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.

Szerző 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 3, 2019

Lilla Juhász

Contents

Chapter 1

Introduction

This case study aims on to make the inversion of a single-element feedforward neural network. The implementation of the inversion is just the last step during the process, due to it needs a lot of preprocessing to make the appropriate model.

There are three main tasks are being tackled in this paper: data mining for extracting information from the data set, neural networks for training the data set, and the inversion itself. The most important thing which interweaves these tasks is machine learning.

Chapter 2

Related Works

2.1 Data Mining

Data mining 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 involves effective data collection called data warehouses, and it uses mathematical algorithms for segmenting the data and evaluating the probability of future events.

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 datas 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 for the dataset is not as easy as it seems. A well-used dataset is large-scaled and carries a lot of useful 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 [6] - 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 [12] 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 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 works as a scientific tool of algorithms and statistical models that utilizes statistical methods to process predefined data sets and to predict future output values for given data inputs.

Application fields of machine learning is quite big. It 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.

For this process, the original data set is usually splitted into multiple sets. There are two data sets, which is 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.

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. Its algorithm analyzes the given data set and processes the labeled data for mapping new examples. Classification and regression are the two types of supervised learning. 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. Neural

networks are parallel machines which can model mathematical functions between inputs and outputs. They have the capability to learn which allows them to gain knowledge through relationships in the training data.

The artificial neural network is a machine learning tool 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.

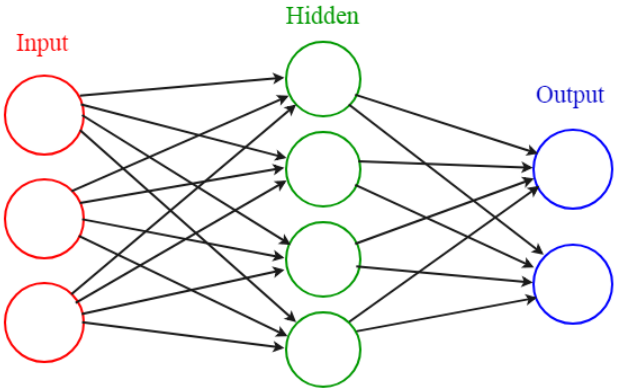

Figure 2.1: A neural network, where the circles represent the artificial neurons as nodes

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 potentionally increases the estimation accuracy.

2.2.2 Inversion

There is a subfield of machine learning that is hardly researched yet, called the inversion problem [9].

A properly trained neural network can model the process that is generating the training data and even the inverse process. 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]. A neural

network is trained to generate SIR pixel values as a function of sonar and environmental parameters. Once trained, the inverted neural network can provide input parameters to generate desired SIR performance in a specified target region.

2.3 Python

Python 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 first-class tool 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.

In the artificial intelligence community, Python is one of the most used language. Numerous artificial intelligence fields' researchers develop in Python, with the assistance of the built-in libraries and several 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. Also the vast majority of the libraries used for data science have Python interfaces.
Besides the prebuilt libraries, choosing Python for artificial intelligence programming can make the development easier and faster with the specific indenting style and the dynamic typing system, which means less coding and more developing. 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 Data Mining

Data mining is a subfield of computer science and statistics that uses machine learning and 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 datas.

The first and most important part is to select the dataset which will be processed. This means two things. Firstly the goal needs to be drawed up. Secondly the information that seems to be useful for the goal needs to be selected.

To decide the usefulness of a feature, it needs to pass a set of quality criteria.

- 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

After the features has selected, they are just raw data and need a revision to reduce the amount of further data cleaning. There are some other criteria for the qualities of good features.

Good feature values should appear more times in a data set. It enables a model to learn how this feature value relates to the label. 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 to anyone.

3.1.1 Preprocessing

Data preprocessing 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 datas. During the preprocessing stage, these datas need to be filtered out.

Data preparation and filtering steps can take considerable amount of processing time. 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.

Data cleaning

The given data in large datasets often contains information that is not clear enough. That can happen from a number of reasons. For example the data can be incomplete as lacking attribute values or certain attributes of interest. The data can be inconsistent or noisy too if it contains errors or too much outlier values that deviates from the expected. There are several techniques for repairing these useless datas.

For missing values, there are two things can be done:
1. Simply ignore the tuple, which is only effective if the tuple contains several missing attributes
2. Fill the missing values manually, that can be done with the mean value or a global constant (like ∞)
Both techniques can cause noise and inaccuracy, but the current tuple may contain important 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 smoothes 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 the data to a function such as regression.

Data inconsistencies can be solved manually by external references, or with the use of knowledge engineering.

Integration and Reduction

Data integration and reduction aims on the same goal, to have a smaller dataset. 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 Future Scaling

Some features of the dataset need some transformation to get an appropriate form for mining. Data transformation involves a bunch of statistical techniques, called future scaling.

Future scaling is a method used to standardize the range of the data. The raw data have a wide range of values that can make noisy data, and in some machine learning algorithms, objective functions will not work properly without normalization. To scale the range of the datas, mathematical procedures can be used.

Normalization is the process of scaling individual samples to have unit norm. Min-max scaling or min-max normalization is the simplest method that rescales the range of features to $[0, 1]$ or $[-1, 1]$. Selecting the target range depends on the nature of the data. The general formula is given as:

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

where x is the original value and x' is the normalized value.
Mean normalization is similar to min-max scaling, where $\bar{x} = average(x)$:

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

Standardization of datasets is a common method to transform the given data as if they comes from standard normally distributed data set. Feature standardization makes the values of each feature in the data have zero-mean and unit-variance.

In practice the shape of the distribution is often ignored and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

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

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 a form that is required for a number of information processing needs. It can be used effectively on categorical features since they have a discrete set of possible values.

Integer encoding converts the nomimal values to numeric values, but it will impose some constraints. These numeric values are integers in increasing order, so they can be problematic during the training process. They can cause inconsistency with the given weights.

One-hot encoding offers a solution for this problem. It creates a binary vector for each categorical feature in the model that represents values as follows:
• For values that apply to the example, set corresponding vector elements to 1.
• Set all other elements to 0.
The length of this vector is equal to the number of elements in that current feature set.

3.1.2 Data Mining Algorithms

After the appropriate training set is given, the following data mining techniques can be used to discover the patterns:
1. Classification: this method helps to classify data in different classes
2. Clustering: identifies data that are like each other to understand the differences and similarities between the data
3. Regression: this is the method of identifying and analyzing the relationship between variables
4. Association: it discovers a hidden pattern in the data set
5. Outer detection: collects the data which do not match an expected pattern or expected behavior
6. Sequential Patterns: this method helps to discover or identify similar patterns or trends in transaction data for certain period
7. Prediction: it analyzes past events or instances in a right sequence for predicting a future event

Data mining's applying techniques come 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 analysis on existing data sets. A widely used data analytics technique is machine learning, because it can process data sets more quickly than it can be done via conventional analytical modeling. Machine learning automates model building, due to it relies on patterns and inferences, instead of explicit instructions.

3.2 Feedforward Neural Networks

There are many different types of neural networks, one of them is the feedforward neural network [7]. It aims on to create a mapping from a properly trained input dataset to an estimated output and it is able to model complex non-linear functions.

A feedforward neural networks units are the artificial neurons, that are conventionally called **nodes**. To determine the value of a node, all the inputs would be multiplied one by one with their weights, and then adding their sums. This type of neural network is called feedforward due to there are no feedback connections in which outputs of the model are fed back into itself.

A typical feedforward neural network consists of inputs, weights and an output. 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. There is an intermediate part between inputs and outputs, called hidden layers. The **hidden layer** contains those type of units which can transform the inputs into a mapping which 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

$$o_k = f_k(i, w)$$

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 to get the expected results. When a single scalar output can be found, o_k can be replaced by o and $f_k(\cdot)$ by $f(\cdot)$.

3.2.1 Single-Layer Perceptron

The perceptron is a single layer feedforward neural network [11]. A single-layer perceptron has only one input and output layer and no hidden layers. 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 and then the weights need to be changed to reduce the error.

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

$$\Delta w = \eta * d * x$$

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.

The sum of the inputs multiplied by the appropriate weights are led directly to the output layer. This weighted sum stands for **dot product** in this context: $z = w_1x_1 + w_2x_2 + \cdots + w_mx_m = \sum_{j=1}^{m} w_jx_j$

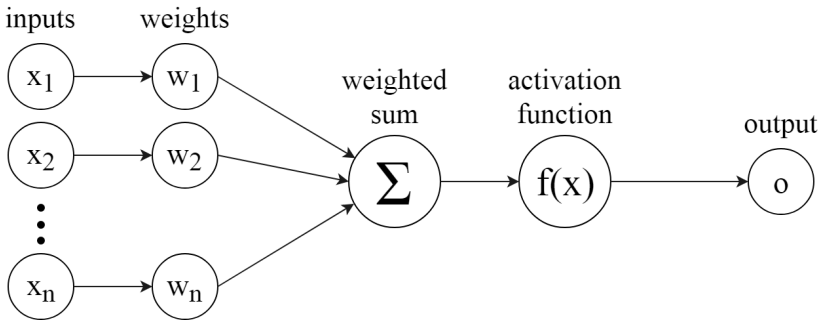

Figure 3.1: The appropriate weights are applied to the inputs and the resulting weighted sum passed to an activation function that produces the output.

The output o is determined by whether the weighted sum $\sum_j(w_j x_j)$ 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.

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

3.2.2 Multi-Layer Perceptron

The multi-layer perceptron is a supervised learning 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 that the multi-layered one 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 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 an amount, which is in general a sum of the weighted neurons come from previous layers. There is a special unit, called **bias** units, 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. Now the definition of dot product expands like this:

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

A neural network was made to learn from changing the connected weights after every piece of data is processed, compared to the expected result, based on the amount of error. A multi-layer perceptron utilizes a supervised learning technique called **backpropagation** for training.

Backpropagation

The main goal of backpropagation is to update all of the weights in the neural network, so they cause 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 nominal value in the output layer is propagated backwards in order to modify the weights values.

The function that is used to compute this error is known as **loss function**. The loss function is a function that maps values of one or more variables onto a real number intuitively representing some "cost" associated with those values. 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, which is the square of the L2 norm of the difference between actual value y and predicted value $\hat{y}$.

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

Gradient descent

Backpropagation uses gradient descent in the calculation of the weights used in the neural network. 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

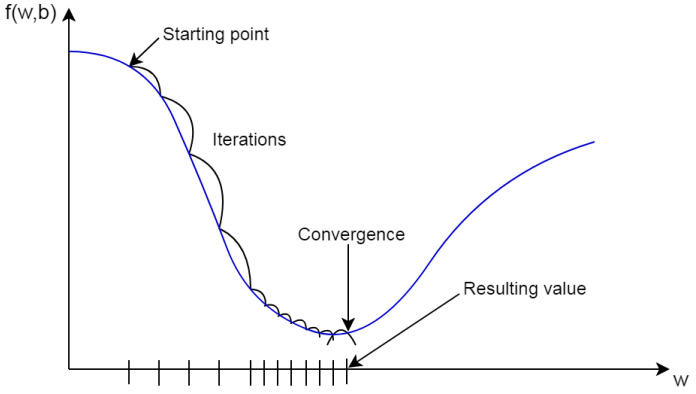

Figure 3.2: The gradient descent is an optimization method used by backpropagation

In backpropagation, the calculation of the gradient proceeds backwards through the network, so the partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backward phase is a more efficient way of computing the gradient at each layer, instead of calculating the gradient of each layer separately.

3.3 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 result 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.

3.3.1 Regression

Regression 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.

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.

In a linear regression task there is one independent variable x, to explain or predict the outcome of the dependent variable y.

$$y = \beta_0 + \beta_1 x + \epsilon$$

where β_0 stands for the intercept, β_1 is the steep of the independent variable and ϵ represents the error value that is the residual of the regression.

In non-linear regression, a statistical model of the form is

$$y \approx f(X, \beta)$$

that relates a vector of independent variables X, and its associated observed dependent variables y. The function f is non-linear in the components of the vector of parameters β, but otherwise arbitrary.

3.3.2 Activation Functions

In 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 algorithms for mapping.

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

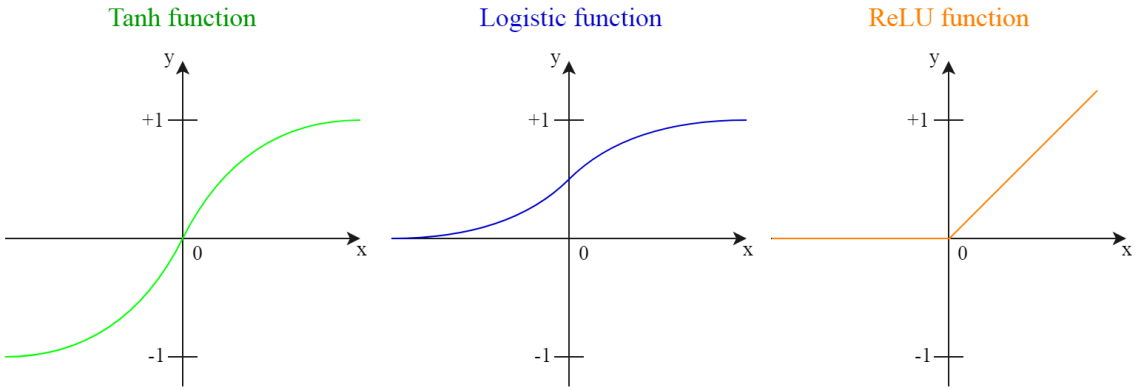

Figure 3.3: The graphs of the most popular non-linear activation functions

The two common activation functions are both sigmoids, and are described by

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

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. Its is called half-rectified, due to if the data set consists any negative values, that will turn into 0 immediately. This operation affects the resulting graph (Figure 3.3), due to the appropriate values will not be mapped.

$$f(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$$

3.3.3 Optimization Methods

Artificial neural networks have different phases in the process of their operation. The procedure of the learning process in a neural network can apply 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. [13][10]

Stochastic Gradient Descent

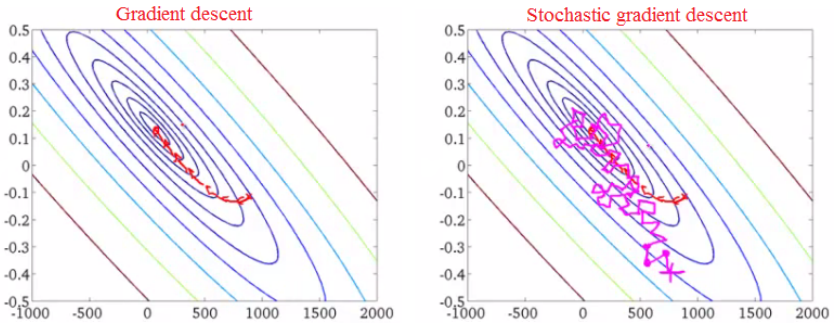

Figure 3.4: The difference between the process of GD and SGD methods

Stochastic gradient descent [3] 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 constrant 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 update equation of the weights w in each iteration is:

$$w_{t+1} = w_t - \eta \nabla_w L \tag{3.1}$$

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 (3.1) to update each weight and bias.
4. Go back to step 1.

Adam

Another method is Adam, whose name is derived from adaptive moment estimation. 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 just 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:

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

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:

$$\hat{m_t} = \frac{m_t}{a - \beta_1^t} \quad \text{and} \quad \hat{v_t} = \frac{v_t}{a - \beta_2^t}$$

Then these estimates are used to update the parameters w with a simple scalar ϵ to prevent division by 0

$$w_{t+1} = w_t - \frac{\mu}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t}$$

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. It 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 $B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$.

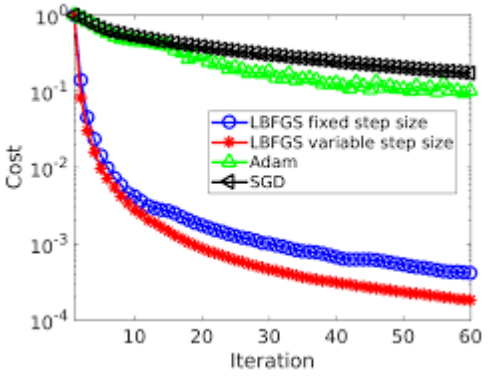

Figure 3.5: The optimization of SGD, Adam and L-BFGS with respect of cost and iteration

3.4 Inversion

As mentioned in subsection 2.2.2, a properly trained neural network can model the process that is generating the training data and the inverse process too. 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 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:
1. find every point that can fit the input
2. define the external points as thresholds
3. the evenly distributed points can be located in the solution set.

3.4.1 Single-Element Inversion

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 reuse them. This process is very 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 potenionally increases the inversion accuracy.

3.4.2 Inversion Methods

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 $L(f(x))$.

Some applications have been proposed that rely on single-element inversion methods.

WLK Inversion

The WLK inversion was named after R. L. Williams, A. Linder and J. Kindermann, 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. **training** the network
2. **inversion**

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. After the training, 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 back-propagated to minimize the loss function and the input vector is updated. This iterative process continues until the error is less than the minimum set value.

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

$$i_k^{t+1} = i_k + t - \eta \frac{\partial E}{\partial i_k^t} \tag{3.2}$$

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

Due to the general feedforward topology, the iteration for inversion in (3.2) can be solved by the dervative of

$$\frac{\partial E}{\partial i_k} = \delta k \qquad k \in I$$

for every δk:

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

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. Thus everything is given in the equation, but the absence of feedback, so the relationship of the neurons is only an assumption.

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, due to it needs a lot of preprocessing to make the appropriate 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 usable data. The selected one is called the Online News Popularity Dataset by Mashable news, that was served by the UCI's Machine Learning Repository.

The dataset is preprocessed by Pandas to be ready for data mining. Since being a publicly available dataset, its preprocessing and transformation has already made, but the dataset even needs some cleaning with the assistance of Scikit-Learn. The dataset contains a lot of outlying values that should be handled, and the dataset necessiates a standard scaling. Furthermore as the dataset is large-scaled, the less important features are simply removed. After this processing phase the dataset can be splitted into training and testing sets by Scikit-Learn.

The training phase consists of the application of machine learning techniques. Different multi-layer perceptron models are fitted on the training set with various parameters. These attributes changed in their hidden layer sizes, activation functions, optimization algorithms, learning rate sizes and alpha values. With the utilization of the inputs, Scikit-Learn trains the training dataset and make predictions to the testing set. The training is very time consuming, since the used dataset is very large, and also the length is depending on the amount of attributes of multi-layer perceptron models. In 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 ended, the best estimator's parameters are shown with its score, and the tested target values and the predicted ones are plotted by Matplotlib.

From the results of the training, the single-element inversion can be performed by Scikit-Learn. It is an iterative process, where the target values are the new predictors with the knowledge comes from the other features. Every iteration calculates one single feature's values and also its accuracy between the feature's original and the calculated values. The results are also plotted by Matplotlib.

These results show the influence of the features to the target. The larger the score, the more important the feature is.

4.1.1 Used Third-Party Libraries

Python can be used effectively with the assistance of some of its third-party libraries [8] [4], which provide numerous effective and easy-to-use models in scientific research.

Pandas [5] 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.

Scikit-Learn

The most important package that is used during the implementation is Scikit-Learn. 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 boolen 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 std to be used for later scaling
- $fit\_transform(X[, y])$ fits to data, then transform 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 std 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 [2] 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 solver for the 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 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

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

After, the selected dataset has captured by Pandas. In this case the used dataset is a .csv file, which contains 39644 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 datas. This dataset is preprocessed by Pandas as a DataFrame object

dataset. 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 made, but the dataset even needs some cleaning. Before doing that, let's give a check on the summarization:

```
dataset_copy.describe()
```

where the first 5 columns results the following:

	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

Figure 4.1: The summarization of the initial dataset

As it can be seen in the table, there are outlier values that would cause noise if they will not be 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 can not be zero. These inconsistent values needs a correction.

4.2.1 Optimising the Dataset

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

```
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 can not be zero. *n_unique_tokens* contains the rate of unique words in the content. Due to it is a rate, it need to be between [0,1]. *average_token_length* contains the average length of the words in the content, which also can not 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* is 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 it 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 splitted into feature X and target y groups.

```
y = dataset_copy.get(key='shares').values
X = dataset_copy.drop(columns='shares').values
```

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

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a supervised machine learning experiment to hold out part of the available data as a testint 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

Now 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*.

```
mlp = MLPRegressor()
param_grid = {
  'hidden_layer_sizes': [(30, 90, 180, 90, 30)],
  'activation': ['logistic', 'tanh', 'relu'],
  'solver': ['lbfgs','sgd', 'adam'],
  'alpha': [0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001],
  'learning_rate_init': [0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001],
  'learning_rate': ['adaptive'],
}
gs = GridSearchCV(mlp, param_grid, cv=2)
gs.fit(X_train, y_train)
y_pred = gs.predict(X_test)
```

This process is very time consuming due to the amount of the given parameters trained on the large-scale dataset. After the trained multi-layer perceptron is ready, the results can be stored and the received values can be plotted by Matplotlib. The score was computed by the L2 norm of the loss function, which means the difference between the original and the predicted value.

```
print(gs.best_params_)
print(gs.score(X_test, y_test))
print('shares', gs.best_params_, gs.score(X_test,y_test), sep='; ', file=open
    ('InversionResults.txt', 'a'))

with open('gridSearchResults.txt', 'a') as f:print("{} \n \n {} \n  \n Best
    Estimator: \n {} \n with {}".format(param_grid, gs.cv_results_, gs.
    best_estimator_, gs.best_score_), file=f)

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

The results of the training are the following where the original values are colored blue and the predicted ones have color orange:

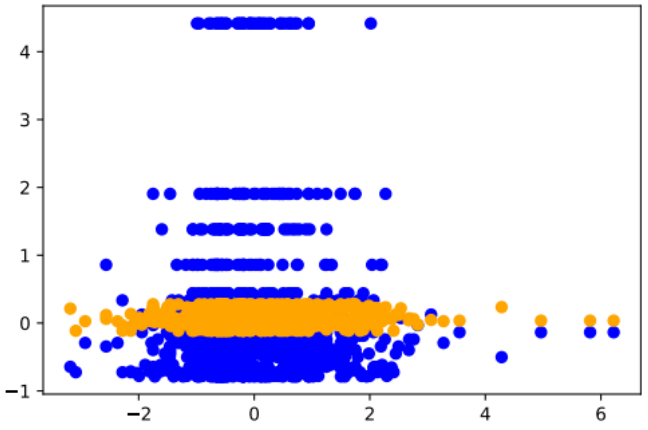

Figure 4.2: The result of the best prediction

```
shares; {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (30, 90,
    180, 90, 30), 'learning_rate': 'adaptive', 'learning_rate_init': 0.0001,
    'solver': 'sgd'}; 0.01229680126603272
```

which means, the score was 0.01229680126603272 and the best estimator parameters were the ReLU activation function and SGD as optimization algorithm, with the use of 0.001 alpha value and 0.0001 initial learning rate.

This really bad score does not mean that the model will be bad at inversion, due to the inversion will show the importance of every single feature to the target. The reason is very simple. The WLK inversion works as the backpropagation algorithm, which means it contains a recursive equation whose iteration for inversion can be solved by computing its derivative. It can be imagined by if the WLK inversion has a forward phase that contains a fitting algorithm where the derivatives are computed and a backward phase where the

recursion is computed by the derivatives. These values makes the new outputs that will be the result of the inversion.

4.2.3 Inverting the MLP

The inversion is an iterative process, which lasts for very long. Here the target values are the new predictors $X\_inverse$ with the knowledge comes from the other features X. Every iteration calculates one single feature's values $y\_inverse$ and also its accuracy between the feature's original and the calculated values.
The new training ($X\_inverse\_train$, $X\_inverse\_test$) and testing ($y\_inverse\_train$, $y\_inverse\_test$) sets has to be made first.

```
X = pd.DataFrame(X)
X.columns = dataset_copy.drop(columns='shares').columns

for X_value in X:
  X_inverse = X.drop(columns=X_value).values
  X_inverse_help = pd.DataFrame(y_train)
  y_pred = pd.DataFrame(y_pred)

  for y_pred_value in y_pred.values:
    y_pred_value = pd.Series(y_pred_value)
    X_inverse_help = X_inverse_help.append(y_pred_value, ignore_index=True)

  X_inverse = pd.DataFrame(X_inverse)
  X_inverse.insert(0,'shares',X_inverse_help)
  X_inverse = X_inverse.values

  y_inverse = X.get(key=X_value).values


  X_inverse_train, X_inverse_test, y_inverse_train, y_inverse_test =
    train_test_split(X_inverse, y_inverse, test_size=0.3, random_state=0)


  [...]
```

Then the predictions can be made by fitting the multi-layer perceptron model with the same parameters as during the training phase. This is an important step to get the appropriate inversion values.

```
X = pd.DataFrame(X)
X.columns = dataset_copy.drop(columns='shares').columns

for X_value in X:
  X_inverse = X.drop(columns=X_value).values
  X_inverse_help = pd.DataFrame(y_train)
  y_pred = pd.DataFrame(y_pred)

  for y_pred_value in y_pred.values:
    y_pred_value = pd.Series(y_pred_value)
    X_inverse_help = X_inverse_help.append(y_pred_value, ignore_index=True)

  X_inverse = pd.DataFrame(X_inverse)
  X_inverse.insert(0,'shares',X_inverse_help)
  X_inverse = X_inverse.values

  y_inverse = X.get(key=X_value).values
```

```
X_inverse_train, X_inverse_test, y_inverse_train, y_inverse_test =
  train_test_split(X_inverse, y_inverse, test_size=0.3, random_state=0)

gs.fit(X_inverse_train, y_inverse_train)
y_inverse_pred = gs.predict(X_inverse_test)

print(gs.best_params_)
print(gs.score(X_inverse_test, y_inverse_test))
print(X_value, gs.best_params_, gs.score(X_inverse_test,y_inverse_test),
  sep='; ', file=open('InversionResults.txt', 'a'))

plt.plot(X_inverse_test, y_inverse_test, 'o', color='blue')
plt.plot(X_inverse_test, y_inverse_pred, 'o', color='orange')
plt.savefig('plots/' + str(X_value) + '_prediction.pdf')
plt.show()
```

In the end of each process, the resulted score and the parameters that was used for the best prediction are written to a file. Furthermore the original values of the features and the inverted ones are plotted.

4.3 Results

The obtained results helps to understand the operation of the Online News Popularity dataset. The inversion revealed that what kind of extent of which features affect the number of shares. The inversion revealed the extent of how features affect the number of shares one by one. Those features that converge the most to 1, have the greatest influence.

These features's scores resulted at least 0.95 accuracy which means, these features affect the number of shares the most.

```
kw_max_max; {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30,
    90, 180, 90, 30), 'learning_rate': 'adaptive', 'learning_rate_init':
    0.01, 'solver': 'lbfgs'}; 0.999283661233451
LDA_04; {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30, 90,
    180, 90, 30), 'learning_rate': 'adaptive', 'learning_rate_init': 0.001, '
    solver': 'lbfgs'}; 0.9978949301566046
kw_min_min; {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30,
    90, 180, 90, 30), 'learning_rate': 'adaptive', 'learning_rate_init':
    0.01, 'solver': 'lbfgs'}; 0.9978265596014307
LDA_00; {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30, 90,
    180, 90, 30), 'learning_rate': 'adaptive', 'learning_rate_init': 0.01, '
    solver': 'lbfgs'}; 0.989874004275313
rate_positive_words; {'activation': 'tanh', 'alpha': 0.01, '
    hidden_layer_sizes': (30, 90, 180, 90, 30), 'learning_rate': 'adaptive',
    'learning_rate_init': 0.001, 'solver': 'lbfgs'}; 0.9858555965431666
rate_negative_words; {'activation': 'tanh', 'alpha': 0.01, '
    hidden_layer_sizes': (30, 90, 180, 90, 30), 'learning_rate': 'adaptive',
    'learning_rate_init': 0.001, 'solver': 'lbfgs'}; 0.9827638672011125
LDA_03; {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30, 90,
    180, 90, 30), 'learning_rate': 'adaptive', 'learning_rate_init': 0.0001,
    'solver': 'lbfgs'}; 0.9631754471905054
self_reference_avg_sharess; {'activation': 'tanh', 'alpha': 0.01, '
    hidden_layer_sizes': (30, 90, 180, 90, 30), 'learning_rate': 'adaptive',
    'learning_rate_init': 0.0001, 'solver': 'lbfgs'}; 0.9583727915872209
```

The value of *kw_max_max* is computed from ranking all article keyword maximum shares (known before publication) in order to get the best keywords. Also, *kw_min_min* is the

worst keyword from the ranking of minimum shared articles. LDA_04, LDA_00 and LDA_03 are computed by the Latent Dirichlet Allocation algorithm, which identified the top relevant topics and then measure the closeness of current article to such topics. The $rate\_positive\_words$ is the rate of positive words among non-neutral tokens and $rate\_negative\_words$ stands for the negative ones. $self\_reference\_avg\_sharess$ means the average shares of referenced articles in Mashable website.

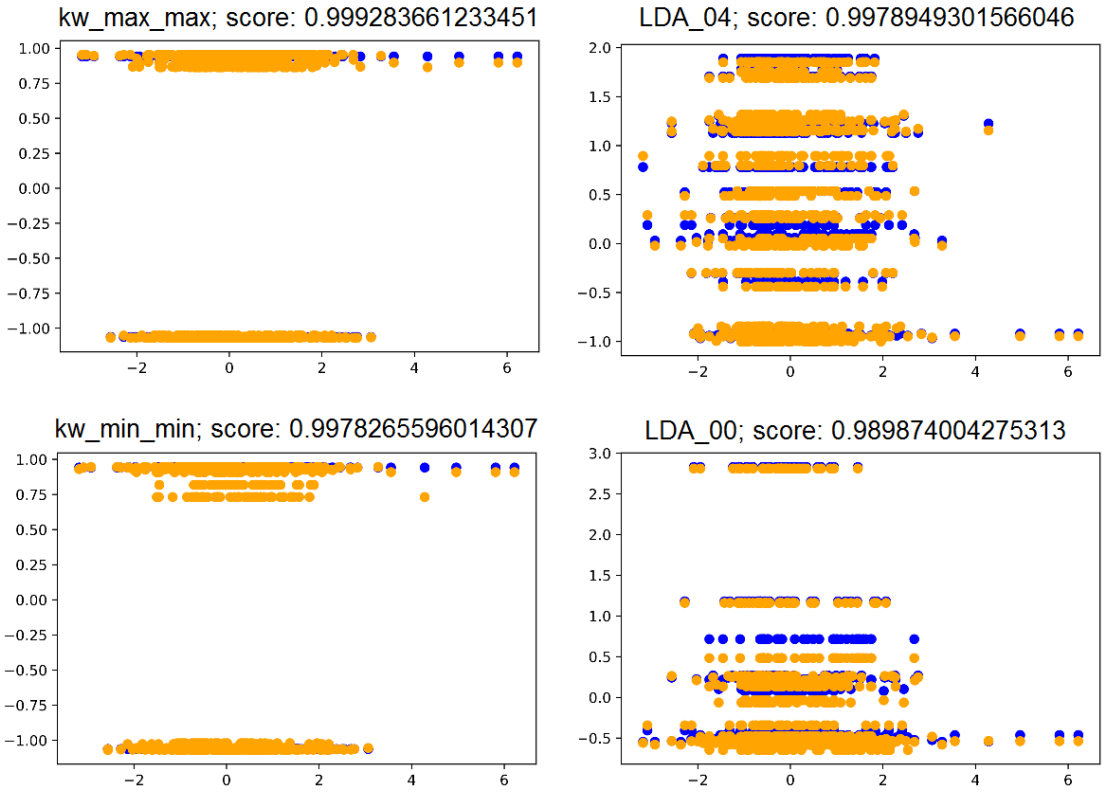

Figure 4.3: The best results of the inversion

At the representation of the resulted values of inversion, the original ones are colored blue and the inverted values have a color orange.

As a summarization it is exciting to see that the best result all came from the combination of almost the same parameters. They are the **hyperbolic tangent** as activation function, **L-BFGS** as optimization method, **0.01** as alpha value and **0.001** or **0.0001** as learning rate.

It is also an interesting result, that this combination is not equal with the best parameters of the training.

Chapter 5

Summary

asd

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

[3] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

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

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

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

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

[8] S.J.R. G, E.A. Christensen, and F.J. Blanco-Silva. *Learning SciPy for Numerical and Scientific Computing - Second Edition*. Community experience distilled. Packt Publishing, 2015.

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

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

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

[12] Md Taufeeq Uddin, Muhammed Patwary, Tanveer Ahsan, and Mohammed Shamsul Alam. Predicting the popularity of online news from content metadata. pages 1–5, 10 2016.

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