Project 2:
Attack of the cancer

Morten T. Berg, Henrik H. Carlsen and Jonas Rønning

September 2021

Abstract

In this project we have used a simple neural network to fit a linear function as well as to categorize breast cancer data. We have tested different activation functions; the identity, Sigmoid, RELU and leaky-RELU, number of hidden nodes and values of the hyper-parameter. The error obtained when fitting the neural network on the continuous function was comparable to the linear regression even for networks with only one hidden nodes. The lowest MSE was 0.014 and it was obtained using leaky-RELU as the activation function. For the categorisation problem the neural networks test accuracy was remarkable stable to the networks architecture. The best accuracy of the networks was 98%, obtained by using RELU and leaky-RELU. This was slightly better than the 94% accuracy obtained by a standard logistical regression.

1 Introduction

Machine learning and neural networks as a tool in science have gained more and more popularity over the last couple of years, with applications stretching from economics to high energy physics. The strengths of machine learning is that it can do calculations efficiently, allowing us to handle more data than have previously been possible [1].

In this project we have tested the performance of a simple "homemade" neural network against logistic regression, for determining if a breast tumor is malignant or benign, and against linear regression methods for fitting a continuous function. We have tested how the networks behave if we tune different parameters and change the activation functions. The parameters in the network and the logistic/linear regression models are optimised using a stochastic gradient decent method.

The theory and methods we use are summarised in section 2, in addition to some of the most central algorithms like back-propagation. The results are presented in section 3 before they are discussed in 4. The project is concluded in section 5. Feedback to the project can be found in section 6.

2 Theory and Methods

This section contains the theory and methods that are relevant for this project. The subsections on Linear regression and Resampling methods are summaries from the report for project 1 [2].

2.1 Linear regression

The linear regression is a simple method for fitting a continuous function. It boils down to assuming that there is a linear relationship between the input given by a design matrix X and the output y. I.e we have the relation [3]

$$y = \beta X, \tag{1}$$

and the problem is to finding the parameters β that minimises a cost function $C(\beta)$. Two populare choices of cost functions are

$$C_{OLS}(\beta) = \frac{1}{2n}\sum(y - X\beta), \tag{2}$$

$$C_{Ridge}(\beta) = \frac{1}{2n}\sum(y - X\beta) + \lambda\sum\beta^2. \tag{3}$$

These defines the OLS and Ridge regression respectively. For these choices of cost functions there exists analytical expressions for the optimal parameters, namely [3]

$$\beta_{OLS} = (X^T X)^{-1} X^T y, \tag{4}$$

$$\beta_{Ridge} = (X^T X + \lambda I)^{-1} X^T y. \tag{5}$$

From this we see that the parameter $\lambda > 0$ is a way of guaranteeing the existence of a inverse.

2.2 Resampling methods

A resampeling method is a way of extrapolating the underlying statistics from a limited data set. A way of doing this is the bootstrap method where one creates a new data set by picking random data points from the original data set. One picks as many data points as there is points in the original sett. The new set is different from the original since we allows the random sample to pic the same datapoint multiple times. This resampling is then done multiple times and one uses the new datasets to calculate averages to find the best values of the property one is interested in [1].

2.3 Regularisation terms

Regularisation terms are terms which are often included in the cost function of certain regression methods to counteract overfitting.[4] The variable λ is referred to as a hyper-parameter and its value is of paramount importance. If λ is too

large then it can lead to underfitting. However, if it is too small it can lead to the regularisation term being of little use and overfitting can be a real danger. The two most common regularisation terms are the l_1 and l_2 regularisation terms which are respectively equation 6 and 7

$$l_1(\lambda) = \lambda \sum_i |\beta_i| \tag{6}$$

$$l_2(\lambda) = \lambda \sum_i \beta_i^2 \tag{7}$$

Here β_i are all the different weights attributed to the data. Moreover the

One of the most common ways of finding the hyper-parameter is to do a grid search, and this is the method which we will utilize most commonly.

2.4 Logistic Regression

The linear regression method is quite successfully when trying to fit data where the output is continuous as in project 1 [2], when the coefficients of the Franke function were learned from its function values. In classification problems however the output is discrete or categorical. Logistic regression deals with binary output much better and is a so called "soft classifier". Soft classifiers outputs the probability of a data point, x_i, being in a category k. In the case of logic regression the probability of a data point, x_i being in a category $y_i = \{0, 1\}$ is given by the sigmoid function8. The sigmoid function has output values between $[0, 1]$, where as the linear regression model uses a linear function that can have values in $[-\infty, \infty]$ which is why the sigmoid is preferred in classification problems.

$$P(t) = \frac{e^t}{1 + e^t} \ . \tag{8}$$

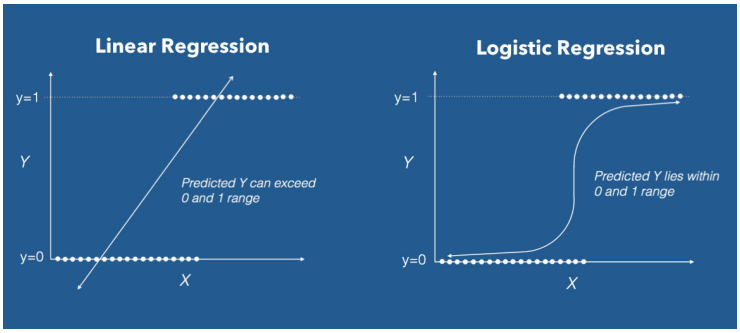

Figure 1: Here we see a comparison of how linear and logistic regression tries to fit a set of data with values of either 0 or 1. Figure from [5].

The model for the general case of $1, 2, ...k...K$ categories is described by Hastie[1] and is based on the log of the ratio between correct and incorrect classification of a data point, x, being modeled as a linear combination of parameters β:

$$log\frac{P(G=1|X=x)}{P(G=K|X=x)} = \beta_{10} + \beta_1^T x \ , \tag{9}$$

$$log\frac{P(G=2|X=x)}{P(G=K|X=x)} = \beta_{20} + \beta_2^T x \ , \tag{10}$$

$$\vdots$$

$$log\frac{P(G=K-1|X=x)}{P(G=K|X=x)} = \beta_{(K-1)0} + \beta_{K-1}^T x \ . \tag{11}$$

These $K-1$ logit transformations carries the constraint that all the probabilities sum to one and stay in the domain $[0, 1]$. The choice of category, K, used as denominator is arbitrary. This can be rewritten by taking the exponential on both sides and solving for $P(G = k|X = x)$, we get:

$$P(G=k|X=x) = \frac{exp(\beta_{k0} + \beta_k^T x)}{1 + \sum_{l=1}^{K-1} exp(\beta_{l0} + \beta_l^T x)} \ , \ k = 1, ..., K-1 \ , \tag{12}$$

$$P(G=K|X=x) = \frac{1}{1 + \sum_{l=1}^{K-1} exp(\beta_{l0} + \beta_l^T x)} \ . \tag{13}$$

The output is classified to belong to the category with the largest probability. For the case of only two categories; $K = 2$, we we switch to a more compact notation [6]; For a data point $\vec{x_i}$ the probability of i belonging to a category $y_i = \{0, 1\}$ is given by the expressions:

$$P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}_i^T \boldsymbol{\theta}}} \ , \tag{14}$$

$$P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}) \ = 1 - \sigma(\boldsymbol{w}_i^T \boldsymbol{w}). \tag{15}$$

Here $\theta = \boldsymbol{w}$ are the weights which are optimized from the data set.The sigmoid function is used on the data in order to give the probability:,$\sigma(\boldsymbol{w}_i^T \boldsymbol{w})$.

What remains is to define a cost function which will be done by using the Maximum Likelihood Estimation (MLE): From a data set, $D = (y_i, \boldsymbol{x}_i)$, where the data can be classified as $y_i = \{0, 1\}$, we draw data points independently. The definitions of the likelihood of observing data under the model and the log likelihood is then given by[6]:

$$P(D|\boldsymbol{w}) = \prod_{i=1}^{n} [\sigma(\boldsymbol{w}_i^T \boldsymbol{w})]^{y_i} [1 - \sigma(\boldsymbol{w}_i^T \boldsymbol{w})]^{1-yi} \ , \tag{16}$$

$$l(\boldsymbol{w}) = \sum_{i=1}^{n} y_i log[\sigma(\boldsymbol{w}_i^T \boldsymbol{w})] + (1 - y_i) log[1 - \sigma(\boldsymbol{w}_i^T \boldsymbol{w})] \ , \tag{17}$$

respectively. The parameters that would give the best fit would then be the parameters that maximizes $l(\boldsymbol{w})$. Since we want the optimization of the cost function to be a minimization problem the cost function becomes:

$$C(\boldsymbol{w}) = -l(\boldsymbol{w}) \ . \tag{18}$$

By utilizing the fact that $\partial_z \sigma(s) = \sigma(s)[1 - \sigma(s)]$ we find that the problem of minimizing $C(\boldsymbol{w})$ becomes:

$$\mathbf{0} = \nabla C(\boldsymbol{w}) = \sum_{i=1}^{n} [\sigma(\boldsymbol{w}_i^T \boldsymbol{w}) - y_i] \boldsymbol{x}_i \ . \tag{19}$$

Which can be solved by methods such as Stochastic Gradient Decent as there is no closed form solution[6].

2.5 Stochastic Gradient Decent

Almost all machine learning algorithm's as well as linear and logistic regression is in reality a minimization problem where we try to find a model that minimizes a multidimensional cost function $C(\beta)$. In general there are few exact analytical solution for the parameters β that minimises this function. So alternatively we have to us numerical methods to find the minimum of $C(\beta)$ with regard to β [7]. One standard method for finding the minimum is the Newton-Raphson method. This is an iterative method where one start with a guess, β_0, and then finds the minimum by iterating [7]

$$\beta_n = \beta_{n-1} - H^{-1}(\beta_{n-1}) g(\beta_{n-1}). \tag{20}$$

Here $H(\beta)$ and $g(\beta)$ are respectively the Hessian and the gradient of the cost function. The Newton-Raphson method has some short comings. First of all it require us to find the inverse of the Hessian. This is something we want to avoid if we have a lot of variables, since it is computationally costly. An other problem is that it might converge to a local minima and not the global. The first of these problems we can solve by replacing the Hessian matrix with a learning rate, η, giving us the gradient decet method [7]

$$\beta_n = \beta_{n-1} - \eta g(\beta_{n-1}). \tag{21}$$

Since the negative of the gradient points in the direction of steepest decent this method basically consist of taking baby steps to to the minimum, much like how we would find the lowest point of a hill wearing blind folds. There are however some problems with the gradient decent. Among other it will in general converge to a local minimum, and not the global minimum that we want. In addition the gradient might be computationally expensive. Both these problems can is delt with by the stochastic gradient decent (SGD). Assuming the cost function can be written as a sum over the data points as [7]

$$C(\beta) = \sum_i c(\beta, x_i). \tag{22}$$

The gradient is then given as

$$g(\beta) = \sum_i \nabla_\beta c_i(\beta, x_i). \tag{23}$$

We can now reduce the computational cost and add randomness by dividing our data points into k minibaches $B_1, B_2, ..., B_k$ and only using data from one of these when calculating the gradient. The gradient decent is then replaced by

$$\beta_n = \beta_{n-1} - \eta \sum_{i \in B_l} \nabla_\beta c_i(\beta, x_i). \tag{24}$$

The minibach, l, is randomly drawn from an uniform distribution. Typically one will sett the number of times one iterate over the number of minibaches, k. An iteration over the minibaches is called an epoch. The randomnes added in the SGD makes it possible for the method to find the global minimum of the function. Also it is less computational intensive compared to the gradient decent, because we only use part of the data to calculate the gradient.

The methods ability to locate the minimum is higly dependent on the learningrate η [7]. If it is to big one will "jump" over the minimum, while if it is to small it will require a lot of iterations before it converges. There is no reason to keep the learning rate fixed, and we can let it get smaller for each iteration so that we take smaller steps when we (hopefully) have gotten closer to the minimum. If we let e be the current epoch, m the number of baches and $i = 1...m-1$ is the iteration over baches we can let the learningrate vary as [7]

$$\eta = \frac{t_1}{t_2 + e \cdot m + i}, \tag{25}$$

where t_1 and t_2 is two positive constants.

2.6 Neural networks

What is a neural network? It is inspired by the brain with its many neurons. Neurons send electrical signals through axons which are received by the dendrites of other neurons and added together in the cell body, and then, if the signal is strong enough to pass the axon hillock it will be sent on through the axon to be picked up by a new dendrite[8].

The artificial neural network functions by taking n inputs ,x, multiplying a weight,w and adding a bias, b, before summing them together in a "node". This gives an activation value, z, defined as:

$$z = \sum_{i=1}^{n} x_i \cdot w_i + b_i \tag{26}$$

$$a = f(z) \tag{27}$$

The activation value is then fed into an activation function f where we name the output of this function, $a = f(z)$. There are many possible choices of activation

functions, we will concern ourself with the sigmoid28, $\sigma(x)$, RELU29, $R(x)$, and LeakyRELU-function 30, $LR(x)$. Below they are listed:

$$\sigma(x) = \frac{1}{1 - e^{-x}} \ , \tag{28}$$

$$R(x) = \begin{cases} 0 & \text{if x} \leq 0 \\ x & \text{if x} > 0 \ , \end{cases} \tag{29}$$

$$LR(x) = \begin{cases} 0.01x & \text{if x} \leq 0 \\ x & \text{if x} > 0 \ . \end{cases} \tag{30}$$

The output,a scalar, is transmitted from the "node" to be interpreted as the networks output. The process describes the most basic for of a neural network; The Single Perceptron Model(SPM)[9]. Such a network is shown in figure 2.

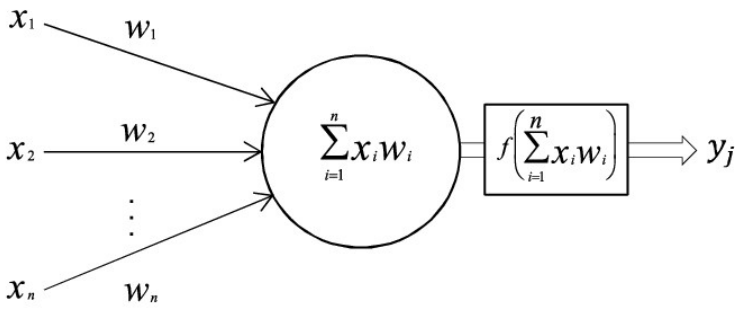

Figure 2: Single perceptron model. Illustration from M. Hjort-Jensen lecture notes[10].

2.6.1 Deep neural networks and the feed forward neural network

The previous section described the simplest neural network, the SPM. It only has one layer, one node and one output. More complicated, deep neural networks, with multiple layers and multiple nodes per layer, can be created. The layers between the input layer and the output layer are referred to as "hidden layers". The process of propagating a signal is the same as for the SPM, however here the output from the neurons will be fed to other neurons repeating the process a number of times based on the number of neurons and layers. Each neuron has a set of weights attributed to their respective inputs as well as a bias which is added when calculating the activation value of the neuron.

There are many ways of structuring deep neural nets, one of them is the Feed Forward Neural Network(FFNN). An example of the feed forward neural network can be seen in figure 3. The FFNN is characterised by having the outputs from one layer being "fed" to the next layers in a chain like fashion. All

the neurons of layer n send their output to the layer $n+1$ where they are used to create the activation values for the respective neurons[10].

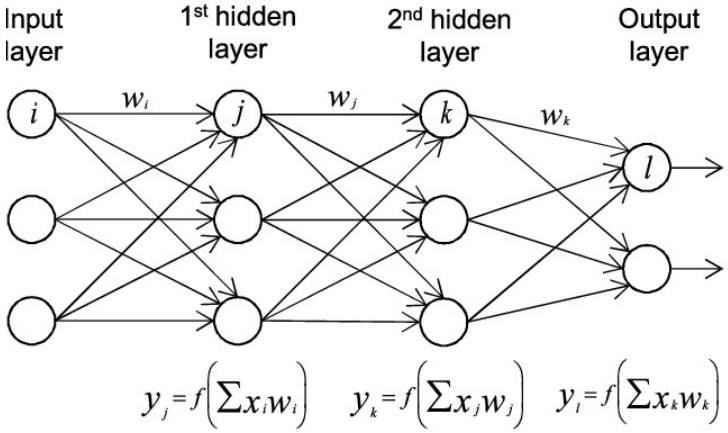

Figure 3: An example of a FFNN with three nodes in the input layer, two hidden layers and two nodes in the output layer. This can easily be upscaled. Figure from M. Hjort-Jensen lecture notes[10].

2.7 Back propagation

The back propagation algorithm[11] is a way to train the weights in a neural network without brute force calculation of the gradients. The algorithm exploits the layered nature of the network to calculate errors between the expected outputs and the outputs which are propagated forward by the network. After the errors have been calculated they are propagated backwards in the network, hence the name. In order to use the algorithm the expressions for derivative of the cost function, E, and the activation function, σ, must be known. What cost-function one uses are problem dependent. For a classification problem one will often use the logistic cost function 18 while one would use the MSE for a regression problem. In addition it is common to add regularisation terms. The algorithm, as described by Mehta et al.[6], is as follows:

1. **Input layer activation:** The activations of all j input nodes, a_j^1 should be calculated.

2. **Feedforward** Compute the weighted input, z^l, and activations, a^l, for each subsequent layer, l, starting from the first one all the way to the last layer L by using:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) , \tag{31}$$

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l , \tag{32}$$

where w_{jk} is the weight for the output from the $k-th$ node of layer $l-1$ to the $j-th$ node in the $l-th$ layer. The receiving node is denoted b_j^l. σ is the activation function.

3. **Output layer error** Calculate the error in the output layer, Δ_j^L using the derivatives of the cost function, E, and the derivative of the activation function, σ':

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial a_j^l}\sigma'(z_j^l) \ . \tag{33}$$

4. **Backpropagate the error** The error is propagated backwards in the network using quantities from layer $l+1$:

$$\Delta_j^l = \left(\sum_k \Delta_k^{l+1} w_{kj}^{l+1}\right)\sigma'(z_j^l) \ . \tag{34}$$

5. **Calculate gradient** The remaining step is to calculate the desired gradients $\frac{\partial E}{\partial b_j^l}$ and $\frac{\partial E}{\partial w_{jk}^l}$ by using the following equations:

$$\frac{\partial E}{\partial b_j^l} = \Delta_j^l \ , \tag{35}$$

$$\frac{\partial E}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1} \ . \tag{36}$$

Because the desired gradients are so many, one for each parameter of the network, and one for each step in the gradient decent, the algorithm needs to be and is very computationally efficient [6]. It uses only one forward pass and one backward pass through the network to calculate desired gradients.

2.8 Vanishing or exploding gradients

A problem occurs in many layered neural networks; Gradients can vanish or explode. This problem is especially prominent in Recurrent neural networks that try to find dependencies in large temporal sequences of data. The problem is linked to the weights between the hidden layers in the temporal loop. We take a look at a network of several layers with a single neuron in each layer in order to better illustrate the problem: Assume all the weights, $w = w_1, w_2, ...,$ are equal. By using equation 35 multiple times on ends up with an equation for the backpropagation in the system[6]:

$$\Delta_j^1 = \Delta_j^L \Pi_{j=0}^{L-1} w\sigma'(z_j) = \Delta_j^L w^L \sigma_c'^L \ . \tag{37}$$

Here the output layer is denoted by L, so this gives the error in the first layer Δ_l^1. If the derivatives if the activation function, $\sigma'(z_j)$, has sufficiently small fluctuations it can be approximated by some constant σ_c'. If that is the case then there are two possible outcomes for systems of many layers.

1. If $w\sigma_c' < 1$ then the gradients vanish.

2. If $w\sigma_c > 1$ the gradients explode.

Using the sigmoid function28 as activation functions often result in vanishing gradients,using the RELU function29 one may encounter the dying RELU problem[12] were some nodes will be killed by vanishing gradients with no way of being activated again. The LeakyRELU function30 proposes an answer to this; its derivative is 0.01 when the input to the function is smaller than zero. This gives the node the possibility of resurgence, although small.

3 Results

3.1 Linear regression

The results of the Ridge regresion for different parameters are shown in figure 4 - 6. The parameters that are varied are the learningrate η, the hyperparameter λ, the number of epochs, and the bach size. In Figure 4 and 5 the standard stochastic gradient methode is used and the parameter that varies between the subfigures are the size of the minibach and the number of epochs respectively. In figure 6 the method of changing the learning rate is used and it is t_1 and t_2 that is varied between the subplots. The OLS behaves for the most part similarly to the first column and is therefore not included in this report. The interested reader is refereed to the material available on the github folder.

3.2 Regression with neural network

Figure 7 to 10 shows the MSE and R2 scores obtained using a network with a single hidden layer. In figure 7 the number of hidden nodes are varied, while in the rest it is the hyperparameter that is of interest and the number of nodes is fixed. In the stochasticgradient the number of epochs and minibaches to 100 and 8, and the learningrate is kept fixed.

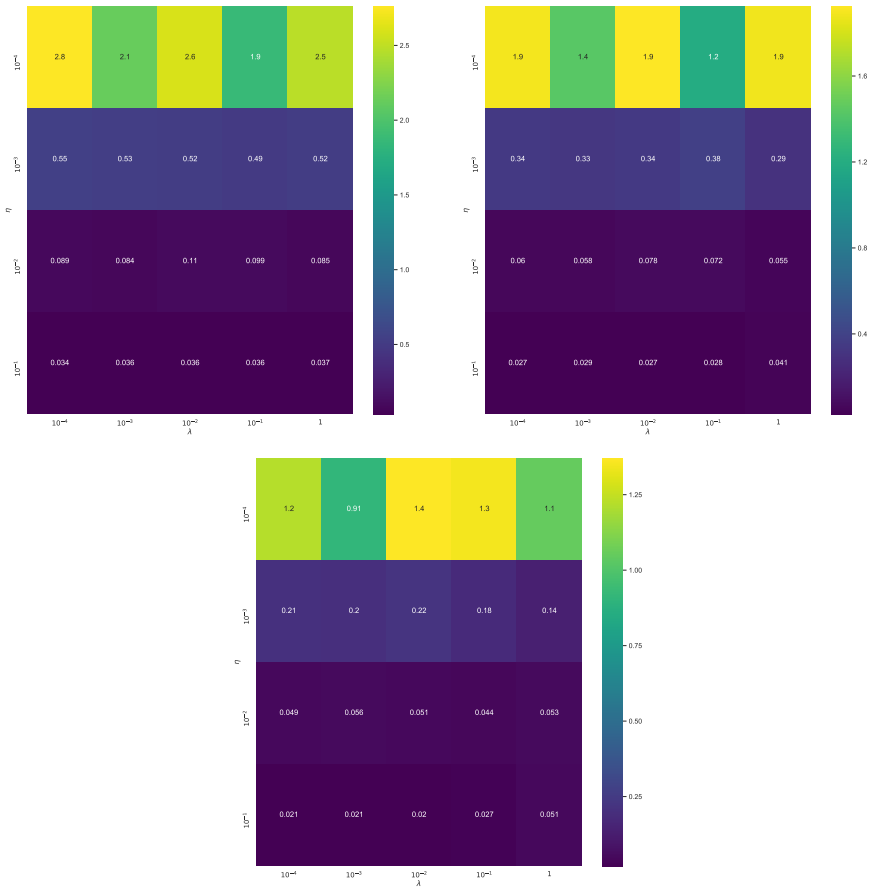

Figure 4: MSE obtained for ridge regression as a function of η and λ. The learningrate is fixed and the number of epochs is 100. From the top left the number of minibaches are 4, 8 and 16.

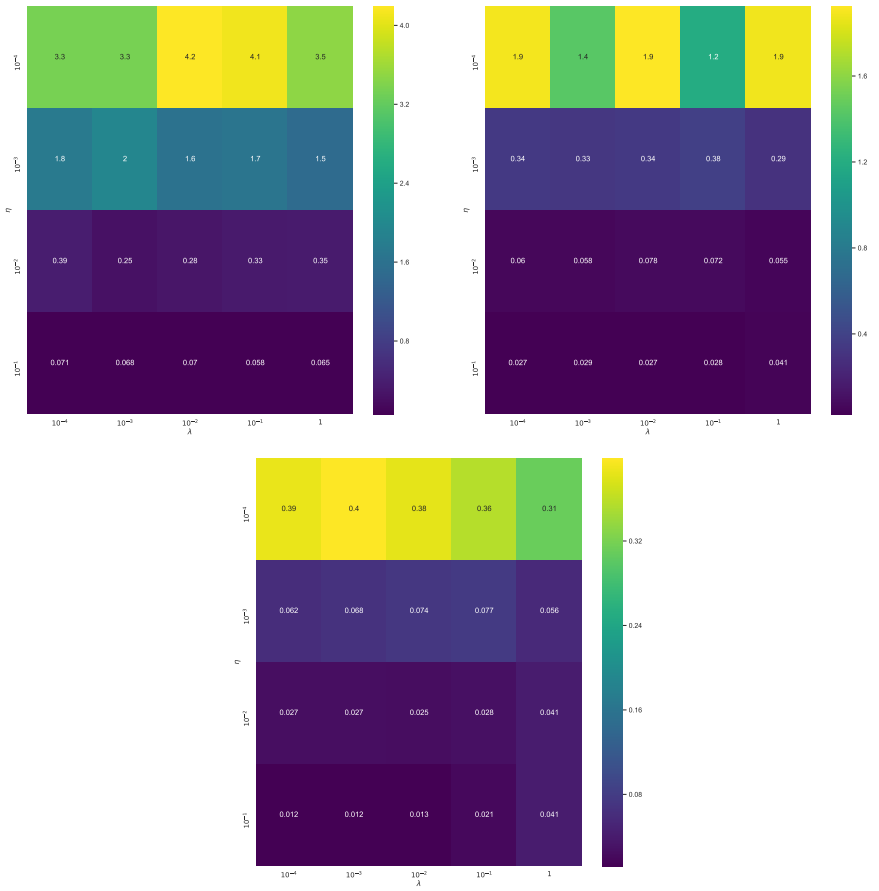

Figure 5: MSE obtained for ridge regression as a function of η and λ. The learningrate is fixed and the number of minibaches is 8. From the top left the number of epochs are 10, 100 and 1000.

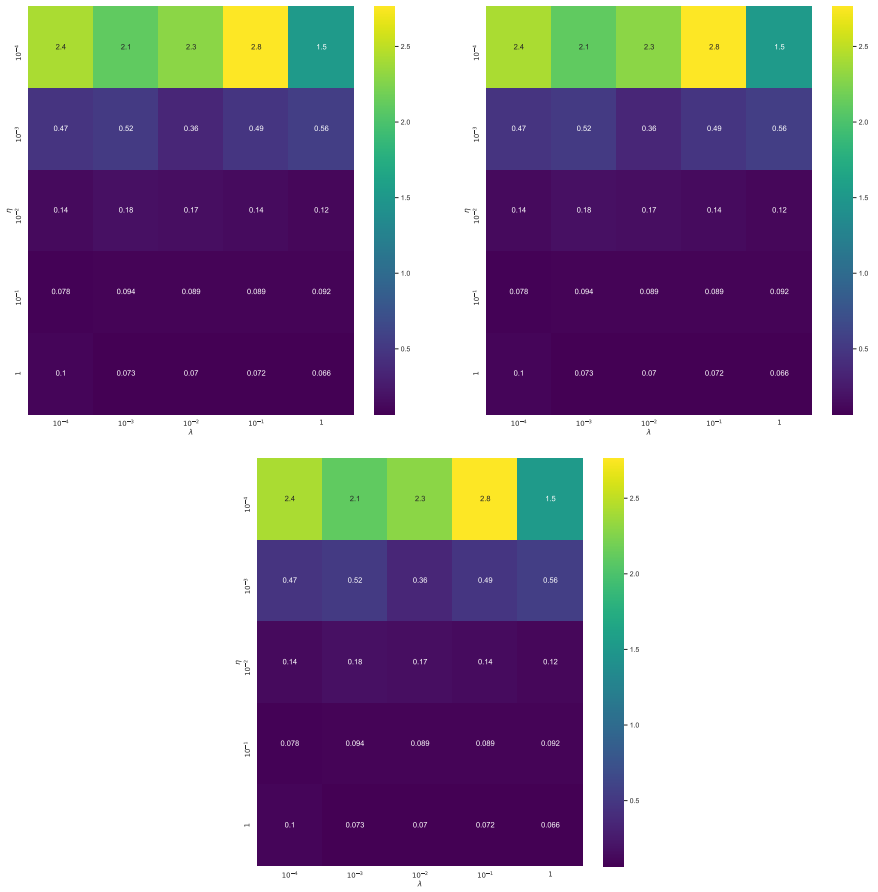

Figure 6: MSE obtained for ridge regression as a function of η and λ. The number of minibaches and epochs is 8 and 100 respectively. The learning rate is varied as described in eq. (25). From the top left t_1, t_2 is $1, 1/\eta$, $2, 2/\eta$ and $5, 5/\eta$.

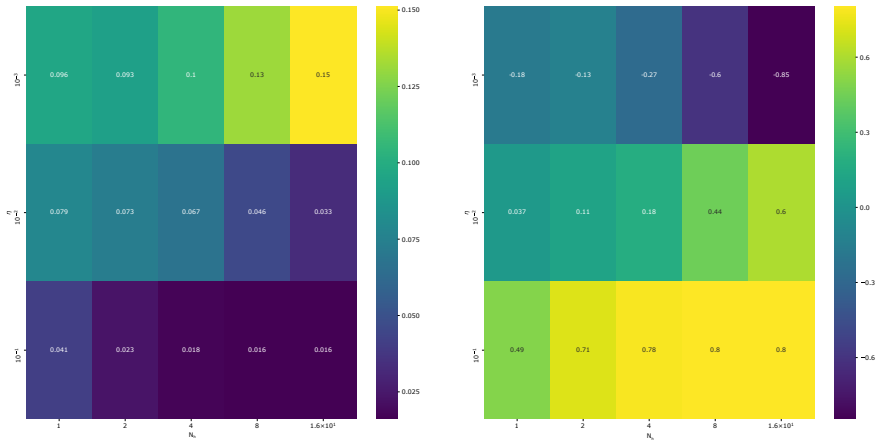

Figure 7: MSE (left) and R2-score (Right) for a single hidden layer network as a function of the number of hidden nodes and the learningrate η. The activationfunction is the Sigmoidfunction and the hyperparameter is set to 0.

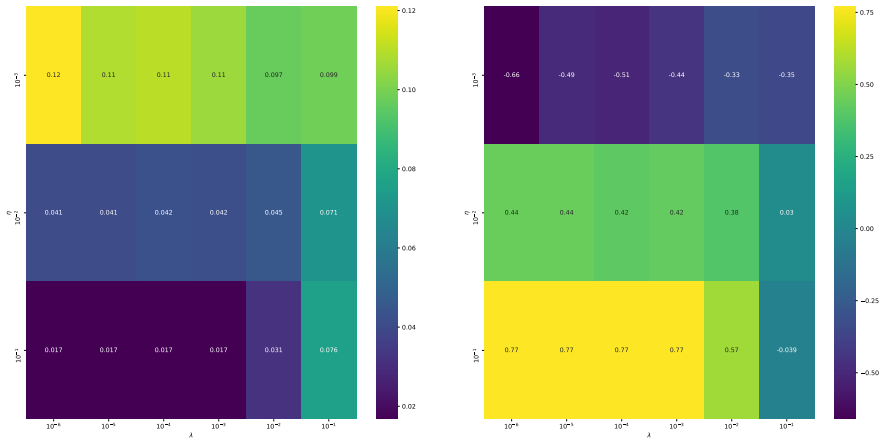

Figure 8: MSE (left) and R2-score (Right) for a single hidden layer network as a function of the hyperparameter λ and the learningrate η. The activationfunction is the Sigmoid function and the number of hidden nodes is 8.

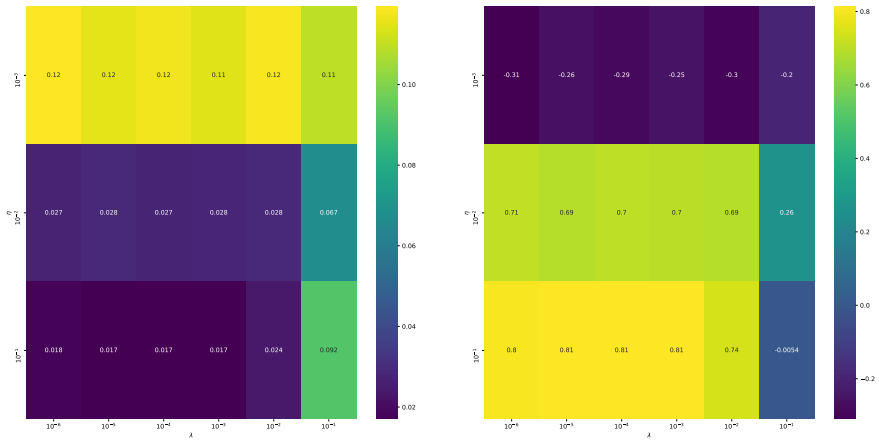

Figure 9: MSE (left) and R2-score (Right) for a single hidden layer network as a function of the hyperparameter λ and the learningrate η. The activation function is the RELU-function and the number of hidden nodes is 8.

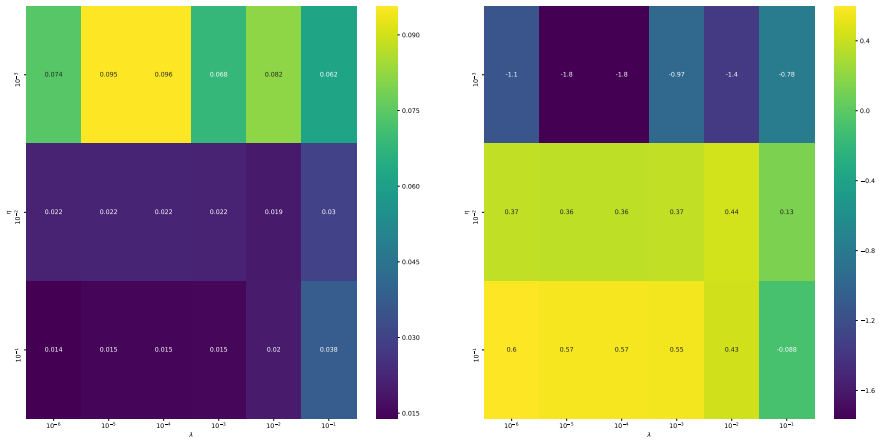

Figure 10: MSE (left) and R2-score (Right) for a single hidden layer network as a function of the hyperparameter λ and the learningrate η. The activation function is the leaky RELU-function and the number of hidden nodes is 8.

3.3 Classification with neural network

In this chapter we test the neural networks performance on classification. Figure 11 to 14 shows how a single hidden layer network classifies breast cancer data. In figure 16 and 17 we show the optimisation process of some networks of different dept as a function of the number of epochs.

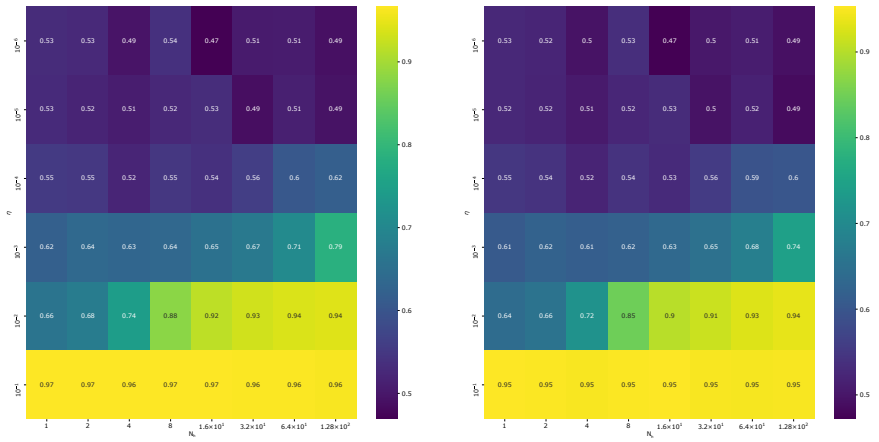

Figure 11: The accuracy of our single hidden layer network as a function of hidden nodes and the learning rate. The left is the training data and to the right we have the test data. We have used Sigmoid as the acctivation function.

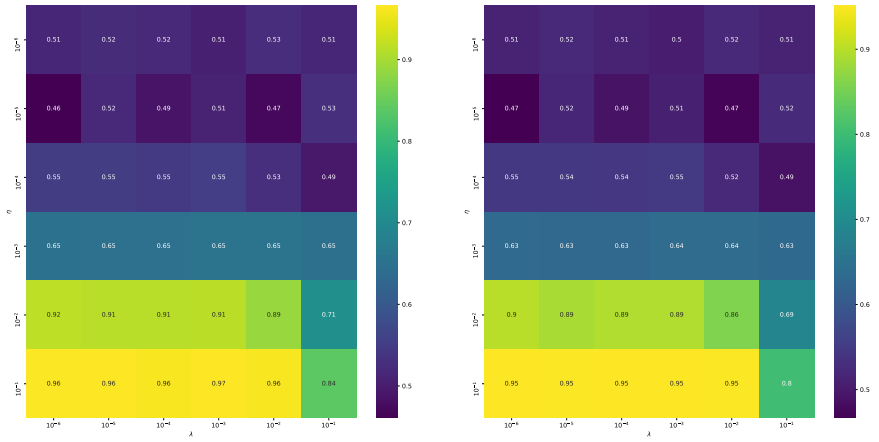

Figure 12: The accuracy of our single hidden layer network as a function of the hyperparameter and the learningrate. Traning data is to the left and test to the right. Sigmoid is used as the activation function.

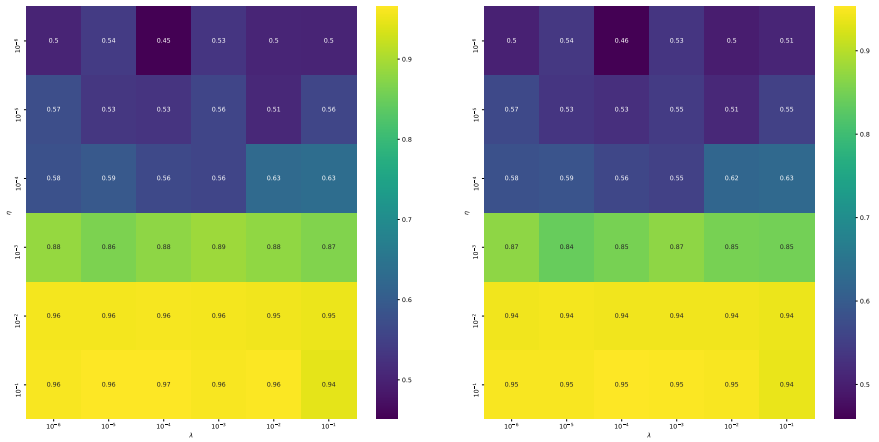

Figure 13: The accuracy of our single hidden layer network as a function of the hyperparameter and the learningrate. Traning data is to the left and test to the right. RELU is used as the activation function.

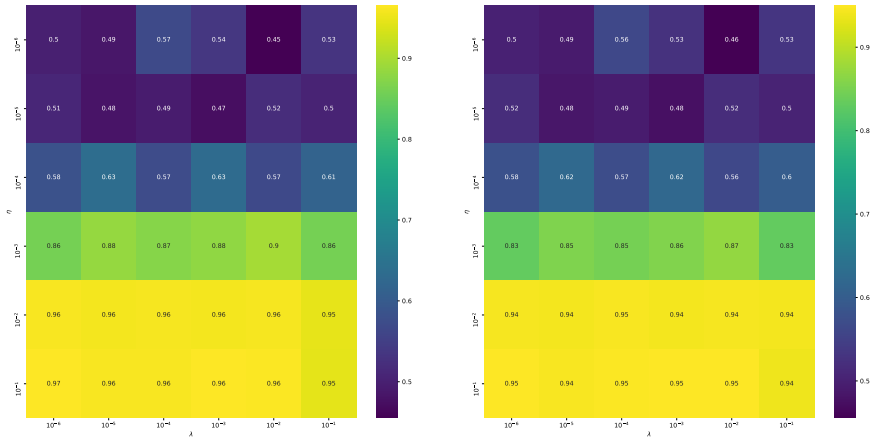

Figure 14: The accuracy of our single hidden layer network as a function of the hyperparameter and the learningrate. Traning data is to the left and test to the right. Leaky-RELU is used as the activation function.

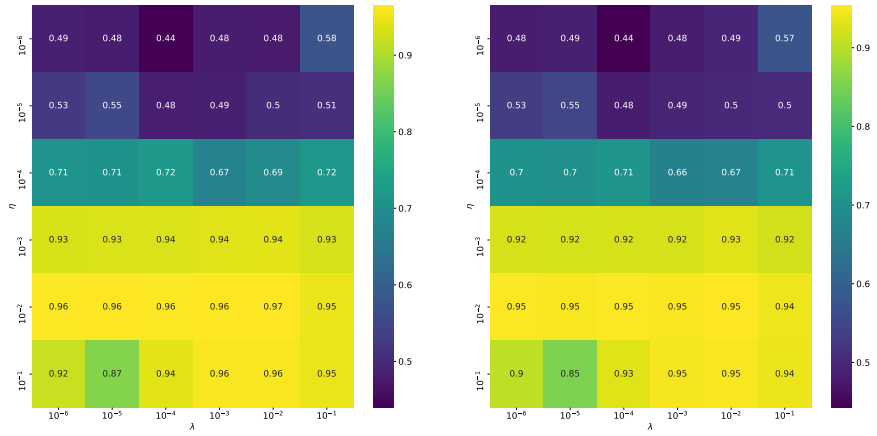

Figure 15: The accuracy of our single hidden layer network as a function of the hyperparameter and the learningrate. Traning data is to the left and test to the right. The Identity is used as the activation function.

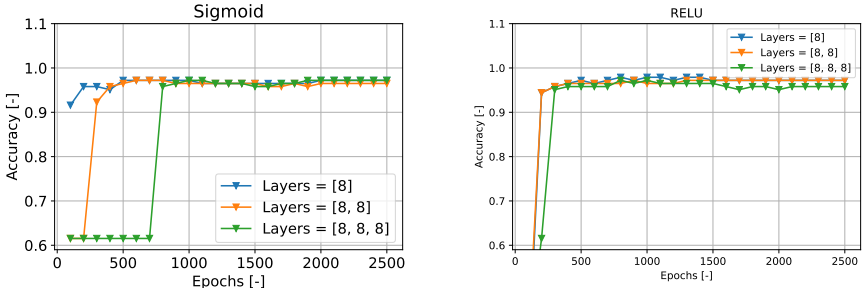

Figure 16: The accuracy obtained by the networks of different dept as a function of epochs. The activation function is Sigmoid(left) and RELU(Right). The value of the hyperparameter $\lambda = 0$ and the learning rate is $\eta = 0.1$.

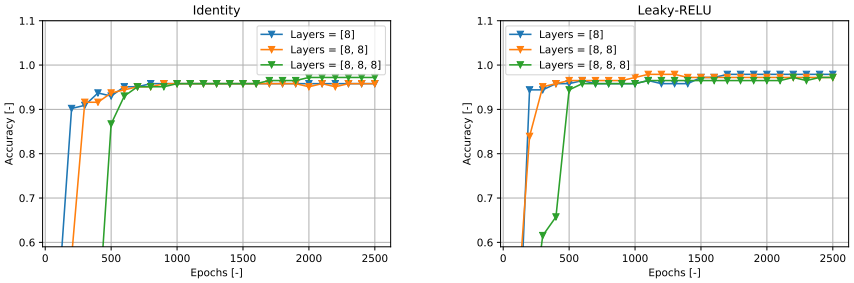

Figure 17: The accuracy obtained by the networks of different dept as a function of epochs. The Identity(left) and leaky-RELU(Right). The value of the hyperparameter $\lambda = 0$ and the learning rate is $\eta = 0.01$(left) and 0.05 (right).

3.4 Logistic regression

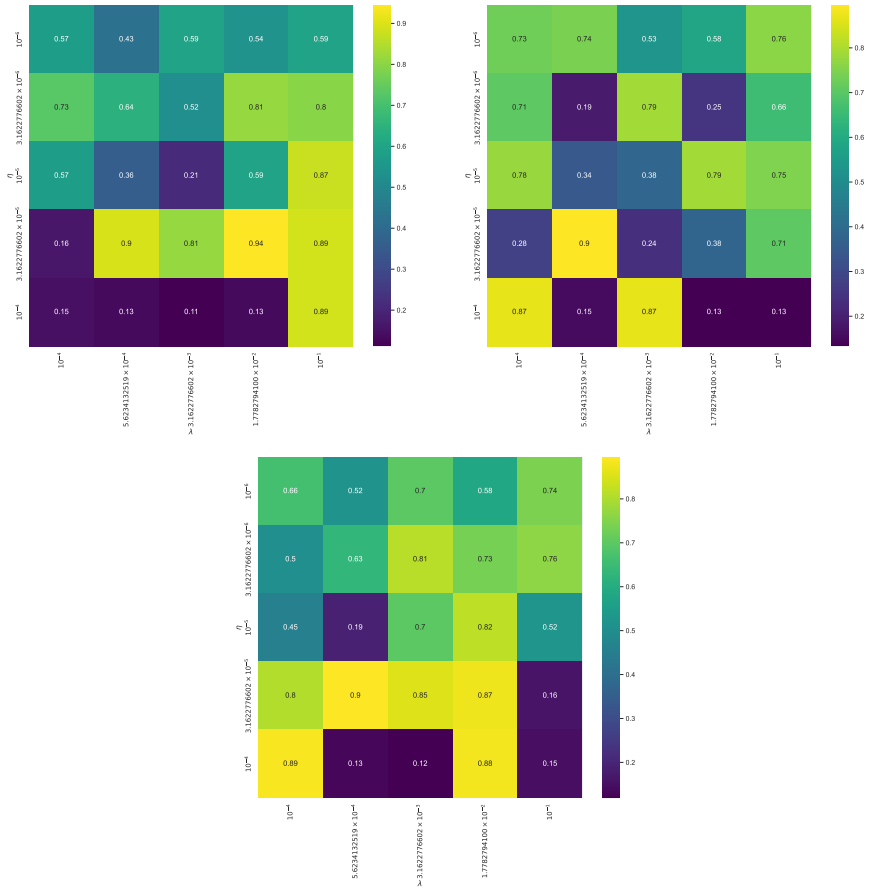

Figure 18: Accuracy of the stochastic gradient descent method with a l^2 regularization term for logistic regression plotted as a function of λ and η. The model was trained over 10, 100 and 1000 epochs respectively and 4 mini-batches per epoch. With a decaying learning-rate as described in 25.

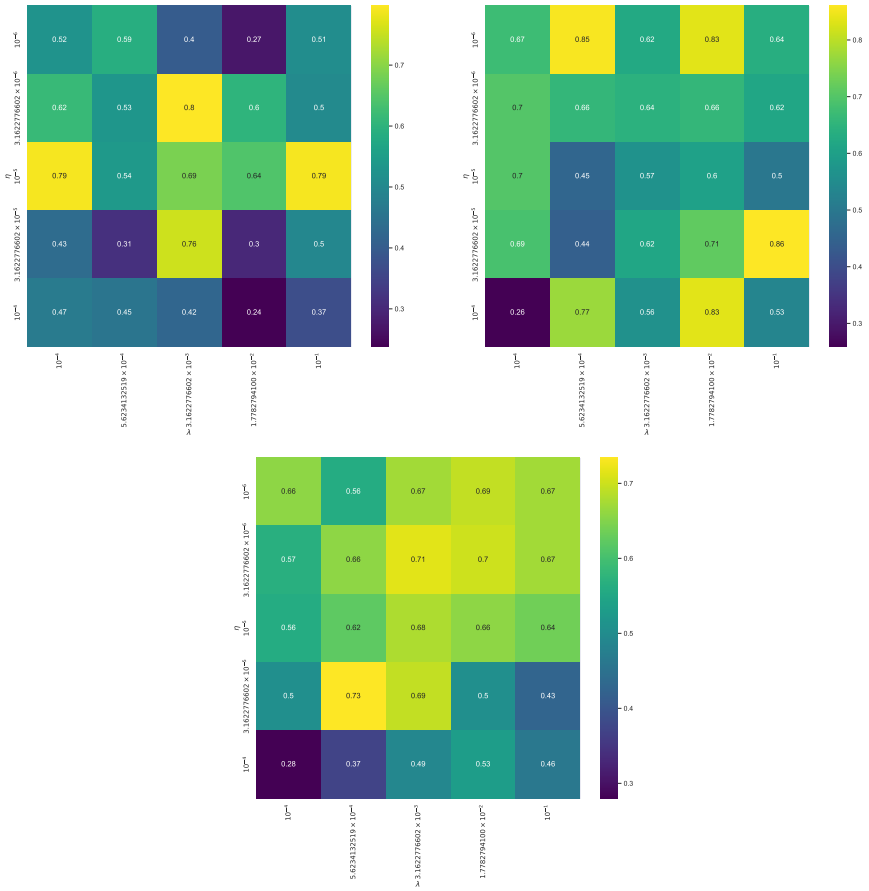

Figure 19: Accuracy of the stochastic gradient descent method with a l^2 regularization term for logistic regression plotted as a function of λ and η. The model was trained over 10, 100 and 1000 epochs respectively and 8 mini-batches per epoch. With a decaying learning-rate as described in 25.

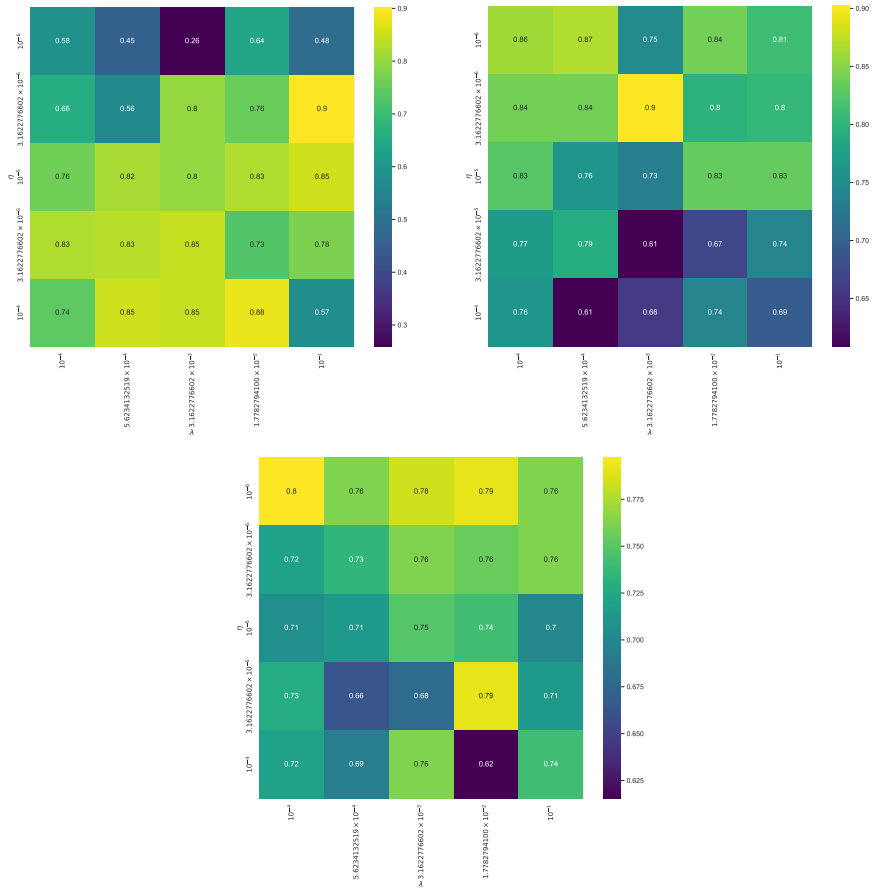

Figure 20: Accuracy of the stochastic gradient descent method with a l^2 regularization term for logistic regression plotted as a function of λ and η. The model was trained over 10, 100 and 1000 epochs respectively and 16 mini-batches per epoch. With a decaying learning-rate as described in 25.

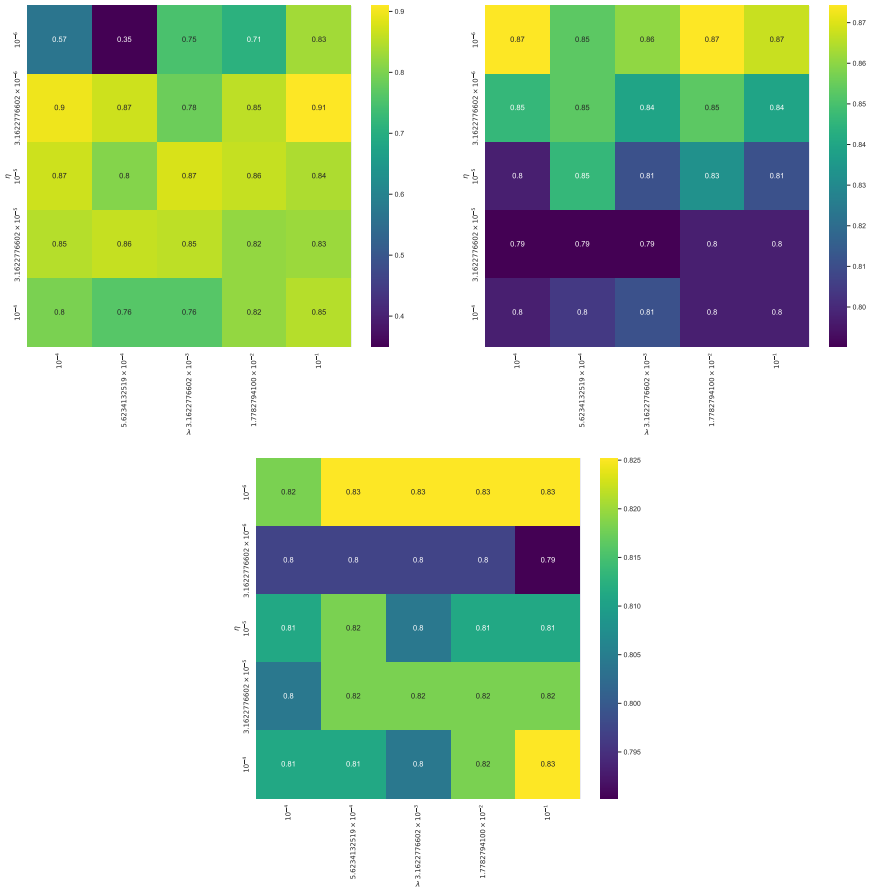

Figure 21: Accuracy of the stochastic gradient descent method with a l^2 regularization term for logistic regression plotted as a function of λ and η. The model was trained over 10, 100 and 1000 epochs respectively and324 mini-batches per epoch. With a decaying learning-rate as described in 25.

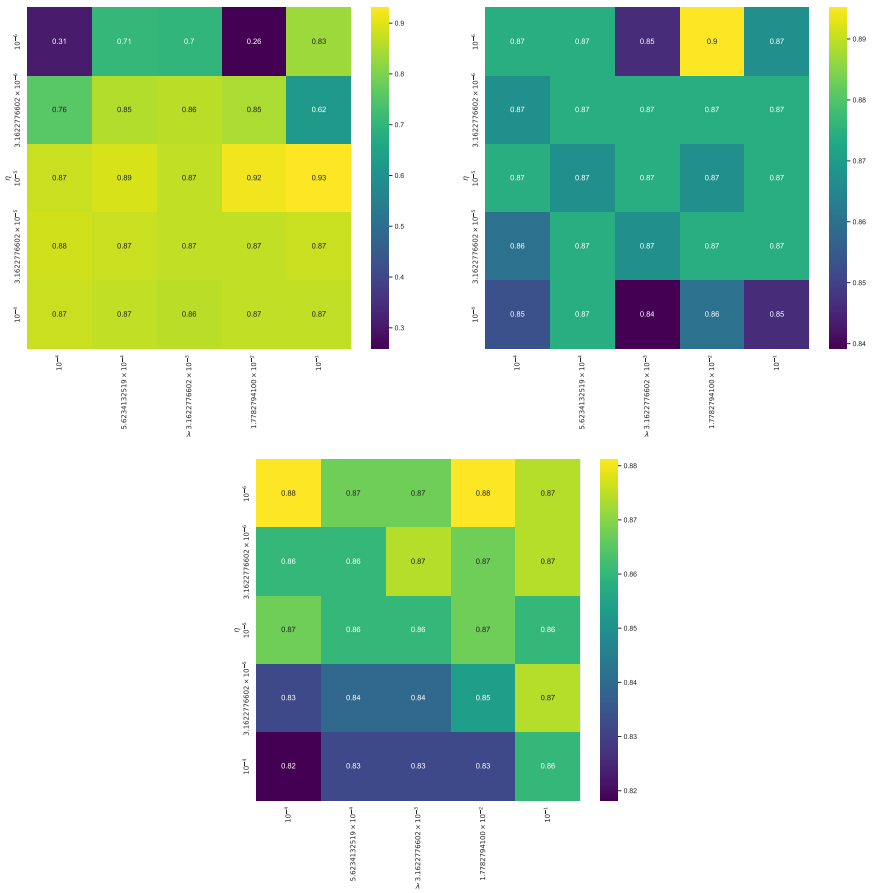

Figure 22: Accuracy of the stochastic gradient descent method with a l^2 regularization term for logistic regression plotted as a function of λ and η. The model was trained over 10, 100 and 1000 epochs respectively and 64 mini-batches per epoch. With a decaying learning-rate as described in 25.

3.4.1 Logistic regression using sklearn

For the logistic regression using sklearn we got a accuracy of 96%.

4 Discussion

4.1 Regression

For the linear regression we see that the convergence of the MSG is strongly dependent on the learning rate as we discussed in the theory section. From figure 4 - 6 we see that if the learning rate is to low the MSE will take to long

to converge. This is quite clear from figure 5 where the number of epochs are varied. Here one see that the MSE becomes better when the number of epochs is increased, but for low learning rate, the top rows, the convergence is slow and the MSE is still higher for 1000 epochs than what we got with a larger learning rate for 10 epochs. If the learning rate is to large the MSE will diverge, which we saw when we tried to sett the learning rate to $\eta = 1$. If we however let η vary as described in the theory section the sgd becomes more stable to the initial rate and we have therefore included $\eta = 1$ in figure 6. The reason it is more stable is of course because we only do a few jumps with the learning rate that caused the divergence before it is small enough to not cause problems anymore. We also see that the error obtained when varying the learning rate is larger than if we pick an optimal fixed learning rate. This is because the learning rate becomes to small to quick. We could address this problem by running with the fixed learning rate for some number of epochs and then letting the learning rate decrees, but we did not implement this in this project.

The Number of minibaches has some effect on the convergence on the sgd. From figure 4 we see that the more minibaches the faster the MSG converges, how much faster depends on the hyperparameter λ and the learning rate η. From our tests 8 seems to work well, and we therefore use it as standard.

4.2 Regression with neural network

The neural network we use for the regression consists of two input nodes where we feed in the position, one hidden layer with a varying number of nodes, and an output node. We tested using a deeper network, but it had no remarkably better performance. For the regression we see that the MSE obtained using a neural network to aprocimate the franke function is comparable with the one we obtain using the Ridge regression with a fifth order polynomial, MSE around 0.03 for 100 epochs, even when we are using only one hidden node, MSE around 0.04. The linear regression tunes 21 parameters, while the nettwork tunes 5. This confirms that the network is quite good at approximating the function. How well the network approximate the function depends on the learning rate η, hyperparameter λ, number of hidden nodes and the activation function. From figure 7 we see that when the number of hidden nodes is increased the MSE drops. If the learning rate is optimal, then the dependence on the number of nodes drops quickly and the difference between 4 nodes and 16 is almost neglectible. Note that when the number of nodes is increases one has to minimise with respect to more parameters and the method might require more iterations. The dependence on the hyperparameter is also quite strong as depicted in figure 8 to 10. In general for the regression it seams like a low hyperparameter gives the best result, which is simmilar to what we observed for the linear regression. In the same figures we can also see what happens if we consider different learning rates. We see that using RELU and Sigmoid gives quite similar results. RELU converges a bit faster with lower learning rate. Leaky RELU is however very different. First of it gives a much faster convergence for low learning rate. But it is the R2 score that is curious. The leaky RELU gives low MSE combined

with a low R2 score. This is hard to interpret.

4.3 Classification with neural network

The preformance of the neural network to the breast cancer data was a bit surprising. When varying the number of hidden nodes in a single hidden layer network all the networks obtained the same maximum test accuracy for the learning rate $\eta = 0.1$. This can be seen in figure 11. The maximum test accuracy in the grid search was 95% and it is regardless of what activation function we use, as seen in figure 12 - 15. Even the Identity function achieves this accuracy. The values of the parameters for which the different activation functions obtains the maximum test accuracy does however vary a bit. Note that one have to be careful when setting the weights and biases in this network, since if they are initialised to large the network will diverge. For the most promising parameter we have run the simulation longer and with deeper networks as seen in figure 16 and 17. Here one see that the accuracy dos not improve when the network becomes deeper. However the deeper networks requires more epochs to get a high accuracy since they contain more parameters and we therfore sugests that these should be avoided. When we do the longer runs we find a maximal accuracy of $97,9\%$ wich is obtained by using the RELU and leaky-RELU functions. Note that the learning rate used for the leaky-RELU is $\eta = 0.05$ and not $\eta = 0.1$ which gives the best results for a single hidden layer network. The reason for this is that the later gives divergences when applied to the deeper networks.

4.4 Classification with logistic regression

The data for the logistic regression is shown in figure 18 to 22, where the figures are displaying the results obtained when using 4, 8, 16, 32 and 64 minibaches. For a low number of minibaches, 4, we see that the model is producing a wide range of accuracies as seen in fig. 18. Some of them is very bad, down to 10% which is quite useless, but if one negates the output these will also produce great results. It gets sligtly better if we increase the number of epochs. For some values of the parameters the model is good also for the small number of baches. For 8 minibaches, fig. 19 we got some good result for some particular values of the parameters, but mostly the results are comparable, or worst, than flipping a coin. The best model for 1000 epochs is worst than the ones with 100. One likely explanation is that the gradient method jumps away from the true minimum and since the learning rate is becoming smaller at each step it do not manage to jump back to it. This unfortunate problem can be solved by checking the cost function doing the run and save the parameters that gives the smallest error. For 16 minibaches, fig. 20, the model starts to be more stable to the parameters. Most of the accuracies are better than flipping the famous coin. Again we see that the best accuracies are gotten by having a small number of epochs. For 32 minibaches, fig. 21, the models are starting to get good. Again we see that a low number of epochs gives the single best accuracy, but a higher

epoch is on average more stable. The last one is for 64 minibaches, fig. 22. For a few iterations we have some parameters that gives very poor fit, but it is her we fin the most accurate fits. When the number of iterations become large the accuracy is more stable, but the peek is smaller.

Moreover the accuracy when using sklearns logistic regression was 96%.

Compared to the neural network classifier the logistic regression did well. It manage to have an compareble maximal accuracy. The draw back is that it seems to be more unstable to the parameters than the neural network. The logistic regression was extremely picky on the lerningrates, and it had a smaller window of convergence than the networks, as one can see on the y-axis.

5 Conclusion

The Wheel of Time turns, and subjects come and pass, leaving memories that become legend. Legend fades to myth, and even myth is long forgotten when the subject that gave it birth comes again. In one subject, called fys-stk by some, a project was concluded [13]. In this project we saw that neural networks was well suited for fitting continuous functions and categorising data. For fitting the continuous data the MSE obtained with the neural network was comparable to the error obtained by the linear regression methods. Moreover the logistic regression with stochastic gradient descent provided many good results for different values of the parameters. Our best fit was only 2% worse than the fit provided by sklearn. This leads us to conclude that our logistic regression was quite good. Comparing it to the classifications done by our feed forward neural network which was both stable and good for a number of different activation functions we can conclude that using either one of them would provide us with good results, although the 2% that got wrong results on their biopsy might disagree.

6 Feedback

This project was okay. It was less stressful than the first one, but that might be due to how we attacked the project.

References

[1] J. Friedman T. Hastie, R. Tibshirani. *The elements of statistical learning : Data mining, inference, and prediction*. Springer Science Business Media, 2009.

[2] Jonas Rønning Morten T. Berg, Henrik H. Carlsen. Project 1. https://github.com/MortenTryti/MachineLearning/blob/main/Project%201/Report.pdf, 2021.

[3] Morten Hjorth-Jensen. Week 35: From ordinary linear regression to ridge and lasso regression. https://compphysics.github.io/MachineLearning/doc/pub/week35/html/week35.html.

[4] Anuja Nagpal. L1 and L2 Regularization Methods. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c, 2017.

[5] Sukriti Macker. Why "regression" is present in logistic regression? https://medium.com/analytics-vidhya/why-regression-is-present-in-logistic-regression-8795755feb54.

[6] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810:1–124, 2019.

[7] Morten Hjorth-Jensen. Week 39: Optimization and gradient methods. https://compphysics.github.io/MachineLearning/doc/pub/week39/html/week39.htmll.

[8] Gerald Ehrenstein and Harold Lecar. The mechanism of signal transmission in nerve axons. *Annual review of biophysics and bioengineering*, 1(1):347–366, 1972.

[9] Jaswinder Singh and Rajdeep Banerjee. A study on single and multi-layer perceptron neural network. pages 35–40, 2019.

[10] Morten Hjorth-Jensen. Week 40: From stochastic gradient descent to neural networks. https://compphysics.github.io/MachineLearning/doc/pub/week40/html/week40.htmll.

[11] David E Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.

[12] Zheng Hu, Jiaojiao Zhang, and Yun Ge. Handling vanishing gradient problem using artificial derivative. *IEEE Access*, 9:22371–22377, 2021.

[13] R. Jordan. *The Eye of The World*. Tor Books, 1990.