University of Miskolc
Institute of Information Science
Department of Information Technology

Single Element Feedforward Neural Network Inversion Methods in Python

Student Research

Készítette:
Lilla Juhász
FWIY50

Témavezető:
Bence Bogdándy

2019.

Szerzői Nyilatkozat

Alulírott Juhász Lilla, a Miskolci Egyetem Gépészmérnöki- és Informatikai Karának hallgatója büntetőjogi és fegyelmi felelősségem tudatában kijelentem és aláírásommal igazolom, hogy ezt a dolgozatot saját magam készítettem, a benne leírt vizsgálatokat – ha ezt külön nem jelzem – magam végeztem el, és az ismertetett eredményeket magam értem el. Adatokat, információkat csak az irodalomjegyzékben felsorolt forrásokból használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Miskolc, April 8, 2019

Lilla Juhász

Contents

Szerzői Nyilatkozat **1**

1 Introduction **3**

2 Related Works **4**
- 2.1 Data Mining . . . 4
  - 2.1.1 The Dataset . . . 4
- 2.2 Machine Learning . . . 5
  - 2.2.1 Artificial Neural Networks . . . 5
  - 2.2.2 ANN Inversion . . . 6
- 2.3 Python . . . 6

3 Theoretical Background **8**
- 3.1 Feature Engineering . . . 8
  - 3.1.1 Data Mining . . . 8
- 3.2 Single-Layer Perceptron . . . 11
- 3.3 Feedforward Neural Networks . . . 12
  - 3.3.1 Multi-Layer Perceptron . . . 12
- 3.4 Training a MLP Model . . . 14
  - 3.4.1 Regression . . . 14
  - 3.4.2 Activation Functions . . . 14
  - 3.4.3 Optimization Methods . . . 15
- 3.5 Inversion . . . 17
  - 3.5.1 Single-Element Inversion Methods . . . 17
  - 3.5.2 Williams-Linder-Kindermann Inversion . . . 18

4 Implementation **19**
- 4.1 Problem Statement . . . 19
  - 4.1.1 Used Third-Party Libraries . . . 20
- 4.2 The Implementation . . . 23
  - 4.2.1 Optimizing the Dataset . . . 24
  - 4.2.2 Training the Neural Network . . . 25
  - 4.2.3 Inverting the MLP . . . 26
- 4.3 Results . . . 28

5 Summary **29**

Chapter 1

Introduction

megirni

Chapter 2

Related Works

2.1 Data Mining

Data mining [31][13] is an interdisciplinary subfield of computer science and statistics, that is represented as a process of discovering patterns and inferences in large data sets. It is designed to extract information from a data set and transform it into a comprehensible structure for further use.

Data mining is a tool for information extraction, processing, representation and summarization.

Stages of data mining:

- Selection: The information that seems to be useful for the prediction needs to be selected.
- Preprocessing: The dataset may include errors, missing values or inconsistent data that need to be filtered out.
- Transformation: Some features of the dataset need some transformation, like normalization, to get an appropriate form.
- Data mining: Now the data mining techniques need to be used that can discover the patterns.
- Evaluation: The patterns are known hence it can be seen not all of them are needed for the prediction.

Data mining has applications in multiple fields, not just like in science and research, but also in business, banking, manufactoring, insurance and so on. It can help companies to develop more effective strategies in a more optimal way.

2.1.1 The Dataset

Selecting the appropriate information that can be useful for the dataset is not as easy as it seems. A well-defined dataset is large-scale and carries a lot of information whereof patterns can be predicted. Collecting enough relevant data is a long process, but there are public datasets that can be used for free.

Online News Popularity dataset [11] - which is a prediction of Mashable news popularity - is publicly available at UCI Machine Learning repository. It aims on predicting the future popularity of news articles from given information that are known before the release of news articles.

Mashable is a digital media website that is founded in 2005. Online News Popularity dataset [25] contains information about 40.000 articles published between 2013 and 2015 on Mashable's website.

Online News Popularity dataset consists of 58 predictive features, 2 other attributes of accessory information and 1 goal field, which is the number of shares. There were nominal features like the day of the publication or the type of the data channel. They were transformed by one-hot encoding. Now all of the predictive features are numeric values. Three types of keywords such as worst, average and best were captured by ranking all articles keyword average shares. Additionally, a bunch of natural language processing features were extracted such as closeness to top Latent Dirichlet Allocation (LDA) topics, title subjectivity, the rate of positive and negative words and title sentiment polarity. Sentiment polarity and subjectivity scores were also computed.

Online News Popularity dataset meets all the requirements for being a well-used dataset and its preprocessing and transformation has already made. However the dataset is not prepared for applying data mining techniques, it even needs some cleaning.

2.2 Machine Learning

Machine learning [18][17][3] is a field of artificial intelligence that gives computers the capability to learn. Learning relies on patterns and inferences, instead of explicit instructions. Machine learning utilizes statistical methods to process predefined data sets and to predict future output values for given data inputs.

Application fields of machine learning is enormous. Machine learning can be used during data processing, such as from the feature selection phase to applying data mining methods, as machine learning models are builded from data mining techniques.

There are also well-known techniques that can be used after analyzing and optimizing the data. Two of the most widely adopted machine learning methods are supervised learning and unsupervised learning.

Supervised learning provides labeled training data, which are used during the prediction phase. Supervised learning's algorithm analyzes the given data set and processes the labeled data for mapping new examples. There are several types of supervised learning that can be divided into two main classes that are classification and regression. In **classification**, the samples can only be discrete types, but in **regression**, the desired output consists of one or more continuous variables and real numbers.

Unsupervised learning is another category of the learning problem, in which the training data consists of a set of input vectors without any corresponding target values. One of the goals in these problems may be to discover groups of similar attributes within the data set, which is called **clustering**.

2.2.1 Artificial Neural Networks

The neural network is a system with some functionalities of the human brain. It is designed to recognize patterns and process all real-world data with respect of these patterns.

The artificial neural network [22][4] is a machine learning model which uses artificial neurons to model the complex process of the human brain. During the learning process, these neurons receive inputs, then change their internal state, which is called activation, to produce the desired outputs. This activation phase processes the given inputs from one neuron to another.

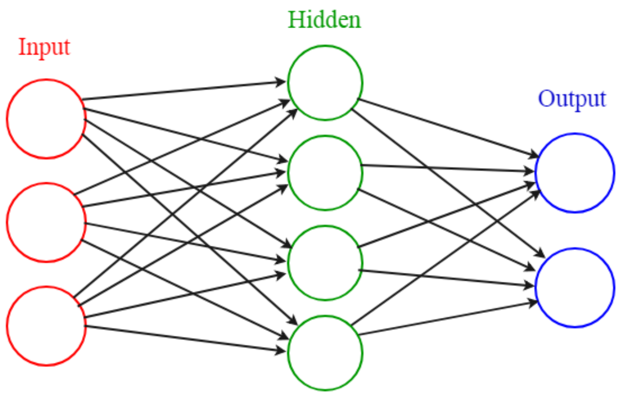


Figure 2.1: A neural network, where the circles represent the artificial neurons as nodes

The artificial neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms that can process complex data inputs. These learning algorithms learn from the experiences that are obtained from processing many examples. Each process yields an output, which, depending on the other outputs can determine which characteristics of the input are needed to construct the correct output. If there are a sufficient number of processed examples, the neural network can generate potential inputs and see if they produce the correct outputs. As the quantity of training data increases, so does the learning time and complexity of the neural network. However, training data also potentially increases the estimation accuracy.

2.2.2 ANN Inversion

There is a subfield of machine learning that have not received much attention since the rise of deep learning. This subfield is the inversion problem [16].

Neural network inversion procedures seek to find one or more input values that produce a desired output response for a fixed set of synaptic weights.

There are many methods which can perform neural network inversion. These can be placed into three broad classes. Exhaustive search should be considered when the dimensionality of the input and allowable range of each input variable is low. Single-element inversion methods are used to find one inversion point per process. Multi-element inversion procedures are able to find numerous inversion points simultaneously.

Even if the inversion problem is not the most popular area in connection with neural networks, it has wide application areas. For example, a single-element inversion task is the problem of sonar performance under various environmental conditions [1].

2.3 Python

Python [28][29] is an interpreted, object-oriented, high-level programming language with dynamic semantics, used for general-purpose programming. It was created by Guido van Rossum and released in 1991.

Python has became a prefered programming language over the last couple decades for scientific computing tasks, including the analysis and processing of large datasets. It is easy to learn, has efficient high-level data structures and a simple but effective approach to object-oriented programming.

Figure 2.2: Python and its library Scikit-Learn are appropriate platforms to make artificial neural networks

In the artificial intelligence community, Python is one of the most used language. Researchers from numerous fields of artificial intelligence develop in Python, with the assistance of the built-in libraries and other sources found on the internet. The usefulness of Python for data science comes primarily from the large and active third-party packages:

- **SciPy** for containing a wide array of numerical tools such as numerical integration and interpolation;
- **Scikit-Learn** for providing a uniform toolkit for applying common machine learning algorithms to data;
- **NumPy** for providing efficient storage and computating multi-dimensional data arrays;
- **Pandas** for providing a DataFrame object along with a powerful set of methods to manipulate, filter, group, and transform data;
- **Matplotlib** for providing a useful interface for creating publication-quality plots and figures;

and many more tools that aimed scientific computing and other machine learning fields. Besides the prebuilt libraries, choosing Python for artificial intelligence programming can be faster with the specific indenting style and the dynamic typing system. Python is platform independent, its interpreter and extensive standard library are available in source or binary form for free.

Chapter 3

Theoretical Background

3.1 Feature Engineering

Data mining is a machine learning tool that uses statistical methods to discover patterns in large data sets. Data mining has various stages to process knowledge and all of the stages run different methods to collect the appropriate data.

The first and most important part is to select the dataset which will be processed. This implies the following: Firstly the goal needs to be cleared up. Secondly the information that seems to be useful for the goal needs to be selected.

To decide the usefulness of a feature, it should pass the some quality requirements.

- Validity: the degree to which the measures conform to defined business rules or constraints
- Accuracy: the degree of conformity of a measure to a standard or a true value
- Completeness: the degree to which all required measures are known
- Consistency: the degree to which a set of measures are equivalent in across systems
- Uniformity: the degree to which a set data measures are specified using the same units of measure in all systems

There are some other criteria for the qualities of good features. A useful feature vector value appears more than one times in a data set, which enables the model to learn the relationships between the feature vector values and the associated labels. This means having many examples with the same value helps the model to see the feature in different settings to determine when it is a good predictor. Also, each feature should have a clear and obvious meaning.

3.1.1 Data Mining

Data preprocessing [24] is a data mining technique that involves transforming raw data into an understandable format. The selected dataset may include errors, missing values or noisy, inconsistent data which have to be filtered during the preprocessing stage.

Data preparation and filtering steps can be time consuming. Data preprocessing includes cleaning, integration and reduction. Sometimes data transformation takes place in the preprocessing too. The product of data preprocessing is the final training set.

The given data in large datasets often contains information that is not clear enough. There are several techniques for repairing these uncertain data.

For resolving missing values, the following steps can be taken:
1. Simply remove the feaure vector, which is only effective if the vector contains several missing attributes.
2. Fill the missing values manually, which can be done with the mean value.
Both techniques can create noise and inaccuracy, but it is important to deal the appropriate technique, because the incorrect feature vector may contain necessary information too.

Noise is a random error or variance in a measured variable. Noisy data needs to be smoothed to get the accurate prediction. Binning methods smooth the value with its neighbour's, so it is just a local smoothing technique. Clustering can be used to detect outlier values by organizing them into similar groups. Data can be smoothed by fitting a function like regression to the data.

Data inconsistencies can be solved manually by external references, or with the use of knowledge engineering. In the known of the meaning of every feature, it can be obvious to recognize an inconsistent value manually. Knowledge engineering uses technical, scientific and social aspects to deal with a feature value's inconsistency.

Data integration and reduction aims on the same goal, which is to make the dataset smaller. During integration, an algorithm is looking through the dataset and looking for redundacies and multiple data.

Data reduction is reducing the volume or the dimension of the dataset, without compromising the integrity of the original dataset. As the analysing can be time consuming, these data reduction techniques are really helpful in large datasets:
1. Dimension reduction: drop the irrelevant or weakly relevant values from the dataset
2. Data compression: use various encoding techniques to reduce the size of the dataset

Transformation and Feature Scaling

Some features of the dataset need to be transformed to get an appropriate form for data mining. Data transformation has a strong connection with **feature scaling** [32][10], which is a statistical method used to standardize the range of the data. Feature scaling uses mathematical procedures to scale the range.

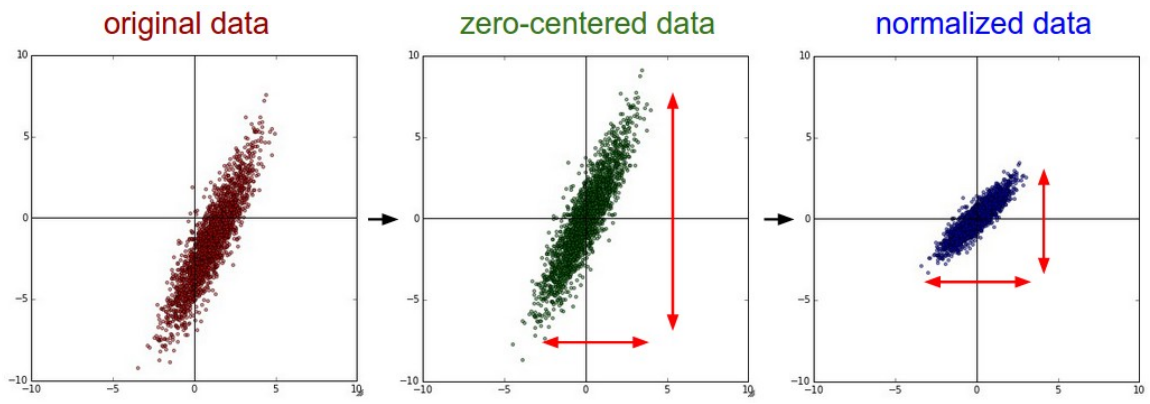


Figure 3.1: Data mining techniques can be used to convert data into understandable form.

Normalization is the process of scaling individual samples to have a unit norm. Min-max scaling is the simplest normalization method that rescales the features to a range of

$[0,1]$ or $[-1,1]$. The selected range depends of the data. It can be written as the following (3.1), where x is the original, and x' is the normalized value:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

Mean normalization (3.2) is similar to min-max scaling, where $\bar{x}$ is the mean of x:

$$x' = \frac{x - \bar{x}}{max(x) - min(x)} \tag{3.2}$$

Standardization is the process to transform the given data as if they comes from standard normally distributed data set. Feature standardization (3.3) makes the values of each feature to have zero-mean and unit-variance.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{3.3}$$

where x is the original feature vector, $\bar{x}$ is the mean of that feature vector and σ is its standard deviation.

Encoding is the process of converting data into an acceptable form for information processing. It can be used effectively on categorical features, because they just use a set of values.

Integer encoding converts the nomimal values to numeric values. These numeric values are integers in increasing order, and they can cause inconsistency with the given weights.

One-hot encoding offers a solution for the inconsistency problem. It creates a binary vector for each categorical feature in the dataset where the values appears as follows:

- For those values, which apply to the example, set the vector values to 1.
- Set other values to 0.

The length of the binary vector is equal to the number values in the current feature.

Data Mining Algorithms

As mentioned, the product of data mining is the training set. For the application of data mining algorithms, the original data set is usually splitted into multiple sets. There are two data sets, which are separated during the creation of the model. The first one is called **training set**, which is used by the machine learning algorithm to gather knowledge and increase accuracy. The other one is the **testing set**, which is used to provide a data set to test function estimation accuracy of the learning algorithm on the training data set.

After the split, the following data mining techniques [23] can be used on the training set to discover the patterns:

1. Classification: This method helps to retrieve information by classifying data into different classes. It can be use to draw further conclusions.
2. Clustering: It can be used to identify data which are similar to each other. It is an effective way to understand the similarities and differences between values.
3. Regression: This method analyzes the relationship between variables, to determine desired unknown values.
4. Association: It can discover hidden patterns in the dataset by identifying special events, for example high correlations between values.

5. Outer detection: It collects outlier values which do not match an expected pattern or expected behavior.
6. Sequential Patterns: This method helps to identify similar patterns for a certain period.
7. Prediction: This method can be used for predicting future values in a dataset with the known of the given data and past events.

The techniques of data mining originates from a wider field called **data analytics**. In a simple explanation, data analytics is the process of examining data sets in order to draw conclusions about the information they contain. It focuses on processing and performing statistical analyzis on existing data sets. A widely used data analytics technique is machine learning.

3.2 Single-Layer Perceptron

The perceptron is a single layer feedforward neural network [27], which has only one input and output layer. The input values are presented to the perceptron, and if the predicted output is the same as the desired output, then the performance is considered satisfactory and no changes to the weights are made. However, if the output does not match the desired output, then the weights need to be changed to reduce the error.

In order to avoid unnecessary iterations, it is important to adjust the weights (3.4) properly.

$$\Delta w = \eta * d * x \tag{3.4}$$

where Δw stands for the current weight, $\eta \leq 1$ is the learning rate which is the size of the required steps, d represents the desired output and x is the input data.

A single-layer perceptron's units are the artificial neurons, that are conventionally called **nodes**. To determine the value of a node, all the inputs would be multiplied by their respective weights, and then summed. This weighted sum stands for **dot product** in this context: $z = w_1x_1 + w_2x_2 + \cdots + w_mx_m = \sum_{j=1}^{m} w_jx_j$

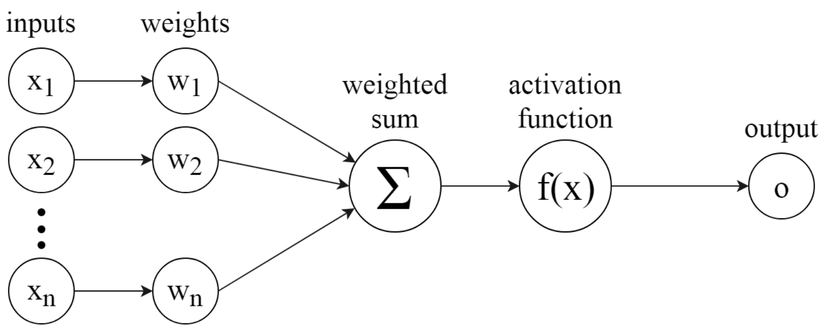


Figure 3.2: The appropriate weights are applied to the inputs and the resulting weighted sum passed to an activation function that produces the output.

The output o in (3.5) is determined by whether the weighted sum $\sum_j(w_jx_j)$ is less than or greater than some threshold value θ, which is the parameter of the neuron. If that value is above a given threshold, it "fires", which means that the neuron gets an activated value.

$$o = \begin{cases} 0, & \text{for } \sum_j(w_jx_j) < \theta \\ 1, & \text{for } \sum_j(w_jx_j) \geq \theta \end{cases} \tag{3.5}$$

3.3 Feedforward Neural Networks

The feedforward neural network [12] is a type of neural networks, which aims to create a mapping from a properly trained input dataset to an estimated output. This type of neural network is called feedforward as there are no feedback connections in which outputs of the neuron are connected to itself. A feedforward neural network is able to model complex non-linear functions.

A typical feedforward neural network consists of input and output layers. There is an intermediate part between inputs and outputs, called hidden layers. The **input layer** is a set of input neurons, where each neuron represents a feature in our data set. The **output** of any feedforward network is the sum of the inputs multiplied by the weights. The **hidden layer** contains units which can transform the inputs into a mapping that the output layer can use. The relationship between the input and hidden layer is determined by the weights of the network.

The output of a trained feedforward neural network can be characterized by (3.6)

$$o_k = f_k(i, w) \tag{3.6}$$

where o_k is the kth neural network output, (i, w) is a vector of the weights and $f_k(\cdot)$ describes the mapping from the input to the kth output, where $f_k(\cdot)$ also contains the structure of the feedforward perceptron. The neural network can be trained if the input and the output are fixed and the weights are set. When a single scalar output can be found, o_k can be replaced by o and $f_k(\cdot)$ by $f(\cdot)$.

3.3.1 Multi-Layer Perceptron

The multi-layer perceptron is a multi-layered feedforward neural network algorithm that learns a function $f(\cdot) : \mathbb{R}^m \mapsto \mathbb{R}^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, \ldots, x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression.

The difference between single- and multi-layered neural networks is that the multi-layered perceptron has one or more hidden layers besides the input and output layer. Except for the input nodes, each node is a neuron that uses a linear or non-linear activation function.

In a multi-layer perceptron, each neuron in one layer is connected with a weight to another neuron in the next layer. Each of these neurons stores a value, which is in general a sum of the weighted neurons which comes from previous layers. There is a special unit, called **bias**, that are not influenced by any values in the previous layer, so they do not have any incoming connections. However they have outgoing connections and they can contribute to the output of the artificial neural network. Bias units stores a constant value, which helps the model to fit the best for the given data. Hence the definition of dot product expands like this (3.7):

$$z = \sum_{j=1}^{m} w_j x_j + bias \tag{3.7}$$

Neural networks are designed to learn from datasets using iterative methods. Estimation value error is calculated from the estimated and the measured values to modify the weights of the connections between neurons. A multi-layer perceptron utilizes a supervised learning technique called **backpropagation** for training.

Backpropagation

The main goal of backpropagation [8] is to update all of the weights in the neural network, in such way that the predicted output to be closer to the target output with minimizing the error of each output neuron and also the network.

The algorithm consists of two phases: the forward phase where the activations are propagated from the input to the output layer, and the backward phase, where the error between the actual and the desired value in the output layer is propagated backwards to modify the weights values.

The function that is used to compute this error is known as **loss function**. For backpropagation, the loss function calculates the difference between the network output and its expected output, after a training example has propagated through the network.

Different loss functions will give different errors for the same prediction, and thus have a considerable effect on the performance of the model. In the training of multi-layer perceptrons, L2 loss function L has been used in (3.8), which is the square of the L2 norm of the difference between actual value y and predicted value $\hat{y}$.

$$L = \sum_{i=1}^{n} (y - \hat{y})^2 \tag{3.8}$$

Gradient descent

Backpropagation uses gradient descent [5] in the calculation of the weights. It is an optimization method, which aims on to minimize a given function to its local minimum by iteratively updating the weights of the model. The input is defined with an initial value and the algorithm calculates the gradient i.e. the partial derivative of the loss curve at this starting point.

The components of gradient descent are the following:

- Learning rate: size of steps took in any direction
- Gradients: the direction of the steps, i.e. the slope
- Cost function: tells the current height, which is the sum of squared errors

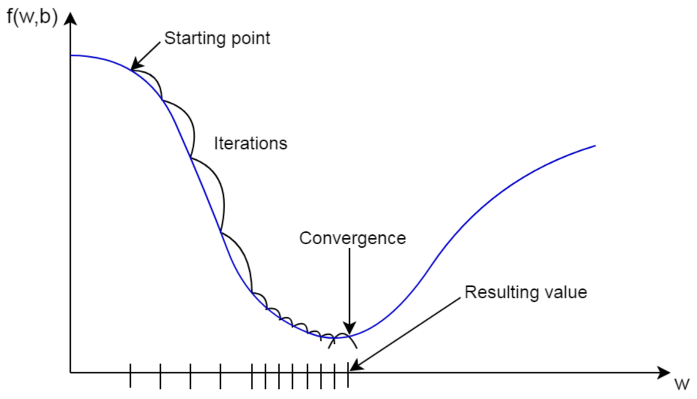


Figure 3.3: The gradient descent is an optimization method used by backpropagation

In backpropagation, the calculation of the gradient passes the network backwards, so the values of the gradient from one layer are reused in the computation of the gradient for the previous layer.

3.4 Training a MLP Model

The task that is being tackled in this paper is to make the inversion of a single-element feedforward neural network. To eventuate this successfully, the appropriate dataset is given and already preprocessed. Hence the main task is now to train a neural network to predict new outputs for the testing set and minimize the loss between the given and the desired outputs from the training set.

The components of a neural network model i.e the activation function, optimization algorithm and the size of the layers play a very important role in effectively training a model and produce accurate results. Different tasks require a different set of functions to give the most optimum results. The used methods and functions are described in the following subsections.

3.4.1 Regression

Regression [2] is a supervised learning task of machine learning that is used to predict values of a desired target variable. It is a statistical technique which requires a data set with the desired output that is consists of one or more continuous variables and real numbers.

By using regression the input vector is mapped onto a given set of values by the network. The network regresses the independent variables, provided by the inputs, onto the dependent variable. The multi-layer perceptron uses non-linear regression while trying to approximate the values of the dataset.

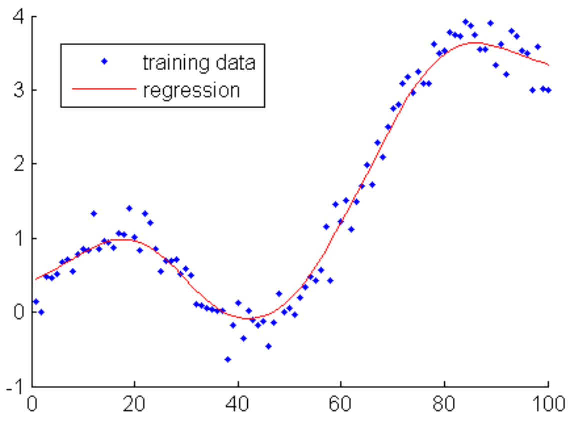


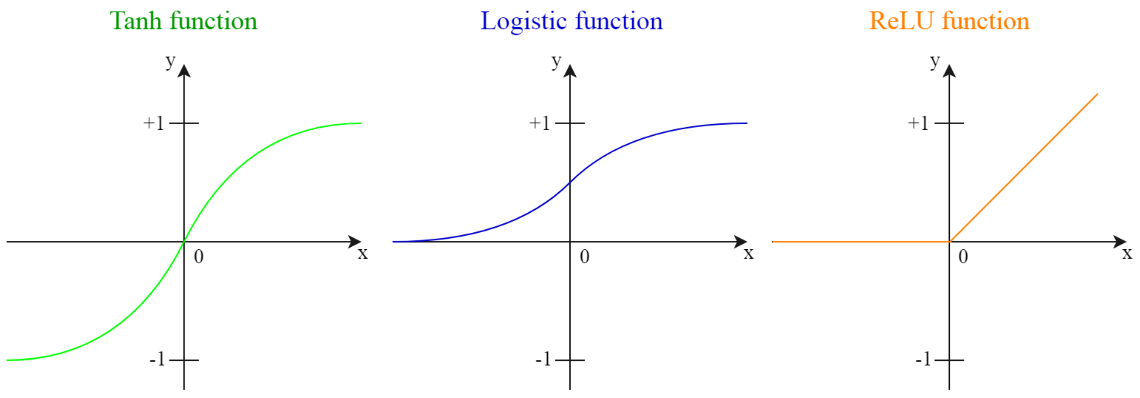
Figure 3.4: Regression is used to find the best fit of the training data

3.4.2 Activation Functions

In connection with artificial neural networks, the activation function is a transitional state of the neurons between other layers. It is a mapping of the previous layers and it maps the resulting value into the desired range, which is usually between -1 and 1. The output of the activation function is then used as input for the next layer, until a desired solution is found. There are several activation functions, and each of them utilizes different methods for mapping.

As the multi-layer perceptron approximates non-linear functions, it will be the most accurate by using non-linear activation functions as well [21]. These non-linear activation functions are known about having more than one degrees and they have a curve in their graph. As non-linear functions can generate non-linear mappings from inputs to outputs, they can be applied on complex data sets.

Figure 3.5: The graphs of the most popular non-linear activation functions



The two common non-linear activation functions are both sigmoids, and described by (3.9)

$$f(x) = tanh(x) \quad \text{and} \quad f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{3.9}$$

The first is a **hyperbolic tangent** that ranges from -1 to 1, while the other is the **logistic** function, which is similar in shape but ranges from 0 to 1.

The **ReLU** function is another type of non-linear functions, which stands for rectified linear unit. It is called half-rectified, because no negative values are allowed. This operation affects the resulting graph (Figure 3.5), as the negative values are shown as zeros (3.10).

$$f(x) = max(0, X) \tag{3.10}$$

3.4.3 Optimization Methods

Artificial neural networks have different phases in the process of their operation. The learning process in a neural network uses different training methods with different characteristics and performance. These training methods use optimization algorithms to update weights and biases i.e. the internal parameters of a model to reduce the error. [30][21][26]

Stochastic Gradient Descent

Stochastic gradient descent is a stochastic approximation of gradient descent optimization, used effectively in large-scaled data sets. It can also work in a system of linear and non-linear equations and can effectively solve unconstrained optimization problems. In contrast to gradient descent, the stochastic method can approximate the true gradient of the cost function because it updates the parameters for each training example, one by one. To demonstrate assume that η is the learning rate, L stands for the loss function and $\nabla_w L = \frac{\partial L}{\partial w}$ is the gradient. Now the updating equation of the weights w in each iteration is:

$$w_{t+1} = w_t - \eta \nabla_w L \tag{3.11}$$

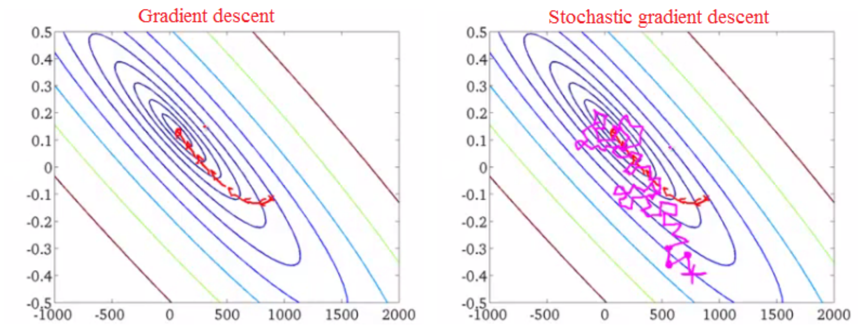


Figure 3.6: The difference between the process of GD and SGD methods

The process of the stochastic gradient descent algorithm is the following:
1. Choose one sample from the dataset (this is what makes it stochastic gradient descent).
2. Calculate all the partial derivatives of loss with respect to weights or biases.
3. Use the update equation (3.11) to update each weight and bias.
4. Go back to step 1.

Adaptive Moment Estimation

Another method is adaptive moment estimation, called Adam. It is a transition between adaptive methods and momentum-based methods. In the algorithm, running averages of both the gradients and the second moments of the gradients are used, which means Adam not only stores the exponentially decaying average of past squared gradients v_t, it also keeps the decaying average of past gradients m_t, similarly to momentum-based methods. The algorithm computes the decaying averages of past m_t and past squared v_t gradients respectively as follows in (3.12):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)L_t \quad \text{and} \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)L_t^2 \tag{3.12}$$

where m_t and v_t are the estimates of the first and second moments, β_1 and β_2 are the decay for gradients and second moments of gradients, and L_t is the loss function.
The first and second moment estimates are in (3.13):

$$\hat{m}_t = \frac{m_t}{a-\beta_1^t} \quad \text{and} \quad \hat{v}_t = \frac{v_t}{a-\beta_2^t} \tag{3.13}$$

Then these estimates are used to update the parameters w with a simple scalar ϵ to prevent division by 0 in (3.14):

$$w_{t+1} = w_t - \frac{\mu}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \tag{3.14}$$

Limited-memory BFGS

Limited-memory BFGS is an approximation of Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, which is an iterative method for solving unconstrained non-linear optimization problems. The difference between them, that L-BFGS uses a limited amount of computer memory. L-BFGS aims on parameter estimation, and the target problem is to minimize $f(x)$ over unconstrained values of the real-vector x where f is a differentiable scalar function. In this method, the Hessian matrix of second derivatives is not computed, but it is approximated by using gradient evaluation updates.

From an initial guess x_0 and an approximate Hessian matrix B_0, the following steps are repeated as x_k converges to the solution:
1. Obtain a direction p_k by solving $B_k p_k = -\nabla f(x_k)$.
2. Perform a one-dimensional optimization to find an acceptable stepsize α_k in p_k.
3. Set $s_k = \alpha_k p_k$ and update $x_{k+1} = x_k + s_k$.
4. Set $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.
5. The solution is $B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$.

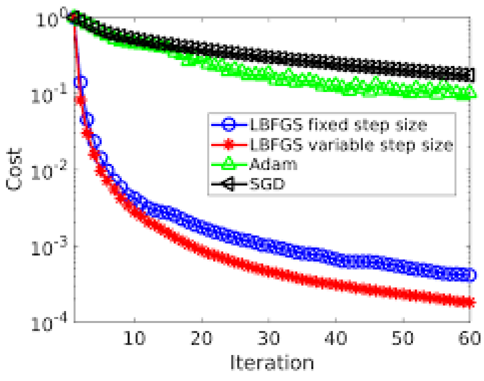


Figure 3.7: The optimization of SGD, Adam and L-BFGS with respect of cost and iteration

3.5 Inversion

Inversion of a neural network consists of clamping the weights and the neural network output while adjusting the input in the neural network until an equality or a best possible fit occurs for one or more values of the input.

Feedforward neural networks aim on to capture system mapping from the given training data. The goal is to find the input values that will result the desired output for the given synaptic weights. Generally it can be determined that a single input can generate numerous outputs.

The process is the following:
1. Fix the weights and the outputs.
2. Find such an input that concurs or mostly fits the supposed input.

To achieve the goal, it is necessary to:
- find every point that can fit the input
- define the external points as thresholds
Then the evenly distributed points can be located in the solution set.

3.5.1 Single-Element Inversion Methods

In the task of single-element inversion only one point is found by the algorithm depending on the initialization. This means it will result a nearer outcome in further processes. Hence it is very important to find a properly training algorithm, because it notes the previously founded points and reuses them. The process of finding the best estimator is time consuming. As the quantity of parameter combinations increases during the training phase, so does the time that the training takes. However, the number of parameter combinations also potentionally increase the inversion accuracy.

To solve the inversion of an unconstrained optimization problem, the inversion method needs to solve the optimization itself in its training phase. After the dataset is properly trained, the inversion problem is the following:

Given some network function $f : X \mapsto Y$, for some $y \in Y$, the appropriate $x \in X$ needs to be found, such that $f(x) = y$. Or more generally, if $L : Y \mapsto \mathbb{R}$ is the loss function defined over the network output, an input x have to be found that minimizes $L(f(x))$.

3.5.2 Williams-Linder-Kindermann Inversion

The WLK inversion was named after R. L. Williams, A. Linder and J. Kindermann [16], who firstly introduced the single-element search method for inversion of real valued neural network. In this algorithm, the inversion problem is set up as an unconstrained optimization problem and solved by gradient descent, similarly to backpropagation.

The method of WLK inversion involves two main steps:
1. computing the deltas for every value
2. updating the weights manually

During the training, the neural network is trained to learn a mapping from input to output. The proper set can be find by minimizing the loss. Thus the neural network learns a functional relationship between the inputs and the outputs. Now all the weights are fixed.

Assume that the initial input vector i_0 is given. Now the recursive equation of the training phase is the following:

$$i_k^{t+1} = i_k + t - \eta \frac{\partial E}{\partial i_k^t} \tag{3.15}$$

t - the index of the iteration,
i_k^t - the kth component of the i^t vector,
η - the learning rate

Because of the general feedforward topology, the iteration for inversion in (3.15) can be solved by the derivative of (3.16)

$$\frac{\partial E}{\partial i_k} = \delta k \quad k \in I \tag{3.16}$$

for every δk in (3.17):

$$\delta j = \begin{cases} \varphi_j'(o_j)(o_j - t_j) :, & j \in O \\ \varphi_j'(o_j) \sum_{m \in H,O} \delta_j w_{jm} :, & j \in I, H \end{cases} \tag{3.17}$$

I, O, H - the set of input, output and hidden neurons,
w_{jm} - the weight value from neuron j to neuron m,
φ_j' - the derivative of the jth neuron squashing function,
o_j - the activation of the jth neuron,
t_j - the desired output of the jth neuron

The derivatives of the neurons need to be solved by backward order from the output to the input.

Chapter 4

Implementation

4.1 Problem Statement

This case study aims on to make the inversion of a single-element feedforward neural network. However the implementation of the inversion is just the last step during the process, because the dataset needs a lot of preprocessing to make the predictable model.

This work belongs in the realm of machine learning application research. There are three main tasks examinated in this paper: data mining for extracting information from a data set, building and testing multilayer perceptron models for, and inverting these feedforward neural network models.

The first task is to choose a dataset which contains predictable data. The selected dataset is called the Online News Popularity Dataset by Mashable news, that was served by the UCI's Machine Learning Repository.

The dataset is preprocessed for data mining by Pandas. Since being a publicly available dataset, its preprocessing and transformation has already done, but the dataset still needs some cleaning with the assistance of Scikit-Learn. The dataset contains a lot of outlying values that should be handled. Furthermore the dataset has a wide range of values, which necessitates a standard scaling. Then the dataset can be split into training and testing sets by Scikit-Learn.

The training phase consists of the application of machine learning techniques [15]. Different multi-layer perceptron models are fitted on the training set with various parameters. These attributes have iterated over different their hidden layer sizes, activation functions, optimization algorithms, learning rate sizes and alpha values. After Scikit-Learn trains the dataset and makes predictions to the testing set. The training is time consuming, since the used dataset is large and the length depends on the amount of attributes of multi-layer perceptron models. At the end of each process, the testing set's output and the predicted output are compared and the difference between them is calculated. If all of the iterations are over, the best estimator's parameters are shown with the score, and the tested target values and the predicted values are plotted by Matplotlib.

Now the main task, inversion can be executed. To perform the WLK inversion, the equations which are mentioned in subsection 3.5.2 have to be implemented [19]. The inversion is defined as a python function to be callable in the main program. In the inversion function, the network is initialized with a random input vector. Output is calculated and compared with the given output. Now the error can be calculated and backpropagated to minimize the loss function and the input vector has updated too. This iterative process continues until the error is less than the minimum set value. The return value is the guessed input value.

At the end, the given testing set's values and the guessed input values are written to a .txt file with the accuracy percent of the inversion. A summarization about the accuracy percent values are described.

4.1.1 Used Third-Party Libraries

Python can be used effectively with the assistance of some of its third-party libraries [7][9], which provide numerous effective and easy-to-use models in scientific research.

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for Python. Pandas has a main object called DataFrame, that is for data manipulation using a set of labeled array data structures. Pandas has tools for reading tabular data and writing data between in-memory data structures and different file formats. The data can be handled integratedly and the data sets are transformable. Some of the examples of this transformation are column insertion and deleting, data set merging and joining, or the hierarchical axis indexing in high-dimensional data models.

Matplotlib is an open source Python library that produces publication-quality plots and figures. It ships with several add-on toolkits, including 3d plotting and assistance for axes. There are several common approaches of plotting in Matplotlib. The most popular is *pyplot*, which is a collection of command style functions where each *pyplot* function makes some change to a figure. Also it is mainly intended for interactive plots and simple cases of programmatic plot generation.

NumPy is a library for Python focused on multi-dimensional arrays and matrices. It provides high-level mathematical functions to operate on these arrays. In NumPy, the n-dimensional array is called *ndarray*, where all the elements of a single array must contain the same type.

The most important attributes of an *ndarray* object are:
- *ndarray.ndim* produces the number of axes of the array.
- *ndarray.shape* produces the dimensions of the array.
- *ndarray.size* is the total number of elements of the array.
- *ndarray.dtype* is an object describing the type of the elements in the array.
- *ndarray.itemsize* produces the size in bytes of each element of the array.
- *ndarray.data* is a buffer, which contains the actual elements of the array.

Scikit-Learn

The most important package that is used during the implementation is Scikit-Learn [20]. It is a separately-developed and distributed third-party extension to SciPy. It integrates classic machine learning algorithms into the scientific Python packages.

Scikit-Learn can be used to solve multi-layer neural network learning problems. Among many others, it features various classification, regression and clustering algorithms. Scikit-Learn contains all the various functions which can be used during the training of the neural network. The training lasts from the processing of the data set, via the iterations, to the inversion itself.

StandardScaler is a subclass of Scikit-Learn's prepropressing class that standardizes features by removing the mean and scaling to unit variance. It has three parameters that are boolean values and the default of all is True:

- *copy* means if the original dataset will be replaced with the scaled one or not
- *with_mean* means if the scaler centers the data before scaling or not
- *with_std* means if the data is scaled to unit variance or not

The standard score of a sample x is calculated as: $z = \frac{(x-u)}{s}$ where u is the mean of the training samples or zero if $with\_mean = False$, and s is the standard deviation of the training samples or one if $with\_std = False$.

StandardScaler has some methods:

- $fit(X[, y])$ computes the mean and the standard deviation to be used for later scaling
- $fit\_transform(X[, y])$ fits to data, then transforms it
- $get\_params([deep])$ gets parameters for this estimator
- $inverse\_transform(X[, copy])$ scales back the data to the original representation
- $partial\_fit(X[, y])$ is the online computation of mean and standard deviation on X for later scaling
- $set\_params(**params)$ sets the parameters of this estimator
- $transform(X[, y, copy])$ performs standardization by centering and scaling

Train_test_split is a subclass of Scikit Learn's selection class, which splits arrays or matrices into random train and test subsets. The randomization is important in ordered data sets. Allowed inputs are lists, NumPy arrays, SciPy-sparse matrices or Pandas DataFrames. Train_test_split returns with four lists that contains the train-test splits of inputs.

The parameters are the following:

- $test\_size$: If $float$, it should be between 0.0 and 1.0 and represents the proportion of the dataset to include in the test split. If int, it represents the absolute number of test samples. If $None$, the value is set to the complement of the train size.
- $train\_size$: If $float$, it should be between 0.0 and 1.0 and represents the proportion of the dataset to include in the train split. If int, it represents the absolute number of train samples. If $None$, the value is automatically set to the complement of the test size.
- $random\_state$: If int, it is the seed used by the random number generator. If $RandomState$ instance, it is the random number generator. If $None$, the random number generator is the $RandomState$ instance used by $np.random$.
- $shuffle$: Whether or not to shuffle the data before splitting.
- $stratify$: It shows if the data is split in a stratified fashion, using this as the class labels.

MLPRegressor [6] is a class of Scikit-Learn, which implements a multi-layer perceptron, that uses the square error as the loss function. It trains iteratively because the partial derivatives of the loss function are computed in each step to update the parameters of the layers. MLPRegressor also supports multi-output regression, in which a sample can have more than one target outputs.

MLPRegressor has several parameters, but only a few of them will be listed, which were used in the implementation:

- *hidden_layer_sizes* is a tuple where the ith element represents the number of neurons in the ith hidden layer.
- *activation* is the activation function for the hidden layer.
  - 'identity' provides the linear function, returns $f(x) = x$
  - 'logistic' provides the logistic sigmoid function, returns $f(x) = \frac{1}{1+e^{-x}}$.
  - 'relu' provides the rectified linear unit function, returns $f(x) = max(0, x)$.
  - 'tanh' provides the hyperbolic tan function, returns $f(x) = tanh(x)$.
- *solver* is the optimization algorithm used in weight optimization.
  - 'sgd' refers to stochastic gradient descent.
  - 'adam' refers to a transition between adaptive methods and momentum-based methods.
  - 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- *alpha* is the L2 regularization parameter's value.
- *learning_rate_init* is the initial learning rate that is used. It controls the step-size in updating the weights. Only used when the solver is *sgd* or *adam*.
- *learning_rate* means the learning rate schedule for weight updates.
  - 'constant' is a constant learning rate given by *learning_rate_init*.
  - 'invscaling' gradually decreases the learning rate at each time step using an inverse scaling exponent.
  - 'adaptive' keeps the learning rate constant as long as training loss keeps decreasing.

MLPRegressor contains built-in methods to perform regression:
- $fit(X, y)$ fits the model to data matrix X and target(s) y.
- *get_params*([*deep*]) gets parameters for this estimator.
- *predict*(X) predicts an output based on previous training and a given testing dataset.
- *score*(X, y[, *sample_weight*]) returns with the coefficient of the determinated prediction.
- *set_params*(***params*) sets the parameters of the estimator.

GridSearchCV [14] is an exhaustive searching class of Scikit-Learn over specified parameter values for an estimator. It can be used in the tuning of the hyper-parameters, which are those parameters that are not directly learnt within estimators. The parameters of the estimator used to apply the methods of GridSearchCV are optimized by cross-validated grid-search over a parameter grid.

The parameters of GridSearchCV are quite wide, but only a few of them are used during the implementation:

- *estimator* implements the estimator interface.
- *param_grid* is a dictionary with the names of the used parameters as keys and lists of parameter settings to try as values.
- *cv* is an integer that determines the cross-validation splitting strategy.
- *n_jobs* stands for the number of jobs to run in parallel. -1 means that the search uses all of the processors.

From GridSearchCV's attributes, *best_params_* is used, which is a dictionary about the parameter setting that gave the best results on the hold out data.

GridSearchCV contains built-in methods for prediction:
- $fit(X[, y, groups])$ method fits with the adjusted parameter grid on the given dataset.

- $get\_params([deep])$ gets parameters for this estimator.
- $predict(X)$, $predict\_log\_proba(X)$ and $predict\_proba(X)$ make the prediction on the test set
- $score(X[, y])$ returns the score on the given data, if the estimator has been refit.
- $set\_params(**params)$ sets the parameters of this estimator.

4.2 The Implementation

The first step was to import all of the necessary libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import inversion
```

The *inversion* library is not a built-in library of Python, it is importing the written function from another .py file.

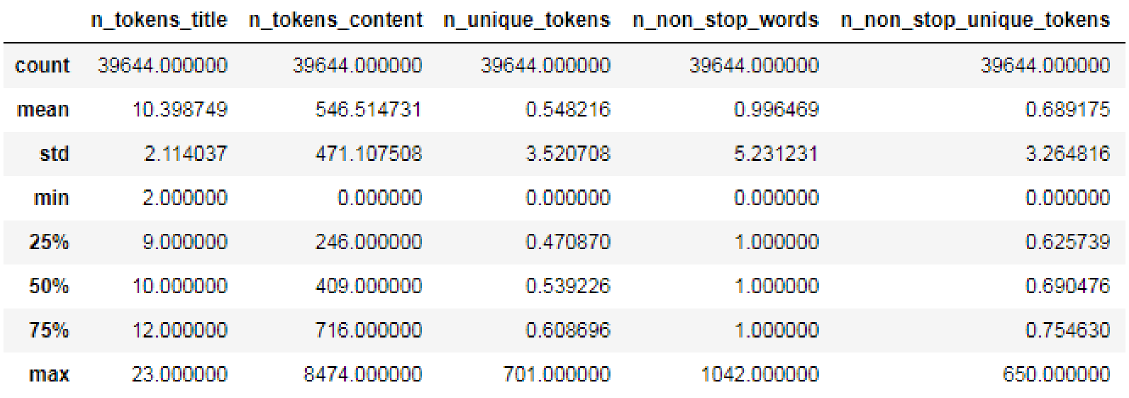
After, the selected dataset has captured by Pandas. In this case the used dataset is a .csv file, which contains 39.644 instances and 61 attributes. The attributes consist of 58 predictive features, 2 other attributes of accessory information and 1 goal field, which is the number of shares. The dataset represents as a matrix, where the columns are the features and the rows are the data. This dataset is preprocessed by Pandas as a DataFrame object. The *url* and *timedelta* columns have been omitted, since they are meta-data and cannot be treated as features.

```
dataset = pd.read_csv('OnlineNewsPopularity.csv')
dataset_copy = dataset.drop(columns=['url','timedelta'])
```

As already mentioned, the dataset's preprocessing and transformation has done, but the dataset still needs cleaning. Before execute the cleaning phase, the summarization of the dataset needs a review to see the data that need to be cleaned.

```
dataset_copy.describe()
```

where the first 5 columns results the following from the initial dataset:



	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens
count	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000
mean	10.398749	546.514731	0.548216	0.996469	0.689175
std	2.114037	471.107508	3.520708	5.231231	3.264816
min	2.000000	0.000000	0.000000	0.000000	0.000000
25%	9.000000	246.000000	0.470870	1.000000	0.625739
50%	10.000000	409.000000	0.539226	1.000000	0.690476
75%	12.000000	716.000000	0.608696	1.000000	0.754630
max	23.000000	8474.000000	701.000000	1042.000000	650.000000

As it can be seen in the table, there are outlying values that would cause noise if they are not handled. Also, in the known of the meaning of each columns, some inconsistencies are occured, like the *n_tokens_content* feature contains the number of words in the content, which cannot be zero. These inconsistent values needs a correction.

4.2.1 Optimizing the Dataset

There are a lot of techniques in data mining for optimizing the data into appropriate forms. In the case of outlying and inconsistent values, due to the huge number of data, the chosen technique was to simply ignore those tuples.

```
dataset_copy = dataset_copy[dataset_copy.n_tokens_content != 0]
dataset_copy = dataset_copy[dataset_copy.n_unique_tokens <= 1]
dataset_copy = dataset_copy[dataset_copy.average_token_length != 0]

dataset_copy = dataset_copy[dataset_copy.num_hrefs <= 100]
dataset_copy = dataset_copy[dataset_copy.num_self_hrefs <= 10]
dataset_copy = dataset_copy[dataset_copy.num_imgs <= 10]
dataset_copy = dataset_copy[dataset_copy.num_videos <= 2]
dataset_copy = dataset_copy.reset_index(drop=True)
```

These reductions need some explanation. As mentioned above, *n_tokens_content* is the number of words in the content, so it cannot be zero. *n_unique_tokens* contains the rate of unique words in the content. Because of *n_unique_tokens* is a rate, it need to be between [0,1]. The *average_token_length* contains the average length of the words in the content, which also cannot be zero.

The other optimizations are for handling the outlying values. *num_hrefs* contains the number of links, *num_self_hrefs* is the number of links to other articles published by Mashable, *num_imgs* has the number of images and *num_videos* stands for the number of videos. Furthermore because this dataset is a Pandas DataFrame and *drop* is a function for removing the whole row, the dataset needs to be reindexed.

After the cleaning, the dataset's other part is still has a big difference between its values, so the whole dataset should be scaled.

```
scaler = StandardScaler()
dataset_copy[:] = scaler.fit_transform(dataset_copy[:])
```

With the usage of StandardScaler's *fit_transform* method, the dataset has been fitted and transformed in one step.

Then the dataset has separated into feature X and target y groups.

```
y = dataset_copy.pop('shares')
X = dataset_copy
```

The column *shares* contains the target values, which are the number of shares.

Learning the parameters of a prediction function and testing the accuracy of the learning on the same data is a methodological mistake: a model that is just repeating the values of the samples, the model would have a perfect score, but it cannot predict anything useful from the data that are not seen yet. This situation is called overfitting. To avoid it, a common practice is when performing a supervised machine learning experiment to hold out a part of the available data as a testing set *X_test*, *y_test*.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=0)
```

Now the training (*X_train*, *X_test*) and testing (*y_train*, *y_test*) sets are made. The *test_size* is a float, which means the testing sets have this proportion from the dataset.

4.2.2 Training the Neural Network

Then the dataset is ready to be trained. The training consists of the application of machine learning techniques. Different multi-layer perceptron models are fitted on the training sets with a set of parameters *param_grid*. These parameters are the number of hidden layers, the activation functions, the optimization algorithms, the alpha value, the learning rate's type and the learning rate's initialize value. Scikit-Learn's training toolkit is called MLPRegressor, which is used as an estimator of GridSearchCV. The process of the training fits the neural network model to the training sets, then tests the accuracy on the testing sets by predicting the output *y_pred*.

hidden layereket beirni, ha lefutottak

```
mlp = MLPRegressor()
param_grid = {
   'hidden_layer_sizes': [(3,10,3)],
   'activation': ['relu', 'logistic', 'tanh'],
   'solver': ['lbfgs', 'adam'],
   'alpha': [0.03, 0.01, 0.003, 0.001],
   'learning_rate_init': [0.03, 0.01, 0.003, 0.001],
   'learning_rate': ['adaptive'],
}
gs = GridSearchCV(mlp, param_grid, cv=2, n_jobs=-1)
gs.fit(X_train, y_train)
```

This process is time consuming due to the amount of the given parameters are trained on a large-scale dataset. The search was running on a supercomputer and the training lasts for days. The score was computed by the L2 norm of the loss function, which means the difference between the original and the predicted value.

It can be seen, that stochastic gradient descent is not on the list of the optimization methods. When the dataset was trained, SGD causes NaN or infinite values during the calculation of the gradients. The solution was to remove SGD from the solvers.

After the trained multi-layer perceptron is ready, the results can be stored and the received values can be plotted by Matplotlib.

```
regressor = gs.best_estimator_
print(regressor, 'score: ', gs.best_score_, sep='\n ', file=open('
    TrainingResult.txt', 'w'))

y_pred = regressor.predict(X_test)

plt.plot(X_test, y_test, 'o', color='blue')
plt.plot(X_test, y_pred, 'o', color='orange')
plt.savefig('./' + 'prediction.pdf')
plt.show()
```

The results of the training contains a summarization of the best estimator and the score. It can be seen that the score was -1.51336177157304122 and the best estimator parameters were the ReLU activation function and L-BFGS as optimization algorithm, with the use of 0.001 alpha value and 0.03 initial learning rate. The training uses 3 hidden layers with $(3, 10, 3)$ neurons in the layers.

kicserelni, ha lefutottak az algoritmusok

```
MLPRegressor(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
    beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(3,
     10, 3), learning_rate='adaptive', learning_rate_init=0.03, max_iter=200,
    momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
    random_state=None, shuffle=True, solver='lbfgs', tol=0.0001,
   validation_fraction=0.1, verbose=False, warm_start=False)
score: −1.5133617715730412
```

The results of the training are shown on the following Figure 4.1, where the original values are colored blue and the predicted ones have color orange:

kicserelni a kepet, ha lefu-tottak

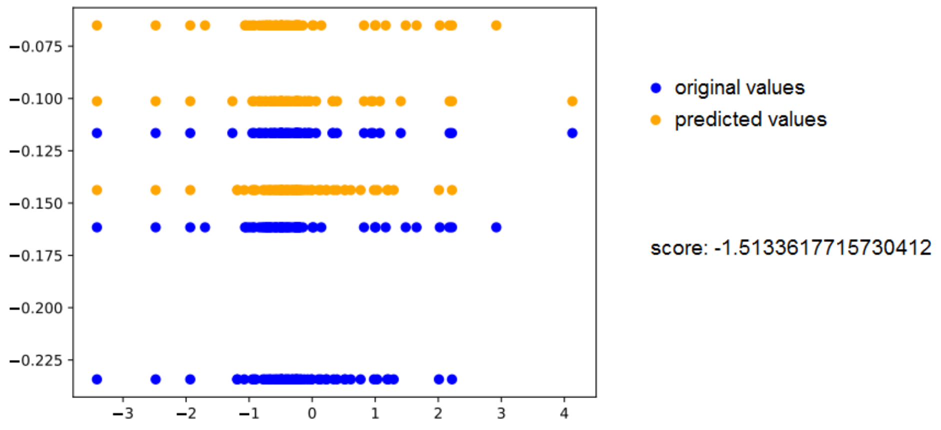


Figure 4.1: The result of the best prediction

It is a long process to train a multi-layer perceptron properly and find the best fitting regression function. During the training, thousands of parameter combinations were trained, but the best fitting parameters were not found. This is the reason, why the score value is low.

4.2.3 Inverting the MLP

The inversion was written as a python function, called *invert*(). The parameters of the invert function are the best estimator, the expected output, the size of the input vector, the step size, which is the learning rate, the number of iterations, and a boolean which stands for the function to be verbose or not.

```
def invert(regressor,
  expected_output,
  input_size,
  step_size,
  iteration_count = 100,
  verbose = False):

  [...]
```

The process of the inversion is the following: The network is initialized with a random input vector, then the sizes of the layer units are defined. The inversion is an iterative process where the equations in the WLK inversion is computed. The return value is the guessed value of the input.

```
def invert(regressor, expected_output, input_size, step_size, iteration_count
     = 100, verbose = False):

  guessedInput = np.random.rand(input_size)
  layer_units = [[input_size] + list(regressor.hidden_layer_sizes) +
     [regressor.n_outputs_]]

  for j in range(0, iteration_count):
    activations = [guessedInput]

    for i in range(regressor.n_layers_ - 1):
      activations.append(np.empty((guessedInput.shape[0],
       layer_units[0][i + 1])))

    regressor._forward_pass(activations)
    y_pred = activations[-1]
    deltas = activations.copy()
    deltas[-1] = _activationFunctionDerivate(activations[-1],
     regressor.activation) * (y_pred - expected_output)

    for i in range(1, len(activations)):
      deltas[-i - 1] = _activationFunctionDerivate(activations[-i - 1],
       regressor.activation) * \ (regressor.coefs_[-i] *
       deltas[-i].T).sum(axis=1)

      if verbose:
        print('#', i)
        print(regressor.coefs_[-i])
        print(deltas[-i])
        print(regressor.coefs_[-i] * deltas[-i].T)
        print((regressor.coefs_[-i] * deltas[-i].T).sum(axis=1))
        print(activations[-i-1])
        print(_activationFunctionDerivate(activations[-i-1],
         regressor.activation))
        print(deltas[-i-1])
        print('——————————————————')

    guessedInput = guessedInput - step_size * deltas[0]

  return guessedInput
```

As it is known, the equation in (3.17) uses the derivatives of the activation functions. A function *_activationFunctionDerivate*() was defined, which computes the derivatives of the used activation.

```
def _activationFunctionDerivate(X, activation):
  if activation == 'tanh':
    return 1.0 / (np.cosh(X)**2)
  if activation == 'logistic':
    log_sigmoid = 1.0 / (1.0 + np.exp(-1 * X))
    return log_sigmoid * (1.0 - log_sigmoid)
  if activation == 'relu':
    return [1.0 if np.any(X > 0.0) else 0.0]
```

The defined inverse function can be called to accomplish the inversion. The inversion is implemented on the testing output set y_test to compute those values from the testing input set X_test, which results the values in the testing output set. Every iteration results a tuple with the values of the guessed output.
The accuracy percent is computed by getting the division of the difference between the guessed value and the desired value, and the range of the output vector y.

```
inversionResults = pd.DataFrame(columns=['accuracy_percent'])

for i, value in enumerate(y_test):
    desired_output = [value]
    guessedInput = inversion.invert(regressor, desired_output, pd.DataFrame(
      X_test).columns.size,
    gs.best_params_['learning_rate_init'])
    guessedInput = pd.DataFrame(guessedInput).T

    accuracy = abs((regressor.predict(guessedInput) - desired_output) /
     (y.max() - y.min()))
    inversionResults = inversionResults.append({'accuracy_percent': accuracy
      [0]}, ignore_index=True)

    print('guessed input vector in X_test: ', np.array(guessedInput),
    'predicted input vector for X_test: ', regressor.predict(guessedInput),
    'desired output value in y_test: ', desired_output,
    'error: ', regressor.predict(guessedInput) - desired_output,
    'accuracy percent: ', accuracy, '\n',
    sep='\n ', file=open('InversionResults.txt', 'a'))

print('summarization: ', inversionResults.describe(), sep='\n ', file=open('
    InversionResults.txt', 'a'))
```

The guessed and predicted input vectors, the desired output values, the error and the accuracy percent are stored and written into a .txt file. The summarization of the accuracy is described.

4.3 Results

Inversion have not received much attention since the rise of deep learning. Neural network inversion procedures seek to find those input values, which produces the given output values. WLK inversion is a type of single-element search methods, which can solve the inversion problem.

The results of the inversion are stored and the summarization of the accuracy percents is the following:

kicserelni majd, ha lefutottak az algoritmusok

```
           accuracy_percent
count          3.000000
mean           0.660939
std            0.177815
min            0.499214
25%            0.565731
50%            0.632248
75%            0.741801
max            0.851354
```

It can be seen that the WLK algorithm succeeds to make the inversion.

Inversion is firmly depends on the result of the training. The more accurate the training is, the more effective the inversion will be. Finding the best fitting parameters to the dataset, especially to large-scale datasets takes great amount of time. The training was running in parallel on the supercomputer with thousands of parameter combinations, but the best fitting combinations are not captured yet. Anyway WLK inversion can produce the features correctly.

Chapter 5

Summary

megirni

Bibliography

[1] Craig A. Jensen, Russell D. Reed, Robert J Marks, Mohamed El-Sharkawi, Jae-byung Jung, Robert T. Miyamoto, Gregory M. Anderson, and J Eggen. Inversion of feedforward neural networks: Algorithms and applications. 02 2001.

[2] M.P. Allen. *Understanding Regression Analysis.* Springer US, 2007.

[3] E. Alpaydin. *Introduction to Machine Learning.* Adaptive Computation and Machine Learning series. MIT Press, 2009.

[4] G.A. Anastassiou. *Intelligent Systems: Approximation by Artificial Neural Networks.* Intelligent Systems Reference Library. Springer Berlin Heidelberg, 2011.

[5] J.A. Anderson. *An Introduction to Neural Networks.* Bradford book. MIT Press, 1995.

[6] B. Bengfort, R. Bilbro, and T. Ojeda. *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning.* O'Reilly Media, 2018.

[7] E. Bressert. *SciPy and NumPy.* Oreilly and Associate Series. O'Reilly, 2012.

[8] Y. Chauvin and D.E. Rumelhart. *Backpropagation: Theory, Architectures, and Applications.* Developments in Connectionist Theory Series. Taylor & Francis, 2013.

[9] D.Y. Chen. *Pandas for Everyone: Python Data Analysis.* Addison-Wesley Data & Analytics Series. Pearson Education, 2017.

[10] G. Dong and H. Liu. *Feature Engineering for Machine Learning and Data Analytics.* Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press, 2018.

[11] Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez. A proactive intelligent decision support system for predicting the popularity of online news. In *EPIA*, 2015.

[12] T.L. Fine. *Feedforward Neural Network Methodology.* Information Science and Statistics. Springer New York, 2006.

[13] J. Han, J. Pei, and M. Kamber. *Data Mining: Concepts and Techniques.* The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011.

[14] K. Jolly. *Machine Learning with scikit-learn Quick Start Guide: Classification, regression, and clustering techniques in Python.* Packt Publishing, 2018.

[15] N. Karayiannis and A.N. Venetsanopoulos. *Artificial Neural Networks: Learning Algorithms, Performance Evaluation, and Applications.* The Springer International Series in Engineering and Computer Science. Springer US, 2013.

[16] J Kindermann and A Linden. Inversion of neural networks by gradient descent. *Parallel Computing*, 14(3):277 – 286, 1990.

[17] R.S. Michalski, J.G. Carbonell, and T.M. Mitchell. *Machine Learning: An Artificial Intelligence Approach*. Number 1. k. Elsevier Science, 2014.

[18] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[19] J. Nazari and O.K. Ersoy. *Implementation of Back-propagation Neural Networks with Matlab*. Technical report (Purdue University. School of Electrical Engineering). Purdue University, School of Electrical Engineering, 1992.

[20] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jacob VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] G. Pillo and F. Giannessi. *Nonlinear Optimization and Applications*. SpringerLink : Bücher. Springer US, 2013.

[22] K.L. Priddy and P.E. Keller. *Artificial Neural Networks: An Introduction*. SPIE tutorial texts. Society of Photo Optical, 2005.

[23] A.K. Pujari. *Data Mining Techniques*. Universities Press, 2001.

[24] D. Pyle. *Data Preparation for Data Mining*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 1999.

[25] He Ren and Quan Yang. Predicting and evaluating the popularity of online news. 2015.

[26] S. Sathasivam. *Optimization Methods in Training Neural Networks*. Universiti Sains Malaysia, 2003.

[27] D.X. Tho. *Perceptron Problem in Neural Network*. GRIN Verlag, 2010.

[28] G. Van Rossum and F.L. Drake. *An Introduction to Python*. Network Theory Limited, 2011.

[29] J. VanderPlas. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, 2016.

[30] T. Veerarajan. *Numerical Methods*. Sigma series. McGraw-Hill Education (India) Pvt Limited, 2007.

[31] M.J. Zaki, J.X. Yu, B. Ravindran, and V. Pudi. *Advances in Knowledge Discovery and Data Mining, Part I: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabat, India, June 21-24, 2010, Proceedings*. Advances in knowledge discovery and data mining : 14th Pacific-Asia conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010 : proceedings. Springer, 2010.

[32] A. Zheng and A. Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, 2018.