DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Constructing Transformations Between Pre-trained Neural Networks

Victor-Constantin Stroescu

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Constructing Transformations Between Pre-trained Neural Networks

Konstruktion von Transformationen zwischen trainierten neuronalen Netzwerken

Author:	Victor-Constantin Stroescu
Supervisor:	Prof. Dr. Christian Mendl
Advisor:	Dr. Felix Dietrich
Submission Date:	15.01.2022

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.01.2022 Victor-Constantin Stroescu

Acknowledgments

I wish to thank my advisor, Dr. Felix Dietrich, for all the support he offered during this thesis in various forms including the explanations of concepts, advice about the path I should take for the completion of the Thesis, as well as all the resources he provided, both in documentation and his previous implementations of similar concepts to those discussed in this thesis. Further I wish to thank Dr. Ertug Olcay for granting me the permission to use the dataset and models developed during the interdisciplinary project "Deep-learning based approaches for fault detection in a rotary mower" by Victor-Constantin Stroescu, for which he was my advisor.

Abstract

This thesis will describe our approach at implementing transformations between neural networks, as they were presented in "Transformations between deep neural networks" by Tom Bertalan, Felix Dietrich, and Ioannis G. Kevrekidis [1], on established, pre-trained neural networks, like Wav2Vec2 [2], XLNet [3], and other similar networks. By creating transformations between these neural networks, we aim to establish equivalence classes between widely used, pre-trained models. For our implementation of the transformations, we will use the approach established in the aforementioned paper, namely diffusion maps with a Mahalanobis-like metric. We will also use Whitney's theorem to estimate the number of measurements required from each neural network to reconstruct all features from the other network. For this purpose, we aim to use different models, which were trained to tackle tasks such as fault detection on sequence data, speech recognition, and text interpretation. The Models used for the representational experiments were dependent on the availability of pre-trained neural networks, but we will also present the procedures that aim to implement the transformation between neural networks that work with popular data types, such as vibration data, text data, sound data.

Contents

1 Introduction

This master thesis will present the work done on the topic of "Constructing Transformations Between Pre-trained Neural Networks", which will include the implementation of established concepts and new contributions to the topic.

1.1 Motivation

The availability of pretrained neural networks for different tasks has only expanded since different tasks have had neural network approaches function as solutions. The library Huggingface lists no less than 25,380 different pretrained open-source models on their website [4]. These are mostly models that deal with tasks defined for sound and text. An approach that could make use of these models by combining select pairs of them to deal with new tasks has a very good opportunity to be an alternative to transfer learning, which also uses old models to deal with new tasks, which would require no training of neural network layers.

This intent is based on the assumption that different neural networks sometimes model the same phenomena observed either through the same or different sensors with very different learned weights and sometimes very different structures. These networks will then be considered part of the same equivalence class, a concept introduced in "Transformations between deep Neural Networks" by Tom Bertalan, Felix Dietrich and Ioannis G. Kevrekidis [1]. If two neural networks belong to the same equivalence class then transformations between the activations of the networks can be defined. This means that there is an invertible function between the two different representations of the same phenomenon within the activation spaces of the two neural networks. The existence of such a function, called a transformation, between two deep neural networks, can both give us a tool to analyze underlying properties of networks by assigning them to equivalence classes, and be used to create a new model using that function to combine the two.

These concepts were first presented and used in the paper "Transformations between deep Neural Networks" [1] on neural networks that were trained on either N-dimensional real-valued vectors of images. The scope of this master thesis is to expand the set of possible neural networks and show that the concepts presented in the aforementioned paper are generally implementable on other types of neural networks. The focus of this thesis will be the implementation of transformations on pretrained neural networks trained for processing of sequence data types.

We will show through the results of experiments performed using a prototype implementation of the presented concepts that these mappings can be created for neural networks trained

on different datasets, such as vibration, speech, and text datasets, and that they achieve similar performances to the networks they were built from.

During this master thesis we will also show that a new model can be created from two neural networks, trained on the previously mentioned dataset types, and one transformation between their activation spaces. This will be proved through the results of the fifth experiment, which will create a speech sentiment analysis model by using a transformation from the activations of a pretrained ASR (Automatic Speech Recognition) model to the activations of a pretrained text sentiment analysis model and the two models.

1.2 Structure

This thesis will start by presenting the theoretical basis and previous work surrounding this topic. Next, we will present the implementation of the general algorithm used for all the experiments encompassed within this work. The experiments serve as the main contribution of this master thesis, since they are used to show the capabilities of the algorithm for multiple models.

Further it will present each individual experiment performed during this thesis, through the use of the implemented algorithm. The sequence in which the experiments are introduced throughout the thesis is thought out such that they follow a progression in terms of introduced complexity. Starting from cases that showcase the algorithm's functionality, a better understanding of the implementation can be supported through graphs that help visualize every step of the process. The later experiments are performed on data with larger dimensionality, that would result in unintuitive figures, but which serve to outline the implementation's usefulness on more complex data with real-world origins (i.e. sound, text).

Thus we will start with a recreation of the first experiment from the paper "Transformations between deep Neural Networks" by Tom Bertalan, Felix Dietrich and Ioannis G. Kevrekidis [1], which is used as the main basis for this thesis. During this experiment we train two neural networks with different architectures which evaluate a one dimensional function on the domain [-1,1] and then create the transformation between the two networks. Due to the low dimensionality of the data we can visualize the entire process in the smallest detail.

All other experiments will use widely available pretrained neural networks, due to both time constraints as well as proof of concept on neural networks for general tasks.

The second experiment will create transformations between two neural networks created for fault classifications based on vibration data gathered by sensors on the machines over a time period. The two neural networks used for this classification are a LSTM [5] and transformer based network [6].

Thus, this second experiment introduces data that has more than one feature, and which is sequantial. The transformation created for the neural networks made to classify such data are created accordingly for handling the added complexity

The test set will then be used to present the metrics for the classification task using both the transformation from the LSTM model to the transformer model and the transformation from the transformer model to the LSTM model. Since both text data and sound data have

a common sequential nature and text data embeddings are typically high dimensional, this experiment should be seen as a stepping stone towards both experiments which will be combined for the last one, which will show a real world application of the transformations.

In the third experiment we will attempt to compute a transformation between two different pretrained open source neural networks for Speech to Text translation from the Huggingface library[4]: Speech2Text2 [7] and Wav2Vec2 [8] using a sampling of the Librispeech dataset [9]. This transformation will be the first step towards the final, fifth, experiment, solidifying the possibility of building a transformation between a network trained to translate speech to text and one to process text, since it introduces the contribution made by this thesis through the implementation of transformations for variable length sequence data. The Librispeech dataset will also be used in order to train the transformation for the fourth experiment and for the final experiment since it contains both text data and the equivalent sound data, which is necessary in order to create equivalent samples and neighborhoods for the two neural networks.

The fourth experiment will attempt a transformation between two neural networks for Text Sentiment Analysis. The two neural networks used for this experiment are a pretrained version of the RoBERTa model [10], and a pretrained version of the XLNet model [3], both downloaded from the Huggingface library [4]. While both pretrained network were trained on different datasets, the transformation will be also trained on the Librispeech dataset [9], this time using the targets as input, since it is text data and we can thus use the same dataset for the computation of the transformation of the final experiment.

The fourth experiment introduces the second contribution to the established concept made by this thesis, the generation of neighborhoods and their activations for neural networks with discrete input domains, in this case networks with text input. This is the last step toward the final experiment.

The last experiment uses one pretrained network from the third experiment, the Wav2Vec2 model[8], and one pretained network from the fourth experiment, the XLNet model [3], and computes a transformation that creates a new model based on the two, which combines their intended usages in a new one. This will showcase another capability of the concept of transformations between neural networks beyond its ability to asses equivalences between neural networks, the ability to create new models from two equivalent neural networks.

With the last experiment we will conclude the experiment description portion of this thesis and thus the main contribution made by this thesis.

This thesis will close with its conclusions. In that chapter we will review the results and discuss their implications. We will also present other possible research avenues that arose from the results of the experiments and the problems that arose during the implementation and experiments.

2 Related Work

This section will present the work previously done which was necessary for the writing of this thesis. It will be split in two sections. The first one will present the work previously done to establish the concepts used, both the direct prerequisites, like the development of the concept of transformations and Whitney's theorem [11], as well as the important mathematical theory necessary for our implementation, like the Mahalanobis distance and Diffusion Maps. The second section presents the papers that introduced the models used in this thesis, a brief description of these models, as well as the main improvements that these models brought over their precursors.

2.1 Concepts

The main source for the concepts that will be presented in this thesis is the paper "Transformations between deep Neural Networks" by Tom Bertalan, Felix Dietrich, and Ioannis G. Kevrekidis and all of the sources cited within, since it is both the precursor and the main inspiration for this thesis.

In order to be able to understand the implementation and experiments presented in this thesis a number of mathematical concepts and notations have to be introduced. These will both function as a theoretical foundation for the algorithms and serve as an introduction to the necessary concepts that explain why the algorithms are effective.

2.1.1 Manifolds

Since the whole implementation is based on the concept of manifolds, they will be the first to be introduced. As Loring W. Tu presents manifolds in the abstract of a chapter of his book "An Introduction to Manifolds": "Intuitively, a manifold is a generalization of curves and surfaces to higher dimensions. It is locally Euclidean in that every point has a neighborhood, called a chart, homeomorphic to an open subset of $\mathbb{R}^n$. The coordinates on a chart allow one to carry out computations as though in a Euclidean space, so that many concepts from $\mathbb{R}^n$, such as differentiability, point-derivations, tangent spaces, and differential forms, carry over to a manifold" [12]. This description not only introduces the concepts of manifold intuitively but also shows why we use the the neighbourhoods of points in our algorithms. These point neighborhoods shall be referred to as *neighborhoods* instead of *charts* throughout the thesis, for maintaining intuitive readability. Two important concepts needed for this thesis were those of Riemannian manifolds and Whitney's theorem [11]. Riemannian manifolds are smooth manifolds which permit the measurement of distance with a Riemannian metric [13]. Whitney's Theorem is used in [1] to argue that for a d-dimensional input manifold we

can guarantee the preservation of its topology only by passing it through layers with at least $2d + 1$ neurons, and as such only neurons that are before any layer with less than $2d + 1$ neurons should be used as input to the transformation.

2.1.2 Mahalanobis Distance

The Mahalanobis distance, first presented in "On the generalized distance in statistics" by Prasanta Chandra Mahalanobis [14] is a generalisation of the euclidian distance which takes the variance in different dimensions into account when computing distances. It will be used when computing the kernel function of the diffusion maps.

2.1.3 Diffusion Maps

The final important concept for this thesis is that of the diffusion maps, which first appeared in "Diffusion maps" by Ronald R. Coifman and Stéphane Lafon [15]. Diffusion maps are coordinates constructed by using the eigenvectors of a Markov matrix which uses as probabilities kernel functions, which represent the distance between points in terms of likelihood [15]. A larger distance between two points represents a smaller likelihood of that state transition. For our implementation we have used a Gaussian kernel, defined as :

$$k(x, y) = exp(-\frac{||x - y||^2}{\epsilon}) \tag{2.1}$$

with the Euclidean distance, $||x - y||^2$, in Equation 2.1 replaced by the Mahalanobis distance. This method was initially proposed in [1], which functions as a guideline for the concepts implemented in this thesis.

2.2 Models and Datasets

The first experiment, done with the purpose of replicating the first example in [1], uses both model definitions, a 1-1-1 model and a 1-8-8-8-1 model and dataset definition, points sampled from the manifold $M = [-1, 1]$, presented in the first example from [1]. The models were trained during this thesis to match the performances presented in [1].

The second experiment uses a dataset from the project presented in [16], which contains 6 recordings of sensors of a rotary mower over a time segment of 20 milliseconds, which is represented through 100 timesteps, as inputs. The targets represent the status of each blade during the whole time of the recording. For the models the status are concatenated to a single value which identifies whether any blade is faulty.

Due to the increase in popularity of models used for sequence data, such as models used for text translation, speech recognition, etc., we have chosen for these experiments performed in this thesis we used a class of models for time sequence data: vibrations, speech and text. For this type of data, two layer types have emerged lately as the most chosen components for new models: LSTM layers [5] and transformer layers [6]. The second experiment uses two

models implemented in [16], each using one of these layers as a central component of the networks.

For experiments 3 and onward, the Librispeech dataset [9] was chosen for the training of the transformations. The Librispeech dataset is a dataset with english speech and the text read to generate the speech. Each entry into the dataset contains a speech component and a text component along with other descriptors which we will not use. The Librispeech dataset [9] fits our needs since we need a dataset with speech data for the third experiment, a dataset for text data for the fourth experiment and a dataset containing both speech data and the textual translation of that data.

The third experiment used widely available encoder-decoder models with pretrained weights for automatic speech recognition. Both models use the same encoder layer with the decoder being different, since the two are trained for different tasks. Both model definitions were taken from the Huggingface library [4] and their pretrained weights were taken from projects hosted on the Huggingface platform. The first model chosen was a Wav2Vec2 model [8], with the weights taken from a project by user "patrickvonplaten" [2] from the platform. It was trained to transform english speech to english text. The second model used was a Speech2Text2 model [7], with the pretrained weights presented in the same article and hosted on the Huggingface platform. This model was trained to perform a translation from english speech to german text.

The models used for the forth experiment also stem from the Huggingface library [4], with their weights hosted on the same platform. Both models were trained for sentiment analysis on english text. An implementation of the XLNet model [3] was chosen as the first model, with weights sourced from the project [17], attributed to the user "textattack", on the platform. The second model used was a RoBERTa[10] implementation from the Huggingface library, with the weights sourced from the project presented in [18].

The presented Wav2Vec2 model [8], with the pretrained weights from [2] and the XLNet model [3] with the weights from [17] will be reused for the fifth and final experiment.

3 Implementation

This chapter of the thesis will present and explain both the implementation of the concepts presented in the previous chapter, chapter 2, and how these were adapted and melded together to create the final algorithms used in the experiments presented in chapter 4. It will start with an outline of the general algorithm, which will give a general idea about how the algorithm works, and then move on to present each individual component of the implementation in detail, thus enabling future readers of this thesis to replicate the pursued experiments.

3.1 Implementation Outline

The outline will present how the algorithm generally works, what its component parts are and how they combine in order to create the code basis used for the experiments.

The algorithms can be split in two categories: algorithms for the computation of the transformation (section 3.2, section 3.3 and section 3.4) and algorithms for the usage and testing of the computation (section 3.5). The first category can be again split in three steps: The generation of the input data necessary for the computation of the transformations, i.e. the samples and the neighborhoods, (section 3.2), the propagation of the input data through the networks to generate the activations (section 3.3), and the computation of the transformation based on the activations (section 3.4)

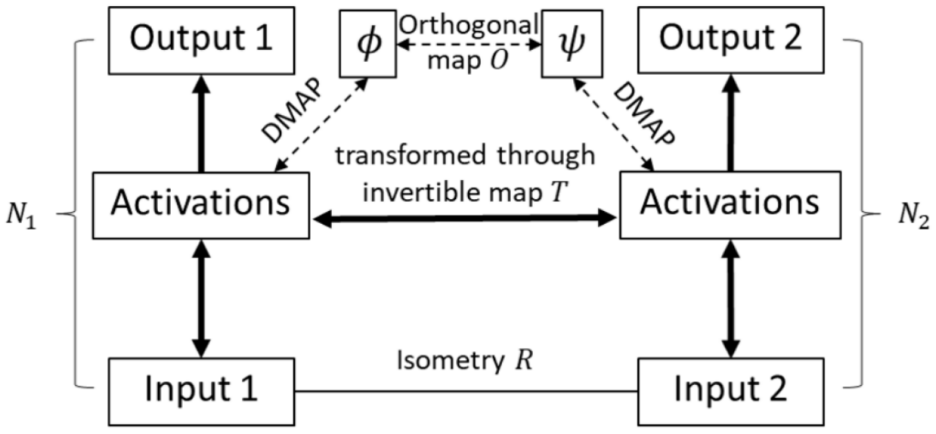


Figure 3.1: Outline of the Transformation

Figure 3.1 shows the general outline of the implementation as presented in [1], which also

served as a guide to the steps necessary for our implementation. Each step presented above can be seen in this outline, with a focus on the component parts of the computation of the transformation.

Based on the outline in figure and the knowledge from [1], we can create a general road map of how the algorithm should be implemented, which we will later describe in a number of steps. The outline of the algorithm to compute the transformation Figure 3.1 indicates the fact that part of the algorithm is the same for both of the neural networks and it functions independently for each of the neural networks up to the point of the orthogonal map O. As such we will present the outline of the algorithm for only one neural network up to that point and it is assumed that the same procedure occurs for the other.

The initial condition that needs to be met for the concept to function is that the input domain of both neural networks needs to either be the same or that there is a diffeomorphism, a smooth, differentiable and invertible function, between the two. Since for all experiment except for the last we use the same input domain, we can use the identity function as the chosen the diffeomorphism. For the fifth experiment the discussion of the input domain manifolds and the existence of the diffeomorphism will be presented in section 4.5. For the outline and description of the implementation the assumption is made that a diffeomorphism exists.

Based on that assumption we choose **N** values from the training domain of the first neural network (which we assume to be a Manifold), which "cover the training domain well". For each of those **N** we compute their mapping to the training domain of the second neural network (also assumed to be a Manifold) using the chosen diffeomorphism. These values will be known as the samples of network one and two respectively.

For each of the **N** samples on the training domain of the first model, we sampled **M** points from a multivariate Gaussian distribution with the mean equal to the value and the same variance for all N values. The variance δ is a hyperparameter. The optimal δ needs to be found for each experiment, and is dependent on the training domain manifold. After we generate the M points on the training domain manifold of the first model we also map those to the training domain of the second model using the chosen diffeomorphism. This procedure changed only for the fourth experiment. The required adaptation will be explained in section 4.4. Each of the **M** values will be called the neighborhood of the sample.

With the **N** samples and their neighborhoods for each training domain we can now present the steps needed to be followed, based on the Outline in Figure 3.1, for each of the neural networks:

1. We have to propagate the samples and the neighborhoods through the neural network up to the activation layer chosen by us. Even though [1] states that we can use activations on different layers, we limit this, for reasons stated further, to a single layer. We then have to record the output values of the chosen activations while passing both the samples and their neighborhoods through the forward pass of the model. The output values recorded will be called sample activations and neighborhood activations respectively.

2. We compute covariance matrices from the the neighborhood activations for each sample.

These will be then used to compute the Mahalanobis distance [14] used by the diffusion map kernel.

3. We create the diffusion map object using the samples and a Gaussian kernel which uses the Mahalanobis distance computed from the inverse of the covariance matrices previously computed.

4. We map the samples to the the diffusion map embedding, and then compute interpolations using a Radial Basis Functions Interpolator object from the datafold library [19] from the sample activations to their diffusion map space representations.

Using the diffusion map representations of the samples we can now compute the orthogonal transformations between them. We will compute the orthogonal transformation in both directions, from the diffusion map space of model 1 to the diffusion map space of model 2 and in reverse, at the same point of the algorithm, since both will be necessary at a later point. In order to compute them we used the orthogonal Procrustes alignment algorithm from the SciPy library [20]. The first column of all of the eigenvectors of the diffusion map is a constant, and as such the including them in the orthogonal procrustes alignment algorithm is suboptimal. This lead to our decision to ignore the first column in the input of the alignment algorithm. This requires a couple of changes in the algorithm which computes the transformation, which will be presented in section 3.4.

After propagating the diffusion map representations of the samples of a model through the matrix generated by the orthogonal procrustes algorithm, they are used as the domain for another Radial Basis Function interpolator, with the targets set to the diffusion map representations of the samples of the other model. This procedure needs again to be used for both directions, in order to generate the interpolation for the inverse of the diffusion map for both models.

The interpolated function for the diffusion map of one model, the orthogonal transformation and the interpolated function for the inverse of the diffusion map of the other model are then called sequentially in the predict function, which is the function called in order to perform the complete transformation from one activation space to the other. Whether the performed transformation will be from model 1 activation space to model 2 activation space or its inverse can be decided at runtime by setting a specific parameter.

Finally, the usage and testing of the algorithm will be presented in section 3.4. Given an already computed transformation, this section of the implementation does a pass of the models with the transformation by passing the input to the initial model, recording the activations at the chosen layer and perform the transformation. The other model is then started with an input and the transformed activations will replace the activations at the chosen layer of the target model. This will be repeated for each of the inputs.

The following sections of this chapter shall highlight the corresponding details for the steps of the algorithm above.

3.2 Sample and Neighborhood Generation

This section will present the algorithms used in order to perform sample and neighborhood generation, the conditions imposed on the algorithms, and the importance of the correctness of these algorithms for the performance of the transformations.

The purpose of the algorithm was to create a set of values that cover the training domain manifold of the chosen neural network well, to create the set of samples, as well as sample the space around each value to create the neighborhood set for each sample.

Based on the formulation of the theoretical concept, two such sets, of samples and their neighborhoods, need to be created for each transformation, with the condition that there is a diffeomorphism between the two sets that is also a diffeomorphism for the two training domain manifolds [1]. This condition could be ignored up to the last experiment, since all the model pairs of experiments one two and three were trained on the same training set, and as such we can assume the diffeomorphism to be the identity function and use the same set of samples and neighborhoods for both models in these experiments. While the two models in experiment four were trained on different domains, both were trained on the same type of data, and as such the argument can be made that both training sets are part of a larger training domain of general text data, which fits the purpose of the neural networks, since there is no constraint on the texts that sentiment analysis can be performed on. For experiment 5 another approach at the generation had to be taken, which will be discussed later in this section.

We can conclude both from the previous paragraph and the fact that the dimensionality and characteristics of training domains vary between experiments, that sample and neighborhood generation have to be tackled on a case by case basis and there is no solution that can function for all training domains and experiments. As such the algorithm is split into four different sections, matching the subsections below, based on the datatype that the training domain contains.

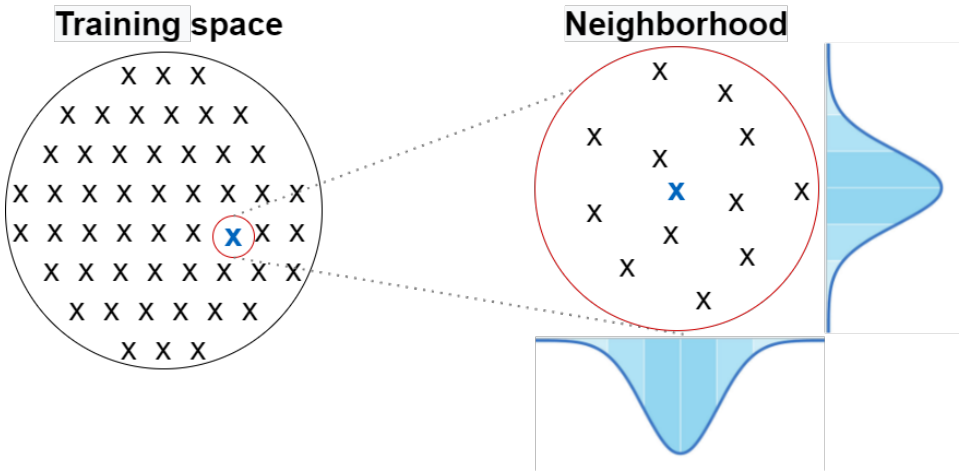


Figure 3.2: General description of the sample and neighborhood generation

Figure 3.2 shows a general visualisation of the sample and neighborhood generation. While this specific case, a two dimensional disk shaped manifold, never occurs in our experiments, it is a useful abstraction of what would occur in the case of an N-dimensional manifold. It shows that the training space manifold should be well covered in all dimensions, and that the neighborhood of each sample will be created by sampling multivariate Gaussian distribution with a number of dimensions equal to that of the training domain manifold, with the mean at the sample and the variance equal in all dimensions. The variance of the multivariate Gaussian distribution is equal to a fourth of the hyperparameter used, since we tried to use the "three-sigma rule"[21] so that the hyperparameter is understood as the space around the sample that the neighborhood covers. We used a division by four instead of three since it seemed to perform better during the experiments.

The generation is dependent on a couple of hyperparameters, which might differ depending on the models for which the transformation is generated and on the training domain manifold of these models. The number of samples, the neighborhood size, and the variance of the multivariate Gaussian distribution are hyperparameters which need to be specified for all generations. Other hyperparameters which specify how the manifold will be sampled will also be necessary for all of the different procedures, but these are dependant on the manifold and will be specified with the generation procedures presented in the following subsections.

3.2.1 One-dimensional Data

This subsection will present the procedure used to generate the samples and neighborhoods for the first experiment and other tasks with similar training domains.

This is the simplest case, since the training manifold is one dimensional and its boundaries are clearly defined. It is given by the experiment setup in [1] that the manifold on which the samples are situated is equal to $M = [-1, 1] \subseteq \mathbf{R}$ for both models. As such the diffeomorphism is, as previously stated, equal to the one dimensional identity function $f(x) = x$. As such only one set of samples and their neighborhoods needs to be computed and they will be used for the training of both models.

Additional to the usual hyperparameters, the number of samples, the size of the neighborhood and the variance of the multivariate Gaussian distribution, the algorithm also requires the lower and upper boundaries of the manifold as hyperparameters. In this case we make the implied assumption that the manifolds from which we select the samples are continuous and closed.

Due to requirements made by neural network layers, we have to generate sets of samples of the shape (number of samples, 1) and for each sample a neighborhood of the shape (size of neighborhood, 1), even if the second dimension is always redundant.

The samples were generated using the function:

$$X_i = LB + i \times \frac{(UB - LB)}{N} \quad \forall i \in \mathbb{Z}, 0 \leqslant i < N \tag{3.1}$$

Where LB is the lower boundary, UB is the upper boundary and **N** is the number of samples. This formula ensures that the manifold is covered well by setting all the points evenly over

the manifold, and that the lower boundary is enclosed in the set.

NX_i is the neighborhood of the sample X_i. It has the shape (neighborhood size,1), for reasons explained previously, and is generated by using the normal function of the pytorch library [22], with the mean equal to Xi, the shape equal to the intended shape of the neighborhood and a variance equal $\delta/4$. Finally, X_i and NX_i are returned by the algorithm.

3.2.2 Vibration Data

The algorithm used to generate samples and neighborhoods from the fault detection in rotary mower dataset [16] is based on the same concepts as the algorithm snippet presented in the previous subsection, especially since we use the same procedure to generate the neighborhoods from the samples, only adapting the shape generated by the python function normal from (neighbourhood shape,1) to (neighbourhood shape, 1, 100, 6) since the shape of each sample changed to (1, 100, 6).

The dataset is comprised of a set of inputs $x \in \mathbf{R}^{100x6}$ and a set of labels. Each input contains 100 time steps of six input features [16]. This determines the maximum dimension of the manifold $M \subseteq \mathbf{R}^{100x6}$, which does not also mean that we know the boundaries of said manifold. This directly results in an inability to create equidistant samples between known boundaries on the vibration data manifold. As a solution to this problem we have used a stochastic approach: If we randomly choose enough samples from the dataset, it will cover the manifold to a satisfactory extent. This assumption can be made since datasets created for training neural networks generally tend to cover the whole domain of possible inputs well enough that the networks trained on them are considered robust.

Using the previously discussed idea, the algorithm generates the set of samples by picking a number of unique indexes in the input dataset equal to **N** and use the arrays at those indexes as the set of samples. For each sample we generate the neighbourhood as previously discussed. The samples and neighborhoods can then be used as inputs for the next step, presented in section 3.3.

3.2.3 Speech Data

Both models used for experiment 3 (section 4.3) were trained on the speech column of the Librispeech dataset [9]. Each input is a variable length sequence with one feature. Since there is a maximal sequence length *max* in the dataset, we can assume that there exists a input domain manifold $M \subseteq \mathbf{R}^{max}$, where all inputs with size lower than that are padded with zeros at the the beginning in order to to become a member of said manifold.

The same difficulty presented previously, the fact that we do not know the boundaries of the manifold, also appears in the case of speech data. As such we use the same procedure of choosing the arrays at random indexes in the dataset, hoping that they cover the training domain well, as the samples.

Our input domain being the Librispeech datasets speech column [9] presents a new challenge, since different samples selected from said dataset do not have the same length. This poses a problem because the data structure used to store the samples and neighborhoods used

to be a pytorch tensor object [22, 9]. Because the tensor object is similar to a multidimensional array, it can not be used to store sequences of differing lengths. We have chosen for this case to use a list of pytorch tensor objects [9] both for samples and neighborhoods, where each position in the samples list represents one sample and the same position in the neighborhoods list represents its neighborhood.

The last experiment presented in section 4.5 also uses the speech data sample generation method. Since it needs to create the mapping between the input manifolds, it will use the Speech to text model from section 4.5, the Wav2Vec2 model [8], as a mapping from the speech data manifold to the text data manifold.

3.2.4 Text Data

The first four experiments contained within the thesis are to be considered stepping stones towards the final, fifth experiment, to be presented in section 4.5. The transformation for the fourth experiment will not be trained on the datasets on which the two models from the fourth experiments were trained, but again on the Librispeech dataset [9] – this time using the target text column as the dataset from which the samples are picked.

The tokenized text data found in the Librispeech dataset [9] is very similar to the speech data in the same data, because they are both sequences of varying length with only one feature. Due to this, we can use the same procedure we used for speech data in subsection 3.2.3 to generate the samples, namely random picking of elements of the dataset as samples which are all stored in a list.

Even though they share similarities, text data is by definition discrete, as opposed to all other presented data types, which are continuous. While this fact does not change the way the samples are generated, the generation of neighborhoods becomes much more complicated.

The solution we have found in order to be able to generate the neighborhoods was to forgo their generation on the discrete space and generate them on the embedding space created by the embedding layer of each neural network, since it is continuous. This solution comes with a drawback, namely the fact that embedding spaces created by the embedding layer of a neural network can differ depending on the training of the model. Since the embedding space is different we can not use the exact same neighborhoods for the two neural networks, and we can not guarantee there is a diffeomorphism between the two sets of neighborhoods and that such a function is also a diffeomorphism between the two manifolds created by the embedding layer.

This special case of neighborhood generation will be discussed at large in section 3.3, since it can be argued that we directly generate the neighborhood activations without any initial neighborhoods.

A future topic that arises from this problem and its sub-optimal solution is the topic of efficiently computing either diffeomorphisms between embedding spaces generated by neural networks for text processing.

3.3 Activation Generation

As presented in the Outline (section 3.1), the next step will be the generation of the sample activations and neighbourhood activations by using the samples and neighbourhoods generated in the previous step, section 3.2, and propagating them through the models up to the activation layer. Since this procedure can be done in parallel for both models and is independent of the other model, we will henceforth refer to the procedure for one model, and it is implied that the procedure is performed on both models of the transformation.

This section is split into three parts. First we will discuss in subsection 3.3.1 preparations that need to be undertaken before any activations can be recorded. We will then present the algorithms developed for the generation of activations in the general case (subsection 3.3.2), which occurs in all experiments except Experiment 4 (section 4.4). Finally we will also showcase what changes needed to be made in order to generate activations for both samples and neighborhoods for the aforementioned experiment in subsection 3.3.3

3.3.1 Preparation

In order to generate activations, the object which does this tasks has to be initialized with a couple of necessary parameters: the neural network used, the activation layer chosen, the dimensionality of the output of the activation layer, the chosen positions of the neurons in the activation layer, and whether the outputs of the activation layer should be reshaped into a 2D matrix with row size equal to the number of samples.

Generally the preparation of the activation generation is done in the initialization of said object. First we set the neural network to the evaluation mode, to make sure that its weights will not change during the computation of the transformation. We will also create an empty list of activations which will function as a temporary container for the outputs of the activations. One important step is the segmentation of the neural network in its component layers, which we will name children, so that we are able to call the chosen activation layer. Since all models used are instances of the pytorch library [22] module class, which function similar to a tree structure, we used a recursive function to gather all the children, inspired by the code snippet found at [23].

Another important part of the preparation is the generation and usage of the pytorch library [22] objects called neural network forward hooks. These objects are attached to a specific layer, in the general case the chosen activation layer and have access to that layer. This includes its inputs, and outputs and its return value replace the outputs of that layer. We have defined a couple of different hooks as member objects of the activation generation class that need to be chosen at the moment of initialisation depending on different possible output options of the activation layer. These options are the ones noticed during the experiment. If new options are required, these will need to be hardcoded inside the class, since no possible implementation was found where they were defined outside of the class. This occurs because the hooks all store the output of the layers in a class member object, which we previously called the temporary container of the activations. Additionally, they also call a custom error at the end which stops the forward pass of the neural network. This was done to shorten the

time in which the activations were generated, by avoiding the completion of the forward pass. The hooks are automatically attached to the chosen layer during the initialisation of the class object. The class also includes a function to detach the hook at the end of the generation, in the case that the neural network needs to be reused.

3.3.2 General case

After finalizing all the necessary preparations, a couple of different functions can be called to generate activations, depending on whether the activations need to be generated for a single sample, for a list of samples or for a list of sample neighborhoods.

The first function we will use is intended to return the values of the chosen positions of the activation layer for one sample. It starts the forward pass of the neural network with the sample, then catches the previously defined custom error and stores the activations from the temporary container. Before returning, it selects the values of the chosen neurons from that layer and reshapes them, based on the previously described parameters, passed during the initialisation of the object, in such a way that they are directly usable by the next step, without any further modifications.

To generate the activations for a list of samples we call the algorithm described in the previous paragraph for each sample in the list and append the result to a list that is then returned by this algorithm.

The generation of the activations of all neighborhoods is done by calling the algorithm for generating the activations of a list of samples for each neighborhood in the list of neighorhoods and append the results to a list which is also the return value of this function. This is possible, from an algorithmic standpoint, there is no difference between generating the activations of a list of samples or generating the activations for a neighborhood of a specific sample.

3.3.3 Text processing models

As was previously discussed in subsection 3.2.4, neighborhoods for transformations between models trained to process text data can not be generated on the discreete input domain, and they will be generated at the embedding layer.

As we did in the general case (subsection 3.3.2), the same hook is placed at the chosen activation layer. Additionally, a new container for the outputs of the embedding layer, and two new hooks, one for storing the embeddings in the container, and one for replacing the embeddings with the content of the container are defined. The first hook is defined similarly to the activation layer hook, the only change being which container it stores the output in. The second one replaces the output of the embedding layer at each forward pass with the current contents of the container – it sets them as the return value of the hook and omits the call of the custom error, since it would be counterproductive.

For each sample we call the first hook on the embedding layer, call the neural networks forward pass on the sample and copy the samples' embeddings from their temporary storage. The hook is then removed from the embedding layer, replacing it with the second one.

Using the sample embeddings we compute the neighborhood embeddings using a similar procedure to that presented in subsection 3.2.1, namely we sample a number of values equal to the neighborhood size from a multivariate Gaussian distribution with mean at the current sample and variance equal to $\delta/4$.

Having now the sample and neighborhood embeddings, we first set the sample embeddings in the embedding container and start the forward pass of the network. During this pass, the second embedding layer hook replaces the output with the sample embeddings The pass ends with the error raised by the activation layer hook. The values stored in the activations container are then appended to a list which stores sample activations. The same procedure is done afterwards using the neighborhood activations instead of the sample activations and appending the values in the activation container to a list of neighborhood activations at the same position as the sample activation of the sample to which the neighborhood corresponds. The second embedding layer hook is then also removed. After all the sample and neighborhood activations were appended to their corresponding lists, both lists are returned by the algorithm.

3.4 Computation of the Transformation

Since all the necessary parameters were computed in the last step, we can now commence with the computation of the transformation. The processes presented in this section will occur twice, once for the transformation from the activations of model one to the activations of model two, and once for the inverse transformation, from the activations of model one to the activations of model two. Both the transformation in the first direction and its inverse are used in all experiments except for the fifth. As such, we will present the process of generating the transformation in one direction with the implication that the same computations occur for the other direction subsequently.

The implementation of the algorithms for computing the transformation are based on the algorithms presented in [1], specifically presented as pseudocode in Algorithm 1 and Algorithm 2 of that paper. The first two steps of Algorithm 1, generation of samples and neighborhoods, the generation of their activations were already presented in section 3.2 and in section 3.3. This section will present the third step of Algorithm 1, which is described in Algorithm 2, along with the last two steps of the first algorithm.

The third step of Algorithm 1, presented in detail in Algorithm 2, details the construction of the diffusion maps from the activations of the samples to a number of eigenvectors. The number of eigenvectors used for each experiment is a hyperparameter that can affect the performance of the transformation to a great extent.

We start by calculating the inverse of the covariance matrices based on the neighborhood activations, since they are required to compute the mahalanobis distance between the samples, as dictated by the first step of Algorithm 2. The computation is done using the numpy library [24] function "cov" to compute the covariance matrix from each neighborhood activation and the function "pinv" from the linalg section of the numpy library [24] to compute the pseudoinverse of the covariance matrix. This was done for each neighbourhood activation

and was placed in an array at same position that the sample it is the neighborhood of was placed in the sample list. The function then returns one array of **N** inverse covariance matrices for each model.

The algorithm for the second step, which computes the Mahalanobis kernel, presented in chapter 2, was provided by Dr. Felix Dietrich in a script named "mahalanobis_kernel.py". This algorithm uses methods from datafold [19], Scipy [20] and numpy [24] to compute the Mahalanobis distance, presented in subsection 2.1.2, between all samples. These distances are then used to create a diffusion map kernel matrix, by employing a Gaussian kernel with a Mahalanobis distance, concepts which we explained in subsection 2.1.3, and store the values returned by the kernel for all sample combinations in a matrix. This matrix will then be the diffusion map kernel matrix used for the computation of the diffusion map eigenvectors.

The third step, the normalization of the kernel matrix and the computation of its non-harmonic eigenvectors with the largest eigenvalues, is also computed using a function, called "dmap", supplied by Dr. Felix Dietrich at the start of the thesis. It takes the samples and inverse covariance matrices as inputs, creates a PCManifold (Principal Component Manifold) object, defined in the pfold section of the datafold library [19], using the sample activations, and then calls its opitmize_parameters function. Using the previously computed Mahalanobis kernel and the number of eigenvectors defined in the parameter we initialize a DiffusionMaps object, defined in the dfold section of the datafold library [19], and call its fit function using the PCManifold object. The DiffusionMaps object now contains the diffusion map eigenvectors required later.

The last step of Algorithm 2 computes the interpolation between the sample activations and the previously calculated diffusion map eigenvectors. Initially, we have attempted to do this using a Geometric Harmonics Interpolator [25] from the datafold library [19] but due to a performance increase and lower computational time required we have opted to change the interpolator to a Radial Basis Function Interpolator [26] (RBFInterpolator) from the SciPy library [20]. Using instances of this interpolator object with the samples as inputs, the eigenvectors of the diffusion map as targets, the thin plate splines option, and a smoothing hyperparameter, we create the interpolation from the model activation spaces to their diffusion map spaces. The inverses are also created using other instances of the same interpolator object, but with the targets being the model activation spaces, and the inputs being the other models diffusion map eigenvectors, which were orthogonally transformed using the orthogonal transformation computed in the next step.

The last necessary component of the transformation is the orthogonal transformation between the two diffusion map eigenvector sets. This is done using an orthogonal procrustes alignment algorithm [27], found as a function in the SciPy [20] linalg section. This algorithm needs to be computed twice, since we need an orthogonal transformation for each direction. Different to the procedure in the second algorithm in the paper [1], we do not compute the orthogonal procrustes on all of the diffusion map eigenvectors, since the first will always be a constant. As such, we compute the orthogonal transformation using the procrustes alignment algorithm on both sets of the diffusion map eigenvectors excluding the first eigenvector in both sets. The transformation between the eigenvectors on the first position is computed

through a scalar. As such, the matrix performing the transformation will have the values computed by the orthogonal procrustes algorithm starting with the second row and the second column, while the first row and column will be filled with zeros excepting the element on the diagonal, which will contain the quotient of the two constant eigenvectors.

With the two interpolations, their inverse, the orthogonal transformation, and its inverse, we can now compute the transformation in both directions. This is done in our program by using the predict function of the transformation generation class object with the activations needed to be, and the direction they should be transformed in. The predict function will output both of the diffusion map space representations along with the transformed activations. This is done for the purpose of visualisation and evaluation of the process and these outputs will be showcased in sections of chapter 4. Since the two neural networks stored in the object are numbered based on their order in the initialisation, we shall call them model 1 and model 2. The direction parameter of the predict function indicates the target activation layer of the transformation and not the input activation layer. If direction 1 is picked, then the transformation performed will be assuming inputs from the activation layer of model 2 and transform them in the activation space of model 1. Direction 2 assumes inputs from the activation layer of model 2 and transforms them in the activation space of model 1.

3.5 Applying the Transformation

After computing the transformation $\mathbf{T}$ and its inverse, $\mathbf{T}^{-1}$, we now also have to implement a way that this transformation can be used. We have chosen not to focus on proving whether two neural networks are equivalent, but to bind two neural networks through an operation that transforms the output of an activation of one network and replace the output of the chosen activation layer on the other network. We thus show our ability to combine two models and a transformation in a new model.

For this purpose we have implemented a class which creates objects able to perform the forward pass of the previously described new model. These objects need to be initialized with the two neural networks, the positions of the activation layers on which the transformation was trained and the transformation itself. The parameters presented before are necessary – without them, the forward pass on the combined model is impossible irrespective of the model used. Additionally, there are optional parameters which deal with having activations on sequence data: two parameters which indicate the sequence axis in the activations and the lists of positions in the sequences on which the transformation was trained (Experiment 2-5); parameters that are used when not all neurons of the activation layer were used: the lists of neurons in each activation layer on which the transformation was trained(Experiemt 2-5); and two booleans which mark if the sequence length of one activation is larger than that of the other (Experiment 4 and 5).

The function predict of these models is the function which implements the forward pass. It must be called with the inputs of the combined model, the direction in which the transformation should be performed, which works as it does in section 3.4, direction should be set equal to the target model, two tuples which indicate the permutation that needs to be performed on the activations before they are passed as input to the transformation and the permutation that the output of the transformation needs to be subjected to, and a boolean variable which indicates whether the target activation layer output is a tuple, a case that occurs for LSTM layers for example, such that the transformed activations are also formatted accordingly.

We will only describe the procedure in the predict function that occurs for one direction, since the procedure for the other is very similar to that. In a very similar fashion as in section 3.3, the first step is the application of two hooks to the chosen layers of the models. The layers are set in a list at the initialization of the algorithm using [23] for accessibility purposes, and as such we only need the positions of the layers in the list for the placement of the hooks. The first hook is placed on the input model and has the purpose of storing the activations temporarily in a variable and stopping the forward pass by throwing a custom error, and the second hook replaces the activations of the target model at the chosen layer with the transformed activations of the first model, stored in the temporary container. In the case of Experiments four and five, the second hook also ensures that the sequence length of the output of the chosen activation layer of the second model is preserved by either trimming them if the initial output of the chosen activation layer was shorter, or padding with mean values of the activations if the initial output was longer. This will be explained in more detail in section 4.4.

After the hooks are in place we will iterate through the list of inputs and for each we will perform the following operations. We will first call the forward pass of the model with the current input. The contents of the temporary container are copied then permuted and reshaped to bring them in a form suitable for the transformation. If necessary, only the outputs of the chosen neurons will be selected. If only a subgroup of the sequence was used for the computation of the transformation, then the sequence is split in subgroups of that length and distance. The algorithm then iterates through them until the whole sequence is transformed. The diffusion map space representations outputted by the transformations predict function will also be stored for visualisation and evaluation purposes. For either the whole output or the current subgroup the transformation is performed and stored in an array that will contain the transformation of the whole sequence. That array will also contain positions for the neurons in the target activation layer for which the transformation is not defined. While at first the transformed activations of those neurons were replaced with zeros, after a number of experiments with this function we have concluded that they should be replaced with the mean of the outputs of the transformed activations, since this implementation choice lead to better performances in section 4.2. After the transformed output of the target network was computed, it is stored and then placed in the temporary container.

With the transformed activations placed in the container, forward pass of the target network is performed and its output is stored. The input of this forward pass varies between experiments and it will therefore be addresed individually in all the sections of chapter 4.

After iterating through all the inputs, the hooks are removed and all the stored variables are returned.

4 Experiments

After presenting the theoretical basis and the way we implemented the presented concepts we will now show, through five experiments, that our implementation is correct, and we will present the capabilities of the presented ideas and our implementation thereof.

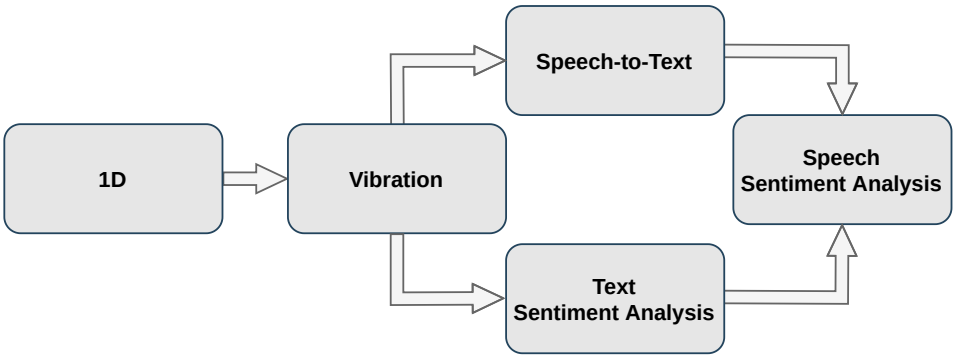


Figure 4.1: Structure of the Experiments

Figure 4.1 presents the main thought behind the ordering of the experiments. The first experiment has multiple purposes. First it is intended to verify if the implementation is correct. Since the exact same models and transformations were used in the first experiment of the paper [1], we could investigate the implementation and its midway results step by step, and compare it with the results presented in [1]. Thus, based on how the transformations therein were performing, the implementation was debugged and changed. The second purpose is that of presentation. Since the function is defined on a one dimensional manifold, the range of the function is a one dimensional manifold, the activations are 1 and 8 dimensional and the embedding dimension is only 3, we can plot the whole process in a understandable way. This enables to accompany the process description with images.

The second experiment functions as a stepping stone towards the next two experiments. It introduces transformations on sequence data, a category in which both speech and text data are included. This experiment will also include a proposed solution for the problem of implementing transformations for data with variable sequence length, which, while not necessary in this case, does also function as an optimisation of the algorithm for this experiment. The dataset used was created during the interdisciplinary project "Deep-learning based approaches for fault detection in a rotary mower" [16], which contains as inputs segments of one hundred timesteps of the recording of six vibration sensors on a rotary mower and as target a boolean value which indicates the status of the machinery: functioning or faulty. The models used were also implemented during said interdisciplinary project, and

both the implemented models as well as their trained weights were used for this thesis. Both models include at least a layer of higher complexity than the layers used in the first experiment, and both have different complex layers. The LSTM-linear model includes a LSTM [5] layer and the 1DConv-transformer-linear-Llnear model includes both a 1D convolutional layer and a transformer layer. The implementation of the transformations for models containing different layers with higher complexity is also a stepping stone towards the implementation of open source pretrained models in experiment 3 and 4, since both include large networks with different, complex layers.

The third experiment implements transformations between neural networks trained for ASR (Automatic Speech Recognition). The two networks used, a Speech2Text2 [7] network and a Wav2Vec2[8] were implemented in the Huggingface library [4], with pretrained weights taken from two projects, mentioned in section 4.3, hosted on the Huggingface platform. Both networks have a much larger number of layers than the models in the previous experiments and both have a higher degree of complexity. The number of activations on the chosen activation layers also increases drastically, and only a selection of them can be used. Since the Wav2Vec2 model has different outputs, written sentences of the inputted speech, than those of the Speech2Text2 model, written sentences of the inputted speech translated in German, it is also the first experiment with two neural networks used for different purposes, a characteristic which also occurs also in the fifth experiment.

The last step needed towards the final experiment is presented in the fourth experiment. In this experiment a transformation is computed between two neural networks that perform text sentiment analysis. A RoBERTa model [10] and an XLNet model [3] were chosen for this experiment. Both of their implementations were again found in the Huggingface library [4], and the pretrained weights, uploaded in two projects mentioned in section 4.4, were also downloaded from the Huggingface platform. This experiment will introduce the generation of neighborhoods on the embedding space instead of the input space and the changes implemented for the computations of transformations on data of variable, unequal sequence length between two networks, also required for the fifth experiment.

The fifth and final experiment presents our attempt at using two pretrained models and a transformation to build a new model with a different usecase than that of the component models. We will take a ASR model and a text sentiment analysis model and build from those a speech sentiment analysis model using a transformation. The samples, their activations and the Mahalanobis kernel Diffusion Map spaces generation procedures from the Wav2Vec2 model[8] from the third experiment and those from the XLNet [3] Model from the fourth experiment will be reused in this experiment. As such, the two experiments can be seen as a necessary components of the final experiment. This thesis' conclusions will also take a larger focus on this experiment.

4.1 1D-Data

Since this thesis was inspired by the paper "Transformations between deep Neural Networks" by Tom Bertalan, Felix Dietrich and Ioannis G. Kevrekidis [1], and implements methods

presented therein, we have decided for our first experiment to reproduce the first experiment from the aforementioned paper.

For this experiment we have used the function $f(x) = -x^2$ defined on the manifold $M = [-1, 1] \subseteq \mathbf{R}$ as the function that both neural networks, N_1 and N_2, attempt to learn.

For the first neural network, N_1, we have chosen a network composed from two linear layers, which take one input feature and output one feature ($x_{in} \in \mathbf{R}$ and $x_{out} \in \mathbf{R}$) and a tanh activation between them. The last linear layer is the output layer of the network.

For the second neural network, N_1, we have chosen a network composed from one linear layer with one input feature and eight output features, two linear layers with eight input features and eight output features and a linear layer with eight input features and one output feature. The last linear layer is again the output layer of the network. Between all linear layers there are tanh activations.

We have generated for both Netwoks the same dataset on which they will be trained. The training dataset consists of 10240 input values generated by random sampling from a uniform distribution on the interval $[-1, 0]$, and their target values computed using the function $f(x) = -x^2$. The validation dataset consists of 512 input values generated by random sampling from a uniform distribution on the interval $[-1, 0]$, and their target values computed using the function $f(x) = -x^2$.

While we assume that our function is defined on the manifold $M = [-1, 1] \subseteq \mathbf{R}$, we will train both networks using only samples from the domain $[-1, 0] \subseteq M$, so that we can see how the transformations perform for the networks on a subset of data for which the networks will under-perform.

Both neural networks were trained using the pytorch lightning library [22], with the Mean Squared Error loss [28], implemented by the pytorch library [29] MSELoss function. The optimizer chosen for the neural networks was an adaptive moment estimation optimizer (Adam) [30] implemented by the pytorch library optimization package function Adam [29]. While the first network was training using a learning rate of 0.001, the learning rate used for the training of the second one was 0.0001. The batch size used was left at the default setting of the pytorch Dataloader [29], namely 1. The training was done using the Trainer object in pytorch lighning, with a maximum numbers of epochs 200 and a early stopping callback which monitors the validation loss, and stops the training if the validation loss does not reach a new minimum for 10 epochs. The first model stops after epoch 52 at a training loss of 2.6748×10^{-5} and a validation loss of 0.1103 and the second model stops after epoch 89 at a training loss of 8.2741×10^{-7} validation loss of 0.0571. The model parameters are saved after the training in order to be able to compare different hyperparameters for the transformations on the same model.

The two models were then used to calculate the function for 51 equidistant values in the interval [-1,1]. The plotting of the results can be seen in Figure 4.2. True shows the values of the function $f(x) = -x^2$ on the domain [-1,1], Model1 signifies the results of the first model on the same domain, Model2 is the label of the line showing the results of the second model on the same domain. As we can see from it, both models have good results on the domain they were trained on, while they are both imprecise on the domain that they were not trained

on. This matches our expectations of the models.

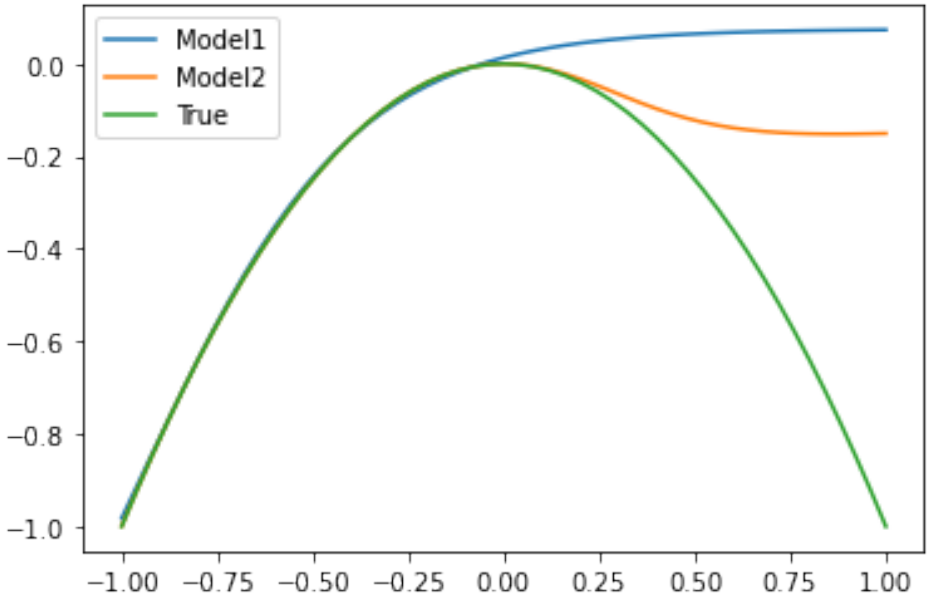
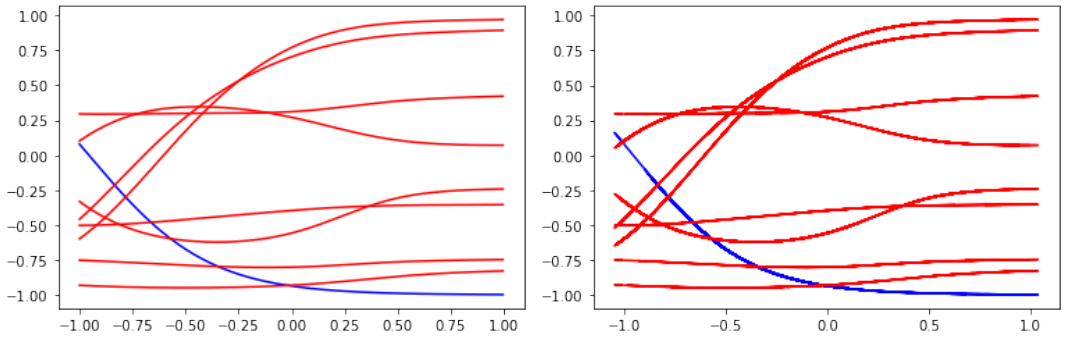


Figure 4.2: Experiment 1 models results

Given the two trained models we can move forward with the generation of the transformation. The first step is the generation of the samples and of their neighbourhoods. As was presented in section 3.2 Neighborhood Generation, we generate the samples equidistantly between the two boundaries of the domain, -1 and 1, and the neighborhoods are generated using the normal function of the pytorch library [29], with mean at the sample for which we generate the neighborhood and a variance of $\delta/4$. For this transformation we generated 512 equidistant samples, with neighborhoods of size 1024, generated with a variance δ equal to 0.05. The same samples and neighborhoods will be used for both of the neural networks.

The second step of the generation of the transformation is the propagation of the samples and neighborhoods through the two models. This was done by attaching to the chosen layer of each neural network the appropriate hook, a concept presented in chapter 3. In this case we have chosen for the first model the tanh activation layer, and for the second model the last tanh activation layer. These layers are found by listing all the layers of the model and getting the element in the list at the positions of the layers. The same hook is attached to both layers. The hooks functionality is that of saving the output of the layer in a list. That list is a member of the activations generator object and as such we can use it as a temporary container for the activations, which stores them during the forward pass of the neural network of each sample and point in a neighborhood, passes them to the output of the generation function, and is afterwards emptied in order to repeat the process for the next sample or member of the neighborhoods. Each sample is passed to the neural network independently, as is every point in the neighborhood. Afterwards the hook is detached, since it prevents the full pass of the neural network for time optimisation, and the models need to be reused in a later step.



(a) Sample activations

(b) Neighborhood activations

Figure 4.3: Sample and neighborhood activations of both models 1&2

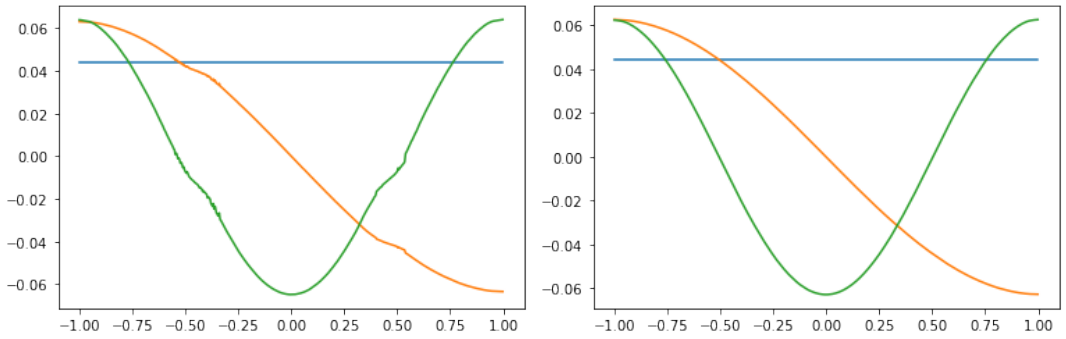
After generating both the sample and the neighborhood activations, we have plotted them in order to have a picture of the activations that the transformation needs to generate in both directions. Figure 4.3 shows the aforementioned plotting. The blue line indicates the activations generated for each point in the [-1,1] domain by the first model. Since the layer has only one output, only one curve is generated from the points. The red lines show the activations generated for the same domain by the second model. Since for each input value there are eight values that the last tanh layer outputs, we can see 8 distinct curves. We can see that the plot of the activations of the samples is very close to that of the neighborhoods, which is the expected outcome and a indicator for the fact that the samples cover the domain well.

The samples, the neighborhoods, the sample activations of both models and the neighborhood activations of both models are all stored in ".pt" files at this point, since we wish to avoid their generation every time the transformations need to be computed.

Having all the component parts required we can now generate the transformations. We start by computing the covariance matrices from the neighborhood activations of each sample for each model. As stated in chapter 3, these are required by the diffusion map algorithm for the understanding of the topology around each sample activation that they provide. As such we compute the set of covariance matrices $C_1 = \{\Sigma_1^k | k \in [0, n)\}$ for the first model and the set of covariance matrices $C_2 = \{\Sigma_2^k | k \in [0, n)\}$ for the second model, with $\Sigma_p^k \in \mathbf{R}^{m_p \times m_p}, p \in 1, 2$, where n is the number of samples, p is the number of the model, and m_p is the number of features that the flattened activations contain for each point in the neighborhood for model p.

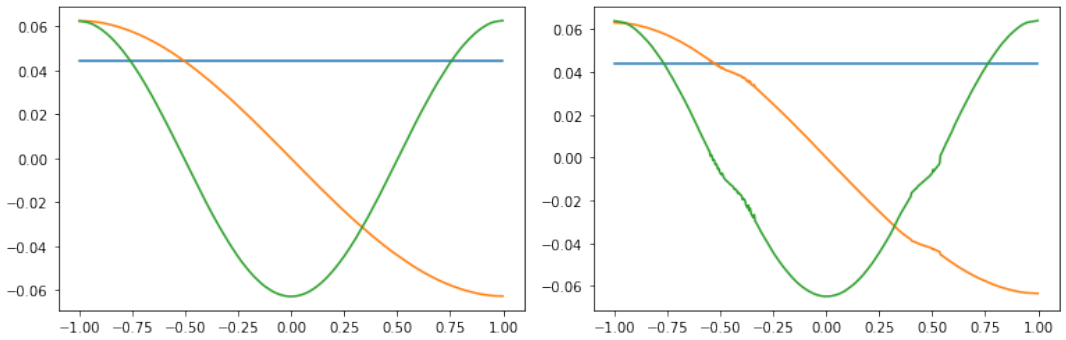
The covariance matrix list of model 1 and the sample activations of model one are then passed to the diffusion map algorithm received from Dr. Felix Dietrich at the start of the thesis, along with a number of hyperparameters explained in chapter 3. For the first experiment we have chosen the number of diffusion map eigenvectors as 3 for both models, the k_min parameter as 50 for both models and the smoothing parameter as 0. The smoothing parameter was chosen as 0 since there has been noticed that no regularization was the best option for the interpolations in these transformation during their testing.

We then generated both diffusion map objects using the datafold library[19]. The first diffusion map generation yielded an epsilon value of 2.805×10^{-6} and the second one an epsilon value of 8.882×10^{-6}. The sample activation values were then mapped to the diffusion map space using the transform function of their respective diffusion map objects. We then compute the interpolation from the sample activation values to their diffusion map representation. The values generated by the interpolation for both models for the samples can be seen in Figure 4.4.



(a) Model 1 (b) Model 2

Figure 4.4: Model 1&2 diffusion map space representation of the activations of inputs from the domain [-1,1]

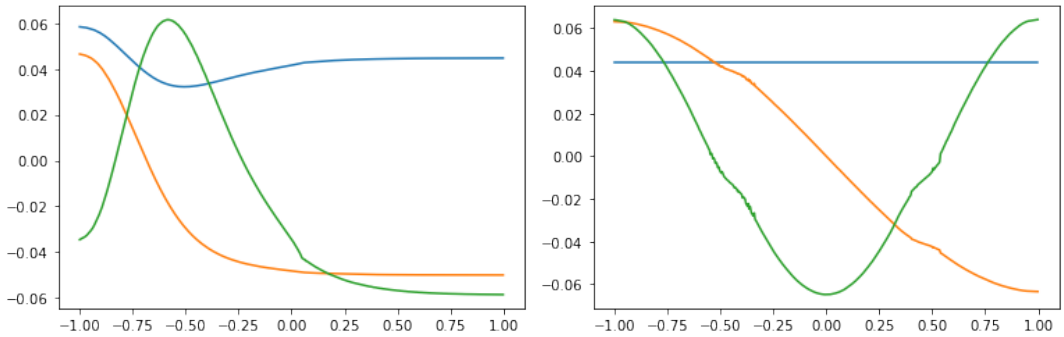


(a) Model 1 (b) Model 2

Figure 4.5: Model 1&2 Orthogonal transformations of the previous diffusion map space representations

With the representations we use the orthogonal procrustes algorithm to create the orthogonal transformations between diffusion map spaces. The first column of the representations are always constants, and as such the algorithm divides one constant by the other and sets on the first position of the diagonal said value. The matrix excepting the first row and column is computed by using orthogonal procrustes for the diffusion map space representations

using all but the first column of both diffusion map eigenvectors. Figure 4.6 shows the difference between using the orthogonal procrustes algorithm with all of the columns of the eigenvectors, the initial attempt, and using the procedure presented above. In that figure we can see that the new idea seems to function much better in this case than the previous one, especially for the constant eigenvector, which remains constant after the transformation. In the previous figure you can see the orthogonal transformation of the diffusion map spaces of the two models. Figure 4.5.



(a) Without improvement (b) With improvement

Figure 4.6: Orthogonal transformations from the model 1 diffusion map eigenvectors with and without the described improvement

Finally, we interpolate from the orthogonal transformations back to the activation space of the other model. Since the mapping is done to the activation space of the model different from the initial, they can be visualized by looking at Figure 4.3 and switching the models. For the interpolations we use a RBFInterpolator (Radial Basis Function Interpolator) object from the SciPy library [20] with thin plate splines. This procedure was selected during this experiment, since the initial plan was to use the Nyström extension [31] to map new points into the diffusion map space and to use an GeometricHarmonicsInterpolator object from the datafold library [19] to do the inverse interpolation. The use of RBFInterpolator objects in both directions since in during multiple runs of the experiment it outperformed the other options.

Thus, with two interpolations and a matrix multiplication we can now map from the activation space of one model to that of another. The mean squared error between the original activations of model 1 and the transformations of the activations of model 2 is 6.693×10^{-13} and the same metric between the transformations from the model 1 activations and the original model 2 activations is 8.848×10^{-16}. This indicates that the implementation was done in a correct manner, since the error was expected to be negligible.

The final step of the first experiment was to test the performance of the algorithm on data that it has not used for training. In that sense we select 1024 equidistant values over the domain [-1,1], to which we added a positive shift of 5×10^{-5}. By using a testing set twice the size of the training set we ensure that at least half the values were not used during training.

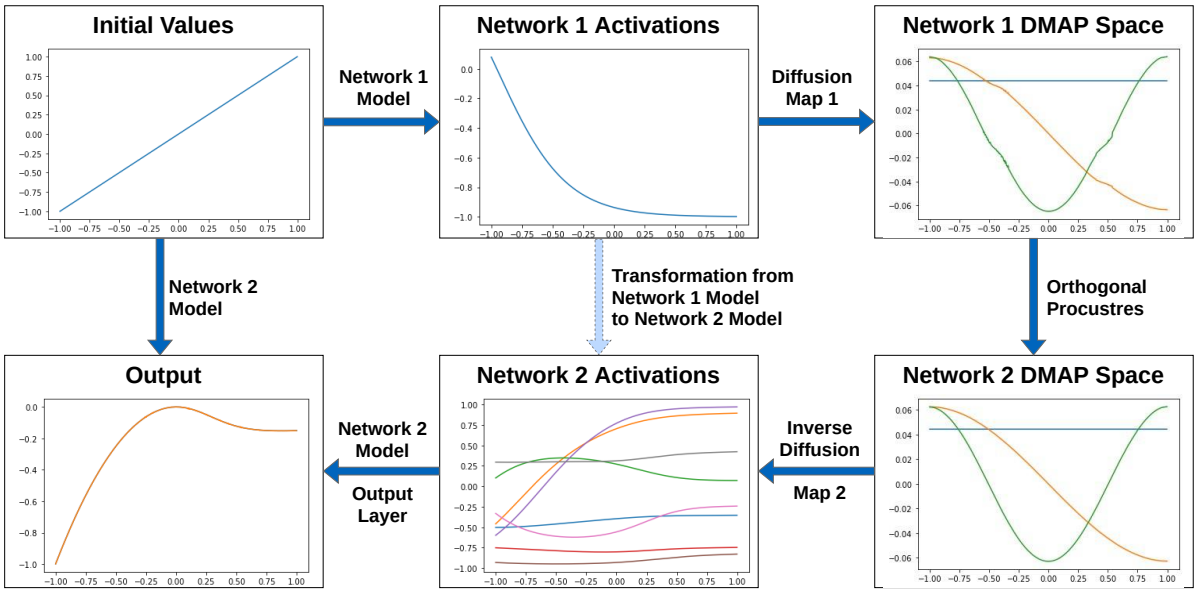


Figure 4.7: Transformation from network 1 to 2

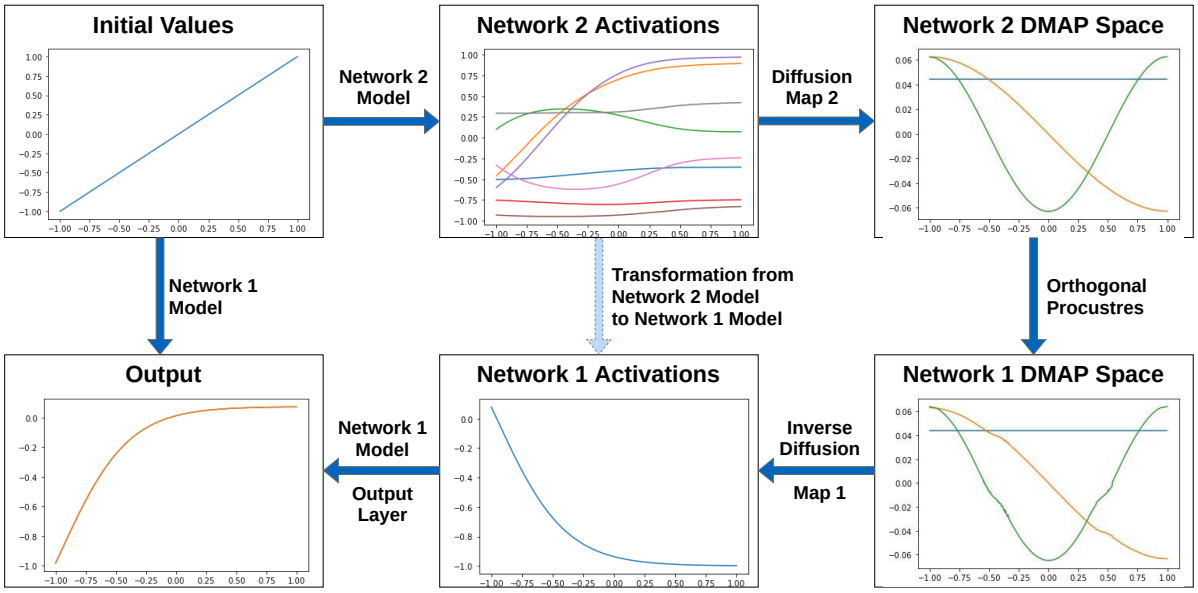


Figure 4.8: Transformation from network 2 to 1

Because the samples not in the training set are located generally at the middle point between two points that were, we can also look at the behaviour of the transformation at those points farthest away from its training set, and thus hope to find all errors.

For the purpose of testing, we place a hook which stores the values of the output at the chosen activations layer of the initial model and a hook which replaces the values of the output at the layer of the final model with the transformed activations.

The algorithm reaches a mean squared error between the output of the second model and the output of the transformation of the first to the second model of 2.198×10^{-8}. The mean squared error between the first model and the transformation to the first model reached was 1.339×10^{-7}. The mean squared error between a diffusion map space representation and the orthogonal transformation of the other was approximately 2.38×10^{-4} in both directions.

The test values are individually propagated through the neural networks up to the point where activation hooks are placed. The algorithm then takes them, flattens them and creates their transformation using the hyperparameter direction, which determines which model is the initial one and which is the final one. The transformed activations are then placed in another array, which replaces at the next forward pass through the final network the activations at the chosen layer, thus generating the output of the neural Network. Figure 4.7 and Figure 4.8 show the progression of such a test through the whole algorithm in both directions.

4.2 Vibrations

The second experiment will lay the ground work necessary for the next two, since multiple concepts implemented and added to the code basis for this experiment will also be used further on.

Both the models and the dataset used during this experiment were developed during the interdisciplinary project "Deep-learning based approaches for fault detection in a rotary mower" by Victor-Constantin Stroescu [16], and were used with the permission of the advisor to that interdisciplinary project, Dr. Ertug Olcay.

The dataset on which both models were trained is a binary classification set with inputs and class labels. Four features of the inputs are outputs of two VSA005 vibration sensors, both recording vibrations in two spatial dimensions, placed on a Krone easy cut 320m mower. The other two are the speed of the main machinery and the power at which the mower operates at that point. These measurement recordings, which are all real-valued numbers, are split in windows of 100 timesteps recorded with a frequency of 5000 Hz, which represent 20 milliseconds, during which the mower is powered. These windows are then the input data of the dataset. The labels are a boolean array of length 14, with each position in the array indicating whether the blade of the mower at that position is faulty. The labels need to be collapsed in a single value that identifies whether any blade was faulty or not, since that is the operations that the models are trained to perform. The dataset contains 499.874 items with 50.37% positive labels and 49.62% negative labels, and as such is baslanced, and is split into 80% training samples, 10% validation samples, and 10% testing samples [16].

For this experiment we will use 650 samples with neighborhoods of size 1024. All of the samples are on a manifold $M \subseteq \mathbb{R}^{100\times6}$, which influences how the neighborhoods will be computed. As was previously discussed in subsection 3.2.2, the boundaries of the manifold are not known, and as such the 650 samples are randomly chosen from the validation dataset. The neighborhood will be generated as discussed in the implementation section Figure 3.2, namely we sample a 600 dimensional multivariate Gaussian distribution with the mean set at

the sample and the variance, δ, equal for all dimensions and set for this experiment at the value 0.005.

The two pretrained models used in this experiment are also a product of the aforementioned interdisciplinary project. The models were developed with the explicit purpose of detecting in real time if a fault was present during the usage of a rotary mower. For this purpose both the number of timesteps required for the computation had to be low, which explains why each sample has the length on the time domain of 20ms, and that the models used had a low amount of layers, and thus a smaller complexity, in order to shorten the computation time of the forward pass as much as possible without a steep decrease in performance.

The first model used is a model composed from a LSTM [5] layer and a linear layer, both implemented using pytorch library [22] modules, which enables the usage of hooks. The LSTM layer reduces the feature size from six to 1 and the linear layer interprets the 1 feature of all 100 timesteps and outputs a single value. The activations are then generated for both the samples as well as the neighborhoods by placing the hook at the output of the LSTM layer, since the output of the LSTM layer is computed by adding two sigmoid layer outputs and one tanh layer output, and as such can be interpreted as the output of activation layers. Since the output of the LSTM layer has 100 timesteps and only one feature, we can consider that the sequence length of one sample activation is 100 and that the number of neuron outpus of one timestep in the one sample activation is 1. The pytorch library[22] implementation of the LSTM layer also has as output a tuple, since, along with the desired output the layer also returns the final cell and hidden states. Using this knowledge we were able to compute the sample activations and neighborhoods for the first model and store them for later use, as we did in section 4.1.

The pytorch library implementations of the 1D convolutional layer, of the transformer encoder layer [6], and of the linear layer are the pieces constructing the second model. Because of this the usage of hooks is also available for this model. The model, which has the exact same input as the first one, starts its forward pass with a one dimensional convolution, which does not modify the number of timesteps in any way but increases the number of features from 6 to 64. Two sequential transformer encoder layers then process the output of the 1 dimensional convolution, without making any changes to the dimensions of the data. The final two linear layers condense the output of the transformer encoder layer. The first linear layer aggregates the information of all timesteps into one value for each feature, and the last linear layer uses that value for all the 64 features to create one single valued output. The hook used to extract the activations of this model will be placed at the output of the last transformer layer based on a similar motivation as was the case for the first model. The output of the last transformer encoder layer will consist of 100 timesteps of 64 features. This means that the sequence length of one sample activation will be again 100 and that the number of neuron outputs for each time step will be 64. Out of those 64, we have chosen to record only 16, since we believe that the 16 were enough to fulfill the conditions posed by Whitney's theorem [11], and thus that the intrinsic dimension of the data for one timestep was 7 or lower. The 16 neurons chosen from the 64 were the ones which. when numbered, would have their corresponding index divisible by 4. For this second model we also computed and

then stored the sample and neighborhood activations needed to compute the transformation.

Now that the samples, their neighborhoods and the activations of the two are available, we can now compute the transformation. For this transformation we have chosen 60 eigenvectors for both dimensions and a k_min of 35. The computation is similar to the one performed for the first experiment with the difference being the sequential nature of the data used for this experiment. For us to be able to efficiently compute the transformations, the decision was made not to use the full sequence in the activation for the computation of the transformations, but different amounts of equidistantly placed points in the time domain, with the purpose of decreasing the dimensionality of the activations which were transformed. We started with 4 slices of 25 points, each having a distance of 4 timesteps to their neighbors. The 4 slices were treated as different activation samples, in order to permit the transformation to be able to map out the full space. To this purpose we needed to reshape the arrays for both the samples and the neighborhoods from 650 samples of 100 timesteps to 6500 samples of 10 selected timesteps. Since, as the following results will show, this attempt was functional and returned proper results, we made the decision to optimize the computation further for the next test, by creating 10 slices of 10 equidistant time steps and then treat the slices again as different activation samples. Based on the fact that this approach worked, we continued these experiments with 50 slices of 2 points in the time domain and finally 100 slices of 1 point. We established how well these attempts performed by evaluating the two combinations of the two models with the transformation or its inverse.

The computation of the outputs of the combinations of the two models and the transformation or its inverse is also similar to the procedure previously presented in the first experiment, but adapted both to sequential data and to the usage of an incomplete activation layer. Additionally, we have taken into consideration the different permutations of the dimensions that the layers perform, since transformer sets the neuron activations in the first dimension, the batch dimension in the second and the sequence in the third, and the LSTM permutes the activations dimension and sequence dimension. As such both the input to the transformation and its output had to be permuted using the pytorch library [22] function "permute" so that they match both the dimensions required by the transformation and the output that the target layer was supposed to have. At the same time we have also accounted for the LSTM layer having the aforementioned tuple outputs. The transformation output from the transformer model to the LSTM model was set for this purpose in a tuple form, such that the following step of the forward pass would run as intended.

In order to use the transformation trained on a subset of equidistant time steps, we split the time domain of the activation to be transformed into k subsets of p equidistant time points, with p equal to the size of the subsets on which the transformations were trained, which will all be individually transformed into the same k subsets of p time steps of the target activations. This creates the important condition on the subsets, that they are created such that the length time domain is divisible by p. This is the reason why for the experiments we have chosen the k values 25, 10, 2 and 1. The amount of time points in each subset on which the transformation was trained can be seen as a indicator of how much context data was required in the time domain at this point of the forward pass to produce the results presented

further.

The other aspect that needs to be addressed for the first time is the usage of an incomplete output of a layer for the transformation, since the transformation was trained only on 16 from the 64 features of the output of the transformation. In that sense the transformation from the transformer encoder output to the LSTM output does not need any adaptation, but the reverse is not true, since the other 48 neurons that the transformation from the LSTM layer output to the transformer encoder layer output does not map need to be filled with a value in order for the forward pass to be able to commence from that layer. With that purpose in mind we initially filled the empty values in the output of the transformer layer with zeros. After a number of experiments we have determined that a better approach is to fill it with values equal to the means of the predicted neurons for each time step. It is unclear why this approach worked better, but our intuition is that the mean is a more neutral value for the output of these layers than zeros, since the weighted sums computed by the proceeding linear layer might reach closer values to those that would have been computed during a forward pass with the true values using the means than using zeros, while the zeros might impact the proceeding layers strongly by triggering a steep decrease of the final weighted summations.

The input for the forward pass of the target model was set to dummy array, filled with zeros, with the same shape as the input to the forward pass of the initial model.

LSTM	**accuracy**	**precision**	**recall**	**F1-score**	**True negative**
Initial	0.8789	0.8168	0.9926	0.8961	0.7526
25 timesteps	0.8740	0.7956	1.000	0.8861	0.7529
10 timesteps	0.8760	0.8028	0.9981	0.8899	0.7529
2 timesteps	0.8828	0.8110	1.000	0.8957	0.7642
1 timestep	0.8604	0.7766	1.000	0.8742	0.7287

Table 4.1: Transformation to LSTM results for ivbrations

Transformer	**accuracy**	**precision**	**recall**	**F1-score**	**True negative**
Initial	0.9658	0.9565	0.9796	0.9679	0.9505
25 timesteps	0.8184	0.8135	0.8069	0.8102	0.8289
10 timesteps	0.8428	0.7937	0.9280	0.8556	0.7569
2 timesteps	0.7988	0.8023	0.7961	0.7992	0.8016
1 timestep	0.7793	0.7683	0.7807	0.7745	0.7780

Table 4.2: Transformation to transformer results for vibrations

Table 4.1 and Table 4.2 present the results of this method on 1024 test samples with different subset lengths on the time domain. The initial column shows the results that the LSTM based model and the transformer encoder based model achieve without using any transformation.

Table 4.1 shows, excepting the initial column, the results of the models built using the transformer encoder based network as initial model and the LSTM based network as the target model, with a transformation as previously discussed, using different sizes of time

step subsets. These attempts are compared to the LSTM benchmark since the target is that the transformation maps correctly to the activations of the LSTM and as such the outputs of the combined model match those of the LSTM model. The same considerations apply to Table 4.2, but with the LSTM model as initial model and the transformer encoder model as target model.

The first conclusion of the results is that our models achieve very good results with LSTM target models and satisfactory results with transformer target models. The second conclusion is that there is only a mild decrease in performance when experimenting with smaller subsets of the time domain, which indicates that the activation manifolds can be split into a number of quasi independent manifolds on the time domain, all having enough in common that a transformation trained on sets of one time step can map from all time step manifolds in the input domain to all time step mainfolds in the target domain independently without a great loss in performance .

4.3 Automatic speech recognition

For the third experiment we have chosen pretrained neural networks designed for automatic speech recognition. This experiment will introduce both concepts used further on for the fourth experiment, as well as the model and the sample, neighborhood and their activations generation used for a the fifth experiment. As such it can be interpreted that this experiment also covers a little under half of the procedure required for the fifth experiment.

For this experiment, the Librispeech dataset [9] was chosen for the training of the transformations, since it was either the full or part of the dataset used for training, validation, and testing of the models for which we compute the transformations. The Librispeech dataset is an open source dataset containing 1000 hours of English speech read from audiobooks. At each index of the dataset there is a list containing 5 entries, of which only two interest us. The two interesting to us are the text and the speech components of the dataset. The speech component contains speech recordings of up to 35 seconds sampled at 16.000 Hz, while the text component contains the text read in order to create the speech component. Since not all readings are done over the full 35 seconds, the speech component has a variable length, with a maximum length estimated, by multiplying the maximum time with the sampling rate, at 560000 values. Since the values are a time series, we can thus view the sound data as having one feature and a variable sequence length. Even though the text component is also a big part of the dataset, for this experiment it will only function as the target, with further analysis of this component being done in section 4.4, where this component is of more interest.

Two model definitions have been chosen for this experiment, both having been downloaded from the Huggingface library[4]. The training weights of these models have also been taken from the Huggingface platform.

The first model chosen was a Wav2Vec2 model, first proposed in "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations" by Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, Michael Auli [8]. The model is formed from a feature encoder created by stacking blocks built from a temporal convolution, a layer normalization and

a GELU activation, which creates from the sound data latent speech representations, that are then fed, together with the inputs again, into a transformer [6], that outputs the text embedding [8], which then is decoded using an embedding decoder implemented in the Huggingface library [4]. The training weights were taken from the project "wav2vec2-librispeech-clean-100h-demo-dist" by user patrickvonplaten [2] on the Huggingface platform.

The second model used was a pretrained version of the model Speech2Text2, proposed in "Large-Scale Self- and Semi-Supervised Learning for Speech Translation" by Changhan Wang, Anne Wu, Juan Pino, Alexei Baevski, Michael Auliand Alexis Conneau [7]. This model's decoder is also based on the Wav2Vec2 model, but adds afterwards a 24-layered transformer [6]. For the decoding process it uses a seven layered transformer that outputs the embeddings of the text. An embedding decoder implemented in the Huggingface library [4] is also used to receive the string representation of the text. The training weights used for this model were taken from the "s2t-wav2vec2-large-en-de project hosted" on the Huggingface platform and presented in the same paper as the model [7]. Different from the first model which converts English speech to English text, the second model converts English speech to German text. This is the first experiment in the thesis where the two networks will have different outputs.

As discussed in subsection 3.2.3, the generation of the samples and neighborhoods will be done by randomly selecting 128 samples from the speech column of the dataset for the samples and then sampling multivariate Gaussian distributions of dimension equal the number of timesteps in the sample, whith the mean set at the sample and a $\delta = 0.05$. The size of the neighborhoods has been set to 75. Since the samples have varying lengths in the time domain, they had to be stored in a list, while the neighborhoods had to be stored in a list, that contains at each position all the neighborhoods for the respective sample. The low number of samples and neighborhoods was driven by the memory limitations imposed for the training of the transformations which were strained by the large size of each sample. Since all the samples and neighborhood activations need to remain in the memory for the computation of the transformation, this measure of limiting the number of samples and size of neighborhoods was necessary.

Sample and neighborhood activation computations were changed in the fact that they needed to be made iteratively, due to the impossibility of bundling multiple samples in a batch, since their lengths are variable. For both models activation layers from the Wav2Vec2 decoder [8] were chosen, since it was considrered that only those two segements were modeling the same phenomenons of speech, since the decoders must model the different linguistic patterns of the two languages. For the Speech2Text2 model [7], we have chosen the 243rd layer, which has a feature dimension of 1024, from which we have chosen 128 neuron outputs to train the transformation on. Our chosen layer of the Wav2Vec2 model [8] was the 28th layer. It has an output feature size of 4096, from which we also chose 128 features for each timestep. The sequence of each samples length remained unchanged during the forward pass of the network until the selected activations. The presented sample and neighborhood activations were stored afterwards and we then commenced with the transformation computation.

The approach of using multiple subsets of time steps of size p for the computation of

the transformation is repeated here. As was previously state, all sequence lengths must be divisible by p for the computation to function afterwards. This is only possible in the case p=1 since the samples have variable sequence length. We know that such a option is feasible due to the last test presented in Table 4.1 and Table 4.2. This then creates the other problem of dimensionality. Since we have 128 sample activations with a maximum sequence length of 560000 with a feature length of 128 for each network, this would presume that we have to compute in the worst case a transformation between two 128 dimensional spaces using approximately 71 million samples. This is not only computationally challenging, but in our view also redundant. For that purpose we selected from each sample activation only 64 randomly selected time steps that are used for the computation of the transformation. We also selected the same 64 time steps from all the neighborhoods of the respective sample. This reduces the number of samples to 8192.

The transformation for these two models were trained using 128 as the number of diffusion map eigenvectors, as proposed in [1], which advises the usage of the feature length as the number of eigenvectors, with k_min for both diffursion maps equal to 50. Since all the sample and neighborhood activations have the same dimensions for one time step, the transformation computation could be done without any additional considerations about the variable sample length and could be performed, and the algorithm from section 3.4 was performed as described.

The computation of the forward pass in the two combined models for testing purposes could now be performed. The computation is similar to that presented in section 4.2, since the sequential nature of the data is present and transformations to incomplete outputs of activation layers occur. Since the variable length of the sequences are always divisible by the number p of time steps in one subset, no other changes caused by the variable length of the data were needed for this computation. The change required on the other hand stems from the nature of the models and not that of the data. The forward functions of the target model could no longer be started with dummy versions of the inputs, since the input is reused at the start of the transformer layer, after the decoder has finished its computation, in both models. Thus the input to the forward function of the target models needs to be the same as that to the initial model. Since the inputs affect the target model, the results that will be presented in the next paragraph should be viewed through the following lens: Without the real input, which have not been transformed in any way, being placed at the start of the target models, the results presented could not be achieved. A work around this type of difficulty still needs to be found in the future if this experiment should work without real inputs for the target models.

Table 4.3 Shows the resulting sentence of the different models used for one example speech outtake. The first row shows the true sentence in English. Since there is no target sentence in German, we have to rely on the Speech2Text2 model that the translation performed is generally correct for our accuracy evaluation. On the other hand we can see in the text of the table that the translation is not completely correct. The second row shows the result of the Speech2Text2 model. The third row presents the sentence computed using the two models and a transformation, with the target model being the Speech2Text2 model.The sentence there

Model	Sentence
True Sentence	'the council of state on hearing of this began also to make ready for eventualities negotiations were still proceeding between the two countries when martin tromp the victor of the battle of the downs'
Speech2Text2	'</s> Als der Staatsrat davon hörte, begann auch die Verhandlungen zwischen den beiden Ländern zu hören, als Martin-Tromp die Schlacht der Downs vorrückte. </s>'
Transformation from Word2Vec2 to Speech2Text2	'</s> Als der Staatsrat davon hörte, begann auch die Verhandlungen zwischen den beiden Ländern, den Siegen der Schlacht der Downs vorzubereiten. </s>'
Word2Vec2	'the council of state on hearing of this began also to make ready for eventualities negotiations were still proceeding between the two countries when martin tromp the victor of the battle of the downs'
Transformation from Speech2Text2 from Word2Vec2	'the council of state on hearing of this began also to make ready for eventualities negotiations were still proceeding between the two countries when martin thromp the victor of the battle of the downs'

Table 4.3: Example of model outputed sentences

also does not match neither a perfect translation of the true value, nor the previous row. The fourth row shows the Wav2Vec2 model results for the example, we can see that is exactly the same as the true value. The fifth row matches also exactly the same the true value and is the output of the combination of the two models and a transformation, with its target being the activations of the Wav2Vec2 model.

Due to a lack of true value for the German sentence we will compare the combined model for speech to German text with the pretrained Speech2Text2 model. In the interest of using the same metric we compare the combined model for speech to English text with the Wav2Vec2 pretrained model. The metric for comparison will be accuracy of the combined model having the same output as the target model. The accuracy for the combined model with target Speech2Text2 was approximatively 0.6971, while the accuracy of the other combined model was 0.8969. This matches our observations on Table 4.3

4.4 Text Sentiment Analysis

In the fourth experiment we will attempt to compute and evaluate transformations between neural networks trained for sentiment analysis. This experiment will introduce concepts further used for the fifth experiment, with the most important one being the computation of transformations for activations with differing sequence length, since the lengths of the outputs of the embedding layer of the two networks used in this experiment differ. Furthermore, the

fourth experiment also covers a little under half of the components used for the fifth one, specifically the generation of the samples, neighborhoods and their respective activations.

The Librispeech dataset [9] will also be used for the training of the transformations evaluated in this experiment. For this training we will use the text column of the items in the dataset. Even though the models were not trained using this dataset, we will use it in the hope that it still covers the text manifold well. The main reason we used to support the usage of the Librispeech dataset [9] is the necessity of the transformations in the two experiments used as components of the fifth experiment was trained on the same dataset. This is the main reason that the Librispeech dataset [9] was used in this thesis, since it contains both speech and text components. The text component in the items of the dataset is a string variable of variable length. The length of the strings is, as expected, noticeably smaller than that of their respective speech corespondent.

We again employed the Hugginface library [4] to choose two widely available model classes for this experiment. As in subsection 3.2.3, we have also used the Huggingface platform to find and download weights for the models trained specifically for the task of sentiment analysis on English text.

An implementation of the XLNet model, first presented in "XLNet: Generalized Autoregressive Pretraining for Language Understanding" by Zhilin Yang et al. [3] was chosen as the first model, with weights sourced from the project "xlnet-base-cased-SST-2/tree/main", attributed to the user "textattack" on the platform [17]. The model is described in [3] as having two-stream attention with a Transformer-XL backbone, and uses Transformer-XL layers combined with layer normalizations and GeLU activations. The model outputs a vector with two values, the maximum of which indicates weather the sentence is positive or negative.

Since the article presenting the XLNet model tends to often compare its results with the ones achieved by BERT and RoBERTa models, we have chosen a the RoBERTa model, first presented in "RoBERTa: A Robustly Optimized BERT Pretraining Approach" by Yinhan Liu et al. as the second model. Its implementation was taken from the Huggingface library, with the weights sourced from the project "sentiment-roberta-large-English" by author "siebert" presented in [18]. This RoBERTa model implementation consists of a pretrained BERT model, which has a transformer architecture, and linear layers which output the final two valued vector which indicates the positive or negative nature of the sentence.

As we have done previously, for this experiment we will randomly select 128 items from the dataset as the samples on which the transformation will be trained. We have previously discussed in subsection 3.3.3 that the neighborhoods can not in this case be directly generated on the inputs to the neural network, since the text data is discrete. For this reason we will generate two sets of neighborhoods, one for each neural network, at the embedding level the neural network, by sampling 128 values from a a multivariate Gaussian distribution of dimensionality equal to that of the embedding, the mean set at the sample embedding and a variance $\delta = 0.05$. Since the generation of the neighborhoods is done seperately at the embedding layer of each model, we can not ensure that the diffeomorphism between the neighborhoods exists and that it is also a diffeomorphism between the embedding spaces.

The neighborhood generation procedure will be done during the activation generation step

of each model for both the samples and their neighborhoods, as discussed in subsection 3.3.2. For the XLNet model [3] we have chosen the output of the 26th layer as the activations on which we will train the transformations. The output has a total of 768 features for each point in the sequence, from which only 24 equidistant features of each sequence point were chosen to be the subject of the transformation. The output of the 116th layer is chosen as the activations for the RoBERTa model [10], which has 1024 total features, from which we select 16 equidistant features for each time step. The embedding layer, where the neighborhoods were generated, was the first layer of both models. Since the number of sequence points generated in the embedding layer of the RoBERTa model [10] is lower than that in the XLNet model [3], the activations of both the samples and the neighborhoods of the XLNet model had to be trimmed down in the sequence dimension to the sequence length of the activations of the RoBERTa model. This is performed, since the activations need to be of equal size in the sequence dimension, so that the transformation from each sequence point of one models activation has a target set in the same sequence point in the activation of the other model. Since the size difference is not large, the amount of information lost for the training of the transformation was decided to be acceptable.

Before storing, the same procedure is performed as in section 4.3. Only 64 points in the sequences are selected from each sample and the same time steps are selected from the samples neighborhoods, which are then treated as different samples and their neighborhoods for the training of the transformation. While this is not as necessary for this experiment as it was for the previous one, we still performed this operation in order for the two experiments to have same procedure up to the generation of the transformation, since both of these will be then used in the final experiment.

With the activations of the samples and those of the neighborhoods we can now compute the transformation mapping. The same procedure as presented in section 4.2 and section 3.4 will be followed. The final sample count was 7218, and the transformation was computed using 50 eigenvectors for both diffusion maps and with a k_min hyperparameter equal to 100.

Having computed the transformation, the final phase, consisting of testing, can now begin. The computation of the forward passes of the two possible combined models is similar to the one presented in section 4.3, since the same hurdles appear: variable sequence length and the selection of only select activation features of both models. The same approaches are taken in this experiment as in the previous ones. On the other hand an additional difficulty is presented by the inequality of the sequence lengths of the two activations. This is a problem since the target model needs to continue the forward pass with an appropriate number of activations features as would have been generated by a forward pass with the true input. To counteract this problem we have found a suitable approach in starting the forward pass of the target model with the input of the initial model and not a dummy structure of the same shape. This is the same approach used for section 4.3 to counteract a different problem that was in that case created by the nature of the models themselves. We use the activations created by the forward pass to determine which shape the activation computed by the transformation should take. If the output of the transformation has a larger sequence length than the activations of the forward pass of the target model would indicate, then the difference is trimmed from

the end of the sequence. In the other case additional positions need to be appended to the transformed activations. Those additional positions are computed by calculating the mean of the whole sequence for each feature individually, and replacing each missing sequence positions at the end of the activations sequence with a mean feature array. Both the trimming and the padding with means operations are performed by the hooks which are placed in the target model. The two boolean parameters which indicate whether a model has a larger sequence length than the other need to be properly set so that the proper hooks operations are selected. This solution used for the inequal sequence length of the activations will also be used in the last experiment.

nr. samples	XLNet	RoBERTa	XLNet (positive)	RoBERTa (positive)
100	0.86	0.64	0.06	0.4
500	0.966	0.892	0.068	0.392

Table 4.4: Experiment 4 test results

Using this algorithm for the computation of the forward pass of the two models which emerge by combining two pretrained models we have performed two tests on different sets of samples chosen at random from the Librispeech dataset [9], their result presented in Table 4.4. Two important comments to this experiment are that the two experiments were performed using transformations trained on different, randomly selected samples and not the same transformations, and that, as in section 4.3, the computation of the forward pass of the target model was started with the inputs of the initial model, and without that measure such results could not be achieved during this thesis.

The columns **XLNet (positive)** and **RoBERTa (positive)** are reference values that show what proportion of the 500 samples the two models identify as having a positive connotation. The results here indicate a high number of sentences with negative and neutral connotation, and a underrepresentation for positive connotations. Since the dataset was not created for sentiment analysis, an imbalance in the dataset is possible. The results in the other columns are as in the previous chapter accuracy of the combined model with the indicated target model relative to the outputs of a forward pass of the target model. We can see in the 100 samples row that the transformation trained at that point did not have such a good performance either for the XLNet target model or for the RoBERTa target model, as those trained for the test using 500 samples, which have impressive performances.

4.5 Speech Sentiment Analysis

The last experiment in this thesis will be our attempt at building a new speech sentiment analysis model by using two previously presented models and a transformation.

For the training of the transformation we will use the Librispeech dataset [9], specifically both the text and sound column. For this experiment we can not assume we have a diffeomorphism between the sound data domain and the text data domain, since some of the conditions required are not fullfiled. Any function from the sound data domain to the text data domain

is clearly not bijective: the same text can be read by different people with different accents and create different sound data. Since it is not bijective, it can not be invertible. We hope that the surjectivity of the function, since for each text there is at least a sound file which represents its reading, is enough in order to create a non invertible transformation from the Automatic Speech Recognition network to the text sentiment analysis network. The surjective function chosen by us for the samples was the relationship between the speech column and the text column of the chosen dataset.

For the last experiment we will use two pretrained networks introduced in the previous experiments. From the third experiment we will adopt the Wav2Vec2 Model[8] with the pretrained weights used in the same experiment and sourced from [2]. The Wav2Vec2 model was chosen due to the fact that it was trained to transform English speech into English text, which is useful since we can make the assumption that the two networks model the same phenomenon with a higher degree of confidence than for the Speech2Text2 model [7] which translates the speech to German. The XLNet Model [3] was chosen from the fourth experiment, and was used along with the previously presented weights [17]. This model was chosen because it will always be the target model part of the combined model developed in the fifth experiment and the combined models in section 4.4 with the XLNet target model had better results than their counterparts for all performed tests. All the model implementations and their weights were imported from the Huggingface library and platform respectively [4], where they were hosted.

Every procedure up to the computation of the orthogonal transformation is adopted from the third and fourth experiments, including the sample, neighborhood and activation generation, as well as the diffusion maps. As such we will select 128 samples from the Librispeech dataset [9] and generate neighborhoods of size 100 with a variance $\delta = 0.05$ using the same method as in section 4.3. Since the Wav2Vec2 model showed a very high performance in transforming English speech to English text, we have used it to generate the samples for the second model. This way we can define the surjective function between the two input manifolds as the function approximated by the Wav2Vec2 neural network. After generating the text samples we use the same procedure as in section 4.4 to generate the neighborhoods at the embedding layer of the XLNet model with the same size of 100 and variance $\delta = 0.05$.

The activations of the samples and their neighborhoods are generated in the same way that they are in the experiments which introduce the models. As such, we have chosen the output of the 28th layer of the Wav2Vec2 model, which has a variable sequence length, a feature length of 4096, from which we choose 128 equidistant features, as the sample and neighborhood activations of the first model. For the activations of the XLNetModel we have chosen the outputs of the 26th layer of the network. The outputs also have a variable sequence length and a feature length of 768, from which 24 equidistant features are chosen for the sample and neighborhood activations used during the training of the transformation.

Using the same approach as in section 4.4, we have tackled the problem of training transformations for activations with different variable sequence length by truncating the longer sequence, in this case the ones of the Wav2Vec2 model activations, to the length of the

other activation sequence of the same sample and neighborhood. In this case the truncation is much more impactful, since the size difference between the sequences in the activations of the Wav2Vec2 model and those in the XLNet model activations is high. This is overlooked accordingly, since the target model is the one with the shorter activation sequences, and as such if we should able to predict from a truncated version of the Wav2Vec2 model activations the full XLNet model activations.

As presented in section 4.3, we have used only 64 randomly selected sequence points in each activation, and will treat them as different samples, in the hopes of creating a transformation from each sequence point in one activation to the same sequence point in the other activation.

The sample and neighborhood activations were stored in the presented form and the next step, the computation of the transformation, was then performed. Since the computation of the transformation was designed to be the same as in the previous two experiments, the same procedure is performed with different hyperparameters. This still occurs even though only one combined model is useful, and as such one of them will never be used. While the model created by combining the Wav2Vec2 network as initial model, the XLNet network as target and the respective transformation will create a speech sentiment analysis model, the combination using the reversed models would take as input an English text and output the same text ideally, which does not represent in our view a valid task. The number of samples used for this training was 5549, the selected number of eigenvectors was 65 for both diffusion maps and both of their k_min parameters were set to 100. After computing the useful transformation we can now go on to the testing of the combined model.

The testing algorithm functions for this experiment as it does for the fourth, since there we already tackled all the problems posed by the task in in that experiment. The challenges faced in this case were, as in section 4.4, using transformations trained between 1 member subsets of the sequences to predicting the whole sequence of the target activation, sequence length changing between the input and the output sequence and predicting only a subset of features for each point in the sequence. This led to us using the same approaches as in section 4.4, by using the transformation on each sequence point independently to predict the same point in the target activation, truncating the length of the sequence of the target activation to the length of the output of that layer from the forward pass, and replacing empty positions in the feature dimension with the mean of the features at that sequence point.

nr. samples	**transformation**	**XLNet (positive)**	**transformation (positive)**
50 | 100	0.62	0.05	0.4
128 | 100	0.63	0.05	0.4
128 | 500	0.654	0.066	0.4

Table 4.5: Experiment 5 test results

After explaining how the computation of the forward pass is computed, we can now present and discuss the results of this experiment. Three runs of this model were performed with different trainings of the transformation and different samples both for training and for testing. The results of these runs can be seen in Table 4.5. The results of this experiment must

be interpreted while keeping in mind that the two experiments were done with different, randomly selected training and testing samples during each run and that, similar to the experiments in in section 4.4, the forward pass of the XLNet was not started using a dummy input of a proper shape but with the true text translation of the input of the Wav2Vec model.

The first column of Table 4.5 indicates the number of training samples followed by the number of testing samples used during the run that produced the results on the same row. The transformation row shows the accuracy of the model created from the XLNet model, the Wav2Vec model and a transformation in predicting the positive nature of the speech relative to the predictions of the XLNet on the equivalent text. The XLNet (positive) column shows how many samples from the test set the XLNet labeled as having a positive meaning, while the transformation (positive) column shows how many samples from the test set our model labeled as positive. While the results seem unfavorable at first glance, if we look at the distribution of the data as detemined by the XLNet model compared to the rate of model identified positives, we can determine that our model has a set of rules, and as such the transformation learns to map some of the activations correctly. We can also determine based on these values that one aspect which might have worsened the results is the approach presented in section 4.2 taken for incomplete feature dimension transformations, where we decided to replace the empty positions with the mean, since the predicted distribution is skewed towards the middle. Another approach, either to estimate the missing features, or to reduce the computational burden posed by transformations of large feature dimensions so that the estimate will not be needed is a possible topic of future work.

5 Conclusion

In this thesis we were able to implement and experiment with the concepts presented in "Transformations between deep Neural Networks" by Tom Bertalan, Felix Dietrich and Ioannis G. Kevrekidis [1]. After describing our implementation and experiments extensively we will now proceed to discuss their results and the possible topics that might emerge from this thesis.

5.1 Discussion of the Results

During this thesis we showed that effective transformations can be created between neural networks that are trained to model different phenomena on different types of sequence data (vibration, speech, text). We also showed that adapting the transformations for sequence data of different lengths, both between the different samples and the different neural networks is doable and that this adaptation produces usable results. Using these transformations we then created new models by using two pretrained neural networks and a transformation between their activation spaces. We have tested the models created in each model and we have achieved optimistic results. While the results of the initial models were never reached, we have to note that little to no hyperparameter optimisation was performed for the experiments.

Another aspect which has to be noted as an achievement was the fact that all four experiments were ran using the same algorithms presented in chapter 3, which shows not only the versatility of the concept of transformations, but also their ability to be used for any two networks which meet the criteria, no matter their internal configuration. As such we have shown through application that the concepts presented in [1] are robust and can be used for networks of different sizes and complexity.

	Experiment 2	**Experiment3**	**Experiment 4**	**Experiment 5**
highest accuracy	0.8828	0.8969	0.966	0.654

Table 5.1: Best results achieved for experiments 2, 3, 4 and 5

Table 5.1 shows the best results achieved by the created models in each experiment. It excludes the first experiment both because accuracy was not used as a metric and because the models created there did not have pretrained networks as components, since we had to train them during the experiment. The results presented for the second experiment are those of the model created by performing transformations from the activations of the transformer model [6] to those of the LSTM model [5]. The number shown represents the accuracy on the dataset, and in that case the achieved accuracy was very close to that of the LSTM model.

The results onwards are comparative accuracy metrics to the model which was the target of the transformation. As such we have used for the third experiment the results of the model build using a transformation from the Speech2Text2 model [7] to the Wav2Vec2 [2] model and for the forth experiment the results of the model incorporating a transformation from the RoBERTa model [10] to the XLNet model [3]. For the last experiment we used the best result created by the model created from the Wav2Vec2 model as initial model and XLNet model as target model and the respective transformation.

5.2 Further Research

Based on both the difficulties faced during this thesis and the results achieved a number of further improvements and possible usages can be found.

Based on the results achieved, one of the first topics opened by our ability to create accurate transformations between the activations of different pairs of neural networks is the analysis of the neural network activation spaces, and implicitly also of the neural networks, using the transformations as a tool for that purpose. Additionally, the creation of large equivalency classes between widely used neural networks could prove a useful endeavor not only for the analysis of the networks themselves, but also for the purpose of creating new models based on the possible combinations created by the equivalency classes.

Other topics also arise from challenges faced during the implementation of the transformations. Such topics include developing a concept for solving the problem posed by different sequence lengths between the activations of the same sample propagated in two different networks and improving the approximation done for neurons which the transformation did not predict in the target network. Another possible path is the optimisation of the transformation generation and computation such that these approximations are not required. These two topics arose due to the comparably worse results that we achieved in the fifth experiment.

While our approach functioned for the neighborhood generation for text data in the embedding layer, problems with this approach have been pointed out in section 4.4, and improvements can be made to these concepts.

Finally, another possible topic of interest would be the usage of transformations followed by other layers for the purpose of establishing similarity measurements between different points on the same manifold, a task which is a current topic in domains such as natural language programming and object recognition.

List of Figures

List of Tables

Bibliography

[1] T. Bertalan, F. Dietrich, and I. G. Kevrekidis. *Transformations between deep neural networks*. 2021. arXiv: 2007.05646 [cs.LG].

[2] patrickvonplaten. *wav2vec2-librispeech-clean-100h-demo-dist*. Last Accessed: 2021-12-27. URL: https://huggingface.co/patrickvonplaten/wav2vec2-librispeech-clean-100h-demo-dist.

[3] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *CoRR* abs/1906.08237 (2019). arXiv: 1906.08237. URL: http://arxiv.org/abs/1906.08237.

[4] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR* abs/1910.03771 (2019). arXiv: 1910.03771. URL: http://arxiv.org/abs/1910.03771.

[5] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[7] C. Wang, A. Wu, J. Pino, A. Baevski, M. Auli, and A. Conneau. *Large-Scale Self- and Semi-Supervised Learning for Speech Translation*. 2021. arXiv: 2104.06678 [cs.CL].

[8] A. Baevski, H. Zhou, A. Mohamed, and M. Auli. *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. arXiv: 2006.11477 [cs.CL].

[9] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. "Librispeech: An ASR corpus based on public domain audio books". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[11] H. Whitney. "Differentiable Manifolds". In: *Annals of Mathematics* 37.3 (1936), pp. 645–680. ISSN: 0003486X. URL: http://www.jstor.org/stable/1968482.

[12] L. W. Tu. "Manifolds". In: *An Introduction to Manifolds*. New York, NY: Springer New York, 2011, pp. 47–83. ISBN: 978-1-4419-7400-6. DOI: 10.1007/978-1-4419-7400-6_3. URL: https://doi.org/10.1007/978-1-4419-7400-6_3.

[13] J. M. Lee. *Introduction to Riemannian Manifolds* -. Cham, Heidelberg, New York, Dordrecht, London: Springer International Publishing, 2019. ISBN: 978-3-319-91754-2.

[14] P. C. Mahalanobis. "On the generalized distance in statistics". In: *Proceedings of the National Institute of Sciences (Calcutta)* 2 (1936), pp. 49–55.

[15] R. R. Coifman and S. Lafon. "Diffusion maps". In: *Applied and Computational Harmonic Analysis* 21.1 (July 2006), pp. 5–30. DOI: 10.1016/j.acha.2006.04.006. URL: https://doi.org/10.1016/j.acha.2006.04.006.

[16] V.-C. Stroescu. *Deep-learning based approaches for fault detection in a rotary mower*.

[17] textattack. *xlnet-base-cased-SST-2/tree/main*. https://huggingface.co/textattack/xlnet-base-cased-SST-2/tree/main. Last Accessed: 2021-12-29.

[18] M. Heitmann, C. Siebert, J. Hartmann, and C. Schamp. "More than a feeling: Benchmarks for sentiment analysis accuracy". In: *Available at SSRN 3489963* (2020).

[19] D. Lehmberg, F. Dietrich, G. Köster, and H.-J. Bungartz. "datafold: data-driven models for point clouds and time series on manifolds". In: *Journal of Open Source Software* 5.51 (2020), p. 2283. DOI: 10.21105/joss.02283. URL: https://doi.org/10.21105/joss.02283.

[20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[21] E. Grafarend. *Linear and Nonlinear Models: Fixed Effects, Random Effects, and Mixed Models*. Jan. 2006, p. 553.

[22] W. Falcon et al. "PyTorch Lightning". In: *GitHub* 3 (2019). URL: https://github.com/PyTorchLightning/pytorch-lightning.

[23] K. (https://stackoverflow.com/users/9005785/kees). *PyTorch get all layers of model*. Last Accessed: 2021-12-07. URL: https://stackoverflow.com/a/65112132.

[24] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[25] R. R. Coifman and S. Lafon. "Geometric harmonics: A novel tool for multiscale out-of-sample extension of empirical functions". In: *Applied and Computational Harmonic Analysis* 21.1 (2006). Special Issue: Diffusion Maps and Wavelets, pp. 31–52. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2005.07.005. URL: https://www.sciencedirect.com/science/article/pii/S1063520306000522.

[26] G. E. Fasshauer. *Meshfree Approximation Methods with Matlab*. WORLD SCIENTIFIC, Apr. 2007. DOI: 10.1142/6437. URL: https://doi.org/10.1142/6437.

[27] P. H. Schönemann. "A generalized solution of the orthogonal procrustes problem". In: *Psychometrika* 31.1 (Mar. 1966), pp. 1–10. DOI: 10.1007/bf02289451. URL: https://doi.org/10.1007/bf02289451.

[28] "Mean Squared Error". In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, 2010, pp. 653–653. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_528. URL: https://doi.org/10.1007/978-0-387-30164-8_528.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[30] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[31] E. J. Nyström. "Über Die Praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben". In: *Acta Mathematica* 54.none (1930), pp. 185–204. DOI: 10.1007/BF02547521. URL: https://doi.org/10.1007/BF02547521.