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, March 31, 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 [3] - 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 [7] 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. Now all of the predictive features are numeric values. There were nominal features like the day of the publication or the type of the data channel. They were transformed by one-hot encoding. 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.

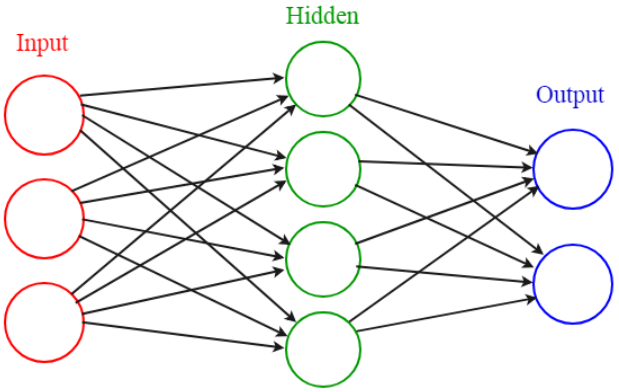

Figure 2.1: A neural network, where the circles represent the artificial neurons as nodes

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 [5].

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:

- **Scikit-Learn** for machine learning tools;
- **SciPy** for containing a wide array of numerical tools such as numerical integration, and interpolation and its package **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'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.

mathematical background, tools.....

data mining techniques, crucial concepts of data mining

data cleaning (?)

3.2 Feedforward Neural Networks

There are many different types of neural networks, one of them is the feedforward neural network [4]. 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 [6]. 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: $z = w_1x_1 + w_2x_2 + \cdots + w_mx_m = \sum_{j=1}^{m} w_jx_j$

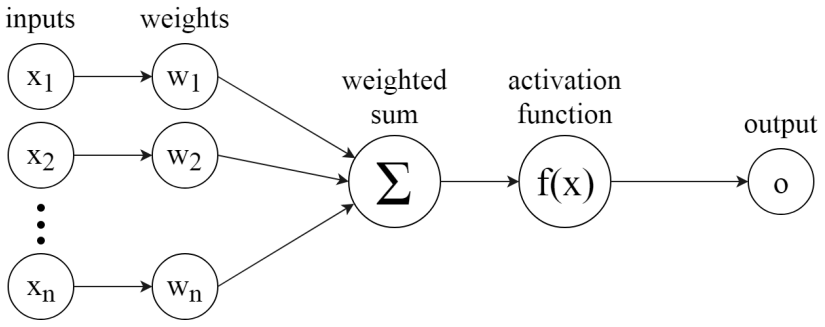

Figure 3.1: The appropriate weights are applied to the inputs and the resulting weighted sum passed to an activation function that produces the output.

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

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

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

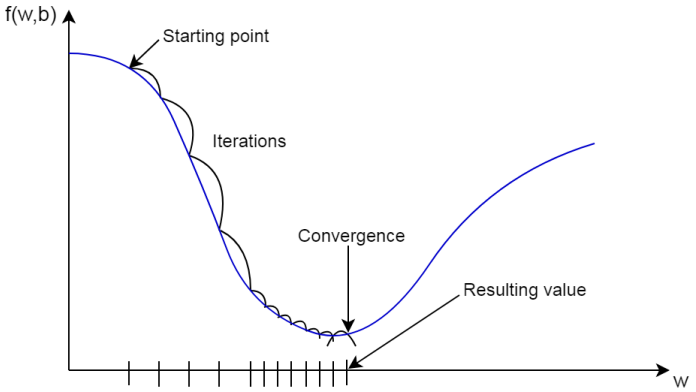

Figure 3.2: The gradient descent is an optimization method used by backpropagation

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.

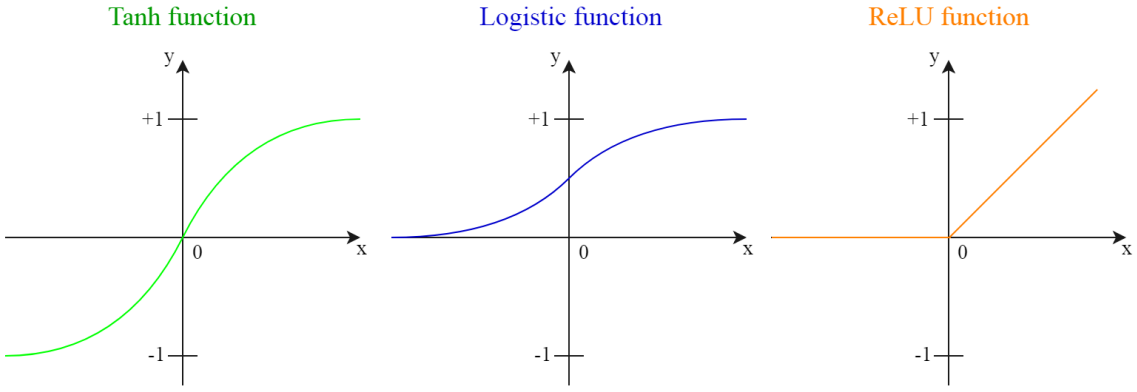

Figure 3.3: The graphs of the most popular non-linear activation functions

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. [?][?]

Stochastic Gradient Descent

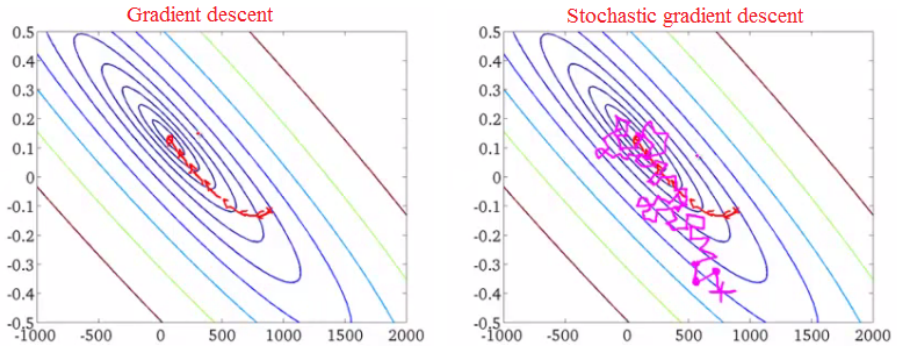

Figure 3.4: The difference between the process of the gradient descent and the stochastic gradient descent methods

Stochastic gradient descent [2] 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 $B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$.

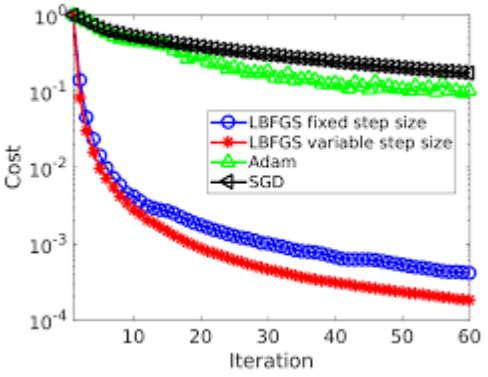

Figure 3.5: The optimization of SGD, Adam and L-BFGS with respect of cost and iteration

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 - the set of input neurons
O - the set of output neurons
H - the set of 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.

Nearest Inversion

Nearest inversion method is another single-element inversion method that can solve constrained optimization problems. The goal is to find that inverse solution that is lying nearest to a specified point. Assume that $f(i)$ is a given function, c is the level of the desired output and i_0 is the initialized point. The method is searching for that i^*, which satisfies $f(i^*) = c$ and be the closest to i_0. For describing the distance, Euclidean distance is used.

3.5 Problem Statement

constrained optimization problem (????)

describing the problem -> inversion!!

Chapter 4

Implementation

4.1 Python

asd

4.1.1 Scikit-Learn

asd

4.1.2 Training Libraries

asd

4.1.3 Inversion Libraries

asd

4.2 The Implementation (?)

new section name (?)

4.2.1 Optimising/Analysing the Dataset (?)

asd

4.2.2 Training the Neural Network

asd

4.2.3 Inverting the MLP

asd

4.3 Results

asd

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

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

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

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

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

[7] 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.