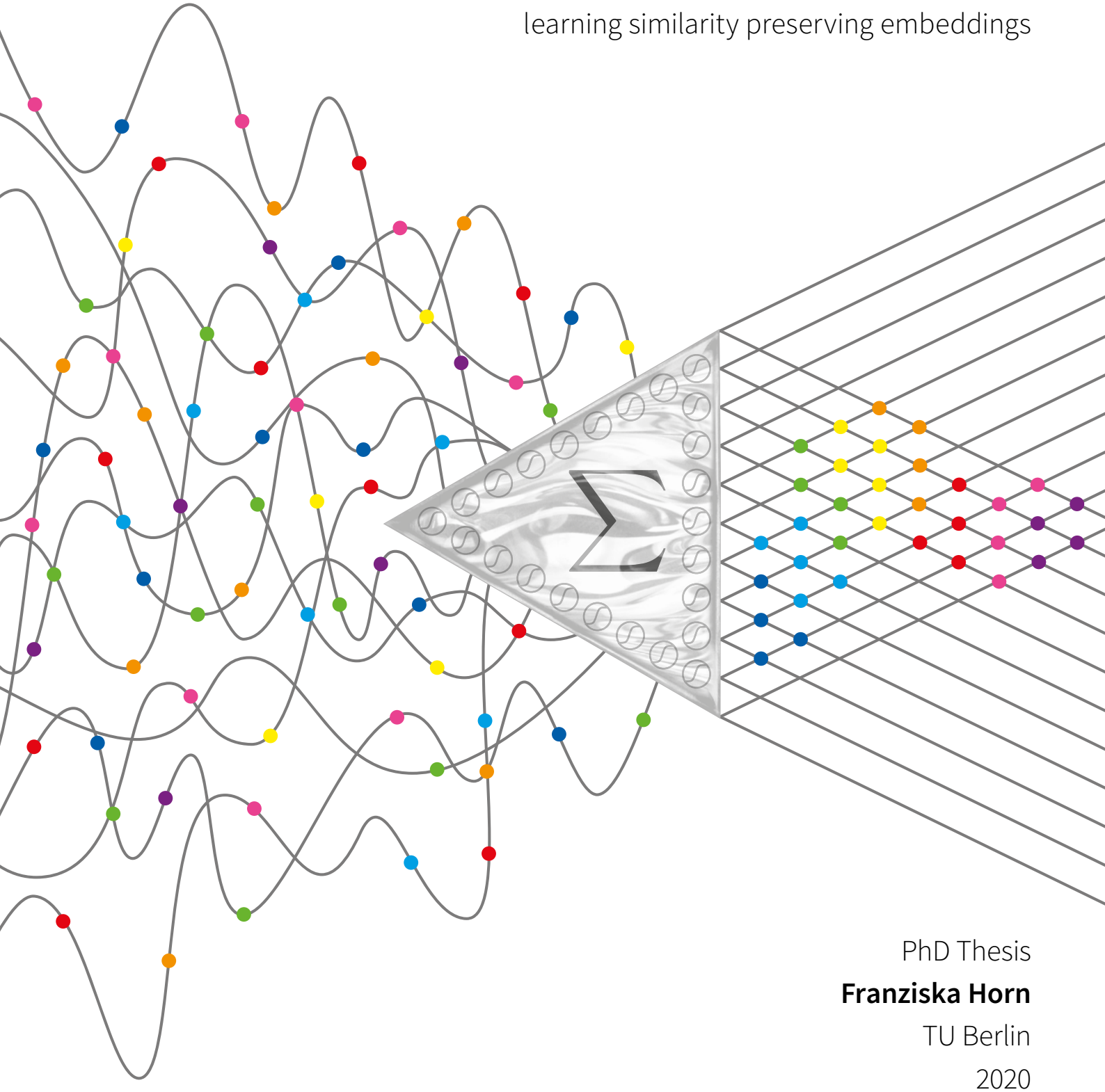
SIMILARITY ENCODER

a neural network architecture for
learning similarity preserving embeddings



PhD Thesis

Franziska Horn

TU Berlin

2020

Similarity Encoder

A Neural Network Architecture for Learning Similarity Preserving Embeddings

vorgelegt von
Master of Science
Franziska Horn

an der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktorin der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Benjamin Blankertz
Gutachter:	Prof. Dr. Klaus-Robert Müller
Gutachter:	Prof. Dr. Alan Akbik
Gutachter:	Prof. Dr. Ziawasch Abedjan

Tag der wissenschaftlichen Aussprache: 30. April 2020

Berlin 2020

Für Opa Walter.

Kein Titel, sondern die unendliche Neugierde
zeichnen den wahren Forscher aus.

Abstract

Matrix factorization is at the heart of many machine learning algorithms, for example, for dimensionality reduction (e.g. kernel PCA) or recommender systems relying on collaborative filtering. Understanding a singular value decomposition (SVD) of a matrix as a neural network optimization problem enables us to decompose large matrices efficiently while dealing naturally with missing values in the given matrix. But most importantly, it allows us to learn the connection between data points' feature vectors and the matrix containing information about their pairwise relations. In this thesis, we introduce a novel neural network architecture termed *Similarity Encoder* (SimEc), which is designed to simultaneously factorize a given target matrix while also learning the mapping to project the data points' feature vectors into a similarity preserving embedding space. This makes it possible to, for example, easily compute out-of-sample solutions for new data points. Additionally, we demonstrate that SimEcs can preserve non-metric similarities and even predict multiple pairwise relations between data points at once. As the first part of the SimEc architecture, mapping from the original (high dimensional) feature space to the (low dimensional) embedding, can be realized by any kind of (deep) neural network, SimEcs can be used in a variety of application areas. As we will demonstrate, SimEcs can serve as a reliable baseline model in pairwise relation prediction tasks such as link prediction or for recommender systems. The pairwise relations and similarities predicted by a SimEc model can also be explained using layer-wise relevance propagation (LRP). Furthermore, SimEcs can be used to pre-train a neural network used in a supervised learning task, which, for example, improves the prediction of molecular properties when only few labeled training samples are available. Finally, a variant of SimEc, called Context Encoder (ConEc), provides an intuitive interpretation of the training procedure of the CBOW word2vec natural language model trained with negative sampling and makes it possible to learn more expressive embeddings for words with multiple meanings as well as to compute embeddings for out-of-vocabulary words.

Zusammenfassung

Die Matrixfaktorisierung ist das Herzstück vieler maschineller Lernalgorithmen, beispielsweise der Dimensionalitätsreduktion (z.B. Kernel-PCA) oder Empfehlungssystemen, die auf kollaborativem Filtern beruhen. Das Verständnis einer Singulärwertzerlegung (SVD) einer Matrix als ein neuronales Netzwerkoptimierungsproblem ermöglicht es uns, große Matrizen effizient zu zerlegen und dabei problemlos mit fehlenden Werten in der gegebenen Matrix umzugehen. Aber vor allem erlaubt es uns, eine Verbindung zwischen den Merkmalsvektoren der Datenpunkte und der Matrix, die Informationen über ihre paarweisen Beziehungen enthält, zu lernen. In dieser Arbeit stellen wir eine neuartige neuronale Netzwerkarchitektur vor, Similarity Encoder (SimEc), welche gleichzeitig eine gegebene Zielmatrix faktorisiert, und das Mapping zur Projektion der Merkmalsvektoren der Datenpunkte in einen ähnlichkeitserhaltenden Einbettungsraum lernt. So können beispielsweise Out-of-Sample-Lösungen für neue Datenpunkte einfach berechnet werden. Außerdem demonstrieren wir, dass SimEcs nicht-metrische Ähnlichkeiten beibehalten und sogar mehrere paarweise Beziehungen zwischen Datenpunkten gleichzeitig vorhersagen kann. Da der erste Teil der SimEc-Architektur, welcher die Abbildung vom ursprünglichen (hochdimensionalen) Merkmalsraum zur (niederdimensionalen) Einbettung darstellt, durch beliebige (tiefe) neuronale Netze realisiert werden kann, können SimEcs in einer Vielzahl von Anwendungsbereichen eingesetzt werden. Wie wir zeigen, können SimEcs als zuverlässiges Basismodell für die Vorhersage paarweiser Beziehungen, wie z.B. Link Prediction oder für Empfehlungssysteme, dienen. Die paarweisen Beziehungen und Ähnlichkeiten, die von einem SimEc-Modell vorhergesagt werden, können auch unter Verwendung von Layer-wise Relevance Propagation (LRP) erklärt werden. Darüber hinaus können SimEcs dazu verwendet werden, um ein neuronales Netzwerk, welches in einer überwachten Lernaufgabe verwendet werden soll, vorzutrainieren, was zum Beispiel die Vorhersage von molekularen Eigenschaften verbessert, wenn nur wenig annotierte Trainingsbeispiele verfügbar sind. Darüber hinaus bietet eine Variante von SimEc, genannt Context Encoder (ConEc), eine intuitive Interpretation des Trainingsablaufs des mit negativem Sampling trainierten CBOW word2vec-Sprachmodells und ermöglicht es, expressivere Einbettungen für Wörter mit mehreren Bedeutungen zu lernen sowie Einbettungen für Wörter außerhalb des Vokabulars zu berechnen.

PREFACE

0.1 Main contributions

This thesis introduces **Similarity Encoder (SimEc)**, a neural network architecture that learns similarity preserving embeddings for data points by simultaneously *factorizing a matrix containing pairwise relations* between the data points while *learning a mapping from the original (high dimensional) feature space to the (low dimensional) embedding space.*

Compared to previous approaches, especially SVD-based methods, the SimEc model has several advantages:

- **SimEcs can efficiently factorize large matrices and easily handle matrices with missing values.** Computing an exact SVD is computationally prohibitively expensive for large matrices and impossible for matrices containing missing values. This would require the use of iterative methods [96] and special weighted error functions [88]. By understanding matrix factorization as a neural network optimization problem [40, 41], SimEcs naturally solve these issues.

- **SimEcs can compute out-of-sample solutions for arbitrary pairwise relations.** Spectral methods, such as kernel PCA, need the similarities between new test points and the original training examples to compute the embeddings for the new points. However, obtaining these pairwise relations for new data points is not always possible, especially when the pairwise relations represent human ratings. Alternatively, the embeddings for new test points, given the factorization of a matrix, could be created by training an additional regression model to learn the mapping from the original feature space to the embedding space. However, this would decrease the embedding quality compared to learning the factorization and mapping simultaneously [33], as it is being done by SimEcs.

- **SimEcs can learn similarity preserving embeddings for non-metric similarities.** Eigenvectors associated with strong negative eigenvalues of a similarity matrix can reveal interesting features in the data [106]. However, spectral methods like kernel PCA only include the embedding directions corresponding to the *largest positive* eigenvalues. SimEcs, on the other hand, preserve the information associated with the eigenvalues with the *largest absolute* values, and are therefore able to capture the features present in the negative part of the spectrum.

- **SimEcs can predict and learn a mapping for multiple pairwise relations at the same time.** Unlike SimEcs, traditional embedding methods are designed to only preserve a single pairwise relation.

- **SimEc predictions and similarities can be explained.** Techniques such as layer-wise relevance propagation (LRP), commonly used to explain predictions of neural network models, can also be applied to SimEcs. By revealing which input features contributed most to a certain pairwise relation prediction, LRP can transform the SimEc output into insights.

As the first part of the SimEc network (mapping from the inputs to the low dimensional embedding) can be chosen freely to fit the task at hand, **SimEcs can adapt to any application area**. In this thesis, I demonstrate the utility of SimEcs for a wide variety of use cases:

- SimEcs can be used to **reduce the dimensionality** of any kind of input data, producing embeddings comparable to those created, e.g., by kPCA or Isomap.
- SimEcs can serve as a reliable baseline model for pairwise relation prediction tasks such as **link prediction** or **recommender systems**.
- SimEcs can be used to **pre-train neural networks used in supervised tasks**, which can improve their performance if there is little labeled training data available for the prediction task. For example, a SimEc setup can be used to pre-train a convolutional neural network (CNN) for an **image classification** task, or a SchNet model [176, 177, 178] to improve the **prediction of molecular properties**.
- In the area of **natural language processing** (NLP), a variant of SimEc, called Context Encoder (ConEc), extends the CBOW word2vec model trained with negative sampling [129, 130], and provides an intuitive interpretation of its training procedure. ConEcs **improve word2vec word embeddings** by creating **more expressive embeddings for words with multiple meanings** and making it possible to compute **out-of-vocabulary embeddings**.

I implemented the Similarity Encoder model in Python using the keras [36] and PyTorch [148] libraries, which enable efficient training on GPUs. The model as well as many usage examples, including the experiments reported in this thesis, are provided online at:

`https://github.com/cod3licious/simec`

Code for the Context Encoder model and experiments can be found here:

`https://github.com/cod3licious/conec`

0.2 Related publications

This thesis is based in large parts on the following publications:

[Hor18] **Predicting Pairwise Relations with Neural Similarity Encoders**
Franziska Horn and Klaus-Robert Müller
Bulletin of the Polish Academy of Sciences: Tech. Sciences, 66(6):821-830, 2018.

[Hor17] **Context encoders as a simple but powerful extension of word2vec**
Franziska Horn
In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 10–14, 2017.

During the time of this research, I also (co-)authored the following peer-reviewed publications:

Forecasting Industrial Aging Processes with Machine Learning Methods
Mihail Bogojeski, Simeon Sauer, Franziska Horn, Klaus-Robert Müller
[Submitted], 2020.

The `autofeat` Python Library for Automated Feature Engineering and Selection
Franziska Horn, Robert Pack, Michael Rieger
In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2019*, pages 111-120. Springer, Cham, 2020.

Automating the search for a patent's prior art with a full text similarity search
Lea Helmers*, Franziska Horn*, Franziska Biegler, Tim Oppermann, Klaus-Robert Müller
PLoS ONE, 14(3):e0212103, 2019.

"What is Relevant in a Text Document?": An Interpretable Machine Learning Approach
Leila Arras, Franziska Horn, Gregoire Montavon, Klaus-Robert Müller, Wojciech Samek
PLoS ONE, 12(8):e0181142, 2017.

Explaining Predictions of Non-Linear Classifiers in NLP
Leila Arras, Franziska Horn, Gregoire Montavon, Klaus-Robert Müller, Wojciech Samek
In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 1–7, 2016.

And preprints:

The DALPHI annotation framework & how its pre-annotations can improve annotator efficiency
Robert Greinacher and Franziska Horn
arXiv preprint arXiv:1808.05558, 2018.

Discovering topics in text datasets by visualizing relevant words
Franziska Horn, Leila Arras, Gregoire Montavon, Klaus-Robert Müller, Wojciech Samek
arXiv preprint arXiv:1707.06100, 2017.

Exploring text datasets by visualizing relevant words
Franziska Horn, Leila Arras, Gregoire Montavon, Klaus-Robert Müller, Wojciech Samek
arXiv preprint arXiv:1707.05261, 2017.

Interactive Exploration and Discovery of Scientific Publications with PubVis
Franziska Horn
arXiv preprint arXiv:1706.08094, 2017.

0.3 Structure of this thesis

In **Chapter 1**, I give a motivation for learning similarity preserving embeddings and predicting pairwise relations with SimEcs, as well as discuss related work. While SVD-based methods yield an optimal factorization of a matrix, they are unable to create embeddings for new test points if their similarity to the original samples can not be computed. Metric learning approaches, on the other hand, find a mapping from the feature space to an embedding space where a target relation between the data points is preserved, but they are designed to learn from very sparse training data. SimEcs combine both ideas and can be trained efficiently to learn a mapping to factorize a dense matrix.

In **Chapter 2**, I explain how matrices can be factorized with neural networks and extend these models to arrive at the SimEc architecture, as presented in [Hor18]. Furthermore, I discuss why SimEcs can learn non-metric similarities as well as how to predict multiple pairwise relations at once.

In **Chapter 3**, I provide examples of how SimEcs learn similarity preserving embeddings, explain the role of different hyperparameters, and show that SimEcs can easily handle noisy input data or missing target values. With the right network architecture and target similarity matrix, SimEcs can easily learn the same solution as "global" methods, such as PCA, as well as "local" methods, such as Isomap, and are capable of creating embeddings for new test points based on arbitrary target similarities. Sections 3.2, 3.3, 3.5, and 3.6 are based on [Hor18].

In **Chapter 4**, I show how SimEcs can be used to predict pairwise relations in practical tasks like link prediction or for recommender systems. While SimEcs do not outperform state-of-the-art models specifically designed for these tasks, they can act as a reliable baseline approach. SimEcs show particular promise for content based recommendations, where they are able to transform the original feature vectors in such a way that the embeddings capture user ratings based similarities, which can be helpful to solve the cold start problem, i.e., to create recommendations for items that did not receive any user ratings so far.

In **Chapter 5**, I give examples for how SimEc predictions can be explained using layer-wise relevance propagation (LRP). This, for example, sheds light on why a certain movie is recommended for a particular user.

In **Chapter 6**, I demonstrate how SimEcs can be used to pre-train different neural network architectures used in supervised learning tasks to improve the performance in an image classification task as well as for the prediction of chemical properties.

In **Chapter 7**, I explain how a variant of SimEc, called Context Encoder (ConEc), implements the same training procedure as the CBOW word2vec model trained with negative sampling. These insights are then used to generate more expressive word embeddings, especially for out-of-vocabulary words and words with multiple meanings, which leads to an improved performance in a named entity recognition task. This chapter is adapted from [Hor17].

Chapter 8 concludes this thesis with a summary of the results.

Acknowledgments

Towards the end of my master's program, I had a job interview with a big consulting firm. One of the questions they asked me was:

"What was your greatest challenge?"

For the life of me, I could not give them an answer. Not that I hadn't accomplished anything so far – but none of it had felt particularly challenging.

Around the same time, Prof. Klaus-Robert Müller, who was supervising my master's thesis at the time, approached me and tried to convince me to stay at his lab for a PhD. And so I opted for a few more years of the comfortable student life. What followed was *a lot* of failures and frustration, which, I have to admit, took some getting used to. But the thing is, when you never really have to work hard for your achievements, you also never feel like you can be truly proud of them. So over the last few years, I slowly learned to appreciate this path paved with rejection, as it eventually lead me to the occasional breakthrough moments that felt like actual accomplishments.

I want to thank you, Klaus, for encouraging me to embark on this journey and helping me stay on course when I would have rather turned around. I know I probably wasn't your easiest student, but you had the patience and gave me the freedom to find my own way and grow with a true challenge.

In the altogether almost 8 years I spent in the Machine Learning (and Neurotechnology) group since writing my Bachelor's thesis there, I had the pleasure of getting to know and working with many wonderful people – Sven Dähne, Johannes Höhne, Benjamin Blankertz, Felix Bießmann, Dominik Kühne, Jan-Saputra Müller, Stephanie Brandl, Andreas Ziehe, Marina Höhne, and many many more: you made this a great place to work and learn!

I would also like to thank my friends and family for their continued support, especially: My parents, for always lending me a sympathetic ear, even when I was complaining about some rather technical problems. Christoph Hartmann and Antje Relitz, for proofreading pretty much every word I have written for this PhD. Ivana Balažević, Mario Stolz, and Ramona Höfer, for the best times in Berlin. Tina Schmidt, for some much needed (wine-heavy) distractions. Marcel Lengert: because you're you. And Stefanie Pieniadz, for always believing in me, even at times I didn't.

Thank you all, I couldn't have done it without you!

Special thanks goes out to Kristof Schütt and Michael Gastegger for the helpful discussions about SchNet (Sec. 6.2) and to my mom for helping me with graphics stuff, especially the title page. This work was partially funded by the Elsa-Neumann scholarship from the universities of Berlin.

Table of Contents

1 Introduction

Pairwise relations, such as similarities, between data points play an important role in many areas of machine learning (ML) [27, 74, 81, 174]. Most commonly used dimensionality reduction methods like t-SNE [119], kernel PCA (kPCA) [173], Isomap [189], and locally linear embedding (LLE) [165] create low dimensional representations of data points by preserving their pairwise similarities, distances, or local neighborhoods in the low dimensional embedding space, e.g., to create informative visualizations of a dataset [83]. Similarity preserving embeddings of data points can also serve as useful feature representations for other (supervised) ML tasks. For example, by computing the eigendecomposition of a kernel (i.e. similarity) matrix, kPCA projects the data into a feature space where data points can become linearly separable and noise in the data can be reduced [128, 135, 172]. In natural language processing (NLP) settings, the popular word2vec model [129, 130] learns an embedding for each word in the vocabulary by relying on the principle that similar words appear in similar contexts [73, 82, 113]. Using word embeddings as features can improve the performance in many NLP tasks such as named entity recognition or text classification [42, 107, 196]. The prediction of pairwise relations themselves is at the heart of important real world ML applications such as the prediction of whether or not a drug could interact with a certain protein [66] or for recommender systems, where the task is to predict the rating a user would give to a certain item [96, 171] or to identify similar items that could be promoted alongside an item of interest [15]. Another active research area is concerned with the analysis of graphs, such as social networks, where the pairwise relations between nodes are of key importance [4, 71].

Pairwise relations between data points can be represented as a rectangular matrix $R \in \mathbb{R}^{m \times n}$, which could, for example, contain the ratings of m items by n users. In the following, we will primarily focus on pairwise similarities between m data points, stored in a square symmetric matrix $S \in \mathbb{R}^{m \times m}$, but also provide examples and discuss how our results generalize to arbitrary pairwise relations R.

In this thesis, we introduce our novel neural network (NN) architecture called *Similarity Encoder* (SimEc), which learns (low dimensional) *similarity preserving embeddings* of data points. To be more precise, a SimEc learns a function $f'(\mathbf{x}_i) = \mathbf{y}_i$ to map a (high dimensional) feature vector $\mathbf{x}_i \in \mathbb{R}^D$ to an embedding vector $\mathbf{y}_i \in \mathbb{R}^d$ such that the scalar product of two embedding vectors approximates some given similarity measure, i.e., $\langle \mathbf{y}_i, \mathbf{y}_j \rangle \approx S_{ij}$ (or in matrix notation: $YY^\top \approx S$). To accomplish this, the full SimEc architecture consists of an arbitrary neural network f', which maps the input matrix $X \in \mathbb{R}^{m \times D}$ to the corresponding embeddings $Y \in \mathbb{R}^{m \times d}$, and an additional linear last layer $W_l \in \mathbb{R}^{d \times m}$, which is multiplied with Y to compute the output of the SimEc, $\hat{S} \in \mathbb{R}^{m \times m}$. Taken together, a SimEc network

computes

$$f(X) = f'(X)W_l = YW_l = \hat{S}$$

and is trained with the objective

$$\min \|S - f'(X)W_l\|_F^2 + \lambda \|S - W_l^\top W_l\|_F^2,$$

where λ is a hyperparameter. By minimizing both $\|S - f'(X)W_l\|_F^2$ and $\|S - W_l^\top W_l\|_F^2$, it is ensured that the factorization of S as $f'(X)W_l$ is symmetric and therefore that $f'(X)f'(X)^\top = YY^\top \approx S$.[1]

A SimEc therefore learns embeddings that factorize some similarity matrix S, similar to the embeddings computed by kPCA via an eigendecomposition of a kernel matrix. However, to compute the embeddings for new test points (i.e. out-of-sample (OOS) solutions) with kPCA, first the kernel map, i.e., the similarities to the training examples for these new data points, needs to be computed. This, however, is not possible if the given target similarities were not computed directly from the feature vectors, i.e., if they were obtained by human similarity ratings or some other unknown process. A SimEc, on the other hand, provides the linear or non-linear mapping function f', with which new data points can be mapped into the similarity preserving embedding space directly (Fig. 1.1). Furthermore, SimEcs can deal with missing values in the similarity matrix S, can embed data points based on metric or non-metric similarities, and can be used to predict multiple pairwise similarities or other relations between data points at once.

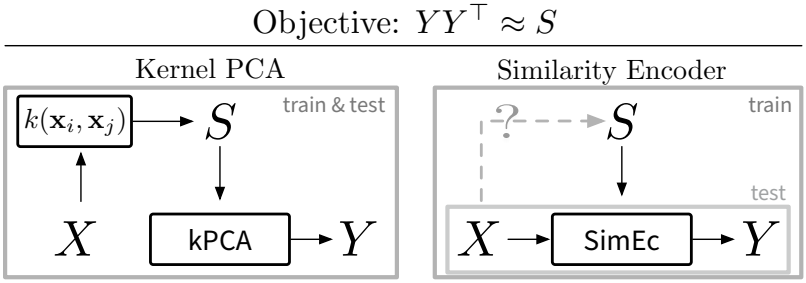


Figure 1.1: Kernel PCA and SimEc both aim to project the data points into an embedding space where the target similarities can be approximated by the scalar product of the embedding vectors, but kernel PCA needs to compute a kernel map, i.e., the similarities to the training data points, to be able to embed new test samples.

In the remainder of this chapter, we present a motivating example to illustrate the shortcomings of spectral (i.e. SVD-based) embedding methods alleviated by SimEcs and discuss other related work. The SimEc architecture is then described in more detail in Chapter 2, including ways to train the model efficiently on large datasets and how to predict arbitrary pairwise relations $R \in \mathbb{R}^{m \times n}$. In Chapter 3, we demonstrate that SimEcs can compute comparable similarity preserving embeddings as spectral methods (e.g. kPCA), while additionally being able to learn a mapping into the embedding space for target similarities of unknown origin or containing missing values, as well as to predict non-metric similarities and multiple similarities at once. In Chapter 4, we explore the possibilities of using SimEcs for practical pairwise relation prediction tasks such as link prediction and recommender systems. In Chapter 5, we explain the pairwise relation predictions made by

[1] For greater clarity, these equations are written in matrix notation, while of course, like other NNs, SimEcs are trained efficiently using a mini-batch stochastic gradient descent based optimization procedure.

SimEcs using layer-wise relevance propagation. In Chapter 6, we demonstrate how SimEcs can be used to pre-train different neural network architectures and thereby improve their generalization performance in supervised learning tasks. In Chapter 7, we show how a variant of SimEc, called Context Encoder (ConEc), extends the word2vec language model to learn better word embeddings for words with multiple meanings as well as to create embeddings for out-of-vocabulary words. Chapter 8 then concludes this thesis with a summary of the results.

1.1 Motivation: *The flowerpot experiment*

The following simple example illustrates the benefits of SimEcs compared to existing spectral dimensionality reduction methods such as kPCA. In the "flowerpot experiment", Gati and Tversky [61] showed 30 subjects images of 16 flowerpots with plants differing in their size and leaf shapes and asked them to rate their similarities, resulting in a 16×16 similarity matrix (Fig. 1.2). For our experiments, we randomly selected two of the flowerpots (5 & 15) as test examples and pretend that they have not been rated, i.e., the corresponding two rows and columns were deleted from the similarity matrix. Additionally, we created a five-dimensional feature vector for each of the 16 flowerpots, where the first feature represents the stem size of the plant, the second feature the leaf shape, and the three other features are just random noise.[2]

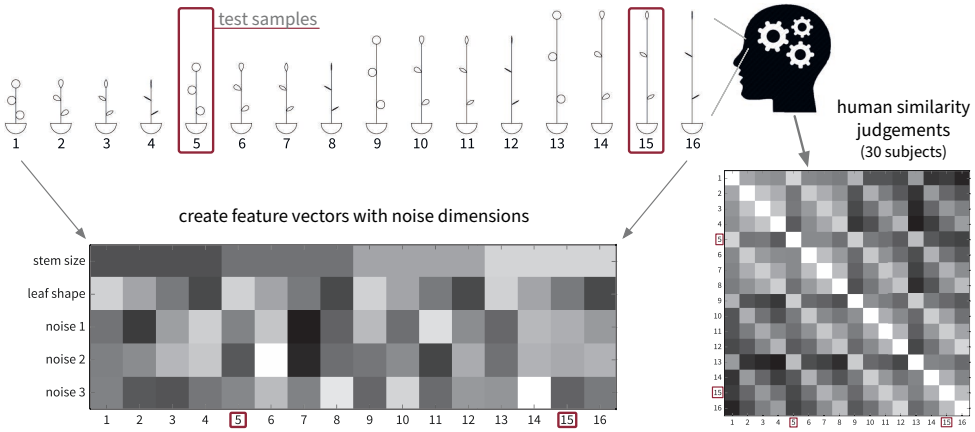


Figure 1.2: Flowerpot experiment [61].

Given the noisy feature vectors and the 14×14 similarity matrix, the task is now to find lower dimensional representations for *all 16* data points, such that in the lower dimensional space two flowerpots are represented close to each other if the humans had judged them as similar.

One possibility to get such lower dimensional embeddings would be to perform an eigendecomposition of the human similarity matrix (Fig. 1.3). In the resulting embedding space, the training points are aligned with respect to stem length and leaf shape of the plants, which corresponds to the human similarity judgments. However, as the similarity ratings for the two new test samples are not available, low dimensional representations for those new points can not be created.[3]

[2] If the subjects were originally given noisy images to compare, they would most likely ignore the noise and still base their similarity judgments only on the meaningful features.

[3] At least not without learning an additional regression model to get a mapping from the input features to the embedding space.

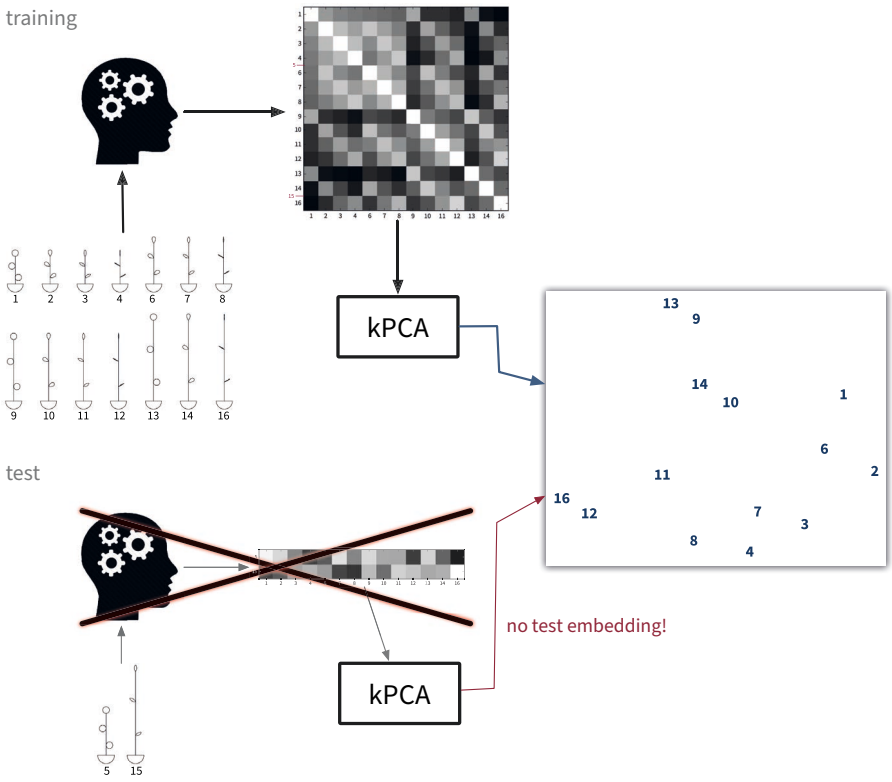


Figure 1.3: Flowerpot embeddings obtained for the training samples by decomposing the given matrix with human similarity ratings.

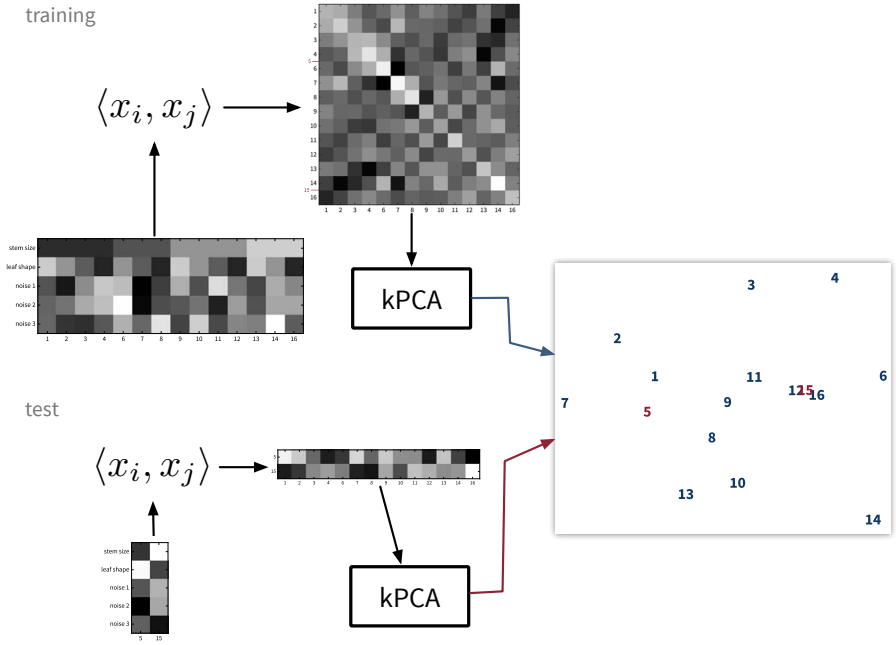


Figure 1.4: Flowerpot embeddings obtained using linear kernel PCA.

Another possibility would be to use the feature vectors that were created for each data point to compute a (in this example linear) kernel matrix and then compute the kPCA embeddings from this matrix (Fig. 1.4). However, since the feature vectors are noisy, the computed kernel matrix does not resemble the human ratings from the original similarity matrix and in the resulting embedding the data points are arranged randomly. Since the kernel map (i.e. the similarities to the training data points) can be computed using the feature vectors associated with the test points, it is possible to project the new data points into the embedding space as well. However, just as the embedded training points, the test points' embeddings fail to preserve the given human target similarities.

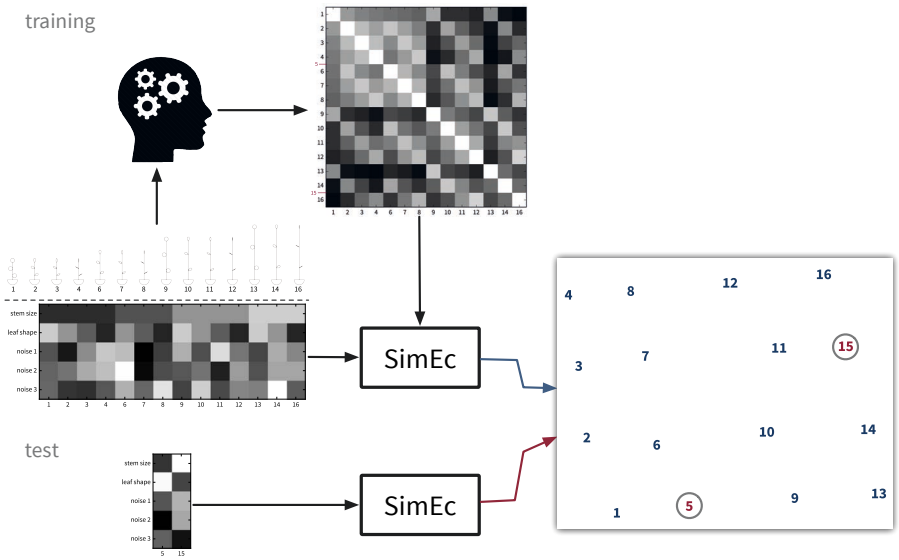


Figure 1.5: Flowerpot embeddings obtained using a linear SimEc.

Similarity Encoders, on the other hand, can easily handle the task posed in the flowerpot experiment (Fig. 1.5). During training, the SimEc network gets as input the feature vectors of the training points and as targets the human similarity matrix. It then learns to associate the relevant input feature dimensions with the target similarities and is able to create meaningful embeddings for the training samples. After training, the feature vectors of new test points are used as input to the trained SimEc network and it then projects them into the lower dimensional space as well. In this way, a SimEc is able to create low dimensional embeddings *for all data points* that preserve the given, *arbitrary target similarities*.

1.2 Related work

In this section, we discuss several other approaches to learning similarity preserving embeddings and predicting pairwise relations. Further related work specific to individual application areas is additionally summarized in the respective chapters on these topics. SimEcs are conceptually related to metric learning models, while implementing a matrix factorization (MF) objective. Both approaches, as well as other hybrid methods, are discussed in detail below, while Table 1.1 gives an overview of the respective learning problems related to SimEcs.

Matrix Factorization (MF) + Regression The optimal (in a least squares sense) low dimensional embeddings to factorize a matrix R or S can be found by computing a singular

Table 1.1: High-level summary of related work.

Approach	Learns...	by...
MF & Reg.	a) $YY^\top \approx S$ b) $f(X) \approx Y$	Eigendecomposition $\min \|Y - f(X)\|_F^2$
Metric L.	$f(\mathbf{x}_i, \mathbf{x}_j) \approx S_{ij}$ or $f(\mathbf{x}_i, \mathbf{x}_i^+) > f(\mathbf{x}_i, \mathbf{x}_i^-)$	$\min \sum (S_{ij} - f(\mathbf{x}_i, \mathbf{x}_j))^2$ $\min \sum \max\{0, 1 - f(\mathbf{x}_i, \mathbf{x}_i^+) + f(\mathbf{x}_i, \mathbf{x}_i^-)\}$
Hybrid	$f(X) = Y$	$\min \|S - YY^\top\|_F^2$
SimEc	$f(X) = f'(X)W_l \approx S$ $\Rightarrow f'(X) = Y \ : \ YY^\top \approx S$	$\min \|S - f'(X)W_l\|_F^2 + \lambda \|S - W_l^\top W_l\|_F^2$

value decomposition (SVD) or eigendecomposition of the matrix and using the d largest eigenvalues and corresponding eigenvectors to compute a low rank approximation of the matrix. However, performing an SVD is computationally very expensive for large matrices, and in these cases requires the use of approximate iterative methods [96]. Furthermore, an exact decomposition can not be computed for matrices that contain missing values, in which case weighted error functions need to be employed [88]. Back in 1982, a simple neural network (NN) was conceived to compute a PCA [144] and in 1992, NNs were proposed as a method to efficiently compute the SVD [40] or eigendecomposition [41] of a matrix while naturally dealing with missing values in the target matrix. As discussed in more detail in Section 2.1, SimEcs are an extension of these NN architectures and retain their benefits.
While the factorization of a matrix may provide suitable embedding vectors for the given training points, as illustrated with the flowerpot experiment, embedding new test points can be tricky. If the kernel map for the test points can be computed, OOS solutions are readily available [25]. For some support vector kernel functions [135, 174, 198], it is even possible to use a manually devised, kernel-specific random mapping from the original input to the kernel feature space to create similarity preserving embeddings for large datasets very efficiently [153]. By interpreting this mapping as a neural network, it can be further fine-tuned to the dataset at hand [5]. However, if instead target similarities of unknown origin are provided, e.g., based on human similarity judgments, it becomes necessary to train an additional regression model to learn the mapping from the original input feature space to the embeddings computed by the spectral method in order to compute OOS solutions [121, 188]. This not only requires an additional computational effort, but the best similarity preserving embeddings that can be realized as a transformation of the original feature vectors might not necessarily correspond to the embeddings found by the spectral method, thereby losing some information in the mapping step, resulting in unnecessarily poor similarity approximations [33].[4] SimEcs, on the other hand, learn the mapping function to create OOS solutions and the factorization of the similarity matrix together and therefore do not have these problems.

[4]For example, eigenvalue d of S might only be slightly larger than eigenvalue $d+1$, however, the dth eigenvector might contain information that is not present in the original feature vectors, while the information encoded in eigenvector $d+1$ can be preserved by a transformation of the feature vectors. By learning the mapping and factorization together, it is possible to create a d-dimensional embedding that instead retains the information from the $d+1$ eigenvalue and -vector, thereby resulting in an only slightly worse approximation of S compared to the spectral method, while not losing any accuracy in the mapping step.

Metric Learning Metric or similarity learning methods aim to learn a function $f(\mathbf{x}_i, \mathbf{x}_j)$ that approximates a given similarity or other pairwise relation between the two inputs $\mathbf{x}_i$ and $\mathbf{x}_j$ [20, 102, 206]. The main distinction between different metric learning approaches can be drawn based on how the training data is provided, which is also directly related to the objective function optimized by the models. In a regression setting, training samples are provided as triplets $(\mathbf{x}_i, \mathbf{x}_j, S_{ij})$, where S_{ij} corresponds to the target value that should be predicted for the input pair $(\mathbf{x}_i, \mathbf{x}_j)$. Instead of a similarity between two data points of the same type, this setup can also be used to predict a numerical pairwise relation between two different kinds of inputs, e.g., the star rating for a movie by a specific user. If exact numerical values can not be provided, the learning problem can instead be formulated in terms of a classification problem: here the training samples come as pairs of positive $(\mathbf{x}_i, \mathbf{x}_i^+)$ and negative $(\mathbf{x}_i, \mathbf{x}_i^-)$ examples, and $f(\mathbf{x}_i, \mathbf{x}_j)$ should predict larger values for the positive than the negative pairs [46, 57, 87, 143]. This setup is again common in recommender systems, where often only implicit feedback is available (e.g. a song a user has listened to compared to songs he did not listen to). Closely related to the classification setup is the triplet learning or ranking approach [11, 34, 175, 199], where similarity data is collected by asking "Is A more similar to B or to C?", resulting in the triplets $(\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)$, where again the predictions for $(\mathbf{x}_i, \mathbf{x}_i^+)$ should be larger than for $(\mathbf{x}_i, \mathbf{x}_i^-)$. This kind of similarity data is generally easier to collect from human subjects, who often have difficulties assigning consistent numerical similarity scores to different pairs of samples. What all three metric learning setups have in common is that the training data is usually very sparse, i.e., instead of a dense similarity matrix, a training set typically only contains the pairwise relations for comparatively few pairs of points. This often lies in the nature of the problem, e.g., if user data is collected.

Most early metric learning models were comparatively simple, e.g., a bilinear function in the form of $f(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i W \mathbf{x}_j^\top$ [34, 69, 202] or its kernelized version [124, 127]. Nowadays, the task is often solved using neural networks [143, 199], either in the form of a siamese network (i.e. two networks with shared parameters) to predict the similarity between two inputs of the same type [30, 70], or two different networks to account for structural differences in the input features (e.g. users and items) [205]. The pairwise relation can either be predicted directly by concatenating input vectors or fusing the two networks at some layer [76], or by using the networks to project both inputs into the same embedding space, where, for example, the predicted relation can then be computed as the cosine similarity between the two embedding vectors [90, 207]. Computing the similarity with a simple dot product speeds up the computation for larger sets of points since a NN only needs to be applied once to each input to map it to the corresponding similarity preserving embedding (like a preprocessing step) instead of predicting the pairwise relations with the network(s) directly from all pairs of inputs.

With a fast training procedure for target pairwise relation matrices where the number of non-zero elements is much smaller than $m \cdot n$ and the benefit of learning multiple mapping functions simultaneously for projecting different kinds of feature vectors into the same embedding space, these neural network based metric learning approaches are very useful for many applications scenarios. SimEcs, on the other hand, are designed to efficiently factorize dense matrices (while being able to handle missing values in the target matrix) and, while they rely on only a single neural network to map input features into a similarity preserving embedding space, we will discuss in the next chapter how they can be trained to *predict multiple pairwise relations at once* based on the same embedding.

Hybrid approaches The hybrid approaches discussed in the following are the models most closely related to SimEcs. Similar to the NN metric learning models, they consist of one or two neural networks that map some input features into an embedding space where the dot product or cosine similarity predicts the target relation, but they are trained to approximate (patches of) a dense pairwise relation or similarity matrix instead of sparse samples. While we only elaborate on work about single neural network models learning similarity preserving embeddings, in accordance with the SimEc architecture, two networks can be trained accordingly.
A single neural network is trained to map the points into a similarity preserving embedding space by computing the embeddings for a batch of training samples and then comparing the pairwise similarities (or distances) of these embedded points against the target similarities to compute the error used in the backpropagation procedure to tune the network's parameters. With this approach, extensions for t-SNE [118] and other classic manifold learning methods [31, 117] were developed, which enable the computation of OOS solutions. A particularly interesting realization of this approach are deep kernelized auto-encoders [93], which train an auto-encoder network with an additional objective to not only minimize the reconstruction error of the data points themselves, but also the mismatch between the dot product of a batch of embedding vectors and the corresponding block from a kernel matrix. The decoder part of the auto-encoder network thereby also provides a mapping from the embedding space back to the original feature space, which can be used to compute the pre-image of an embedding vector [128]. By directly minimizing $\|S - YY^\top\|$, these methods successfully learn similarity preserving embeddings that factorize the given target matrix. However, because they always operate on batches of points, these methods scale quadratically and efficient training is highly dependent on the choice of the batch size, requiring either lots of memory or many combinations of randomly chosen samples to cover all pairwise similarities. In an effort to improve on this, e.g., the method of auxiliary coordinates [33] can be used to train a NN in an alternating fashion, in one step optimizing the mapping from the input to the embedding space, in the other step improving the similarity approximation of the embedding itself [52, 216]. As we will see in Section 2.2, while the weight matrix of the last layer of the SimEc architecture could be interpreted as a set of auxiliary coordinates as well, training a SimEc network does not require alternating steps in the optimization procedure and the training time still scales linearly with the number of training samples.

2

The Similarity Encoder Model

In this chapter, we first describe how neural networks (NNs) can realize the computation of an SVD of a rectangular matrix $R \in \mathbb{R}^{m \times n}$ [40] and the eigendecomposition of a square symmetric matrix $S \in \mathbb{R}^{m \times m}$ [41]. Then we detail how these models can be extended to arrive at the Similarity Encoder (SimEc) neural network architecture [86].

2.1 Matrix factorization with neural networks

With singular value decomposition (SVD), a matrix $R \in \mathbb{R}^{m \times n}$ can be decomposed as

$$R = U \Sigma V^\top,$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ contain the eigenvectors of $RR^\top$ and $R^\top R$ respectively, while the corresponding eigenvalues are stored in $\Sigma \in \mathbb{R}^{m \times n}$. By using only the d largest eigenvalues and corresponding eigenvectors, a low rank approximation of R can be obtained, i.e., $R \approx U_{[:,:d]} \Sigma_{[:d,:d]} V^\top_{[:d,:]}$.

By setting $W_1 = U_{[:,:d]} \sqrt{\Sigma_{[:d,:d]}}$ and $W_2 = \sqrt{\Sigma_{[:d,:d]}} V^\top_{[:d,:]}$, the low rank approximation of R can be rewritten as

$$R \approx W_1 W_2 = I_m W_1 W_2,$$

where $I_m \in \mathbb{R}^{m \times m}$ is the identity matrix.

A simple feed forward neural network $f(\mathbf{x}_i)$ can now be constructed with two layers defined by the weight matrices $W_1 \in \mathbb{R}^{m \times d}$ and $W_2 \in \mathbb{R}^{d \times n}$ and without any non-linear activation functions. Given some input vector $\mathbf{x}_i \in \mathbb{R}^m$, the first layer computes

$$f'(\mathbf{x}_i) = \mathbf{x}_i W_1 = \mathbf{y}_i,$$

where we call $\mathbf{y}_i \in \mathbb{R}^d$ the embedding of the ith data point $\mathbf{x}_i$, with $i \in \{1, ..., m\}$. With both layers, the network computes

$$f(\mathbf{x}_i) = f'(\mathbf{x}_i) W_2 = (\mathbf{x}_i W_1) W_2 = \mathbf{y}_i W_2 = \hat{\mathbf{r}}_i, \tag{2.1}$$

the n-dimensional vector $\hat{\mathbf{r}}_i$. Expressed in matrix notation, given an input matrix $X \in \mathbb{R}^{m \times m}$ the network first computes an embedding matrix $Y \in \mathbb{R}^{m \times d}$ and from it the output $\hat{R} \in \mathbb{R}^{m \times n}$:

$$f(X) = f'(X) W_2 = (X W_1) W_2 = Y W_2 = \hat{R}.$$

If the network is trained (with backpropagation using a mini-batch stochastic gradient descent optimization procedure) to minimize the mean squared error of its output to a target matrix R, i.e.

$$\min \ \|R - f(X)\|_F^2 = \min \ \|R - XW_1W_2\|_F^2 ,$$

while using as input to the network the identity matrix, i.e., $X = I_m$, then once the weights of the network have converged to a local optimum, W_1W_2 is a low rank approximation of the matrix $R \in \mathbb{R}^{m \times n}$ [40].

When computing an SVD of a matrix, the eigenvectors stored in the matrices U and V are orthogonal (i.e. $V^\top V = I_n$), which can be added as a further constraint to the cost function:

$$\min \ \|R - I_mW_1W_2\|_F^2 + \lambda \left\| I_d \circ W_2W_2^\top - W_2W_2^\top \right\|_F^2 ,$$

where λ is a hyperparameter to control the strength of this regularization.[1]

Should R contain missing values, then the error used in the backpropagation procedure to tune the network's parameters is only computed with respect to the available entries of the matrix. In this case especially, it can be advantageous to additionally use other regularization techniques such as adding ℓ_2 regularization terms to the cost function.

As the decomposition of a square symmetric matrix $S \in \mathbb{R}^{m \times m}$ into its eigenvalues and -vectors is a special case of an SVD (where $V = U$), the same NN can be used, only with an additional constraint to learn a symmetric factorization, i.e., to encourage $I_mW_1 = Y = W_2^\top$. This can be achieved with the cost function

$$\min \ \|S - I_mW_1W_2\|_F^2 + \lambda \left\| S - W_2^\top W_2 \right\|_F^2 ,$$

where, after convergence, $YW_2 \approx W_2^\top W_2 \approx YY^\top \approx S$. This results in an embedding $Y \in \mathbb{R}^{m \times d}$ similar to the eigenvector based embedding found by kernel PCA.

2.2 Similarity Encoders

Now that the factorization of a matrix R or S is expressed in terms of optimizing a neural network, this setup can be further extended to yield our SimEc architecture. In particular, the first linear layer of the neural network, $f'(\mathbf{x}_i) = \mathbf{x}_iW_1 = \mathbf{y}_i$, can be replaced by any kind of (deep) neural network to map arbitrary feature vectors $\mathbf{x}_i \in \mathbb{R}^D \ \forall i \in \{1, ..., m\}$ into the low dimensional embedding space (Fig. 2.1). Equation (2.1) then becomes

$$f(\mathbf{x}_i) = f'(\mathbf{x}_i)W_l = \mathbf{y}_iW_l = \hat{\mathbf{r}}_i,$$

where again $\mathbf{y}_i \in \mathbb{R}^d$ is the embedding of the ith data point $\mathbf{x}_i$ and $W_l \in \mathbb{R}^{d \times n}$ is the weight matrix of the last (linear) layer of the full network $f(\mathbf{x}_i)$, while $f'(\mathbf{x}_i)$ could, for example, be a convolutional neural network (CNN) mapping images into the embedding space. This Similarity Encoder network is again trained to minimize $\|R - f(X)\|_F^2$, thereby learning the factorization $R \approx f'(X)W_l = YW_l$.

[1] While this will encourage orthogonal rows in W_2, since the rows do not need to have unit length, the values on the diagonal of $W_2W_2^\top$ should not be penalized. This kind of regularization is usually only necessary if d is chosen to be greater than the number of significant eigenvalues. Please note that the rows of W_2 are not necessarily ordered by the magnitude of the corresponding eigenvalues.

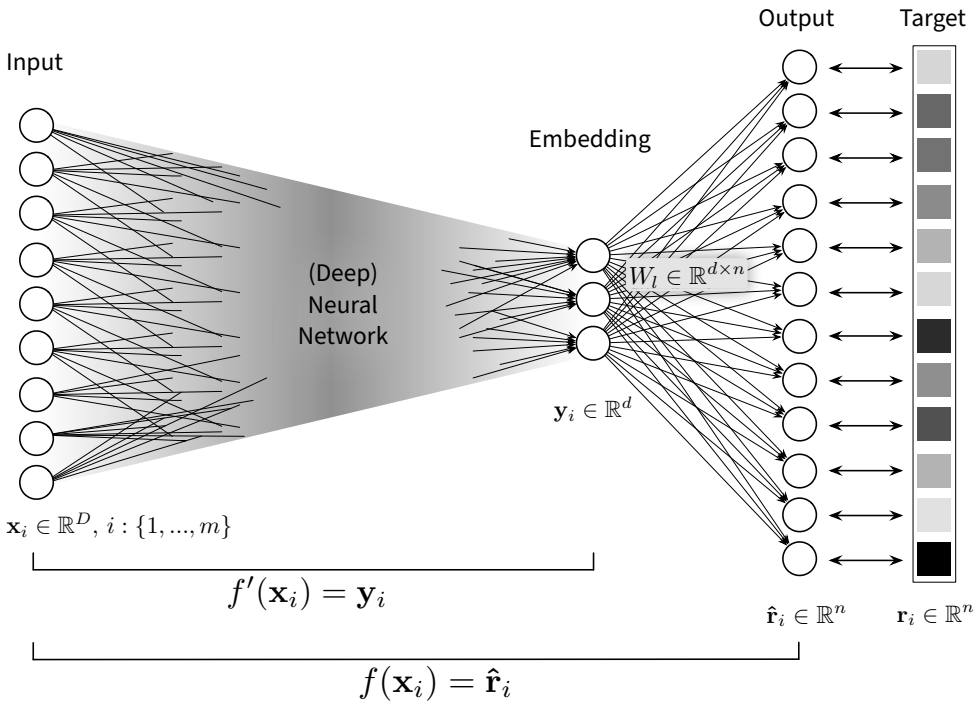


Figure 2.1: Similarity Encoder (SimEc) architecture. A (deep) neural network, $f'(\mathbf{x}_i)$, is used to map an original feature vector $\mathbf{x}_i \in \mathbb{R}^D$ to its embedding $\mathbf{y}_i \in \mathbb{R}^d$. This embedding is then multiplied by another weight matrix $W_l \in \mathbb{R}^{d \times n}$, which corresponds to the last layer of the full SimEc network $f(\mathbf{x}_i)$, to compute $\hat{\mathbf{r}}_i \in \mathbb{R}^n$, i.e., the approximation of one row of the target matrix $R \in \mathbb{R}^{m \times n}$. After the SimEc is trained to minimize $\|R - f(X)\|_F^2$ given the full feature matrix $X \in \mathbb{R}^{m \times D}$, it then computes a rank d approximation of R as $f(X) = f'(X)W_l = YW_l$.

If the output of the SimEc should always be in a specific range, e.g., if the target matrix R contains star ratings from 1 to 5,[2] it may be beneficial to add an additional non-linearity after computing $f'(\mathbf{x}_i)W_l$ to ensure the predicted values are within this range. However, there should never be any non-linearity at the last layer of $f'(\mathbf{x}_i)$ as the embedding values $\mathbf{y}_i$ should be able to assume unconstrained values.

Regularization terms can again be added to the cost function as discussed before. However, it should be noted that the constraint to encourage a symmetric factorization of a similarity matrix S, i.e., the regularization term $\left\|S - W_l^\top W_l\right\|_F^2$, can significantly increase the computational complexity of the optimization procedure, as computing $W_l^\top W_l$ scales with m^2. However, in practice it is often enough to only train with a subsample of S using $n \ll m$ targets, i.e., optimizing

$$\min \ \left\|S_{[:,:n]} - f'(X)W_l\right\|_F^2 + \lambda \left\|S_{[:n,:n]} - W_l^\top W_l\right\|_F^2,$$

with $W_l \in \mathbb{R}^{d \times n}$ and $f(X) = \hat{S} \in \mathbb{R}^{m \times n}$, which greatly reduces the overall complexity and memory requirements of the training procedure. Even though the number of targets in the output is reduced, all m training examples can still be used as input during training.

[2] In practice, the training of a SimEc usually converges fastest when the target matrix is first max-normalized (values between 0 and 1) or standardized (mean 0 & standard deviation 1).

When using only a small number of targets n, $S_{[:n,:n]}$ can also be decomposed into its eigenvalues and -vectors to compute the optimal weight matrix W_l (i.e., such that $W_l^\top W_l \approx S_{[:n,:n]}$). By initializing the SimEc's last layer with these values, the network often converges faster during training.

Instead of limiting the number of targets, it might also be worth considering whether it is necessary to enforce a symmetric factorization of S (as $YW_l \approx W_l^\top W_l \approx YY^\top$) at all. If the SimEc only needs to predict the similarities between a new sample and the existing samples or even just between the existing samples themselves, e.g., to fill missing values in S, then the regularization term can in practice be ignored. The similarities between a new sample $\mathbf{x}_j$ and the training samples can then be computed as

$$f(\mathbf{x}_j) = f'(\mathbf{x}_j)W_l = \mathbf{y}_j W_l = \hat{\mathbf{s}}_j \in \mathbb{R}^m,$$

instead of $f'(\mathbf{x}_j)f'(X)^\top = \mathbf{y}_j Y^\top$.

A similar choice should be made when factorizing a rectangular matrix R. By default a SimEc only learns the mapping from one input feature space to the embedding space and then predicts the values of R by multiplying this embedding with W_l. This is sufficient in many cases. For example, for an established social network site, thousands of pieces of new content are uploaded every second and at the same time older content becomes irrelevant, while the user base remains fairly constant. In such a scenario it might be sufficient to simply predict which users might be interested in a new piece of content, which can be done by using the full SimEc network to predict $f(\mathbf{x}_i) = \hat{\mathbf{r}}_i$ for some content feature vector $\mathbf{x}_i \in \mathbb{R}^D$. Nevertheless, it is also possible to train a second SimEc network to additionally project the set of n users into the same embedding space as some m items, thereby making it possible to predict ratings for both new items *and* new users as the scalar product of their embedding vectors. For this, a SimEc network f_1 is first trained on one set of feature vectors $X_1 \in \mathbb{R}^{m \times D}$ to approximate R (or a subset of it). After the training is complete, these feature vectors are projected into the embedding space to yield $f_1'(X_1) = Y_1 \in \mathbb{R}^{m \times d}$. Then, a second SimEc f_2 can be trained using the second set of feature vectors $X_2 \in \mathbb{R}^{n \times P}$ to approximate $R^\top$ (or again a subset of it), only that in this case the weights of the last layer are kept fixed as $W_l = Y_1^\top$. Both SimEcs together then provide mapping functions for two different kinds of input feature vectors into the same embedding space such that $f_1'(X_1)f_2'(X_2)^\top = Y_1 Y_2^\top \approx R$.

Preserving non-metric similarities Non-metric similarities are characterized by an eigenvalue spectrum with significant negative eigenvalues. Spectral embedding methods, such as kPCA, require positive semi-definite similarity matrices to compute the low dimensional embedding of the data and would in this case discard the information associated with the negative eigenvalues. However, Laub et al. [106] have shown that this negative part of the eigenvalue spectrum can reveal interesting features in the data and therefore should not be ignored.

A non-metric similarity matrix S is equal to the difference between two similarity matrices S_1 and S_2, where S_1 has the same p positive eigenvalues as S, while the non-zero eigenvalues of S_2 correspond to the q negative eigenvalues of S. Correspondingly, a factorization of S into $YY^\top$ would need to capture the relation between S_1 and S_2, i.e.,

$$S = S_1 - S_2 \quad \Leftrightarrow \quad YY^\top = Y_p Y_p^\top - Y_q Y_q^\top.$$

However, the only way to get this negative part of the product $YY^\top$ would be for the values of Y_q to be imaginary, which is generally not desirable for such embeddings.

With SimEcs it is nevertheless possible to approximate a non-metric similarity matrix S. Since during training S is approximated as $f'(X)W_l = YW_l$ and not $YY^\top$, some dimensions of Y and W_l can have opposite signs, which makes it possible to not only approximate S_1 but also $(-S_2)$. In this case the regularization term $\left\|S - W_l^\top W_l\right\|_F^2$ would be counterproductive.[3]

Predicting multiple pairwise relations at once SimEcs can also be trained explicitly to preserve the information provided by multiple similarity matrices $S_1, ..., S_k$. The easiest way to do this is to simply compute the average of these similarity matrices and then train a SimEc as before on this averaged S. Please note that because SimEcs preserve the information associated with the d largest eigenvalues, the embedding only captures all k similarities if the largest eigenvalues of the k similarity matrices are equal. Therefore, before computing their average, the similarity matrices should first be normalized by dividing them each by their respective largest eigenvalue.

If the focus is not on the similarity preserving embedding itself, but rather it is important to accurately predict multiple similarities or other pairwise relations at the same time, then the SimEc network can be extended to have multiple last layers, i.e., by choosing $W_l \in \mathbb{R}^{d \times n \times k}$ a SimEc can compute

$$f(X) = f'(X)W_l = YW_l = \hat{R} \in \mathbb{R}^{m \times n \times k}.$$

Similarly, in addition to a last layer W_l, the SimEc network can also be extended by a mirrored version of $f'(\mathbf{x}_i)$, thereby adding a decoder part to the network, which can be used to compute the pre-image of an embedding like in the deep kernelized auto-encoder networks [93].

[3] It should be noted that a d-dimensional SimEc embedding generally captures the information associated with the d eigenvalues with the largest absolute values; should the magnitude of the largest negative eigenvalue be smaller than the first d positive values, then this information will still be ignored.

3

Learning Similarity Preserving Embeddings

In this chapter, we give various examples how Similarity Encoders (SimEcs) learn similarity preserving embeddings. First, we show how SimEcs can create the same embeddings as spectral methods such as PCA and Isomap, simply by adapting the architecture of the embedding network and choosing an appropriate target similarity matrix (Sec. 3.1). Then we demonstrate that SimEcs can learn a mapping from an original input feature space into a similarity preserving embedding space even if the target similarities were not computed from the original feature vectors (Sec. 3.2). Additionally, we discuss the influence of regularization and the number of targets on the embedding quality (Sec. 3.3), as well as show that SimEcs can learn a faithful embedding from noisy input data (Sec. 3.4) and when the target similarity matrix contains over 90% missing values (Sec. 3.5). Finally, we demonstrate that SimEcs can predict non-metric similarities and multiple similarities at once (Sec. 3.6).

Comparison of the SimEc performance to related work As SimEcs simultaneously factorize a similarity matrix and learn a mapping into the similarity preserving embedding space, we compare a SimEc's solution to the results found by a combination of the eigendecomposition of S, to get optimal similarity preserving embeddings, and an additional regression model, trained to learn the mapping from the original feature space to the embedding space. As the embeddings produced by the regression model will at most be as good as the original embeddings created by decomposing S [33], in most experiments we only report the optimal performance achieved by the eigendecomposition as a reference. Unless stated otherwise, the target similarity matrices used in the experiments were centered (as it is being done for kernel PCA as well [135]) and, if necessary, normalized to be in the range $[-1, 1]$. The code to replicate the experiments, including specific hyperparameter choices and other settings, can be found online.[1]

3.1 Data points in global and local contexts

Manifold learning algorithms aim to discover a low dimensional manifold that contains all data points in some original high dimensional feature space. The concept of low dimensional manifolds in high dimensional spaces is best illustrated by two popular manifold learning datasets, the "S-curve" and the "cut sphere" (Fig. 3.1). In both datasets, the original data is three-dimensional, however, all data points lie on a two-dimensional manifold, in these

[1] `https://github.com/cod3licious/simec`

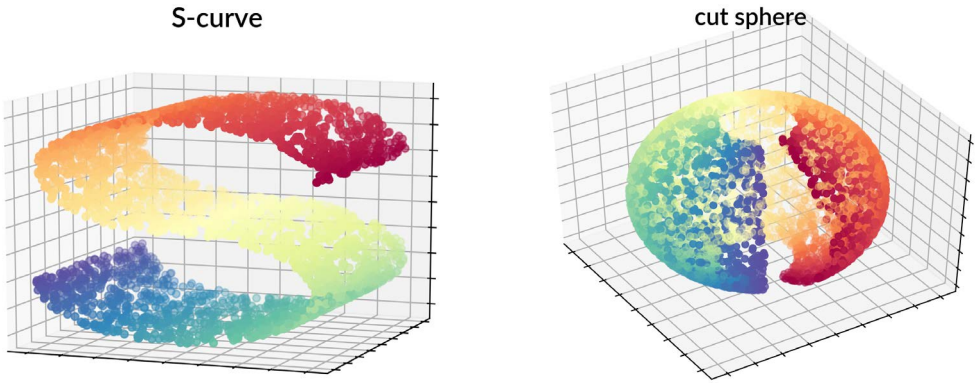


Figure 3.1: Manifold learning datasets "S-curve" and "cut sphere": the data points lie on a two-dimensional manifold in the original three-dimensional space.

cases bend to the shape of an S or wrapped around a sphere that is open on one side and with the poles cut off.

Manifold learning algorithms, such as locally linear embedding (LLE) [165] or Isomap [189], try to find the manifold in which the data points are embedded and project the data to a lower dimensional space where this manifold is unfolded. This is achieved by constructing a similarity matrix for the data points based on their nearest neighbors, e.g., in the case of Isomap by computing the geodesic distances between the points. Then, just like with kernel PCA (kPCA), the lower dimensional embedding is found by computing an eigendecomposition of this similarity matrix. Indeed, Isomap can nicely unroll the manifolds in the two datasets (Figs. 3.2 and 3.3 top right).[2]

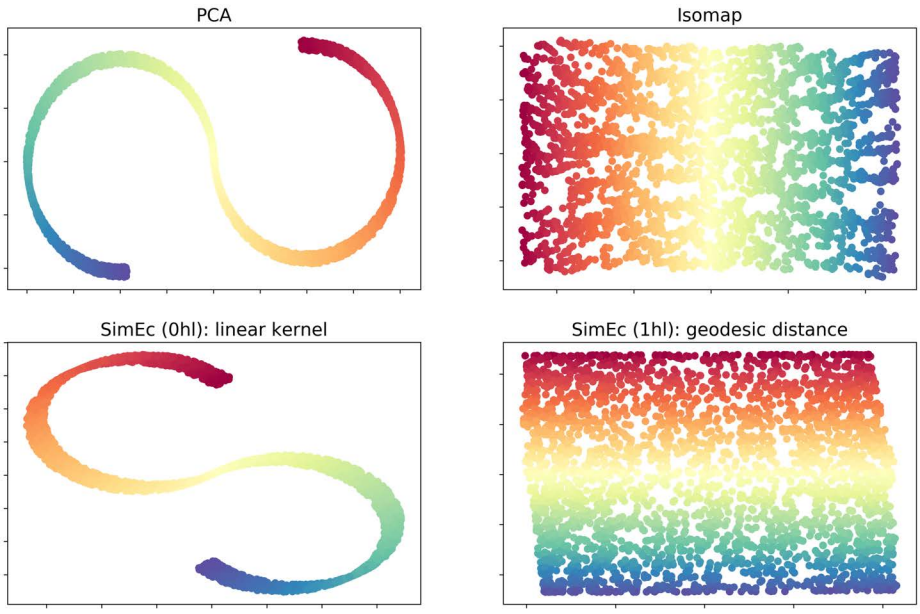


Figure 3.2: Two-dimensional embeddings of the "S-curve" dataset with spectral methods PCA and Isomap as well as corresponding SimEc setups with zero or one hidden layers (hl).

[2]The S-curve dataset was sampled with 3000 data points and the cut sphere dataset with 5000 points. The Isomap solution was computed using the scikit-learn implementation [150] by considering ten nearest neighbors when computing the geodesic distance.

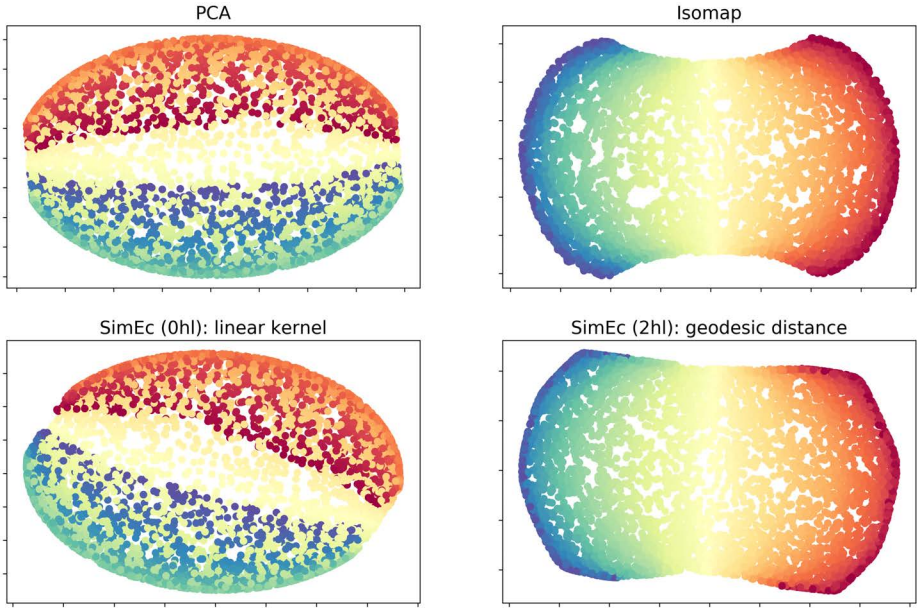


Figure 3.3: Two-dimensional embeddings of the "cut sphere" dataset with spectral methods PCA and Isomap as well as corresponding SimEc setups with zero or two hidden layers (hl).

Discovering the low dimensional manifold on which the data points lie can be very useful in some applications, while in other scenarios it is more important to instead unveil the global structure of a dataset, e.g., show that in the "S-curve" dataset, the points are arranged in the shape of an S, and the "cut sphere" in 3D is actually round. This can be achieved with methods such as PCA [149], or kPCA [173] with a similarity matrix that captures this kind of global information, like a linear kernel or an RBF kernel with a large bandwidth (Figs. 3.2 and 3.3 top left).

Different embeddings can be computed from the same high dimensional data by constructing and decomposing similarity matrices that capture either the global structure of the dataset or the data points' local contexts. The same variety of embeddings can also be created with a SimEc network, simply by using one of these different similarity matrices as targets and adapting the architecture of the first part of the network.

Using the original three-dimensional coordinates of the data points as input to the SimEc network, a mapping to a two-dimensional embedding space can be learned by using either a linear kernel as a target similarity matrix, to create an embedding similar to the one obtained by PCA (and equivalently linear kPCA), or a similarity matrix based on the geodesic distances, to mimic the solution found by Isomap (Figs. 3.2 and 3.3 bottom). As PCA is a linear method, its solution can be recreated by a linear SimEc, i.e., where the first part of the network, $f'(\mathbf{x}_i)$, contains no non-linear hidden layers (hl). Isomap, on the other hand, is a non-linear method, so to learn a mapping that unfolds the data manifold in a similar way, the SimEc network also needs to include one ("S-curve") or two ("cut sphere") additional non-linear hidden layers.

3.2 Similarities from labels

To demonstrate that SimEcs can learn the connection between data points' feature vectors and an unrelated target similarity matrix S, we compute pairwise similarities between data points based on their class labels. For this we use a subset of the MNIST dataset [111],

which contains 28×28 pixel images depicting handwritten digits. We randomly subsampled 10k images from all classes, of which 80% are assigned to the training set and the remaining 20% to the test set. The input feature vectors contain the 784 pixel values of each image, which were normalized by their maximum value and centered to have zero mean. The target similarity matrix computed from the class labels is 1 for a pair of images depicting the same digit and 0 elsewhere.

With increasing embedding dimensionality d, the mean squared error (MSE) between the target similarity matrix S and its approximation $\hat{S}$, computed as the dot product of the embedding vectors, $YY^\top$, is expected to decrease. The eigendecomposition of S provides the optimal similarity preserving embeddings. However, as for new test samples the class based similarities are not available, OOS solutions can not be created as there is no mapping from the original input feature space to the embedding space.

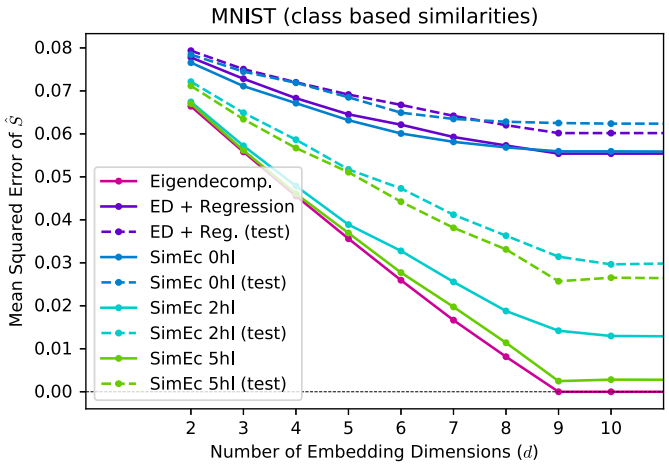


Figure 3.4: Mean squared errors between the class label based target similarity matrix S and its approximation $\hat{S}$, computed as the dot product of the embedding vectors, $YY^\top$, with increasing embedding dimensionality d.

As shown in Fig. 3.4, the embeddings produced by a linear SimEc, where $f'(\mathbf{x}_i)$ consists of only a single linear layer mapping the input vectors into the embedding space, are comparable to those of a linear ridge regression model that learned the connection between the feature vectors and the embeddings produced by the eigendecomposition of S. By using a SimEc with a deeper NN $f'(\mathbf{x}_i)$, with several non-linear hidden layers to map the feature vectors into the embedding space, the error of the approximation gets very close to that of the eigendecomposition.

3.3 A closer look at hyperparameters

Next, we investigate the influence of hyperparameter choices on the SimEc solution. For this, like in the previous section, a subset of the MNIST dataset is used, and a SimEc with one additional hidden layer is trained to create ten-dimensional embeddings to approximate an RBF kernel matrix. Corresponding embeddings created with kernel PCA serve as a reference.

First, we analyze the influence of the regularization term $\lambda \left\| S - W_l^\top W_l \right\|_F^2$ (Fig. 3.5 left panel). While the output of the SimEc network, YW_l, always faithfully approximates the target similarities, the dot product of the embedding vectors, $YY^\top$, only achieves similar

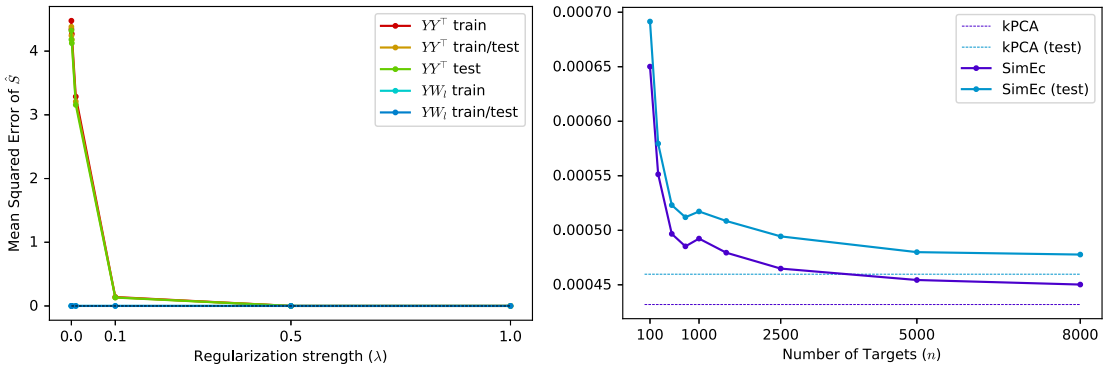


Figure 3.5: *Left:* Importance of the regularization term $\lambda \left\| S - W_l^\top W_l \right\|_F^2$ to ensure not only the output of the SimEc network, YW_l, approximates the target similarity matrix, but also the dot product of the embedding vectors, $YY^\top$. With YW_l it is only possible to predict the similarities between new samples and those used for training the network, while with $YY^\top$ the similarities between new test samples can be computed as well. *Right:* Even if only a fraction of the available targets is used for training, the mean squared error between $YY^\top$ and S is close to the optimal error achieved by kernel PCA.

accuracies when a symmetric factorization of S is enforced.

As discussed in the previous chapter, this regularization term dramatically increases the computational complexity and memory requirements of the training procedure, as it scales quadratically with the output dimensionality. However, often only a fraction of the targets is required for $YY^\top$ to approximate S reasonably well (Fig. 3.5 right panel). Therefore, the computational cost of training a SimEc can be easily limited by choosing an appropriate number of output targets.

3.4 Dealing with noisy input data

As we have already demonstrated by using similarities based on class labels as targets (Sec. 3.2), SimEcs can easily learn the mapping from an original high dimensional feature space to a similarity preserving embedding space, even if the given target similarities were not computed (directly) from the original feature vectors. This ability is important also when dealing with noisy input data.

For the following experiments, we used 5000 randomly selected images from all classes of the MNIST dataset. The original 784-dimensional input vectors were mean-centered and used to compute a linear kernel matrix, which serves as the target similarity matrix in the task. Two cases of noisy input data are considered: In the first case, the noise is added to the original feature vectors, where the standard deviation (std) of the noise is varied relative to the standard deviation of the original data. This corresponds to a setting where a set of sensors measures relevant but noisy data. In the second case, the original data is left as is, but additional dimensions are added to the feature vectors that contain noise with the same standard deviation as the original features. This corresponds to a "big data" scenario, where lots of sensors collect high quality data, but only a fraction of the sensors measures data that is actually relevant for the task at hand. Additionally, in both cases two types noise are considered, either random white noise or correlated noise drawn from a multivariate normal distribution with zero mean and a random positive semi-definite covariance matrix.

If a target similarity matrix and input feature vectors are provided separately and it is unclear how (or if) one was computed from the other, it is advisable to first compute an

eigendecomposition of the target similarity matrix to get optimal embeddings and then train a regression model to learn a mapping from the original feature vectors to the embedding space to compute the embeddings for new test samples. Alternatively, of course, instead a SimEc could be used to both factorize the target matrix and get the mapping into the embedding space at the same time. On the other hand, if the target similarity matrix was computed from the original input data using a known kernel function, then kernel PCA can be used not only to get optimal embeddings, but also to map new data points into the embedding space by computing their kernel map (i.e. similarity to the training samples).

It was previously demonstrated that kernel PCA can easily deal with moderate amounts of (random) noise [128, 135, 172]. However, computing an embedding with kPCA from extremely noisy data, especially in the case of correlated noise, leads to large mean squared errors between the original linear kernel matrix computed from the noise free data and the approximation computed as the dot product of the kPCA embeddings. This problem is not encountered though when training a SimEc with both the noisy input data and the separate noise-free linear kernel target matrix (Figs. 3.6 and 3.7; note the absolute MSE values on the y-axis). This shows that, due to the ability to learn the connection between input vectors and arbitrary target similarities, SimEcs can easily deal with extremely noisy input data.

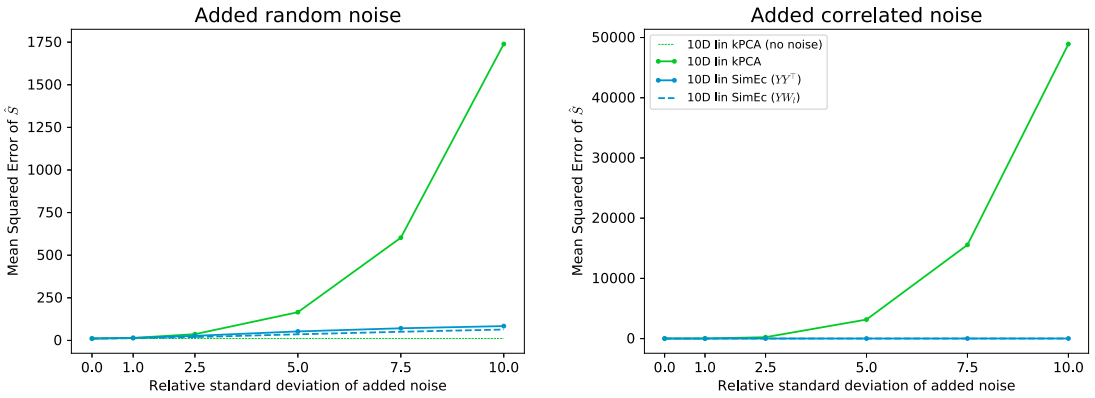


Figure 3.6: Random *(left)* or correlated noise *(right)* with varying std added to the input data and corresponding MSEs of ten-dimensional kPCA and SimEc embeddings.

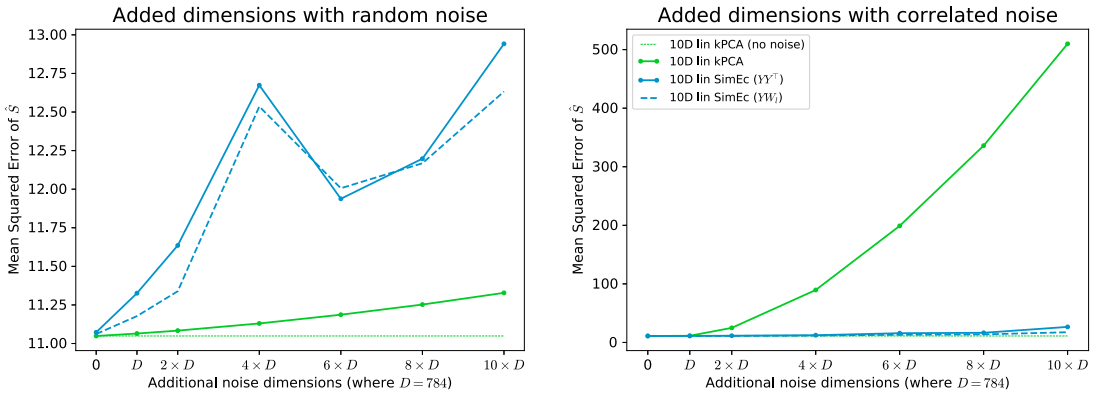


Figure 3.7: Additional input dimensions containing random *(left)* or correlated noise *(right)* and corresponding MSEs of ten-dimensional kPCA and SimEc embeddings.

3.5 Dealing with missing target similarities

As pairwise data can be expensive to collect or be systematically unavailable (e.g. in movie ratings), target matrices will often contain many missing values. An exact eigendecomposition of a matrix with missing values can not be computed, and instead these entries in the matrix need to be filled, e.g., by the mean of the given targets. However, this results in an almost linear increase in the mean squared error between the full target matrix (in this case an RBF kernel matrix) and the approximation computed as $YY^\top$ (Fig. 3.8). With the embeddings created with a SimEc, on the other hand, the target similarities can be faithfully approximated even if the target matrix contains over 90% missing values. For this, the error in the backpropagation procedure, when training the SimEc network, is computed by only considering the existing entries in the target matrix.

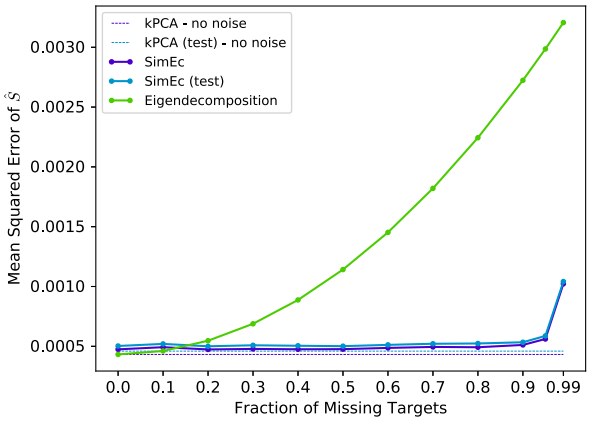


Figure 3.8: Influence of missing values in the target similarity matrix. Kernel PCA computed on the full matrix serves as the optimal reference error, while the green curve depicts the error resulting from the eigendecomposition of the matrix where the missing values were filled by the mean of the matrix.

3.6 Predicting non-metric similarities and more

In the following experiments, we demonstrate that SimEcs can predict non-metric similarities and multiple similarities at once. For these experiments, we randomly subsampled 5k images from the MNIST dataset depicting zeros and sevens and we refer to this as the "MNIST 0/7" dataset. The target similarity matrix S was computed using the Simpson similarity coefficient on binarized feature vectors:

$$S_{ij} = \frac{\#\{\text{pixels that are black in both } i \text{ and } j\}}{\min\{\#\{\text{black pixels in } i\}, \#\{\text{black pixels in } j\}\}}.$$

As previously shown by Laub et al. [106], the eigenvalue spectrum of this similarity matrix contains significant negative eigenvalues and embeddings based on the corresponding eigenvectors reveal interesting features. While the embedding based on the largest (positive) eigenvalues (i.e. kPCA solution) separates the data points by class (Fig. 3.9 top left), an embedding based on the most negative eigenvalues sorts the images by stroke weight (Fig. 3.9 top right). SimEcs are able to create embeddings based on such non-metric similarities as well (Fig. 3.9 bottom). Yet, while the embedding, Y, learned by a SimEc (with one hidden layer) captures the features associated with the negative eigenvalues, the embedding vectors'

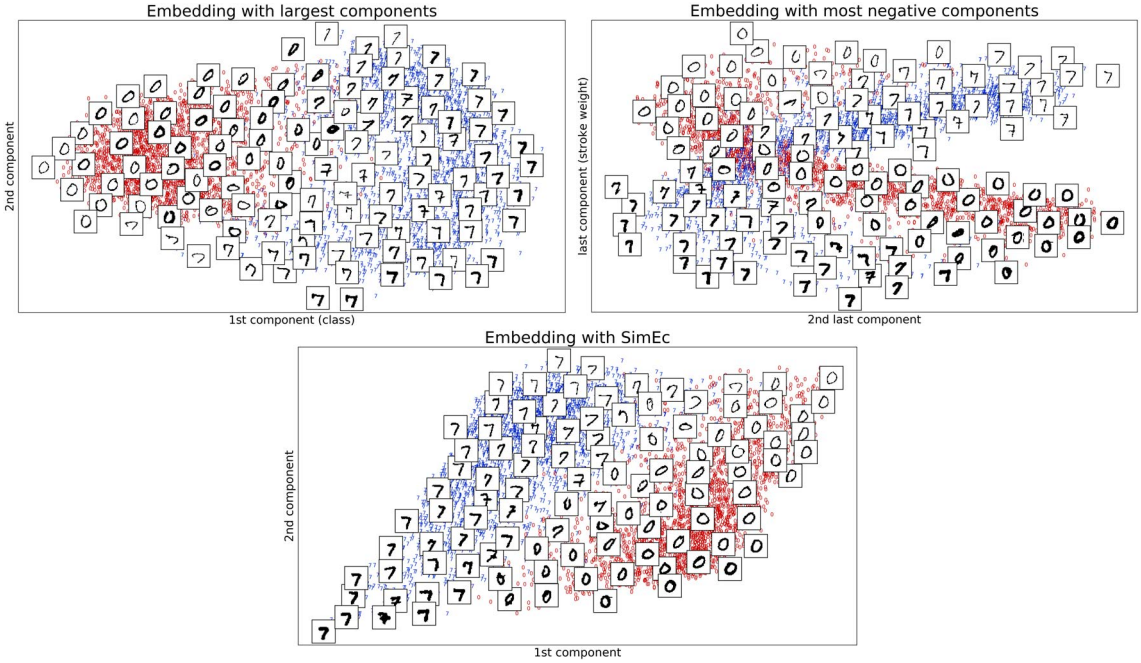


Figure 3.9: Embedding of the MNIST 0/7 dataset based on the largest *(top left)* and most negative *(top right)* eigenvalues of the Simpson similarity matrix, as well as a SimEc embedding of dimensionality $d = 2$ based on the normalized sum of the similarity matrices associated with the largest and most negative eigenvalues[4] *(bottom)*.

dot product, $YY^\top$, would not optimally approximate S, as for this the dimensions associated with the negative eigenvalues would have to be imaginary. However, by computing $\hat{S} = YW_l$ the non-metric similarities can be predicted quite well (Fig. 3.10), with errors closer to those of the embeddings based on both positive and negative eigenvalues instead of those of the embeddings based only on the largest positive eigenvalues (i.e. a regular kPCA embedding).

As discussed in the previous chapter, a non-metric similarity matrix can be decomposed as $S = S_1 - S_2$, where S_1 and S_2 can be computed as the dot product of the embeddings based on positive and negative eigenvalues respectively. Besides preserving features corresponding to both parts of the eigenvalue spectrum, SimEcs can also be used to directly predict these two similarity matrices simultaneously. This can either be accomplished by computing a new similarity matrix as $S_1 + S_2$, or by stacking the two matrices, thereby creating a tensor $\in \mathbb{R}^{m \times m \times 2}$. To preserve the information present in both similarity matrices to an equal extent, S_1 and S_2 first have to be normalized by their respective largest eigenvalue, as SimEcs generally learn embeddings based on the overall largest eigenvalues.

Unsurprisingly, the mean squared error between either S_1 or S_2 and $\hat{S}$ computed with a SimEc trained to approximate $S_1 + S_2$ is worse than that of a SimEc trained specifically to approximate either S_1 or S_2 alone (Fig. 3.11). The dot product of the embedding vectors $YY^\top$ of a SimEc trained to approximate the tensor containing the stacked matrices S_1 and S_2 also results in errors comparable to those of the $S_1 + S_2$ SimEc, because a single embedding contains the information about both similarity matrices here as well. However,

[4] A similar plot can be created by training a SimEc with $d \approx 10$ on the non-metric similarity matrix S ($d > 2$ because the largest negative eigenvalue is smaller than several of the larger positive EVs), then rotating the resulting SimEc embeddings with PCA and selecting those two components that correspond to the largest positive and largest negative EV.

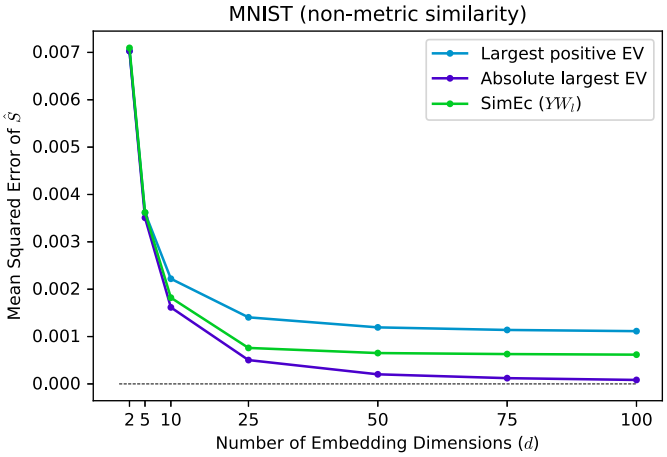


Figure 3.10: Mean squared errors of the non-metric similarity matrix S and the dot product of the embeddings based on the largest positive eigenvalues, the embeddings based on the largest absolute eigenvalues (where dimensions associated with negative eigenvalues were cast as imaginary numbers), and the prediction of S with a SimEc as YW_l.

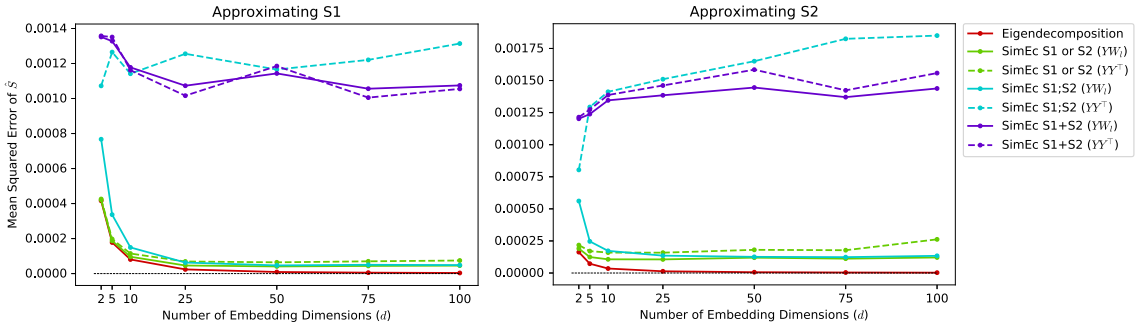


Figure 3.11: Mean squared errors when approximating either S_1 *(left)* or S_2 *(right)*. The eigendecomposition of the respective matrix yields the optimal similarity preserving embedding. Depicted in green are the errors achieved with a SimEc trained to approximate either S_1 or S_2 alone; shown in cyan are the errors achieved with a SimEc trained to approximate the tensor containing the stacked matrices S_1 and S_2; while the purple curves show the errors achieved with a SimEc trained to approximate the matrix $S_1 + S_2$. Continuous lines depict the prediction of S as YW_l, while dashed lines correspond to the approximation as $YY^\top$.

the prediction of the individual similarity matrices in the tensor as YW_l yields errors as low as the prediction of the SimEc trained to approximate only one of the matrices, because the last dimension of the tensor W_l contains information specific to either one of the similarity matrices.

4 Predicting Pairwise Relations in Practice

In this chapter, we illustrate how a Similarity Encoder (SimEc) can be used to predict pairwise relations in practical applications such as link prediction and recommender systems.

The relation between a pair of entities can either be binary (e.g. item i was bought by user j) or quantitative (e.g. item i received a four out of five stars rating by user j). The known relations between entities are stored in a matrix $R \in \mathbb{R}^{m \times n}$, which is typically very very sparse, and the task is to fill in the missing values (e.g. to predict whether user j might be interested in item i).

Applications that involve the prediction of pairwise relations between entities can be very diverse and include tasks such as link prediction ("Was Barack Obama born in the USA?") [29, 47, 141, 181, 195, 209], recommender systems ("Would this user likely buy that toaster?") [18, 43, 62, 90, 96, 158, 203, 205], or drug-target interaction prediction ("Could drug A interact with protein B?") [35, 56, 66, 137, 208, 210, 219]. Needless to say, there exists a huge body of research in each of these fields, with very specialized models specifically designed to solve each task, fine tuned for a certain dataset, and tweaked to get even the last 0.05% in performance improvement – because if one shows someone the right toaster at the right time, money can be made. With SimEcs, we can not – and do not aim to – compete with these models. However, what we do want to show in this chapter is that a plain vanilla SimEc can serve as a good baseline model across different pairwise relation prediction tasks.

Before discussing the tasks of link prediction and recommender systems in more detail, let us first differentiate further between different pairwise relation prediction scenarios. As already mentioned, tasks like product recommendation essentially consist of predicting missing entries in a large matrix R containing pairwise relations, e.g., the user ratings for some items. However, besides the sparse matrix containing the pairwise relations, generally one can also construct some feature vectors for the entities (i.e. items and users), for example, based on textual descriptions. These can come in especially handy when predictions need to be made, e.g., for new items that did not receive any user ratings so far (cold start problem) [62, 203, 204].

Three pairwise relation prediction task types with increasing difficulty can be distinguished (on the example of product recommendation):

T1: Predict missing ratings for existing items and users.

T2a and T2b: Predict ratings for new items and existing users (a) or new users and existing items (b).

T3: Predict ratings for new items and new users.

For tasks **T2a/b** and **T3**, feature vectors describing items and/or users are required, which might not always be available (e.g. due to privacy concerns).

4.1 Link prediction

Link prediction is a very straightforward pairwise relation prediction task from category **T1**: Given two entities e_i and e_j, predict whether the relation r_k holds between them, i.e., if the triplet (e_i, r_k, e_j) should be assigned the label '`true`' or '`false`'. For example, (*Obama*, *born_in*, *USA*) = `true` and (*Paris*, *capital_of*, *Germany*) = `false`. Instead of learning a prediction model for a single pairwise relation, however, link prediction tasks typically require the prediction of several tens or even hundreds of relations between a set of entities. Furthermore, for some of the relation types, only a handful of positive examples might be included in a dataset, which makes it essential to treat this as a multi-task learning problem in order to acquire as much information as possible about individual entities across all their relations.

Datasets For our experiments, we used two datasets in two variations each: The *FB15k* dataset [29] is a subset of Freebase, which is a large database with facts about the real world. The dataset contains 14,951 entities and 1,345 relations. The *FB15k-237* dataset [193] is a subset of FB15k, where all the inverse relations contained in the validation and test sets of FB15k were removed, as these lead to overly optimistic performance measures. FB15k-237 contains slightly fewer entities than FB15k and only 237 relations.
The *WN18* dataset [29] is a subset of WordNet, which contains lexical relations between words. The *WN18RR* dataset [47] is again a subset of WN18 with the inverse relations removed. Both datasets contain 40,943 entities and 18, respective 11, relations.

Related work Popular link prediction models [29, 47, 141, 181, 195, 209] typically learn embedding vectors for the set of entities across all relations as well as some weights specific to each relation. Most often, this involves the factorization of a tensor containing the relations of the training triplets. For a more detailed comparison of different link prediction models we refer the interested reader to [14], from which we also reproduced the results achieved with some of the state-of-the-art models on the datasets mentioned above as a comparison to the performance reached with SimEcs.

SimEc approach For the link prediction task, a SimEc is trained with one-hot encoded entity vectors $\mathbf{x}_i \in \mathbb{R}^m$ for all m entities as input and a binary matrix $R \in \mathbb{R}^{m \times m}$, containing the known positive relations between the entities, as the target. The SimEc architecture consists of three layers: the first (linear) layer maps the m-dimensional input vector to a d-dimensional entity embedding (with $d = 100$), the second (linear) layer consists of a $d \times d$ weight matrix, and the last layer maps the d-dimensional embedding back to an m-dimensional output vector, which is then compared to the corresponding row from the target matrix R. Since the targets are binary, a sigmoid activation function is applied to the output of the SimEc network and the backpropagation error is computed using the binary cross-entropy loss function.
As many of the relations in the examined datasets only contain very few examples, it is important to use the data available on all relations to train the network in order to learn meaningful entity embeddings. Therefore, first a target matrix *containing all relations* is compiled and the SimEc network is trained for several hundred epochs to predict all relations between the entities at once. Afterwards, the target matrix for a single relation is used

to fine-tune the weights of the SimEc network for the prediction of this specific relation between the entities. Here, to avoid overfitting on the few available training examples, the validation set is used for early stopping, i.e., as soon as the results on the validation set stop improving, the training procedure is terminated and the model is evaluated on the test set.

At test time, for one relation r_k and some embedding e_i, the pairwise relation scores for all $(m-1)$ other entities are predicted and the entities are sorted according to these scores in descending order (while ignoring those entities with a known relation to e_i). Then the rank of the target entity e_j in the sorted list is determined. Based on these ranks, the performance measures are computed: the mean rank (MR), the mean reciprocal rank (MRR), and the fraction of hits when considering only the first ten, three, or only the first element in the list (H@10, H@3, H@1).

Results The link prediction results on all four datasets are reported in Tables 4.1 and 4.2, where the performance of the other models is taken from [14], based on the original results reported in the respective paper of each model. As expected, SimEcs do not outperform the state-of-the-art models, which were specifically designed for the task of link prediction. Nevertheless, the performances are comparable and a SimEc even achieves better results than some of the older models on the FB15k-237 dataset.

Table 4.1: Link prediction results on the WN18 and WN18RR datasets. Results of other models taken from [14].

	WN18					WN18RR				
	MR	MRR	H@10	H@3	H@1	MR	MRR	H@10	H@3	H@1
DistMult [209]	902	.822	.936	.914	.728	5110	.430	.490	.440	.390
ComplEx [195]	–	.941	.947	.936	**.936**	5261	**.440**	.510	**.460**	**.410**
ConvE [47]	**374**	**.943**	**.956**	**.946**	.935	**4187**	.430	**.520**	.440	.400
SimEc	*528*	.709	.928	.827	.572	6076	.279	.409	.344	.196

Table 4.2: Link prediction results on the FB15k and FB15k-237 datasets. Results of other models taken from [14].

	FB15k					FB15k-237				
	MR	MRR	H@10	H@3	H@1	MR	MRR	H@10	H@3	H@1
DistMult [209]	97	.654	.824	.733	.546	254	.241	.419	.263	.155
ComplEx [195]	–	**.692**	**.840**	**.759**	**.599**	339	.247	.428	.275	.158
ConvE [47]	**51**	.657	.831	.723	.558	**244**	**.325**	**.501**	**.356**	**.237**
SimEc	269	.478	.724	.568	.345	458	*.281*	*.441*	*.309*	*.201*

4.2 Recommender systems

Recommender systems are about better understanding user behavior and similarities between items. We first examine the task of predicting the rating a user might give to a certain item (where items for which a high rating was predicted would be advertised to this user). Then we turn to content based recommendations, where similar items should be identified based on their content descriptions alone (as opposed to similar patterns in user ratings), which

enables the promotion of related items alongside an item a user is interested in, even if these items did not receive any user ratings so far (cold start problem).

Dataset For our experiments, we used the 10m movielens dataset[1] [72], which contains ten million star ratings for over 10,000 movies by almost 70,000 users – which means about 690 million ratings are missing. In addition to these pairwise relations, we downloaded for every movie (if available) the corresponding information from The Movie Database using their API[2]. This includes the movies' genres, director, keywords, and other information. From this additional data, a binary feature vector is constructed for every movie. There is no additional data available about the users.
For the rating prediction task, train/test splits are created depending on the task scenario (T1/T2a) by either randomly subsampling the 10m pairwise ratings directly (70% train, 30% test) or randomly selecting 30% of the movies for the test set and subsequently removing all their ratings from the training set.

Rating prediction

The rating for a specific item i by a user j can be decomposed into the sum of the overall average of the ratings, μ, the average of the ratings of this item, μ_i, as some items might be generally of higher or lower quality than others, the average of the ratings this user has previously given, μ_j, as some users are more critical than others, and then a residual ϵ [96]:

$$R_{ij} = \mu + \mu_i + \mu_j + \epsilon.$$

A lot of information is already captured by these averages and it is therefore useful to compute a residuals matrix as

$$R'_{ij} = R_{ij} - (\mu + \mu_i + \mu_j)$$

and use only the entries of this residuals matrix to train a subsequent model, which can then pick up on the remaining patterns in the pairwise relations. For a given user and item, such a model will then predict the residual and the final rating is computed by adding the respective averages to this [18, 88, 96, 171].

As there are feature vectors available for the movies, but not the users, only the task scenarios **T1** (prediction of missing ratings for existing movies and users) and **T2a** (prediction of ratings for new movies and existing users) are examined in the following experiments. Nevertheless, for completeness, we quickly discuss some baseline methods and SimEc setups that can be used to solve all three tasks.

Baseline methods Several simple approaches can be used to predict the rating (or residual of the rating) of an item by a user:

Predict average: This is the simplest possible baseline: ignore the residuals and just predict a rating R_{ij} as $\mu + \mu_i + \mu_j$ (or for new items and users as the overall average μ) [96, 131, 184]. This solves **T1**, **T2a/b**, **T3**.

Predict average + SVD of the residuals matrix: By factorizing the residuals rating matrix using (iterative, weighted) singular value decomposition (SVD), one can compute a low rank approximation of the residuals matrix and use these approximate values as predictions for the missing values [88, 96, 171]. This can only be used to solve **T1**.

[1] `https://grouplens.org/datasets/movielens/10m/`
[2] `https://api.themoviedb.org/`

Predict average + SVD & Regression: Given some feature vectors for items or users and the low rank approximation of the residuals matrix computed with SVD, the mapping from the items' (or users') feature vectors to their low dimensional embeddings can be learned by a regression model. By computing the scalar product of the embedding vectors, the residuals can be approximated as before. This is an extension of the SVD approach to additionally solve either **T2a** or **T2b**, or **T3** if models are learned for both sides of the factorization.

Regression/Classification model: This approach is different from the so-called latent factor models discussed above. Here, a regression or classification model (depending on the form of the pairwise data, i.e., continuous ratings or binary interactions) is trained by using as input the feature vectors for an item and a user and as the target their relation (see Sec. 1.2 for further details on such metric learning models). One possible realization of such a model could involve two neural networks to map the individual feature vectors into a common lower dimensional embedding space, where the relation between two instances can be computed as the cosine similarity of their embedding vectors [35, 43, 90, 205]. This approach can be used to solve all tasks **T1**, **T2a/b**, **T3**, provided the corresponding feature vectors are available.

SimEc approach Different SimEc setups can improve upon the SVD & Regression setup discussed above:

SimEc with identity matrix I as input: By training a SimEc to factorize the residuals matrix using the identity matrix I as input, the solution obtained with an SVD can be recreated. Correspondingly, this only solves **T1**.

SimEc with feature vectors X as input: By using either item or user feature vectors X as input to the network when learning the factorization of the residuals matrix, the connection between the items'/users' features and their pairwise relations is learned, which allows for a rating prediction for new items/users, i.e., this additionally solves **T2a** or **T2b**.

Train a second SimEc with feature vectors and fixed last layer weights: After training, e.g., a SimEc with item feature vectors as input to learn the factorization of the residuals matrix, this SimEc can be used to compute the item embeddings Y. Then a second SimEc can be constructed, which uses the user feature vectors as input to factorize the transpose of the residuals matrix. However, in this case, the weights of the last layer of the SimEc are fixed by setting them to the transpose of the embedding matrix Y computed for the items. After this second SimEc is trained, both SimEcs can be used to compute item and user embeddings respectively and then the rating can be predicted by computing the scalar product of both embedding vectors. This way, the ratings for new items and users can be predicted given their feature vectors, i.e., this approach can be used to solve all tasks **T1**, **T2a/b**, **T3**.

If the rating matrix contains explicit ratings (i.e. numerical ratings or likes and dislikes), all available entries can be used to train the above models. If, on the other hand, the pairwise relations in the matrix only represent implicit feedback (e.g. the user listens to music by certain artists, which indicates he likes them, but it is unclear whether he does not listen to other artists because he is unaware of them, or because he does not like them) [159], then the given entries in the matrix can be used as positive examples and additionally a random sample of the missing entries can be used as negative examples [205].

Results For both tasks **T1** and **T2a**, the missing ratings are either predicted as the respective average ratings, or by additionally considering the predicted residuals computed with an SVD or SimEc. Table 4.3 lists the root mean squared errors (RMSE) achieved with the different methods on both tasks. As an SVD computes the optimal approximation of the residuals matrix (in a least squares sense), it is unsurprising that this yields the best results for task **T1**. However, when using the feature vectors as input to predict the ratings for new movies (task **T2a**), the best results are achieved with a SimEc.
It is important to note here, that the regression model used to learn the mapping to the SVD embeddings is a ridge regression model, where the value of the regularization parameter was determined in a cross-validation. Similarly, the SimEc architecture did not contain any non-linear hidden layers, which means both models were linear and had therefore, in principle, the same capabilities of arriving at equally good solutions. However, the fact that a SimEc outperformed the SVD & Regression combination again demonstrates, that it is better to learn the factorization and mapping to the embedding space simultaneously [33], even if the embeddings obtained by SVD alone might yield a better approximation of the target matrix.

Table 4.3: Overview of the movie rating prediction results (as RMSEs). The best result in each task scenario is in bold.

	average	+ SVD	+ SimEc (I)	+ SVD & regression	+ SimEc (X)
T1	0.88614	**0.85891**	0.87660	0.88014	0.86796
T2a	0.97610	-	-	0.97332	**0.96897**

Outlook: Interpretation of predicted ratings In addition to accurately predicting a user's rating for a certain item and therefore generating valuable recommendations, it might also be interesting to understand *why* a user might like a certain item. For this, *layer-wise relevance propagation* (LRP) [6, 8, 9, 10, 13, 95, 104, 132, 134] can be used to identify the features of an item that most contributed to a positive or negative predicted rating. LRP decomposes the predicted rating and propagates it back through the neural network to show the contributions of each feature at the input layer. This way, the predictions made by SimEc models can be made more transparent. We will demonstrate this in the next chapter.

Content based recommendations

For content based recommendations, items are compared only based on their descriptions or other features to identify similar items (Fig. 4.1). This can be useful for recommending similar items alongside an item that a user is interested in or to generate recommendations for new items that did not receive any user ratings yet (cold start problem) [62, 96]. However, identifying two items similar in content, e.g., by computing the cosine similarity of some given feature vectors, does not automatically mean that a user interested in one of these items is also interested in the other. Instead, it is important to find a similarity measure for the items that aligns with the similarity patterns observed in the user ratings [15, 48, 123, 205].

The user ratings based similarity between items can be computed as the cosine similarity between the items' residual ratings vectors. Indeed, the cosine similarity between the movies' feature vectors does not align well with what users rated similarly, i.e., movies that are similar with respect to their feature vectors are not necessarily liked by the same users (Fig. 4.2 left panel). However, by using a SimEc (with one additional non-linear hidden layer)

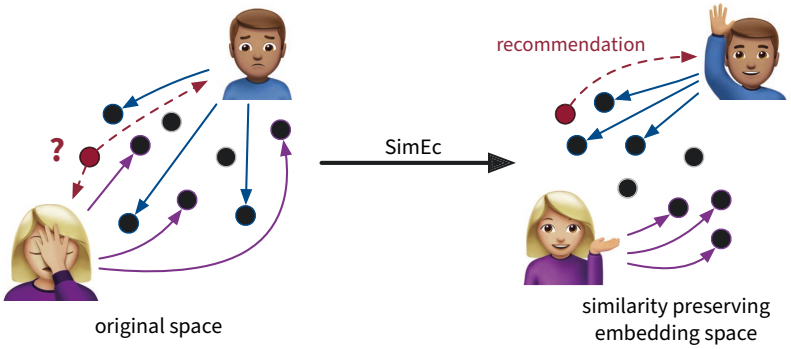


Figure 4.1: Content based product recommendation before and after embedding the item feature vectors into a similarity preserving embedding space. Each dot represents an item and the arrows to the items indicate the user preferences. The task is to recommend a new item (red dot) to those users that might be interested in this item, knowing only the item's description.

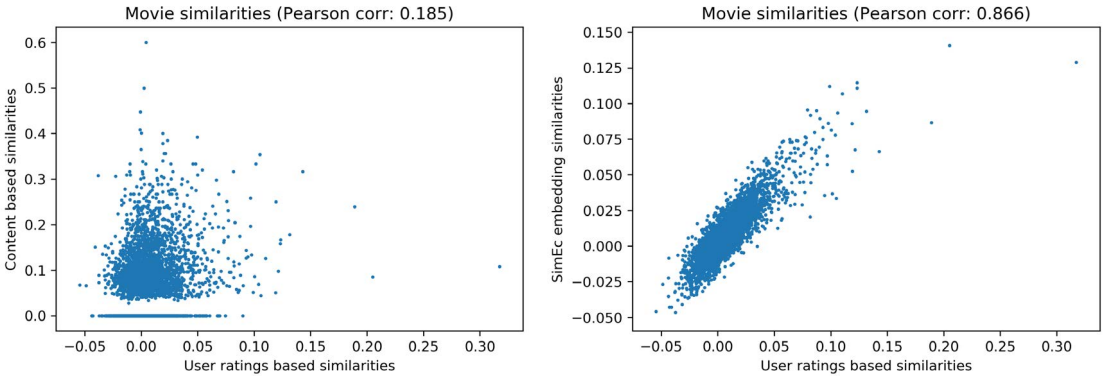


Figure 4.2: Correlations between the similarities based on user ratings (*x-axis*) and simple content based movie similarities (*left*) and similarities after projecting the original feature vectors into a similarity preserving embedding space with a SimEc (*right*). These results are based on a subset of 2k movies that received at least 1000 user ratings.

to learn a mapping from the movies' original feature vectors into an embedding space where the user ratings based similarities are preserved, the scalar product of these embedding vectors results in a similarity prediction that is aligned fairly well with the users' perception of similar items (Fig. 4.2 right panel). Therefore, by first projecting an item's feature vector into the SimEc embedding space and then computing the dot product with all other items' embedding vectors, the most similar items can be identified and recommended alongside the item of interest – even if this item never received any user ratings itself.

5 Explaining SimEc Predictions

In this chapter, we revisit some of the previous experiments and examine one new dataset to demonstrate how the pairwise relation predictions of a Similarity Encoder (SimEc) can be explained using a technique called *layer-wise relevance propagation* (LRP) [8, 9, 10, 13, 95, 104, 105, 132, 134].

5.1 Deconstructing predictions and similarity scores

Explanations of a machine learning model's predictions are often given in a visual form, e.g., as saliency or heat maps. However, when "visualization" and "similarity" is mentioned in the same sentence in the machine learning literature, the research usually focuses on the identification of similar items and their placement close to each other in a two-dimensional plot such as created with t-SNE or self-organizing maps [19, 126]. However, what we are interested in is the visualization of *why* two items are similar, i.e., which of their features contributes the most to their high or low similarity score.

Background & related work To the best of our knowledge, the interpretation and explanation of similarities has previously only been studied with respect to images using techniques specifically tailored to CNNs [185], where the images are passed through the CNN until after the last pooling layer to create an embedding for each image and the similarity is then computed and explained w.r.t. the cosine similarity of these embeddings. However, the CNN itself is trained on a classification task, not on a specific similarity measure that is supposed to be explained. Furthermore, this approach is specifically tailored to CNN architectures with average or max pooling layers, while a SimEc makes no assumptions about the embedding network and LRP can be applied to a variety of network architectures.

LRP aims to make predictions of machine learning models more transparent by showing how much each of the input dimensions of a sample contributed (positively or negatively) to the prediction. For example, in a linear model the prediction for a sample is computed as the scalar product of the input feature vector $\mathbf{x}_i \in \mathbb{R}^D$ and the model's weight vector $\mathbf{w} \in \mathbb{R}^D$, while possibly adding an intercept term b and applying a non-linearity σ to get the probability prediction score for a certain class:

$$\hat{y}_i = \sigma\left(b + \langle \mathbf{x}_i, \mathbf{w} \rangle\right) = \sigma\left(b + \sum_{k=1}^{D} x_{ik} \cdot w_k\right).$$

Here, the contribution of a single input dimension k to the model's final prediction is proportional to $x_{ik} \cdot w_k$. These contributions can then be plotted as a heat map over the input space, which allows humans to grasp the model's reasoning at a glance (Fig. 5.1).

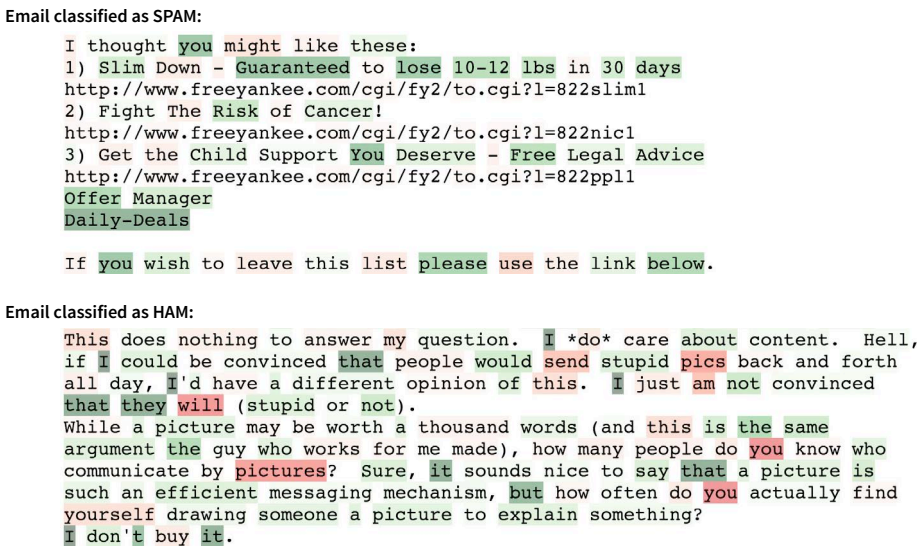


Figure 5.1: Spam e-mail classification with words highlighted depending on whether they contributed positively *(green)* or negatively *(red)* to the prediction of the class 'spam' *(top)* or 'ham' *(bottom)*. Words such as "free", "guaranteed", and "please" are indicative of spam e-mails; furthermore, in regular e-mails people generally express their own opinions ("I"), while spam e-mails directly address the reader ("you") [84, 85].

With layer-wise relevance propagation, this principle of decomposing the prediction score into individual contributions can be applied to deep neural networks: First, the final output at the last layer of a NN is broken down w.r.t. the incoming connections. Then these contributions are propagated backwards through the subsequent layers all the way to the input layer, where the contributions of the individual features can then be identified [8, 9, 10, 13, 95, 104, 105, 132, 134].

Explaining similarity scores Similar to the predictions of a linear classifier, traditional similarity measures [160] can be decomposed as well. For example, a linear kernel (or the cosine similarity, if the individual feature vectors are normalized to have unit length) is just the scalar product of two feature vectors. Therefore, to determine which features from both data points contributed most to their (dis-)similarity, one can simply use the values obtained when multiplying the individual feature dimensions together before summing them up. Other similarity coefficients (e.g. the Rogot-Goldberg score introduced in Sec. 6.2 Eq. 6.1) are computed by counting how many features two data points have in common and in how many attributes they differ. Usually, the focus is on the features both samples share (i.e. the numerator in the formula) and the similarity coefficients differ in how they weight this number by the remaining features (denominator). For the explanation of the score, instead of counting how many attributes the two data points have in common (represented as a in Eq. 6.1), these features can be represented as a binary vector, with a 1 at the feature dimensions that both samples share. Normalizing this vector by the respective denominator then yields the individual contribution of each feature to the final similarity score.

Applying LRP to SimEc

SimEc predictions When using the full SimEc network, e.g., to predict pairwise relations, LRP can be applied like to an ordinary classification network, using instead of the class as the target the output dimension corresponding to the pairwise relation of interest (e.g. for recommender systems the rating prediction for a specific user).
As a special case, if the SimEc network is linear, e.g., when the rating for a movie i by a user j is predicted as

$$r_{ij} = \mathbf{x}_i W_1 W_{2[:,j]} \qquad \text{with } \mathbf{x}_i \in \mathbb{R}^D, W_1 \in \mathbb{R}^{D\times d}, W_2 \in \mathbb{R}^{d\times n},$$

then the contributions of the individual input features can also be obtained by instead using a square matrix $X_i \in \mathbb{R}^{D\times D}$ as input, which has the values of $\mathbf{x}_i$ on the diagonal, resulting in a vector $\mathbf{r} \in \mathbb{R}^D$ where each dimension contains the contribution of the respective input feature.

SimEc similarities When predicting similarities between two data points as the scalar product of their SimEc embeddings, LRP has to be applied twice. For this, first the two original input feature vectors are propagated through the first part of the SimEc network to get their respective embedding vectors and a new last layer is added to the SimEc consisting of these two embeddings. Then, LRP is applied for each of the data points by using the data point's feature vector as input to the adapted SimEc network and the embedding of the respective other data point as the LRP target.
Again, a special case constitutes a SimEc with a linear first part, i.e., when the embedding for a point i is computed as $\mathbf{y}_i = \mathbf{x}_i W_1$. This results in the similarity between two points i and j being computed as

$$\mathbf{y}_i \mathbf{y}_j^\top = \mathbf{x}_i W_1 (\mathbf{x}_j W_1)^\top = \mathbf{x}_i W_1 W_1^\top \mathbf{x}_j^\top \qquad \text{with } \mathbf{x} \in \mathbb{R}^D, \mathbf{y} \in \mathbb{R}^d, W_1 \in \mathbb{R}^{D\times d}.$$

With this reformulation, the scalar product of the embedding vectors becomes

$$\sum_{k=1}^{d} y_{ik} \cdot y_{jk} = \sum_{k=1}^{D} (\mathbf{x}_i W_1 W_1^\top)_k \cdot x_{jk},$$

where again the individual contribution of each feature to the similarity between the two data points is stated in the sum.

5.2 Explaining SimEc predictions in practice

In this section, the flowerpot experiment from the introduction as well as the prediction of movie ratings from the previous chapter are examined more closely, including anecdotal evidence explaining individual SimEc predictions. Additionally, a new dataset with images depicting a variety of shoes associated with different attributes is analyzed to show how the similarity explanations for the same images change depending on the target similarity learned by the SimEc.

Flowerpot experiment As a first illustrative example, we return to the flowerpot experiment from the introduction (Sec. 1.1). Here, the task was to find similarity preserving embeddings for 16 flowerpots given their feature vectors (with the stem size, leaf shape, and three random noise features) as well as a 14×14 similarity matrix generated from human

ratings. When computing the linear kernel on the original feature vectors, the three noise dimensions contribute over-proportionally to the similarity scores, resulting in a poor kPCA embedding (Fig. 1.4). A linear SimEc, on the other hand, correctly learns to compute the (dis-)similarity between two flowerpots based on their stem size and leaf shape (Fig. 5.2).

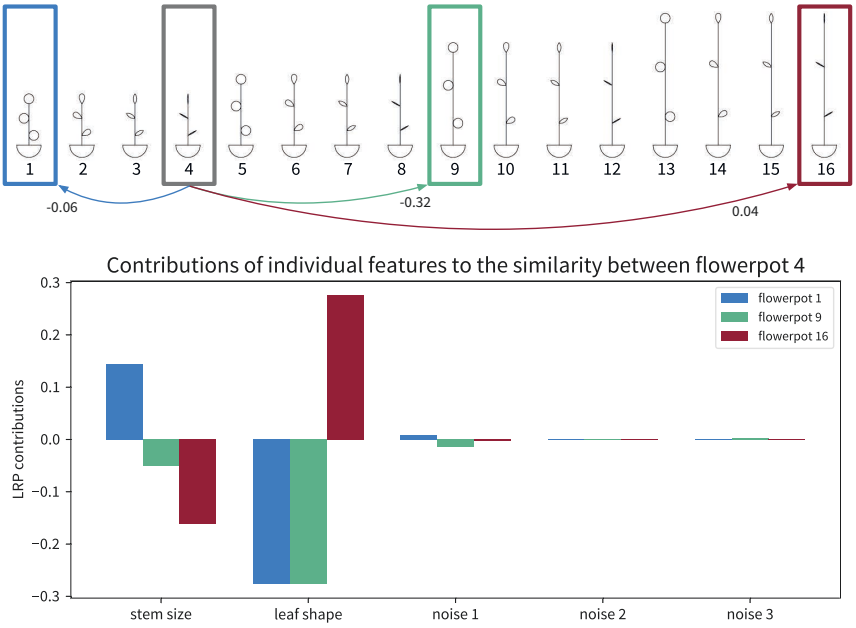


Figure 5.2: Explanations of the SimEc (dis-)similarities between flowerpots based on the contributions of the individual input features. *Top:* Selected plants and corresponding SimEc similarity scores. *Bottom:* Contributions of individual features of each of the three plants to the final similarity score.

Movie rating predictions Some movies can appeal to their audiences for different reasons. For example, the horror-comedy *Shaun of the Dead* from 2004 might appeal both to horror/zombie movie fans as well as a broader, comedy-loving audience. To examine this effect in terms of the movie features that were decisive in the prediction of a high rating, we selected two users, both of which had rated more than 1000 movies and gave *Shaun of the Dead* a 5 star rating with a residual rating larger than 1. To get a better understanding of the taste of these two users, their top 10 above average rated movies as well as the genres of their top 100 rated movies are shown in Fig. 5.3. Since here the focus is not so much on getting correct out-of-sample predictions (although it only makes sense to explain predictions if they are believed to be correct), but rather to examine the contributions of individual features to the final prediction, the SimEc network for this task is trained using all available data from all users with more than 900 ratings (which results in 1063 SimEc targets), i.e., including all ratings for the two selected users. Besides this, the SimEc architecture and training are the same as in Sec. 4.2, i.e., a linear SimEc is trained with the movie feature vectors (consisting of one-hot encoded genres, keywords, and directors) as inputs to predict the residual ratings. After training, the SimEc predicts an above average residual rating for both users – but the movie features that contributed most to these predictions are very different for these users (Fig. 5.4). While for one of the users the genre *Horror* and the keyword *zombie* were amongst the most influential attributes, for the other user, *Horror* contributed negatively to the prediction. However, it should be noted that the keyword

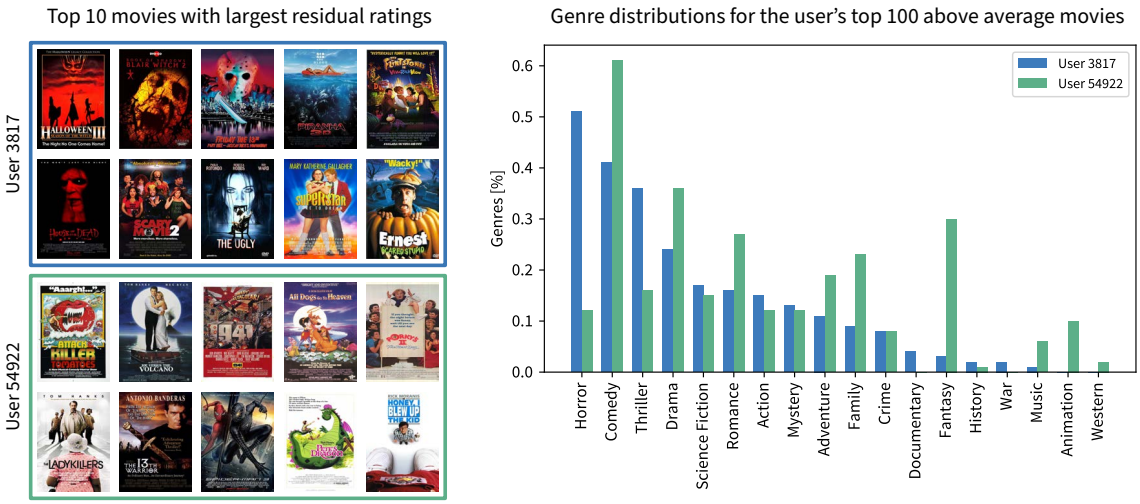


Figure 5.3: User profiles for the two selected exemplary users: top 10 movies with the highest residual ratings as well as genres of the top 100 movies (numbers do not add up to 100% as some movies are assigned multiple genres).

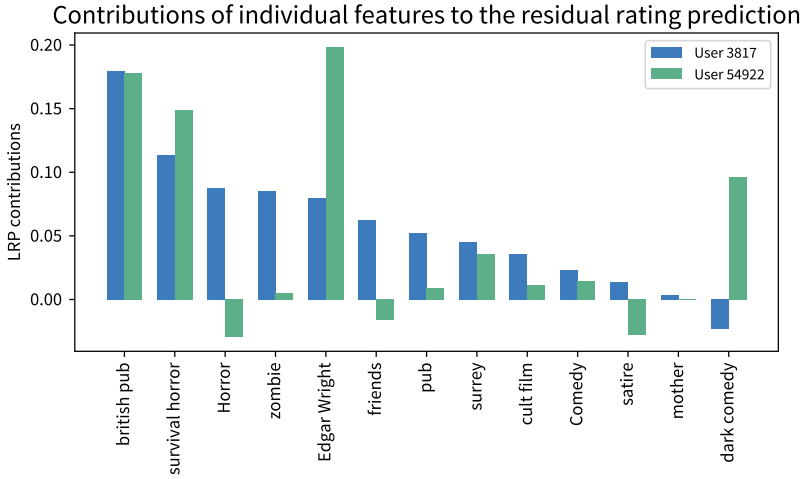


Figure 5.4: Explanations of the SimEc rating predictions in terms of movie features for both users.

survival horror contributed positively for both users, even though it is expected that a user who is not fond of horror movies in general would also not appreciate survival horror movies very much. This is likely an effect due to overfitting, as these keywords are rather specific (e.g. *surrey*, indicating the London suburb Surrey, UK, where the movie takes place). The results (both the prediction accuracy as well as the meaningfulness of the contributions of the individual features) might be greatly improved if more expressive and generalizable feature vectors were available for the movies.

Similarities of shoes The Zappos50k dataset [212, 213] contains about 50k 136×136 pixel color images of different types of shoes. The shoes are all oriented the same way in the images and for each shoe there is meta-data available, including the type of shoe, the brand, the heel height, as well as the closure mechanism (e.g. laces or zipper; can be more than one). We only included shoes for which the heel height as well as the closure mechanism

was given, which resulted in about 30k images. Based on these attributes, two different similarities between the shoes were computed: the first is based on the absolute difference of the heel heights (normalized to be between 0 and 1) and the second is the Simpson similarity of closure mechanisms of each shoe, i.e., *number of closures in common* divided by *minimum number of closures listed for either of the shoes*. The SimEc network used for the analysis was constructed from a small convolutional neural network (CNN) with four convolutional layers with ReLU activations and two max pooling layers mapping to a 512-dimensional embedding, which then mapped to 1000 output units with a linear layer to predict the target similarities for the input images. The SimEc was trained for two epochs on all images for both of the target similarity functions. The LRP analysis of the network was performed using the iNNvestigate Python library [6], which takes a keras model as input and generates an analysis model that reveals the contributions at the input layer for a certain prediction.

We analyzed both the SimEc predictions as well as the similarity between the SimEc embeddings: For the first scenario, three of the 1000 targets were chosen and for each input image we asked "why is this shoe similar to the target shoe (w.r.t. the chosen similarity function)?" (Figs. 5.5 and 5.6). For the second scenario, pairs of images were used to answer the question "why are these two shoes similar to each other?" (Fig. 5.7). For this, different images were used as input to the CNN to create the respective embeddings, which were then used to replace some of the columns of the SimEc's last layer's weight matrix. With this adapted weight matrix, LRP was applied to the model as before, however, since now each of the input images also has its respective target entry, the similarity between two images can be analyzed on both of the images. While the similarity explanations for a SimEc trained on the heel height similarity focus mostly on the lower or back part of the shoes, especially striking for the high heel shoes (Figs. 5.5 and 5.7), the explanations for the closure similarity instead highlight the upper parts of the shoes, e.g., the ankle strap or laces (Fig. 5.6).

Figure 5.5: Explanations of the SimEc predictions for three target shoes (first column) and four input images with the target similarity function based on the shoes' heel heights.

Figure 5.6: Explanations of the SimEc predictions for three target shoes (first column) and four input images with the target similarity function based on the shoes' closure mechanisms.

Figure 5.7: Explanations of the heel-height-similarities of pairs of images, predicted as the scalar products of their SimEc embeddings (mirrored at the diagonal, i.e., imagine the transpose of the matrix to match up the image pairs).

6 Similarity Pre-Training for Supervised Tasks

In this chapter, we demonstrate how, with the right target similarities, a Similarity Encoder (SimEc) setup can be used to pre-train the weights of an arbitrary neural network (NN) architecture to improve this network's performance in a supervised learning task.

Background & related work

For a NN model to achieve a good performance in a supervised learning task, such as image classification, it needs to be trained on a large dataset, typically containing several tens or even hundreds of thousands of labeled examples. This is especially relevant for "deep" NN models, such as multi-layer convolutional neural networks (CNN) [110] used, e.g., for computer vision tasks, which generally comprise several million parameters. If a large labeled dataset is not available for a particular task, e.g., in few shot learning scenarios [59], it can be beneficial to *pre-train* the network on the labeled data from a similar task [3, 50, 53, 146, 179] or on a larger collection of unlabeled examples that might be easier to come by [23, 109, 145, 154, 201]. This way, the NN can already pick up on basic statistics and patterns in the data, before it is then fine-tuned on the data available for the task of interest.

The idea of learning something on one task (the source task) and then utilizing this knowledge to obtain a better accuracy on a different task (the target task) is also called *transfer learning* [22, 58]. Transfer learning is highly related to other practices [147, 197] such as multi-task learning (trying to perform well on multiple learning tasks at once) [168] or domain adaptation (correcting for shifts in the data distribution, e.g., caused by changes in the data collection pipeline) [60, 98, 120, 186, 187]. For the purposes of this investigation, we are only interested in transfer learning in the form of initializing the weights of one NN with those from another network trained on a related task (Fig. 6.1).

In the past, when there were fewer large labeled datasets available than today, NN models were often pre-trained on an unsupervised task, i.e., with unlabeled examples. This can, for example, be accomplished with an *auto-encoder* (AE) network [80, 190], which consists of an encoder part, which maps the original input data to a low dimensional embedding, and a decoder part, which maps the embedding to an output of the same dimensionality as the input data. Such an AE network is trained to minimize the mean squared error (MSE) of the output and the input data, i.e., an AE learns to compresses the input in the encoder part of the network, and then reconstructs the original input from the embedding in the decoder part. The pre-training of a NN model with an AE can either be done layer by layer, or by simply training the whole AE network and then discarding the decoder part and replacing it with a new layer that maps from the embedding to the target outputs for the supervised task [24, 79].

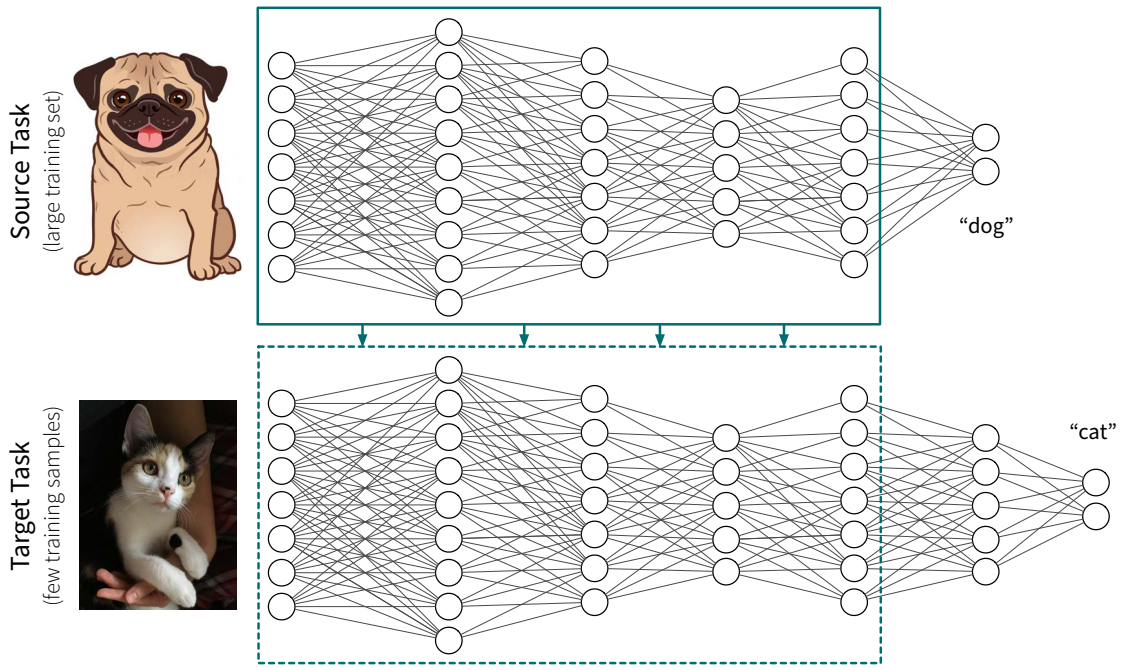


Figure 6.1: Transfer learning with pre-trained weights. The weights of the neural network for the task of interest (target task) are initialized by copying some of the weights from a network with a similar architecture that was trained on a related task (source task).

Nowadays, not only large labeled datasets, such as ImageNet [169] for image classification, but also NN models already trained on these datasets are publicly available [100]. Therefore, when faced with a supervised task with little training data, it has become common practice to use one of these pre-trained models, optionally freeze its weights,[1] and replace the last layer, which maps to the output, with a new layer that is appropriate for the current task (e.g. makes predictions for different classes). This network is then fine-tuned on the data available for the task of interest. Especially for image classification tasks, pre-training on supervised tasks has been proven vastly more successful than on unsupervised tasks [23, 109, 179], presumably because labels are required for a NN model to learn the distinction between relevant pixels displaying, e.g., faces or objects, and irrelevant pixels in the background.

Exactly why or under which circumstances initializing a NN with pre-trained weights leads to a better performance on a supervised learning task is still debated in current literature. NNs are known to be universal function approximators [45], easily capable of memorizing random labels and inputs [7, 215]. Nevertheless, the vastly over-parametrized NNs used today in practice generalize remarkably well to new data points [1, 17, 37, 68, 116, 170]. This is mainly assumed to be due to the properties of the (small batch) stochastic gradient descent (SGD) optimization procedure, which acts as an implicit bias and results in low norm weights [49, 94, 138, 139, 151, 183, 200]. However, while the local minimum found by SGD corresponds to a solution that generalizes comparatively well, SGD is still heavily influenced by the weight initialization and usually finds a minimum very close to the initialization point [112, 136, 218]. Therefore, introducing a bias towards a specific local minimum by initializing a network's weights via pre-training can still be beneficial. The

[1] Freezing weights of a pre-trained network means this part of the network acts as a fixed feature extraction module. However, doing this is generally not recommended as possible co-adaptation effects between the copied and non-copied layers of the source network can result in a suboptimal performance on the target task [211, 217].

minimum that is found after pre-training generally does not result in a better accuracy on the training data, but a lower generalization error on the test set, i.e., pre-training can be treated as a form of implicit regularization [53], similar to data augmentation [67, 78]. The generalization boost yielded by a favorable initialization of the weights can often even be observed when the network is later fine-tuned on a rather large dataset [53, 211].

Crucial to an improved performance via pre-training is the similarity between the source and target tasks [2, 217], i.e., they should rely on similar input features (photographs and medical scans might already be too different) [103, 157] as well as solve a related problem (e.g. image classification tasks with different classes, not an image classification and an image segmentation task) [12, 21, 140, 214]. If the source and target tasks are too different, transfer learning may not yield an improved performance [75, 97, 152] or can even result in "negative transfer" [163] when the initialization of the network's weights from the pre-training is very far from a local minimum such that, especially with little training data, the network can not recover from this initialization and does not converge to a good solution. Depending on the similarity of the source and target tasks, the capacity of the network further influences the possibility or severity of negative transfer, i.e., especially small networks will not benefit from being pre-trained on an unrelated source task [32, 53], just like their performance might be impaired by other forms of regularization.

To get a better feeling for the circumstances under which pre-training may be helpful, our first experiment (Sec. 6.1) examines the relation between task similarity, network capacity, and training set size to highlight the contributions of these factors to the generalization boost achieved with transfer learning on an image classification task. W.r.t. task similarity, learning similarities with a SimEc network should be a perfect source task: the same features are used as inputs and in the extreme case, the target similarities learned by the SimEc can be calculated directly from the prediction targets of the target task, i.e., a model can be pre-trained on a semantically identical yet structurally different task. After the SimEc network is trained on the source task, its last layer, mapping from the embedding to the approximated similarities, is replaced according to the requirements of the target task. We demonstrate the effectiveness of such a similarity-based pre-training empirically in an image classification task (Sec. 6.1) and for the regression problem of predicting chemical properties of molecules (Sec. 6.2).

6.1 Pre-training for image classification

The first set of transfer learning experiments are conducted on an image classification task. In the first experiment, we examine under which conditions transfer learning, in the form of initializing a neural network with pre-trained weights, is successful in general. Then we explore to what extent pre-training with similarities can improve the performance on such a supervised learning task.

The image classification dataset employed for both experiments is the CIFAR10 dataset [99] (Fig. 6.2 top), which consists of altogether 60k 32×32 pixel RGB images, distributed equally amongst ten classes of objects and animals. All images were mean-centered and normalized to have a standard deviation of 1. We used the standard train/test split (50k/10k images respectively) and for each task randomly selected 5k images from the training set as a validation set for early stopping.

Does pre-training help in general?

Whether pre-training will improve the performance on the following supervised learning task is determined by a variety of circumstances. In this experiment, we examine the connection between the performance gain through pre-training and the size of the training set for the target task, the model capacity (in the form of a varying number of units in the first hidden layer of the network), and the relatedness of the source and target tasks (Fig. 6.2).

The whole dataset is split into two tasks A and B, where A contains all images belonging to the first five classes, while B contains the images of the remaining five classes. Note that this leads to a balanced split with an equal number of animal and object classes in both tasks. In this experiment, task B is the target task for which we are interested in the test performance. We examine three different pre-training scenarios with source tasks more (B; A *&* B) or less (A) related to the target task (B). In each case, the network is trained for 25 epochs using 20k training samples of the respective source task. After pre-training, in three runs with different random seeds, the network weights are fine-tuned for another 25 epochs on 25-20,000 randomly selected training examples from the target task B before the test accuracy is computed. After each epoch during pre-training as well as fine-tuning of the network, the validation set is used to compute the validation accuracy to determine the best model. The neural network model used for the task is a simple feed-forward network with two hidden layers, where the number of units in the first hidden layer is varied to examine

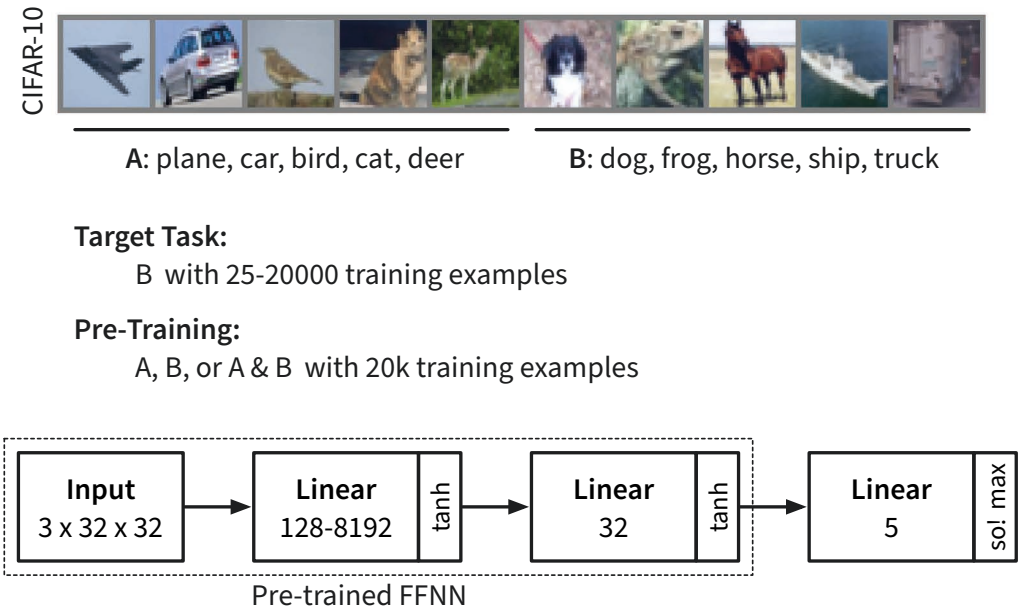


Figure 6.2: Overview of the experimental setup for the general transfer learning experiment. The ten classes of the CIFAR10 dataset [99] are divided amongst tasks A and B, where B constitutes the target task and pre-training is performed either on the target task itself, a random selection of 20k images from tasks A *and* B, or on task A. A simple feed-forward neural network (FFNN) is used for the task, where in the pre-training cases the weights between the input and first hidden layer, as well as the first and second hidden layers are copied from the network trained on the respective source task, while the weights from the second hidden layer to the output layer are always randomly initialized. The network is then trained/fine-tuned using a varying number of training samples from the target task B before computing the accuracy on the test set.

the effect of different model capacities on the transfer learning performance.

The accuracies on the test set of task B for the different pre-training scenarios are shown in Fig. 6.3. As expected, the generalization boost achieved by pre-training the network is strongest for larger network architectures, which would generally require more training samples, as well as for very similar source tasks (B or A & B). Pre-training on a somewhat unrelated task (A), on the other hand, is only helpful if the network has a large enough capacity (here 2^{11} and 2^{13} units in the first hidden layer), as here the network needs to additionally represent some features not directly related to the target task [53]. While the largest performance improvements can be observed for smaller training set sizes, the generalization boost from most pre-training setups is still present when the fine-tuning happened on the whole training set, even though the networks were always trained until convergence. Together, these results show that pre-training can result in a substantial performance gain for a supervised learning task independent of the training set size if the source task is "sufficiently related" to the target task, where the required task similarity is determined by the network's capacity.

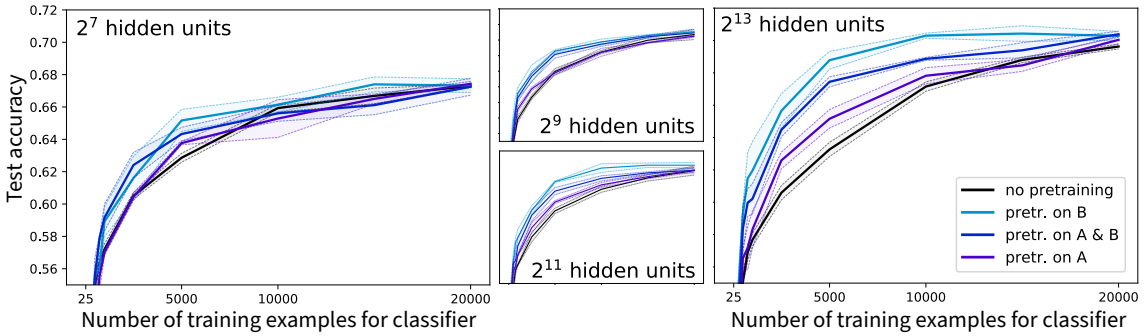


Figure 6.3: Results for the general transfer learning experiment. The accuracies on the test set for the four transfer learning conditions (no pre-training, pre-training on the target task (B) itself, pre-training on a source task that is related to the target task (A & B), and pre-training on a somewhat unrelated source task (A)) are shown for networks of different complexities (128-8192 units in the first hidden layer) and using a varying number of training samples to fine-tune the network on the target task. The fine-tuning/training on the target task was done using three different random seeds; the bold lines show the mean test accuracy while the shaded areas display the standard deviation.

Pre-training with similarities

As demonstrated in the previous experiment, transfer learning in the form of initializing a neural network with pre-trained weights can lead to an improved performance on a supervised learning task if the source and target tasks are related enough. The aim of this experiment is now to demonstrate that learning similarities with a SimEc architecture can be a suitable source task.

The target task in this experiment (Fig. 6.4) is to learn all ten classes of the full CIFAR10 dataset. The neural network architecture used for this task is a small convolutional neural network (CNN), where all but the weights mapping from the last hidden layer to the output are initialized with pre-trained weights. For the pre-training, we compare a SimEc architecture with different target similarities (unsupervised and supervised) to two auto-encoder setups (unsupervised) as well as a pre-training on the target task itself (supervised). The pre-training is always performed for 50 epochs using the full dataset. The

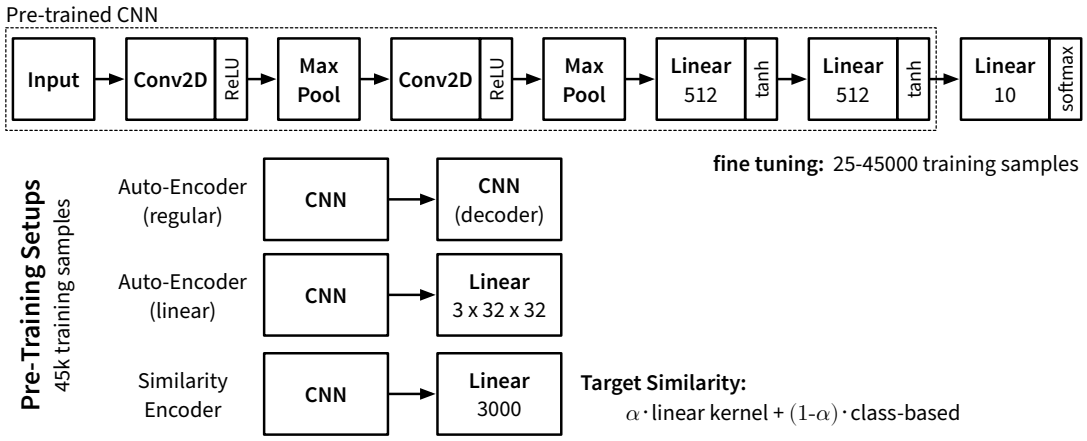


Figure 6.4: Overview of the experimental setup for the SimEc transfer learning example. The neural network used to solve the target task is a small convolutional neural network (CNN) consisting of two convolutional layers followed by max pooling operations, two fully connected layers, and the output layer to predict the ten classes of the full CIFAR10 dataset. All weights except those mapping from the last hidden layer to the output are initialized with the pre-trained weights from different source tasks. The fine-tuning on the target task is then performed using a varying number of training samples. Besides a pre-training on the target task itself ("CLF labels" in Fig. 6.5), several unsupervised and semi-supervised pre-training scenarios are explored with auto-encoder and SimEc architectures. For the auto-encoder pre-training, two architectures are tested, the first one is a regular auto-encoder, where the full CNN is mirrored after last hidden layer to map back to the original input. The second architecture, which we call a linear auto-encoder, consist of only a single linear layer after the last hidden layer of the CNN to reconstruct the input. This linear auto-encoder has approximately the same number of parameters as the SimEc architecture, which also consists of only a single additional linear layer after the CNN to map to the 3000 target similarities. For the SimEc setup, different target similarities are tested: in the unsupervised learning case, the linear kernel, computed using only the raw images, is used, while in the (semi-)supervised setups the target similarities consist of a weighted average between the linear kernel and the class-based similarity, which is 1 for two images of the same class and 0 otherwise.

fine-tuning/training on the target task is performed for 50 epochs as well, using 5k training examples as a validation set to determine the best model used for testing.

The test accuracy achieved with different pre-training setups on the CIFAR10 dataset is shown in Fig. 6.5. As expected, pre-training on the target task itself ("CLF labels") yields the biggest performance gain for smaller training sets, as the CNN was able to learn very helpful representations during the pre-training. The unsupervised pre-training with both auto-encoder (AE) architectures as well as a SimEc with the linear kernel as the target similarity does not improve the performance on the target task. This is most likely due to the relatively small network, which seems to lack the capacity to represent both features useful for the reconstruction of the samples as well as for the classification task [53]. Interestingly, the performance of the linear auto-encoder is much worse than that of the unsupervised SimEc pre-training, which performs at the level of the regular auto-encoder, even though the linear AE and SimEc architecture have approximately the same number of parameters. When trained with a (semi-)supervised target similarity, a SimEc pre-training improves the performance on the target task and can even yield a better performance for

larger training sets than a pre-training on the target task itself. This means that even if one has a large training set available for a target task, it can still be beneficial to pre-train the network on class-label-based similarities to achieve an additional generalization boost [211], thereby establishing the benefit of a SimEc pre-training.

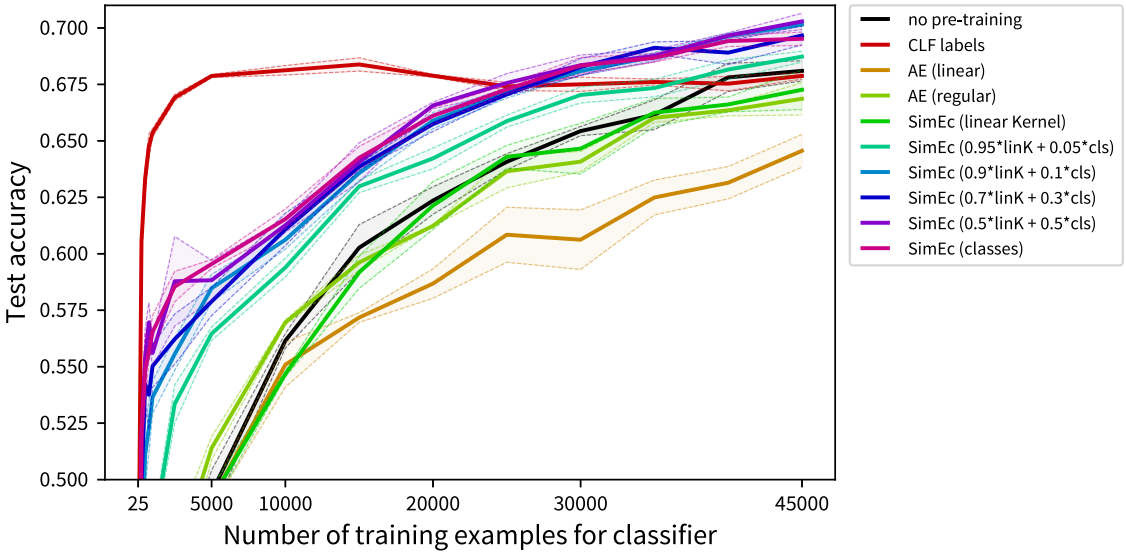


Figure 6.5: Results on the image classification task (CIFAR10) with different pre-training setups. The fine-tuning/training on the target task with a varying number of training samples was done using three different random seeds; the bold lines show the mean test accuracy while the shaded areas display the standard deviation.

The success of a SimEc pre-training hinges on the availability of a target similarity matrix that relates to the following supervised learning task. While other unsupervised similarity functions, e.g., the Wasserstein/Earth Mover's Distance between the color-histograms of the images [166], might be more auspicious for this image classification task than the linear kernel, these are still not flexible enough to truly capture semantic similarity in images. This is why we had created the artificial target similarity function computed as a weighted average between the linear kernel and the class-based similarities. As we will see in the next section, for other tasks, better unsupervised similarity functions are readily available and positively influence the impact of the SimEc pre-training.

6.2 Pre-training for the prediction of chemical properties

In this section, we demonstrate how a SimEc pre-training can improve the prediction of chemical properties of molecules. Unlike the image classification problem discussed in the previous section, this is a regression task. While transfer learning for the prediction of chemical properties was previously explored by pre-training on a larger, but lower quality dataset for the same task [180], we are interested in how transferable parameters are across different tasks, especially when the network was pre-trained on an unsupervised learning task, i.e., one that did not require the computation of costly labels for each molecule.

Predicting properties of molecules, such as their energy level, can, for example, help with the development of new materials [65]. While there exist exact methods to calculate these properties [44], these are computationally very expensive and take a long time, which is why machine learning models are increasingly employed to predict these properties instead

[16, 26, 51, 63, 142]. While a neural network might take several hours or days to train, computing predictions for new molecules is very fast.

A state-of-the-art model for the prediction of molecular properties is the SchNet neural network architecture [176, 177, 178]. One of the difficulties of this prediction task is that it is non-trivial to construct an informative input feature vector for a molecule, as all molecules consist of different numbers and types of atoms. The raw data available for each molecule includes the atoms it consists of and their three-dimensional coordinates, from which, e.g., the bonds between different atoms in the molecule can be derived. Instead of explicitly constructing feature vectors from these atom coordinates that could then be used as input to a prediction model [55, 91, 133], SchNet uses these raw atom coordinates directly to *learn* an informative representation for each molecule, from which its properties can then be predicted (Fig. 6.6). SchNet does this by learning an embedding for each kind of atom and then using the coordinates of the atoms in a molecule to iteratively compute interactions between the different atom embeddings. This yields a set of unique atom embeddings for each molecule. These are then used as input to a feed-forward neural network (FFNN), which is applied to each of the atom embeddings to compute the atom specific contribution to the target property. Depending on the type of property (extensive or intensive)[2] the individual atom contributions are then summed up or averaged to yield the final prediction. We use the SchNet implementation provided in the `schnetpack` Python library for our experiments.[3]

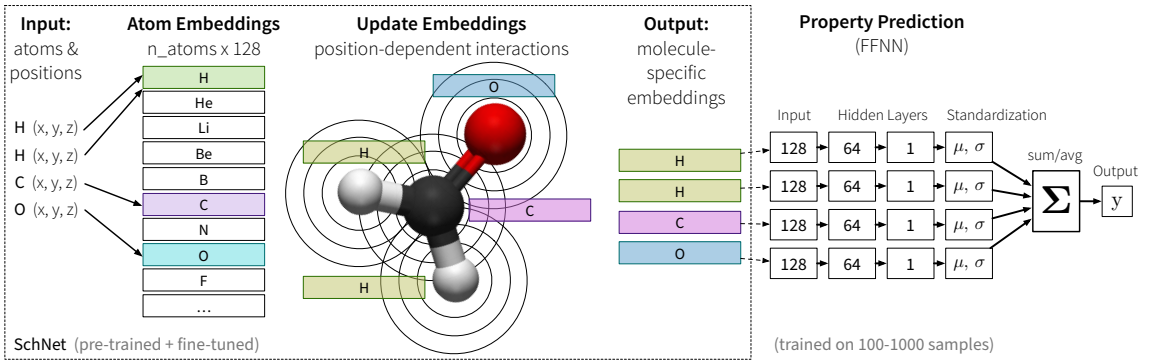


Figure 6.6: Molecular property prediction with SchNet.

The following experiments are performed on the QM9 dataset [155, 167], which consists of over 130k molecules made up of up to nine heavy atoms. For each molecule, in addition to its atoms and their coordinates, the dataset includes several chemical properties that were computed for it. Exemplary, we show the results for one extensive (energy U0) and two intensive (HOMO and LUMO) properties, however, as many of these properties are related (Fig. 6.7), results for other properties should be similar. For most of the experiments, only the number of molecules used for training the models are reported; unless stated otherwise, the validation set used for early stopping (usually after around 700-900 epochs) consists of 10k molecules, while all remaining samples of the 130k molecule dataset were used for testing.

[2]The characterization of a chemical property as extensive or intensive is made based on how the property changes with the size of the system: extensive properties, such as the mass, are additive for subsystems and therefore dependent on the size of a molecule, while intensive properties, e.g. the density, are not dependent on the system's size.

[3]`https://github.com/atomistic-machine-learning/schnetpack`

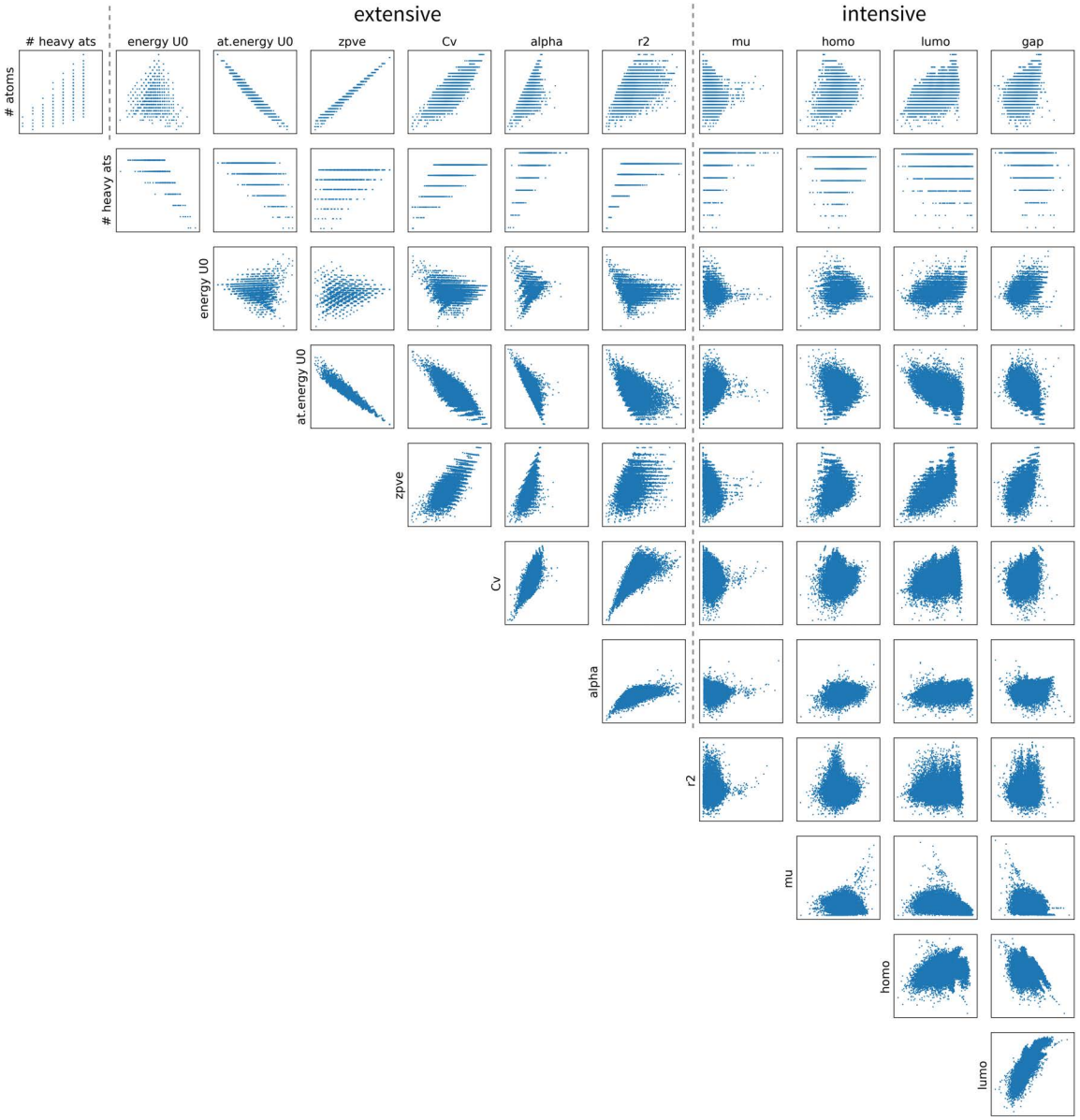


Figure 6.7: QM9: Overview of different molecular properties and how they are related.

In the following sections, we first establish a baseline for the effectiveness of pre-training in the molecular property prediction task by pre-training SchNet on the target property, as well as related properties as source tasks. However, in these setups, the pre-training is relying on the same costly labels as the regular training of the model for the target task. In a next step, different molecular similarity functions, such as fingerprint similarities and the FCHL kernel, are explored to determine their potential for a SimEc pre-training of SchNet. These similarities are unsupervised in the sense that they are computed from only the structural information associated with the molecules. Finally, the SimEc pre-training of SchNet with these similarities is evaluated and we present possible extensions of the model to further improve the performance.

Pre-training with properties

To establish a baseline for the benefits of pre-training SchNet, we explore in how far the prediction of the properties with the SchNet model can be improved when pre-training SchNet either on the target task itself or on a related property. For this, the full SchNet & feed-forward network architecture is trained on 100k molecules to predict each of the three selected properties energy U0, HOMO, and LUMO (= source tasks). Then, the atom embeddings and weights of a new SchNet network are initialized with the pre-trained network of the respective source task and the whole network is fine-tuned (SchNet) / trained (FFNN) on the target task using only 100-1000 training samples.

When using as a source task the same property as in the target task (dark blue lines in Figs. 6.8 & 6.9), the error on the training set is reduced significantly, while the performance on the test set for most properties is only markedly improved when more than 500 training samples are available for the fine-tuning. This suggests that a pre-training on the target property itself leads to overfitting of the network for very small training set sizes.

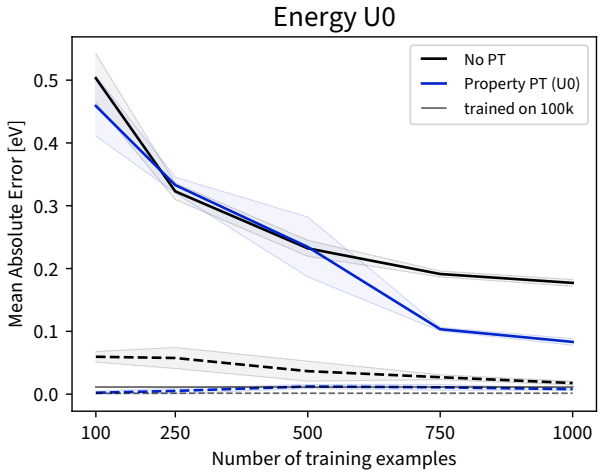


Figure 6.8: Mean absolute errors of energy U0 predictions on the training (dashed lines) and test sets (solid lines) with and without property pre-training (PT). The light gray lines show the train and test errors when SchNet is trained on the target task with a training set consisting of 100k molecules, which also acts as the source task for the property PT. The shaded areas show the standard deviation, computed by fine-tuning the full SchNet architecture with three different random seeds.

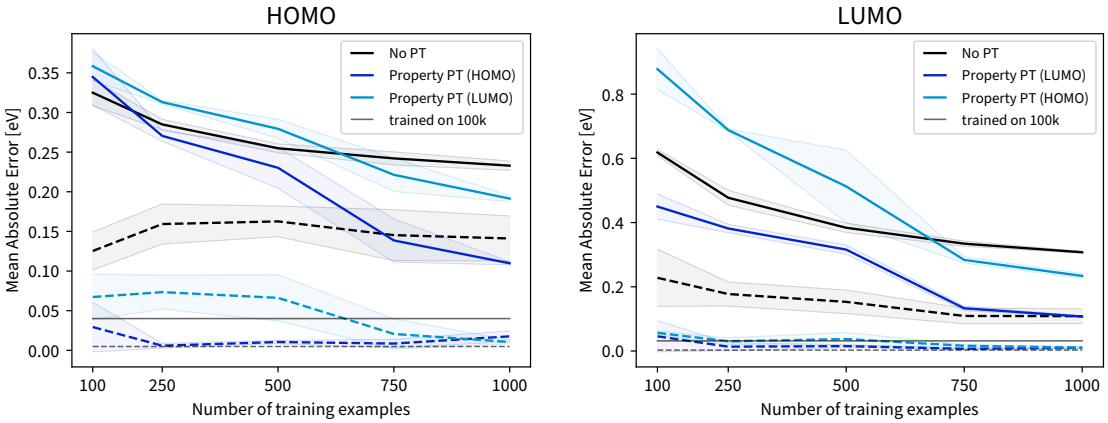


Figure 6.9: Mean absolute errors of HOMO and LUMO predictions on the training (dashed lines) and test sets (solid lines) with and without property pre-training (PT). The light gray lines show the train and test errors when SchNet is trained on the target task with a training set consisting of 100k molecules, which also acts as the source task for the property PT. The shaded areas show the standard deviation, computed by fine-tuning the full SchNet architecture with three different random seeds.

When using a different (but conceptually related) property as a source task (HOMO and LUMO respectively; light blue lines in Fig. 6.9), while the performance on the training set is still better than without using pre-training, the test error is actually worse for smaller training set sizes. Based on this poor generalization performance in this transfer learning setup, a pre-training on different properties does not seem worthwhile if only very few training examples are available for fine-tuning the network on the target task.

Similarities between molecules

While the pre-training with properties yields an improved performance at least for larger training sets, it of course still requires the expensive computation of large amounts of reference data. Ideally, we would like to utilize the structural information of the molecules for a pre-training instead. This information is captured by a variety of similarity measures, which can be used as targets in a SimEc pre-training of SchNet.

One family of computationally cheap unsupervised similarity functions for molecules are the so-called fingerprint similarities, which are computed based on the structure of the molecules alone (Fig. 6.10). First, topological fingerprints are computed for all molecules, which are binary vectors where individual bits are set to 1 based on the hash value of topological paths of varying lengths along the bonds of the molecule [108, 161]. From these vectors, the similarity between two molecules is then computed using different similarity coefficients [92, 156, 192]. An example of such a similarity coefficient is the Rogot-Goldberg similarity [162], which is defined as

$$
\begin{aligned}
S_{ij} &= \frac{a}{2a+b+c} + \frac{d}{2d+b+c} \\
\text{with: } a &: \text{number of attributes both } i \text{ and } j \text{ share} \\
b &: \text{number of attributes } i \text{ has and } j \text{ lacks} \\
c &: \text{number of attributes } i \text{ lacks and } j \text{ has} \\
d &: \text{number of attributes both } i \text{ and } j \text{ lack.}
\end{aligned}
\tag{6.1}
$$

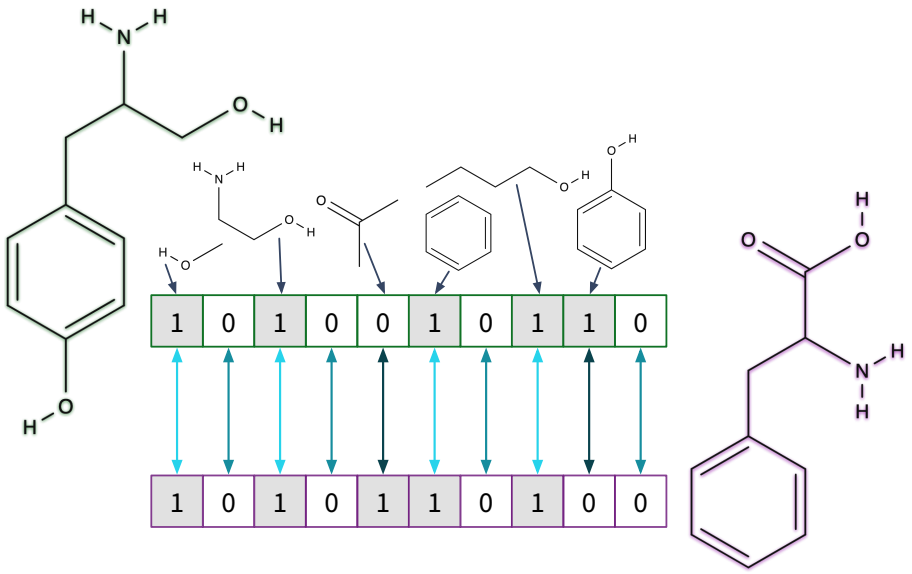


Figure 6.10: Binary topological fingerprints of two molecules are computed based on the substructures occurring in the molecules. The similarity between two molecules is then computed by checking which substructures occur in both or only one of the molecules.

We computed the fingerprint similarities between the molecules using the RDKit Python library.[4] It implements a multitude of different similarity coefficients, many of which are highly correlated (Fig. 6.11).

To limit the computational cost, we picked only three of these similarity coefficients for the following experiments: in addition to the above mentioned Rogot-Goldberg similarity these are the McConnaughey similarity [122] (which is strongly correlated with the Kulczynski similarity [101]), defined as

$$S_{ij} = \frac{a^2 - bc}{(a+b)(a+c)}$$

and the Sokal similarity [182], defined as

$$S_{ij} = \frac{a}{a + 2b + 2c}.$$

As these three similarity coefficients are highly correlated with the remaining similarity coefficients and therefore provide a good coverage and variety, we conclude that results with other fingerprint similarities should be similar.

In addition to the fingerprint similarities, we also use the FCHL kernel [38, 54], implemented in the QML Python library [39], to compute molecular similarities. This kernel function is computed as the sum of all atom-to-atom similarities of two molecules, which in turn are computed from atom representations based on an atom's period and group in the periodic table as well as the interactions between its neighboring atoms. By using the period and group of the atoms to create their representations, kernel ridge regression models based on the FCHL kernel are capable of generating predictions even for molecules containing elements not seen in the training dataset, whereas SchNet is only able to learn representations for atoms encountered during training. Due to the additive nature of the FCHL kernel, we

[4] `http://www.rdkit.org`

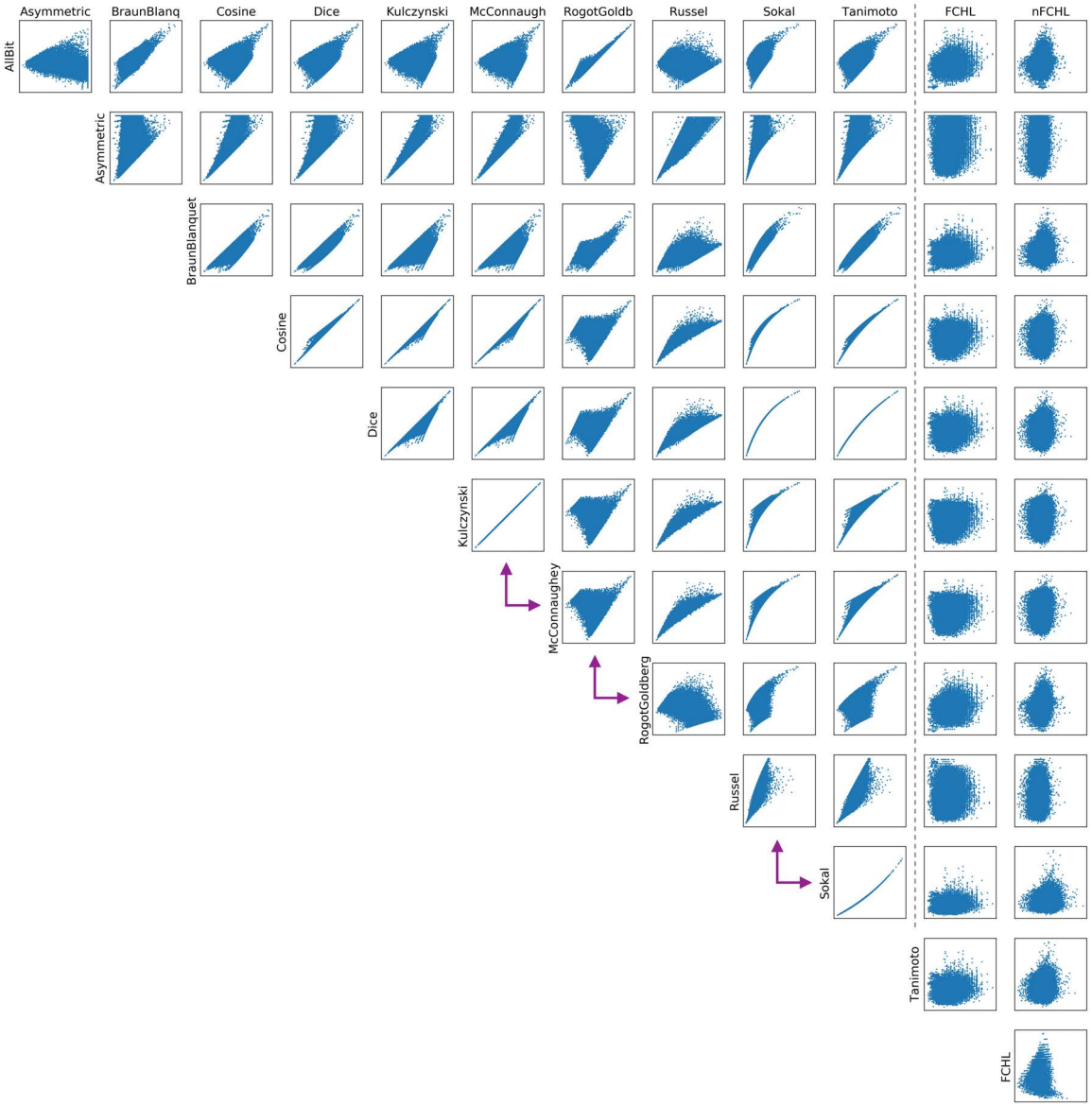


Figure 6.11: Overview of different similarity functions and how they are related. The purple arrows indicate the fingerprint similarities used for further experiments. The similarities on the left of the dashed line are computed using topological fingerprints of molecules (as implemented in the RDKit Python library), while the two right most similarities are computed using the FCHL kernel (from the QML Python library).

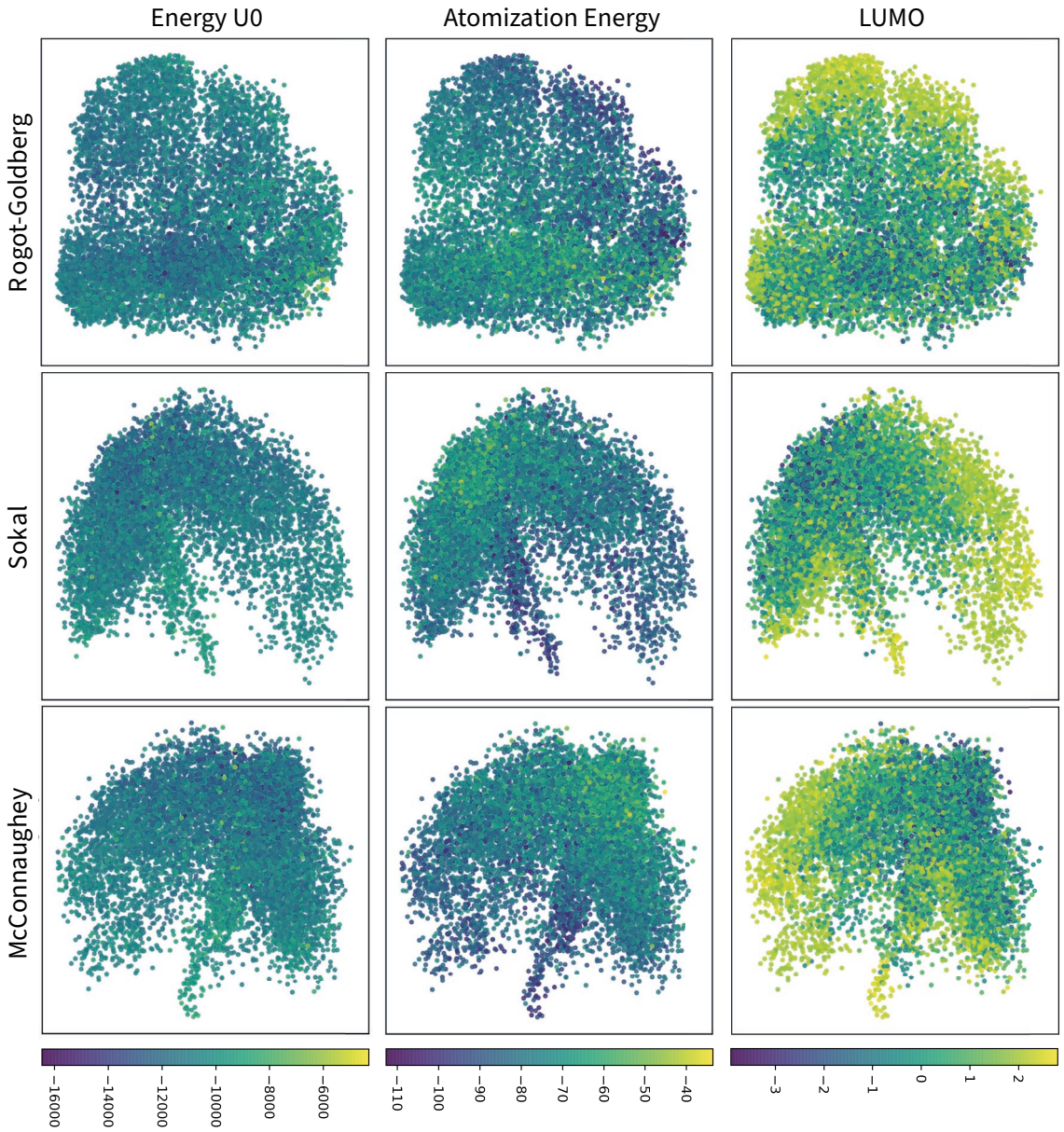


Figure 6.12: First two kernel PCA components computed on the similarity matrices of the fingerprint similarities. Each dot represents one molecule, colored according to the respective property.

additionally report results on a normalized version of it (nFCHL), computed by dividing the FCHL similarity S_{ij} of two molecules i and j by $n_i \cdot n_j$, the product of the number of atoms in molecule i and j respectively.

To get a first impression of the five similarity function used in the following SimEc pre-training, Figs. 6.12 & 6.13 show scatter plots of the respective first two kernel PCA (kPCA) components, colored by different molecular properties.[5] For computational reasons, the scatter plots for the (n)FCHL kernel were computed using only 1k randomly selected molecules, while for the fingerprint similarities 10k samples were used. Both the FCHL and

[5]The atomization energy of a molecule is computed by subtracting from the energy U0 the individual atom reference energies of all the atoms in the molecule.

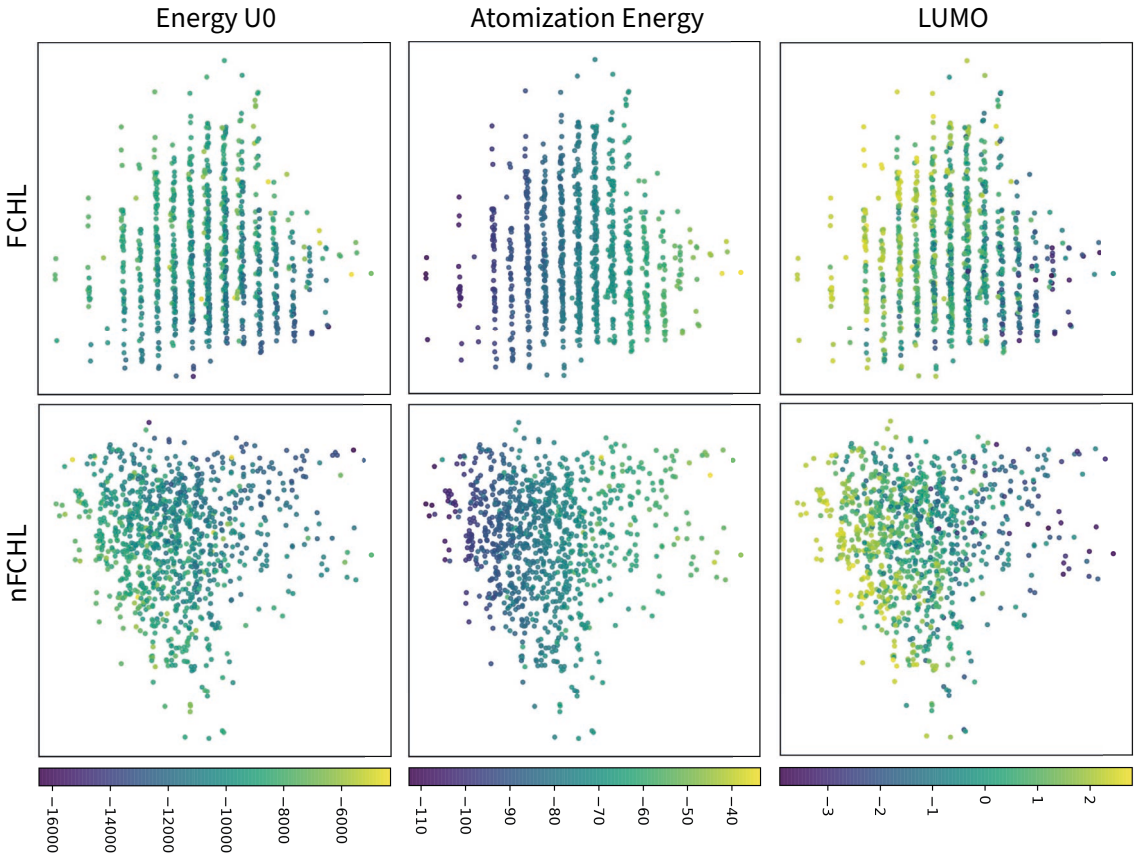


Figure 6.13: First two kernel PCA components computed on the similarity matrices of the FCHL kernel functions. Each dot represents one molecule, colored according to the respective property.

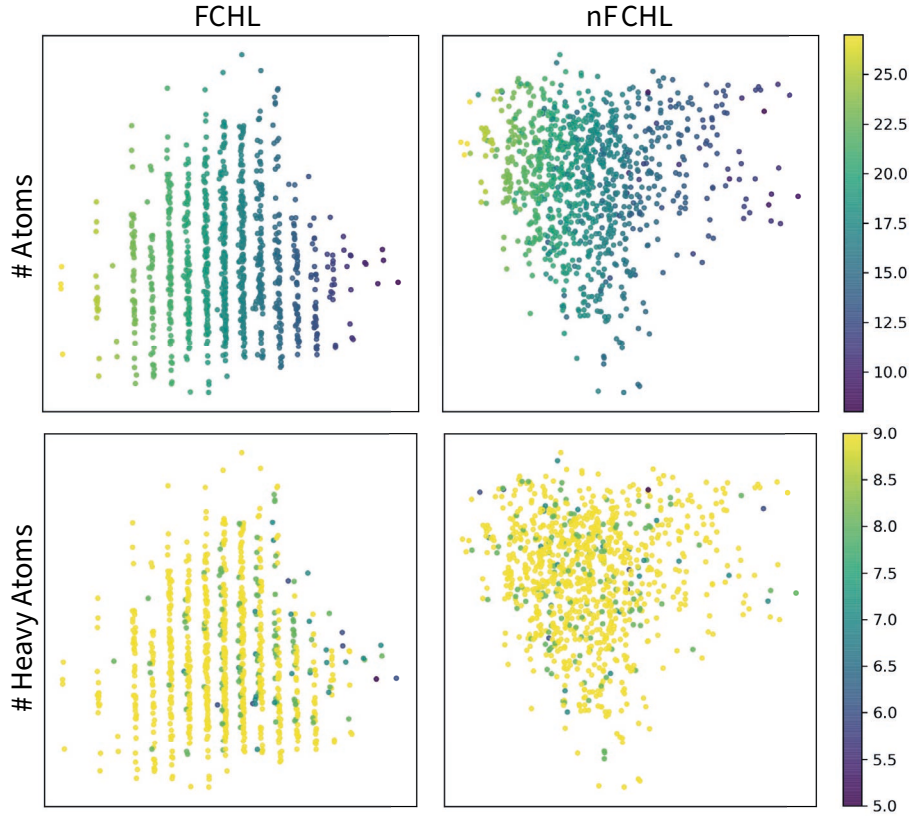


Figure 6.14: First two kernel PCA components computed on the FCHL and nFCHL kernel matrices, colored by the number of (heavy) atoms.

nFCHL similarities are strongly related to the atomization energy. The prominent stripe structure in the first kernel principal component of FCHL is due to the number of atoms in the molecules (Fig. 6.14), since the FCHL kernel is computed as the sum of the atomwise similarities; the influence of which is reduced in the nFCHL similarity.

As another indication of how promising a SimEc pre-training with the respective similarities might be, we examined the prediction errors achieved when using the 128-dimensional kPCA embeddings of the respective similarity matrices together with a linear ridge regression model to predict the properties (Figs. 6.15 & 6.16). The kPCA was computed based on a 1000×1000 kernel matrix and then applied to the kernel map of all 130k molecules to create the embeddings. The linear regression model was then fitted with a variable number of training examples, ranging from 100 to 10k molecules, and using the kPCA embeddings as input and one of the molecules' properties as the target. To improve the predictions of the fingerprint and nFCHL similarities, especially for the atomization energy, the number of atoms in each molecule was used as an additional input feature (compare Fig. 6.15 left and right panels). The reported prediction errors were computed on 100k test molecules, where training and test splits are based on three different random seeds. The regularization parameter of the ridge regression model was set automatically in the internal cross-validation loop of the model.

While the molecular property predictions with the kPCA embeddings computed from the fingerprint similarity matrices are quite poor, embeddings computed from the FCHL kernel are very informative features and, together with a simple linear regression model, yield more accurate predictions, at least for the atomization energy, than a SchNet model

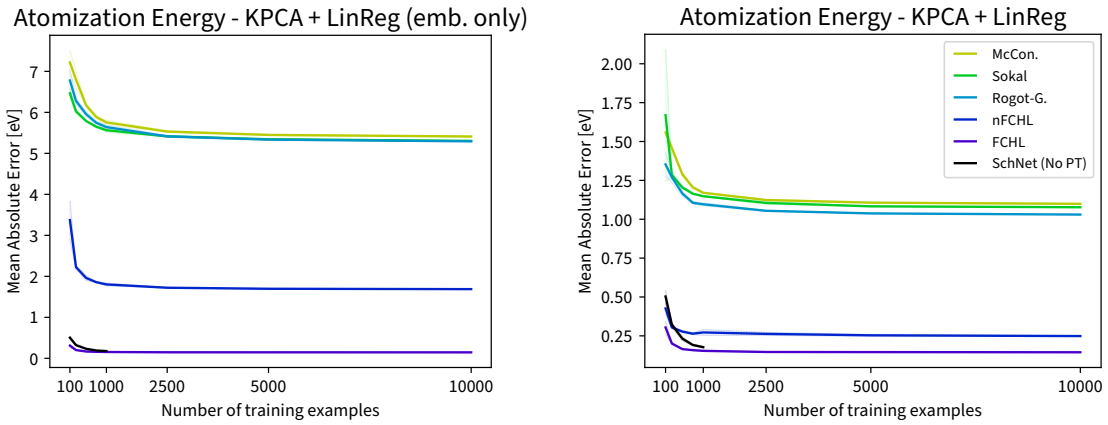


Figure 6.15: Mean absolute errors of atomization energy predictions on the test dataset with a linear regression model trained on kPCA embeddings *(left)* or on kPCA embeddings and the number of atoms of each molecule as an additional input feature *(right)*. The shaded areas show the standard deviation, computed by training and evaluating the linear regression model on three different random train/test splits.

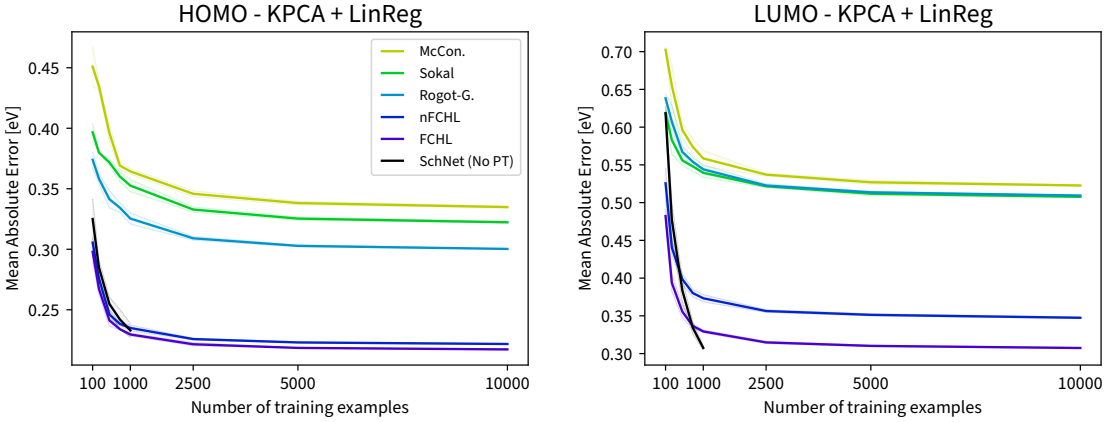


Figure 6.16: Mean absolute errors of HOMO and LUMO predictions on the test dataset with a linear regression model trained on kPCA embeddings and the number of atoms of each molecule as features. The shaded areas show the standard deviation, computed by training and evaluating the linear regression model on three different random train/test splits.

trained on a very small training set.

Pre-training with SimEcs

For the SimEc pre-training of SchNet (Fig. 6.17), the individual SchNet atom embeddings of each molecule are first added or averaged to create a single 128-dimensional embedding vector for the molecule. This embedding is then multiplied by the linear SimEc last layer to predict the target similarities. To limit the computational cost, only 1000 molecules were randomly chosen as targets, but all 130k molecules of the QM9 dataset are used as inputs to train the network, i.e., a 130k × 1k target similarity matrix had to be computed for each of the five similarity functions to conduct the SimEc pre-training.

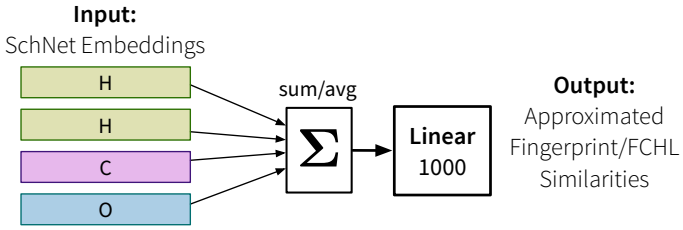


Figure 6.17: SimEc pre-training of SchNet.

For the SimEc pre-training, the target similarity matrix is first normalized, either by standardizing it (subtracting the overall mean from all values and dividing them by the standard deviation; labeled 'std' in the plots), or by scaling the similarities to be between 0 and 1 (min/max scaling; labeled 'max'). Together with the aggregation strategy (sum/avg) for creating the molecules' embeddings from their atom embeddings, this results in four hyperparameter combinations that were tested for the pre-training (in addition to the selection of a suitable learning rate) for each of the five similarity functions. The network was trained for up to 100 epochs on all 130k molecules and usually converged after about 10-30 epochs. While the SimEc pre-training with the three fingerprint similarities was fairly robust with a good convergence for almost all aggregation and normalization combinations and across a relatively broad range of learning rates, training the SimEc architecture with the (n)FCHL kernel proved to be a bit more difficult and only the min/max normalization yielded acceptable results, possibly due to the additive nature of the kernel function.

As a first check to ensure the SimEc pre-training itself was successful, the resulting molecule embeddings, just like the kPCA embeddings in the previous section, were used together with a linear ridge regression model to predict the molecular properties (Figs. 6.18 & 6.19). Noticeably, the prediction errors with the SimEc embeddings based on the fingerprint similarities are lower than with the corresponding kPCA embeddings. This might in part be due to the significant negative eigenvalues of these similarity matrices, which are not captured by the kPCA components but can be learned by a SimEc (compare Sec. 3.6), as well as the additional domain knowledge inherent in the SchNet model used to construct the SimEc embeddings. The results with the (n)FCHL SimEc embeddings, on the other hand, are worse than with the corresponding kPCA embeddings, which indicates that the SchNet network might not be able to fully capture all the details that went into the FCHL

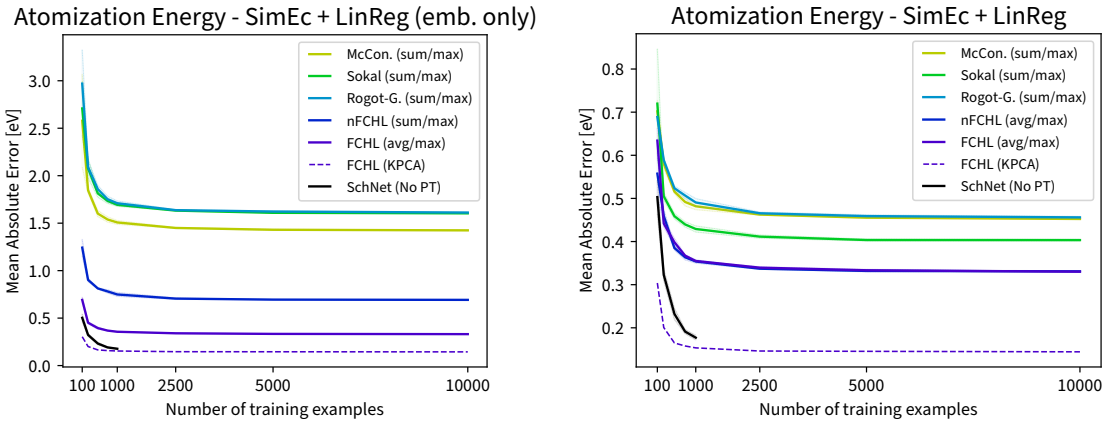


Figure 6.18: Mean absolute errors of atomization energy predictions on the test dataset with a linear regression model trained on SimEc embeddings *(left)* or on SimEc embeddings and the number of atoms of each molecule as an additional input feature *(right)*. For each similarity, only the results of the best aggregation (avg or sum) and normalization (std or max) hyperparameter combination are shown. The shaded areas show the standard deviation, computed by training and evaluating the linear regression model on three different random train/test splits.

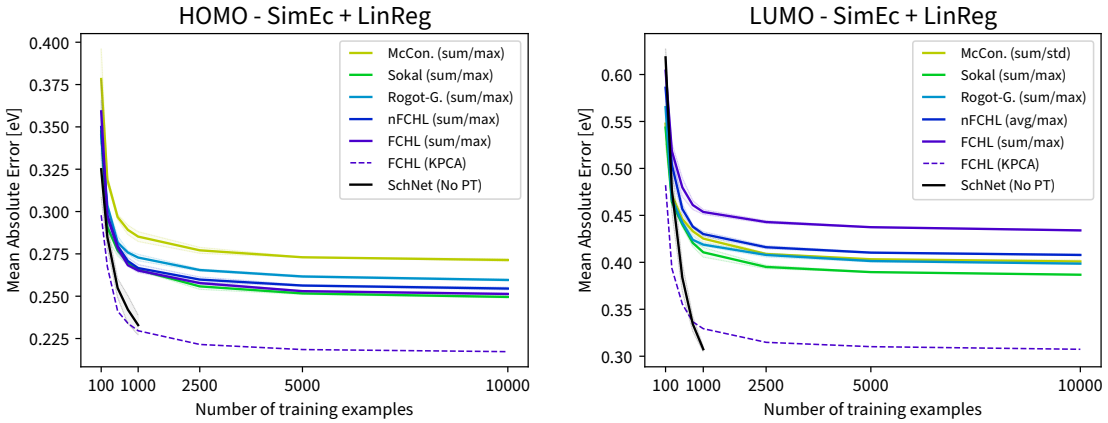


Figure 6.19: Mean absolute errors of HOMO and LUMO predictions on the test dataset with a linear regression model trained on SimEc embeddings and the number of atoms of each molecule as features. For each similarity, only the results of the best aggregation (avg or sum) and normalization (std or max) hyperparameter combination are shown. The shaded areas show the standard deviation, computed by training and evaluating the linear regression model on three different random train/test splits.

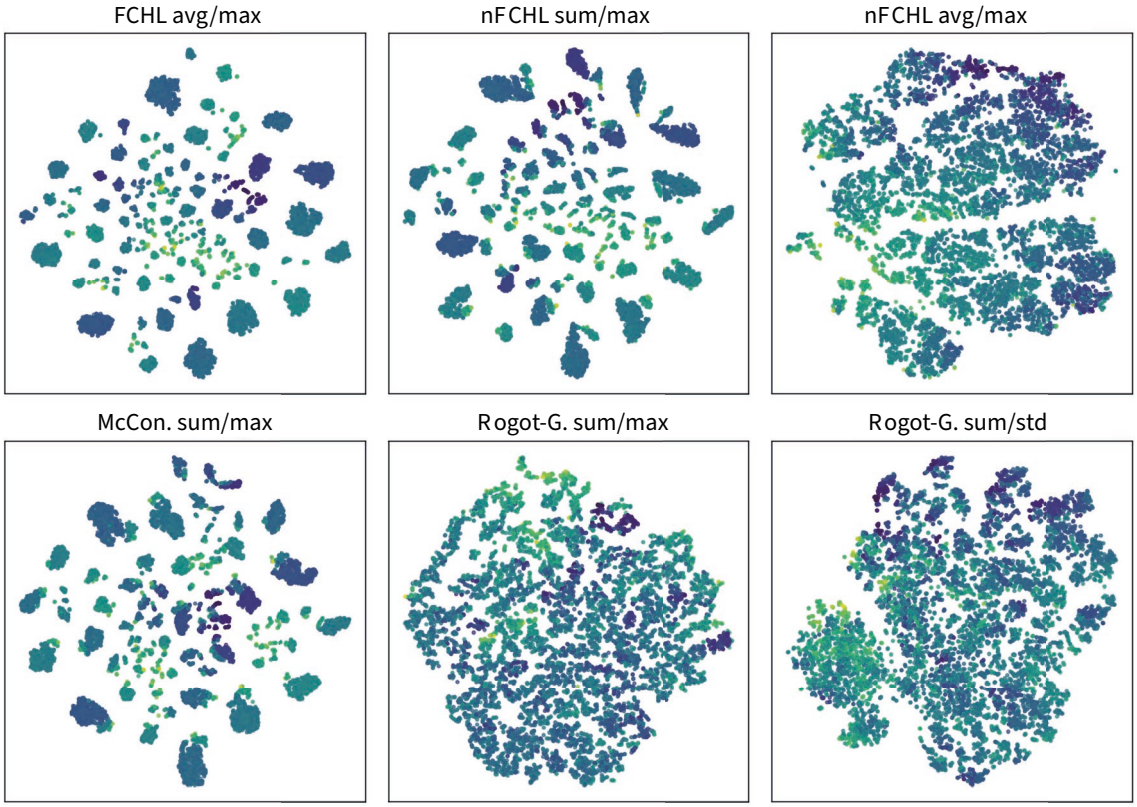


Figure 6.20: T-SNE visualizations of 10k molecules' 128-dimensional SimEc embeddings, where the SimEcs were trained with different target similarities and aggregation/normalization settings. Each dot represents one molecule, colored by its atomization energy. From left to right and top to bottom the plots are sorted from best to worst performance when predicting the molecules' atomization energy with a linear regression model from the embeddings. The two-dimensional visualizations were created with identical t-SNE hyperparameters.

kernel computation.[6]

Not only the target similarity, that the SimEc was trained with, has an influence on the accuracy when using the SimEc embeddings to predict the molecules' properties with a linear regression model (e.g. for the atomization energy, (n)FCHL yields much better results than the fingerprint similarities). Within the same similarity function class, the results also vary based on the different aggregation and normalization settings. Two-dimensional t-SNE [119] visualizations of the different SimEc embeddings (Fig. 6.20) shed light on these discrepancies: Those embeddings that result in low prediction errors (e.g. FCHL avg/max) form tight clusters in the t-SNE visualizations, which suggests that the molecules'

[6]This finding is corroborated by the fact that the embedding error, i.e., the mean squared error between the product of the SimEc embeddings and the target similarity matrix, is much higher than the embedding error of the kPCA embeddings for the (n)FCHL kernel. A similar behavior of SimEcs was observed in other experiments as well, where, for example, for a class-based similarity matrix a fairly deep net was needed to achieve a SimEc embedding error comparable to that of an eigendecomposition (Fig. 3.4). For the fingerprint similarities, on the other hand, the SimEc embedding errors were fairly low, indicating that the decomposition of these similarity matrices could be learned quite well with a SchNet-SimEc architecture of the given complexity. However, as the ultimate goal, i.e., the target task in this transfer learning setting, is the prediction of the molecular properties, not the decomposition of the FCHL kernel matrix, the SchNet architecture is chosen according to this and the SimEc results just are what they are.

representations include features unique to each of these groups. In contrast, embeddings less suited for a prediction with a linear regression model (e.g. Rogot-Goldberg sum/std) resulted in t-SNE plots with a more global structure, indicating that the representations of different molecules have more in common. While a linear regression model benefits from embeddings containing very distinctive features, as we will show below, when it comes to using the corresponding SimEc weights to initialize SchNet, the resulting performances are better for the more inclusive representations.

Overall, within the constraints of the chosen SchNet architecture, the SimEc pre-training seems to have been successful. Although the SimEc embeddings themselves can already be fairly informative, a linear regression model is not sufficient to generate accurate predictions from them and the performance of the regular SchNet prediction model (black lines in Figs. 6.18 & 6.19), which includes a multi-layer feed-forward NN to generate the property predictions from the SchNet embeddings, is still superior.

Finally, just like for the property pre-training, the SchNet model is initialized with the pre-trained weights from the SimEc training and the full SchNet+FFNN model is fine-tuned on a small number of training examples to predict the three properties (= target task). With the SimEc pre-training of SchNet, the test errors improve and, unlike with the property pre-training, the training errors are almost unchanged, i.e., the model generalizes better (Figs. 6.21 & 6.22; Table 6.1). While a pre-training is generally most helpful when there is only a small number of training samples available for the target task, similar as in the image classification experiment in the previous section, an additional generalization boost, although small, can still be observed when SchNet is trained on a larger dataset (Fig. 6.23).

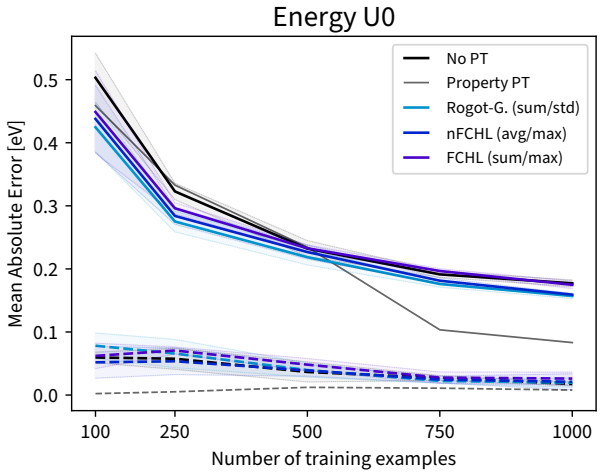


Figure 6.21: Mean absolute errors of energy U0 predictions on the training (dashed lines) and test sets (solid lines) with and without a SimEc pre-training (PT). For each similarity, only the results of the best aggregation (avg or sum) and normalization (std or max) hyperparameter combination are shown. The light gray lines show the mean train and test errors when SchNet is pre-trained on the target task itself. The shaded areas show the standard deviation, computed by fine-tuning the full SchNet architecture with three different random seeds.

Only the results for the Rogot-Goldberg fingerprint similarity are shown in the plots; for HOMO and LUMO this was the best performing similarity, while for energy U0 all three fingerprint similarities (with the sum/std setting) performed equally well. Interestingly,

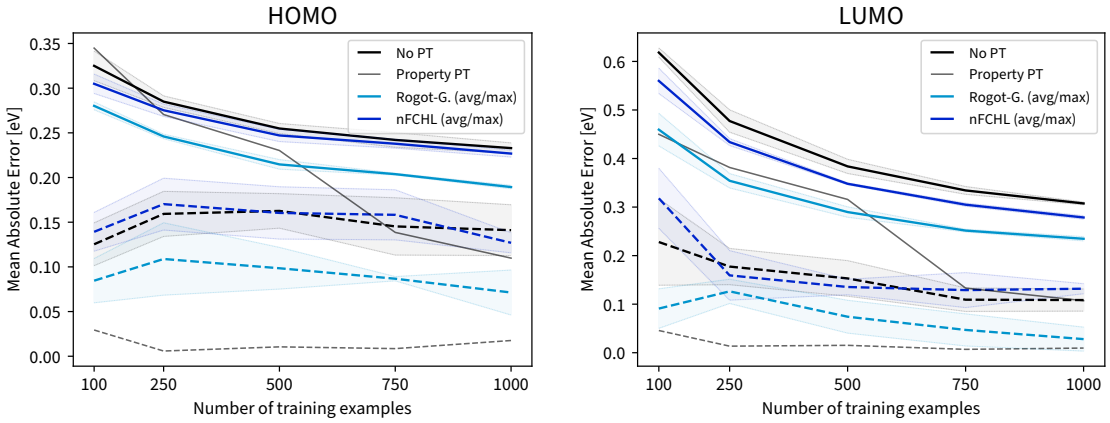


Figure 6.22: Mean absolute errors of HOMO and LUMO predictions on the training (dashed lines) and test sets (solid lines) with and without a SimEc pre-training (PT). For each similarity, only the results of the best aggregation (avg or sum) and normalization (std or max) hyperparameter combination are shown. The light gray lines show the mean train and test errors when SchNet is pre-trained on the target task itself. The shaded areas show the standard deviation, computed by fine-tuning the full SchNet architecture with three different random seeds.

Table 6.1: Prediction results for (PG-)SchNet models with different pre-trainings as well as a linear regression (LR) model trained on FCHL kPCA embeddings. The models were trained on 100 or 500 molecules; average mean absolute errors ± standard deviation were computed by training/fine-tuning the models with three different random seeds.

	Energy U0		**HOMO**		**LUMO**	
	100	500	100	500	100	500
FCHL kPCA&LR	0.310±0.04	0.166±0.00	0.294±0.00	0.242±0.01	0.473±0.01	0.356±0.01
SchNet (no PT)	0.503±0.04	0.232±0.01	0.325±0.02	0.255±0.01	0.618±0.01	0.384±0.01
SchNet Prop.	0.459±0.05	0.234±0.05	0.345±0.04	0.230±0.03	0.450±0.04	0.316±0.01
SchNet Rogot-G.	0.425±0.04	0.219±0.01	0.280±0.01	0.215±0.01	0.459±0.03	0.290±0.01
SchNet nFCHL	0.438±0.05	0.227±0.01	0.305±0.01	0.247±0.01	0.560±0.03	0.348±0.00
PG-SchNet (no PT)	0.528±0.03	0.241±0.01	0.326±0.01	–	0.617±0.02	–
PG-SchNet FCHL	0.439±0.04	0.210±0.02	0.305±0.01	–	0.581±0.01	–

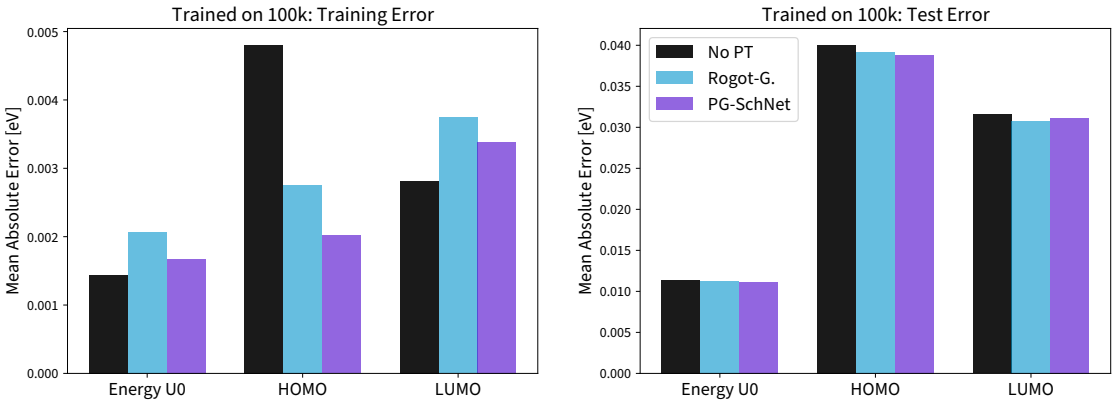


Figure 6.23: Mean absolute errors for all three properties on the training *(left)* and test sets *(right)* using SchNet without a SimEc pre-training *(black)*, SchNet pre-trained on a fingerprint similarity *(blue)*, and the PG-SchNet architecture without pre-training *(purple)*. The models were trained on 100k molecules with a single random seed. The SimEc pre-training was conducted with the Rogot-Goldberg similarity with the aggregation/normalization combinations sum/std for energy U0 and avg/max for HOMO and LUMO.

while the sum/max hyperparameter combination worked best for most similarities and properties when using the SimEc embeddings together with a linear regression model to predict the properties, here in the pre-training setup the results are very different and the best hyperparameter choices seem to be more dependent on the property type and corresponding aggregation in the SchNet FFNN architecture (sum for energy U0 and avg for HOMO and LUMO), as well as favoring pre-trainings with more "global" embeddings, as visible in the corresponding t-SNE plots (Figs. 6.24 and 6.25). While the t-SNE visualizations created from SchNet embeddings when the network was trained with 100k molecules display a global structure with large clusters (if any), the t-SNE plots obtained from embeddings where SchNet was trained on only 100 molecules show many small clusters, indicating that the network might have picked up on the distinguishing characteristics of individual molecules instead of learning patterns that hold true for a larger set of molecules (Fig. 6.24). This is especially striking for SchNet trained to predict LUMO: when the network is trained on 100k molecules, one large yellow blob emerges in the t-SNE plot, while when trained on only 100 molecules, the yellow dots are scattered across multiple smaller clusters. When SchNet is pre-trained with a SimEc, on the other hand, even when fine-tuned on only 100 molecules, more global patterns can be observed in the t-SNE plot. While a pre-training with a SimEc with "global" embeddings (nFCHL avg/max) also results in more global SchNet representations after fine-tuning, a pre-training with "local" embeddings (nFCHL sum/max) fails to create these global representations (Fig. 6.25) and correspondingly results in lower accuracy improvements.

Overall, a SimEc pre-training with one of the fingerprint similarities, especially the Rogot-Goldberg similarity, yields the biggest performance improvements, while a pre-training with the (n)FCHL kernel, despite its superior results in the kPCA + linear regression experiments, does not improve the performance of SchNet significantly. This was somewhat expected considering the inadequate performance of the corresponding SimEc embeddings together with a linear regression model discussed above.

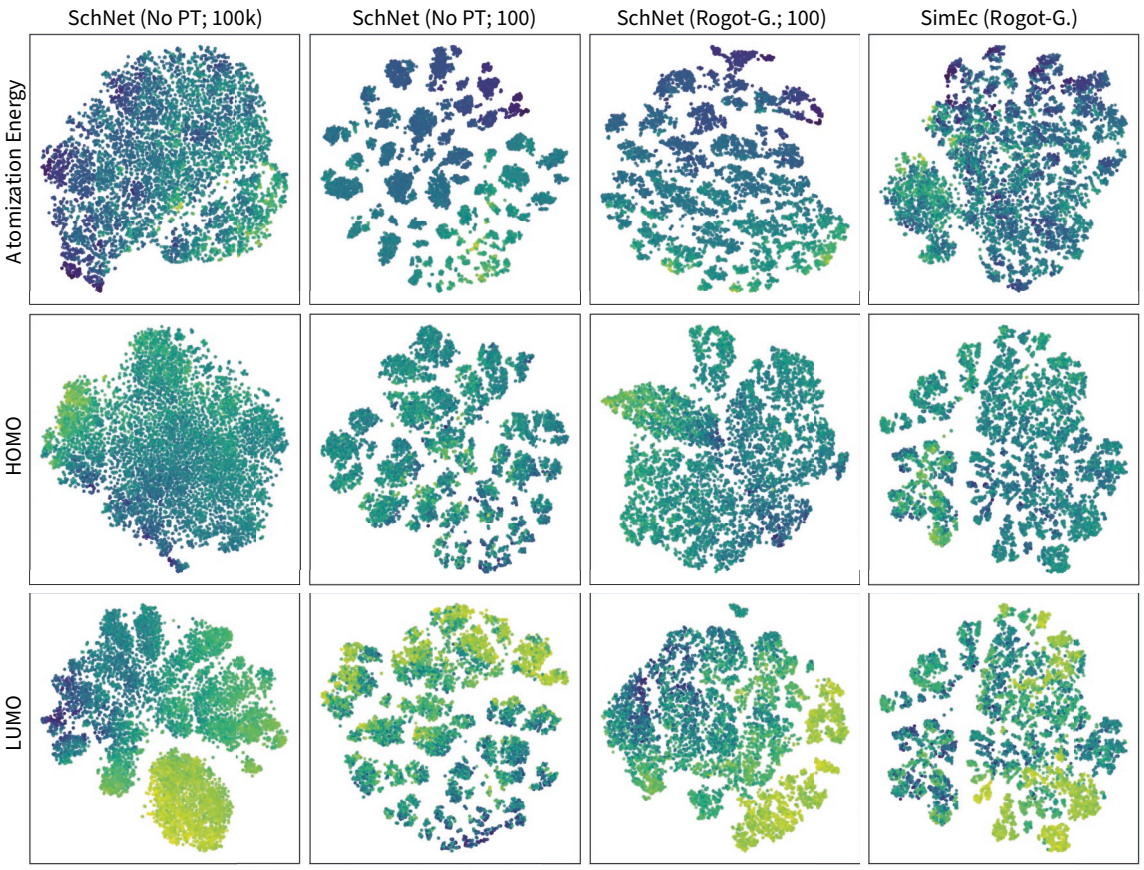


Figure 6.24: T-SNE visualizations of 10k molecules' 128-dimensional SchNet and SimEc embeddings. Each dot represents one molecule, colored by the property stated in each row, which is also the property that SchNet was trained/fine-tuned to predict. The SchNet molecule embeddings were computed by summing up the atom embeddings of each molecule (t-SNE visualizations created from averaged atom embeddings look structurally similar). From left to right, the t-SNE visualizations are created from embeddings from a SchNet network without pre-training (PT) trained with 100k molecules; SchNet without PT trained with 100 molecules; SchNet pre-trained with a SimEc and fine-tuned on 100 molecules; and the SimEc embeddings from the pre-training (with the SimEc trained to approximate the Rogot-Goldberg similarity, with aggregation/normalization sum/std for energy U0 and avg/max for HOMO & LUMO, i.e., the respective best pre-training settings).

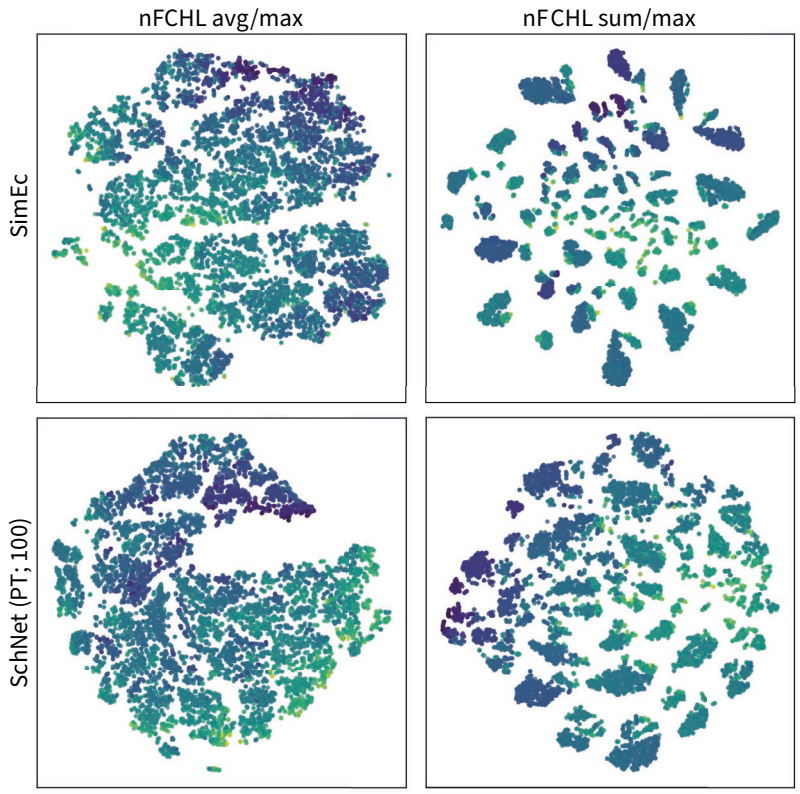


Figure 6.25: T-SNE visualizations of 10k molecules' 128-dimensional SimEc *(top)* and SchNet embeddings *(bottom)*. Each dot represents one molecule, colored by its atomization energy. The SchNet architecture was pre-trained with the SimEc from the plot above it and fine-tuned with 100 molecules to predict the molecules' energy U0.

Atomwise SimEc pre-training

In light of the comparatively poor performance of SchNet pre-trained with the (n)FCHL kernel, we devised the "atomwise" SimEc pre-training setup (Fig. 6.26) as an attempt to adapt the SimEc pre-training of SchNet to the additive nature of the FCHL kernel. Since the FCHL kernel is computed as the sum over the atomwise similarities of two molecules, the linear SimEc last layer is extended to reflect this: Instead of aggregating the atom embeddings of each molecule to create a single molecule embedding vector that is multiplied by the SimEc last layer, the atom embeddings of each input molecule are now multiplied with the SimEc last layer directly, where this last layer is extended to include n_i atom embeddings for each molecule i of the T target molecules. These individual atom-to-atom similarity matrices of each target molecule are then summed up to compute the predicted (n)FCHL similarity score for the respective molecule pair. This setup results in a vast increase in parameters, since instead of a $128 \times T$ weight matrix, the SimEc last layer now consists of $\sum_{i=1}^{T} 128 \cdot n_i$ parameters, where the T target molecules contain on average $n_i = 18$ atoms.

We only tested the atomwise SimEc pre-training for the prediction of the energy U0, as the (n)FCHL kernel is most strongly related to this property and correspondingly one would expect the largest improvements of a pre-training for this property. Both the FCHL kernel as well as its normalized version (nFCHL) were used as target similarities. Concerning the normalization of the similarity matrix, again the standardization (mean: 0, std: 1) of the matrix resulted in a very poor convergence during the SimEc training, while the min/max

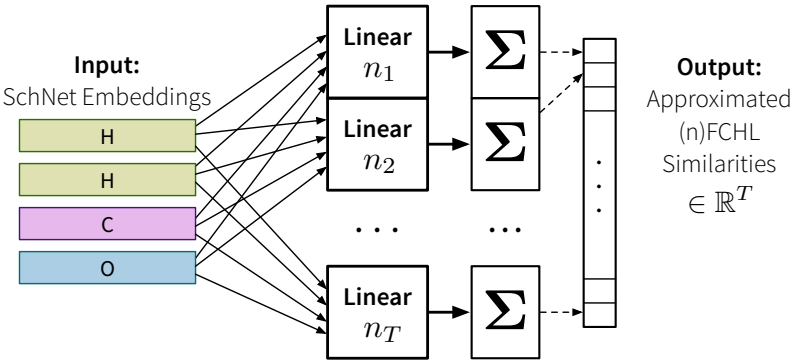


Figure 6.26: Atomwise SimEc pre-training of SchNet. The linear SimEc last layer contains n_i atom embeddings for each of the T target molecules. The input molecule's atom embeddings are multiplied directly with the atom embeddings of the T target molecules and these atom-to-atom similarity matrices are then summed up for each target molecule to yield the final similarity prediction.

scaling (min: 0, max: 1), as well as an additional std-scaling (min: 0, std: 1), worked quite well. Due to the vast increase in parameters in the SimEc last layer, we only tested this pre-training setup with $T = 50, 100,$ and 200 target molecules, compared to the 1000 targets used in the regular SimEc pre-training. With 50 target molecules, this results in approximately the same number of parameters in the SimEc last layer as before.

Unfortunately, this atomwise SimEc pre-training did not yield an improved performance compared to the regular SimEc pre-training with these similarity functions (on par for nFCHL, slightly worse for FCHL). Consequentially, the performance is still below that of the fingerprint similarities and still far from that of the FCHL kPCA embeddings together with a linear regression model. Apparently, this architectural change is still not enough to fully capture the intricacies of the FCHL kernel.

Period/Group-SchNet (PG-SchNet)

As a different attempt to improve the SimEc pre-training results with the (n)FCHL kernel, we adapted the original SchNet architecture to construct atom embeddings based on an element's period and group in the periodic table (Fig. 6.27). This period/group-SchNet (PG-SchNet) architecture is inspired by the fact that the computation of the FCHL kernel heavily relies on the atoms' periods and groups as well.

Except for the Lanthanides and Actinides (group 3 periods 6 & 7), every element is uniquely defined by its period and group in the periodic table, which therefore makes it possible to construct unique atom embeddings by concatenating the respective period and group embeddings. Since elements in the same period or group share certain characteristics, decomposing the atom embeddings this way means incorporating further domain knowledge into the SchNet architecture while reducing the number of embeddings that need to be learned, which also acts as a kind of regularization. Furthermore, this makes it possible to construct meaningful atom embeddings for elements not in the original dataset, provided other elements from the same period and group were encountered during training. Only the way the initial atom embeddings are constructed is changed in PG-SchNet, all other aspects of the prediction problem (Fig. 6.6) as well as the SimEc pre-training (Fig. 6.17) remain the same.

The prediction errors of PG-SchNet for energy U0 without pre-training, as well as with

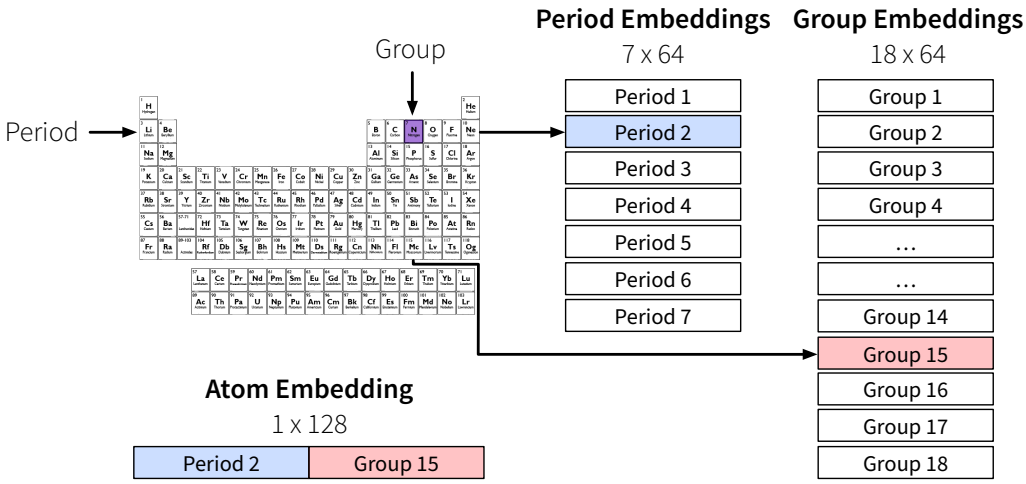


Figure 6.27: Period/Group-SchNet (PG-SchNet) atom embedding construction from the respective period and group embeddings.

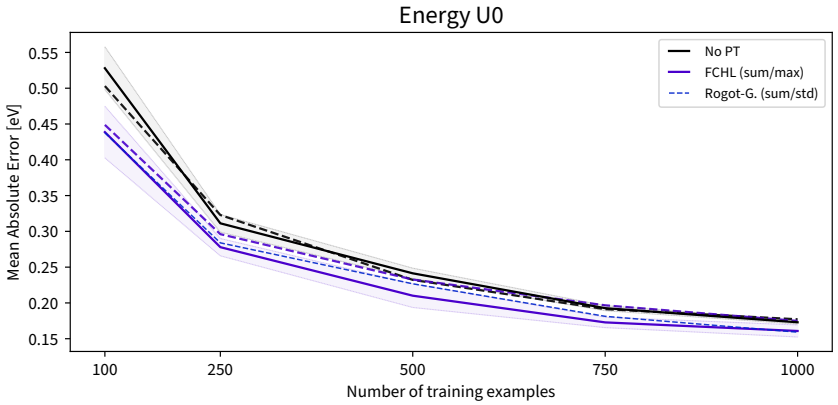


Figure 6.28: Mean absolute errors of energy U0 predictions on the test set with and without a SimEc pre-training (PT). The solid lines show the mean test errors of the PG-SchNet architecture, while dashed lines correspond to the respective errors with regular SchNet. The shaded areas show the standard deviation, computed by fine-tuning the full PG-SchNet architecture with three different random seeds.

a pre-training with the Rogot-Goldberg or nFCHL similarity, are on the same level as the respective results using the regular SchNet architecture. A pre-training of PG-SchNet with the FCHL kernel, however, leads to slightly better results, comparable to SchNet pre-trained on a fingerprint similarity (Fig. 6.28; Table 6.1). This indicates that the PG-SchNet architecture is better suited for learning the information captured by the FCHL kernel. For HOMO and LUMO, PG-SchNet pre-trained on the nFCHL kernel or the Rogot-Goldberg similarity (avg/max) yield slightly better results than the respective SchNet prediction errors, however, the improvement of the nFCHL pre-training still does not come close to the error achieved by pre-training the regular SchNet architecture with the Rogot-Goldberg similarity (which might be expected considering (n)FCHL's strong relation to the atomization energy). When trained on a larger dataset, the prediction with PG-SchNet is slightly better compared to the regular SchNet architecture (Fig. 6.23), although it should be noted that this was only tested with a single random seed.

The QM9 dataset used in these experiments only contains molecules made up of the atoms ‘H’ (Period 1, Group 1) and ‘C’, ‘N’, ‘O’, ‘F’ (Period 2, Groups 14-17). With this limited set of atoms, only the Period 2 embedding of PG-SchNet is learned across multiple atoms, whereas the embedding for ‘H’ is effectively learned in the same way as with the regular SchNet architecture. Nevertheless, even on this dataset minor performance improvements can be achieved using a PG-SchNet architecture, therefore it can be assumed that, especially together with a SimEc pre-training, PG-SchNet might be a worthwhile option when the training data contains a more diverse set of atoms and predictions need to be made for molecules with atoms not encountered during training.

7

Better Word Embeddings from Local Context

In this chapter, we show how a variant of Similarity Encoder (SimEc), called Context Encoder (ConEc), provides a simple but powerful extension of the word2vec neural language model that can be used to compute embeddings for out-of-vocabulary (OOV) words as well as better embeddings for words with multiple meanings.

Representation learning is very prominent in the field of natural language processing (NLP). For example, word embeddings learned by neural language models (NLMs) were shown to improve the performance when used as features for supervised learning tasks such as named entity recognition (NER) [42, 196]. The popular *word2vec* model [129, 130] learns meaningful word embeddings by considering only the words' local contexts. Thanks to its shallow architecture, it can be trained very efficiently on large corpora. The model, however, only learns a single representation for words from a fixed vocabulary. Consequently, if in a task a new word is encountered that was not present in the texts used for training, it is not possible to create an embedding for this word without repeating the time consuming training procedure of the model.[1] Furthermore, a single embedding does not optimally represent a word with multiple meanings. For example, "Washington" is both the name of a US state as well as a former president and only by taking into account the word's local context can one identify the proper sense.

Based on an intuitive interpretation of the continuous bag-of-words (CBOW) word2vec model's negative sampling training objective, we propose an extension of the model we call *Context Encoder* (ConEc). This allows for an easy creation of OOV embeddings, as well as a better representation of words with multiple meanings by simply multiplying the trained word2vec embeddings with the words' average context vectors. As demonstrated on the CoNLL 2003 challenge, the classification performance in a NER task can be significantly improved when using as features the word embeddings created with ConEcs instead of word2vec.

Related work In the past, NLMs have addressed the issue of polysemy, i.e., the possibility of a word having multiple meanings, in various ways. For example, sense2vec is an extension of word2vec, where in a preprocessing step all words in the training corpus are annotated with their part-of-speech (POS) tag and then the embeddings are learned for tokens consisting of the words themselves and their POS tags. This way, different representations are generated,

[1] In practice the model is trained on such a large vocabulary that it is rare to encounter a word that does not have an embedding. Yet there are still scenarios where this is the case, for example, it is unlikely that the term "W10281545" is encountered in a regular training corpus, but one might still want its embedding to represent a search query like "whirlpool W10281545 ice maker part".

e.g., for words that are used both as a noun and verb [194]. Other methods first cluster the contexts in which the words appear [89] or use additional resources such as WordNet to identify multiple meanings of words [164]. One possibility to create OOV embeddings is to learn representations for all character n-grams in the texts and then compute the embedding of a word by combining the embeddings of the n-grams occurring in it [28]. However, none of these NLMs are designed to solve both the OOV and polysemy problem at the same time. Furthermore, compared to word2vec they require more parameters, resources, or additional steps in the training procedure. ConEcs, on the other hand, can generate OOV embeddings as well as improved representations for words with multiple meanings by simply multiplying the matrix of trained word2vec embeddings with the words' average context vectors.

7.1 ConEc as a simple but powerful extension of word2vec

Word2vec (Fig. 7.1) learns d-dimensional vector representations, referred to as word embeddings, for all m words in the vocabulary. It is a shallow NLM with parameter matrices $W_1, W_2 \in \mathbb{R}^{m \times d}$, which are tuned iteratively by scanning huge amounts of text sentence by sentence. Based on some context words, the CBOW word2vec model tries to predict the target word between them. Mathematically, this is realized by first computing the sum of the embeddings of the context words by selecting the appropriate rows from W_1. This vector is then multiplied by several rows selected from W_2: one of these rows corresponds to the target word, while the others correspond to k 'noise' words selected at random (negative sampling). After applying a non-linear activation function, the backpropagation error is computed by comparing this output to a label vector $\mathbf{t} \in \mathbb{R}^{k+1}$, which is 1 at the position of the target word and 0 for all k noise words. After the training of the model is complete, the word embedding for a target word is the corresponding row of W_1.

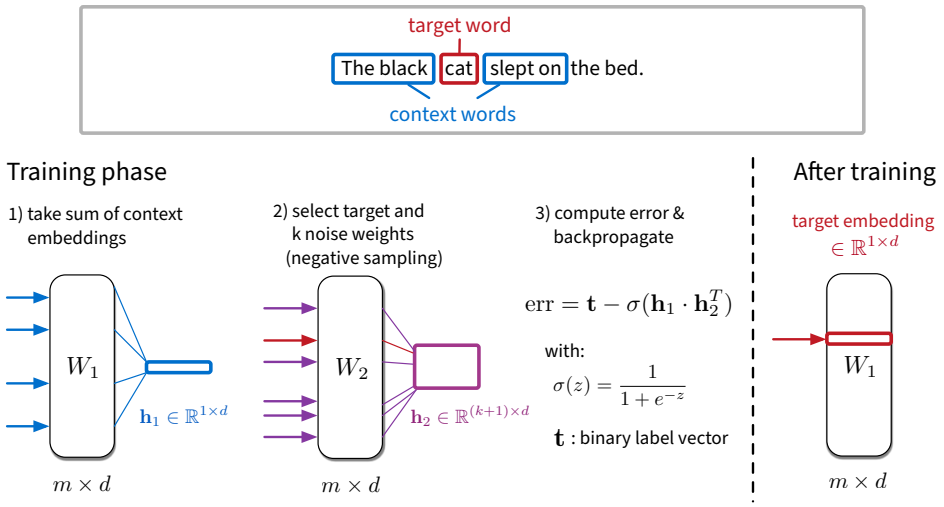


Figure 7.1: Continuous bag-of-words (CBOW) word2vec model trained with negative sampling [64, 129, 130].

Similar words tend to appear in similar contexts [73]. For example, two words synonymous with each other could be exchanged for one another in almost all contexts without a reader noticing. Based on the context word co-occurrences, pairwise similarities between all m words of the vocabulary can be computed, resulting in a similarity matrix $S \in \mathbb{R}^{m \times m}$ (or for a single word w the vector $\mathbf{s}_w \in \mathbb{R}^m$) with similarity scores between 0 and 1. These similarities should be preserved in the word embeddings, e.g., the cosine similarity between

the embedding vectors of two words used in similar contexts should be close to 1, or, more generally, the dot product of the matrix with word embeddings $Y \in \mathbb{R}^{m \times d}$ should approximate S. Obviously, the most straightforward way of obtaining word embeddings satisfying $YY^\top \approx S$ would be to compute the singular value decomposition (SVD) of the similarity matrix S and use the eigenvectors corresponding to the d largest eigenvalues [114, 115]. However, as the vocabulary typically comprises tens of thousands of words, performing an SVD of the corresponding similarity matrix is computationally far too expensive. Yet, while the similarity matrix would be huge, it would also be quite sparse, as many words are of course not synonymous with each other. If only a small number k of random words is picked, chances are their similarities to a target word would be close to 0. Therefore, while the product of a single word's embedding $\mathbf{y}_w \in \mathbb{R}^d$ with the matrix of all embeddings Y should result in a vector $\hat{\mathbf{s}}_w \in \mathbb{R}^m$ close to the true similarities $\mathbf{s}_w$ of this word, when considering only a small subset of $\hat{\mathbf{s}}_w$ corresponding to the word itself and k random words, it is sufficient if this approximates the binary vector $\mathbf{t}_w \in \mathbb{R}^{k+1}$, which is 1 for the word itself and 0 elsewhere.

The CBOW word2vec model trained with negative sampling can therefore be interpreted as a neural network (NN) that predicts a word's similarities to other words (Fig. 7.2).

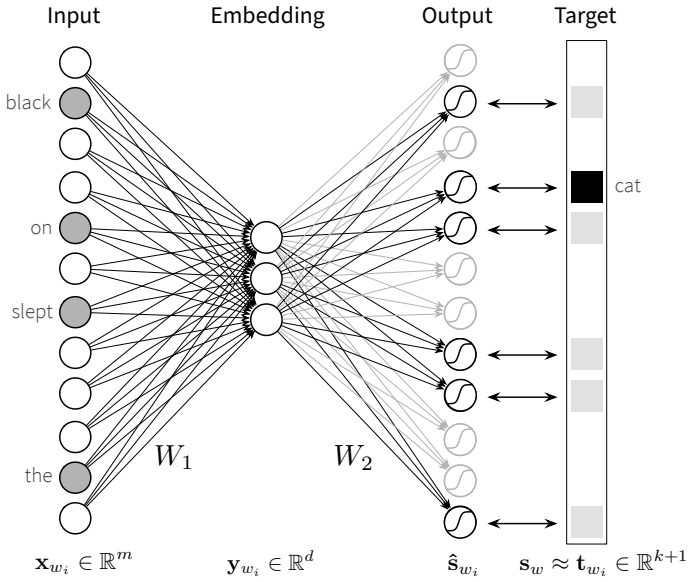


Figure 7.2: Context Encoder (ConEc) NN architecture corresponding to the CBOW word2vec model trained with negative sampling. A ConEc is an instance of a SimEc with no hidden layers and a non-linear activation after the last layer of the full network, trained with sparse binary inputs and targets with mostly missing values.

During training, for each occurrence i of a word w in the texts, a binary vector $\mathbf{x}_{w_i} \in \mathbb{R}^m$, which is 1 at the positions of the context words of w and 0 elsewhere, is used as input to the network and multiplied by a set of weights W_1 to arrive at an embedding $\mathbf{y}_{w_i} \in \mathbb{R}^d$ (the summed rows of W_1 corresponding to the context words). This embedding is then multiplied by another set of weights W_2, which corresponds to the full matrix of word embeddings Y, to produce the output of the network, a vector $\hat{\mathbf{s}}_{w_i} \in \mathbb{R}^m$ containing the approximated similarities of the word w to all other words. The training error is then computed by comparing a subset of the output to a binary target vector $\mathbf{t}_{w_i} \in \mathbb{R}^{k+1}$, which serves as an approximation of the true similarities $\mathbf{s}_w$ when considering only a small number of random words. We refer to this interpretation of the word2vec CBOW model trained with

negative sampling as a *Context Encoder* (ConEc). Indeed, a ConEc is simply a Similarity Encoder (SimEc) with no hidden layers and a non-linear activation after the last layer of the full network, and which is always trained with very sparse binary inputs and targets with mostly missing values.

While the training procedure of the ConEc is identical to that of word2vec, there is a difference in the computation of a word's embedding after the training is complete. In the case of word2vec, the word embedding is simply the row of the tuned W_1 matrix. When considering the idea behind the optimization procedure, we instead propose to create the representation of a target word w by multiplying W_1 with the word's average context vector $\mathbf{x}_w$, as this better resembles how the word embeddings are computed during training.

We distinguish between a word's 'global' and 'local' average context vector (CV): The global CV $\mathbf{x}_{w_{\text{global}}}$ is computed as the average of all binary CVs $\mathbf{x}_{w_i}$ corresponding to the N_w occurrences of w in the whole training corpus:

$$\mathbf{x}_{w_{\text{global}}} = \frac{1}{N_w} \sum_{i=1}^{N_w} \mathbf{x}_{w_i},$$

while the local CV $\mathbf{x}_{w_{\text{local}}}$ is computed likewise, but considering only the n_w occurrences of w in a single document or paragraph. We can now compute the embedding of a word w by multiplying W_1 with the weighted average of both CVs:

$$\mathbf{y}_w = (a \cdot \mathbf{x}_{w_{\text{global}}} + (1-a)\, \mathbf{x}_{w_{\text{local}}})^\top W_1 \tag{7.1}$$

with $a \in [0, 1]$. The choice of a determines how much emphasis is placed on the word's local context, which helps to distinguish between multiple meanings of the word [125].[2] As an out-of-vocabulary word does not have a global CV (as it never occurred in the training corpus), its embedding is computed solely based on the local context, i.e., setting $a = 0$.

With this new perspective on the model and optimization procedure, another advancement is feasible. Since the context words are merely a sparse feature vector used as input to a NN, there is no reason why this input vector should not contain other features about the target word as well. For example, the feature vector $\mathbf{x}_w$ could be extended to contain information about the word's case, POS tag, or other relevant details. While this would increase the dimensionality of the first weight matrix W_1 to include the additional features when mapping the input to the word's embedding, the training objective, and therefore also W_2, would remain unchanged. These additional features could be especially helpful if details about the words would otherwise get lost in preprocessing (e.g. by lowercasing) or to retain information about a word's position in the sentence, which is ignored in a BOW approach. These extended ConEcs are expected to create embeddings that even better distinguish between the words' different senses by taking into account, for example, if the word is used as a noun or verb in the current context, similar to the sense2vec algorithm [194]. But instead of explicitly learning multiple embeddings per term, like sense2vec, only the dimensionality of the input vector is increased to include the POS tag of the current word as a feature, which is expected to improve generalization if few training examples are available.

7.2 Experimental results

The word embeddings learned by word2vec and Context Encoders are evaluated on the CoNLL 2003 NER benchmark task [191]. We used a CBOW word2vec model trained with

[2] This implicitly assumes a word is only used in a single sense in one document or paragraph.

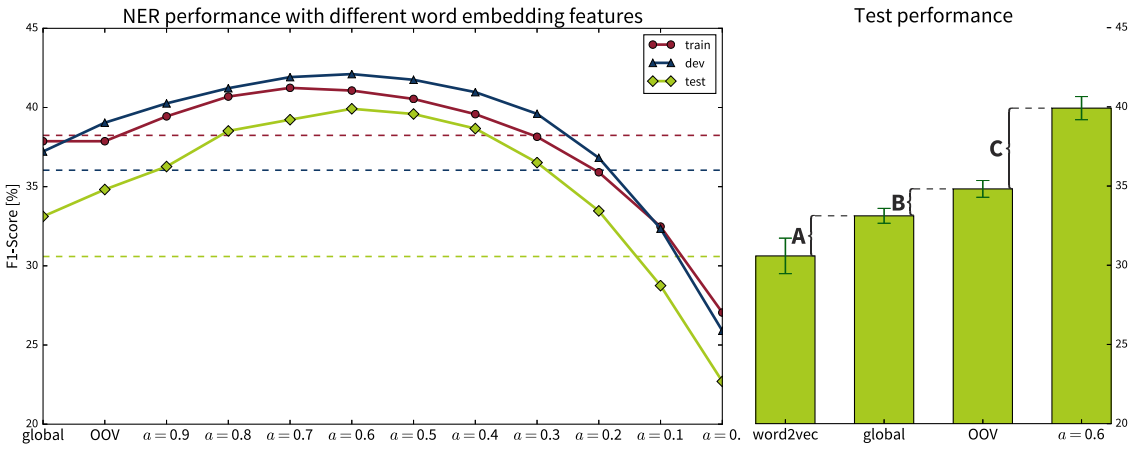


Figure 7.3: Results of the NER task based on three random initializations of the word2vec model. *Left panel:* Overall results, where the mean performance using word2vec embeddings (*dashed lines*) is considered as our baseline; all other results are computed with ConEc embeddings using various combinations of the words' global and local CVs. *Right panel:* Increased performance (mean and standard deviation) on the test fold when using ConEcs: Multiplying the word2vec embeddings with global CVs yields a performance gain of 2.5 percentage points (A). By additionally using local CVs to create OOV word embeddings, another 1.7 points are gained (B). When using a combination of global and local CVs (with $a = 0.6$) to distinguish between the different meanings of words, the F1-score increases by another 5.1 points (C), yielding a F1-score of 39.92%, which marks a significant improvement compared to the 30.59% reached with the original word2vec features.

negative sampling as described above with $k = 13$, an embedding dimensionality d of 200, and a context window of 5 words. The word embeddings created by a ConEc are built directly on top of the word2vec model by multiplying the original embeddings (W_1) with the respective context vectors. Additionally, we evaluated the word embeddings on a standard word analogy task [129]. Code to replicate the experiments is available online.[3]

Named Entity Recognition The main advantage of Context Encoders is their ability to use local context to create OOV embeddings and distinguish between the different senses of words. The effects of this are most prominent in a task such as NER, where the local context of a word can make all the difference, e.g., to distinguish between the "Chicago Bears" (an organization) and the city of Chicago (a location). We tested this on the CoNLL 2003 NER task by using the word embeddings as features together with a logistic regression classifier. The reported F1-scores were computed using the official evaluation script. The results achieved with various word embeddings in the training, development, and test part of the CoNLL task are reported in Fig. 7.3. It should be noted that we are using this task as an extrinsic evaluation to illustrate the advantages of ConEc embeddings over the regular word2vec embeddings. To isolate the effects on the performance, we are only using these word embeddings as features, while typically the performance on this NER challenge is much higher when other features, such as a word's case or POS tag, are included as well.

The word2vec embeddings were trained on the documents used in the training part of the task.[4] OOV words in the development and test parts were represented as zero vectors. With three parameter settings, we illustrate the advantages of ConEcs:

[3] `https://github.com/cod3licious/conec`

[4] Since this is a very small corpus, we trained word2vec for 25 iterations on these documents.

A) Multiplying the word2vec embeddings by the words' average context vectors generally improves the embeddings. To show this, ConEc word embeddings were computed using only global CVs (Eq. 7.1 with $a = 1$), which means OOV words again have a zero representation. With these embeddings (labeled 'global' in Fig. 7.3), the performance improves on the dev and test folds of the task.
B) Useful OOV embeddings can be created from the local context of a new word. To show this, the ConEc embeddings for words from the training vocabulary ($w \in m$) were computed as in *A)*, but now the embeddings for OOV words ($w' \notin m$) were computed using local CVs (Eq. 7.1 with $a = 1 \,\forall\, w \in m$ and $a = 0 \,\forall\, w' \notin m$; referred to as 'OOV' in the figure). The training performance obviously stays the same, because here all words have an embedding based on their global contexts. However, there is a jump in the ConEc performance on the dev and test folds, where OOV words now have a representation based on their local contexts.
C) Better embeddings for a word with multiple meanings can be created by using a combination of the word's average global and local CVs as input to the ConEc. To show this, the OOV embeddings were computed as in *B)*, but now for the words occurring in the training vocabulary, the local context was taken into account as well by setting $a < 1$ (Eq. 7.1 with $a \in [0, 1) \,\forall\, w \in m$ and $a = 0 \,\forall\, w' \notin m$). The best performances on all folds are achieved when averaging the global and local CVs with around $a = 0.6$ before multiplying them with the word2vec embeddings. This clearly shows that ConEc embeddings created by incorporating local context can help distinguish between multiple meanings of words.

Analogy task To show that the word embeddings created with Context Encoders capture meaningful semantic and syntactic relationships between words, we evaluated them on the original analogy task published together with the word2vec model [129].[5] This task consists of many questions in the form of "*man* is to *king* as *woman* is to XXX" where the model is supposed to find the correct answer *queen*. This is accomplished by taking the word embedding for *king*, subtracting from it the embedding for *man*, and then adding the embedding for *woman*. This new word vector should then be most similar (with respect to the cosine similarity) to the embedding for *queen*.[6] The word2vec model was trained for ten iterations on the `text8` corpus,[7] which contains around 17 million words and a vocabulary of about 70k unique words, as well as the training part of the `1-billion` benchmark dataset,[8] which contains over 768 million words with a vocabulary of 486k unique words.[9] The ConEc embeddings were then constructed by multiplying the word2vec embeddings with the words' average global context vectors obtained from the same corpus as the word2vec model was trained on. To achieve the best results, we also had to include the target word itself in these context vectors.

The results of the analogy task are shown in Table 7.1. To capture some of the semantic relations between words (e.g. the first four task categories) it can be advantageous to use Context Encoders instead of word2vec. One reason for the ConEcs' superior performance on some of the task categories, but not others, might be that the city and country names compared in the first four task categories only have a single sense (referring to the respective location), while the words asked for in other task categories can have multiple meanings.

[5] `https://code.google.com/archive/p/word2vec/`
[6] Readers familiar with Levy, Goldberg, and Dagan [114] will recognize this as the 3CosAdd method. We have tried 3CosMul as well, but found that the results did not improve significantly and therefore omitted them here.
[7] `http://mattmahoney.net/dc/text8.zip`
[8] `http://code.google.com/p/1-billion-word-language-modeling-benchmark/`
[9] In this experiment we ignored all words that occur less than 5 times in the training corpus.

For example, "run" can be used as both a noun or a verb, additionally, in some contexts it refers to the sport activity, while other times it is used in a more abstract sense, e.g., in the context of someone running for president. Therefore, the results in the other task categories might improve if the words' context vectors were first clustered and then the ConEc embedding was generated by multiplying the word2vec embeddings with the average of only those context vectors corresponding to the word's sense most appropriate for the task category.

Table 7.1: Accuracy on the analogy task with mean and standard deviation computed using three random seeds when initializing the word2vec model. The best results for each category and corpus are in bold.

	text8 (10 iter)		1-billion	
	word2vec	Context Encoder	word2vec	Context Encoder
capital-common-countries	63.8±4.7	**78.7**±0.2	79.3±2.2	**83.1**±1.2
capital-world	34.0±2.1	**54.7**±1.3	63.8±1.4	**75.9**±0.4
currency	15.4±0.9	**19.3**±0.6	13.3±3.6	**14.8**±0.8
city-in-state	28.6±1.0	**43.6**±0.9	19.6±1.7	**29.6**±1.0
family	**79.6**±1.5	77.2±0.4	78.7±2.2	**79.0**±1.4
gram1-adjective-to-adverb	11.0±0.9	**16.6**±0.7	12.3±0.5	**13.3**±1.1
gram2-opposite	24.3±3.0	24.3±2.0	**27.6**±0.1	21.3±1.1
gram3-comparative	**64.3**±0.5	63.0±1.1	**83.7**±0.9	76.2±1.1
gram4-superlative	**40.3**±2.1	37.6±1.5	**69.4**±0.5	56.2±1.2
gram5-present-participle	30.5±1.0	**31.7**±0.4	**78.4**±1.0	68.0±0.7
gram6-nationality-adjective	**70.6**±1.5	67.2±1.4	83.8±0.6	**83.8**±0.5
gram7-past-tense	30.5±1.8	**33.0**±0.6	**53.9**±0.9	49.2±0.7
gram8-plural	**49.8**±0.3	49.2±1.2	**62.7**±1.9	56.7±1.0
gram9-plural-verbs	**41.0**±2.5	30.1±1.9	**68.7**±0.2	45.0±0.4
total	42.1±0.6	**46.5**±0.1	**57.2**±0.3	55.8±0.3

8 Conclusion

Representing intrinsically complex structured data is an ubiquitous challenge in machine learning. While spectral methods such as kernel PCA provide optimal similarity preserving embeddings by computing the eigendecomposition of a similarity matrix, they are unable to produce out-of-sample (OOS) solutions for new test samples if their similarity to the original training examples can not be computed. Neural network based methods provide a mapping function from an original input feature space to the embedding space and can therefore also approximate the pairwise relations between new data points. However, existing methods were not devised to predict non-metric similarities or multiple pairwise relations at once.

Similarity Encoder (SimEc) are a novel neural network (NN) architecture designed for simultaneously factorizing a target matrix with pairwise relations while learning a mapping from the original input feature space into a similarity preserving embedding space. As we have demonstrated in multiple experiments, SimEcs can provide OOS solutions even if the target similarities were obtained by an unknown process such as human ratings, they can efficiently handle missing values in the target matrix or noisy inputs, and they are able to predict non-metric similarities as well as multiple similarities at once.

Depending on the SimEc architecture – linear or with multiple hidden layers – and the given target similarities, SimEcs can easily recreate solutions found by various spectral methods such as (kernel) PCA or Isomap. Additionally, SimEcs can learn a mapping into a similarity preserving embedding space for target similarities not computed from the original feature vectors, such as similarities based on class labels, whereas creating the embeddings for new test points based on the SVD of such a similarity matrix would require training an additional regression model. This, however, can decrease the embedding quality compared to learning the factorization and mapping simultaneously [33].

Non-metric similarities are characterized by an eigenvalue spectrum with a strong negative part. As it was previously shown, the eigenvectors associated with these negative eigenvalues can reveal very interesting features of the data [106]. However, traditional spectral methods, such as kernel PCA, require positive semi-definite similarity matrices or would simply ignore the components associated with the negative eigenvalues and only preserve the information present in the largest positive eigenvalues. Similarly, neural network based embedding methods that operate on batches of points to approximate a corresponding symmetric part of the target similarity matrix [31, 93, 117, 118] are unable to capture the information associated with the negative part of the eigenvalue spectrum, as this would require that the embeddings contain imaginary numbers in some dimensions. SimEcs, on the other hand, predict the target similarities as the dot product of the embedding Y, computed by mapping the original feature vectors into the embedding space with the first part of

the network, and the last layer of the full SimEc network, W_l. As these two matrices can have opposing signs in some dimensions, SimEcs are able to retain the features present in the components corresponding to the negative eigenvalues; indeed, SimEcs will preserve the information associated with the eigenvalues of largest absolute magnitude. This is also important to remember when using a SimEc to preserve multiple pairwise relations at once: to give equal weight to the targets contained in multiple matrices, these matrices should first be normalized by dividing them by their respective absolute largest eigenvalue. To predict multiple pairwise relations at the same time, the last layer of a SimEc network can simply be extended to a tensor to approximate the tensor containing the different target matrices.

The prediction of pairwise relations is an important component of many practical applications, such as link prediction or for recommender systems. While SimEcs do not outperform state-of-the-art models specifically developed to solve these tasks, SimEcs can serve as a reliable multi-purpose baseline model. Especially when dealing with extremely sparse target matrices and if mappings from multiple feature spaces (e.g. for items and users) are required, it might be advantageous to instead use a model with two neural networks mapping into the same embedding space to predict a single target value [70, 90, 205]. While SimEcs can easily handle missing values in the target matrix by computing the backpropagation error only based on the given values, the model is designed to efficiently factorize denser matrices. Similarly, it is possible to learn a second mapping from a different feature space into the same embedding space by training a second SimEc with a fixed last layer set to the embeddings computed by the first SimEc. However, in practice this will introduce additional noise and lead to less accurate mappings compared to those created by models that learn multiple embedding functions simultaneously.

One application area where we see great promise for using SimEcs is content based recommendations. As SimEcs are able to learn the mapping from items' original feature vectors to an embedding that preserves similarities relevant to user behavior, the model has the potential to improve recommendations for new items that did not receive any user ratings yet. This might even improve suggestions based on full text similarity searches, such as for identifying related scientific literature when writing a new paper [83] or when searching for a patent's prior art [77].

As machine learning models make their way into our everyday lives, for example, by detecting irregularities in medical imaging or as critical components in self-driving cars, explaining their predictions is now more important than ever. A popular technique to do this for neural network models is layer-wise relevance propagation (LRP) [9, 13, 95, 132], which deconstructs the prediction of a NN to reveal which of the input features contributed most strongly to the decision of the model. As we have demonstrated, it is possible to use LRP to explain the predictions and similarities computed by a SimEc network. This way, for example, it could be made more transparent to a user why s/he is recommended certain products.

As the first part of the SimEc architecture, mapping from the original input features to the embedding space, can be realized by any (deep) neural network, SimEcs can adapt to a variety of application areas. This also makes it possible to use SimEcs to pre-train neural networks for different supervised learning tasks. If only a few labeled examples are available to train a (deep) neural network on a supervised task, the prediction accuracy can often be improved by pre-training the network on a related task with more data or on unlabeled examples, e.g., using an auto-encoder network. Pre-training a network using a SimEc, i.e., by replacing the prediction layer with a last layer to approximate some target similarities, can also improve the performance of this network in the following supervised task. However, as we have shown by pre-training a CNN with a SimEc for an image classification task, for

this to work it is important that the target similarities used to train the SimEc are well aligned with the similarities of the data points' labels. On the other hand, by constructing a similarity matrix from the class labels of the training samples, the generalization boost obtained through the SimEc pre-training is still present even when the final network is fine-tuned on a large training set.

One supervised learning task where the SimEc pre-training shows particular promise is for the prediction of chemical properties of molecules. Generating training data for this task, i.e., computing e.g. the atomization energy of molecules, is computationally very expensive and takes a long time. But by using topological fingerprints together with common similarity coefficients, it is possible to efficiently compute similarities between molecules based on their structural information alone. By pre-training the SchNet neural network architecture [176, 178] with a SimEc on such a similarity measure, the predictions of the molecules' properties are more accurate compared to only training SchNet on a few hundred labeled training samples alone. Additionally, while in the past pre-training on unlabeled data with auto-encoders (AEs) has also resulted in improved performances on some supervised tasks, AEs can not be used to pre-train a network architecture like SchNet. It is notoriously difficult to represent molecules with different numbers and types of atoms as fixed feature vectors, which is why SchNet instead learns a meaningful representation for each molecule using embeddings for each of its atoms and computing interactions between them based on the atoms' positions in space. Therefore, this network is not trivial to reverse to create the characteristic mirror architecture used in auto-encoders. And, as SchNet operates directly on the molecule's atoms and their coordinates, there are no feature vectors available that an AE could aim to reconstruct with its output. When performing the pre-training with a SimEc, on the other hand, one is not faced with these problems. Instead, the molecules' representations learned by SchNet can be used together with a SimEc last layer to approximate the target similarities computed from the molecules' structures. While a SimEc pre-training of SchNet with fingerprint similarities has resulted in an improved prediction performance, a pre-training with the more informative FCHL kernel as a target similarity matrix did not yield the expected performance boost. While the Period/Group architecture extension of SchNet, where atom embeddings are constructed from their respective period and group embeddings, resulted in a more effective FCHL SimEc pre-training, these changes were still not enough to capture all details and domain knowledge that go into the computation of this kernel and contribute to its superior performance when used to predict the molecules' properties with kernel ridge regression. Nevertheless, the PG-SchNet model might be useful in other scenarios when predictions need to be made for molecules with atoms not encountered during training, for which a regular SchNet model would not be able to produce an informative embedding.

Natural language processing (NLP) is another field where using SimEcs proved beneficial. With a simple architecture and the ability to learn meaningful word embeddings efficiently from texts containing billions of words, word2vec remains one of the most popular neural language models used today. However, as only a single embedding is learned for every word in the vocabulary, the model fails to optimally represent words with multiple meanings. Additionally, it is not possible to create embeddings for new (out-of-vocabulary) words on the spot. Based on an intuitive interpretation of the continuous bag-of-words (CBOW) word2vec model's negative sampling training objective in terms of predicting context based similarities, we motivated an extension of the model we call Context Encoder (ConEc). By multiplying the matrix of trained word2vec embeddings with a word's average context vector, out-of-vocabulary (OOV) embeddings and representations for a word with multiple meanings can be created based on the word's local contexts. The superiority of these word

embeddings was demonstrated by using them as features in the CoNLL 2003 named entity recognition (NER) task. ConEcs are a simple variant of SimEcs, with a single linear layer mapping to the embedding, trained on very sparse inputs and binary targets with mostly missing values. By viewing the word2vec model as a neural network that gets as input a binary feature vector indicating a word's contexts words, a further advancement of the model is possible: additional features, such as a word's case or part-of-speech (POS) tag could be used as input as well. This could help retain information that might otherwise get lost in preprocessing (e.g. by lowercasing) or to include details about the position of a word in a sentence, which is not considered in a bag-of-words (BOW) approach. We expect that using such extended feature vectors will further improve the expressiveness of the resulting word embeddings.

Bibliography

[1] Madhu S Advani and Andrew M Saxe. "High-dimensional dynamics of generalization error in neural networks". In: *arXiv preprint arXiv:1710.03667* (2017).

[2] Muhammad Jamal Afridi, Arun Ross, and Erik M Shapiro. "On automated source selection for transfer learning in convolutional neural networks". In: *Pattern recognition* 73 (2018), pp. 65–75.

[3] Pulkit Agrawal, Ross Girshick, and Jitendra Malik. "Analyzing the performance of multilayer neural networks for object recognition". In: *European conference on computer vision*. Springer. 2014, pp. 329–344.

[4] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. "Distributed large-scale natural graph factorization". In: *Proceedings of the 22nd International Conference on World Wide Web*. ACM. 2013, pp. 37–48.

[5] Maximilian Alber, Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Fei Sha. "An Empirical Study on The Properties of Random Bases for Kernel Methods". In: *Advances in Neural Information Processing Systems*. 2017, pp. 2760–2771.

[6] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, et al. "iNNvestigate neural networks!" In: *Journal of Machine Learning Research* 20.93 (2019), pp. 1–8.

[7] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, et al. "A closer look at memorization in deep networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 233–242.

[8] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. "Explaining Predictions of Non-Linear Classifiers in NLP". In: *Proceedings of the 1st Workshop on Representation Learning for NLP*. Association for Computational Linguistics. Aug. 2016, pp. 1–7.

[9] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. ""What is relevant in a text document?": An interpretable machine learning approach". In: *PLoS ONE* 12.8 (2017), e0181142.

[10] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. "Explaining Recurrent Neural Network Predictions in Sentiment Analysis". In: *arXiv preprint arXiv:1706.07206* (2017).

[11] Yuval Atzmon, Uri Shalit, and Gal Chechik. "Learning sparse metrics, one feature at a time". In: *Feature Extraction: Modern Questions and Challenges*. 2015, pp. 30–48.

[12] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. "Factors of transferability for a generic convnet representation". In: *IEEE transactions on pattern analysis and machine intelligence* 38.9 (2015), pp. 1790–1802.

[13] Sebastian Bach, Alexander Binder, Grégoire Montavon, et al. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation". In: *PLoS ONE* 10.7 (2015), e0130140.

[14] Ivana Balažević, Carl Allen, and Timothy M. Hospedales. "Hypernetwork Knowledge Graph Embeddings". In: *International Conference on Artificial Neural Networks*. Springer. 2019, pp. 553–565.

[15] Oren Barkan and Noam Koenigstein. "Item2vec: neural item embedding for collaborative filtering". In: *26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2016, pp. 1–6.

[16] Jörg Behler. "First Principles Neural Network Potentials for Reactive Simulations of Large Molecular and Condensed Systems". In: *Angew. Chem. Int. Edit.* 56.42 (2017), pp. 12828–12840. ISSN: 1521-3773.

[17] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[18] Robert M Bell and Yehuda Koren. "Lessons from the Netflix prize challenge". In: *Acm Sigkdd Explorations Newsletter* 9.2 (2007), pp. 75–79.

[19] Sean Bell and Kavita Bala. "Learning visual similarity for product design with convolutional neural networks". In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 98.

[20] Aurélien Bellet, Amaury Habrard, and Marc Sebban. "A survey on metric learning for feature vectors and structured data". In: *arXiv preprint arXiv:1306.6709* (2013).

[21] Shai Ben-David and Reba Schuller. "Exploiting task relatedness for multiple task learning". In: *Learning Theory and Kernel Machines*. Springer, 2003, pp. 567–580.

[22] Yoshua Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 2012, pp. 17–36.

[23] Yoshua Bengio, Aaron Courville, and Pierre Vincent. "Representation learning: A review and new perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828.

[24] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. "Greedy layer-wise training of deep networks". In: *Advances in Neural Information Processing Systems*. 2007, pp. 153–160.

[25] Yoshua Bengio, Jean-françcois Paiement, Pascal Vincent, et al. "Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering". In: *Advances in Neural Information Processing Systems*. 2004, pp. 177–184.

[26] Jacob Biamonte, Peter Wittek, Nicola Pancotti, et al. "Quantum machine learning". In: *Nature* 549.7671 (2017), p. 195.

[27] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[28] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606* (2016).

[29] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. "Translating Embeddings for Modeling Multi-relational Data". In: *Advances in Neural Information Processing Systems*. 2013.

[30] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature verification using a" siamese" time delay neural network". In: *Advances in Neural Information Processing Systems*. 1994, pp. 737–744.

[31] Kerstin Bunte, Michael Biehl, and Barbara Hammer. "A general framework for dimensionality-reducing data visualization mapping". In: *Neural Computation* 24.3 (2012), pp. 771–804.

[32] Sirui Cai, Yuchun Fang, and Zhengyan Ma. "Will Outlier Tasks Deteriorate Multitask Deep Learning?" In: *International Conference on Neural Information Processing*. Springer. 2017, pp. 246–255.

[33] Miguel A Carreira-Perpinán and Max Vladymyrov. "A fast, universal algorithm to learn parametric nonlinear embeddings". In: *Advances in Neural Information Processing Systems*. 2015, pp. 253–261.

[34] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. "Large scale online learning of image similarity through ranking". In: *Journal of Machine Learning Research* 11.Mar (2010), pp. 1109–1135.

[35] Xing Chen, Chenggang Clarence Yan, Xiaotian Zhang, et al. "Drug–target interaction prediction: databases, web servers and computational models". In: *Briefings in Bioinformatics* 17.4 (2015), pp. 696–712.

[36] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[37] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. "The loss surfaces of multilayer networks". In: *Artificial intelligence and statistics*. 2015, pp. 192–204.

[38] Anders S Christensen, Lars A Bratholm, Felix A Faber, David R Glowacki, and O Anatole von Lilienfeld. "FCHL revisited: faster and more accurate quantum machine learning". In: *arXiv preprint arXiv:1909.01946* (2019).

[39] Anders S Christensen, Felix A Faber, Bing Huang, et al. *QML: A Python toolkit for quantum machine learning*. 2017.

[40] Andrzej Cichocki. "Neural network for singular value decomposition". In: *Electronics Letters* 28.8 (1992), pp. 784–786.

[41] Andrzej Cichocki and Rolf Unbehauen. "Neural networks for computing eigenvalues and eigenvectors". In: *Biological Cybernetics* 68.2 (1992), pp. 155–164.

[42] Ronan Collobert, Jason Weston, Léon Bottou, et al. "Natural language processing (almost) from scratch". In: *Journal of Machine Learning Research* 12 (2011), pp. 2493–2537.

[43] Paul Covington, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM. 2016, pp. 191–198.

[44] Christopher J Cramer. *Essentials of computational chemistry: theories and models*. John Wiley & Sons, 2004.

[45] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[46] Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. "Information-theoretic metric learning". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 209–216.

[47] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. "Convolutional 2D Knowledge Graph Embeddings". In: *Association for the Advancement of Artificial Intelligence*. 2018.

[48] Hao Ding, Ichigaku Takigawa, Hiroshi Mamitsuka, and Shanfeng Zhu. "Similarity-based machine learning methods for predicting drug–target interactions: a brief review". In: *Briefings in Bioinformatics* 15.5 (2013), pp. 734–747.

[49] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. "Sharp minima can generalize for deep nets". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1019–1028.

[50] Jeff Donahue, Yangqing Jia, Oriol Vinyals, et al. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International Conference on Machine Learning*. 2014, pp. 647–655.

[51] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, et al. "Convolutional networks on graphs for learning molecular fingerprints". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2224–2232.

[52] Gintare Karolina Dziugaite and Daniel M Roy. "Neural network matrix factorization". In: *arXiv preprint arXiv:1511.06443* (2015).

[53] Dumitru Erhan, Yoshua Bengio, Aaron Courville, et al. "Why does unsupervised pre-training help deep learning?" In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 625–660.

[54] Felix A Faber, Anders S Christensen, Bing Huang, and O Anatole von Lilienfeld. "Alchemical and structural distribution based representation for universal quantum machine learning". In: *The Journal of chemical physics* 148.24 (2018), p. 241717.

[55] Felix A Faber, Luke Hutchison, Bing Huang, et al. "Machine learning prediction errors better than DFT accuracy". In: *arXiv preprint arXiv:1702.05532* (2017).

[56] Shobeir Fakhraei, Louiqa Raschid, and Lise Getoor. "Drug-target interaction prediction for drug repurposing with probabilistic similarity logic". In: *Proceedings of the 12th International Workshop on Data Mining in Bioinformatics*. ACM. 2013, pp. 10–17.

[57] Masoud Faraki, Mehrtash T Harandi, and Fatih Porikli. "Large-scale metric learning: A voyage from shallow to deep". In: *IEEE transactions on neural networks and learning systems* 29.9 (2017), pp. 4339–4346.

[58] Li Fei-Fei. "Knowledge transfer in learning to recognize visual objects classes". In: *Proceedings of the International Conference on Development and Learning (ICDL)*. 2006, p. 11.

[59] Li Fei-Fei, Rob Fergus, and Pietro Perona. "One-shot learning of object categories". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.4 (2006), pp. 594–611.

[60] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. "Unsupervised visual domain adaptation using subspace alignment". In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2960–2967.

[61] Itamar Gati and Amos Tversky. "Representations of qualitative and quantitative dimensions." In: *Journal of Experimental Psychology: Human Perception and Performance* 8.2 (1982), p. 325.

[62] Mustansar Ali Ghazanfar, Adam Prügel-Bennett, and Sandor Szedmak. "Kernel-mapping recommender system algorithms". In: *Information Sciences* 208 (2012), pp. 81–104.

[63] Garrett B Goh, Charles Siegel, Abhinav Vishnu, and Nathan O Hodas. "ChemNet: A transferable and generalizable deep neural network for small-molecule property prediction". In: *arXiv preprint arXiv:1712.02734* (2017).

[64] Yoav Goldberg and Omer Levy. "word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method". In: *arXiv preprint arXiv:1402.3722* (2014).

[65] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, et al. "Automatic chemical design using a data-driven continuous representation of molecules". In: *ACS central science* 4.2 (2018), pp. 268–276.

[66] Mehmet Gönen. "Predicting drug–target interactions from chemical and genomic kernels using Bayesian matrix factorization". In: *Bioinformatics* 28.18 (2012), pp. 2304–2310.

[67] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[68] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. "Qualitatively characterizing neural network optimization problems". In: *International Conference on Learning Representations*. 2015.

[69] Matthieu Guillaumin, Jakob Verbeek, and Cordelia Schmid. "Is that you? Metric learning approaches for face identification". In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 498–505.

[70] Raia Hadsell, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping". In: *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. Vol. 2. IEEE. 2006, pp. 1735–1742.

[71] William L Hamilton, Rex Ying, and Jure Leskovec. "Representation Learning on Graphs: Methods and Applications". In: *arXiv preprint arXiv:1709.05584* (2017).

[72] F Maxwell Harper and Joseph A Konstan. "The movielens datasets: History and context". In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5.4 (2016), p. 19.

[73] Zellig S Harris. "Distributional structure". In: *Word* 10.2-3 (1954), pp. 146–162.

[74] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[75] Kaiming He, Ross Girshick, and Piotr Dollár. "Rethinking imagenet pre-training". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 4918–4927.

[76] Xiangnan He, Lizi Liao, Hanwang Zhang, et al. "Neural collaborative filtering". In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.

[77] Lea Helmers, Franziska Horn, Franziska Biegler, Tim Oppermann, and Klaus-Robert Müller. "Automating the search for a patent's prior art with a full text similarity search". In: *PLoS ONE* 14.3 (2019), e0212103.

[78] Alex Hernández-Garcia and Peter König. "Data augmentation instead of explicit regularization". In: *arXiv preprint arXiv:1806.03852* (2018).

[79] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural Computation* 18.7 (2006), pp. 1527–1554.

[80] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

[81] Thomas Hofmann and Joachim M Buhmann. "Pairwise data clustering by deterministic annealing". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.1 (1997), pp. 1–14.

[82] Franziska Horn. "Context encoders as a simple but powerful extension of word2vec". In: *Proceedings of the 2nd Workshop on Representation Learning for NLP*. Association for Computational Linguistics. Aug. 2017, pp. 10–14.

[83] Franziska Horn. "Interactive Exploration and Discovery of Scientific Publications with PubVis". In: *arXiv preprint arXiv:1706.08094* (2017).

[84] Franziska Horn, Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. "Discovering topics in text datasets by visualizing relevant words". In: *arXiv preprint arXiv:1707.06100* (2017).

[85] Franziska Horn, Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. "Exploring text datasets by visualizing relevant words". In: *arXiv preprint arXiv:1707.05261* (2017).

[86] Franziska Horn and Klaus-Robert Müller. "Predicting Pairwise Relations with Neural Similarity Encoders". In: *Bulletin of the Polish Academy of Sciences: Technical Sciences* 66.6 (2018), pp. 821–830.

[87] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, et al. "Collaborative metric learning". In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 193–201.

[88] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets". In: *ICDM'08. Eighth IEEE International Conference on Data Mining, 2008*. IEEE. 2008, pp. 263–272.

[89] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. "Improving word representations via global context and multiple word prototypes". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*. ACL. 2012, pp. 873–882.

[90] Po-Sen Huang, Xiaodong He, Jianfeng Gao, et al. "Learning deep structured semantic models for web search using clickthrough data". In: *Proceedings of the 22nd ACM international conference on information & knowledge management*. ACM. 2013, pp. 2333–2338.

[91] Sabrina Jaeger, Simone Fulle, and Samo Turk. "Mol2vec: unsupervised machine learning approach with chemical intuition". In: *Journal of chemical information and modeling* 58.1 (2018), pp. 27–35.

[92] Mark A. Johnson (Editor) and Gerald M. Maggiora (Editor). *Concepts and Applications of Molecular Similarity*. New York, USA: John Willey & Sons, 1990.

[93] Michael Kampffmeyer, Sigurd Løkse, Filippo M Bianchi, Robert Jenssen, and Lorenzo Livi. "Deep Kernelized Autoencoders". In: *Scandinavian Conference on Image Analysis*. Springer. 2017, pp. 419–430.

[94] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima". In: *arXiv preprint arXiv:1609.04836* (2016).

[95] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, and Sven Dähne. "PatternNet and PatternLRP–Improving the interpretability of neural networks". In: *arXiv preprint arXiv:1705.05598* (2017).

[96] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems". In: *Computer* 42.8 (2009).

[97] Simon Kornblith, Jonathon Shlens, and Quoc V Le. "Do better imagenet models transfer better?" In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 2661–2671.

[98] Wouter M Kouw and Marco Loog. "An introduction to domain adaptation and transfer learning". In: *arXiv preprint arXiv:1812.11806* (2018).

[99] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[100] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[101] S. Kulczynski. "Die Pflanzenassociationen der Pienenen". In: *Bulletin International de L'Academie Polonaise des Sciences et des letters, classe des sciences mathematiques et naturelles, Serie B, Supplement II,* 2 (1927), pp. 57–203.

[102] Brian Kulis et al. "Metric learning: A survey". In: *Foundations and Trends® in Machine Learning* 5.4 (2013), pp. 287–364.

[103] Andrew K Lampinen and Surya Ganguli. "An analytic theory of generalization dynamics and transfer learning in deep linear networks". In: *arXiv preprint arXiv:1809.10374* (2018).

[104] Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. "Analyzing Classifiers: Fisher Vectors and Deep Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2912–2920.

[105] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, et al. "Unmasking Clever Hans predictors and assessing what machines really learn". In: *Nature communications* 10.1 (2019), p. 1096.

[106] Julian Laub and Klaus-Robert Müller. "Feature discovery in non-metric pairwise data". In: *Journal of Machine Learning Research* 5 (2004), pp. 801–818.

[107] Quoc V Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *arXiv preprint arXiv:1405.4053* (2014).

[108] Andrew R Leach and Valerie J Gillet. *An introduction to chemoinformatics*. Springer Science & Business Media, 2007.

[109] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[110] Yann LeCun, Bernhard E Boser, John S Denker, et al. "Handwritten digit recognition with a back-propagation network". In: *Advances in Neural Information Processing Systems*. 1990, pp. 396–404.

[111] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[112] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[113] Omer Levy and Yoav Goldberg. "Neural word embedding as implicit matrix factorization". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2177–2185.

[114] Omer Levy, Yoav Goldberg, and Ido Dagan. "Improving distributional similarity with lessons learned from word embeddings". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 211–225.

[115] Omer Levy, Yoav Goldberg, and Israel Ramat-Gan. "Linguistic Regularities in Sparse and Explicit Word Representations." In: *CoNLL*. 2014, pp. 171–180.

[116] Henry W Lin, Max Tegmark, and David Rolnick. "Why does deep and cheap learning work so well?" In: *Journal of Statistical Physics* 168.6 (2017), pp. 1223–1247.

[117] David Lowe and Michael Tipping. "Feed-forward neural networks and topographic mappings for exploratory data analysis". In: *Neural Computing & Applications* 4.2 (1996), pp. 83–95.

[118] Laurens van der Maaten. "Learning a parametric embedding by preserving local structure". In: *International Conference on Artificial Intelligence and Statistics*. 2009, pp. 384–391.

[119] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[120] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. "Domain adaptation: Learning bounds and algorithms". In: *arXiv preprint arXiv:0902.3430* (2009).

[121] Jianchang Mao and Anil K Jain. "Artificial neural networks for feature extraction and multivariate data projection". In: *IEEE Transactions on Neural Networks* 6.2 (1995), pp. 296–317.

[122] Bayard Harlow McConnaughey. *The determination and analysis of plankton communities*. Lembaga Penelitian Laut, 1964.

[123] Brian McFee, Luke Barrington, and Gert Lanckriet. "Learning content similarity for music recommendation". In: *IEEE transactions on Audio, Speech, and Language Processing* 20.8 (2012), pp. 2207–2218.

[124] Brian McFee and Gert Lanckriet. "Learning multi-modal similarity". In: *Journal of Machine Learning Research* 12.Feb (2011), pp. 491–523.

[125] Oren Melamud, Ido Dagan, and Jacob Goldberger. "Modeling word meaning in context with substitute vectors". In: *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*. 2015.

[126] Dieter Merkl and Andreas Rauber. "On the similarity of eagles, hawks, and cows: visualization of semantic similarity in self-organizing maps". In: *Proc. Int'l Workshop on Fuzzy-Neuro-Systems*. Citeseer. 1997.

[127] Alexis Mignon and Frédéric Jurie. "Pcca: A new approach for distance learning from sparse pairwise constraints". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 2666–2672.

[128] Sebastian Mika, Bernhard Schölkopf, Alex J Smola, et al. "Kernel PCA and de-noising in feature spaces". In: *Advances in Neural Information Processing Systems*. 1999, pp. 536–542.

[129] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[130] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality". In: *Advances in Neural Information Processing Systems*. 2013, pp. 3111–3119.

[131] Andriy Mnih and Ruslan R Salakhutdinov. "Probabilistic matrix factorization". In: *Advances in Neural Information Processing Systems*. 2008, pp. 1257–1264.

[132] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. "Explaining nonlinear classification decisions with deep taylor decomposition". In: *Pattern Recognition* 65 (2017), pp. 211–222.

[133] Grégoire Montavon, Matthias Rupp, Vivekanand Gobre, et al. "Machine learning of molecular electronic properties in chemical compound space". In: *New Journal of Physics* 15.9 (2013), p. 095003.

[134] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for interpreting and understanding deep neural networks". In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[135] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. "An introduction to kernel-based learning algorithms". In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 181–201.

[136] Vaishnavh Nagarajan and J Zico Kolter. "Generalization in deep networks: The role of distance from initialization". In: *arXiv preprint arXiv:1901.01672* (2019).

[137] André CA Nascimento, Ricardo BC Prudêncio, and Ivan G Costa. "A multiple kernel learning algorithm for drug-target interaction prediction". In: *BMC Bioinformatics* 17.1 (2016), p. 46.

[138] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. "Exploring generalization in deep learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5947–5956.

[139] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. "In search of the real inductive bias: On the role of implicit regularization in deep learning". In: *arXiv preprint arXiv:1412.6614* (2014).

[140] Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, et al. "Domain adaptive transfer learning with specialist models". In: *arXiv preprint arXiv:1811.07056* (2018).

[141] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. "A Three-Way Model for Collective Learning on Multi-Relational Data". In: *ICML*. Vol. 11. 2011, pp. 809–816.

[142] Frank Noé, Alexandre Tkatchenko, Klaus-Robert Müller, and Cecilia Clementi. "Machine learning for molecular simulation". In: *arXiv preprint arXiv:1911.02792* (2019).

[143] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. "Deep metric learning via lifted structured feature embedding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4004–4012.

[144] Erkki Oja. "Simplified neuron model as a principal component analyzer". In: *Journal of Mathematical Biology* 15.3 (1982), pp. 267–273.

[145] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748* (2018).

[146] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. "Learning and transferring mid-level image representations using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1717–1724.

[147] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), pp. 1345–1359.

[148] Adam Paszke, Sam Gross, Soumith Chintala, et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[149] Karl Pearson. "On lines and planes of closest fit to systems of point in space". In: *Philosophical Magazine* 2.11 (1901), pp. 559–572.

[150] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2825–2830.

[151] Tomaso Poggio, Andrzej Banburski, and Qianli Liao. "Theoretical Issues in Deep Networks: Approximation, Optimization and Generalization". In: *arXiv preprint arXiv:1908.09375* (2019).

[152] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. "Transfusion: Understanding transfer learning for medical imaging". In: *Advances in Neural Information Processing Systems*. 2019, pp. 3342–3352.

[153] Ali Rahimi and Benjamin Recht. "Random features for large-scale kernel machines". In: *Advances in Neural Information Processing Systems*. 2007, pp. 1177–1184.

[154] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. "Self-taught learning: transfer learning from unlabeled data". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 759–766.

[155] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. "Quantum chemistry structures and properties of 134 kilo molecules". In: *Scientific Data* 1 (2014).

[156] John W Raymond and Peter Willett. "Maximum common subgraph isomorphism algorithms for the matching of chemical structures". In: *Journal of Computer-Aided Molecular Design* 16.7 (2002), pp. 521–533.

[157] Ievgen Redko, Amaury Habrard, and Marc Sebban. "On the analysis of adaptability in multi-source domain adaptation". In: *Machine Learning* 108.8-9 (2019), pp. 1635–1652.

[158] Steffen Rendle. "Factorization machines". In: *2010 IEEE International Conference on Data Mining*. IEEE. 2010, pp. 995–1000.

[159] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. "BPR: Bayesian personalized ranking from implicit feedback". In: *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press. 2009, pp. 452–461.

[160] Konrad Rieck and Pavel Laskov. "Linear-time computation of similarity measures for sequential data". In: *Journal of Machine Learning Research* 9 (2008), pp. 23–48.

[161] David Rogers and Mathew Hahn. "Extended-connectivity fingerprints". In: *Journal of Chemical Information and Modeling* 50.5 (2010), pp. 742–754.

[162] Eugene Rogot and Irving D Goldberg. "A proposed index for measuring agreement in test-retest studies". In: *Journal of Clinical Epidemiology* 19.9 (1966), pp. 991–1006.

[163] Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. "To transfer or not to transfer". In: *NIPS 2005 workshop on transfer learning*. Vol. 898. 2005, pp. 1–4.

[164] Sascha Rothe and Hinrich Schütze. "Autoextend: Extending word embeddings to embeddings for synsets and lexemes". In: *arXiv preprint arXiv:1507.01127* (2015).

[165] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* 290.5500 (2000), pp. 2323–2326.

[166] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The earth mover's distance as a metric for image retrieval". In: *International journal of computer vision* 40.2 (2000), pp. 99–121.

[167] Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. "Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17". In: *Journal of Chemical Information and Modeling* 52.11 (2012), pp. 2864–2875.

[168] Sebastian Ruder. "An overview of multi-task learning in deep neural networks". In: *arXiv preprint arXiv:1706.05098* (2017).

[169] Olga Russakovsky, Jia Deng, Hao Su, et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[170] Itay Safran and Ohad Shamir. "On the quality of the initial basin in overspecified neural networks". In: *International Conference on Machine Learning*. 2016, pp. 774–782.

[171] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. *Application of dimensionality reduction in recommender system-a case study*. Tech. rep. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[172] Bernhard Schölkopf, Sebastian Mika, Chris JC Burges, et al. "Input space versus feature space in kernel-based methods". In: *IEEE Transactions on Neural Networks* 10.5 (1999), pp. 1000–1017.

[173] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear component analysis as a kernel eigenvalue problem". In: *Neural Computation* 10.5 (1998), pp. 1299–1319.

[174] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[175] Matthew Schultz and Thorsten Joachims. "Learning a distance metric from relative comparisons". In: *Advances in Neural Information Processing Systems*. 2004, pp. 41–48.

[176] Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus-Robert Müller, and Alexandre Tkatchenko. "Quantum-chemical insights from deep tensor neural networks". In: *Nature communications* 8 (2017), p. 13890.

[177] Kristof T Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, et al. "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions". In: *Advances in Neural Information Processing Systems*. 2017, pp. 991–1001.

[178] Kristof T Schütt, Huziel E Sauceda, Pieter-Jan Kindermans, Alexandre Tkatchenko, and Klaus-Robert Müller. "SchNet–A deep learning architecture for molecules and materials". In: *The Journal of Chemical Physics* 148.24 (2018), p. 241722.

[179] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.

[180] Justin S Smith, Benjamin T Nebgen, Roman Zubatyuk, et al. "Approaching coupled cluster accuracy with a general-purpose neural network potential through transfer learning". In: *Nature communications* 10.1 (2019), pp. 1–8.

[181] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. "Reasoning with neural tensor networks for knowledge base completion". In: *Advances in Neural Information Processing Systems*. 2013, pp. 926–934.

[182] Robert R Sokal, Peter HA Sneath, et al. "Principles of numerical taxonomy." In: *Principles of numerical taxonomy.* (1963).

[183] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. "The implicit bias of gradient descent on separable data". In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 2822–2878.

[184] Nathan Srebro and Tommi Jaakkola. "Weighted low-rank approximations". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 720–727.

[185] Abby Stylianou, Richard Souvenir, and Robert Pless. "Visualizing Deep Similarity Networks". In: *arXiv preprint arXiv:1901.00536* (2019).

[186] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press, 2012.

[187] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. "Covariate shift adaptation by importance weighted cross validation". In: *Journal of Machine Learning Research* 8.May (2007), pp. 985–1005.

[188] Yee W Teh and Sam T Roweis. "Automatic alignment of local representations". In: *Advances in Neural Information Processing Systems*. 2003, pp. 865–872.

[189] Joshua B Tenenbaum, Vin De Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *Science* 290.5500 (2000), pp. 2319–2323.

[190] Naftali Tishby, Fernando C Pereira, and William Bialek. "The information bottleneck method". In: *arXiv preprint physics/0004057* (2000).

[191] EF Tjong, Kim Sang, and F De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition". In: *Proceedings of CoNLL-2003*. Ed. by Walter Daelemans and Miles Osborne. Edmonton, Canada, 2003, pp. 142–147.

[192] Roberto Todeschini, Viviana Consonni, Hua Xiang, et al. "Similarity coefficients for binary chemoinformatics data: overview and extended comparison using simulated and real data sets". In: *Journal of chemical information and modeling* 52.11 (2012), pp. 2884–2901.

[193] Kristina Toutanova, Danqi Chen, Patrick Pantel, et al. "Representing Text for Joint Embedding of Text and Knowledge Bases". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015.

[194] Andrew Trask, Phil Michalak, and John Liu. "sense2vec-A fast and accurate method for word sense disambiguation in neural word embeddings". In: *arXiv preprint arXiv:1511.06388* (2015).

[195] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. "Complex Embeddings for Simple Link Prediction". In: *International Conference on Machine Learning*. 2016.

[196] Joseph Turian, Lev Ratinov, and Yoshua Bengio. "Word representations: a simple and general method for semi-supervised learning". In: *Proceedings of the 48th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 384–394.

[197] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. "Simultaneous deep transfer across domains and tasks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4068–4076.

[198] Vladimir N Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995.

[199] Andreas Veit, Serge Belongie, and Theofanis Karaletsos. "Conditional similarity networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 830–838.

[200] Rene Vidal, Joan Bruna, Raja Giryes, and Stefano Soatto. "Mathematics of deep learning". In: *arXiv preprint arXiv:1712.04741* (2017).

[201] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103.

[202] Kilian Q Weinberger and Lawrence K Saul. "Distance metric learning for large margin nearest neighbor classification". In: *Journal of Machine Learning Research* 10.Feb (2009), pp. 207–244.

[203] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, and Alexander J Smola. "Joint training of ratings and reviews with recurrent recommender networks". In: *ICLR 2017 Workshop Track*. 2017.

[204] Hao Wu, Zhengxin Zhang, Kun Yue, et al. "Dual-regularized matrix factorization with deep neural networks for recommender systems". In: *Knowledge-Based Systems* 145 (2018), pp. 46–58.

[205] Ledell Wu, Adam Fisch, Sumit Chopra, et al. "StarSpace: Embed All The Things!" In: *arXiv preprint arXiv:1709.03856* (2017).

[206] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. "Distance metric learning with application to clustering with side-information". In: *Advances in Neural Information Processing Systems*. 2003, pp. 521–528.

[207] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. "Deep Matrix Factorization Models for Recommender Systems". In: *IJCAI*. 2017, pp. 3203–3209.

[208] Yoshihiro Yamanishi, Masaaki Kotera, Minoru Kanehisa, and Susumu Goto. "Drug-target interaction prediction from chemical, genomic and pharmacological data in an integrated framework". In: *Bioinformatics* 26.12 (2010), pp. i246–i254.

[209] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. "Embedding Entities and Relations for Learning and Inference in Knowledge Bases". In: *International Conference on Learning Representations*. 2015.

[210] Muhammed A Yıldırım, Kwang-Il Goh, Michael E Cusick, Albert-László Barabási, and Marc Vidal. "Drug—target network". In: *Nature Biotechnology* 25.10 (2007), p. 1119.

[211] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems*. 2014, pp. 3320–3328.

[212] Aron Yu and Kristen Grauman. "Fine-grained visual comparisons with local learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 192–199.

[213] Aron Yu and Kristen Grauman. "Semantic Jitter: Dense Supervision for Visual Comparisons via Synthetic Images". In: *International Conference on Computer Vision (ICCV)*. Oct. 2017.

[214] Amir R Zamir, Alexander Sax, William Shen, et al. "Taskonomy: Disentangling task transfer learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3712–3722.

[215] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530* (2016).

[216] Hui Zhang, Huaping Liu, Rui Song, and Fuchun Sun. "Nonlinear non-negative matrix factorization using deep learning". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016, pp. 477–482.

[217] Wei Zhang, Yuchun Fang, and Zhengyan Ma. "The Effect of Task Similarity on Deep Transfer Learning". In: *International Conference on Neural Information Processing*. Springer. 2017, pp. 256–265.

[218] Yaoyu Zhang, Zhi-Qin John Xu, Tao Luo, and Zheng Ma. "A type of generalization error induced by initialization in deep neural networks". In: *arXiv preprint arXiv:1905.07777* (2019).

[219] Xiaodong Zheng, Hao Ding, Hiroshi Mamitsuka, and Shanfeng Zhu. "Collaborative matrix factorization with multiple similarities for predicting drug-target interactions". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2013, pp. 1025–1033.

Similarity Encoder

A Neural Network Architecture for Learning Similarity Preserving Embeddings

Matrix factorization is at the heart of many machine learning algorithms, for example, for dimensionality reduction (e.g. kernel PCA) or recommender systems relying on collaborative filtering. Understanding a singular value decomposition (SVD) of a matrix as a neural network optimization problem enables us to decompose large matrices efficiently while dealing naturally with missing values in the given matrix. But most importantly, it allows us to learn the connection between data points' feature vectors and the matrix containing information about their pairwise relations. In this thesis, we introduce a novel neural network architecture termed *Similarity Encoder* (SimEc), which is designed to simultaneously factorize a given target matrix while also learning the mapping to project the data points' feature vectors into a similarity preserving embedding space. This makes it possible to, for example, easily compute out-of-sample solutions for new data points. Additionally, we demonstrate that SimEcs can preserve non-metric similarities and even predict multiple pairwise relations between data points at once. As the first part of the SimEc architecture, mapping from the original (high dimensional) feature space to the (low dimensional) embedding, can be realized by any kind of (deep) neural network, SimEcs can be used in a variety of application areas. As we will demonstrate, SimEcs can serve as a reliable baseline model in pairwise relation prediction tasks such as link prediction or for recommender systems. The pairwise relations and similarities predicted by a SimEc model can also be explained using layer-wise relevance propagation (LRP). Furthermore, SimEcs can be used to pre-train a neural network used in a supervised learning task, which, for example, improves the prediction of molecular properties when only few labeled training samples are available. Finally, a variant of SimEc, called Context Encoder (ConEc), provides an intuitive interpretation of the training procedure of the CBOW word2vec natural language model trained with negative sampling and makes it possible to learn more expressive embeddings for words with multiple meanings as well as to compute embeddings for out-of-vocabulary words.

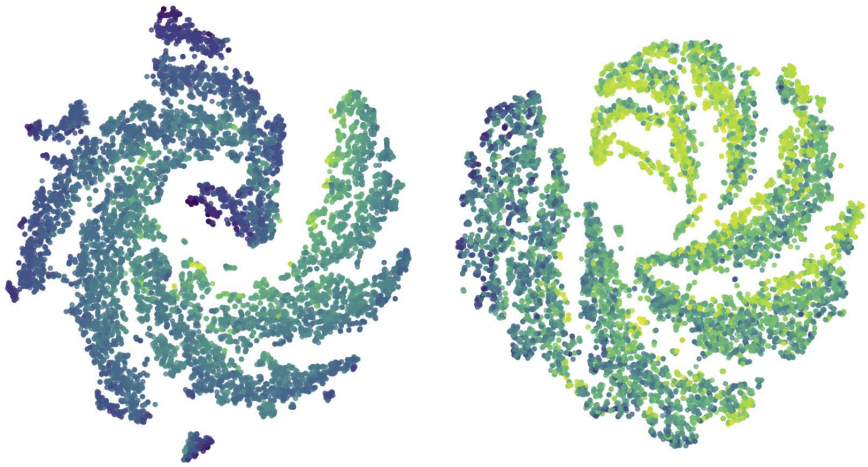


Figure 0: T-SNE visualizations of 10k molecules' 128-dimensional SchNet embeddings. The SchNet architecture was pre-trained with a SimEc trained to approximate the Rogot-Goldberg molecular fingerprint similarity (aggregation: sum, normalization: max) and then the network was fine-tuned to predict the energy U0 of 100 training molecules. Each dot represents one molecule. In the plot on the left, the individual SchNet atom embeddings of a molecule were averaged to compute the final molecule embedding and the points are colored based on the molecules' atomization energy. In the plot on the right, the atom embeddings were summed up and the dots are colored based on the molecules' LUMO values. Further details can be found in Chapter 6.