universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Learning Low Dimensional Representations for K-Means with K-Competitive Autoencoders“

verfasst von / submitted by

Lukas Miklautz, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik UG2002
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Inform.Univ. Dr. Claudia Plant

Contents

List of Figures

List of Tables

List of Abbreviations

ACC	Clustering **ACC**uracy
AE	**A**uto**E**ncoder
FFN	**F**eed **F**orward **N**etwork
DCN	**D**eep **C**lustering **N**etwork
DEC	**D**eep **E**mbedded **C**lustering
IDEC	**I**mproved **D**eep **E**mbedded **C**lustering
KATE	**K**-competitive **A**utoencoder for **TE**xt
NMI	**N**ormalized **M**utual **I**nformation
PCA	**P**rincipal **C**omponent **A**nalysis
relu	**re**ctified **l**inear **u**nit
ROC-AUC	**R**eceiver **O**perating **C**haracteristic - **A**rea **U**nder the **C**urve
sig	**sig**moid function
tanh	**tan**gens **h**yperbolicus function
WTA	**W**inner **T**akes **A**ll

1 Introduction

1.1 Motivation

Over the last decade Data Mining and Machine Learning have been the center of attention. Especially, the hype around Deep Learning has started a new Spring in artificial intelligence research. This was due to impressive results for supervised learning over a range of tasks for image [7], [38], text [59] and speech data [22], but interesting applications in unsupervised learning as well [17], [28], [60].

The success of deep neural networks depends on their ability to learn a distributed hierarchy of features, often called representation. To illustrate this, assume the following example of a neural network with three layers and a data set containing pictures of faces. A learned hierarchy from the first to the last layer would go from simple to more abstract ones. E.g. the first layer could contain features of edges, the second layer combines those to construct higher level features like mouths, ears, noses and the third layer could then consist of different faces. In this sense, the job of a neural network could be summarized as automated feature engineering. Thats, why deep learning is particularly useful for image, text and speech data, where it is difficult for humans to engineer suitable features.

In contrast to the hype surrounding deep learning, clustering is one of the most fundamental challenges in data mining and machine learning and has been studied for several decades. Intuitively, clustering is the task of grouping similar objects together. An example are recommender systems at online shops, in which customers with similar buying behavior get similar recommendations. In times of big data, traditional clustering algorithms like k-means [34] or DBSCAN [14], face new challenges. These challenges called for innovation, which has lead to several improvements and new algorithms, see [30] for an overview. All these improvements have stayed mostly secluded from the deep learning framework and only recently there is research trying to combine them. The idea is that unsupervised neural networks, so called autoencoders [64], can learn a low dimensional and meaningful representation of the input data, which in turn can help to improve clustering results. In the last years there have been several approaches to combine k-means with deep learning, which showed promising results, see [18], [62], [63] for some of the most prominent ones.

1.2 Problem Statement

All the methods mentioned in [18], [62], [63] are based on the autoencoder architecture which serves as a non-linear dimensionality reduction technique for the input data. The basic autoencoder is quite prone to overfitting which is why [18], [62], [63] all use variants of it. There are several architectures which are used in the literature and all implement some kind of regularization scheme, e.g. the denoising autoencoder [60],

contractive autoencoder [53] or k-sparse autoencoder [40]. None of them works best in all cases and each of them has benefits and disadvantages. That is why they should be chosen according to the data and the desired application.

Many of the existing autoencoder techniques have only been evaluated for image data sets [28], [40], [41], [53], [60]. Especially, text data has been a difficult scenario for existing autoencoders, due to its high dimensionality and sparsity [12]. This originates from the intuitive relationship between potentially large language vocabularies and sparse document vectors, which may contain only a few words of the available language. A recent autoencoder variant, called KATE, k-competitive autoencoder for text [12], has achieved promising results on text data. Similar to k-sparse and wta autoencoders [40], [41] it uses a competition layer, but includes an energy redistribution among winners [1].

There are many data sets which share characteristics with text data sets. Some of these include sparsity of relevant information, many noise points, redundant features or outliers. For a selection of relevant real world data sets with outliers, see e.g. [9]. For all of these a basic autoencoder would probably not be able to learn a meaningful representation. That is why this thesis adapts the KATE algorithm to other data besides text. The key idea is that its benefits will transfer to the above mentioned settings and potentially improve the clustering results.

1.3 Thesis Focus and Key Contributions

The existing techniques which combine deep learning and clustering follow a two step approach [18], [62], [63]. In the first step an autoencoder variant is trained on the input data. The learned low dimensional representation is then clustered via k-means. The second step differs for each method, but in general it can be thought of as a step wise refinement of the initial clustering. That's expressed in the fact that points in a cluster are slowly drawn closer to their center. Without going into the details, it can already be seen that the first step is crucial for the final results. In particular, if the used autoencoder architecture fails to learn a good representation of the input data, the clustering could become meaningless.

Thats the reason, why in this master thesis the focus is to improve this first step of learning a meaningful low dimensional representation. This is a huge topic in machine learning and is summarized under the terms "representation learning" or "feature learning" [6]. To further limit the scope, the emphasis will be on cross sectional data sets with outliers, noise and redundant information. Additionally, in representation learning a found representation is considered as good if it is beneficial for some other downstream task like classification or clustering. Here, this task will be clustering with a partitional algorithm like k-means.

The key contributions of this master thesis are the following. First, the specific effects of the competition scheme used in KATE will be studied via various synthetic and real world data experiments. Second, from these empirical insights the KATE algorithm which has by now only been successfully used on text data [12], will be adapted and evaluated for other data types. Third, the adapted algorithm will be benchmarked

[1] A detailed description of this mechanism can be found in the literature review section.

on multiple challenging real world data sets, against state of the art autoencoder architectures.

1.4 Thesis Outline

Chapter 2 provides a brief review of standard data mining and machine learning techniques. The focus will be on unsupervised machine learning and some of the main algorithms are introduced. At first the fundamental notion of cluster analysis will be explained, together with the popular k-means algorithm. After that, the idea of dimensionality reduction is presented using the example of principal component analysis.

Chapter 3 gives an overview of the history of neural networks and talks about the recent progress in deep learning. The concept of a feed forward neural network and its key components will be discussed. This will serve as the basis to understand the autoencoder algorithm, which is the focus in this thesis.

Chapter 4 reviews the topic of representation learning. It includes an overview of the field and its challenges. After that a literature review of the most popular and latest techniques in the field of representation learning is conducted. The literature review contains different autoencoder variants like denoising, stacked or variational autoencoders. Additionally, autoencoders which include clustering are discussed, namely deep embedded clustering and deep clustering network.

Chapter 5 is discusses all experiments conducted in this thesis. First, the motivation for the specific experiment settings is discussed. After that, the strategy for finding an optimal use of the k-competitive layer in autoencoder architectures is explained, together with the generation of the synthetic data sets. The experiments are then conducted in the following order. First, the k-competitive layer is evaluated without any non-linear activation functions, to better understand its particular effects. After that different experiments are conducted to find a suitable non-linear activation function. The robustness to noise is afterwards evaluated with different synthetic noise data sets. The last section includes experiments with different real world data sets with outliers.

Chapter 6 concludes the master thesis. It summarizes all found results and discusses the findings in a broader picture. Possible future works will be discussed as well as short and long term goals.

2 Data Mining

2.1 Overview

Data Mining, in a general definition by Leskovec et al refers to the "discovery of models for data" [33]. According to this interpretation there are several topics which can be summarized under the term of "model". From the statistical modeling perspective it can be seen as the construction of a "statistical model", which is the underlying data generating distribution [33]. Data mining as "summarization" [33] refers to the task of finding similar groups in data and represent this group with e.g the cluster centroids.

The machine learning view is another perspective, which is often used synonymously with data mining [33]. Murphy [45] defines "machine learning as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data,[...]". In general, the lines between machine learning and data mining are blurry, but often machine learning is seen as a subset of data mining. All these perspectives serve as an attempt to describe the term data mining.

One broad categorization which is often done in data mining and machine learning is to distinguish between supervised and unsupervised learning problems [25]. Supervised learning refers to the situation in which for each observation x_i there exists a corresponding label y_i where $i = 1, ..., n$. The task is then to find a model which relates the observations to the label with the goal of predicting the label for new and unseen data. Unsupervised learning on the other hand only uses unlabeled data x_i. Here, the focus is to find out more about the underlying structure of the data. Some questions that could be answered with unsupervised learning are e.g. what groups of similar objects can be found in the data, how was the data generated or what are the most important features of the data?[1] In the following, two fundamental research directions from the unsupervised learning branch will be discussed, namely clustering and dimensionality reduction.

2.2 Clustering

Intuitively, clustering can be understood as a method that finds groups with similar characteristics in data. A more technical definition is given by Jain [24], who describes cluster analysis as "the formal study of methods and algorithms for grouping, or clustering, objects according to measured or perceived intrinsic characteristics or similarity". In this definition there are a few terms which need to be further explained. A cluster is a group of objects which share a high degree of similarity within that group. The "size, shape and density" [24] of clusters can be different for each of them, e.g. see figure 5.1 for an example of four Gaussian clusters in two dimensions. Here all four

[1] For an introduction to the topics of supervised and unsupervised learning, see [25]

clusters have the same size, shape and density and only differ in their location. According to [24], "an ideal cluster can be defined as a set of points that is compact and isolated". Here compactness refers to a high degree of within cluster similarity and "isolation" to a high degree of dissimilarity between objects of different clusters. It is often hard to find an ideal clustering, especially because it is possible that there are multiple truths. For example assume a data set with apples and pears. If one is only looking at color, there might be red, green and yellow apples, but green and yellow pears as well. In contrast, when considering the shape only, there might be distinct groups for apples and pears.[2] In practice it is common to consider the opinion of a domain expert, to find out the "significance and interpretation" [24] of a cluster. This already highlights the challenges of finding the measures of "similarity" for objects, which highly depends on the chosen features. Another consideration is how to measure this similarity, while this highly depends on the used algorithms it is often some distance metric, e.g. for k-means the default choice is the euclidean distance [24].

The three main purposes of clustering identified by Jain [24] are first, to gain insight in the data by finding its "underlying structure". Second, "to identify the degree of similarity among" objects. Third, as a technique of compression, by "organizing the data and summarizing it through cluster prototypes". These three points highlight the usefulness of clustering for the scientific community and illustrate why the development of clustering methodology was an "interdisciplinary endeavor" [24]. In the next section, the k-means clustering algorithm is introduced.

2.2.1 K-Means

The k-means algorithm is one of the most popular and simplest clustering techniques. Despite its simplicity it performs quite well in many different scenarios. K-means has been "independently discovered in different scientific fields" [24], among which two publications are [34], [37]. It belongs to the group of partitional methods, which means that it finds all "clusters simultaneously as a partition of the data" and does not "impose a hierarchical structure" [24].

Given a data set X, which consists of n data points x_i, the k-means algorithm will find K clusters C_k, where $k = 1, ..., K$. The objective of k-means "is to minimize the sum of the squared error over all K clusters", where the squared error is calculated with the empirical mean, also called centroid of a cluster μ_k and all the points in that cluster [24]. This is put together in equation 2.1:

$$L(C) = \Sigma_{k=1}^{K} \Sigma_{x_i \epsilon C_k} \|x_i - \mu_k\| \tag{2.1}$$

The k-means algorithm works in three steps [24]. First, K clusters are selected, which serve as an initial partition. Second, a new partition is generated by assigning each point to its closest centroid. Third, new cluster centers are calculated via the empirical mean of the cluster members. The second and third step are repeated until convergence. Note, that this is a greedy procedure. This means that k-means is only able to find local minima and should be run several times with different initializations.

[2] For a detailed discussion of how to handle multiple truths in clustering, see [67].

The basic k-means algorithm requires three parameters, the number of clusters K, the distance metric and the initial partitioning. Its default distance metric is the euclidean metric, which causes k-means to "find spherical or ball-shaped clusters in data" [24]. Other metrics are the Mahalanobis distance [42] or Manhattan distance. To choose the optimal value of K, a number of heuristics exist, from which the simplest is to just run k-means with different values of K and choose the one with the lowest error [24]. There are several extensions of k-means, e.g. k-means++[2] or X-Means [51]. X-Means chooses the number of K automatically via a model complexity criterion like Akaike Information or Bayesian Information. K-means++ uses a random seeding strategy for the initialization step and can achieve higher speedup and accuracy.

2.3 Dimensionality Reduction

Dimensionality reduction, as the name indicates, refers to a group of techniques which aim to find a lower dimensional projection from a higher dimensional data space [45]. Intuitively, dimensionality reduction aims to "capture the essence of the data" [45]. The notion behind this set of techniques is that, despite the high dimensionality of the input, there may be only a few "degrees of variability, corresponding to latent factors" [45] in the data. E.g. for natural images some latent factors might be lighting, pose or identity. This means that most of the "interesting" information is only in a small subspace.

Dimensionality reduction techniques can be used as preprocessing step, before other downstream tasks like clustering or classification are applied[3]. Another application is visualization, in which a high dimensional space is projected down to two or three dimensions. Below, one of the most popular dimensionality reduction techniques, called principal component analysis, will be discussed.

2.3.1 Principal Component Analysis

Principal Component Analysis [49], PCA, is one of the most common dimensionality reduction techniques [45]. Put simply, PCA "looks" for the directions of most variance, in that it only concentrates on the most "interesting" features. A typical setting for PCA is a data set with many correlated variables, in which it summarizes the data in a lower dimensional space. By definition of PCA this lower dimensional subspace captures most of the variability of the original data set [25].

Given a data set X, which consists of n d-dimensional data points x_i, the steps to perform PCA are first, to compute the covariance matrix of X, called Σ. Second, compute the eigenvalues and and eigenvectors of Σ. Third, select the k biggest eigenvalues and their eigenvectors. The choice of k is left to the user and is often made on the basis of the percentage of explained variance[4]. Last, the original data set X is transformed via the matrix of selected eigenvectors V, which results in $\hat{X} = X * V$. Each feature in this new space $\hat{X}$ is a linear combination of the d original features.

[3]For a discussion why a global dimensionality reduction technique, like PCA, for high dimensional data might not always be a good solution, see [30]

[4]The percentage of explained variance is just the ratio of the sum of the k selected eigenvalues and the sum of all eigenvalues.

Due to this, PCA is a linear dimensionality reduction technique and faces some limitations, e.g. when it comes to linearly inseparable data. There are several extensions of PCA, e.g. Kernel-PCA [56], which projects the data into a higher dimensional space where it is linearly separable and then applies PCA. The downside for this is that the type of kernel has to be chosen by the user. Another approach are autoencoders [16], which can learn a kernel function from the data, but are more complicated to train. Under some restrictions it can be shown that a linear autoencoder basically performs PCA, for a prove see [5].

3 Neural Networks

3.1 Overview

Deep learning is one of todays hot topics and is already widely known beyond the research community. This was due to impressive results for supervised and unsupervised learning algorithms for image [7], [17], [28], [38], [60], text [59] and speech data [22]. Despite everything one would think with the current hype, deep learning is not a new topic. Actually, it has been researched under the name "cybernetics" in the 1940s-1960s, "connectionism" in the 1980s-1990 and only since 2006 it is known as "deep learning" [16]. These first two resurgences have both ended in a research winter. The cybernetics era ended with the now famous XOR problem and critique by Minsky [43] and the connectionism hype came to a halt due to computational limitations and the lack of available data [16]. Today, most of these issues are gone, the XOR problem has been solved by multilayer perceptrons [16], the computational limitations have been alleviated by moore's law [44] and Big Data took care of the rest. To understand the inner workings of deep learning algorithms, the next section introduces the simplest form of an artificial neural network, the feed forward neural network.

3.2 Feed Forward Network

The feed forward[1] neural network, FFN, often called deep feed forward network or multilayer perceptron, is the basic model for all other advanced deep learning techniques [16] . Intuitively, FFN's are nothing more than "function approximation machines" [16], which try to learn a mapping $y = f(x, W)$. Here, x is the input data, W are the learnable parameters and y is the target output, which could be a label in classification or some number in regression tasks. For unsupervised networks, like the autoencoder, y would be a reconstruction of the input x, denoted as $\hat{x}$.

A FFN can be thought of as a non-linear version of a generalized linear model [8] similar to equation 3.1.

$$y(x, w) = f(\Sigma_i w_i g(x)) \tag{3.1}$$

where w is the weight parameter vector and g is a possibly non-linear composition of functions. The layers and neurons of a FFN can be described by such a composition. Let $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ be three functions, than a typical composition for a FFN looks like $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ [16]. Here, $f^{(1)}$ is the first layer connecting the input x to the second layer $f^{(2)}$, which then connects to the third layer $f^{(3)}$. The last layer, in this case $f^{(3)}$, is also called output layer. All layers between the input and output layer are called

[1] The models are called "feed forward" due to the way the information flows form the input x to the output y, without any feedback connections [16].

hidden layers, in this case there is only on hidden layer $f^{(2)}$. The number of functions in the composition is called the depth, from which the term "deep learning" arises [16]. Figure 3.1 depicts the directed acyclic graph of a FFN, where x is the D-dimensional input vector, z is a hidden layer with M neurons and $\hat{x}$ is the output vector with K neurons. The weight matrix $W^{(1)}$ connects the input layer to the hidden layer and $W^{(2)}$ connects the hidden layer to the output layer. x_0 and z_0 are bias variables. The term neuron refers to one value in the hidden layer vector. The number of neurons in one hidden layer refers therefor to its dimensionality and "the width of the model" [16]. The layers can be seen as vector to vector functions or as vector to scalar functions. The latter is called a unit, which takes in all the previous units and calculates an activation value for one of the neurons in the next layer.

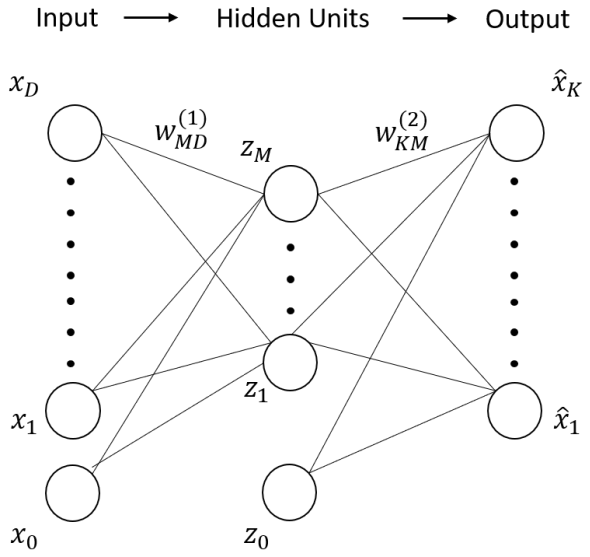


FIGURE 3.1: Directed acyclic graph of a fully connected feed forward neural network. Figure adapted from [8].

The activation value is a measure of how "excited" the neuron is for seeing a specific input. There are many different activation functions used today, e.g. the (leaky) rectified linear unit, the tangens hyperbolicus function or the sigmoid function, see figure 5.10 for a plot. Activation functions are motivated through biological neurons [16], which can either fire or not, given some input. In some sense, activation functions try to approximate this behavior with a differentiable function[2]. Equation 3.2 shows the computation of one hidden unit's activation z_j.

$$z_j = g(\Sigma_{i=1}^{D} w_{ji}^{(1)} x_i + x_0 w_{j0}^{(1)}) \tag{3.2}$$

where $w_{ji}^{(1)}$ is the weight parameter that connects the input vector x with hidden unit z_j and $x_0 w_{j0}^{(1)}$ is a bias term with its weight, which allows the whole function to shift. Now putting all things together, the equation for the FFN with one hidden layer in figure 3.1 is shown in 3.3.[3]

[2]To be specific the function only needs to have defined left and right derivatives, e.g. the rectified linear function's derivative at point 0 is undefined [16].

[3]This equation has been adapted from [8].

$$\hat{x}_K(X, W) = o(\Sigma_{j=1}^{M} w_{kj}^{(2)} g(\Sigma_{i=1}^{D} w_{ji}^{(1)} x_i + x_0 w_{j0}^{(0)}) + z_0 w_{k0}^{(2)}) \tag{3.3}$$

where g is the activation function of the hidden layer, o is the output function of the output layer, x_0 and z_0 are bias terms. After introducing the basics, the next question is how should the weight parameters be set?

Learning in FFN's works via gradient based optimization of a loss function. Intuitively, the loss function measures how far off the FFN was with its prediction. One of the most common ones is the mean squared error, see equation 3.4.

$$J(W) = \Sigma_{i=1}^{N} (y_i - f(x_i))^2 \tag{3.4}$$

The value of the loss serves as a signal for the FFN to update its weights and to improve its performance. The goal is then to minimize the loss, between the prediction $\hat{y} = f(x)$ of a FFN and the ground truth y. The update of the weight parameters is done via gradient descent [10]. The error gradients are efficiently computed via the backpropagation algorithm [55][4]. Equation 3.5 shows an update step for a batch gradient descent method [8].

$$W_{t+1} = W_t - \eta \cdot \Delta J(W_t) \tag{3.5}$$

$\Delta J(W_t)$, refers here to the gradient of the loss function at time t and η is the step size, often also called the learning rate. Equation 3.5 shows a **batch** gradient update, this means it uses the whole training data to compute the gradient. This is often not feasible due to speed and memory limitations. Thats why it is more common to use stochastic gradient descent [16], which performs a parameter update for each training example x_i, see equation 3.6.

$$W_{t+1} = W_t - \eta \cdot \Delta J(W_t, x_i) \tag{3.6}$$

Unfortunately, this approach has some downsides as well as it often results in noisy and unstable training. To counteract this, the learning rate η has to be set very small, which in turn increases the runtime [16]. That's why minibatch training is a compromise between those two extremes. It is a common method to make training more efficient and less noisy. It uses only a small subset of training examples, a minibatch, for each step to compute the gradient. For a discussion of possible trade offs, see chapter 8.1.3 of [16]. These are the basics for gradient descent methods, but almost all others build upon those. There exists a huge plethora of literature on nonlinear optimization procedures and it would be out of scope to discuss them all here, but some popular ones are Adam [27], Adagrad [13] or Adadelta [66]. After introducing the basics of FFN's an unsupervised neural network, called autoencoder will be explained.

3.3 Autoencoder

The autoencoder is an unsupervised neural network, which has been researched since the 80ies [64]. Autoencoders are a special case of feed forward neural networks, that are "trained to attempt to copy its input to its output" [16]. The aim of it is not to

[4]See Chapter 5.2 of [8] for a detailed introduction to backpropagation.

simply copy its input, but to learn an encoding in the hidden layer z, which captures the most important parts of the data. Among other things, it is used for dimensionality reduction, visualization or representation learning.

The autoencoder consists of two parts, the encoder and the decoder. The encoder projects the input x to the hidden layer, which generates the encoding $z = g(\Sigma_{i=1}^{D} w_{ji}^{(1)} x_i + x_0 w_{j0}^{(0)})$. The decoder then takes z and tries to reconstruct the original input x from it, which produces the decoding $\hat{x} = o(\Sigma_{j=1}^{M} w_{kj}^{(2)} z + z_0 w_{k0}^{(2)})$. The reconstruction $\hat{x}$ is then compared to the original with a loss function. For $K = D$ figure 3.1 is an autoencoder, the encoder part is the left side from input to hidden layer and the decoder part is the right side from hidden layer to output.

There are different categorizations for this "vanilla" autoencoder. It is called overcomplete if the hidden layer has a higher dimension than the input vector. An undercomplete autoencoder refers to the situation in which the hidden layer has a lower dimension than x, which creates a bottleneck during training. This setting constraints the autoencoder to capture only the most important features of the training data [16]. It is symmetric if the depth and width of the encoder network is the same as for the decoder network, otherwise it is called asymmetric.

The training is similar to the FFN, except that the loss function is adapted. Instead of comparing some label to the predicted output, the input is compared to its reconstruction. The mean squared error now looks like equation 3.7.

$$J(W) = \Sigma_{i=1}^{N} (x_i - \hat{x})^2 \tag{3.7}$$

For autoencoders, but for FFN's as well the mean squared error is used for real valued outputs. For binary outputs the cross entropy loss is used, see equation 3.8 for the autoencoder version.

$$J(W) = -\sum_{i} (x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)) \tag{3.8}$$

The binary cross entropy loss is minimized if $\hat{x}_i$ is close to x_i and it reaches perfect reconstruction with $x_i = \hat{x}_i$.

Compared to principal component analysis or kernel PCA, the autoencoder is able to learn non-linear projections in the hidden layer from the data, without explicit assumptions for the kernels. Unfortunately, the vanilla autoencoder is prone to overfitting, this is particularly true in the overcomplete setting. Much of autoencoder research has therefor concentrated on creating architectures which implement regularization schemes. One simple countermeasure, that is used in almost all autoencoder architectures, is called weight tying. It refers to the practice of reusing the transposed encoder weights for the decoder network. This reduces the number of parameter and increases its training speed. More sophisticated approaches are using new architectures or created loss functions which enforce specific properties, but the basic idea is the same for most of them. They try to complicate the training process for the autoencoder so that it is forced to learn a meaningful representation of the data. Some of these approaches are discussed in chapter 4.

4 Representation Learning

4.1 Overview

The success or failure of a machine learning model depends heavily on the representation of the input data. In practice this manifests in the fact that most effort in deploying a data mining algorithm goes into data gathering, preprocessing and feature engineering, while the importance of the used learning algorithm is often negligible. Representation learning, also called feature learning, is about "learning representations of the data that make it easier to extract useful information when building classifiers or other predictors" [6]. The goal of representation learning is to automate or augment the feature engineering process, especially in cases were human engineered feature may be insufficient or need expert knowledge. In this literature review the focus lies on unsupervised deep learning based methods to automatically extract "good" features from the input data.

4.2 Challenges

In contrast to supervised learning, representation learning does not have a clear objective, except for the elusive criteria of improving classification or clustering results later on. In a recent survey Bengio, Courville, and Vincent [6] propose several criteria for good representations that might be beneficial for a variety of machine learning tasks. Some of them are discussed below.

"Expressiveness" [6], refers to the capability of a low dimensional representation, at least lower than the input dimension, to capture a huge number of possible input configurations. "Depth and abstraction", these two concepts summarize the idea that deep learning facilitates the learning of an abstraction hierarchy, which represents low level features on one end and high level ones on the other. The general assumption is thus, that deeper networks can capture more abstract concepts, which are "generally invariant to most local changes of the input" [6]. Bengio, Courville, and Vincent [6] argue that despite the beneficial nature of the above criteria, they have some unwanted effects, e.g. invariant features are defined as being invariant to local changes that are not useful for a specific task, which leads to a loss of information. Thats why [6] argue that the most robust procedure for representation learning is the disentanglement of factors of variation. The advantage of disentanglement is that information is preserved and the underlying factors of the input data are found. This leads to the question of how to achieve disentanglement?

One solution might be to enforce "general-purpose priors" [6], that are beneficial for a wide range of tasks. A subset of such general-purpose priors are smoothness of the learned function, hierarchical organization of explanatory factors, shared factors

across tasks e.g. transfer learning, natural clustering of "named categories", sparsity of learned representation [47] and simplicity of factor dependencies.

All this research has to be taken with a grain of salt, because it is still an open question if downstream tasks profit from unsupervised representation learning in general. The competing field of transfer learning has a similar goal, but different approach. Instead of purely unsupervised learning, it uses neural networks which have been trained on a similar supervised learning problem and transfers its weight to the new neural network. This is based on the observation that transfered weights improve other neural networks, for some recent successes see [29], [39].

4.3 Literature Review

The following is a literature review of recent techniques used in representation learning with autoencoders. The goal here is to show on one hand a summary for different proposed autoencoder architectures and on the other to look at their usage in clustering frameworks. Remember from chapter 3, that most of the innovations in autoencoder architectures can be seen as clever regularization schemes to avoid overfitting. This is needed to prevent the normal autoencoder from learning the trivial identity function, which just copies the input data to the output layer[1].

4.3.1 Denoising Autoencoder

Denoising autencoders [60] extend the simple autoencoder by adding corruption noise to the input data and then try to reconstruct the original input. The corruption process used by the authors randomly sets k values of the input vector x_i to zero, resulting in $\tilde{x}_i$. The corrupted input is then fed to the autoencoder which tries to reconstruct the original vector x_i, by optimizing loss function 4.1, where $f(x)$ is the learned embedding and $g(f(\tilde{x}_i))$ is the reconstruction of the corrupted input $\tilde{x}_i$.

$$\mathcal{L}(\times) = L(x, g(f(\tilde{x}))) \tag{4.1}$$

The goal of the corruption process is not to learn a denoising function, but to regularize the autoencoder to learn a useful representation of the input function.

4.3.2 Stacked Autoencoder

A stacked autoencoder is very similar to a multilayer autoencoder. The only difference is the training procedure. Stacked autoencoders use a greedy layer-wise pre-training scheme [7], while normal autoencoders are trained with the complete architecture from the start. This scheme works in the following way, first train a one-layer-autoencoder which maps the input x_i to an encoding z_0 and then drop the decoder afterwards. The encoding z_0 is then used as input for a second one-layer-autoencoder, which is trained to produce a new encoding z_1 and again the decoder part is dropped. This step can be repeated until the desired architecture is created. After the stack has been trained, the last encoding z_n can be used for clustering or classification with a separate algorithm

[1] Additionally, these regularization techniques allow the training of an overcomplete autoencoder, in which the encoding has a higher dimension than the input data.

like k-means or Support Vector Machines [58]. The type of autoencoders used in each pre-training step can be chosen from the large pool of available architectures, e.g. denoising autoencoders [61]. It should be noted that in supervised learning the use of unsupervised pre-training has recently declined [16], but is still successfully used in deep clustering methods [18], [62], [63].

4.3.3 Contractive Autoencoder

Contractive autencoders [53] extend the vanilla autoencoder by using a penalty term on the second derivative of the reconstruction cost function. The introduced penalty term penalizes the sensitivity to a specific input x_i. The sensitivity is measured by the Frobenius norm of the Jacobian Matrix of the encoding $\|J_f(x)\|_F^2$. Intuitively this makes sense, as the first derivative specifies the slope of a function. In addition to the penalty term, the authors add the autoencoder reconstruction loss $L(x, g(f(x)))$ to avoid the trivial solution of a constant value. From this motivation the authors propose loss function 4.2, where $f(x)$ is the learned embedding, $g(f(x))$ the reconstruction of input x from the training data D_n and λ controls the regularization effect.

$$\mathcal{J}(\times) = \Sigma_{x \epsilon D_n}(L(x, g(f(x))) + \lambda \|J_f(x)\|_F^2) \tag{4.2}$$

4.3.4 Variational Autoencoder

The variational autoencoder [28] is a probabilistic variant of the autoencoder architecture. Instead of learning an encoding of an input x_i it learns a probability distribution from which latent variables z can be sampled. The latent variables are then used in the decoder network to generate (reconstruct) the input x_i. Besides of learning a hidden representation of the data, variational autoencoders enable the generation of new unseen samples after training. This works in the following way, the encoder parameterized as e.g. a fully connected neural network $q_\phi(z|x)$ learns the mean and the diagonal of the covariance matrix for sampling latent variables z from a Gaussian distribution $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$. The decoder network $p_\theta(x|z)$ takes the sampled z and learns the mean and diagonal of the covariance matrix for generating (reconstructing) the input x_i by sampling from a Gaussian distribution $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$. There are many techniques which have been build upon this framework e.g. [21], [52], [57]. While this is an interesting research direction for generative modeling, recent papers provide evidence, that variational autoencoders might not be the best choice for clustering [3], [11].

4.3.5 K-Sparse Autoencoder

The k-sparse autoencoder [40] is similar to a linear autoencoder and the only difference is the learning procedure. During training, the k-sparse autoencoder enforces population sparsity by only selecting the k largest activations in the hidden layer for each input feature and sets all other neurons to zero. The selection of the top k activations is the only non-linearity in the network. One issue that arises when training the k-sparse autoencoder is that it greedily assigns individual hidden units to training cases and

prevents other units from adjusting. This problem can be solved by a scheduling technique proposed in section 4.2.1 of [40]. After training, the α largest activation values are used, where α is a hyperparamater.

4.3.6 Winner-Take-All Autoencoder

A variant of the above mentioned k-sparse ([40]) technique is the winner-take-all (WTA) autoencoder [41] which introduces a lifetime sparsity across a minibatch of input features, by only keeping α percent of the highest activated neurons. Another difference to the k-sparse autoencoder is that the WTA autoencoder uses a non-symmetric architecture with relu [46] non-linearities and a linear decoder. This architecture removes the need of k-sparse for alpha scheduling during training and test time.

4.3.7 K-Competitive Autoencoder for Text

The k-competitive autoencoder for text (KATE) [12] is a recently proposed technique to learn a meaningful representation of text data. This has been a notoriously hard task for existing autoencoder architectures, due to the high dimensionality and sparsity of text data. Intuitively, this arises from the fact that language vocabularies are very large while a document vector may only contain a small percentage of the available words. [12] propose a shallow autoencoder architecture with one hidden layer, which performs competition between neurons. The competition is a form of regularization which keeps KATE from overfitting and allows it to learn a meaningful representation of the input data. The key idea behind this, is that similar to economic competition the neurons are specializing on specific patterns of the training data, which makes them distinct from each other and avoids trivial solutions, e.g. learning the identity function or overfitting to noise. During training only the k hidden neurons with the highest positive and negative activations z_i are selected and the other "loser" neurons are set to zero. The absolute activations of the positive and negative loser neurons are summed up and redistributed to their respective winners, [12] call this step "energy redistribution". Figure 4.1 shows how the competition during training for one feed forward step works, when $k = 2$. First, the activations of the input features $x_1 ... x_d$ are calculated via $h_i = tanh(Wx_i + b)$. The $\frac{k}{2} = \frac{2}{2} = 1$ largest positive and negative values are kept and all other "losers" are set to zero. Next, the energy of the positive and negative losers is calculated and assigned to their winner neurons. The amplification parameter α controls how strong the effect of winning is and according to [12] should be set to $\alpha > k/2$. It is important to note, that during error backpropagation winner and loser neuron parameters are updated, due to the α connection. This helps to alleviate the problem of dead neurons, which can be an issue for related techniques like k-sparse autoencoders [40].

During test time KATE works like a normal autoencoder, because the parameters are already trained to be distinctive. This means, that the input data is simply projected to the embedding space. KATE is related to techniques like k-sparse [40] and Winner-Take-All autoencoders [41]. The main difference between those two is that KATE does not simply set all loser neurons to zero, but redistributes their energy. This has two important effects. First, an increase of the amplification parameter α can be thought of

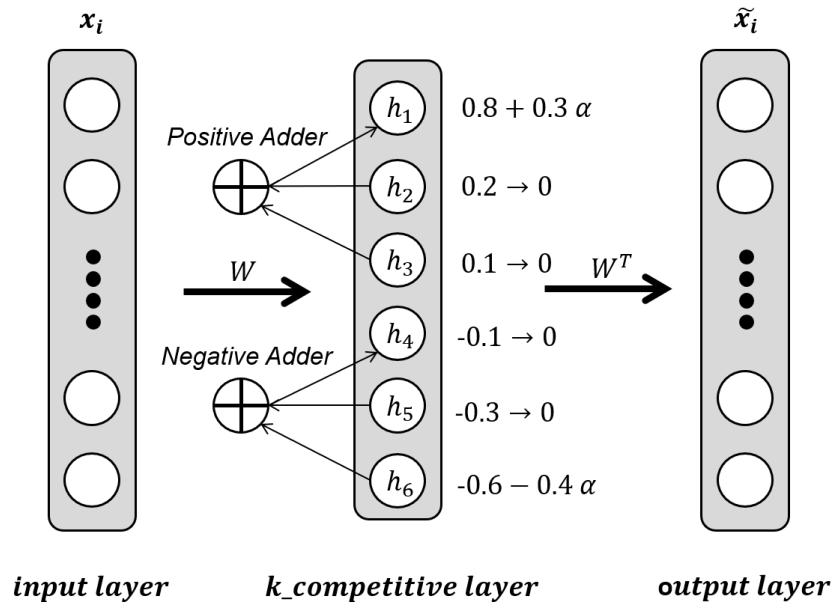


FIGURE 4.1: The figure shows one feed forward step of KATE during training for $k = 2$. Figure adapted from [12].

as an increase in the reward for winning in the feed forward phase. Second, during backpropagation the gradients are still flowing through the loser neurons.

On a side note, it is interesting to see that the winner-takes-all learning rule comes from the field of computational neuroscience and it is inspired by biological neural networks, see [35], [65].

4.3.8 Deep Embedded Clustering

Deep Embedded Clustering (DEC) developed by Xie, Girshick, and Farhadi [62] is one of the first techniques that simultaneously learns a feature representation and cluster assignments with a deep neural network. The general outline of this technique has inspired further research due to its impressive results, like [18]. The main contribution of [62] is the formulation of a loss function that quantifies a desirable clustering representation. For this they use the Student's t-distribution, with $\alpha = 1$ degrees of freedom, to measure the distance between embedded points z_i and centroids μ_j, see equation 4.3. The rationale behind this is, that equation 4.3 is heavier tailed than e.g. a gaussian or delta distribution, which allows for similar features to be mapped close together (local structure preservation) and can be evaluated fast, this idea has been adapted from t-sne, for a detailed explanation see section 2 and 3 of [36].

$$q_{i,j} = \frac{(1 + \|z_i - \mu_j\|^{\frac{2}{\alpha}})^{-\frac{\alpha+1}{2}}}{\Sigma'_j (1 + \|z_i - \mu'_j\|^{\frac{2}{\alpha}})^{-\frac{\alpha+1}{2}}} \tag{4.3}$$

From $q_{i,j}$ in equation 4.3 they measure the KL Divergence from a desired target distribution P, see equation 4.4.

$$L = KL(P||Q) = \Sigma_i \, \Sigma_j \, p_{i,j} \, \log \frac{p_{i,j}}{q_{i,j}} \tag{4.4}$$

The authors define the following desirable properties for a target distribution P:

- strengthen predictions
- put more emphasis on data points assigned with high confidence
- normalize loss contribution for each centroid to prevent large clusters from distorting the hidden feature space

$$p_{i,j} = \frac{\frac{q_{i,j}^2}{f_j}}{\Sigma_{j'} \frac{q_{i,j'}^2}{f_{j'}}} \tag{4.5}$$

Equation 4.5 fulfills the above properties. The term $q_{i,j}^2$ strengthens high confidence predictions which leads to improved cluster purity. The term $f_i = \Sigma_i q_{i,j}$ renormalizes by frequency per cluster while summing up the soft cluster frequencies.

During training DEC uses a two step approach. First, it learns a low dimensional representation, also called encoding, of the input data by using a stacked denoising autoencoder architecture [61]. The learning is done via greedy layer wise pretraining [7]. This procedure builds an autoencoder layer by layer, where the first is trained from the input features, the second takes the encoding of the first autoencoder as input and builds a new encoding. This is iteratively repeated until the desired network size is reached. Second, DEC takes the pretrained autoencoder, discards the decoder part and performs an initial k-means clustering on the received encoding. In the following steps the initial clustering is fine tuned to improve the results via backpropagating the gradients from the loss function 4.4. This jointly optimizes cluster centers μ_j and network parameters θ until less than a threshold value of T percent of points change their cluster assignment.

One problem of DEC is that during fine tuning the local structure of the embedding can be lost, which weakens its representational power and can decrease clustering performance. This is due to discarding the decoder part of the autoencoder. The algorithm IDEC [18] suggest a new approach which keeps the decoder part during fine tuning. The new loss function 4.6 combines the reconstruction loss L_r and clustering loss L_c, where $\gamma > 0$ controls the degree of space distortion [18].

$$L = L_r + \gamma L_c \tag{4.6}$$

A comparison of the two methods on the MNIST data set [31] can be seen in figure 3 of [18]. The two plots on the right end show that DEC mixes the red and blue points, while IDEC keeps them separated. This shows that the reconstruction loss L_r from equation 4.6 can preserve local structure, while DEC just pulls nearby points closer to their respective cluster centroids.

4.3.9 Deep Clustering Network

Yang, Fu, Sidiropoulos, *et al.* [63] improve clustering results by combining a deep autoencoder with k-means clustering, they call their method DCN, Deep Clustering Network. Their goal is to receive a "k-means friendly" space, which is defined as a low dimensional embedding were the data instances are evenly spread around their respective centroids. Similar to [18] they keep the decoder module during clustering to avoid distortion of the latent space. In contrast to [62] and [18], they do not use soft label assignments or the Student's t-distribution from [36] to compare initial cluster centroids to the points in embedded space. Instead, they use hard cluster assignments directly obtained from k-means performed on the embedded space. The latter decision forces them to use an alternating optimization scheme, were network and clustering parameters are updated separately. During training they optimize the cost function 4.7, which balances the reconstruction error with the "k-means friendliness".

$$\min_{\mathcal{W},\mathcal{Z}} L^i = l(g(f(x_i)), x_i) + \frac{\lambda}{2}||f(x_i) - Ms_i||_2^2 \tag{4.7}$$

Here $\mathcal{W}$, $\mathcal{Z}$ refer to the parameters of the encoder and decoder respectively, $g(f(x_i))$ refers to the reconstruction of the input x_i from the encoding $f(x_i)$, matrix M holds each cluster centroid vector and s_i is a one hot encoded vector for the cluster label. The left hand side of the addition is quantifying the reconstruction loss l, [63] are using mean squared error, and the right hand side enforces the notion that encoded points should be near their cluster centroids, by using the squared euclidean distance $\|\cdot\|_2^2$. The hyperparameter λ is used to balance those two terms.

The optimization step is split in two steps. First, they perform layer wise pretraining as described in [7]. Similar to [62] and [18] they use a stacked denoising autoencoder architecture [61], with tied weights and ReLu activation functions [46]. After pre-training they perform k-means on the encoding to get initial values for the centroids M and cluster assignments s_i. Second, they alternate between optimizing the cluster centroids and assignments together and the network parameters $\mathcal{W}$, $\mathcal{Z}$. While one pair is optimized the other stays fixed, for details of the optimization procedure, see section 4.2 of [63].

The authors evaluate their method on synthetic and real world datasets. The synthetic data experiments will be explained in detail and repeated in section 5.

5 Validation

5.1 Motivation

Deep Learning research is mostly driven by empiricism and theory is still behind the current fast paced publication cycle in the field. Due to this lack of theory many insights into deep learning architectures rely on experiments with synthetic and real world data. This is why this thesis focuses on empirical insights as well. The objectives of these experiments consist of two parts.

The first is to use synthetic experiments to gain understanding of various competition-driven autoencoders, like WTA, k-sparse and k-competitive, while the focus is on the latter. This is motivated by the results of KATE, where Chen and Zaki [12] confirmed experimentally that it can learn meaningful representations from text data, which remained challenging for existing architectures due to its high dimensionality and sparsity. It is through this characteristic that the idea was born to use the k-competitive layer in other challenging settings, which include noise, outliers or redundant information.

This leads right to the second goal, which is learning a meaningful representation from such data. The hypothesis is the following. Competition-driven autoencoders are robust to settings with noise, outliers or redundancy. This allows them to learn better representations than other autoencoder architectures. The performance of the learned representation is measured in their capability to improve clustering results. The improvement is not exclusive to clustering and other downstream task could benefit as well, but the focus here is in purely unsupervised techniques. To further restrict the scope, the only clustering algorithm which is considered is k-means. This choice is due to the widespread use of k-means and its simple design.

To summarize the above, the main motivation behind these experiments is to find a suitable autoencoder architecture which improves clustering results for a partitional clustering algorithm like k-means. The next section gives an overview of how the experiments are approached and evaluated.

5.2 Strategy

The strategy behind conducting the experiments consists of two approaches. First, use synthetic experiments to test for specific capabilities of the different architectures and gain insights. Second, use real world data sets which include outliers to compare the performance among learned representations for k-means clustering. The detailed approach is as follows.

First, find an optimal use for the k-competitive layer in an autoencoder architecture, which transfers its strengths to other data types than text. This is done by synthetic experiments to properly understand the inner workings and effects of competition.

This includes an evaluation of the k-competitive layer isolated from other nonlinear effects, by using it in a linear autoencoder.

After that, the k-competitive layer will be tested with different activation functions. In these experiments the goal is to find out if the tanh activation, which is used in [12], is still a reasonable choice for non-text data.

Next, the k-competitive layer's robustness to noise and redundant data is evaluated. For this different data sets are synthetically generated. The insights from the above experiments will be combined to make reasonable choices on the use of the k-competitive layer in an autoencoder, its training and the setting of its hyper parameters. This adapted algorithm will then be tested against several outlier benchmark data sets taken from [9] and different methods.

5.2.1 Compared Methods

The choice of comparison methods is determined by the following factors. First, the method should be based on autoencoders to allow for a side by side evaluation. Second, only unsupervised methods are considered. This means that no label information is used during training at all. Third, relation between methods should be considered. To be precise the ideas behind the k-competitive layer are connected to the k-sparse autoencoder [40] and the winner-takes-all autoencoder [41] (WTA).

These considerations led to the decision that the experiments always include a comparison between a vanilla autoencoder, a k-sparse autoencoder, a WTA autoencoder and an autoencoder with k-competitive layer. PCA is included as a baseline for a linear dimensionality reduction method and due to the fact that an autoencoder can be seen as a non-linear extension of PCA [5]. For the real world experiments the different autoencoder architectures will be used in different configurations. These include multiple layers, an overcomplete hidden layer and different activation functions. All methods are compared to the state of the art autoencoder clustering network IDEC [18], which was introduced in chapter 4.3.8. Additionally, the benchmark results of the best methods from the outlier detection survey of [9] are stated. Next, the evaluation between these algorithms is explained.

5.2.2 Evaluation

The evaluation of clustering performance is done by running k-means++[2] on the learned representations of the comparison methods. K-means++ is used due to its improved initialization scheme, which allows for a fair comparison. The found cluster labels are then compared to the ground truth.

The used performance metrics for the synthetic data sets are normalized mutual information score (NMI) and clustering accuracy (ACC) as is done in [18], [62], [63] and test error on a hold out test set to measure the reconstruction performance. The found representations are compared visually and judged on their ability to reconstruct the original low dimensional structure of the synthetic data sets. For higher dimensional encodings, PCA is used to plot it in two dimensions.

The real world data sets from [9] are heavily unbalanced, with outliers in the minority. That's why the evaluation metric is the Receiver Operating Characteristic - Area

Under the Curve (ROC-AUC). The ROC-AUC is recommended in [9], because it "inherently adjusts for the imbalance of class sizes typical of outlier detection tasks". To further understand what each evaluation criterion measures, they are explained briefly.

Normalized Mutual Information

The mutual information score measures the similarity between different clusterings for the same data [50]. It is based on mutual information, a general approach to determine the dependency between two random variables [45]. Contrary to the pearson correlation coefficient it is not limited to linear dependency.

The mutual information score is calculated in the following way, based on [50], see equation 5.1.

$$MI(U,V) = \Sigma_i^{|U|} \Sigma_j^{|V|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|} \tag{5.1}$$

Here U_i and V_j are two clusterings of the same data set, which has N instances. $|U_i|$ are the number of samples in cluster U_i and same for $|V_j|$, which is the number of samples in cluster V_j. If the ground truth is known, the found clustering can be compared to it. To make the comparison easier the score can be normalized between $[0, 1]$, which is then called Normalized Mutual Information score. An NMI of 0 indicates no mutual information and perfect correlation is met when it reaches 1, [50].

Cluster Accuracy

The unsupervised clustering accuracy (ACC) is widely used in deep clustering research, [18], [62], [63], which is why it is included here as well. Xie, Girshick, and Farhadi [62] suggests, that for comparison between methods, the number of ground truth labels should be set equal to the number of clusters, the score in equation 5.2 allows then for a comparison.

$$ACC = max_m \frac{\Sigma_{i=1}^{N} I_{(l_i = m(c_i))}}{N} \tag{5.2}$$

where l_i is the true label, c_i is the found cluster label and the function $m()$ ranges over all possible one to one mappings. This works in the following way. The metric takes a cluster label c_i and a ground truth label l_i and finds the best matching between them.

ROC-AUC

The ROC-AUC is computed from the area under the receiver operating characteristic curve. The ROC curve is computed by plotting the true positive rate against the false positive rate for different threshold values, [45]. By integrating the area under the curve the ROC-AUC value is calculated. The two rates refer to a confusion matrix, which is often used in classification settings. In the binary case of two classes 0 and 1 the rates can be explained in the following way. The true positive rate, often called sensitivity, is the ratio between the labels which have been correctly predicted as 1 and the total number of elements in class 1. The false positive rate, also known as false

alarm, is the ratio between the number of elements which have been predicted wrong as 1, but belong to class 0 and the total number of elements in class 0. Due to this normalization of the rates, the ROC-AUC adjusts automatically for class imbalances, which makes it a good choice for outlier detection evaluation.

5.2.3 Implementation

All experiments have been implemented in Python 3 [54].[1] To do this the k-competitive, k-sparse, WTA and vanilla autoencoder have all been implemented in Python by using the PyTorch framework[2] [48]. The implementation of the k-competitive autoencoder is based on the original Tensorflow[1] implementation by Chen and Zaki [12], but was reimplemented in PyTorch for more flexible experimentation. For IDEC the official Python implementation by [18] was used.[3] Additionally, the experiment settings, the preprocessing, data generation and evaluation have been implemented in Python. The NMI, ROC-AUC and k-means++ is used from the scikit-learn package[50] and the implementation of cluster accuracy is taken from [18]. To reproduce the experiments all settings have been managed via JSON files. This will be explained in the next section, where the process of generating the synthetic data is introduced.

5.3 Data Generation

The generation of synthetic data is performed via a Python script which allows for several configurations. These include the number, position, spread and dimensionality of Gaussian distributed clusters, the amount of noise points and noise dimension which should be added, a random seed for reproducibility and the non-linear transformations which should be performed. The non-linear transformations have been taken from the code provided by [63]. To make each data generating step reproducible a JSON file is used to pass all the necessary parameters to the generator. The settings for the synthetic data h_i before transformation consists of four clusters with 10 000 data points in two dimensions with zero noise, see figure 5.1.

After the data from figure 5.1 is generated, three new data sets are constructed from it. This is done by applying three different non-linear transformations to project the four 2-dimensional clusters into a 100-dimensional space, see equations 5.3, 5.4 and 5.5, which have been taken from [63].

$$x_i = \sigma(U\sigma(Wh_i)) \tag{5.3}$$

where $W \epsilon \mathbb{R}^{10x2}$, $U \epsilon \mathbb{R}^{100x10}$ and $\sigma()$ is the nonlinear sigmoid function.

$$y_i = \sigma(Wh_i)^2 \tag{5.4}$$

and

$$z_i = tanh(\sigma(Wh_i)) \tag{5.5}$$

[1] The code is available at the Gitlab server of the Data Mining Research Group at the University of Vienna.

[2] The PyTorch documentation can be found at https://pytorch.org/docs/stable/index.html

[3] www.github.com/XifengGuo/IDEC

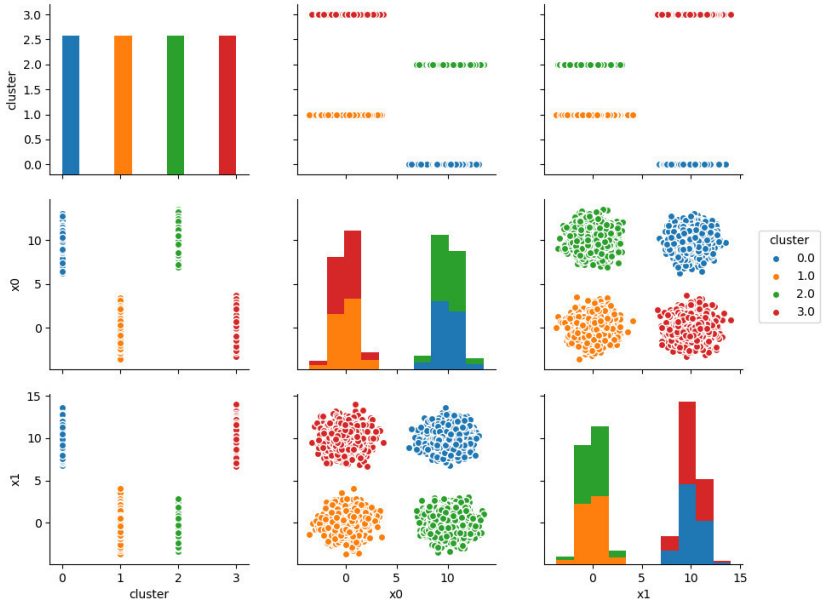


FIGURE 5.1: This figure shows the generated data h_i before transformation. There are four clusters in two dimension and each cluster consists of 2500 data points. This initial setting is taken from [63].

where $W \epsilon \mathbb{R}^{100x2}$, $\sigma()$ is the nonlinear sigmoid function and $tanh()$ is the nonlinear hyperbolic tangent function.

For each of these transformations the original data h_i is difficult to reconstruct and standard clustering algorithms or linear dimensionality reduction techniques are hardly effective to find the four real clusters, see figure 1 of [63]. During training the original data h_i is treated as unobservable and the algorithms learn only from the transformed data. The goal is then to reconstruct the original data h_i as best as possible.

5.3.1 Noise Data Generation

To evaluate the performance of the k-competitive layer in situations with noisy or non-relevant information two additional synthetic data sets are generated. The first data set uses additional noise points from which some can be considered as outliers from the original four clusters, see figure 5.2. The level of noise ranges from 0.01 to 1.0, this means for 10 000 data points there are additional 100 to 10 000 randomly generated noise points. Additionally, another version of this data set is constructed, which distributes the noise points even farther away from the four clusters. The generated data with noise $\tilde{h_i}$ is transformed by equations 5.3, 5.4 and 5.5 as well. These experiments evaluate how robust the autoencoder architectures are to outliers and how the learned representations differ from the same data without noise.

The second data set $\hat{h_i}$ uses additional noise dimensions to evaluate how well the autoencoders perform in settings with many redundant dimensions. The setting is similar as above, at first the generated four clusters with 10 000 data points in total are

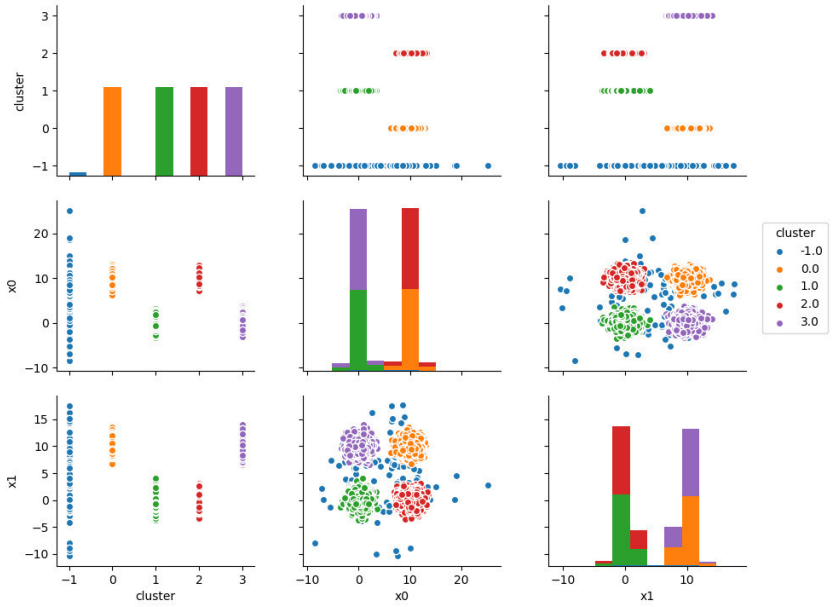


FIGURE 5.2: This figure shows the generated data h_i with additional noise points before transformation. There are four clusters in two dimensions and each cluster consists of 2500 data points. This initial setting is taken from [63]. Additionally, there are 100 randomly generated noise points added, denoted with label -1.

used and then additional uniform noise dimensions are added, see figure 5.3. The generated data is then transformed in the same way as the other data sets, with equations 5.3, 5.4 and 5.5.

5.4 Experiments

The motivation, strategy and data generation for the different experiments have been discussed above. In this section the details and results of each experiment will be presented. To summarize the above, the main goal of the experiments is to improve the understanding of using competition in an autoencoder as well as evaluating the clustering performance against similar architectures.

5.4.1 Linear k-Competitive Autoencoder

The following experiments illustrate how well a k-competitive layer [12] performs in a linear autoencoder with only one hidden layer[4]. This means that the only non-linearity in the autoencoder is the competition between neurons in the hidden layer. Chen and

[4]Due to an implementation detail in [12] the original k-competitive layer does not work as expected in settings with two hidden neurons and $k = 1$. In the Python implementation of [12] positive values are preferred for this particular edge case, thats why for the following experiments an alternative competition scheme has been used. First the winner in absolute values is determined and after that the loser is set to zero. Energy is only distributed if both values are negative or positive otherwise the winner does not gain anything from winning (the loser value is still set to zero).

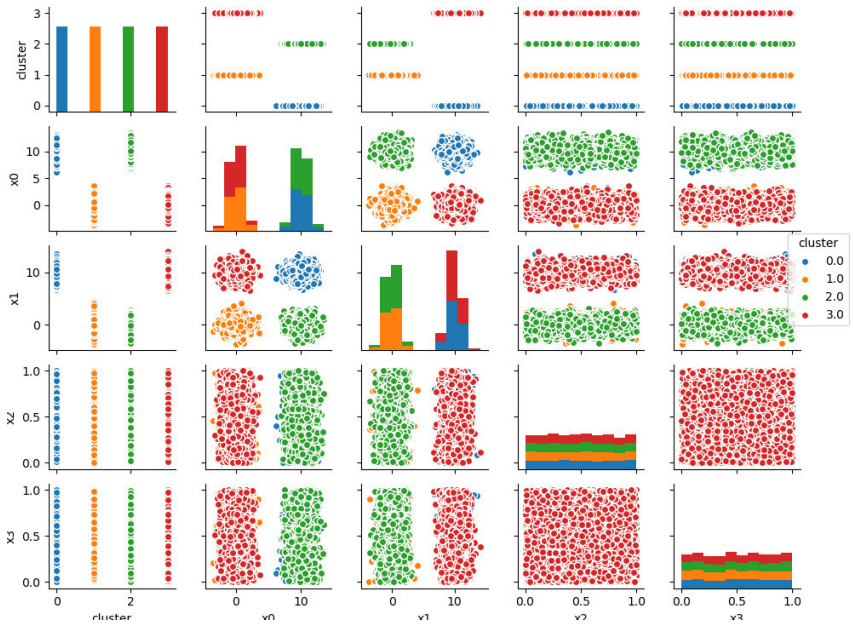


FIGURE 5.3: This figure shows the generated data h_i with two additional uniform noise dimensions, denoted as $\hat{h}_i$. There are four clusters in two dimension and each cluster consists of 2500 data points. This initial setting is taken from [63].

Model	Activation
k-competitive [12]	linear
k-sparse [40]	linear
WTA [41]	ReLu
autoencoder	linear

TABLE 5.1: The experimental setting of models for the linearity experiments in section 5.4.1.

Zaki [12] used the binary cross entropy as a loss function for KATE, when working with text data. In contrast, here the considered loss function is the mean squared error, which is a reasonable choice for real valued data. Table 5.1 summarizes the compared models, where each uses one hidden layer with two hidden neurons, is trained for ten epochs and has a batch[5] size of 64. The value of the amplification parameter α has been held constant at 1.05 over all experiments in this section, because its performance was fairly robust to the setting of α.

First the different models were run against the data from figure 5.1, before any non-linear transformations have been applied. Two example results can be seen in figure 5.4. The figure on the left shows that the vanilla autoencoder learned the identity mapping and the original data distribution has been preserved. The figure on the right is the learned representation by a linear autoencoder with a k-competitive layer, where $k = 1$. As can be seen each horizontal pair of clusters is pushed together, namely clusters 0-2 and 1-3. There are two interesting observations in this simple example. First, the k-competitive layer has a strong regularization effect, in this case it hinders

[5]Batch will be used synonymous to minibatch, which is common in the literature [16].

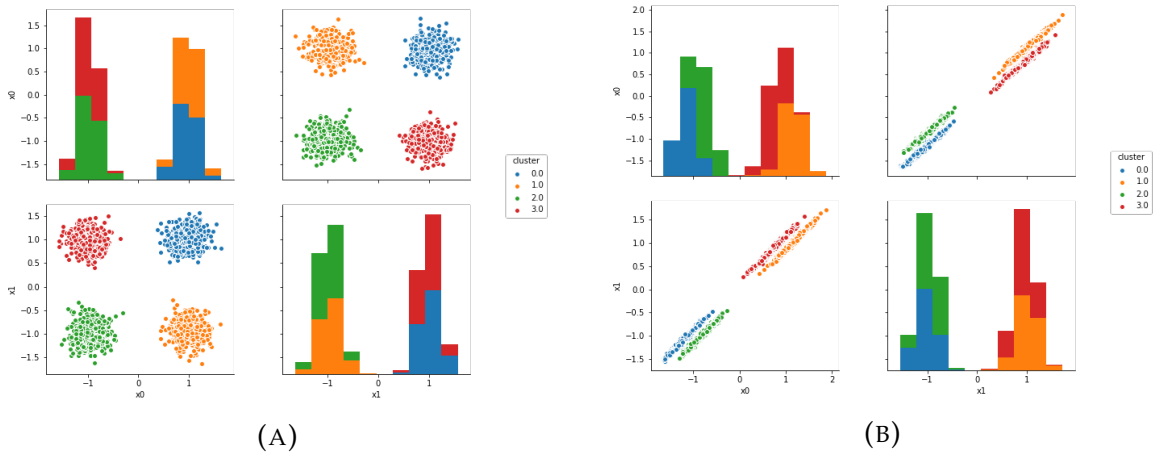
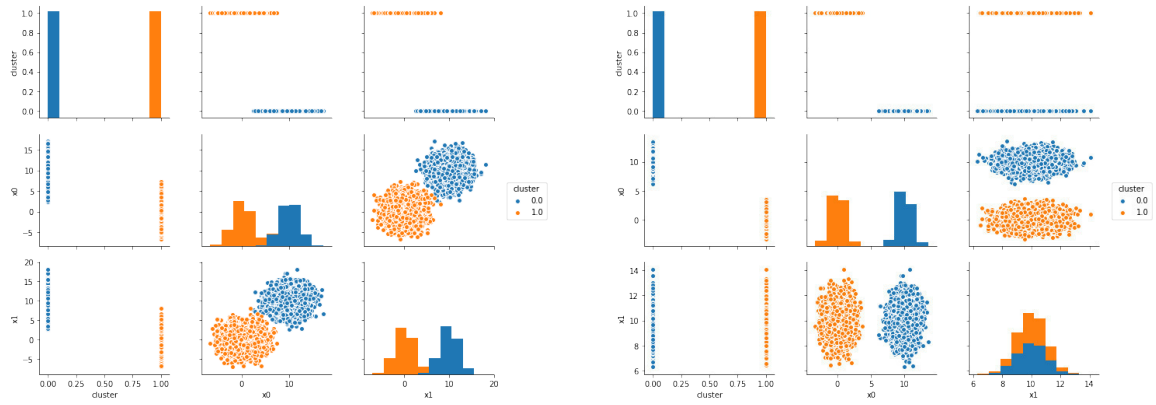


FIGURE 5.4: The above figure shows the learned 2-dimensional representations from the generated data in figure 5.1 before any non-linear transformations. (A) shows the 2-dimensional encoding found by a linear autoencoder. (B) shows the 2-dimensional encoding found by a linear autoencoder with k-competitive layer.

the autoencoder to learn the identity mapping. Second, the competition layer pushes 2 clusters together and tries to map them to a linear function. One hypothesis for this behavior is that competition between neurons makes them specialists for distinct patterns. As there are only two hidden neurons the four clusters are split evenly between them. This might be due to an effect that the amount of discernible patterns is limited by the number of neurons.

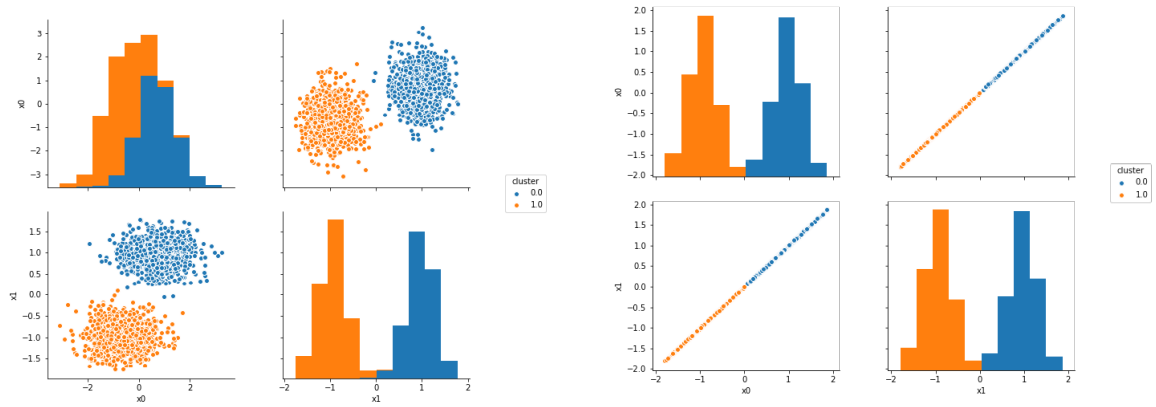
To investigate this behavior additional experiments are conducted, in which only two clusters in different configurations are used, see figure 5.5. The overlapping clusters from figure 5.5a are separable in one dimension and overlap in the other. This data set is used to validate against the hypothesis that neurons in the k-competitive layer learn more distinct patterns. The learned representations from the linear autoencoder and the linear k-competitive autoencoder can be seen in figure 5.6. While the original data in figure 5.5a overlaps only slightly in both dimensions, the linear autoencoder in figure 5.6a learned an identity mapping which mostly preserves the distributions in both dimensions. Figure 5.6b shows that the regularization effect of the competition prevents the linear k-competitive autoencoder from learning the identity mapping, thus the original space is not perfectly recovered. Interestingly, the two horizontal clusters are not pushed together like in figure 5.4b, but have been kept separately. In fact the histograms show that the distinction between the two clusters has increased slightly compared to figure 5.5a. These insights provide further evidence for the above hypotheses.

The data set from figure 5.5b contains two clusters which are easily separable in one dimension, but overlap in the other. This experiment should provide further insight in the capability of the neurons to separate the clusters and learn distinct features. In figure 5.7 the learned representations of the two linear autoencoder variants can be seen. The linear autoencoder learned a transformation which slightly rotated the two clusters, this results in an overlap in both dimensions, see histograms in figure 5.7a. The separation of the two clusters in both dimensions by the k-competitive layer can



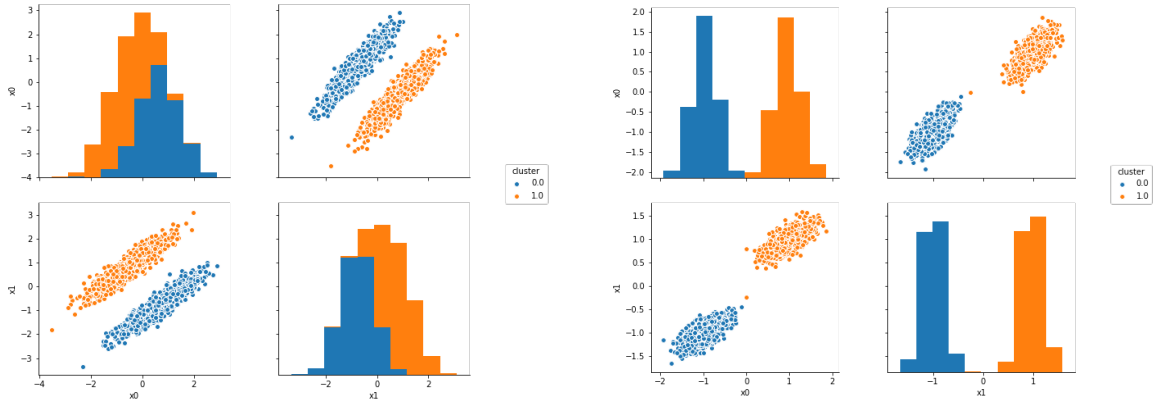
(A) (B)

FIGURE 5.5: The above figure shows two synthetically generated clusters in two different configurations. (A) shows two clusters with higher standard deviation, which slightly overlap in both dimensions. (B) shows two clusters which are separable in dimension x_0, but overlap in x_1.



(A) (B)

FIGURE 5.6: The above figure contains the learned representations from the 2-dimensional data set with two overlapping clusters in figure 5.5a. (A) shows the representation learned by a linear autoencoder. (B) shows the representation learned by a linear autoencoder with k-competitive layer.



(A) (B)

FIGURE 5.7: The above figure contains the learned representations from the 2-dimensional data set with two stacked clusters in figure 5.5a. which are separable in one dimension, but overlap in the other. (A) shows the representation learned by a linear autoencoder. (B) shows the representation learned by a linear autoencoder with k-competitive layer.

be seen in figure 5.7b. The histograms show a clear separation in both dimensions, compared to the original data in 5.5b. When comparing the learned representation in the full 2-dimensional space it is clear that the learned representation is different to the original data. While this further strengthens the aforementioned hypothesis, another trade-off is highlighted, namely the one between the separation of distinct patterns and the preservation of the original space.

To further explore this the learned representation of the linear k-competitive autoencoder has been observed for each training epoch, which is not plotted here due to space constraints. It can be seen that during training the distinctiveness in each neuron increases, but in the same time the reconstruction error on the validation set increases, which can be seen in figure 5.8.

Additionally, to the visual inspection of the learned representations, a clustering via k-means++ has been performed. The results in figure 5.9 show the performance of k-means on the learned representation of all models listed in table 5.1 over the synthetic data sets. The x-axis indicates which data set was used, "original" denominates the 2-dimensional data set h_i from 5.1, "transformed sig" denotes the data set which was constructed by transforming h_i via the sigmoid function from equation 5.3, the same is true for "transformed exp" (5.4) and "transformed tanh" (5.5). The last figure shows the test loss of the models measured as the mean squared reconstruction error on a hold out test set, PCA (violet line) and kmeans on the unmodified data (brown line) have no reconstruction error due to the nature of their algorithm. The overall performance is quite bad and k-means on the original data or in combination with PCA works better than on the representations learned by the neural networks. This highlights the limited capabilities of networks without non-linear activation functions. Only WTA [41], which uses the non-linear relu activation function performs quite well over the data set.

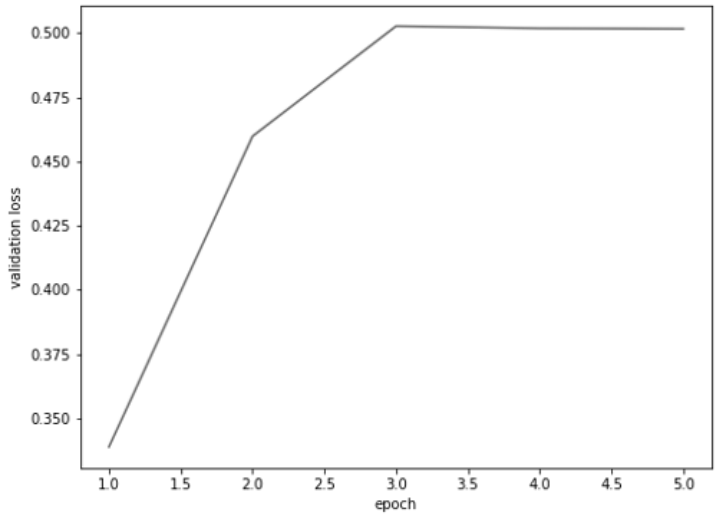


FIGURE 5.8: This figure shows the steep increase in validation loss for each epoch, while training the linear k-competitive autoencoder on data set 5.5b.

Conclusion

The conducted experiments shed some light on the behavior of the k-competitive layer when isolated from the influence of any non-linear activation function. A few interesting observations have been made. First, the k-competitive layer increases the distinctiveness of each neuron during training. This can result in nicely separated clusters like in figure 5.7b or in merged clusters, see figure 5.4b. The latter could be caused by the restriction of using only two neurons in the hidden layer or by the lack of a non-linear activation function. Second, there is a clear trade-off between validation error and separability. The reason for this is that the original space of the data sets used in the experiments has clusters which are close together or do overlap in some dimensions. The k-competitive layer tries to increase the distance between these clusters, which distorts the learned representation and increases the reconstruction error during validation. That is why for real data sets it is important to try different values for k and number of hidden neurons. Additional validation on the non-linearly transformed synthetic data sets have shown that the performance of linear autoencoders is below that of PCA, highlighting the need for non-linear activation functions, which will be discussed next.

5.4.2 Nonlinear k-Competitive Autoencoder

In this section the same experiments as in chapter 5.4.1 will be conducted, but the main goal here is to study the behavior of the k-competitive layer with different non-linear activation functions and evaluate its performance for clustering tasks. The following activation functions will be examined. The hyperbolic tangent (tanh) function, the rectified linear unit (relu) function and its variant leaky relu, see figure 5.10 for a 2-dimensional plot of the activation functions. The tanh function is used in the original paper by [12] and serves therefor as benchmark against the other two. The reason for choosing the relu function is twofold. First, it has been successfully used in several

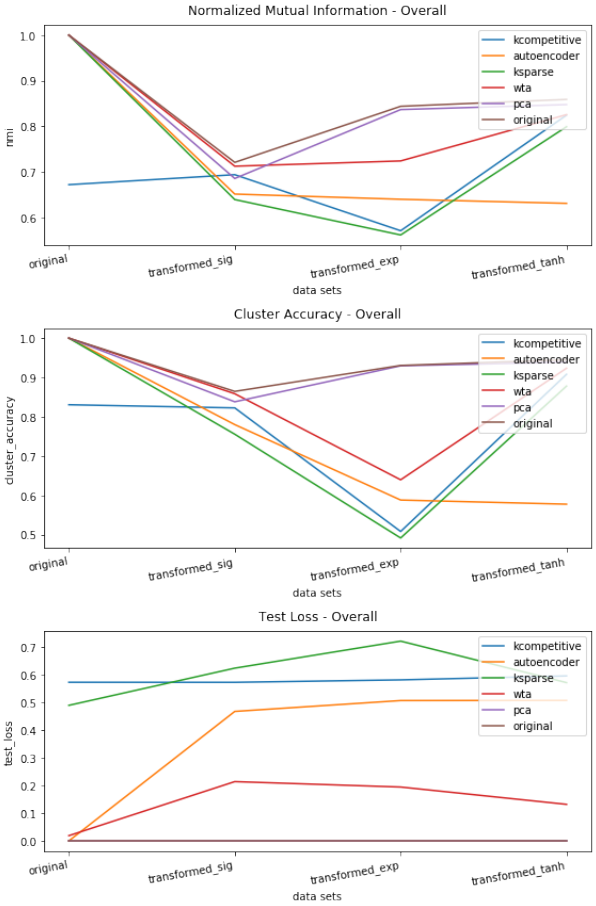


FIGURE 5.9: The above figure shows the clustering performance for k-means on the learned representation of all linear models listed in 5.1 measured in normalized mutual information and cluster accuracy.

neural network architectures [16], [20]. Second, the relu function makes the competition algorithm simpler and faster by only allowing positive values. The leaky relu function tries to avoid a weakness of the "normal" relu function, which is called dead neurons. The latter means, that all activations are set to zero, which stops the gradient from propagating. Thats why it has a negative slope, which can be handled like a hyperparameter, but is often set to 0.1. Deciding which activation function to use in hidden units is an active area of research and there are not "many definitive guiding theoretical principles" [16]. That said, it is usually a process of trial and error and it's not possible to predict the outcome beforehand [16]. Therefor, the different activation functions will be evaluated on several synthetic data sets. Look at table 5.2 for the experiment setting. Note, that each activation function is used together with the weight initialization scheme proposed in the literature.

In the first experiment the non-linear k-competitive autoencoders are trained on the data set with four clusters and its non-linear transformations, see figure 5.11. The overall performance measured in normalized mutual information is quite poor, but still an improvement over the performance in the linear setting, for comparison refer to figure 5.9. Two reasons why this could be happening will be discussed.

First, according to the experiments conducted in section 5.4.1, competition has a high regularization effect in the linear setting. This effect might very well transfer to the

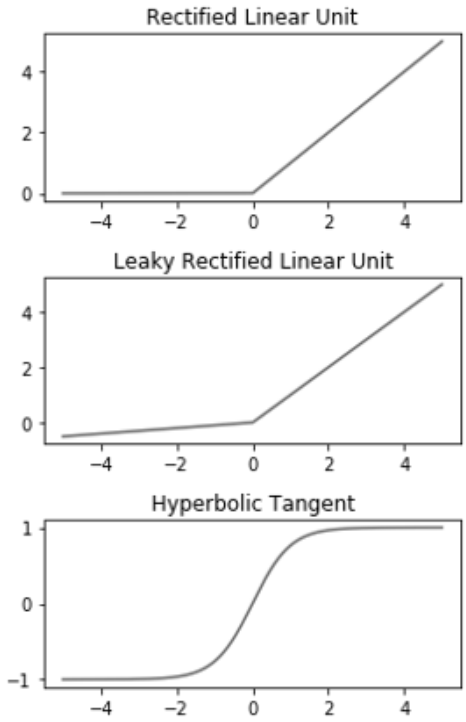


FIGURE 5.10: The above figure shows three common non-linear activation functions. The top figure shows the relu activation function, which outputs only numbers greater or equal to zero. The middle one is the leaky relu function with a negative slope of 0.1. The bottom figure is the tanh function, which is defined on the interval [-1,1].

non-linear setting and could hinder the neurons from learning anything useful in the 2-dimensional case. Second, in addition to the regularization effect, the amplification of energy to winner neurons regulated through α could push the non-linear activations to extreme values, by making winner neuron values very large[6]. The next experiments will shed some light on these suspicions.

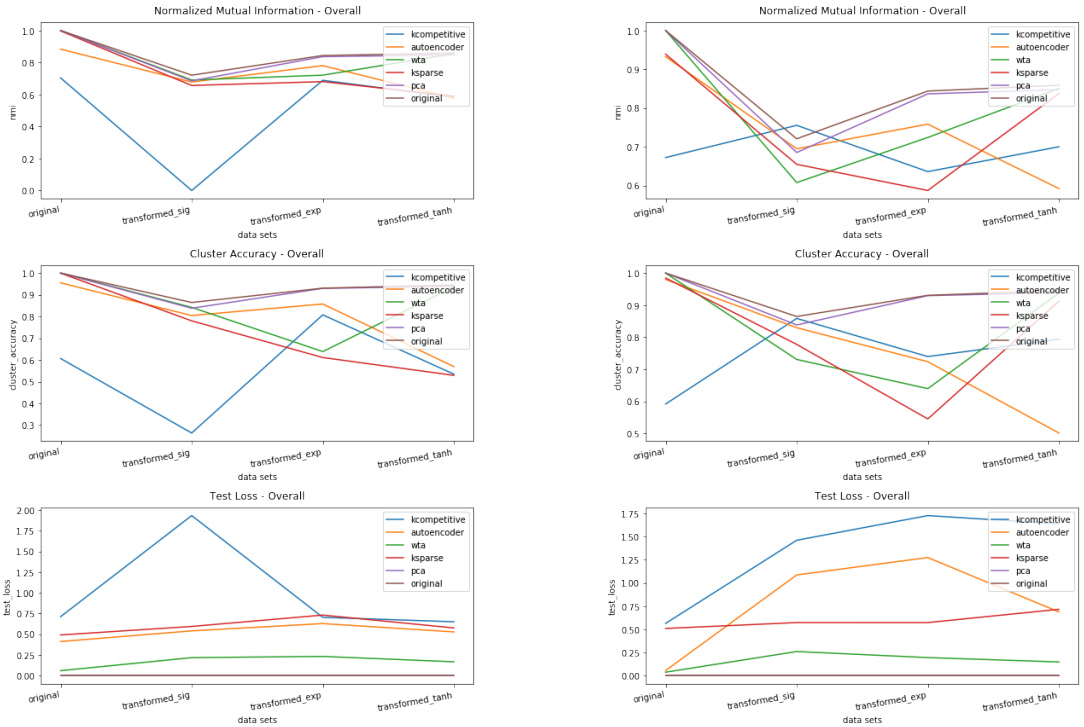
The approach is to break down the questions in smaller parts. First, it will be examined whether α has any effect on clustering performance at all. Due to this several configurations of the α parameter are evaluated according to their NMI for all three activation functions on the transformed sigmoid data set. The plot in figure 5.12 indicates that the tanh and leaky relu activation function are quite sensitive to the setting of the alpha value. Note, that the recommended value for the tanh function, from [12] of $\alpha = \frac{k}{2} = \frac{1}{2}$, does not hold for this data set. The relu function which only uses positive winners is very robust to the choice of α, which provides some evidence of its small competitive effect in the 2-dimensional case. Overall, it can be said that for the tanh and leaky relu function which both allow positive and negative winners the value of α matters. Additionally, competition has always an effect, regardless of the alpha setting. Whereas, for the relu function the effect of α is minor.

Next, the specific effect of α on the tanh and leaky relu function is explored. Figure

[6]Extreme values here means that e.g. for the tanh function all values are either 1 or -1 and for the relu function all values would be zero. This is sometimes referred to as the saturated regime of an activation function [16].

Model	Activation	Initialization
k-competitive [12]	tanh	xavier [15]
k-competitive	(leaky) relu	he [20]
autoencoder	tanh	xavier
autoencoder	(leaky) relu	he
k-sparse [40]	linear	xavier
WTA [41]	relu	he

TABLE 5.2: The experimental setting of models for the nonlinearity experiments in section 5.4.2. Each model has one layer, two hidden neurons, is trained for ten epochs and has a batch size of 64.



(A) The results for the k-competitive layer with relu nonlinearity

(B) The results for the k-competitive layer with tanh nonlinearity

FIGURE 5.11: The above figure shows the clustering performance for k-means on the learned representation of all models listed in 5.2 measured in normalized mutual information and cluster accuracy.

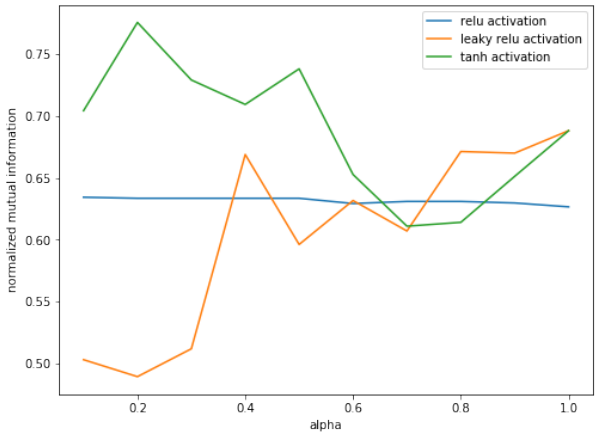


FIGURE 5.12: This figure shows a comparison between the tanh, relu and leaky relu activation function for different values of α, evaluated on the "transformed sig" data set from equation 5.3. The other settings where kept the same as in table 5.2.

5.13, depicts the ratio of positive to negative activations of both neurons. This means that a value larger than one, indicates a higher number of positive than negative values. The first observation is that for most values of alpha e.g. $\alpha = 0.4$ the leaky relu activation causes one neuron to focus on positive and the other on negative values. A slightly weaker, but similar effect can be observed for the tanh function. There are some values of α which seem to enforce a specific value distribution in both values, e.g. $\alpha = 0.2$ or $\alpha = 0.4$. Overall, the pattern for the tanh function is not as convincing as for the leaky relu one.

Additionally, when looking at the encoding found by the autoencoder with tanh activation and $\alpha = 1.0$ in figure 5.14, it can be seen that most values are on the saturated regimes. This highlights the aforementioned problem, that the competition can push the activations to extreme values. The competition makes each dimension as distinct as possible, which in fact can be seen in the histograms of both neurons. Interestingly this effect does not change much in the 2-dimensional case, when observing the encodings for different α values.

According to the results of the experiments above, it can be concluded that, at least for the 2-dimensional case competition has a regularizing effect which is too strong and overtakes the ability to reconstruct the data properly. To further explore this, settings with higher dimensional encodings are investigated. Here, the number of four hidden units has been chosen to make competition balanced between two positive and two negative winners. Additionally, the number of winner neurons is set to $k = 2$. Due to the experiment before, it is clear that the tuning of the α parameter is quite important. Thats why hyperparameter tuning of α for each data set was conducted, this results in the values seen in table 5.3. It shows that the NMI score has been considerably improved compared to the performance in figure 5.11. For the transformed sigmoid data set the NMI for tanh increased for about 0.03 points. The NMI for relu increased by 0.74 points, but this has to be taken with a grain of salt, as the previous NMI of zero was due to the dead neuron problem. For the other two data sets the increase is considerably higher. About 0.25 for the transformed exp and 0.2-0.3 points for

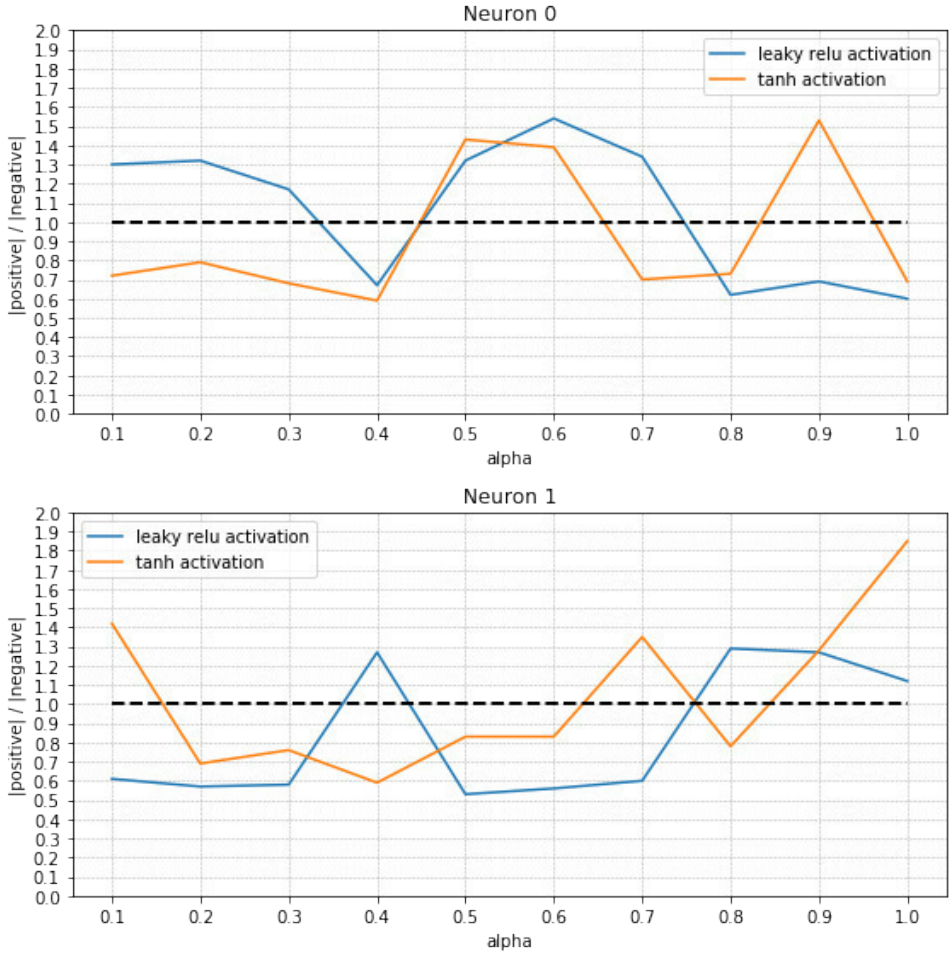


FIGURE 5.13: This figure shows a comparison between the tanh and leaky relu activation function in both neurons for different values of α on the transformed sig" data set from equation 5.3.

the transformed tanh data set. This improvement is impressive, but as mentioned before the correlation between NMI performance and reconstruction of the original data space is not perfect. Additionally, the found encodings are now in the 4-dimensional space, while the true underlying data has only 2 dimensions (figure 5.1), which makes a direct visual comparison difficult. Still, what can be evaluated is how well each cluster is separated in each dimension. In figure 5.15 the encodings of the best performing models on the transformed tanh and exp data set can be seen. One obvious observation is that the reconstructed space does not resemble the original data in any of the 2-dimensional sub-dimensions. The encoding with the relu function (figure 5.15a) shows at least some separation between clusters, especially on the x_2 vs x_3 plot and x_0 vs x_2. The learned 4-dimensional representation from the exponentially transformed data set (figure 5.15b) is despite its high NMI not well separated. Interestingly, it has the same problem as in the 2-dimensional space. Namely, most points lie on the saturated regimes of the tanh function close to -1 or 1, see figure 5.14 for a comparison. To better visualize the found encodings in 2-dimensions, a principal component analysis has been performed. The results can be seen in figure 5.16. These plots are quite similar to the original data space and indicate that there might be some redundancy in the

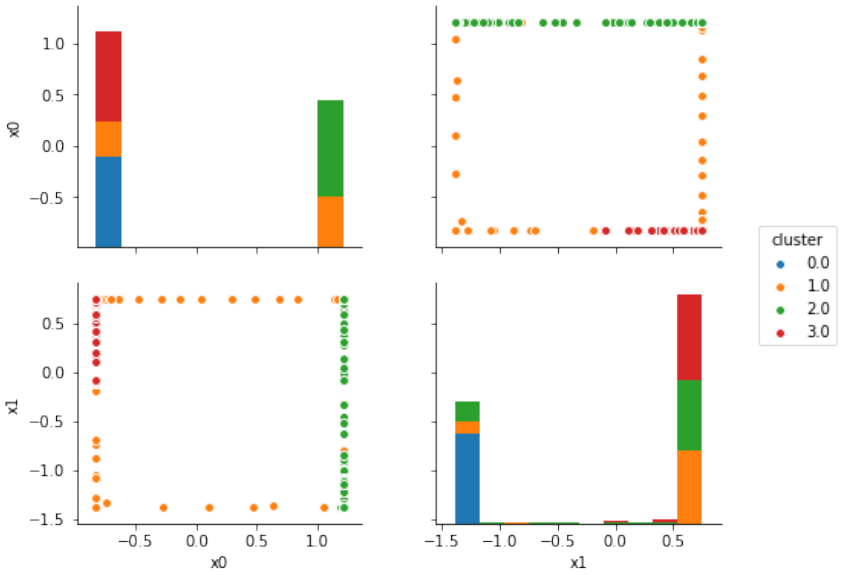


FIGURE 5.14: The figure above shows the encoding for the tanh activation function and $\alpha = 1.0$ corresponding to figure 5.13.

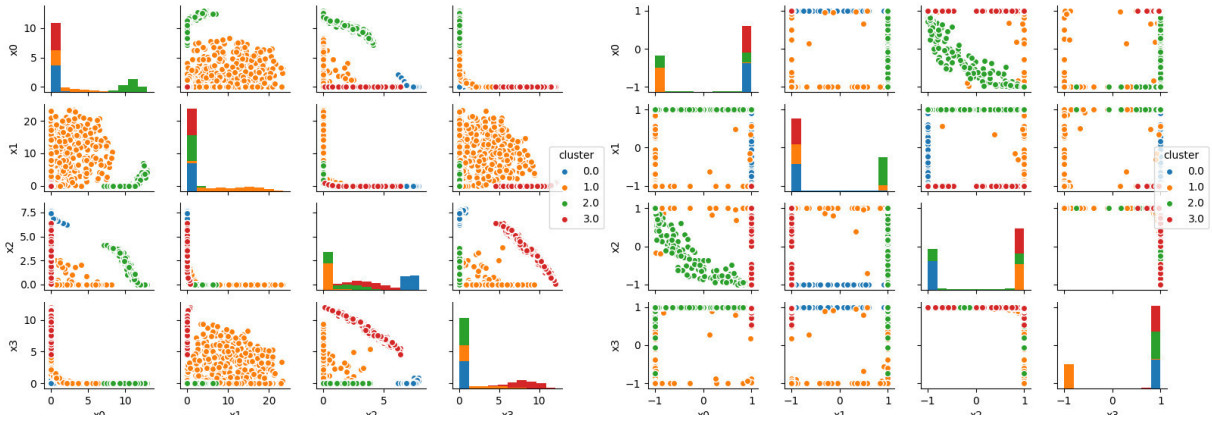
4-dimensional encodings, which can be reduced via PCA. While these results are quite promising, it has to be noted that the more complicated sigmoid transformed data set (equation 5.3) has despite its high NMI a quite bad reconstruction, as can be seen in figure 5.17.

Conclusion

The conducted experiments brought several insights for choosing the value of k, α, the number of hidden neurons and a suitable activation function. First, when using the k-competitive layer in an autoencoder it is important to see α as an additional hyper-parameter which has to be tuned extensively. This is in contrast to the findings of [12], who suggest to set it to $\alpha = \frac{k}{2}$ when using the tanh activation function and text data. At least, the conducted experiments show that this suggestion does not hold for the used data sets. The strong regularizing effect of competition which was observed in the linear experiments in section 5.4.1, does transfer to the non-linear activation functions as well. In fact, the non-linearities introduce further complexity when training the neural network. This results in the problem of reaching saturated regimes, which hinders the learning process. The relu function was less sensitive to the competition and the setting of α, when using only two neurons, but in this case its NMI was quite bad overall. The overall conclusion of using competition with only two neurons is that the regularization effect is too strong and obstructs a proper reconstruction of the original data space. The effect of the competition is more balanced when using at least four neurons, this setting balances the amount of positive and negative winners when $k = 2$. This resulted in a major improvement over the two dimensional case for all non-linear activation functions used, see table 5.3. To allow a comparison with the original data set 5.1, PCA was used to project the 4-dimensional encodings to 2-dimensions. This resulted in a nicely separated representation for the tanh and exponentially transformed data, see figure 5.16. For the more challenging sigmoid transformed data set

data set	activation	α	NMI	NMI(previous)
transformed sigmoid (5.3)	tanh	1.6	**0.7881**	0.7553
	leaky relu	1.0	0.7639	0.6615
	relu	0.2	0.7427	0.0
transformed exponential (5.4)	tanh	0.5	**0.9214**	0.6360
	leaky relu	1.0	0.8844	0.7188
	relu	0.2	0.8524	0.6889
transformed tanh (5.5)	relu	0.1	**0.9431**	0.5829
	tanh	0.4	0.9053	0.7001
	leaky relu	1.0	0.8914	0.7714

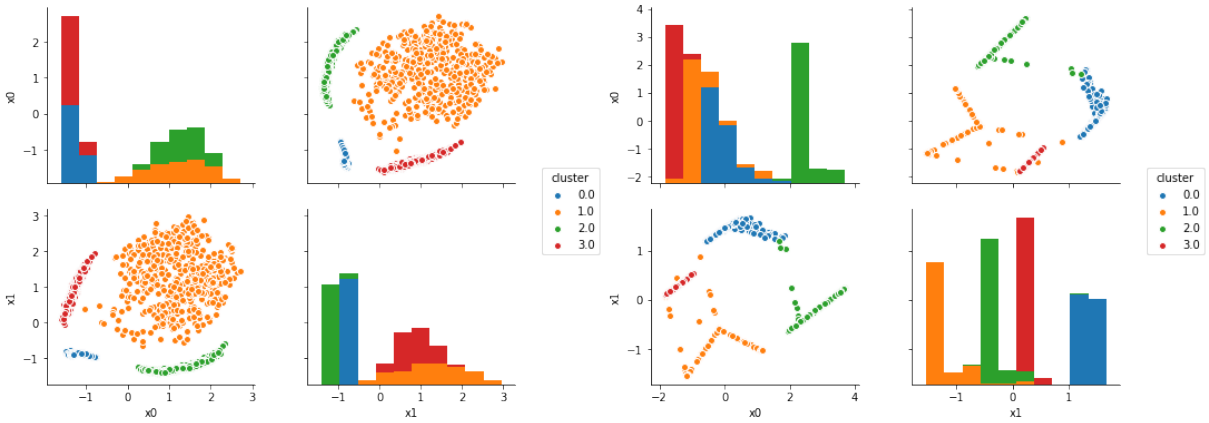
TABLE 5.3: The results for the different data sets and activation functions for the k-competitive layer with four hidden neurons and $k = 2$. The parameter α has been tuned for each data set and activation function separately.



(A) Encoding of tanh data set with relu non-linearity. ($NMI = 0.9431$)

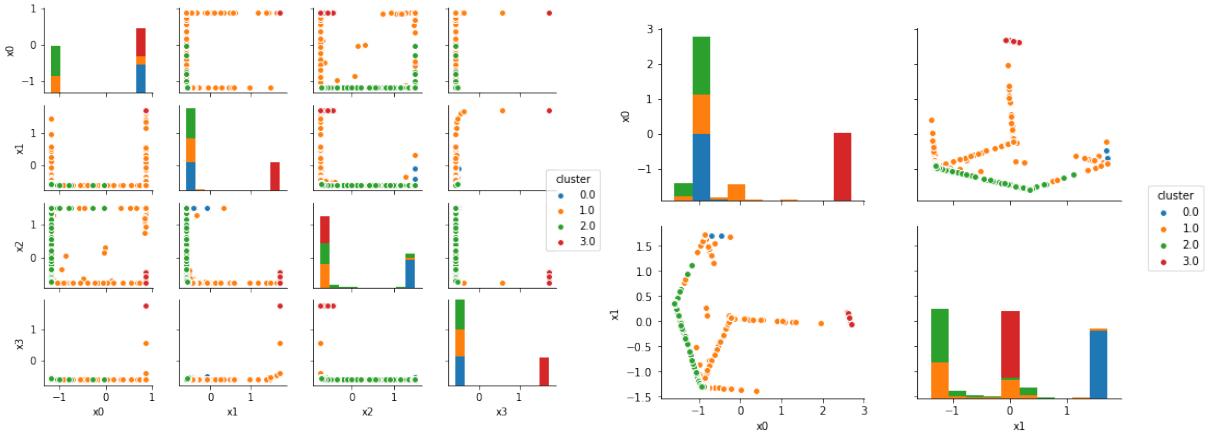
(B) Encoding of exp data set with tanh non-linearity. ($NMI = 0.9214$)

FIGURE 5.15: The above figures show the best NMI performer from table 5.3 on the tanh (equation 5.5) and exponentially (equation 5.4) transformed data sets.



(A) PCA on figure 5.15a (B) PCA on figure 5.15b

FIGURE 5.16: The above figures show the two principal components for each encoding found in figure 5.15.



(A) Encoding of sigmoid data set with tanh non-linearity. ($NMI = 0.7881$)

(B) PCA on figure 5.17a

FIGURE 5.17: The above figures show the best NMI performer from table 5.3 on the sigmoid (equation 5.3) transformed data set.

the reconstruction is not that similar to the original data space, despite its high NMI, see figure 5.17.

Overall it can be said, when only considering the NMI performance the tanh activation function worked best. An inspection of the learned low dimensional representation shows that the original data space is often not well reconstructed. This downside of the tanh function is tightly connected to the issue of saturated regimes. Most activation values for the k-competitive layer are either -1 or 1, which forces the representations to be degenerated, e.g. one cluster is mapped to one very dense point. Whether this is desired, depends on the application, e.g. multiview clustering [67] might want to avoid a situation in which clusters are merged together in multiple subspaces. The relu function has overall a slightly lower NMI than the other two , but the low dimensional representations from the transformed data sets, are looking quite similar to the original data space. The leaky relu function is performing quite well over all three transformed data sets. It seems, that it can combine the best of both worlds. It has no saturation points and allows for positive and negative competition. The problem is that competition is unfair. A positive winner receives always a higher activation value than the negative winner, simply due to the smaller negative slope.

Additionally, after choosing an activation function the following should be considered. When using an activation function which allows for both positive and negative winner, the number of neurons and the value of k should be an even number, to make competition balanced. For the relu the choice of α and k does not matter that much in settings with less than 3 hidden neurons. If the dimensionality is larger, k should be set to the desired level of competition.

5.4.3 Noise Experiments

One of the main reasons to explore the k-competitive layer was due to its superior performance in settings with very sparse data, see [12], which was a challenge for previous autoencoder architectures. The main idea is that this characteristic transfers to similar settings with sparse information, e.g. noisy data or data with redundant dimensions. To test this behavior the k-competitive layer is evaluated against two data sets, which have been discussed in detail in chapter 5.3.1. The first has a different number of noise dimensions (figure 5.3) and the second has additional noise points (figure 5.2). Both data sets are non-linearly transformed by equation 5.3, here referred to as sig transformation. The results of the previous experiments indicated that the tanh activation function worked quite well over a variety of settings and will therefore be used here. Additionally, the α parameter has been tuned for each noise setting and the number of hidden units has been set to eight, to compensate for a more difficult setting. The setting of all compared models can be seen in table 5.4.

The experiments are conducted to check whether the k-competitive autoencoder is robust to redundant dimensions[7]. The rationale behind this is the previously seen effect of the k-competitive layer to squeeze together one cluster from multiple dimensions onto one point, this effect might be enforced by very similar dimensions. The results can be seen in figure 5.18. Interestingly, the non-linearly transformed data does not seem to be a great challenge for all compared techniques, but the k-competitive

[7]Redundant dimensions are here defined as dimensions without information, namely uniformly distributed noise dimensions.

Model	Activation	Initialization
k-competitive [12]	tanh	xavier [15]
autoencoder	tanh	xavier
k-sparse [40]	linear	xavier
WTA [41]	relu	he [20]

TABLE 5.4: The experimental setting of models for the noise experiments in section 5.4.3. Each model has one hidden layer, eight neurons, is trained for three epochs and has a batch size of 64.

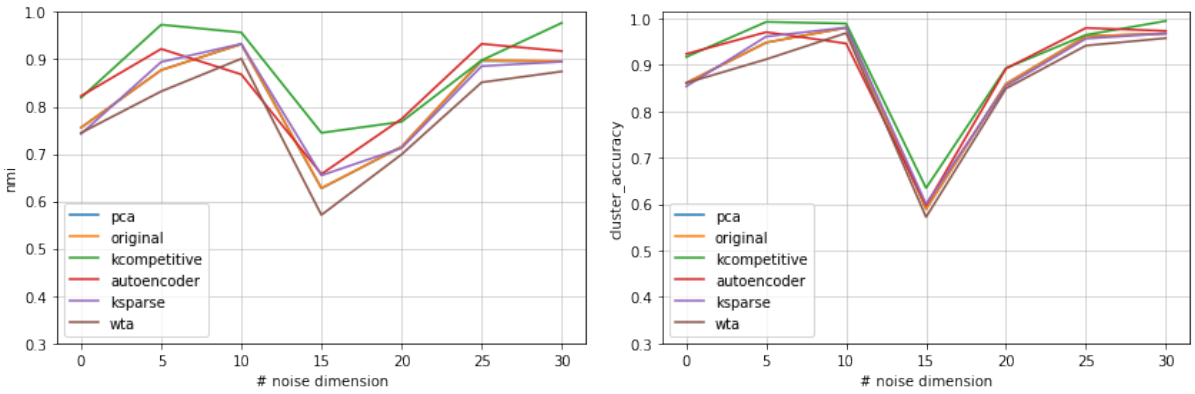
autoencoder is performing quite well in regards of NMI score. For the cluster accuracy the results are less distinct. This might be more an effect of the experiment setting, than of the algorithms tested. The more noise dimensions are added, the less strong is the effect of the non-linear sig transformation and therefor the less difficult is the data set. This might explain, why the performance goes up again, after 20 noise dimensions. The conclusion for this experiment is therefor, that the results have to be taken with a grain of salt and in total not much can be said, except that all algorithms performed quite well.

In figure 5.19 the results for the experiment with different amount of added noise points can be seen. The x-axis indicates the number of noise points added to the original data from 5.1, e.g. there are 10 000 points in the original data, then for a noise level of 0.1 there are 1 000 noise points added. Here the performance makes more sense intuitively, as the NMI and cluster accuracy are deteriorating when the amount of noise is increased. The k-competitive layer performance in regard of NMI is superior across different noise levels. Again, the picture looks different for cluster accuracy, here the performance is less distinct. In fact at an noise level of 1.0 the cluster accuracy of the k-competitive autoencoder is even lower than for the normal autoencoder. This last result has to be considered carefully, due to the fact that a noise level of 1.0 means that there are 10 000 noise points added. These noise points are quite close to the real clusters, which causes the clustering to include them in one of the four clusters and in a real world setting this might make sense. In order to isolate this effect another data set is generated, in which all the noise points are farther away from the original four clusters. The results of this experiment are depicted in figure 5.20. [8] Again, a similar behavior as before can be observed, but the performance at a noise level of 0.1 is about 0.2 points lower for both NMI and cluster accuracy than before. These results, indicate that even for noise points which are farther away the k-competitive layer performs quite robust.

Conclusion

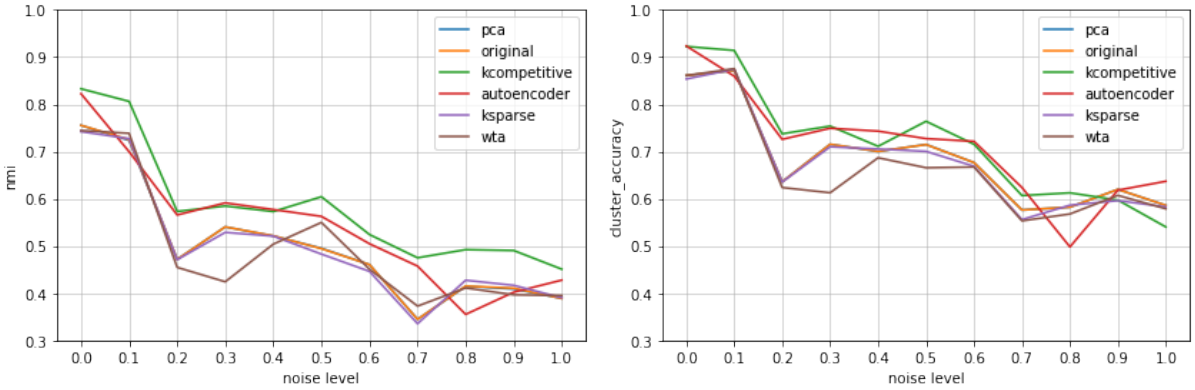
The above experiments show that the k-competitive layer can increase the the performance of k-means in the presence of noise points and outliers. The results for uniform noise dimensions are less distinct, but still in favor for the k-competitive layer. Additionally, as before the synthetic data sets have been non-linearly transformed via equation 5.3 to a 100 dimensional space. In these difficult settings the performance of simple k-means and pca dropped. The non-linear dimensionality reduction via

[8] Note, that the range of noise is now $[0.01, 0.1]$, which was necessary as this is a more difficult setting.



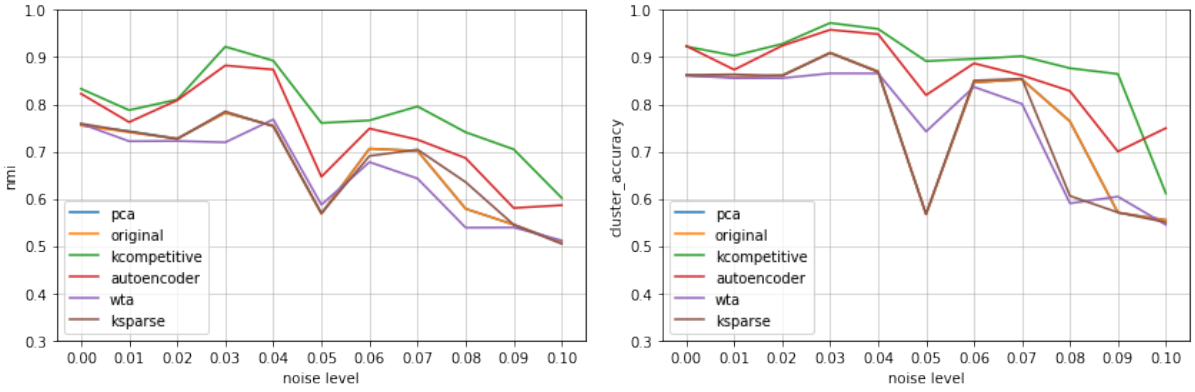
(A) (B)

FIGURE 5.18: The above figures depict the performance of the k-competitive layer against other techniques on the synthetic data set from figure 5.3. The number of noise dimensions has been increased by multiples of five. Figure (A) shows normalized mutual information and figure (B) the cluster accuracy.



(A) (B)

FIGURE 5.19: The above figures depict the performance of the k-competitive layer against other techniques on the synthetic data set from figure 5.2. The x-axis indicates the number of noise points added to the original data from 5.1. Figure (A) shows normalized mutual information and figure (B) the cluster accuracy.



(A) (B)

FIGURE 5.20: The above figures depict the performance of the k-competitive layer against other techniques on an adapted version of the noise experiments, where the noise is distributed farther away from the clusters. Figure (A) shows normalized mutual information and figure (B) shows the cluster accuracy.

both the vanilla autoencoder and the k-competitive autoencoder improved the NMI scores. Overall, it can be stated that the features of KATE, which have been observed in [12], do transfer to these difficult settings. Now that an appropriate setting for the k-competitive layer has been found, it will be evaluated against real world data sets.

5.4.4 Real World Data Sets

Experiments with synthetic data sets are an important tool for understanding and they serve as a proof of concept. However, their relevance to practical problems is limited. To complement the previous experiments, this section includes an evaluation on real world data sets. The data sets were chosen so that they have similar properties to the synthetic data sets. This was not straight forward, because the synthetic data sets each focused on a specific problem. In practice, all tested situations can occur at once and with varying degrees. A good compromise could be reached with the data sets from Campos, Zimek, Sander, *et al.* [9].

In their survey paper they wanted to find appropriate benchmark data sets to evaluate different outlier detection methods. These data sets have some properties which make them interesting for representation learning. They include natural[9] outliers[10], mixed type data and duplicate values. Campos, Zimek, Sander, *et al.* [9] chose 23 data sets from the UCI Machine Learning Repository [4], which they considered good benchmarks. For each of these they generated various versions, where each version had different preprocessing steps, e.g. normalization, removal of duplicate values, downsampling of the outlier class and different handling of categorical attributes.

[9]Natural outliers refer here to outliers that arise from the semantics of the data set, e.g. sick vs healthy patients in medicine. Natural is used here in contrast to artificially downsampling one class to be the minority, which is often done for outlier detection evaluation[9].

[10]An outlier refers here to "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [19].

To be clear the goal of this thesis is not to build an outlier detection algorithm. The outlier data sets serve as a test on how robust the different autoencoders are, in consideration of performance and learned low dimensional representation. These experiments will help to gather evidence for the following hypothesis.

There is not one optimal autoencoder architecture for every situation, instead they should be selected for different situations. Similar to the use of Convolutional Neural Networks [32] for image data or LSTMs [23] for sequential text data. According to the synthetic experiments, the k-competitive autoencoder is more effective with cross sectional data that include outliers, noise and redundancy. To evaluate this hypothesis experiments on real world data sets are necessary. From the selection by [9], five data sets were chosen for evaluating the algorithms. All five data sets have more than ten dimensions and at least five percent of outliers, the details of the data sets are briefly discussed.

Annthyroid (AnnthX {δ})

The Annthyroid data set contains medical data on hypothyroidism, which is a disorder of the endocrine system. It consists of three classes "normal", "hyperfunction" and "subnormal functioning", from which classes other than normal were labeled as outliers by [9]. The data set includes 7200 observations and 21 features, from which 15 are categorical and six are numerical. The experiments include four different versions of this data set. Two have $X = 5$ percent of outliers, which was achieved by artificial downsampling of the data. The other two versions have $X = 7$ percent of outliers, which corresponds to the real amount of outliers. Each of these two is then split in a version which includes duplicate values or not, indicated by the "δ" symbol.

Cardiotocography (Cardio22)

The Cardiotocography data set contains data of heart diseases and consists of three classes referring to different diagnoses, "normal", "suspect" and "pathological". Again, classes other than normal were labeled as outliers by [9]. The data set has 2126 observations, 471 outliers (22 percent) in total and is 21-dimensionsal. The version used in the experiment is normalized and contains no duplicate values.

HeartDisease (HD44)

The HeartDisease data set contains data about heart conditions, were again healthy patients are labeled as inliers and the rest as outliers. It consists of 270 observations, from which 120 are outliers (44 percent) and has 13 numerical features.

Hepatitis (Hepa16)

The Hepatitis data set contains predictions of whether a patient infected with hepatitis will die or survive. Fatal cases are labeled as outliers. The data set consists of 80 observations, 13 outliers and 19 numerical features.

WPBC (WPBC24)

The Wisconsin Prognostic Breast Cancer (WPBC) has been used in the literature before [26] and is therefor included in the survey of [9]. Keller, Muller, and Bohm [26] labeled the class "R" as outlier and class "N" as inlier. The data contains 198 instances, 47 outliers (about 24 percent) and 33 numerical features.

Model Setting

All models listed in table 5.5 were trained and evaluated on the above data sets. Each of them was split into training (70 percent), evaluation (10 percent) and test (20 percent) data sets. Each model was trained for 5 epochs, except IDEC which was pretrained for 200 epochs. After training, the encoding was constructed for the whole data set on which k-means++ was executed.

As the data sets vary in size and number of samples, the parameters had to be adapted. The details are as follows. For the small data sets with less than 1000 instances the batch size was set to four, with a learning rate of 1 and for the large Annth and Cardio22 data sets it was set to 64, with a learning rate of 1.8. Note, that for all architectures except IDEC, the AdaDelta [66] optimizer was used, which adjusts its learning rate automatically. IDEC uses the ADAM [27] algorithm, and the learning rate was left at the default value of 0.001.

The abbreviation **(ML)** in table 5.5 refers to the use of multiple layers. In case of the k-competitive layer this means that competition is done in every layer with k set equal to 40 percent of hidden layer units. For k-sparse and WTA the original papers only did the competition in the hidden layer, which was applied here as well. For the Annth and Cardio22 data set three hidden layers were used with 20x10x6 dimensions and for the remaining data sets the dimensionality was 10x6x6. **(OC)** refers to an overcomplete hidden layer, were the hidden layer's dimensionality is larger than the input's. Again, different sizes were chosen. This was done to compensate for the difficulty and dimensionality of each data set. The size of the overcomplete layer is constrained by the number of samples a data set consists of. For each data set different sizes where evaluated, which resulted in the following choices. For the WPBC 38 hidden units were chosen, because despite its high dimensionality (33) there is only a small amount of samples (198). The Annthyroid and Cardiotocography have over 2000 samples and both 21 dimensions, which is why the layer size was selected to be almost twice as large, with 40 hidden neurons. The Hepatitis and HeartDisease data set are both lower dimensional and have few instances, which is why the hidden layer size was set to 20.

Results

The ROC-AUC values for all data sets are reported in table 5.6. The first four columns show the different variants of the Annthyroid data set, where δ denotes that duplicate values were not removed.

As mentioned before the goal of this thesis is not to build an outlier detection algorithm, but the outlier benchmark data sets serve as an interesting setting for the autoencoders. That is why the best outlier detection algorithms from [9] are out of competition, but their ROC-AUC values are still reported. It is interesting to see that with a learned representation and k-means++ the ROC-AUC values get quite close to

Model	Activation	Initialization	Hidden Layers
AE	tanh	xavier	1
AE (ML)	tanh	xavier	3
AE (OC)	tanh	xavier	1
k-comp (ML)	tanh	xavier	3
k-comp (OC)	tanh	xavier	1
k-comp (relu)	relu	he	1
k-comp (tanh)	tanh	xavier	1
k-sparse	linear	xavier	1
k-sparse (ML)	linear	xavier	3
k-sparse (OC)	linear	xavier	1
WTA	relu	he	1
WTA (ML)	relu	he	3
WTA (OC)	relu	he	1
IDEC	relu	xavier	1

TABLE 5.5: The experimental setting of all models for the real world outlier data in section 5.4.4.

the best performers. Except for the WPBC24 data set, none of the models can beat the outlier detection methods. For the k-competitive layer it can be seen once again, that the tanh activation works best across the different data sets. Among the autoencoder based techniques it can be seen that the highest values are at the competitive architectures, k-comp, k-sparse and WTA. The vanilla autoencoder (AE) is performing quite consistently, but cannot outperform the competition driven architectures. This is even the case for IDEC, which uses a denoising autoencoder[60] during pretraining. This might be due to the two step approach of IDEC. In which the denoising autoencoder failed to learn a good representation during pretraining, which resulted in a bad initial clustering. This in turn forced points to be attracted to the wrong clusters during fine tuning.

	Annth5 δ	**Annth5**	**Annth7 δ**	**Annth7**	**Cardio22**	**HD44**	**Hepa16**	**WPBC24**
AE	0.58	0.63	0.36	0.39	0.60	0.25	0.77	0.55
AE (ML)	0.48	0.48	0.49	0.49	0.46	0.67	0.41	0.49
AE (OC)	0.58	0.50	0.62	0.54	0.37	0.66	0.26	0.40
k-comp (ML)	0.34	0.64	**0.67**	0.31	0.60	0.30	0.80	0.47
k-comp (OC)	0.39	0.56	0.62	0.38	0.64	0.73	0.66	0.38
k-comp (relu)	0.37	0.38	0.42	**0.65**	**0.68**	0.69	**0.83**	0.40
k-comp (tanh)	**0.62**	**0.68**	0.39	0.63	0.64	**0.75**	**0.83**	0.39
k-sparse	0.41	0.39	0.41	0.39	0.58	0.40	0.23	**0.59**
k-sparse (ML)	0.38	0.60	0.62	0.46	0.59	0.70	0.13	0.38
k-sparse (OC)	0.57	0.38	0.39	0.60	0.58	0.27	0.77	0.41
wta	0.55	0.51	0.42	0.43	0.47	0.52	0.42	0.37
wta (ML)	0.51	0.54	0.50	0.52	0.57	0.67	0.37	0.57
wta (OC)	0.53	0.51	0.52	0.50	0.52	0.26	0.23	0.51
PCA	0.39	0.38	0.38	0.38	0.42	0.23	0.80	0.40
k-means	0.39	0.39	0.40	0.39	0.42	0.24	0.23	0.40
IDEC[18]	0.50	0.49	0.52	0.50	0.31	0.25	0.28	0.48
Best from [9]	0.71	0.73	0.69	0.69	0.68	0.79	0.83	0.58

TABLE 5.6: In the above table are the ROC-AUC values for each model. A δ next to a data set denotes that duplicate instances were not removed. (ML) refers to the usage of multiple layers and (OC) to overcomplete hidden layers, for details see 5.4.4. The highest values of the models are highlighted. The last row shows the best performing outlier detection methods from the Campos, Zimek, Sander, *et al.* [9] survey.

6 Conclusion

Deep learning research has mostly focused on image, text and speech data. This applies to competition driven architectures [12], [40], [41] as well. Their performance on cross sectional data sets with a high amount of noise, outliers or redundancy is not as well studied. In this thesis one of these architectures, the k-competitive autoencoder for text (KATE) [12] was extensively studied and its robustness to the above settings has been evaluated. KATE has shown promising results for text data, which is high dimensional and sparse. The idea was that this characteristic would transfer to similar settings, if KATE is adapted to non-text data.

The approach to adapt KATE was first, to better understand the effect of the k-competitive layer by conducting several synthetic experiments. To isolate it from any other effects it was evaluated without an activation functions. The results showed that the competition among neurons increases the distinctiveness in and between neurons. For simple settings with two clusters, this resulted in nicely separated distributions in both neurons, like figure 5.6. When the same is used for a setting with four different clusters, the regularization effect of competition merges cluster together, see figure 5.4. This demonstrates the limited capacity when using only two neurons and the strong effect of regularization. This was also observed in the higher test error, when comparing the reconstruction with the original data.

After that, different activation functions were evaluated, to test the non-linear effects on competition. Extensive experiments showed that the tanh activation is the best choice over a range of different tasks, this is in accordance to [12]. After that, the k-competitive layer was evaluated on synthetic data with noise points, noise dimensions and outliers. The results showed that the k-competitive layer performed best.

The above experiments resulted in the following choices for the k-competitive autoencoder. The loss function is changed from cross entropy to mean squared error. The tanh activation function and the Adadelta optimizer are retained. While the experiments showed that the architecture can stay mostly the same, several practical guidelines for tuning have been found. Contrary to Chen and Zaki [12], who argue that the hyperparameter α should be set to $\frac{k}{2}$, the experiments showed that the choice of α is very important. A hyperparameter search should be conducted around the value of $\frac{k}{2}$. The number of winner neurons k has to be chosen, in consideration with the number of hidden neurons. The problem is that it frequently happens that competition overtakes the reconstruction ability, which results in distorted encodings.

To complement the synthetic experiments, suitable real world data sets where chosen from [9]. These data sets include many outliers and redundant values, which serve as a nice setup to test the robustness of the architectures. To be clear the goal of this thesis is not to build an outlier detection algorithm, but use the data sets as a benchmark. The results show that, among the autoencoders the highest performance was achieved with the competition based architectures, k-comp, k-sparse and WTA. They outperformed IDEC as well, which might indicate that the denoising autoencoder used

in the pretraining phase is not a suitable choice for data with outliers. The rational here is that the representation it learned results in a bad initial clustering. This gets worse in the fine tuning step, where points are attracted to the wrong clusters.

6.1 Achievements

An in depth empirical evaluation of the k-competitive autoencoder was conducted in comparison to other competition based architectures. The key characteristics of the k-competitive layer have been identified, which are robustness to sparsity[12], noise, redundant dimensions and outliers. This robustness has been evaluated with synthetic and real world data sets. The k-competitive autoencoder has performed better than related architectures and even outperformed IDEC in this situations. Additionally, a weakness in the IDEC[18] architecture to data sets with outliers has been observed during experimenting, while not tested, this is very likely to transfer to related architectures like [62], [63], because they all share the same architecture in the pretraining step.

6.2 Future Works

6.2.1 Short Term Goals

Current state of the art deep clustering algorithms, like DEC, IDEC or DCN all learn a representation and then fine tune the representation for a specific clustering result. A short term goal is to use the k-competitive autoencoder in the IDEC architecture's pretraining phase, when cross sectional data sets with outliers are encountered.

6.2.2 Long Term Goals

The long term goal is to find architectures which can learn representations which are suitable to a specific clustering algorithm, e.g. for k-means the autoencoder should learn a representation which enforces the clusters to be Gaussian distributed.

A Abstract

A.1 English

Recent research that combines deep learning and clustering, often called deep clustering, shows promising results [18], [62], [63]. Most of these techniques share the same approach. First, they train an autoencoder to project the input data to a lower dimension. The lower dimensional representation is then clustered by k-means to get initial cluster labels. From this, the learned representation is refined in a step wise fashion, by attracting the points in a cluster closer to its center. This approach has some issues, e.g. if the autoencoder does not learn a good representation, all subsequent steps are doomed to failure.

Most of deep learning research is concerned with image, text or speech data, thus it is not surprising to see that there are weaknesses for other data types. In this master thesis one such weakness is identified for cross sectional data sets which include outliers. In this setting many autoencoders fail to learn a meaningful representation and therefor distort the initial clustering.

One solution is to use an adapted version of the KATE[12] (k-competitive autoencoder for text) architecture. It can learn a meaningful representation of cross sectional data in settings with noise, redundancy and outliers. This is shown in an empirical study of the competition scheme used in KATE. In this study the k-competitive layer is evaluated and adapted for other data types. The adapted algorithm was then benchmarked on multiple challenging real world data sets, against state of the art autoencoder architectures.

A.2 Deutsch

Die Kombination von Deep Learning und Clustering, oft auch unter dem Namen Deep Clustering vereint, ist relativ neu, zeigt aber bereits vielversprechende Ergebnisse [18], [62], [63]. Die meisten der aktuellen Deep Clustering Techniken teilen den gleichen Ansatz. Zuerst wird ein Autoencoder trainiert, um die Eingangsdaten auf eine niedrigere Dimension zu projizieren. Diese niedrigdimensionale Darstellung wird dann mit dem k-means Algorithmus geclustert, um erste Cluster-Labels zu erhalten. Aus diesem Clustering wird die erlernte Darstellung schrittweise verfeinert, indem die Punkte in einem Cluster näher an sein Zentrum gezogen werden. Dieser Ansatz hat einige Probleme, z.B. wenn der Autoencoder im ersten Schritt keine gute Darstellung lernt, sind alle nachfolgenden Schritte zum Scheitern verurteilt.

Zusätzlich, beschäftigt sich der größte Teil der Deep Learning Forschung mit Bild-, Text- oder Sprachdaten, daher ist es nicht verwunderlich, dass es Schwächen bei anderen Datentypen gibt. In dieser Masterarbeit wird eine solche Schwachstelle für

Querschnittsdatensätzen mit Ausreißern identifiziert. In dieser Situation lernt der Autoencoder keine sinnvolle Darstellung, wodurch das Clustering verzerrt wird.

Eine mögliche Lösung dafür wurde durch die Verwendung einer angepassten Version des KATE[12] Algorithmus (k-competitive autoencoder for text) gefunden. KATE kann eine sinnvolle niedrigdimensionale Darstellung von Querschnittsdaten mit Rauschen, Redundanz und Ausreißern lernen. Das wird durch eine empirische Untersuchung des in KATE verwendeten Wettbewerbsverfahren gezeigt. In dieser Studie wurde der k-competitive layer ausgewertet und für andere Datentypen angepasst. Der angepasste Algorithmus wurde dann mittels mehreren anspruchsvollen Datensätzen gegenüber anderen Autoencoder-Architekturen verglichen.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[2] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding", in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[3] G. Arvanitidis, L. K. Hansen, and S. Hauberg, "Latent space oddity: On the curvature of deep generative models", *ArXiv preprint arXiv:1710.11379*, 2017.

[4] K. Bache and M. Lichman, "Uci machine learning repository (http://archive. ics. uci. edu/ml), university of california, school of information and computer science", *Irvine, CA*, 2013.

[5] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima", *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.

[6] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives", *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[7] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks", in *Advances in neural information processing systems*, 2007, pp. 153–160.

[8] C. Bishop, *Pattern recognition and machine learning*. Springer, 2006. [Online]. Available: `https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/`.

[9] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, "On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study", *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 891–927, 2016.

[10] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées", *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

[11] N. Chen, A. Klushyn, R. Kurle, X. Jiang, J. Bayer, and P. van der Smagt, "Metrics for deep generative models", in *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, 2018, pp. 1540–1550. [Online]. Available: `http://proceedings.mlr.press/v84/chen18e.html`.

[12] Y. Chen and M. J. Zaki, "Kate: K-competitive autoencoder for text", in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 85–94.

[13] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011. [Online]. Available: `http://dl.acm.org/citation.cfm?id=2021068`.

[14] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.", in *Kdd*, vol. 96, 1996, pp. 226–231.

[15] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets", in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[18] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation", in *International Joint Conference on Artificial Intelligence (IJCAI-17)*, 2017, pp. 1753–1759.

[19] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[21] I. Higgins, L. Matthey, X. Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and A. Lerchner, "Early visual concept learning with unsupervised deep learning", *ArXiv preprint arXiv:1606.05579*, 2016.

[22] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups", *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] A. K. Jain, "Data clustering: 50 years beyond k-means", *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[25] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[26] F. Keller, E. Muller, and K. Bohm, "Hics: High contrast subspaces for density-based outlier ranking", in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, IEEE, 2012, pp. 1037–1048.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980.

[28] D. P. Kingma and M. Welling, "Auto-encoding variational bayes", *ArXiv preprint arXiv:1312.6114*, 2013.

[29] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?", *ArXiv preprint arXiv:1805.08974*, 2018.

[30] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering", *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 1, p. 1, 2009.

[31] Y. LeCun, "The mnist database of handwritten digits", *Http://yann. lecun. com/exdb/mnist/*, 1998.

[32] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series", *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[33] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge university press, 2014.

[34] S. Lloyd, "Least squares quantization in pcm", *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[35] W. Maass, "On the computational power of winner-take-all", *Neural Computation*, vol. 12, no. 11, pp. 2519–2535, 2000.

[36] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne", *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[37] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations", in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Oakland, CA, USA, vol. 1, 1967, pp. 281–297.

[38] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten, "Exploring the limits of weakly supervised pretraining", *ArXiv preprint arXiv:1805.00932*, 2018.

[39] ——, "Exploring the limits of weakly supervised pretraining", *ArXiv preprint arXiv:1805.00932*, 2018.

[40] A. Makhzani and B. J. Frey, "K-sparse autoencoders", *CoRR*, 2013. arXiv: 1312.5663. [Online]. Available: http://arxiv.org/abs/1312.5663.

[41] ——, "Winner-take-all autoencoders", in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 2791–2799. [Online]. Available: http://papers.nips.cc/paper/5783-winner-take-all-autoencoders.

[42] J. Mao and A. K. Jain, "A self-organizing network for hyperellipsoidal clustering (hec)", *Ieee transactions on neural networks*, vol. 7, no. 1, pp. 16–29, 1996.

[43] A. P. Minsky Marvin, *Perceptrons*. MIT Press, 1969.

[44] G. E. Moore, "Cramming more components onto integrated circuits", *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[45] K. Murphy, *Machine learning, a probabilistic perspective*. MIT Press, 2012.

[46] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[47] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images", *Nature*, vol. 381, no. 6583, p. 607, 1996.

[48] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch", 2017.

[49] K. Pearson, "On lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[51] D. Pelleg, A. W. Moore, *et al.*, "X-means: Extending k-means with efficient estimation of the number of clusters.", in *Icml*, vol. 1, 2000, pp. 727–734.

[52] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models", *ArXiv preprint arXiv:1401.4082*, 2014.

[53] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction", in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, Omnipress, 2011, pp. 833–840.

[54] G van Rossum, "Python tutorial, technical report cs-r9526, centrum voor wiskunde en informatica (cwi), amsterdam."", 1995.

[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, p. 533, 1986.

[56] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis", in *International Conference on Artificial Neural Networks*, Springer, 1997, pp. 583–588.

[57] C. K. Sø nderby, T. Raiko, L. Maalø e, S. r. K. Sø nderby, and O. Winther, "Ladder variational autoencoders", in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 3738–3746. [Online]. Available: http://papers.nips.cc/paper/6275-ladder-variational-autoencoders.pdf.

[58] V. N. Vapnik, "The nature of statistical learning theory", 1995.

[59] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need", in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[60] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders", in *Proceedings of the 25th international conference on Machine learning*, ACM, 2008, pp. 1096–1103.

[61] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion", *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[62] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis", in *International conference on machine learning*, 2016, pp. 478–487.

[63] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering", in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 3861–3870. [Online]. Available: `http://proceedings.mlr.press/v70/yang17b.html`.

[64] L. Yann, "Modeles connexionnistes de lapprentissage", PhD thesis, PhD thesis, These de Doctorat, Universite Paris 6, 1987.

[65] A. L. Yuille and D. Geiger, "Winner-take-all mechanisms", 1995.

[66] M. D. Zeiler, "ADADELTA: an adaptive learning rate method", *CoRR*, vol. abs/1212.5701, 2012. arXiv: `1212.5701`. [Online]. Available: `http://arxiv.org/abs/1212.5701`.

[67] A. Zimek and J. Vreeken, "The blind men and the elephant: On meeting the problem of multiple truths in data from clustering and pattern mining perspectives", *Machine Learning*, vol. 98, no. 1, pp. 121–155, 2015, ISSN: 1573-0565. DOI: `10.1007/s10994-013-5334-y`. [Online]. Available: `https://doi.org/10.1007/s10994-013-5334-y`.