Pattern Classification Using Stacked Autoencoders

Philipp Grafendorfer, Christina Kastner, Karin Lassnig, Tamara Maier, Alexander Rothschadl

Abstract

Our project dealt with stacked autoencoders. These are a specific type of neural networks which are trying to learn an approximation to the identity function, so as to give an output that is similar to the input. At first sight this seems to be a pretty simple task, but autoencoders can be used for a wide range of interesting exercises. In addition, they give a good insight into the functionality of neural networks and motivate a lot of exciting experiments.

In our work, we had to focus on a couple of important applications. We used autoencoders for classifying our data, for dimensionality reduction and for denoising.

On the following pages we will describe the theoretical foundations and the relevant results we achieved.

Introduction

This paper shall give an overview of the topics we dealt with during our project. We will explain what we did, which methods we used and which results we obtained.

In the first section, we discuss the theoretical background. We explain what neural networks are and how they work. After this we can define autoencoders and show some of their most interesting characteristics.

Subsequently we move on to the practical part. We describe the data sets that we used as well as the experiments we constructed.

Finally we take a closer look at our results, where we highlight the most fascinating outcomes.

Theoretical Background of Autoencoders

Artificial Neural Networks

Artificial neural networks (ANN's) are inspired by biological neural networks, but can also be explained without them. Remarkable advantages over other deep learning concepts are the high degree of parallel processing and the fact that they yield comparatively good results.

Components. Artificial neural networks consist of

- Neurons (simple processing units)
- Connections and weights (directed weighted connections between the neurons of consecutive layers)
- Activation function(s) (yield(s) the "strength" of each neuron)
- Learning rule (e.g., back-propagation)

Furthermore, an artificial neural network is characterized by the following layer structure:

- one input layer
- one or more hidden layers
- one output layer

Notation. σ ... activation function
a^i_j ... activation value of the j-th neron in the i-th layer
w_{jk} ... weight from the k-th neuron in the $(i-1)$-th layer to the j-th neuron in the i-the layer
b^i_j ... bias of the j-th neuron in the i-th layer

Activation function.

$$a^i_j = \sigma\left(\sum_k \left(w^i_{jk} \cdot a^{i-1}_k\right) + b^i_j\right)$$

We can choose from different kinds of activation functions. Each of these has its strengths and weaknesses. A good choice is one that assures a fast training process by a fast approximation.

Sigmoid/logit function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

WIR HABEN BISHER DAS WORT 'STACKED' FALSCH VERSTANDEN

Relu (rectified linear unit) function:

$$\sigma(z) = \max(z, 0)$$ (EINFACH CALC.)

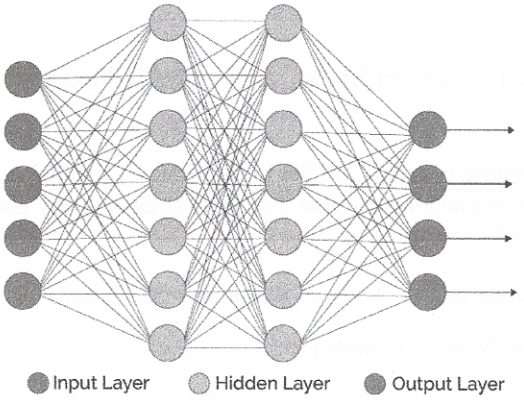


Figure 1. Model of a an ANN with two hidden layers. FONT KLEINER

Autoencoders

An autoenccoder is a special neural network, which is used for ~~the~~ unsupervised learning of efficient codings. The objective is to learn a compressed representation for a set of data and therefore extracting essential features P
(→ dimensionality reduction). More precisely, an autoencoder tries to learn an approximation to the identity function with the goal of producing an output that is similar to the input.

An autoencoder consists of three components:

- Encoder: compresses the input and produces the code.
- Code
- Decoder: reconstructs the input using only this code.

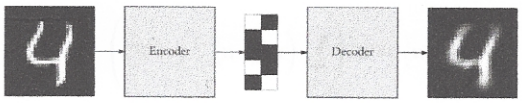
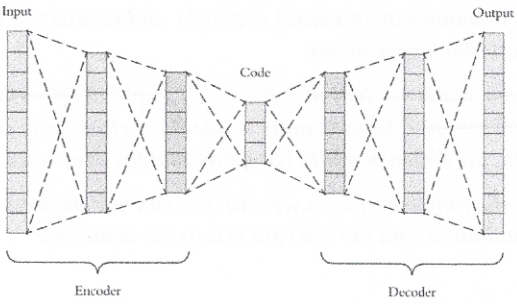


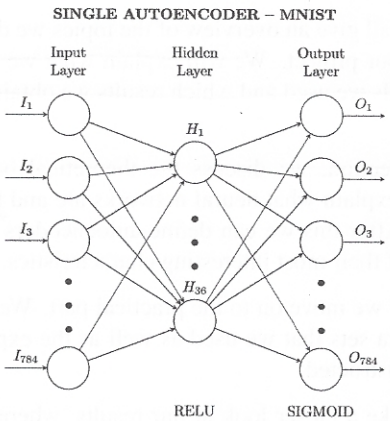
Figure 2. Example of an autoencoder.
→ ● BILD KOMMT VON KERAS BLOG

Properties of Autoencoders

- Number of layers: Just like a general artificial neural network, an autoencoder consists of one input layer, one ("single-layered") or more ~~("stacked")~~ hidden layers and one output layer.
- Data-specific: Autoencoders can only handle compression tasks meaningfully for which they were trained to. For instance, an autoencoder which was trained on handwritten digits will not handle landscape photos properly.
- Lossy: The output will be poorer than the input, since it is a weaker representation.
- Unsupervised learning: The input patterns are not labeled. Instead, the autoencoder itself tries to detect similarities and to generate pattern classes.
- Loss function: Measures the information loss between the compressed representation of the data and the decompressed representation. One possible choice is the mean squared error.



CAP.?



CAP.?

Linear Autoencoders → ANHANG

We now want to look at two interesting results regarding linear autoencoders. These are autoencoders with a linear activation function.

A linear autoencoder is optimal under certain conditions. This means, it yields a reconstruction error that is

SEMI-SUPERVISED LEARNING

– for a given number of hidden layers – smallest possible. We assume that the decoder is linear and we consider the squared difference error.

Proof: For the proof we will use the following theorem.

Theorem: Let A be an arbitrary matrix with singular value decomposition:

$$A = U\Sigma V^T,$$

where U, V are orthogonal matrices and Σ is a diagonal matrix.

Let $U_{\cdot<k}, \Sigma_{\cdot<k,<}, V^T_{\cdot<k}$ be the decomposition where we only keep the k largest singular values.

Then the matrix B of rank k ($k <$ rank$A$) that is closest to A[1]

$$B^* = \text{argmin}_{B \text{ such that rank}B=k} \|A - B\|_F$$

is

$$B^* = U_{\cdot<k}\Sigma_{<k,\leq k}V^T_{\cdot\leq k}.$$

Now we can start with the sketch of the proof.

When we train the autoencoder, we want to minimize the sum of squared differences:

$$\min_? \sum_t \frac{1}{2} \sum_i \left(X_i^{(L)} - \underbrace{\hat{X}_i^{(L)}}_{\text{based on linear autoencoder}} \right)^2 \geq \min_{W^*h(x)} \frac{1}{2} \|X - W^*h(X)\|_F^2,$$

where
$X^{(L)}$ ... training examples
X ... matrix where the columns are the $X^{(L)}$'s
$h(X)$ ... matrix of all hidden layers
W^* ... decoder weights

$$\text{argmin}_{W^*h(x)} \frac{1}{2}\|X - W^*h(X)\|_F^2 = \left(W^* \leftarrow U_{\cdot<k}\Sigma_{<k,<k}, h(X) \leftarrow V^T_{\cdot<k}\right)$$

based on the previous theorem, where $X = U\Sigma V^T$ and k denotes the size of the hidden layer.

[1] That means we want to find an approximation B of A that is less complicated. We want to find an approximation that minimizes the Frobenius norm.

Let's show that $h(X)$ is a linear encoder.

$$\begin{aligned}
h(X) &= V^T_{\cdot<k} \\
&= V^T_{\cdot<k}(X^TX)^{-1}(X^TX) \\
&= V^T_{\cdot<k}(V\Sigma^TU^TU\Sigma V^T)^{-1}(V\Sigma^TU^TX) \\
&= V^T_{\cdot<k}(V\Sigma^T\Sigma V^T)^{-1}V\Sigma^TU^TX \\
&= V^T_{\cdot<k}V(\Sigma^T\Sigma)^{-1}\Sigma^TU^TX \\
&= V^T_{\cdot<k}V(\Sigma^T\Sigma)^{-1}\Sigma^TU^TX \\
&= I_{\cdot<k}(\Sigma^T\Sigma)^{-1}\Sigma^TU^TX \\
&= I_{\cdot<k}\Sigma^{-1}(\Sigma^T)^{-1}\Sigma^TU^TX \\
&= I_{\cdot<k}\Sigma^{-1}U^TX \\
&= \Sigma^{-1}_{\cdot\leq k,\leq k}(U_{\cdot\leq k})^TX
\end{aligned}$$

Thus, we have an encoder that was computed by taking all the inputs and linearly transforming them. This means that

$$\underbrace{\Sigma^{-1}_{\cdot<k,\cdot<k}(U_{\cdot<k})^T}_{=:W}$$

is a linear encoder.

Hence, $h(X)$ can be expressed as a linear encoder. $h(X)$ is even the best solution for the minimization problem. Therefore, an optimal pair of encoder and decoder is

$$\underbrace{\Sigma^{-1}_{\cdot<k,\cdot<k}(U_{\cdot<k})^T}_{=W} \quad \text{and} \quad \underbrace{(U_{\cdot<k}\Sigma_{\cdot<k,<k})}_{=W^*}h(X)$$

for the sum of squared errors difference
for an autoencoder with a linear decoder,
where optimality means "has the lowest training reconstruction error". As the training error converges to the generalization error if we have more and more training data, a linear autoencoder will do extremely well for a large amount of training data.

If the inputs are normalized as follows:

$$X^{(L)} \leftarrow \frac{1}{\sqrt{T}}\left(\underbrace{X^{(L)}}_{\text{input}} - \underbrace{\frac{1}{T}\sum_{L'-1}^{T} X^{(L')}}_{\text{average of all training inputs}}\right);$$

the autoencoder corresponds to the Principal Component Analysis (PCA). To show that an autoencoder with linear activation functions is equivalent to PCA, we look at the simplified case of an autoencoder with only one hidden layer.

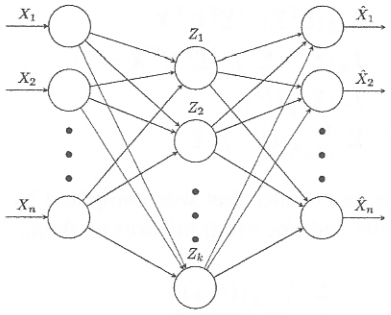
At first, we look at the functionality of PCA:

Let X be the original data, z the reduced data and $\hat{X}$ the reconstructed data from the reduced representation.

Then we can write PCA as

$$\begin{aligned} Z &= B^T X \\ \hat{X} &= BZ \end{aligned}$$

Now take a look at an autoencoder with the following architecture:



CAP?

with basically $X \to Z \to \hat{X}$.

Since the activation functions are linear, $\mathcal{O}(X) = X$, we can write the autoencoder as

$$\hat{X} = W_1 W_2 X,$$

where W_1, W_2 are the weights of the first and the second layer.

Now, if we set $W_1 = B$, $W_2 = B^T$, we obtain

$$\hat{X} = W_1 W_2 X = W_1 (W_2 X) = W_1 Z = BZ,$$

which is the same solution that we had for PCA.

This equivalence is obviously only valid for autoencoders that have a bottleneck layer smaller than the input layer.

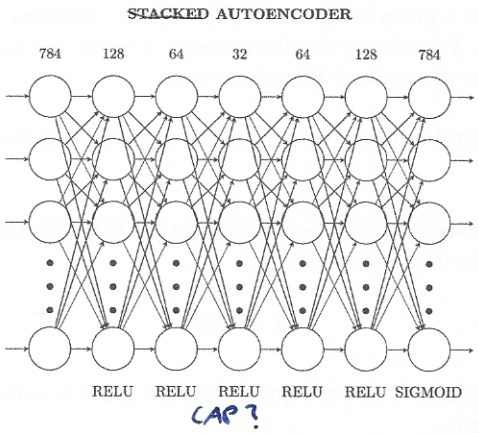
DEEP

~~Stacked~~ Autoencoders

Stacked autoencoders have multiple hidden layers. We start with the raw inputs to the first hidden (feature) layer and obtain the first-order features. The outputs of the first hidden layer are passed to the second hidden layer, whereupon we get the second-order features and so on – until we reach the last feature layer. From there the values are passed to a classifier. To improve the results we can apply some kind of backpropagation (e.g. gradient descent).

DER IST NICHT STACKED

DEEP



CAP?

STACKED AUTOENCODER BESCHREIBUNG

Experimental Setup

Description of Datasets

Semeion. The `Semeion` dataset consists of 1,593 handwritten digits from 0 to 9 written by 80 different persons. The handwritten digits were saved in a 16×16 pixel image, whereas a white pixel was scaled to 0 and a black pixel was scaled to 1.
Every person was asked to write each digit twice – at first accurately and then in a faster way. The dataset consists of 256 attributes (columns) of binary values, which will be used as input variables and 10 columns from 0 to 9 representing the output class. The dataset contains 1,593 records (rows) for 1,593 handwritten digits.

MNIST. The `MNIST` dataset is a combination of two `NIST`'s databases: Special Database 1 and Special Database 3. These databases also contain handwritten digits between 0 and 9 written by high school students and employees of the United States Census Bureau.
The difference between `Semeion` and `MNIST` is that in `MNIST` the digits were saved in a 28×28 pixel image, whereas a pixel was scaled between 0 and 255. The scaling is indicating the lightness or darkness of a pixel, with higher numbers meaning darker. The `MNIST` dataset has a set of 60,000 training examples and 10,000 test examples.

Ionosphere. The `Ionosphere` dataset has 351 instances and 34 attributes. It classifies the radar returns from the ionosphere in two classes. If the radar signals show evidence of some type of structure in the ionosphere the instances are classified as "good". The radar signal of instances classified as "bad" passed through the ionosphere.

RAUS NEHMEN

ZU WENIG DATEN

ROBO. The data was collected as the robot SCITOS G5 navigated through a room moving along the wall in a clockwise direction. The robot used 24 ultrasound sensors arranged circularly around its "waist" to navigate through a room. The numbering of the ultrasound sensors starts at the front of the robot and increases in clockwise direction.
The dataset consists of 5,456 instances and 24 attributes, one for every ultrasound sensor. The instances are assigned one of the four classes: "Move-Forward", "Sharp-Right-Turn", "Slight-Left-Turn" or "Slight-Right-Turn" depending on the direction the robot should move.

Our Experiments

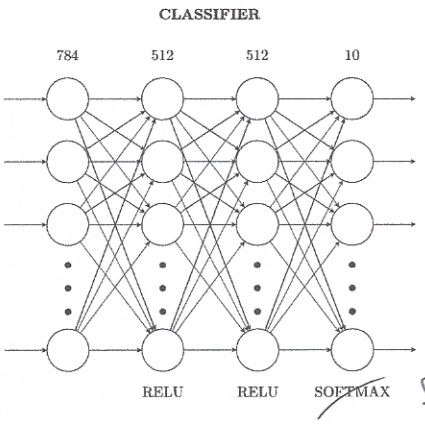
Classifier

BEST TRY TO GET THE BEST CLASSIFIER WITH FNN.

For the classifier we introduce a new function, the so-called **softmax function** (also known as **normalized exponential function**). It is a generalization of the logistic function, which squashes a K-dimensional vector $\mathbf{z}$, where $z_1, \dots, z_K \in \mathbb{R}$, to a normalized vector $\sigma(\mathbf{z})$ of real values in the range $[0, 1]$, where the sum of the coordinates is equal to 1.

$$\sigma : \mathbb{R}^K \to [0,1]^K, \quad \sigma(\mathbf{z})_j \mapsto \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}},$$

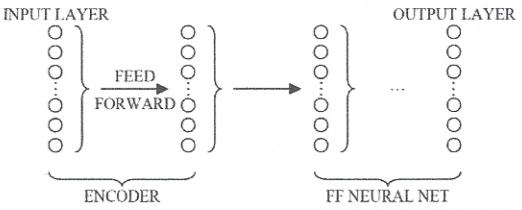
for $j \in \{1, \dots, K\}$.



SIGMOID

WIT STACKED AE

Dimensionality Reduction (~~Feature Selection~~). We investigate, how the number of features influences the accuracy.

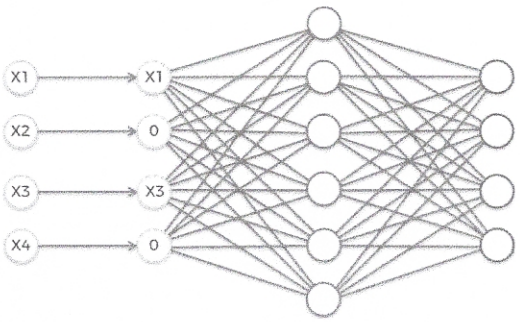


WE COULD DROP THE DECODING LAYER

WITH STACKED AE

Denoising. When there are more nodes in the hidden layer than there are inputs, the network is risking to learn the identity function (instead of extracting features), meaning that the output equals the input, making the autoencoder virtually useless. Denoising autoencoders solve this problem by corrupting the data on purpose by randomly turning some of the input values to zero.

APPENDIX



- CLASSIFIER WITH STANDARD NEURAL NET
DATA/
- IMAGE RECONSTRUCTION WITH STACKED AUTOENCODER
~~- IMAGE RECONSTRUCTION WITH DEEP AUTOENCODER~~

0. ABSTRACT FIG. CAPTION KLEINER

1. INTRODUCTION

THEORETICAL BACKGROUND → WEGLASSEN

2. ANN

3. AUTOENCODERS

- PROPERTIES

LINEAR AUTOENCODERS → ANHANG

- DEEP AUTOENCODER
- STACKED AUTOENCODER → SCHREIBE ICH NOCH

4. EXPERIMENTAL SETUP

- 'STANDARD NEURAL NET'
- TRAIN STACKED AUTOENCODER
- DATA COMPRESSION & RECONSTRUCTION
- CLASSIFICATION
- DENOISING

5. EXPERIMENTS

- MNIST / DESCRIPTION → MNIST_SAE_COMMENTS.ZIP
- ROBO
- SERIES