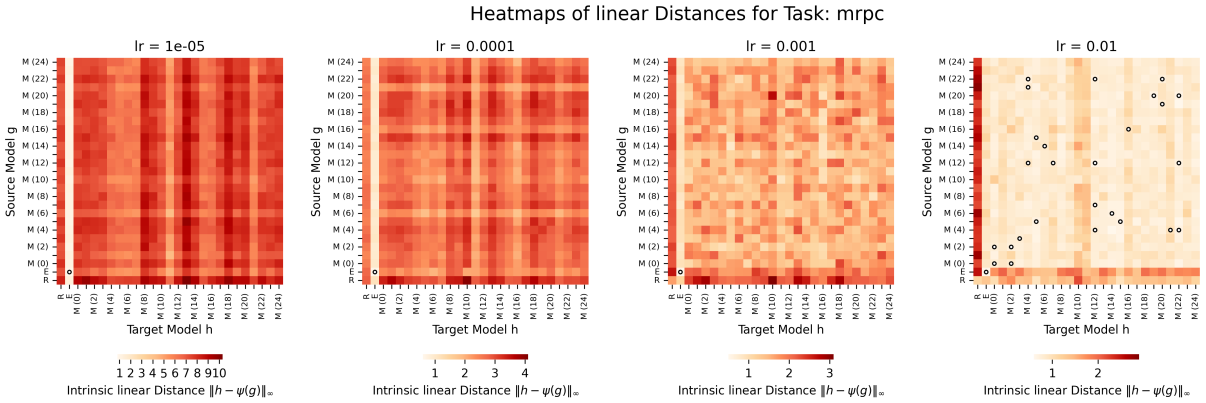


Figure 9.3: Heatmaps of intrinsic distances for affine models on MRPC across learning rates. Pairs with intrinsic distance < 1 are circled.

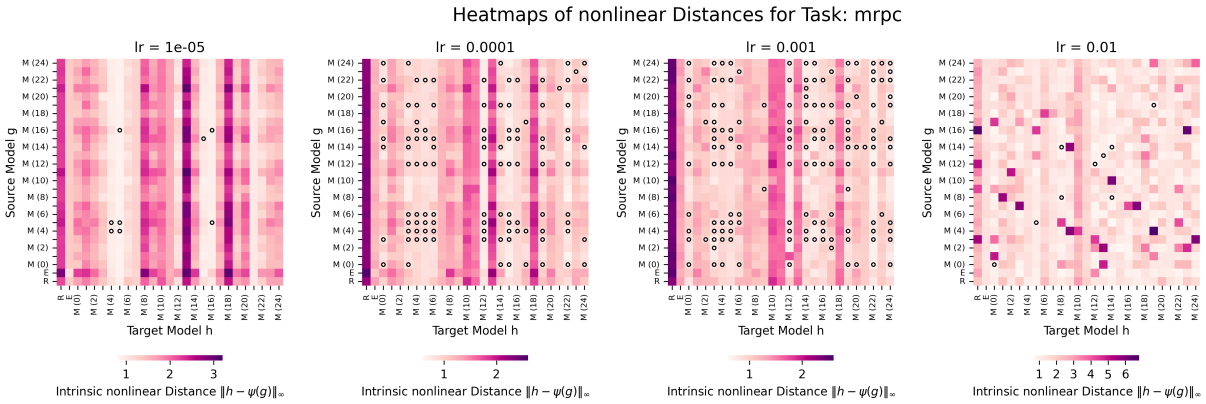


Figure 9.4: Heatmaps of intrinsic distances for non-linear models on MRPC across learning rates. Pairs with intrinsic distance < 1 are circled.

Across all learning rates, we observe that non-linear transformations yield substantially more homotopic model pairs, especially at lower learning rates. This supports our previous findings from Figure 9.2 and Table 9.3: non-linear models not only achieve smaller average distances but also induce denser and more coherent homotopy structures among encoders.

At higher learning rates, however, both model classes exhibit fewer such pairs, reflecting training instability and diminished approximation quality. This reinforces the earlier observation that effective representation alignment critically depends on proper learning rate tuning, particularly for complex transformation classes such as $\mathcal{F}_{\mathrm{Lip}_1}$.

9.3 Experiment on Extrinsic Homotopy and Performance Similarity

To complement our analysis of intrinsic alignment, we now investigate how well model representations align in terms of task-specific outcomes via *extrinsic homotopy*.

In Section 7.2, we defined the extrinsic distance

$$d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g) = \sup_{\psi \in \mathcal{C}_{V,W}} \inf_{\varphi \in \mathcal{C}_{V,W}} \| \operatorname{softmax}_\lambda(\psi \circ g) - \operatorname{softmax}_\lambda(\varphi \circ h) \|_\infty,$$

which measures how well a model h can be approximated on the output probability space Δ^{N-1} via admissible nonlinear transformations of another model g.

To assess this alignment from a task-based perspective, we compare the extrinsic homotopy distance $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(g, h)$ with classification agreement as introduced in Section 7.3.

We first analyze the extrinsic homotopy distances across the `MRPC` benchmark for different learning rates. For each pair of models h and g, we visualize the extrinsic distance in a heatmap, shown in Figure 9.5. In addition, we provide corresponding heatmaps for prediction agreement in Figure 9.6.

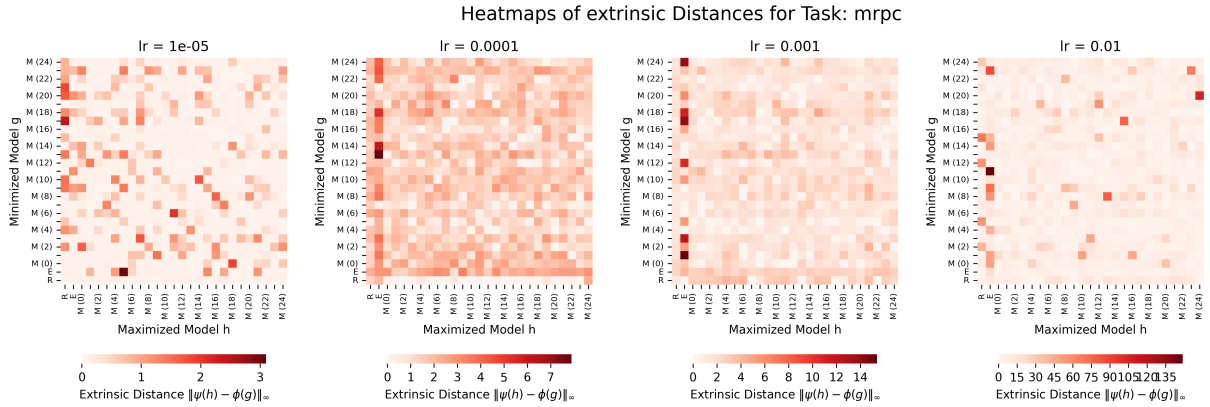


Figure 9.5: Heatmap of extrinsic nonlinear models on `MRPC` across learning rates.

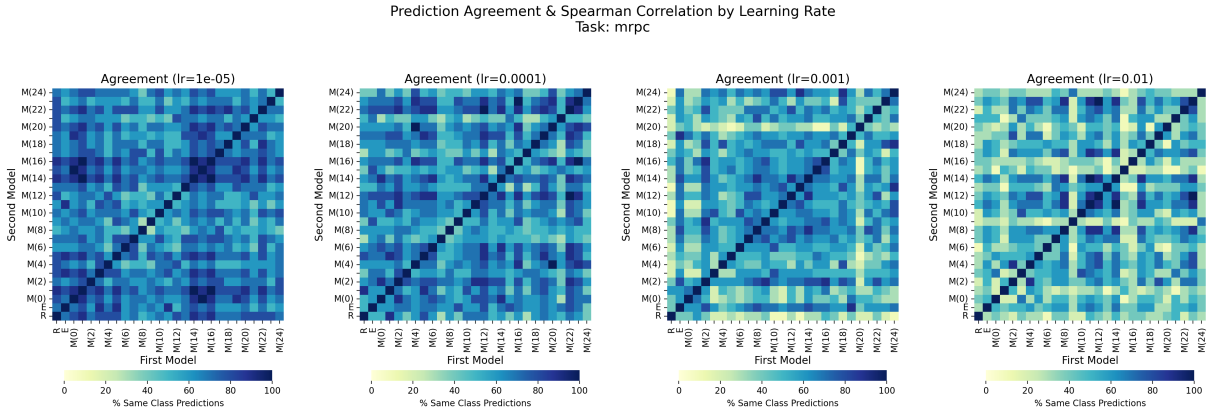


Figure 9.6: Heatmap of performance based models on `MRPC` across learning rates

Figure 9.6 shows the prediction agreement between all model pairs on the MRPC benchmark, while Figure 9.5 visualizes the corresponding extrinsic homotopy distances.

At moderate learning rates (e.g., lr $= 10^{-3}$), we observe both high agreement and low extrinsic distances, indicating functional similarity among models that generalize similarly. In contrast, at very low or high learning rates, prediction agreement becomes less structured, and extrinsic distances increase, suggesting instability in training or divergent representational behavior.

Interestingly, even in cases with low agreement, extrinsic distances may remain small, which reflects the power of non-linear transformations to align models that produce dissimilar predictions.

Finally, the asymmetry in the distance maps reflects the directional nature of the extrinsic distance definition, where model h is approximated via transformations of model g, but not necessarily vice versa.

We now turn to the analysis of extrinsic homotopy and performance-based similarity on the `SST-2` task. Figure 9.7 shows heatmaps of extrinsic distances across different learning rates, and Figure 9.8 displays the corresponding prediction agreement and Spearman correlation matrices.

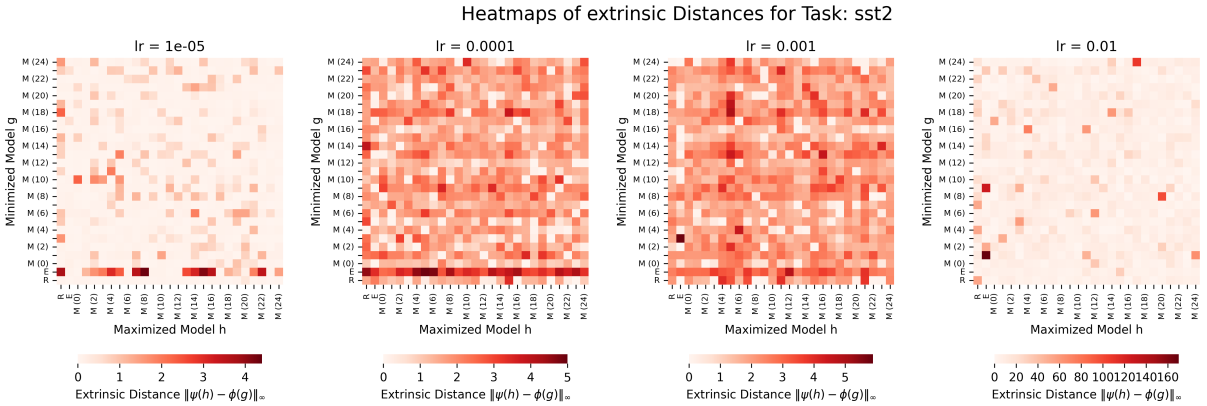


Figure 9.7: Heatmaps of extrinsic nonlinear distances on `SST-2` across different learning rates.

Figure 9.7 shows the extrinsic homotopy distances across different learning rates for the `SST-2` task, while Figure 9.8 presents the corresponding prediction agreement between all model pairs.

At a very low learning rate (lr $= 1\mathrm{e}{-5}$), extrinsic distances are generally low, and most model pairs can be aligned well via non-linear transformations, despite relatively low prediction agreement. For lr $= 1\mathrm{e}{-4}$ and lr $= 1\mathrm{e}{-3}$, the distances remain within a narrow and stable range (approximately $0 - 5$), indicating that the learned representations across models are structurally compatible and can be effectively reconciled, even if their predictions differ.

Compared to `MRPC`, where extrinsic distances often exceed 10, `SST-2` exhibits a more consistent representational geometry across the model family. This suggests that fine-

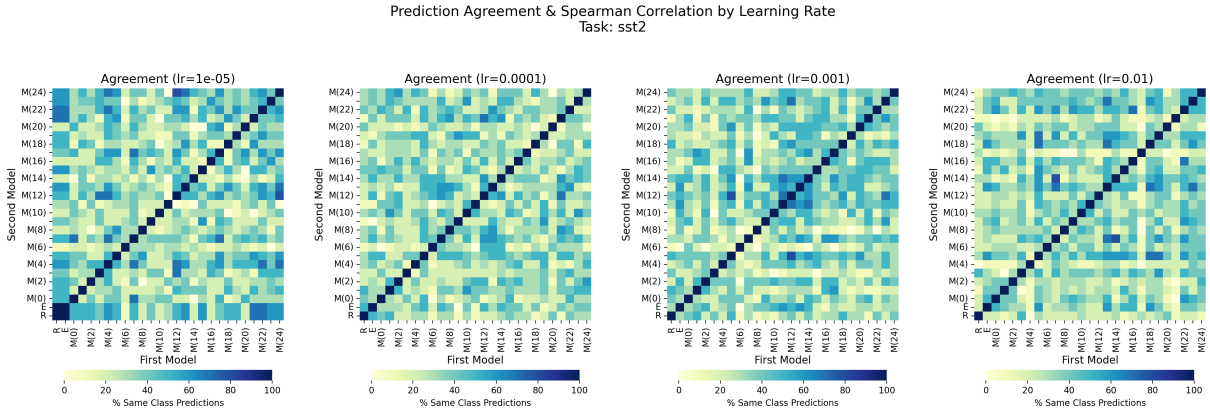


Figure 9.8: Heatmaps of performance-based similarity (agreement and Spearman correlation) on SST-2 across learning rates.

tuning onSST-2 induces less divergent changes in the encoder space and facilitates alignment under non-linear transformations.

At a higher learning rate (lr $= 1\text{e}{-2}$), we again observe a degradation in prediction agreement, and while some extrinsic distances remain low, the overall structure becomes noisier and less interpretable.

We now analyze the CoLA task with respect to extrinsic homotopy and performance-based similarity. Figure 9.9 shows the extrinsic distances, while Figure 9.10 presents the corresponding prediction agreement between model pairs.

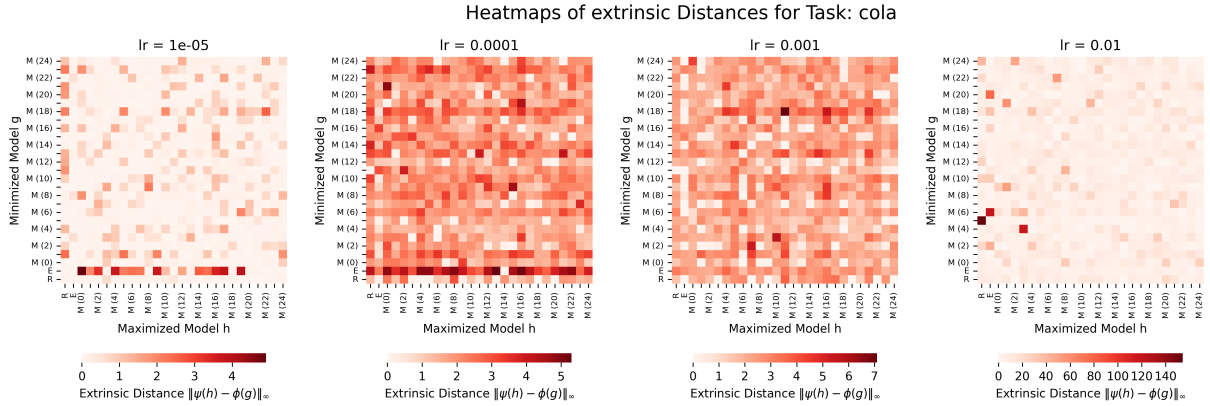


Figure 9.9: Heatmaps of extrinsic nonlinear distances on CoLA across different learning rates.

For the CoLA task, we observe a distinct behavior compared to other benchmarks. At the lowest learning rate (lr $= 1\text{e}{-5}$), almost all model pairs achieve near-perfect prediction agreement, and the corresponding extrinsic distances remain low. This suggests that the models are undertrained and likely converge to trivial classifiers, which is consistent with known difficulties on CoLA due to class imbalance.

As the learning rate increases, model predictions begin to diverge, and prediction agreement decreases. At lr $= 1\text{e}{-3}$, the agreement heatmaps show unstructured variation, and extrinsic distances rise accordingly—indicating growing representational and functional dissimilarity.

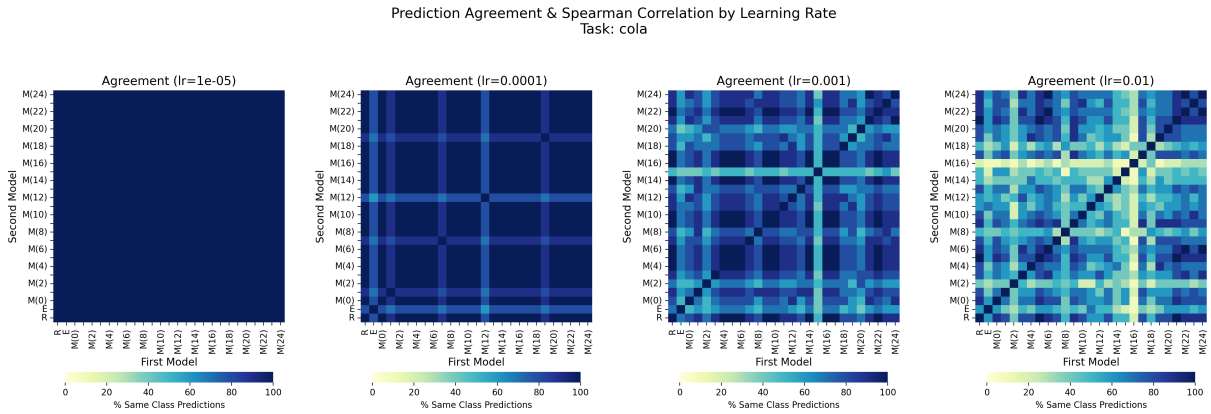


Figure 9.10: Heatmaps of performance-based similarity (agreement and Spearman correlation) on CoLA across learning rates.

At the highest learning rate (lr $= 1\mathrm{e}{-2}$), prediction agreement becomes highly inconsistent, and individual model pairs exhibit extreme extrinsic distances (up to 140), highlighting training instability. Overall, the results illustrate that extrinsic homotopy is highly sensitive to training dynamics and reveals instability that is not always visible through accuracy-based metrics alone.

Finally, we again observe that the extrinsic homotopy matrix is asymmetric. This aligns with the theoretical foundations of extrinsic homotopy, confirming that the relation is directional and cannot be inferred from symmetric similarity metrics such as agreement or correlation.

9.4 Consistency between Intrinsic and Extrinsic Distance

In this section, we analyze the relationship between intrinsic and extrinsic homotopy between encoders across multiple GLUE tasks. To ensure comparability and focus on meaningful transformations, we restrict our analysis to mappings whose estimated Lipschitz constant is below 1.5, filtering out unstable or overly complex functions.

Intrinsic homotopy measures the distance between encoders purely based on the structure of their internal representations—independent of any downstream task. In contrast, extrinsic homotopy quantifies the functional effect of aligning encoders in the context of a specific task, here binary classification on SST-2, MRPC, and QNLI.

Our goal is to investigate whether intrinsic similarity (i.e., representational closeness) predicts functional similarity under smooth transformations. This builds on previous work by Chan et al. [2], who found alignment between intrinsic and extrinsic similarity in the affine case. We extend their approach to the non-linear setting and test whether a similar relationship holds under more general transformations.

To this end, we analyze the correlation between intrinsic and extrinsic homotopy distances for each task and learning rate. Each point in the plots below represents a pair

of encoders. The x-axis shows the intrinsic distance $d_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g)$, and the y-axis shows the extrinsic homotopy distance $d^{\mathcal{H}}_{\mathcal{F}_{\mathrm{Lip}_1}}(h, g)$.

We focus on three tasks with qualitatively different homotopy behavior:

- `QNLI`: shows a strong alignment between intrinsic and extrinsic distances.
- `SST-2`: exhibits no meaningful correlation between the two.
- `MRPC`: lies between both extremes with moderate, learning-rate-dependent structure.

This selection allows us to highlight the diverse ways in which structural and functional similarity may interact.

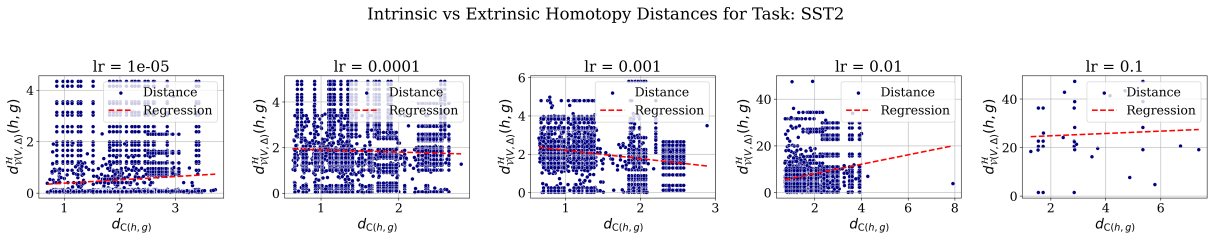


Figure 9.11: Intrinsic vs. extrinsic distances for encoder pairs on `SST-2`.

For `SST-2`, we observe very low extrinsic distances for all encoder pairs at low learning rates, with no clear dependence on intrinsic distance. The regression lines are flat or slightly negative across learning rates $\leq 10^{-3}$, suggesting that structurally dissimilar encoders can be aligned well in terms of task output. At higher learning rates (≥ 0.01), extrinsic distances increase dramatically, while the correlation with intrinsic distance remains weak. This implies that for sentiment classification, intrinsic geometry does not reliably reflect task behavior, possibly due to sparse or high-level features driving classification.

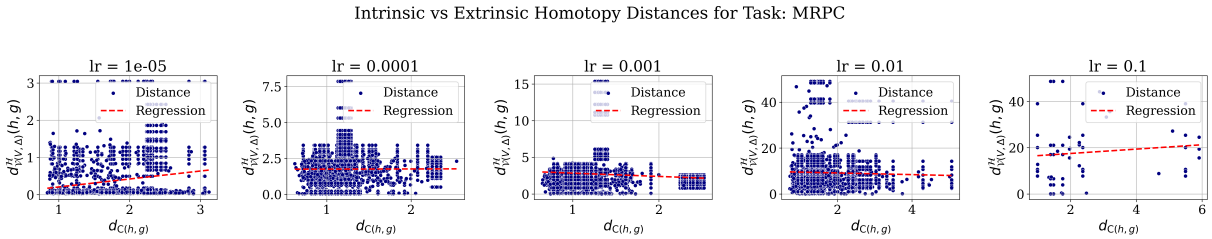


Figure 9.12: Intrinsic vs. extrinsic distances for encoder pairs on `MRPC`.

The `MRPC` task lies between `SST-2` and `QNLI` in terms of alignment. At low learning rates, there is a moderate positive trend between intrinsic and extrinsic distance, with fewer extreme outliers than in `SST-2`. As learning rates increase, the variance in extrinsic distance grows, but the general trend persists, albeit more weakly. This suggests that representational similarity may be partially predictive of task behavior for paraphrase detection, but only under stable training conditions.

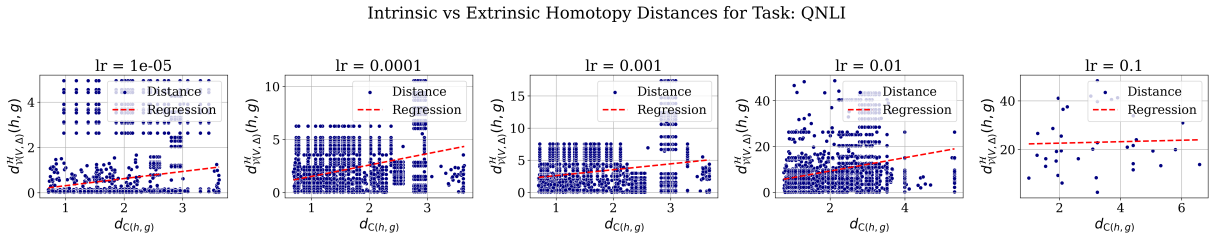


Figure 9.13: Intrinsic vs. extrinsic distances for encoder pairs on QNLI.

QNLI. For QNLI, we observe a clear and consistent positive correlation between intrinsic and extrinsic distance across all learning rates. This indicates that encoders with similar internal representations also exhibit similar behavior on the task. Even at higher learning rates, where extrinsic distances increase, the regression slope remains positive and robust. This alignment may reflect the task's reliance on fine-grained semantic matching, which is more directly encoded in the latent space.

Overall, our findings demonstrate that the relationship between intrinsic and extrinsic homotopy is highly task-dependent. For some tasks, such as QNLI, intrinsic similarity can meaningfully predict downstream behavior. In others, like SST-2, functional similarity emerges independently of representational closeness. This suggests that structural alignment in representation space is neither necessary nor sufficient for downstream task alignment, and motivates task-aware evaluation of model similarity.

10 Conclusion and Future Work

In this work, we introduced a homotopy-based framework for comparing language encoder models using two types of similarity: intrinsic similarity, which focuses on internal representations, and extrinsic similarity, which considers how well models can be transformed into each other based on their task-specific outputs. Our approach relies on learning non-linear, 1-Lipschitz-bounded transformations between model representations and estimating distances using the ℓ^∞ -norm.

Compared to the affine approach by Chan et al. [2], we found that non-linear transformations lead to more homotopic model pairs, especially when trained with smaller learning rates. This suggests that non-linear mappings allow for better alignment and reveal deeper structural relationships between encoders than affine baselines.

In the intrinsic setting, these transformations produced particularly strong results, uncovering denser connections between encoder pairs. In the extrinsic setting, we compared homotopy distances to performance-based similarity measures such as classification agreement and Spearman correlation. While high agreement often corresponded to low homotopy distance, we also observed notable exceptions: some models made similar predictions but were far apart in terms of homotopy, and others showed functional closeness despite low agreement. This indicates that homotopy-based similarity captures deeper aspects of model behavior that are not visible from output similarity alone. Moreover, extrinsic homotopy distances were asymmetric transforming model g into model h was not always as easy as the reverse, highlighting a directional structure that symmetric metrics cannot capture.

We also examined how intrinsic and extrinsic homotopy relate to each other across different tasks. While Chan et al. [2] suggested a close link in the affine case, our results showed that this relationship depends heavily on the task. For example, we observed a clear positive correlation in `QNLI`, no correlation in `SST-2`, and moderate correlation in `MRPC`. This shows that intrinsic and extrinsic similarity capture different aspects of model behavior and should be analyzed together to gain a more complete picture.

In summary, our work shows that homotopy-based similarity, especially with non-linear transformations, offers a powerful and flexible way to compare language models. It goes beyond simple agreement or correlation and helps reveal deeper structural relationships between models.

Limitations. Despite the promising results, this work has several limitations. First, our experiments were restricted to `BERT`-based models and GLUE classification tasks, which limits the generalizability to other architectures or domains. Second, the method

showed sensitivity to hyperparameters, particularly the learning rate, which influenced training stability and distance estimates. Additionally, the transformation networks were limited to simple feedforward architectures with spectral normalization, which may constrain expressiveness.

Future Work. While this study focused on language models and classification tasks from the GLUE benchmark, our approach is broadly applicable and opens up several directions for future research.

First, one promising application lies in the field of IT security. Since homotopy-based similarity captures structural and functional alignment between models, it could be used to detect anomalies introduced by data poisoning or backdoor attacks. For example, poisoned models might exhibit abnormal extrinsic or intrinsic distances when compared to clean reference models, even if their test accuracy remains high. Investigating whether homotopy measures can serve as early indicators of such manipulations could be a valuable direction for applied research.

While we were unable to fully reproduce Chan et al. [2] reported results on held-out data, we found that our training-based measurements exhibit patterns that are broadly consistent with their findings. This suggests that differences in evaluation methodology, particularly the choice of data split, may partially account for the discrepancy.

Finally, future work should address the current limitations by extending the analysis to more diverse model architectures, datasets, and transformation classes. This includes considering multilingual models, generative tasks, and more expressive transformation networks beyond fully connected layers with spectral normalization.

Bibliography

[1] Luca Antiga, Eli Stevens, and Thomas Viehmann. *Deep learning with PyTorch*. Manning, 2020.

[2] Robin Chan, Reda Boumasmoud, Anej Svete, Yuxin Ren, Qipeng Guo, Zhijing Jin, Shauli Ravfogel, Mrinmaya Sachan, Bernhard Schölkopf, Mennatallah El-Assady, and Ryan Cotterell. On affine homotopy between language encoders. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[3] Cheng-Shang Chang, Wanjiun Liao, Yu-Sheng Chen, and Li-Heng Liou. A Mathematical Theory for Clustering in Metric Spaces . *IEEE Transactions on Network Science and Engineering*, 3(01):2–16, January 2016.

[4] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020.

[5] Ryan Cotterell, Anej Svete, Clara Meister, Tianyu Liu, and Li Du. Formal aspects of language modeling.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[8] Tammo tom Dieck. *Mengentheoretische Topologie*. 2009.

[9] Otto Forster and Florian Lindemann. *Analysis 2: Differentialrechnung im $\mathbb{R}^n$, gewöhnliche Differentialgleichungen*. Grundkurs Mathematik. Springer Fachmedien Wiesbaden, Wiesbaden, 2025. ISSN 2626-613X, 2626-6148.

[10] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[11] Paul Geuchen, Dominik Stöger, Thomas Telaar, and Felix Voigtlaender. Upper and lower bounds for the Lipschitz constant of random neural networks, April 2025.

[12] Kaan Gokcesu and Hakan Gokcesu. Generalized huber loss for robust learning and its efficient minimization for a robust statistics, 2021.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, October 2020.

[14] Jean Goubault-Larrecq. *Non-Hausdorff Topology and Domain Theory: Selected Topics in Point-Set Topology*. New Mathematical Monographs. Cambridge University Press, Cambridge, 2013. Available in hardback.

[15] Palash Goyal, Sumit Pandey, and Karan Jain. *Deep Learning for Natural Language Processing*. Apress, Berkeley, CA, 2018.

[16] Jiawei Han, Micheline Kamber, and Jian Pei. 2 - getting to know your data. In *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 39–82. Morgan Kaufmann, Boston, third edition edition, 2012.

[17] Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):4037–4058, 2021.

[18] A. Jung. *Machine Learning: The Basics*. Machine Learning: Foundations, Methodologies, and Applications. Springer Nature Singapore, 2022.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[20] Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *ACM Comput. Surv.*, 57(9), May 2025.

[21] Kevin P. Knudson. *Algebraic Topology*. De Gruyter, Berlin, Boston, 2024.

[22] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019. ISSN: 2640-3498.

[23] Serge Lang. *Algebra*. Graduate Texts in Mathematics. Springer, New York, NY, 2002. ISSN: 0072-5285, 2197-5612.

[24] F. William Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del Seminario Matematico e Fisico di Milano*, 43(1):135–166, 1973.

[25] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, volume 44 of *Proceedings of Machine Learning Research*, pages 196–212, Montreal, Canada, 11 Dec 2015. PMLR.

[26] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. In *International Conference on Learning Representations*, 2020.

[27] Witold Pedrycz and Shyi-Ming Chen. Deep learning: Concepts and architectures.

[28] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[29] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[30] Thibault Sellam, Steve Yadlowsky, Ian Tenney, Jason Wei, Naomi Saphra, Alexander D'Amour, Tal Linzen, Jasmijn Bastings, Iulia Raluca Turc, Jacob Eisenstein, Dipanjan Das, and Ellie Pavlick. The multiBERTs: BERT reproductions for robustness analysis. In *International Conference on Learning Representations*, 2022.

[31] Bart M. N. Smets. Mathematics of neural networks (lecture notes graduate course), 2024.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[33] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

[34] D. Werner. *Funktionalanalysis*. Springer-Lehrbuch. Springer, 2005.

[35] Alex H Williams, Erin Kunz, Simon Kornblith, and Scott Linderman. Generalized shape metrics on neural representations. In *Advances in Neural Information Processing Systems*, volume 34, pages 4738–4750. Curran Associates, Inc., 2021.

[36] G.R. Wood and B.P. Zhang. Estimation of the Lipschitz constant of a function. *Journal of Global Optimization*, 8(1), January 1996.

[37] Zhi-Hua Zhou. *Machine Learning*. Springer Singapore, 2021.

Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Master Thesis als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.
2. Ich erkläre hiermit, dass ich diese Master Thesis selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum und Unterschrift

Presented by:	Bettina Zieger
Student ID:	3401509
Study Programme:	Master Mathematics
Time Frame:	1. March 2025 – 31. July 2025
Supervisor:	Prof. Dr. Stefan Körkel
Secondary Supervisor:	Prof. Dr. Wolfgang Lauf
External Supervisor:	M. Sc. Daniel Kowatsch, Fraunhofer AISEC