LU TP 17-27
May 2017

Using LASSO regularization as a feature selection tool.

Erik Torstensson

Department of Astronomy and Theoretical Physics, Lund University

Bachelor thesis supervised by Mattias Ohlsson

Abstract

The subject of deep learning has become increasingly popular, especially for machine learning applications where a large number of input variables have to be processed. However, there are instances of problem solving, where a full understanding of the variables is of high importance. When dealing with data sets containing a large number of input variables, the established methods of feature selection require a considerable time investment. Regularization is a method typically associated with prevention of overtraining, but in this study, the possibility is explored of using LASSO regularization as a feature selection tool. The input variables of several data sets were ranked with respect to a measure of synaptic weight magnitude. A conclusion was drawn that this method is a very fast and efficient way of filtering out less important variables.

Populärvetenskaplig sammanfattning

Artificiella neuronnät är ett samlingsnamn för maskininlärningsmetoder som försöker efterlikna biologiska system i sin struktur. Ett neuronnät förbereds, med så kallad träning, för till exempel mönsterigenkänning, genom att man presenterar exempel på den typ av data man vill identifiera. Inom ett visst intervall uppdateras nätverkets inre struktur, med målet att utsignal anpassas till den signal som identifierar datatypen. När nätverket kan generalisera den samlade informationen, och så bra som möjligt fylla den funktion som ändamålet kräver, är träningen klar. Detta kan liknas till en hjärnas plasticitet och förmåga att anpassa sig till en ny färdighet eller kunskap. Inom mönsterigenkänning handlar det ofta om att identifiera nya exempel, som aldrig presenterats för nätverket tidigare, och att placera dem i de kategorier som man tidigare har tränat för.

Feature selection är metoder som hittar, och väljer ut, de variabler som i högsta grad påverkar ett ändamål, och sållar bort de variabler som minst påverkar ändamålet. Vilken typ av variabler som insignalen består av beror helt på ändamålet, och kan vara allt ifrån ett fåtal variabler, till vektorer av stor dimension. Inom medicin och diagnostisering ställs ett särskilt högt krav på att resultat kan tolkas in i minsta detalj, och detta är en av de stora drivkrafterna till att utveckla bättre metoder för att bearbeta variabler.

Regularisering är en särskild typ av funktionsanpassning som innebär att man med ytterligare mått på komplexitet, kan tvinga en funktion till att anta en mjukare och mer generaliserad form. Ett problem som ofta förekommer i samband med träning av neuronnät är så kallad överinlärning, då ett nätverk lär sig detaljer i träningsexempel istället för att generalisera, och regularisering är att effektivt sätt att motverka detta.

LASSO (Least Absolute Shrinkage and Selection Operator), är en regulariseringsmetod som användes i den här studien. Det intressanta med LASSO är att den ställer ett särskilt högt krav på nätverkets inre struktur och de variabler som insignalen innehåller. Frågeställningen för studien var: ”Kan regularisering av neuronnät med LASSO, ge upphov till en användbar rangordning av variabler?”

Contents

1 Introduction

Artificial neural networks have become increasingly popular for their versatility and wide range of applications. A use for specialized artificial neural networks that is actively being researched [3], is the possibility of letting a network process raw data and automatically separate useful input variables from random noise and less relevant variables. For high performance applications the fine details of a network become vital, and understanding exactly how input data affects the network performance is needed to justify the results.

Preparing a network, for a task such as classification, is done through a process known as training. Examples of data are presented to the network, and the network's inner structure is updated until the output signal conforms with the so called label, which is the predetermined classification of the example. A fully trained network is capable of approximating any function [2], given enough complexity of its structure.

Feature selection, is the subject of studying the input variables of a data set, and extracting information about its intricacies (features). As a concrete example, the input vector of a data set could have variables that do not affect the classification, and the variables that do affect the classification could be of varying importance. A deep neural net would be able to discern and find the relevant features thanks to the many layers of abstraction [1], known better as feature learning, but there are also methods for dealing with these issues in a more hands-on way. The most naive form of feature selection is to rank input variables according to some measure of importance, and removing variables from the bottom of the list. Finding the most important subset of input variables can greatly reduce the complexity of both the problem formulation and the method leading to its solution. Reduced complexity is sought after in machine leaning for more efficient training, less redundancy in network design, and a better understanding of how input data affect problem solutions.

A network that has been trained with few or too specific examples from a data set can become overtrained, which is similar to an overfitted function approximation. In general, this correlates with network weights being large or in excess. This is expressed by a difference in the errors during training and errors when evaluating data never seen before. The overtrained network becomes an expert in classifying the training data set, and does poorly in terms of generalization. By using methods such as regularization, which is the addition of penalties of the complexity of the network, function approximations can be forced to become more generalized. In most cases this means reducing the size of the network weights.

LASSO (Least Absolute Shrinkage and Selection Operator) [13] is a regularization technique with a particularly strong ability to force synaptic weights to zero. In this study, we explore the possibility of LASSO being a solution to both overtraining and feature selection. Forcing weights close to zero is a very effective way of reducing complexity, and the resilience of surviving weights, can be associated with connected nodes, resulting in a straightforward measure of ranking.

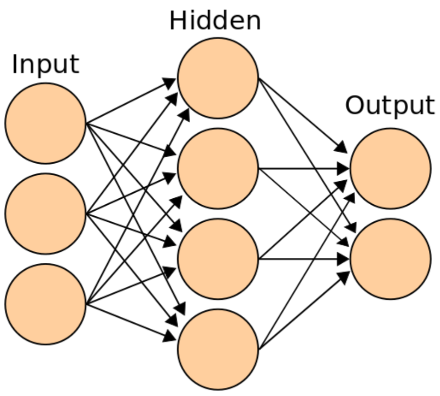


Figure 1: An example schematic [6] of a fully connected multi layer perceptron with three input nodes, four hidden nodes, and two output nodes.

2 Background and Theory

2.1 Multi layer perceptron

A multi layer perceptron (MLP) is based on the original perceptron by Rosenblatt [12] and uses multiple layers of computational nodes to achieve non linear function approximation. The layers are referred to as the input layer, output layer, and one or more hidden layers [4]. The connections between nodes in a network are called synaptic weights [4], and define the network function and performance. In most cases the network is fully connected, which means that every node in one layer is connected to every node in adjacent layers (figure 1).

Input vectors $\mathbf{x}$ can contain anything from continuous variables, to discrete or binary variables. For the first hidden layer with nodes j, the product between input vector and synaptic weight returns the so called induced local field v_j (1) [4]:

$$v_j = \sum_i w_{ij} x_i + w_{0j} \ , \tag{1}$$

where w_{0j} is the weight in each node associated with the applied bias (+1). The bias acts as a buffer and can control the solution distance from the origin. A non-linear activation function φ is required for the network to function as a universal approximator [2], and is a condition for non-linearity of solutions [4]. The universal approximation theorem [2] states that a feed-forward network with at least one hidden layer can approximate any continuous function in a given closed space. The network signal passes through an activation function in the hidden layer(s) and in the output layer, and can be different for each layer. A sigmoidal function, such as the hyperbolic tangent (2), figure 2a, or the logistic function (3), figure 2b, are among the most common activation functions, and they can be interpreted as soft step functions [4]

$$\varphi(v_j) = \tanh(v_j) \ , \tag{2}$$

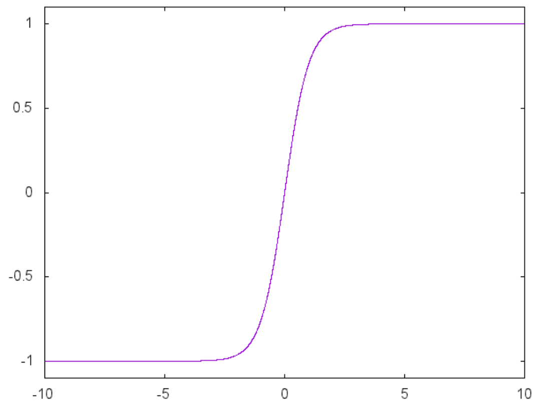
$$\varphi(v_j) = \frac{1}{1 + e^{-v_j}} \ . \tag{3}$$

Another commonly used activation function is the rectified linear unit (ReLU) (4), figure 3

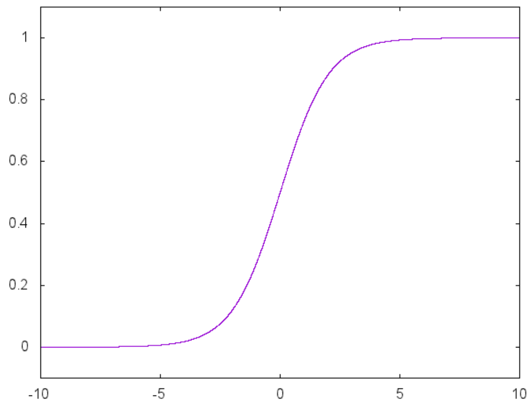
$$\varphi(v_j) = \begin{cases} v_j & \text{if } v_j > 0 \\ 0 & \text{else} \end{cases} . \tag{4}$$

The output of the hidden layer, defined by the activation functions $\varphi_{hidden}(v_j)$, can also be interpreted as the input of the next layer y_j. This defines the induced local field u of a single output node as:

$$u = \sum_j w_j y_j + w_0 . \tag{5}$$



(a) Hyperbolic tangent.



(b) The logistic function.

Figure 2: Two sigmoidal activation functions.

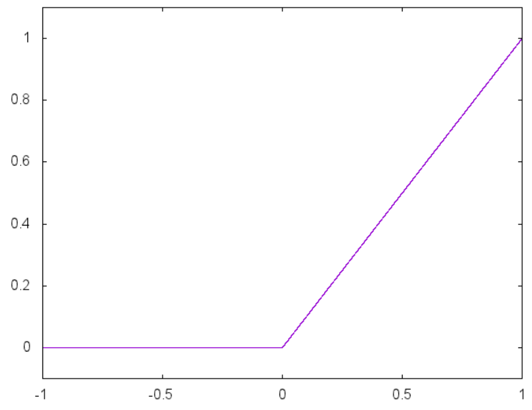


Figure 3: The rectified linear unit activation function.

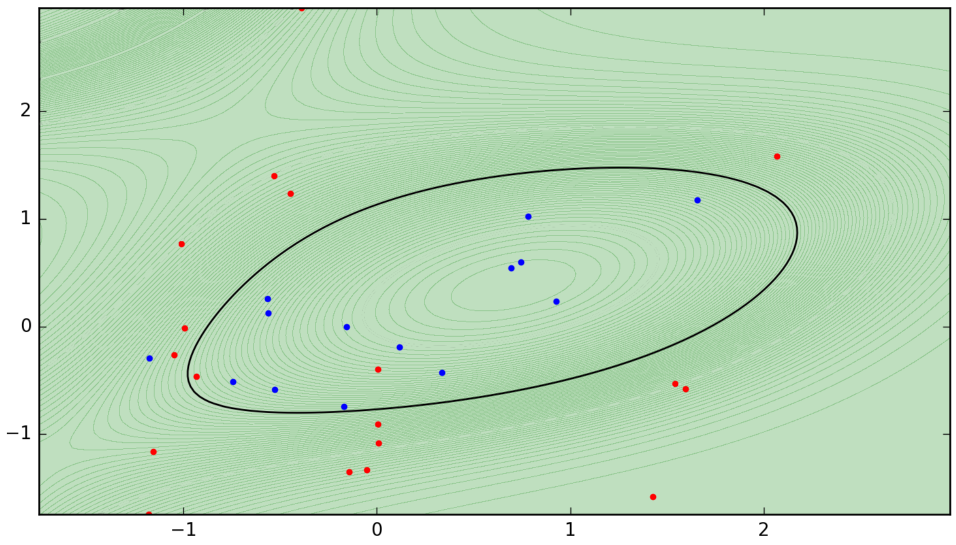


Figure 4: The contour identifying two classes, red and blue, in a two dimensional solution space. The data set was randomly generated and separated into two classes by the median value of an arbitrary polynomial function. A network was trained to separate the two classes, and ended up with an elliptically shaped decision boundary. In this example, 3 out of 32 data points were classified wrong, as there was some overlap between the classes.

2.2 Problem formulation

The general idea of the MLP learning algorithm is to turn an input vector $\mathbf{x}$ into an output signal z that is as close as possible to the desired output signal d. This becomes possible because of the universal approximation theorem [2]. Given an input vector $\mathbf{x}$ and corresponding label d the goal is to adjust network weights in such a way as to match the network output with the desired output. This is achieved by minimizing some measure of error based on the difference between network output and label of the example. Input data for classification is in most cases structured in a labelled set $\{\mathbf{x}_n, d_n\}$.

Solution spaces can be separated with a decision boundary of lower dimension. For example, an input vector of two continuous variables can be separated by a line cutting across a two dimensional plane, or in any other shape such as the elliptical contour in figure 4.

2.3 Training

The process of changing a network to minimize a given error measure is called training [4]. The training is considered complete, when the chosen error measure has converged to a constant value, or the error measure is under a given threshold.

2.3.1 Loss function

The loss function is the measure of error of a given network. Our choice of loss function is the mean square error (MSE) with respect to output signal and label. The mean square error is given by:

$$\mathcal{E} = \frac{1}{N} \sum_{n}^{N} \frac{1}{2} (d_n - z_n)^2 \, , \tag{6}$$

where z_n and d_n are output and label for example n, and N is the number of examples in the data set. Additional terms can be added to the loss function, depending on the application, but the error is the central component.

2.3.2 Gradient descent

In the method of gradient descent, the direction of each weight update is opposite to the rate of change (gradient) of the loss function. This moves the network loss function in small steps towards the nearest local minimum in the loss function solution space. The change Δw_{ij} of each weight is:

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}}{\partial w_{ij}} \, . \tag{7}$$

The step size η is known as the learning rate. In every time step t, the weights of each layer are updated as following.

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \, . \tag{8}$$

2.3.3 Backpropagation

Training an MLP is done using the so called backpropagation algorithm [4]. Backpropagation is the implementation of gradient descent, that refers to the network output being the forward evaluation, and the gradient of the loss function being the backwards propagated evaluation. Using gradient descent, the weight update is proportional to the partial derivative of $\mathcal{E}$, which can be calculated using the chain rule. For a single output node with induced local field $u(n)$ and output signal $z_n = \varphi_{output}(u(n))$, the following expansion is made with respect to weights w_j between the hidden and output layers:

$$\frac{\partial \mathcal{E}}{\partial w_j} = \sum_n^N \frac{\partial \mathcal{E}}{\partial z_n} \frac{\partial z_n}{\partial u(n)} \frac{\partial u(n)}{\partial w_j} \, , \tag{9}$$

$$\frac{\partial \mathcal{E}}{\partial z_n} = \frac{\partial}{\partial z_n} \frac{1}{N} \sum_n^N \frac{1}{2}(d_n - z_n)^2 = -\frac{1}{N}(d_n - z_n) \, , \tag{10}$$

$$\frac{\partial z_n}{\partial u(n)} = \frac{\partial}{\partial u(n)} \varphi_{output}(u(n)) = \varphi'_{output}(u(n)) \, , \tag{11}$$

$$\frac{\partial u(n)}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_j w_j y_j(n) = y_j(n) \, . \tag{12}$$

The error of the network $\mathcal{E}$, is backpropagated through the hidden nodes j, and an expansion can be made with respect to weights w_{ij} between input and hidden layers:

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \sum_n^N \frac{\partial \mathcal{E}}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ij}} \, , \tag{13}$$

$$\frac{\partial \mathcal{E}}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)} \frac{1}{N} \sum_n^N \frac{1}{2}(d_n - z_n)^2 = \tag{14}$$

$$= -\frac{1}{N} \sum_n^N (d_n - z_n) \frac{\partial z_n}{\partial y_j(n)} = \frac{1}{N} \sum_n^N (z_n - d_n) \frac{\partial z_n}{\partial u(n)} \frac{\partial u(n)}{\partial y_j(n)} \, , \tag{15}$$

$$\frac{\partial z_n}{\partial u(n)} = \varphi'_{output}\left(u(n)\right) \, , \tag{16}$$

$$\frac{\partial u(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)} \sum_j w_j y_j(n) = w_j \, , \tag{17}$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial}{\partial v_j(n)} \varphi_{hidden}(v_j(n)) = \varphi'_{hidden}(v_j(n)) \, , \tag{18}$$

$$\frac{\partial v_j(n)}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_i w_{ij} x_i(n) = x_i(n)\,. \tag{19}$$

The full expressions for the partial derivatives of $\mathcal{E}$ with respect to weights are:

$$\frac{\partial \mathcal{E}}{\partial w_j} = \frac{1}{N} \sum_n^N (z_n - d_n)\, \varphi'_{output}(u(n)) \cdot y_j(n) \qquad \text{weights after the hidden layer.} \tag{20}$$

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{1}{N} \sum_n^N (z_n - d_n)\, \varphi'_{output}(u(n)) \cdot w_j \cdot \varphi'_{hidden}(v_j(n)) \cdot x_i(n) \qquad \text{weights after the input layer.} \tag{21}$$

2.3.4 Dynamic learning rate

A dynamic learning rate was adopted [10] that greatly improves the performance of gradient descent.

$$\eta_{t+1} = \begin{cases} \eta_t \cdot \gamma & \text{if } \mathcal{E}(t+1) \geq \mathcal{E}(t) \\ \eta_t \cdot (1 + \frac{1-\gamma}{10}) & \text{otherwise} \end{cases}\,, \tag{22}$$

where gamma is a manually adjusted decay parameter less than one. This dynamic learning rate is usually called a bold driver [10], because it slightly increases the learning rate while the training is progressing towards lower values of the loss function, while in general you want the learning rate to decrease as the loss function converges to an optimal value.

2.3.5 Resilient Backpropagation (RPROP)

An alternative method for updating weights [11] was implemented to test the consistency of the program. RPROP uses learning rates corresponding to individual weights, and dynamically updates the learning rates very similarly to (22). This means that the learning rates can be of entirely different magnitudes, and some weights might converge at an optimal value early while others require more time.

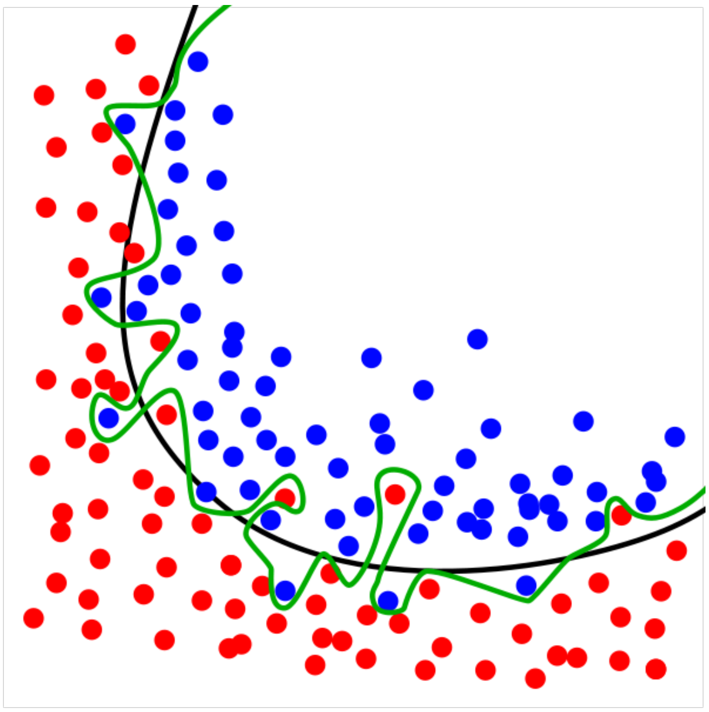


Figure 5: Diagram [5] showing overfitting of a classifier.

2.4 Overtraining

Overtraining, is the concept of a completed network training having been misguided in one or multiple ways. If the examples presented in the training data are not fully representative of the problem at hand, the network can inadvertently be trained to give too specific solutions, and never reach the generalized solution. A too specific solution would mean that the network recognizes details in a training data set, where there would be none in a data set with more examples. The function to be approximated is in most cases more smooth than the training data set presented to the network. Training data can be limited for many reasons and this makes proper validation of results of utmost importance. Data sets could also contain some amount of noise that would be averaged to zero with a larger amount of training examples.

Given that a network has a large enough number of nodes, and therefore weights, overtraining can be interpreted as the weights going beyond the required magnitude, to fully classify the training data set. A naive comparison of overtraining can be made with overfitting of a two dimensional decision boundary. Figure 5 shows how two classes of data point, in red and blue, can be separated by either a smooth curve, shown in black, or an extremely complicated curve, shown in green. The black curve does not correctly classify all data points in the example, but in many real cases there is enough uncertainty, and random noise, for this to be the correct decision boundary. Some caution has to be taken into account that input variables in machine learning are not always continuous variables, but also discrete or binary. A method for handling overtraining, called regularization, will be discussed in the following section.

3 Methods

3.1 Regularization

Regularization is a method for reducing overtraining by introducing an additional term to the loss function. In general it can be said that a regularization term would penalize a function approximation for deviating from its most generalized form. To summarize, it can be found [3, 4] that a lower complexity of a network usually leads to finding a more generalized solution. A typical regularization term for an MLP is a parameter α multiplied by some function f of the synaptic weights w_{ij}. For all mentions of regularization in this study, it is implied that the penalty is applied to the first layer of weights only. It is reasonable to expect a separate regularization parameter for each layer of weights, but for the purpose of feature selection it was not explored at this time. Given a loss function $\mathcal{E}$, a regularization term $\mathcal{E}_c$ is added giving a total loss function:

$$\mathcal{E}_{tot} = \mathcal{E} + \mathcal{E}_c \tag{23}$$

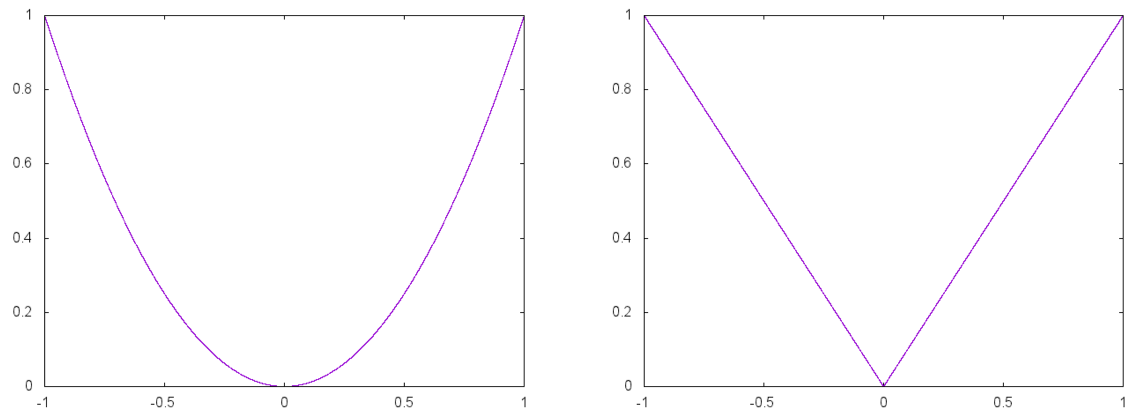

(a) L_2 weight decay. (b) LASSO penalty for weight decay.

Figure 6: L_2 compared to LASSO.

3.1.1 Weight decay

Methods of regularization where weights are penalized for their magnitude are called weight decay [4]. A term is added to the loss function which directly increases the error when weights are increased. This turns the training into a battle between lowering weight magnitudes and lowering the total error. The most common form of weight decay is the L_2 (24), which is the square of the weight multiplied by a regularization parameter. See figure 6a for a representation of this penalty. The penalty added to the loss function is:

$$\mathcal{E}_c = \alpha \sum_{ij} w_{ij}^2 \, . \tag{24}$$

3.1.2 LASSO

LASSO (Least Absolute Shrinkage and Selection Operator) is a type of linear regression method that applies the constraint of the sum of absolute values of synaptic weights to the loss function [3, 13]. This is a more extreme form of weight decay, and is effectively the same thing as an L_1-penalty. The regularization parameter α controls the slope of the penalty (figure 6b).

$$\mathcal{E}_c = \alpha \sum_{ij} |w_{ij}| \, . \tag{25}$$

As our gradient descent relies on the partial derivative of the loss function, we can easily deduce that the partial derivative of the LASSO regularization term is the sign of the weight. Some caution has to be taken for the case of a weight being exactly zero.

$$\frac{\partial \mathcal{E}_c}{\partial w_{ij}} = \alpha \cdot sign(w_{ij}) \, . \tag{26}$$

3.2 Using LASSO as a feature selection tool

The method proposed in this paper, is that in addition to using LASSO for regularization, the same constraint on weight magnitudes can give rise to a measure of input variable importance. Without any regularization, the magnitude of weights is somewhat arbitrary, but given a constraint of magnitude, a relative difference in weight resilience should become apparent. By associating weights with preceding connected nodes, this provides us with a naive, easy to implement, ranking of node importance. With respect to the input nodes, this is the same as ranking the input variables, and can be considered a form of feature selection. Overtraining is not a condition for this method of feature selection to be viable, but it is often a problem that can be solved simultaneously.

The absolute values of weights, associated with each input node, were summed into a node magnitude coefficient:

$$c_i = \sum_{j} |w_{ij}| \, . \tag{27}$$

An illustration of the synaptic connections associated to one input node can be seen in figure 7. This data was represented and ranked in two different ways. First, the weight magnitudes were normalized to the coefficient of largest magnitude, and plotted as a function of the regularization parameter. The ranking was done at the optimal regularization parameter with respect to the validation error. Second, the previous function was integrated, giving a measure that is similar to the average value of coefficients. This directly gives the second method of ranking. A summary of both methods of ranking can be seen in table 1.

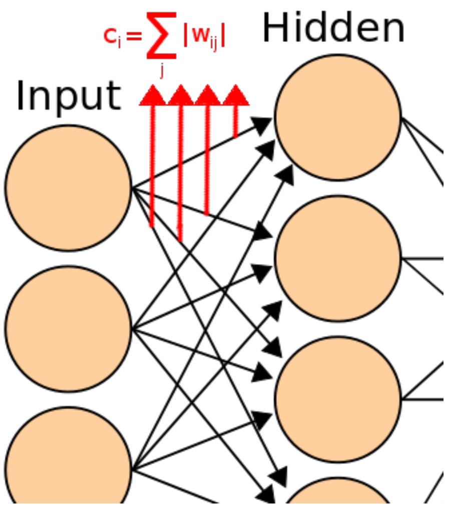


Figure 7: [6] The magnitude of synaptic weights, connected to each node in the preceding layer, were summed into a coefficient of node magnitude. This measure was used to rank nodes in descending order.

Method (a)	Method (b)
Normalized weight magnitudes, associated with each input node, summed into a node magnitude coefficient and represented as a function of the regularization parameter.	The function of method (a) integrated over the regularization parameter, which is similar to the average value of coefficients.

Table 1: The two proposed methods of ranking input variables using LASSO regularization.

3.3 Cross validation

To test the performance of a trained network, it is essential that there is as little overlap as possible between data used for training and data used for validation. Our validation method of choice is k-fold cross validation, which entails that a given data set is divided into k equal parts. The network is trained and validated k times, with a full reset between each run. One part is used for the validation, and the other k-1 parts are used for the training. This can be repeated as many times as necessary, resulting in an average of n times k networks trained and validated.

3.4 Normalization

Normalizing the input variables improves general performance [4]. If input variables are of substantially different magnitude, the training takes a lot more time to reach convergence. The following normalization was applied to the input variables of every data set, leaving them with a mean of zero and a variance of one.

$$x \quad \rightarrow \quad \frac{x- < x >}{\sigma_x} \, . \tag{28}$$

3.5 Implementation

An implementation of an MLP, and related validation methods, with more than 70 manually adjustable parameters, was written in the Python programming language. A minimal amount of external libraries were used, including SciPy [8] for matrix operations, and Matplotlib [7] for plotting data. Short scripts, also written in Python, were used for generating data sets, sorting and ranking data, editing a large number of plots simultaneously, and calculating differences between lists of rankings. The entire project had approximately 1500 lines of code, and 1 GB of collected data (including plots).

3.6 Datasets

Several data sets were used for experiments and validation of proposed methods. The data sets presented in the results were all provided by the supervisor Mattias Ohlsson. Two generated data sets were customized to give a concrete example of how extraneous input variables affect results, and how they can be handled by proposed methods. Pima and PacVsN are medical data sets, representing real world examples of problems where feature selection is of great importance.

3.6.1 Pima

The Pima Indians diabetes data set can be found at the UCI Machine Learning Repository [9]. This is a medical data set, from the National Institute of Diabetes and Digestive and Kidney Diseases (1990-05-09), classifying people with or without a positive diabetes diagnosis. The data contains 768 examples with eight input variables that are thought to be correlated with a positive diagnosis. A short description of each input variable can be seen in table 2.

1	Number of times pregnant
2	Plasma glucose concentration
3	Diastolic blood pressure
4	Triceps skin fold thickness
5	2-Hour serum insulin
6	Body mass index
7	Diabetes pedigree function
8	Age

Table 2: Description of the input variables in the Pima Indians diabetes data set [9].

3.6.2 Generated sets

Two randomly generated sets were studied, containing eight and twelve inputs, each one being 1500 examples in size. In the set with eight inputs, the first four inputs are Gaussian distributed variables, and the next four are random binary variables. In the second set with twelve inputs, the first eight inputs are the same as in the first, and the four additional variables are Gaussian distributed variables that were not used at all for the classification of examples. The function used for classification was the following, where the set was sorted by y and cut into two classes by the median value:

$$y = x_1x_2 + (x_1)^2 + x_3x_4 + 4x_5x_6 + 2x_7x_8 + \exp(x_3)$$

By looking at this function, an educated guess would be that input variables 3 and 1 had the greatest effect on the classification, followed by 5 and 6.

3.6.3 PacVsN

PacVsN is a medical data set containing 351 input variables and 341 examples. The input variables represents blood sample measurements of certain proteins, related to the immune response system, that may or may not be related to a positive diagnosis of pancreatic cancer. Measurements of both patients with a positive diagnosis and a control group of healthy individuals are part of the examples.

4 Results

For our purposes, there was no preprocessing of data sets (other than normalization), and the label was treated as a predetermined unchanging truth. Any erroneous label or noisy input variable were handled by methods such as regularization. A number of hidden nodes between 8 and 32 was chosen for each data set, to make sure the training error was minimized when no regularization was applied. Bias of each layer was set to a constant (+1) with bias weights being exempt from regularization. A regularization parameter was implemented for every layer of weights, but only the parameter of the input weights was non-zero, and that is the parameter presented as the x-axis in every plot. Early stopping of training was allowed, when the rate of change of the mean square error was sufficiently low, or the instantaneous value was under 0.02.

4.1 Pima

Training a network for the Pima data set was relatively fast and provided an excellent first example of overtraining. Hyperbolic tangent was used as the activation function for both layers of the networks. The training error, with no regularization applied, was close to zero, while the validation error was 0.32 ± 0.03.

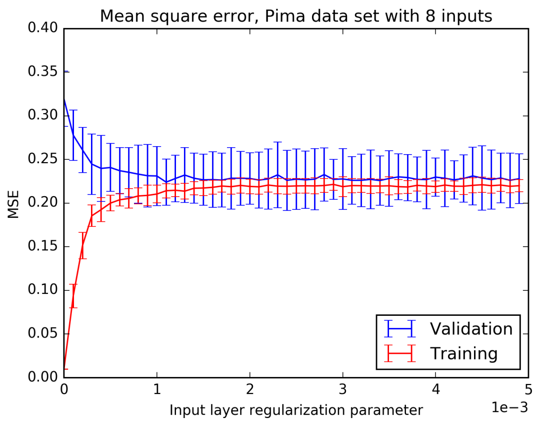


Figure 8: The mean square error of training and validation of the Pima data set.

4.1.1 Improvements by regularization

Increasing the regularization parameter reduced the difference between training and validation error very quickly, and they converged towards a value of 0.23 ± 0.02. See figure 8 for the errors as a function of regularization parameter.

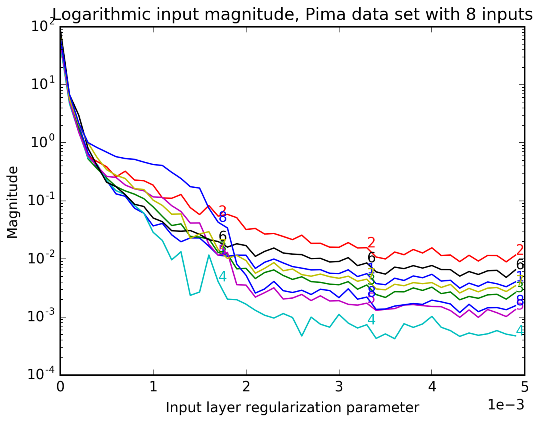


Figure 9: The summed logarithmic magnitudes of the weights associated with input nodes during training of the Pima data set.

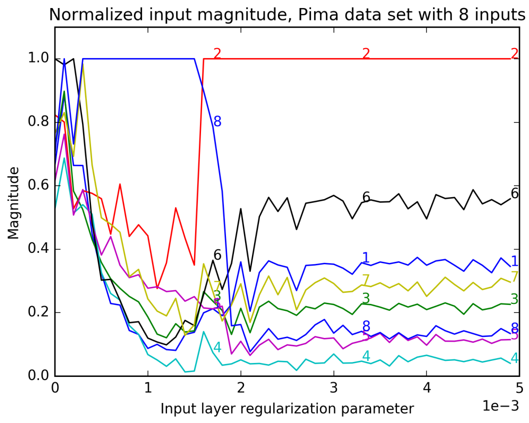


Figure 10: The summed and normalized magnitudes of the weights associated with input nodes during training of the Pima data set.

2	6	1	7	3	8	5	4

(a) Ranking of input node magnitude at optimal regularization.

2	6	8	7	1	3	5	4

(b) Ranking of total area of normalized input node magnitude.

Table 3: Two methods of ranking the input variables in the Pima diabetes data set [9].

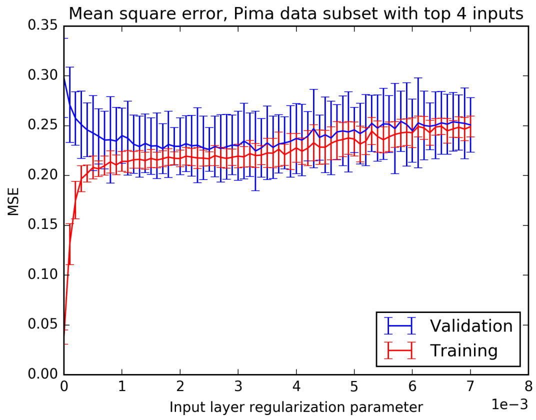


Figure 11: The mean square error of training and validation of the Pima data subset with the top 4 input variables of table 3a.

4.1.2 Features

The magnitudes of weights, associated with each input variable, are presented in figures (9, 10). Table 3 shows the ranking of input variables at the optimal value of the regularization parameter as well as the ranking of the area under the curves of figure 10. To validate the ranking of the data set, an additional training session was performed, using only the top 4 input variables of table 3a. The error of this validation can be seen in figure 11.

4.2 Generated sets

The generated sets were easy to train for and had relatively little overtraining. For the set with extra inputs, that had no part in the classification, the validation error was slightly higher regardless of regularization. The errors as a function of the regularization parameter for both sets can be seen in figures (12, 13). Hyperbolic tangent was used as the activation function for both layers of the networks.

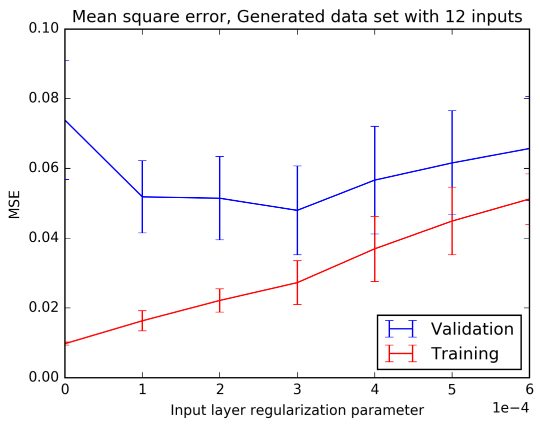


Figure 12: The mean square error of training and validation of the generated data set with 12 input variables.

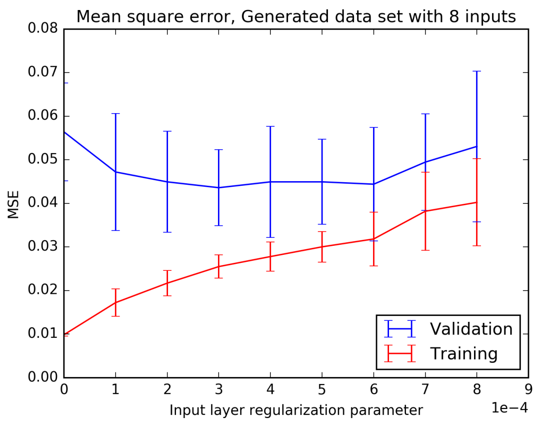


Figure 13: The mean square error of training and validation of the generated data set with 8 input variables.

4.2.1 Improvements by regularization

Regularization made a small but noticeable reduction of the validation error, and the training session was ended when the validation error started to increase. The optimal regularization parameter was 0.0003 ± 0.0001 for both sets.

4.2.2 Features

The ranking of input variables was very similar for both sets, and using both methods, and corresponds very well to the prediction of input variables 3, 1, 5, and 6 being the most important. Regularization effectively reduced the weights associated with the extraneous input variables to zero. Representations of the weight magnitudes can be seen in figures (14, 15, 16, 17). The final rankings of input variables can be seen in tables (4, 5).

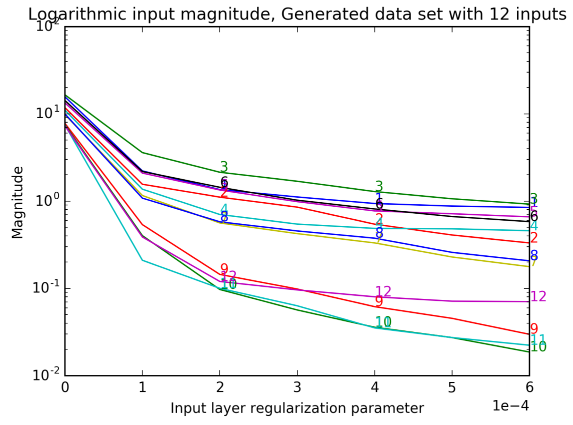


Figure 14: The summed logarithmic magnitudes of the weights associated with input nodes during training of the generated data set with 12 input variables.

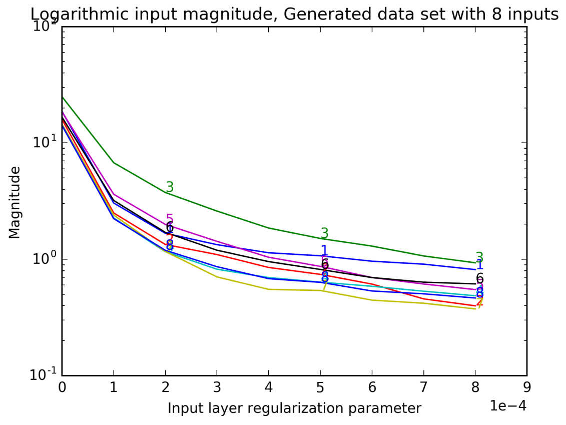


Figure 15: The summed logarithmic magnitudes of the weights associated with input nodes during training of the generated data set with 8 input variables.

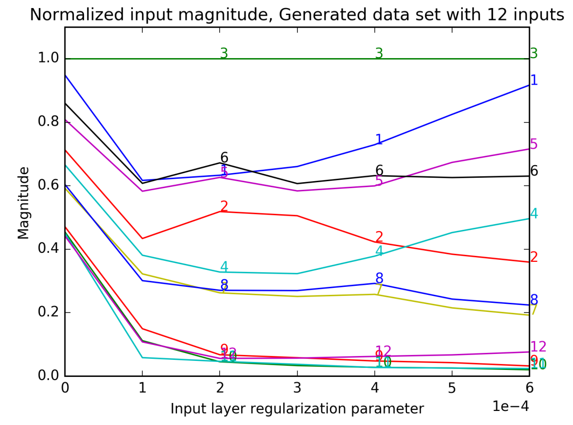


Figure 16: The summed and normalized magnitudes of the weights associated with input nodes during training of the generated data set with 12 input variables.

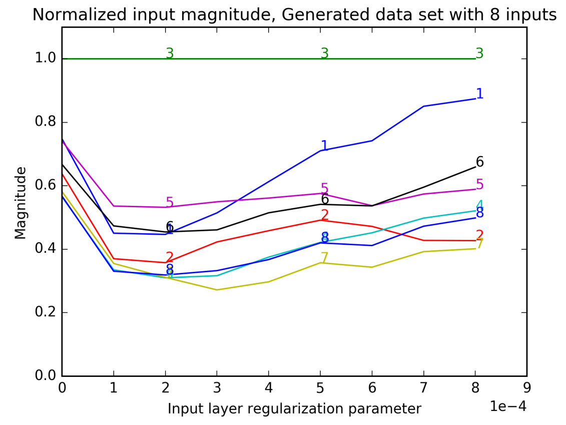


Figure 17: The summed and normalized magnitudes of the weights associated with input nodes during training of the generated data set with 8 input variables.

3	1	6	5	2	4	8	7	9	12	11	10

(a) Ranking of input node magnitude at optimal regularization.

3	1	6	5	2	4	8	7	9	12	10	11

(b) Ranking of total area of normalized input node magnitude.

Table 4: Two methods of ranking the input variables of the generated data set with 12 input variables.

3	5	1	6	2	8	4	7

(a) Ranking of input node magnitude at optimal regularization.

3	1	5	6	2	4	8	7

(b) Ranking of total area of normalized input node magnitude.

Table 5: Two methods of ranking the input variables of the generated data set with 8 input variables.

4.3 PacVsN

Training the PacVsN data set without regularization resulted in a particularly overtrained network. The training and validation errors of this data set can be seen in figure 18. ReLU was used as the activation function for the hidden layer, and the hyperbolic tangent was used for the output node.

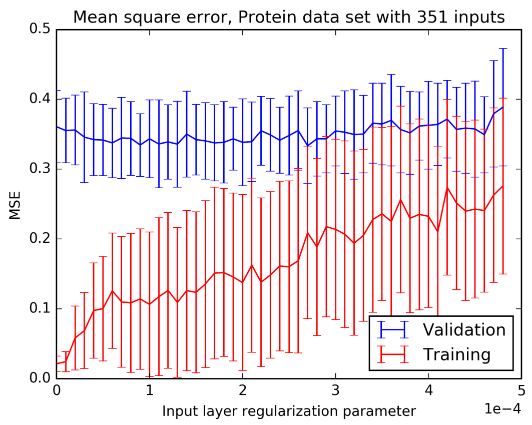


Figure 18: The mean square error of training and validation of the PacVsN data set with 351 input variables.

4.3.1 Improvements by regularization

Overtraining was greatly reduced with increasing regularization parameter and the optimal value was 0.00017 ± 0.00013. The training session was ended when the validation error surpassed the value at no regularization.

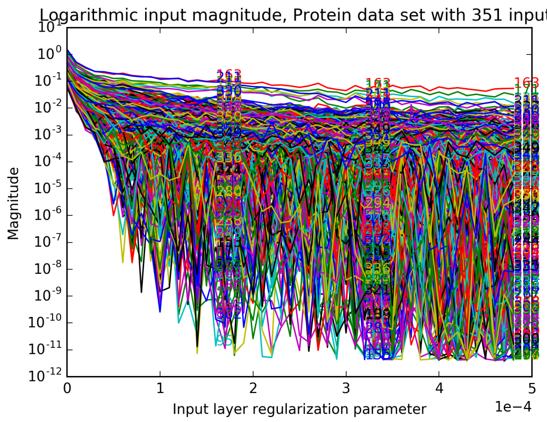


Figure 19: The summed logarithmic magnitudes of the weights associated with input nodes during training of the PacVsN data set with 351 input variables.

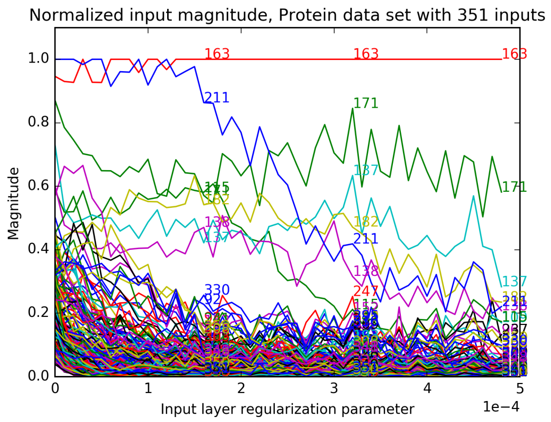


Figure 20: The summed and normalized magnitudes of the weights associated with input nodes during training of the PacVsN data set with 351 input variables.

4.3.2 Features

Representations of the magnitude of weights associated with each input node can be seen in figures (19, 20). The top 48 ranking input variables (arbitrary cut off point) using both methods can be seen in tables (6a, 6b). For comparison, the top 48 input variables of two different ranking methods have been included in tables (6c, 6d). To validate the rankings, four additional training sessions were performed, using a different subset of input variables each time. First, the top 10 inputs from table 6a were used and produced the errors of figure 21. Next, the bottom 10 inputs of the full ranking of the data set, using method (a), were used and produced the errors of figure 22. The top 10 inputs of table 6c were tested, as seen in figure 23, to see if this ranking was closest to the truth as assumed. Finally, as a control to all previous tests, a training was performed with the first 10 input variables as they appeared in the data set, and the errors can be seen in figure 24.

163	211	115	182	171	137	138	330	92	308	93	241
316	295	259	152	237	149	99	96	120	193	248	144
22	209	109	265	129	122	145	4	303	205	8	83
110	147	261	208	252	169	128	312	1	141	247	290

(a) Ranking of input node magnitude at optimal regularization.

163	171	211	137	182	115	138	152	259	330	237	241
295	96	308	93	92	209	1	193	109	149	248	316
147	120	145	205	4	83	265	312	98	110	99	303
348	247	23	101	129	290	22	253	272	341	16	169

(b) Ranking of total area of normalized input node magnitude.

163	129	303	96	290	138	8	295	262	117	77	184
349	181	252	288	323	98	318	231	227	246	120	339
312	329	202	110	348	248	345	115	270	205	294	330
307	1	239	101	147	341	72	49	299	152	86	125

(c) Externally sourced ranking (by Mattias Ohlsson) using a sophisticated method outside the scope of this study.

163	137	171	115	138	211	182	149	237	96	290	110
272	330	259	241	93	308	86	205	146	1	219	316
109	209	4	145	271	328	309	285	97	129	152	98
295	101	181	190	192	22	303	168	195	29	120	16

(d) Externally sourced ranking (by Mattias Ohlsson) using the removal of one input variable at a time and comparing the relative difference in validation error.

Table 6: Four methods of ranking the input variables of the PacVsN data set with 351 input variables. The table is read from left to right and from top to bottom. The last two methods of ranking were provided by Mattias Ohlsson.

Compared tables	Overlap (out of 48)	Mean distance of overlap
6a vs. 6b	40	5.98
6a vs. 6c	21	14.60
6a vs. 6d	34	7.47
6b vs. 6c	19	16.40
6b vs. 6d	30	9.57
6c vs. 6d	18	17.70

Table 7: Metric of the relative differences between the top 48 input variables from the four methods of ranking the PacVsN data set.

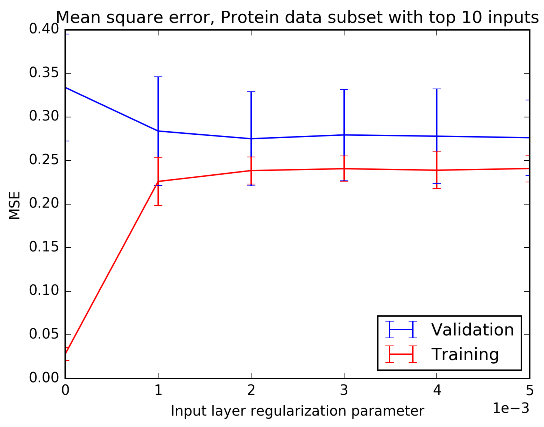


Figure 21: The mean square error of training and validation of the PacVsN data set using the top 10 input variables of table 6a.

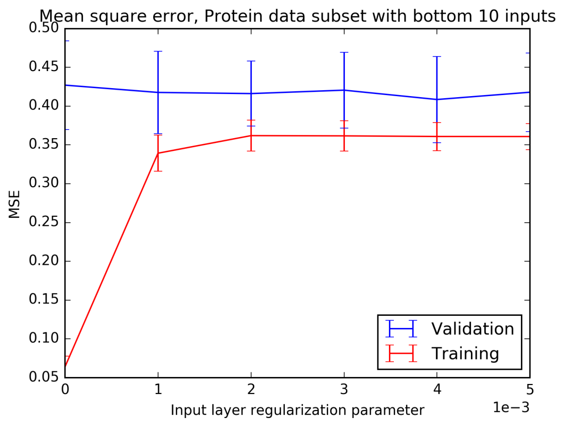


Figure 22: The mean square error of training and validation of the PacVsN data set using the bottom 10 input variables of table 6a.

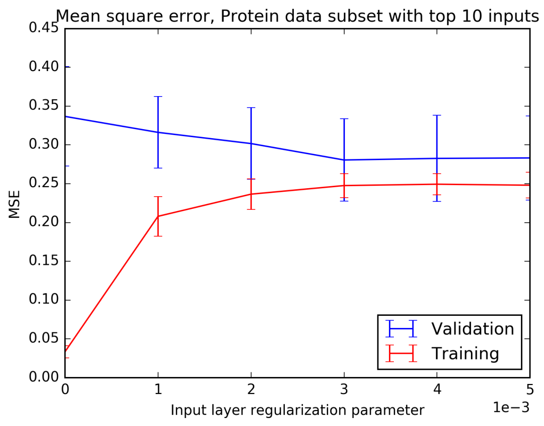


Figure 23: The mean square error of training and validation of the PacVsN data set using the top 10 input variables of table 6c.

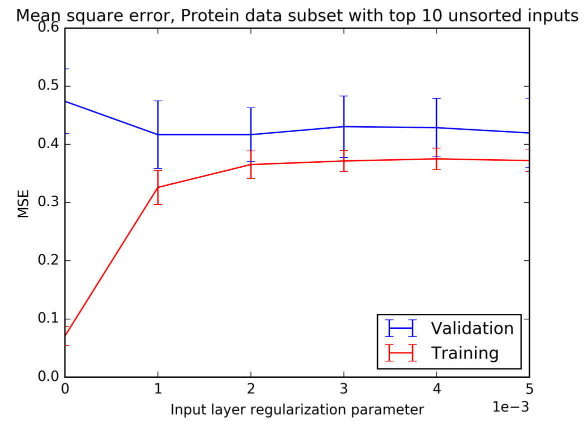


Figure 24: The mean square error of training and validation of the PacVsN data set using the first 10 input variables as they appear in the data set.

5 Discussion

5.1 Justifications for using hyperbolic tangent over the logistic function

The only reason for using the hyperbolic tangent over the logistic function in this study, was the convenience of the student being more familiar with the former function. For neural network tasks, using MSE as the loss function, and where the only use of the output signal is for classification, there is not a big difference in performance between the functions. When using the hyperbolic tangent, the output of the network is in the range [-1,1], and the optimal labels for classes are +1 and -1, but other than that, the curve profile of the output is very similar to the logistic function, as seen in figure 2.

5.2 Interpreting results

The ranking of input variables in the Pima data set seems consistent with the conventional view of diabetes, with blood glucose concentration, variable 2, being on top of the list and "Triceps skin fold thickness", variable 4, being on the bottom of the list. The only contrasting variable in the two methods of ranking was age, variable 8, which was ranked very high for low values of the regularization parameter. Perhaps this can be explained by the variable being correlated with the positive label but not being a good general classifier for both labels. The training using only the top 4 variables of table 3a (figure 11), confirms that the features selected contain enough information to give the same validation error as the full set.

For the generated data sets, regularization quickly reduced the weights associated with extraneous input variables to zero. The ranking was reasonable for both sets and no extra validation was needed as the error was clearly lower in the set without extraneous inputs. Worth noting is that the ranking didn't change much while increasing the regularization parameter. The extraneous input variables were ranked the lowest even with no regularization.

In the training of the PacVsN data set, a reasonable subset of most important input variables was found. The validation error when training using only the top 10 input variables (table 6a) was considerably lower at optimal regularization (figure 21). The validation error using the top 10 input variables of the outside sourced ranking (table 6c) was slightly better (figure 23), but that is to be expected since this ranking was the result of a sophisticated feature selection algorithm.

It is important to understand that even if the features selected by two different methods are completely different, they could still contain the same amount of information. If two different variables contain the same amount of information about the classification, one of them could be ranked highly while the other is neglected.

5.3 On proper validation of feature selection

In this study every feature selection was performed on and evaluated using the same data set. There is a non trivial concern that features selected are specific to the data set and would not be a general solution to the problem that the data is representing. This is similar to the issue of overtraining, as they are problems that arise from incomplete information. To fully validate a method of feature selection, multiple data sets representing the same problem would have to be used, for cross validation of the method itself.

5.4 Evaluating LASSO as a method of preventing overtraining

LASSO regularization is a very effective methods of preventing overtraining in applications using multi layer perceptrons. The only downside might be that finding the optimal regularization parameter requires some trial and error.

5.5 Evaluating LASSO as a method of ranking input variables

The greatest benefit of using LASSO as a feature selection tool would be that it is very fast in comparison to most other methods. Judging by the current results, the features found are at least somewhat close to the truth in terms of importance. If nothing else, the method has proven to be very efficient in filtering out less relevant input variables.

5.6 Discussing the methods of ranking, other possibilities

The methods of ranking proposed in this paper can both be interpreted as the relative resilience of weights under the constraint of weight decay. Assuming that the loss function gradient is greater for weights of higher importance, the resilience is directly related to the importance of the weight. The second method proposed in this paper, of integrating the weight magnitudes as a function of the regularization parameter, can be interpreted as the average of the weight magnitudes, or the resilience of the weights "over time". An example of another method that could be explored is comparing the relative difference in weight magnitudes between no regularization and optimal regularization.

5.7 Comparisons to other methods of ranking input variables

Two additional ranking lists for the PacVsN data set were provided by Mattias Ohlsson, that were produced using methods beyond the scope of this study. The second one of them was using a method of removing one input variable at a time and comparing the relative difference in validation error. These rankings were used to compare and validate the results attained. A metric of the overlap between the top 48 variables of the rankings can be seen in table 7. When tested in training sessions using the top 10 input variables from each ranking, the optimal validation errors were very similar (figures 21, 23).

References

[1] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.

[2] G. Cybenko. Approximation by superpositions of a sigmoidal function, 1989.

[3] Isabelle Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[4] S.O. Haykin. *Neural Networks and Learning Machines.* Pearson Education, 2011.

[5] https://commons.wikimedia.org/wiki/User:Chabacano. Overfitting.svg, 2008. [Online; accessed 07-May-2017].

[6] https://en.wikipedia.org/wiki/User:Cburnett. Artificial neural network.svg, 2006. [Online; accessed 13-May-2017].

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[8] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017-03-27].

[9] M. Lichman. UCI machine learning repository, 2013.

[10] Mattias Ohlsson. Lecture notes, 2017.

[11] Martin Riedmiller. Rprop - description and implementation details, 1994.

[12] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[13] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.