Stockholms universitet

A Comparison of Logistic Regression and Neural Networks for Binary Classification Problems

Anna Basetti

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2019:6
Matematisk statistik
Juni 2019

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2019:6**
http://www.math.su.se

A Comparison of Logistic Regression and Neural Networks for Binary Classification Problems

Anna Basetti*

June 2019

Abstract

Binary classification is the task of classifying data points into either one of two groups, based on their coordinates (explaining variables). Logistic regression and neural networks are two of the most widely used classification models. The former has its roots in traditional statistics, while the latter originates from the younger field of machine learning. Neural networks have seen a renaissance during the past years, and have been surrounded by a great deal of hype and an aura of mystery, while logistic regression kept being regarded as a more robust, down-to-earth method. The scope of this thesis is to dissect the two models, highlighting their similarities and differences, both theoretically and practically.

The first part of this work presents the theory behind logistic regression and neural networks. A close look at the structure of a vanilla neural network reveals that such a model is a rather natural expansion of logistic regression: in fact, a vanilla neural network with the sigmoid as its activation and output function and the cross-entropy error function is exactly a logistic regression in the hidden nodes, as well as each hidden node is a logistic regression in the explanatory variables. However, the flexibility gained through the addition of nonlinearity makes the neural network a more powerful method when the data at hand is not linearly separable.

The second part of this work consists of three experiments, performed on three different simulated data sets exhibiting different geometrical properties. For each experiment, one logistic regression and three neural networks with varying numbers of nodes in the hidden layer were fitted, and the resulting decision boundaries were plotted. The performance of each model was then tested on 50 simulated validation data sets. The results coincided with the expected performances, proving that neural networks do serve as a flexible and powerful extension of logistic regression, in areas where predictive power, rather than interpretability, is the priority.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: anna.basetti@outlook.com. Supervisor: Ola Hssjer.

Acknowledgements

I would like to thank my supervisor Ola Hössjer for his readily available help, valuable feedback and guidance through the writing of this thesis.

Contents

1 Introduction

1.1 Background

Classification tasks consist in classifying data points into two or more groups, based on a number of explanatory variables [1]. As much of our behavior is dictated by context, classification problems lie at the core of decision making. For instance, the course of action adopted by a doctor dealing with a patient will depend on the patient's diagnosis. Whether we decide to show private information to someone is dependent on us classifying their face as one we know — *recognizing* it. Recognition is but a classification problem, as the names "digit recognition" and "facial recognition" suggest.

Classification models stem from a demand for automation of decision making processes. There exists a variety of such models. Two of the most widely used ones are logistic regression (LR) and artificial neural networks (ANN). The former originates from classical statistics, while the latter developed in the fields of computer science and machine learning [2]. This study aims to present the two models, highlighting their similarities and differences, both theoretically and practically. For this purpose, we will focus on *binary* classification problems, where the task is to assign data points one of two possible groups [3].

1.2 Outline

The thesis begins in Section 2, presenting the background and theory for logistic regression and neural networks. Section 3 describes the method used for simulation of the data and modeling of the LR and ANN. The structure of the experiments and the results are presented in sections 4 and 5, respectively. The text ends with an analysis of the results and a discussion on possible improvements and expansions to this work.

2 Theory

The theory for both logistic regression and neural networks is taken from [3], [1] and [2], unless stated otherwise.

Typically, a binary classification problem includes a number p of explanatory variables, or predictors, which can be categorical or continuous, and one response variable, which is by nature categorical. This work focuses solely on continuous explanatory variables, coded as an explanatory (column) vector $\tilde{\mathbf{X}} = (X_1, X_2, \ldots, X_p)^T$. We will furthermore be restricting ourselves to $p = 2$, as one of the main purposes of this work is to provide the reader with an intuitive understanding of the similarities and differences between logistic regression and neural networks, and these are most easily

visualized in the case of a predictor plane. As for the response variable, there are two equally common approaches to coding it. One is to have a single integer valued response variable $Y = I_1$, where I_i is the indicator function for the current observation belonging to class i for $i = 1, 2$. This corresponds to the response being coded as 1 if the observation belongs to the first class, and as 0 otherwise. Another approach is to code the response variable as a column vector $\mathbf{Y} = (I_1, I_2)^T$, corresponding to $(1, 0)$ in the case of class 1, and $(0, 1)$ in the case of class 2. The latter is most common in multi-class classification problems. Though the two approaches are equivalent for binary classification tasks, the former is slimmer and easier to read, and so it will be used throughout this thesis.

The goal of a classification model is to, given a set of labelled data points (i.e. a set of observations for which both the explanatory vector $\mathbf{x}$ and the response variable y are known), fit a model which can predict with reasonable accuracy the response vector for a new data point (i.e. a new $\mathbf{x} = \mathbf{x}^*$) drawn from the same distribution. In the field of machine learning this is known as *supervised* learning. In other words, a classification model which has been fitted to a labelled data set should be able to correctly classify a new point, provided that the point comes from the same distribution and lies within the range of values which the model was trained on.

Both logistic regression and neural networks are *discriminative models*. These models classify data in two steps. The first one consists in computing an estimated probability for each class. In binary classification, this boils down to mapping the conditional probability

$$\pi(\mathbf{x}) := P(Y = 1 \mid \mathbf{X} = \mathbf{x}) = 1 - P(Y = 0 \mid \mathbf{X} = \mathbf{x}).$$

The second step consists in setting a threshold c — so called *cutoff* — such that if

$$\pi(\mathbf{x}^*) > c$$

then the point $\mathbf{x}^*$ is assigned to the first class, otherwise to the second. The choice of cutoff is dependent on the nature of the problem: a lower cutoff will reduce the number of observations belonging to class 1 that get misclassified as class 2 ("false negatives"), but only at the price of misclassifying more observations of class 2 as class 1 ("false positives"). As this study focuses on predictive power, we will be using a cutoff of 0.5. In a different scenario, e.g. when testing for a serious illness, one might want to choose a considerably lower (or higher) cutoff.

Geometrically, both models will classify a new observation $\mathbf{x}^*$ by drawing a *decision boundary* dividing the x_1, x_2 plane into two or more regions, and then assigning a class based on which region the point $\mathbf{x}^*$ is located in. The decision boundary corresponds to the level curve $\pi(\mathbf{x}) = c$. Sections 2.1 and 2.2 cover the theory behind the methods for computing $\pi(\mathbf{x})$ employed in logistic regression and neural networks, respectively.

2.1 Logistic Regression

2.1.1 Background and Model

Logistic regression belongs to the family of linear models for classification. The model borrows its name from the logistic function, or *sigmoid*,

$$\sigma(v) = \frac{1}{1 + \exp(-v)},$$

illustrated in Figure 1. The logistic function plays an important role in many a classification model. Bayes' theorem, together with the law of total probability, gives that the conditional probability for class one given an observed value $\mathbf{x}$ can be written as

$$P(C_1 \mid \mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{x} \mid C_1)P(C_1)}{P(\mathbf{x} \mid C_1)P(C_1) + P(\mathbf{x} \mid C_2)P(C_2)},$$

where C_i denotes the observation belonging to class i. Defining v as the natural logarithm of the ratio of the two terms in the denominator

$$v := \log \frac{P(\mathbf{x} \mid C_1)P(C_1)}{P(\mathbf{x} \mid C_2)P(C_2)}$$

we find that the conditional probability for class one is precisely the image of v on the sigmoid curve:

$$P(C_1 \mid \mathbf{x}) = \pi(\mathbf{x}) = \frac{1}{1 + \exp(-v)} = \sigma(v).$$

Interesting qualities of the sigmoid are that it satisfies the symmetry property

$$\sigma(-v) = 1 - \sigma(v)$$

and that it is a *squashing* function, mapping the whole real axis into the interval $]0, 1[$. This is of course particularly useful when we want our function to output a probability, as is the case with both logistic regression and neural networks. The inverse of the sigmoid is the *logit* function

$$\text{logit}(v) = \log \frac{v}{1 - v}.$$

Given a p-dimensional input $\tilde{\mathbf{x}} = (x_1, \ldots, x_p)^T$, a logistic regression models the probability of the corresponding point belonging to the first class as

$$\pi(\tilde{\mathbf{x}}) = \frac{\exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p\right)}{1 + \exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p\right)} = \frac{\exp\left(\beta_0 + \tilde{\boldsymbol{\beta}}^T \tilde{\mathbf{x}}\right)}{1 + \exp\left(\beta_0 + \tilde{\boldsymbol{\beta}}^T \tilde{\mathbf{x}}\right)},$$

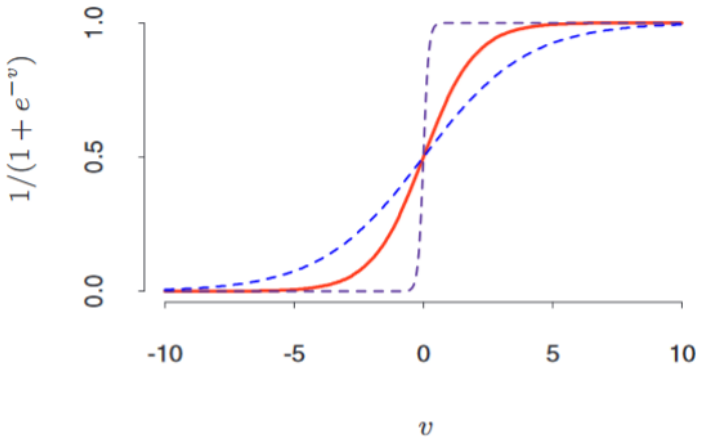

Figure 1: The logistic sigmoid function $\sigma(v)$ (red curve). Included are also $\sigma(sv)$ for $s = 0.5$ (blue curve) and $s = 10$ (purple curve). The figure is from [3].

where $\tilde{\boldsymbol{\beta}} = (\beta_1, \ldots, \beta_p)^T$ is the vector of effect parameters for the p predictors and β_0 is a bias term. By adjusting the input vector to feature a 1 in the first column and including β_0 in the parameter vector, the notation simplifies to

$$\pi(\mathbf{x}) = \frac{\exp\left(\boldsymbol{\beta}^T\mathbf{x}\right)}{1 + \exp\left(\boldsymbol{\beta}^T\mathbf{x}\right)} = \sigma(\boldsymbol{\beta}^T\mathbf{x}),$$

where $\mathbf{x} = (1, x_1, \ldots, x_p)^T$ and $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$. The model is, in other terms, a linear regression of the logit, or *log-odds*

$$\text{logit}\big(\pi(\mathbf{x})\big) = \log\frac{\pi(\mathbf{x})}{1 - \pi(\mathbf{x})} = \sigma^{-1}\big(\sigma(\boldsymbol{\beta}^T\mathbf{x})\big) = \boldsymbol{\beta}^T\mathbf{x}$$

on the explaining variables $x_1, \ldots, x_p$.

A linear model in the predictors corresponds to a linear decision boundary: for any p, the decision boundary will be a hyperplane in the p-dimensional feature space. For $p = 2$ that takes the form of a straight line in the x_1, x_2 plane. The classification of a new point is solely dependent on which side of the line it falls on.

2.1.2 Fitting Logistic Regression Models

We fit — *train* — our model on a training set of N observations with explaining vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and responses $y_1, \ldots, y_N$. The logistic regression model is fitted by maximizing the conditional likelihood of the responses given the observed values of the predictors. If $\pi_i := \pi(\mathbf{x}_i)$ is the probability of the i-th observation belonging to class 1, the responses $y_1, \ldots, y_N$ are

drawn from N independent Bernoulli distributions, with $Y_i \sim \text{Bern}(\pi_i)$. The likelihood function is then

$$\begin{aligned} L(\boldsymbol{\beta}) &= \prod_{i=1}^{N} \Big\{ I_{1,i} \cdot P(Y_i = 1 \mid \mathbf{X} = \mathbf{x}_i) + I_{2,i} \cdot P(Y_i = 0 \mid \mathbf{X} = \mathbf{x}_i) \Big\} \\ &= \prod_{i=1}^{N} \Big\{ \pi_i^{y_i} (1 - \pi_i)^{1 - y_i} \Big\}, \end{aligned}$$

where $I_{k,i}$ is the indicator of y_i belonging to the k-th class. This gives the log-likelihood

$$\begin{aligned} l(\boldsymbol{\beta}) &= \sum_{i=1}^{N} \big\{ y_i \log \pi_i + (1 - y_i) \log(1 - \pi_i) \big\} \\ &= \sum_{i=1}^{N} \big\{ y_i \cdot \boldsymbol{\beta}^T \mathbf{x}_i - \log[1 + \exp(\boldsymbol{\beta}^T \mathbf{x}_i)] \big\}. \end{aligned} \tag{2.1}$$

The log-likelihood is maximized by solving the Score equations

$$\begin{aligned} 0 = \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_0} &= \sum_{i=1}^{N} \Big\{ y_i - \frac{\exp(\boldsymbol{\beta}^T \mathbf{x}_i)}{1 + \exp(\boldsymbol{\beta}^T \mathbf{x}_i)} \Big\} = \sum_{i=1}^{N} \{ y_i - \pi_i \} \\ 0 = \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_j} &= \sum_{i=1}^{N} \Big\{ y_i x_{j,i} - x_{j,i} \frac{\exp(\boldsymbol{\beta}^T \mathbf{x}_i)}{1 + \exp(\boldsymbol{\beta}^T \mathbf{x}_i)} \Big\} = \sum_{i=1}^{N} x_{j,i} (y_i - \pi_i), \quad j = 1, \ldots, p. \end{aligned}$$

These are $p+1$ nonlinear equations in $\boldsymbol{\beta}$ which can be solved via the Newton-Raphson algorithm.

Expansion by automated model selection

Simple Logistic Regression is a purely linear model, and thus only suitable when one has reason to believe data can be linearly separated with respect to the input features. These features, however, need not always coincide with the raw explanatory variables. A nonlinear model can be achieved by performing a logistic regression on a set of *derived features*, i.e. (nonlinear) functions of the original explanatory variables. In the case of low-dimensional data following an easily intelligible pattern, such derived features may be added manually, but high-dimensional or complex data would make this option impractical. A common way to overcome this obstacle is to start with a large number of derived features and successively remove features or limiting their effect on the outcome, until an optimal model is achieved. This is precisely what the Least Absolute Shrinkage and Selection Operator, or LASSO, method seeks to do. It is part of a wider family of *shrinkage methods*, designed to slim down the initial model to an optimum. Instead

of directly maximizing the log-likelihood, a penalized version is introduced:

$$l^*(\tilde{\boldsymbol{\beta}}) = \sum_{i=1}^{N} y_i \tilde{\boldsymbol{\beta}}^T \mathbf{x}_i - \sum_{i=1}^{N} \log\left(1 + \exp(\tilde{\boldsymbol{\beta}}^T \mathbf{x}_i)\right) - \lambda \sum_{j=1}^{p} |\beta_j| \ ,$$

where the constant parameter λ controls the amount of shrinkage of the coefficients $\tilde{\boldsymbol{\beta}} = (\beta_1, \ldots, \beta_p)$. By shrinking the coefficients to be exactly 0, the LASSO is able to effectively remove features from the model and achieve an optimal model. This is a powerful and widely used method to expand the number of data distributions that logistic regression may be used for, and was presented for completeness. However, we want to highlight that this is a separate method, not belonging strictly to nor being part of logistic regression. Therefore, no shrinkage methods were used in this study, and instead only simple logistic regressions on the original explaining variables were performed.

2.2 Neural Networks

2.2.1 Background and Model

Artificial Neural Networks (ANN) are a class of machine learning algorithms inspired by the learning process adopted by biological neural networks in the animal brain. Learning in the brain is made possible by continuous adjustments to the synaptic connections between neurons. The ability to mimic this process lies at the core of the architecture of artificial neural networks. This work will be dealing with the simplest form of ANN, multilayer perceptrons with a single hidden layer, sometimes referred to as "vanilla" neural networks. The vanilla neural network is a network consisting of a number of nodes, or *neurons*, organized in one input, one hidden and one output layer. It is a *feedforward* network, meaning that information can only move from one layer to the next. Each unit in a given layer is connected to all of the units in the next, forming a bipartite graph, as illustrated in Figure 2. Each connection is assigned a weight. The set of all the weights will be the set of parameters that are being optimized during model fitting.

The hidden layer consists of a number M of neurons, each of which receives a linear combination $\alpha_{1m}x_1 + \ldots + \alpha_{pm}x_p$ of the explanatory variables plus a bias term α_{0m}, and outputs the image Z_m of the input through a nonlinear *activation function*. The activation function is usually chosen to be the sigmoid function, giving

$$Z_m = \sigma(\boldsymbol{\alpha}_m^T \mathbf{X}), \quad m = 1, \ldots, M \ ,$$

where $\boldsymbol{\alpha}_m = (\alpha_{0m}, \alpha_{1m}, \ldots, \alpha_{pm})^T$ and $\mathbf{X} = (1, X_1, \ldots, X_p)^T$. If we let T_k denote the linear combination $\beta_{1k}Z_1 + \ldots + \beta_{Mk}Z_M$ plus a bias term β_{0k}, and let $\mathbf{T} = (T_1, \ldots, T_K)^T$, we have that each of the neurons $k =$

$1, \ldots, K$ in the output layer receives the whole vector $\mathbf{T}$ and outputs the image π_k of the input through an *output function* $g_k(\mathbf{T})$. The choice of output function depends on the task at hand. For regression problems, the output layer consists of one neuron using the identity function. For binary classification problems, it suffices to have one output node outputting the conditional probability π for class 1. The most widely used function for binary classification is, again, the sigmoid. For K-class classification, the sigmoid can be extended to the *softmax* function

$$g_k(\mathbf{T}) = \text{softmax}_k(\mathbf{T}) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} ,$$

and the output layer comprises K nodes, each mapping the conditional probability π_k of class k. In summary, our model for binary classification takes the form

$$\begin{aligned} Z_m &= \sigma(\boldsymbol{\alpha}_m^T \mathbf{X}), \quad m = 1, \ldots, M, \\ \pi(\mathbf{X}) &= P(Y = 1 \mid \mathbf{X}) = \sigma(\boldsymbol{\beta}^T \mathbf{Z}), \end{aligned}$$

where $\mathbf{Z} = (1, Z_1, \ldots, Z_M)^T$ and $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_M) = (\beta_{01} - \beta_{02}, \ldots, \beta_{M1} - \beta_{M2})$, as illustrated in Figure 2.

The popularity of the sigmoid function as an activation function is a consequence of its resemblance with a step function, as shown in Figure 1. The sigmoid can be seen as a smooth, differentiable approximation to the 0-1 step function. The use of an activation function, and particularly the use of one reminiscent of the 0-1 step function, was inspired by biological neural networks: in the human brain, a neuron only activates — *fires* — when it receives an input exceeding a certain value, known as the *threshold of excitation*. If we were to use a linear activation function, the neural network would collapse into a linear model, but an ANN with as few as one hidden layer and an arbitrary squashing function (a bounded, non-constant activation function) can approximate any continuous function from one finite-dimensional space to another arbitrarily well, provided that enough nodes in the hidden layer are available [4][2]. This result is known as the Universal Approximation Theorem.

2.2.2 Fitting Vanilla Neural Networks

The aim of this section is to summarize the theory behind vanilla neural networks. Unless stated otherwise, the theory is taken from [3], [1] and [2].

2.2.3 Loss Function

In the case of p predictors, the Vanilla Neural Network described in Section 2.2.1 has $(p+1)M$ α-parameters for the connections between the input and

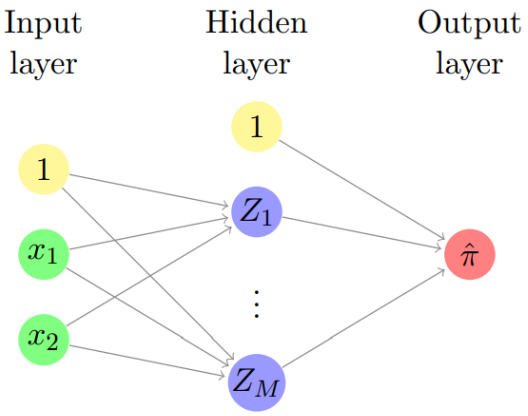

Figure 2: A vanilla neural network with M hidden units

hidden layers, and $M + 1$ β-parameters between the hidden and output layers. These parameters — *weights* — are optimized via the minimization of a *loss* or *error function* $R(\boldsymbol{\theta})$ over the complete set of weights $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$. For regression tasks, the loss function is typically chosen to be the mean squared error

$$R(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2,$$

where $\hat{y}_i$ is the predicted value of y_i. Mean squared error was also common for classification problems in the 1980s and 1990s, but was gradually replaced by the *cross-entropy* function

$$R(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{k=1}^{K} I_{k,i} \log \hat{\pi}_k(\mathbf{x}_i),$$

where $I_{k,i}$ is the indicator of y_i belonging to the k-th class, and $\hat{\pi}_k(\mathbf{x}_i)$ is the estimated conditional probability of y_i belonging to the k-th class. The switch to cross-entropy losses is believed to be one of the main factors which contributed to the improvement in performance of ANNs with sigmoid and softmax outputs [2]. Golik et al. also found, in their 2013 conference paper, that with randomly initialized weights the cross-entropy allowed to find a better local optimum than squared loss [5].

For binary classification, the cross-entropy function takes the form

$$R(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \Big\{ I_{1,i} \log \hat{P}(Y_i = 1 \mid \mathbf{X} = \mathbf{x}_i) + I_{2,i} \log \hat{P}(Y_i = 0 \mid \mathbf{X} = x_i) \Big\}$$
$$= -\sum_{i=1}^{N} \Big\{ y_i \log \hat{\pi}(\mathbf{x}_i) + (1 - y_i) \log(1 - \hat{\pi}(\mathbf{x}_i)) \Big\},$$

which is exactly the negative of (2.1). Thus, minimizing the cross-entropy loss function is equivalent to maximizing the log-likelihood of a logistic regression in the hidden units.

2.2.4 Gradient Descent

Picture a child learning how to play basketball. The child starts with a first shot which is as good as random, and then observes the outcome. Suppose the ball could not reach the basket: next time the child will shoot harder. If the second shot is too long, too high or low, too far left or right, the child will keep observing the outcomes and adjusting their technique until they figure out just how hard and in what direction they need to shoot in order to score.

This is the idea behind *gradient descent*. Gradient descent is a two-step algorithm consisting of a forward and a backward sweep, and the most common approach to minimizing the loss function in neural networks. In the forward sweep, the output is computed using the current weights, as well as the corresponding value of the loss function. The first forward sweep will use random weights, typically drawn from a uniform distribution. Then, the gradient $\Delta_{\boldsymbol{\theta}} R(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$ of the loss function with respect to all the model parameters is computed. In the backward sweep, each parameter in the model is updated. Each gradient descent update has the form

$$\beta_m^{new} = \beta_m^{old} - \eta \sum_{i=1}^{N} \frac{\partial R_i(\boldsymbol{\theta})}{\partial \beta_m}, \tag{2.2}$$

$$\alpha_{jm}^{new} = \alpha_{jm}^{old} - \eta \sum_{i=1}^{N} \frac{\partial R_i(\boldsymbol{\theta})}{\partial \alpha_{jm}}, \tag{2.3}$$

where η is known as the learning rate, and is discussed in the homonymous section below. A sweep over the whole training set is referred to as an *epoch*. With the right choice of starting weights and learning rate, the algorithm gradually converges to a minimum of the loss function by moving in the direction of the negative gradient. As this is the direction of fastest decrease for the objective function, the algorithm is also called *steepest descent*.

Batch and Online Learning

The updates in (2.2) and (2.3) are made using a sum of the error derivatives over an entire epoch: this is known as *batch learning*. An alternative to batch learning is *online* learning, where weights are updated after each training point is processed. On large training sets, online learning is believed to yield a faster convergence [6], but for the small-size training sets which will be used in this work, batch learning will prove to be sufficient, and is the approach we will be using. It is also common to use a mixture of the two, so-called *mini-batch* learning, where the training set is split into smaller "mini" sets and batch learning is performed over each mini-batch.

Learning Rate
The parameter η is known as the *learning rate*. The learning rate determines how fast the ANN will move along the surface of the loss function, but this need not be equivalent to the speed of convergence. A higher learning rate can potentially mean faster convergence to a minimum, but it carries the risk of having too big a step size and not converging at all. Typically, the learning rate is set to a small constant, resulting in a slower but more certain convergence. Another approach, known as *line search*, is to compute the update for several values of η and choose the one that results in the smallest value for the loss. For our work, we will be setting the learning rate to a small constant, as this is the most common approach.

Multiple Minima
One problem faced by neural networks is the non-convexity of the loss function, which typically has several local minima corresponding to different values of the loss function. The nature of gradient descent is that of moving "downhill" until a minimum is found, regardless of the local or global nature of that minimum. This is illustrated in Figure 3. Typically it is neither necessary nor desirable to find the global minimum of $R(\boldsymbol{\theta})$ (see the section on overfitting below), and in general there is no way of knowing whether the found minimum is local or global, but in order to avoid suboptimal solutions it is often needed to compare several minima. This is usually done by performing gradient descent with a number of different starting weights, and choosing the model that yields the best general predictive power.

Standardization of the inputs
Standardization of the inputs is of particular importance in neural networks, as the scaling of the inputs will determine the subsequent scaling of the α-weights. It is therefore recommended to always standardize all inputs to have mean zero and standard deviation one before fitting the model. This both ensures that all inputs are treated equally by the model, and makes it possible to choose a meaningful range for the starting weights. With standardized inputs, the starting weights are usually drawn from a uniform distribution over the interval $(-0.7, 0.7)$. This is precisely what we did in

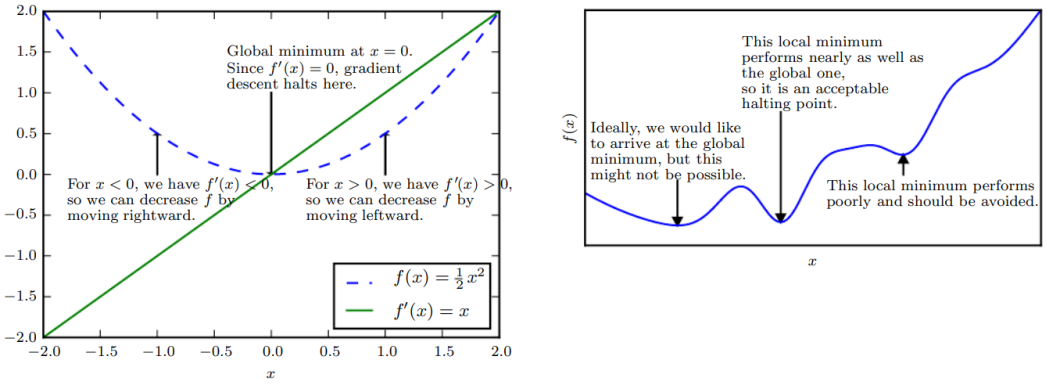

Figure 3: Properties of gradient descent. The figures are taken from [2].

our simulations.

2.2.5 Overfitting

Normally we want to avoid the global minimum of $R(\boldsymbol{\theta})$, as this will generally be an overfit solution. There exist several techniques to prevent overfitting. Here we introduce two widely used such techniques.

The perhaps most natural approach is that of *early stopping*, i.e. to simply stop training the network before it reaches a minimum. This is usually done by keeping track of the loss over both the training set and a validation set, and stopping when the latter starts increasing, as we expect that to correspond to the point at which the model starts overfitting. Because of its intuitiveness, we chose to adopt this approach in this work. Another method to avoid overfitting is to add a penalty term $\lambda J(\boldsymbol{\theta})$ to the loss function, where

$$J(\boldsymbol{\theta}) = \sum_{j,m} \alpha_{jm}^2 + \sum_{m} \beta_m^2$$

and λ is a positive tuning parameter. This results in an additional term $2\lambda\alpha_{jm}$ in the update (2.3) of the α-weights and a corresponding $2\lambda\beta_m$ in the update (2.2) of the β-weights. Bigger values of λ will shrink the weights toward zero. The best value for λ can be found via cross-validation.

2.3 Theoretical Comparison of the Two Models

In Section 2.2.3 we showed how minimizing the cross-entropy loss function is equivalent to maximizing the log-likelihood of a logistic regression in the hidden units. Indeed, a vanilla neural network with the sigmoid as activation and output function and the cross-entropy as loss function is exactly a logistic regression in the hidden nodes. In addition, each hidden node

performs a logistic regression in the explanatory variables. If one so wished, one could describe logistic regression as a neural network with a linear activation function and the sigmoid or softmax output function. The theoretical similarities between the two models are thus many.

However, there are insurmountable differences between them when it comes to the number of data distributions that can be analyzed. Simple logistic regression is a linear classifier, and as such it can only meaningfully be used when one has reason to believe the data can be separated by a line in two dimensions, or by a hyperplane in higher dimensions. Logistic regression, as well as other linear models, can be made to fit nonlinear decision boundaries by expanding the feature space to include nonlinear functions of the original features. The logistic regression will then map a linear function of these basis functions, but this function need not be linear with respect to the original explaining variables. Through this process, one may fit a logistic regression to any type of data with satisfactory results. However, the process is complicated and relies on either a deep understanding of the geometry of the data from the person fitting the model, or some automated shrinkage method such as the LASSO described in Section 2.1.2. Both of these may become impractical as the size of the feature space grows. Neural networks are an ingenious solution to this very problem. In a neural network, the basis functions depend on parameters that are optimized together with the regression parameters. These are, respectively, the α and β weights in the model. The hidden layer is but a space of derived features. The introduction of nonlinearity through the activation function allows the network to create nonlinear derived features. More specifically, the use of a squashing function (a bounded, non-constant — and thus nonlinear — activation function) allows vanilla neural networks to approximate any Borel measurable function from one finite-dimensional space to another arbitrarily well, provided that enough nodes in the hidden layer are available [4]. The theory around Borel measurable functions lies beyond the scope of this thesis, for us it will suffice to say that all continuous functions are Borel measurable [2]. Finally, we want to highlight that while logistic regression is in itself a linear model, which may be extended to nonlinear data through the use of an external method for feature selection, neural networks are not constrained by a predefined mathematical relationship between dependent and independent variables [7], and possess therefore an inherent flexibility which vastly surpasses that of logistic regression.

3 Simulation and Modeling

A simulation study was carried out to test the predictive power of logistic regression and vanilla neural networks in different scenarios. Three different data sets were simulated, with the difference lying mainly in the geometry of

the simulated data. Then, for each of the data sets, a logistic regression and three neural networks with 3, 5 and 15 neurons were fitted to the training data. The models were then tested on 50 simulated validation data sets, and their performances were compared. The simulation is carried out entirely in R.

3.0.1 Fitting of the Logistic Regression

The logistic regressions were fitted using the package `Glmnet`.

3.0.2 Fitting of the Neural Network

For each experiment and setting, three neural networks were fitted with 3, 5 and 15 hidden units, respectively. These will be referred to as NN_3, NN_5 and NN_{15}. We decided to implement our own neural network instead of using a ready library, with the aim of understanding the model in more detail. Since all the inputs underwent a standardization, the starting weights were each drawn from a uniform distribution over the interval $(-0.7, 0.7)$. They were then tuned by gradient descent, with the updates being of the form presented in (2.2) and (2.3). This was repeated for a total of five times, with five different simulation seeds, due to the reasons described in Section 2.2.4, and the best resulting model was chosen. In all runs, a constant learning rate of 0.015 was used.

Computation of the Parameter Updates
The parameter updates are of the form presented in (2.2) for the β-weights and (2.3) for the α-weights. To compute these updates, the partial derivatives of the loss function are needed, with respect to each parameter. As the loss function is a sum over all the data points, its derivative will be a sum of the derivatives for all data points:

$$\frac{\partial R(\boldsymbol{\theta})}{\partial w} = \sum_{i=1}^{N} \frac{\partial R_i(\boldsymbol{\theta})}{\partial w}$$

for any weight w,where

$$R_i(\boldsymbol{\theta}) = -y_i \log(\hat{\pi}(\mathbf{x}_i) + (1 - y_i) \log(1 - \hat{\pi}(\mathbf{x}_i))$$

is the loss function at the ith data point. Repeated application of the chain rule yields

$$\frac{\partial R_i(\boldsymbol{\theta})}{\partial \beta_m} = \Big(- \frac{y_i}{\hat{\pi}(\mathbf{x}_i)} + \frac{1 - y_i}{1 - \hat{\pi}(\mathbf{x}_i)} \Big) \sigma'(\boldsymbol{\beta}^T \mathbf{Z}_i) Z_{m,i} \ , \tag{3.1}$$

$$\frac{\partial R_i(\boldsymbol{\theta})}{\partial \alpha_{jm}} = \Big(- \frac{y_i}{\hat{\pi}(\mathbf{x}_i)} + \frac{1 - y_i}{1 - \hat{\pi}(\mathbf{x}_i)} \Big) \sigma'(\boldsymbol{\beta}^T \mathbf{Z}_i) \beta_m \cdot \sigma'(\boldsymbol{\alpha}_{\boldsymbol{m}}^{\boldsymbol{T}} \mathbf{x}_{\boldsymbol{i}}) x_{j,i} \ , \tag{3.2}$$

where $\mathbf{Z}_i = (Z_{1,i}, \ldots, Z_{M,i})^T$. This gives the parameter updates

$$\beta_m^{new} = \beta_m^{old} - \eta \sum_{i=1}^{N} \Big(- \frac{y_i}{\hat{\pi}(\mathbf{x}_i)} + \frac{1 - y_i}{1 - \hat{\pi}(\mathbf{x}_i)} \Big) \sigma'(\boldsymbol{\beta}^T \mathbf{Z}_i) Z_{m,i} \ , \tag{3.3}$$

$$\alpha_{jm}^{new} = \alpha_{jm}^{old} - \eta \sum_{i=1}^{N} \Big(- \frac{y_i}{\hat{\pi}(\mathbf{x}_i)} + \frac{1 - y_i}{1 - \hat{\pi}(\mathbf{x}_i)} \Big) \sigma'(\boldsymbol{\beta}^T \mathbf{Z}_i) \beta_m \sigma'(\boldsymbol{\alpha_m^T x_i}) x_{j,i} \ . \tag{3.4}$$

These updates constitute a type of batch learning, as discussed in Section 2.2.4. Equations (3.1) and (3.2) have two common factors. These factors are only computed once for each training epoch and data point, i.e. for $i = 1, \ldots, N$. Then, for $m = 1, \ldots, M$, $Z_{m,i}$ and $\beta_m \sigma'(\boldsymbol{\alpha_m^T x_i})$ are computed (for the updates of β and α, respectively). Lastly, for $j = 0, 1, 2$, the last factor in the α-update is computed. This is done in a nested fashion — with j nested in m and m nested in i— so that the same calculation never has to be performed twice. At the end of each epoch, all of the β and α-weights are updated according to (3.3) and (3.4).

Cross Validation for Early Stopping

Early stopping was employed in order to avoid overfitting. As earlier described in Section 2.2.5, this consists in halting the training before a minimum is reached. There exist a number of possible criteria for deciding exactly when training should be halted, none of which is universally recommended. One possible approach is to stop training as soon as the speed of decrease of the training error (i.e. the loss evaluated over the training set) falls below a certain level, i.e. when the learning curve "flattens". This is somewhat based on the assumption that the learning curve looks like the one on the left side of Figure 4, where we can see that the validation error starts increasing as the training error curve flattens. However, the learning curve will oftentimes resemble the one plotted on the right side of the same figure — rather than the idealized left one — both for training and validation error, in which case the aforementioned approach does not appear as intuitive: there is no certainty of the onset of overfitting coinciding with a flattening of the training error curve. One common approach to overcome the need of such an assumption is to have both one training and one or several validation sets, and to keep track of the actual validation error as training progresses. Then the process can be halted as soon as the validation error shows an increasing trend. For this purpose, we could have simulated a number of validation sets from the same distribution as the training set, on which to keep track of the validation error. This is easily done in a simulation study, but is not viable in a real case — where we would hardly need a classification model if the original distribution were known well enough to simulate validation data sets from it. The solution to this problem comes

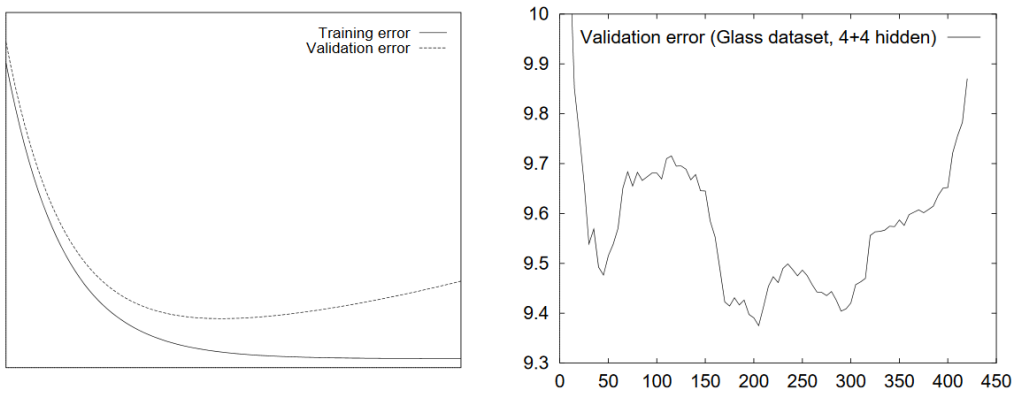

Figure 4: Idealized learning curve (left) versus a real learning curve (right), as a function of the number of iterations. The figure is taken from [8].

in the form of *K-fold cross-validation.* The method consists in dividing the training set into k uniformly drawn subsets and using $k-1$ for training and the remaining set for validation, until all of the sets have been used for validation exactly once. For our experiments we started with a k equal to 10, but later found 5 to yield better results. This is likely due to the small size of the training sets, with the total number of observations for each experiment being close to 200, so that 5-fold cross-validation yielded more stable results that were less dependent on exactly what section was left out for validation.

Choice of Stopping Criterion

At the time of writing, there is no universally recommended answer to exactly when training should be halted in cross-validation based early stopping. The stopping criterion is chosen in an ad-hoc fashion by most researchers [8]. In this study, the following method was used: at each iteration, the loss was evaluated over both the current training and validation set. Gradient descent updates were performed until the validation set loss had been non-decreasing over the last 150 iterations, or the number of iterations reached 2000. At this point, the process was halted and the minimum over the validation loss was retrieved, as well as the corresponding value for the training loss. Because our loss function consists of a sum over data points, some rescaling was needed to adjust for the cross-validation data sets being smaller than the entire, final data set. The training loss was thus divided by the number of observations in the current training set, and the result was stored in a list together with the corresponding results for the rest of the cross-validation runs. To avoid contamination by possible extreme values (e.g. results of runs in which the stopping criteria never was met), values outside two standard deviations from the mean were removed from the list. Finally, the mean of the resulting list was multiplied by the total number of

observations in the training set, and stored in a variable named *threshold*. The final model was trained on the whole training set, until the loss became equal to or lower than the threshold. The fitted model was then tested on 50 validation sets which were simulated from the same distribution as the training data.

4 Experiments

In all three experiments, the starting point is one or several bivariate normal distributions, which are then manipulated to obtain data sets organized in different geometrical shapes. The data was simulated using the R-package `mvrnorm`.

4.1 One Linear Boundary

In the first experiment, the training data was simulated as two bivariate normal distributions with mean vectors $(4, 0)$ and $(0, 0)$, and the unitary matrix as covariance matrix. Initially, 100 data points were drawn from the first distribution and assigned a Y-value of 1, corresponding to class 1. Then, 100 more points were drawn from the second distribution and assigned a Y-value of 0, corresponding to class 2.

Before fitting the model, both the x_1 and the x_2 inputs were standardized by subtracting the sample mean and dividing by the sample standard deviation. In our case, the standardization of the x_2 inputs was somewhat unnecessary as they were already originally drawn from a standard normal distribution, but for the sake of safety and of similarity with real-life scenarios, we decided to perform it.

As Figure 5 illustrates, the data can be separated by a straight line, and therefore we expect the logistic regression and the neural network to perform similarly.

4.2 Piecewise linear boundaries

In the second experiment, the training data was simulated as three bivariate normal distributions, two of which belonged to the same class. The three distributions had mean vectors $(-4, 0), (4, 0)$ and $(0, 0)$, and the unitary matrix as covariance matrix. Initially 50 data points were drawn from the first distribution and assigned a Y-value of 1, then 50 more points were drawn from the second distribution and assigned a Y-value of 1, and finally, 100 points were drawn from the third distribution and assigned a Y-value of 0. This is illustrated in Figure 6.

As in the previous experiment, all inputs were standardized before fitting the models.

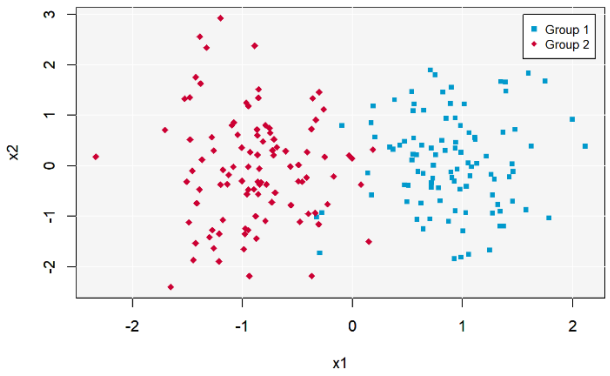

Figure 5: Simulated training set for the first experiment, after standardization of the inputs.

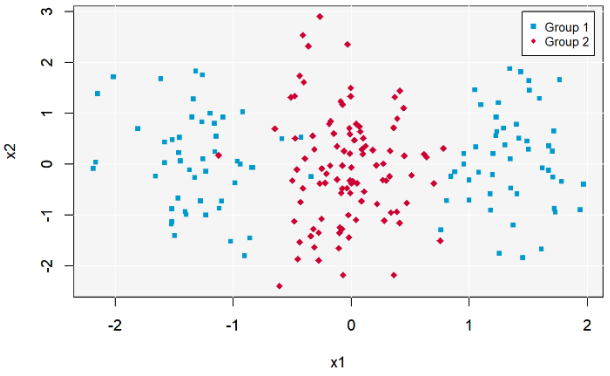

Figure 6: Simulated training set for the second experiment, after standardization of the inputs.

4.3 Nonlinear Boundaries

For the third experiment, 400 data points were drawn from a bivariate normal distribution with mean vector $(0, 0)$ and unitary covariance matrix. Then, the points for which $x_1^2 + x_2^2 < 0.49$ were assigned a Y-value of 1, forming class 1. Class 2 consisted of the points for which $0.65 < x_1^2 + x_2^2 < 1.33$, which were assigned a Y-value of 0. The points satisfying neither of the conditions were ignored, resulting in a total training set of size 160. This resulted in a training set where class 1 is surrounded by class 2, as illustrated in Figure 7, and the number of observations from each class is roughly the same, with class 1 and 2 consisting of 81 and 79 observations, respectively.

As for the other experiments, all inputs were standardized before fitting the models.

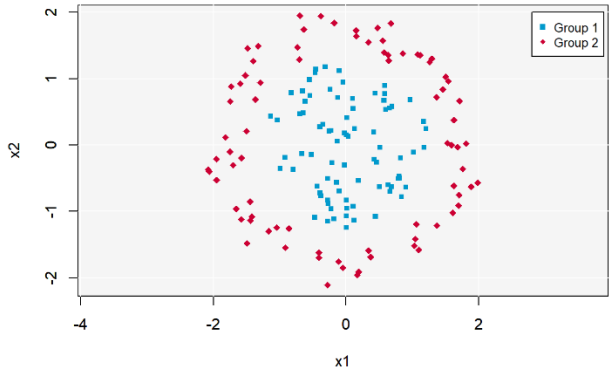

Figure 7: Simulated training set for the third experiment, after standardization of the inputs.

5 Results

5.1 Validation of the Fitted Models

Misclassification Rate

A widely used measure for the predictive power of a model is its *misclassification rate*, defined as the proportion of wrongly classified points

$$\text{MR} = \frac{1}{N}\sum_{i=1}^{N} I(\hat{y}_i \neq y_i)$$

where $\hat{y}_i$ is the predicted class for the ith observation and $I(\hat{y}_i \neq y_i)$ is the indicator function for the predicted class being erroneous.

Type A and Type B Error

The number of misclassified points can be divided into two categories: type A and type B errors. Errors of type A, or *false positives*, are points of class two which have been mistakenly assigned class one. Viceversa, errors of type B, or *false negatives*, are points belonging to class one that have been erroneously classified as class two.

For each experiment, 50 data sets were simulated from the same distribution as the original training data, and used for validation of the fitted models. Each model was tested on all of the validation data sets and the model's misclassification rate, type A error and type B error were recorded for each data set. Tables 1-3 show the mean and standard deviation of the misclassification rates over the 50 sets, both given in percents, as well as the mean type A and type B errors. The last column contains the run time for the training of the model, in the format "run time for fitting of the neural network + run time for cross-validation". Though not strictly associated with validation, this measure will prove itself useful for model comparison.

5.2 Experiment number 1

The results of the first experiment are summarized in Table 1. The neural network with five hidden neurons exhibits the lowest misclassification rate (MR) , followed by the logistic regression and NN_{15}. The four models have nearly identical MR-variance. All the models except NN_3 appear to be slightly biased toward class one. The fitted decision boundaries are illustrated in Figure 8. One major difference between the logistic regression and the three neural networks is the run time, with NN_5 taking 786 times longer than LR. It should be noted that about 93% of this time is due to the 5-fold cross-validation rather than the fitting of the neural network itself. The result might have differed, had another method been used to avoid overfitting. However, the model fitting time for NN_5 is still 56 times that of LR. Thus, in terms of comparison of the two models, the result is unaffected. Model NN_{15} shows a misclassification rate about 5% higher than NN_5. We interpret this as some overfitting taking place, despite the countermeasures adopted. Figure 8 shows how the fitted decision boundary is no longer a straight line, but an S-shaped curve. Model NN_3 is the worst-performing, with a misclassification rate 13% higher than that of NN_5.

5.3 Experiment number 2

Table 2 summarizes the results of the second experiment. The best model here is NN_5, with the lowest misclassification rate, variance and errors of class A. The model appears to be moderately biased towards class two, having almost twice as many false negatives as false positives, but still correctly classifies 96.7% of the data, corresponding to 193.4 out of 200 data points. A quick inspection of the data illustrated in Figure 9 reveals that the error is most likely due to the classes overlapping around the decision boundary, and therefore not necessarily an inherent flaw in the model. Even here, NN_{15} exhibits a worse overall performance than NN_5, with a misclassification rate about 9% higher, indicating that some overfitting has occurred. With a misclassification rate of 50.01%, the LR is as good a classification model as a coin toss, and NN_3 is not much better with a MR of 40.47%. The discrepancies between the run times for the models are even greater than in the first experiment, with NN_5 having a run time 1410 times longer than LR. Having observed such differences in predictive power, however, a comparison of run times is hardly relevant.

5.4 Experiment number 3

The results of the third experiment are summarized in Table 3. The lowest misclassification rate over the three experiments is achieved here, probably due to complete separation of the classes. The best-performing model is clearly NN_{15}, with the lowest misclassification rate, variance and errors of

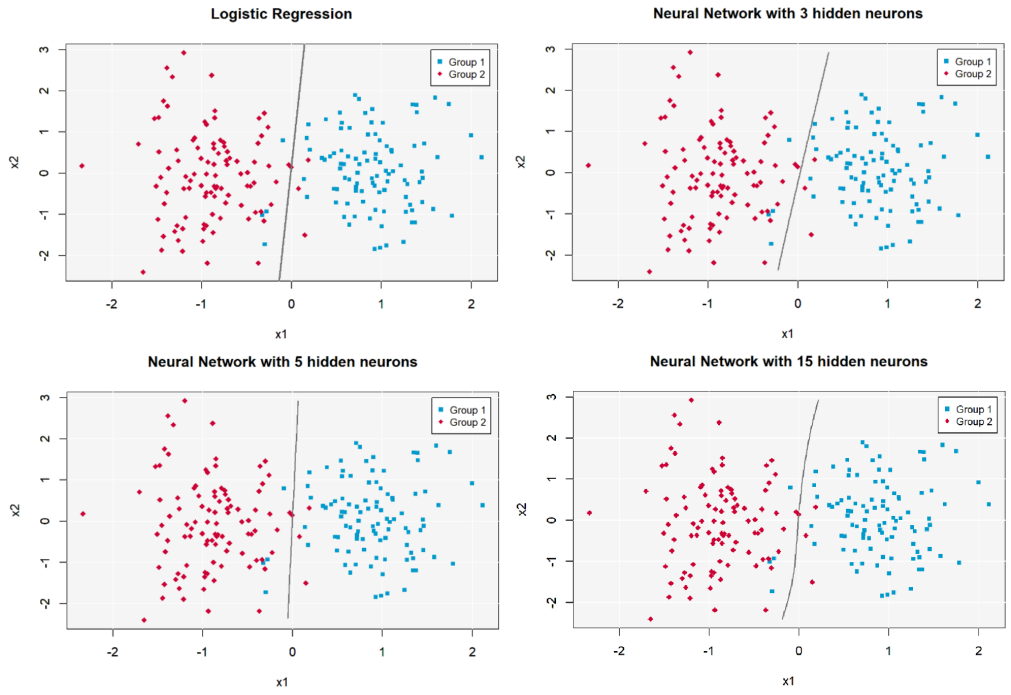

Figure 8: Decision boundaries for experiment nr 1, when the classifier is based on logistic regression or one of three vanilla neural network models with varying number of nodes in the hidden layer.

both types. It is followed by NN_5, NN_3 and lastly LR. The differences in run times are as harsh as in the previous experiment, but Figure 10 illustrates how the number of neurons in the hidden layer directly contributes to a better approximation of the true decision boundary, which we know to be circular.

Table 1: Results of Experiment nr 1

Classifier	Mean MR	SD_{mr}	Err A	Err B	Run Time
LR	2.37	1.31	2.66	2.08	0.3
NN_3	2.60	1.32	2.58	2.62	7+245
NN_5	2.30	1.31	2.52	2.08	17+219
NN_{15}	2.41	1.31	2.68	2.14	119+959

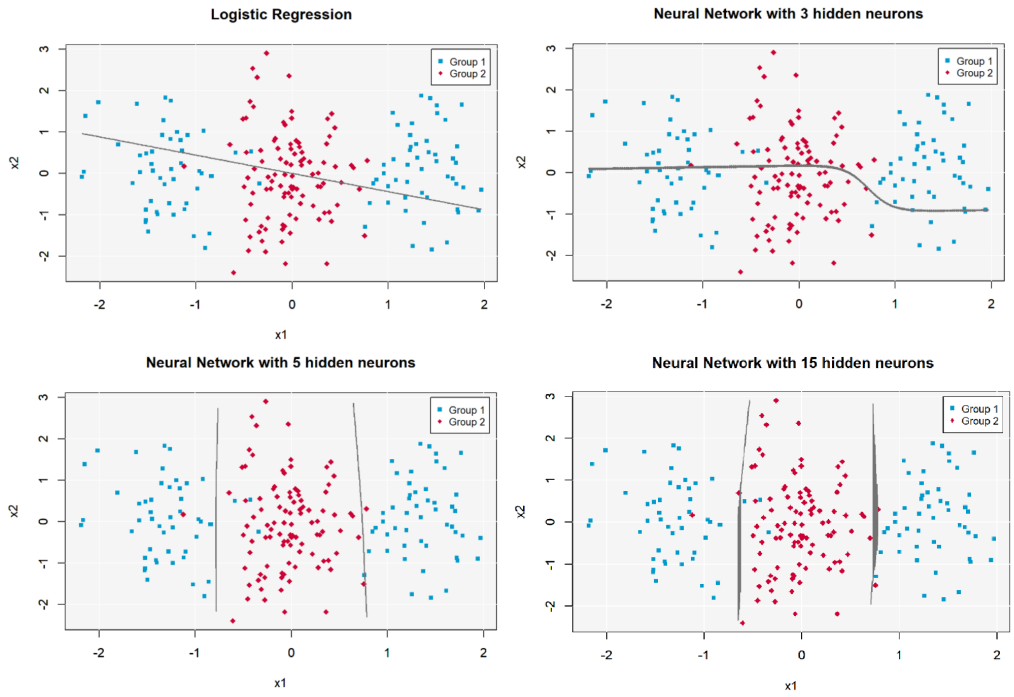

Figure 9: Decision boundaries for experiment nr 2, when the classifier is based on logistic regression or one of three vanilla neural network models with varying number of nodes in the hidden layer.

Table 2: Results of Experiment nr 2

Classifier	Mean MR	SD_{mr}	Err A	Err B	Run Time
LR	50.01	2.11	50.02	50.00	0.3
NN_3	40.47	3.96	44.46	36.48	97+367
NN_5	3.30	1.08	2.38	4.22	31+392
NN_{15}	3.59	1.24	4.14	3.04	130+891

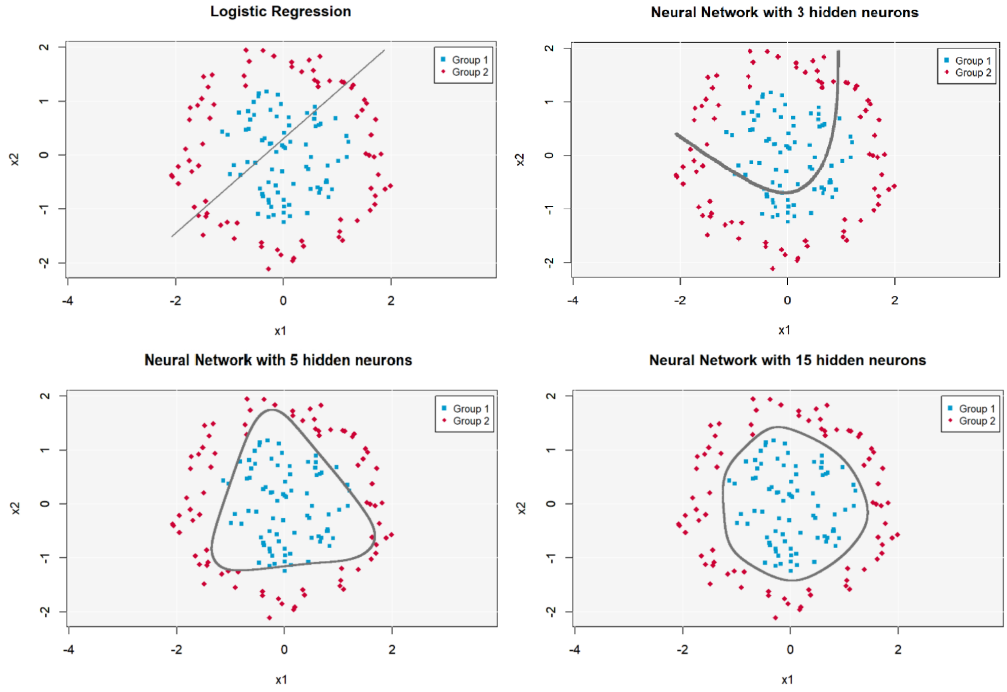

Figure 10: Decision boundaries for experiment nr 3, when the classifier is based on logistic regression or one of three vanilla neural network models with varying number of nodes in the hidden layer.

Table 3: Results of Experiment nr 3

Classifier	Mean MR	SD_{mr}	Err A	Err B	Run Time
LR	47.21	5.73	44.52	32.9	0.2
NN_3	34.15	4.65	25.94	30.06	47+172
NN_5	8.62	2.49	6.38	7.76	48+622
NN_{15}	1.24	1.44	0.64	1.4	212+1754

6 Discussion

6.1 Predictive Power

In experiment 1, where the classes could be separated by one linear boundary, linear regression obtained a misclassification rate about 3% higher than that of NN_5, with equal variance. We conclude that in the case of linearly separable data, logistic regression and a vanilla neural network with an appropriate number of hidden neurons have similar predictive power. Ex-

periments 2 and 3 illustrated the flexibility of neural networks in handling nonlinearity. Here, a simple logistic regression on the explaining variables could not compete with a neural network equipped with enough hidden neurons. The results of these last two experiments suggest that the use of logistic regression is restricted to cases in which one knows that the classes can be linearly separated. This is not entirely true. It is in fact extremely common for researchers to apply logistic regression to nonlinear data, via the introduction of derived features. The key difference lies in the fact that these features must be derived either "by hand" or by some automated model selection procedure such as the LASSO discussed in Section 2.1.2, which can be impractical for complex or high-dimensional data.

6.2 Run Time

The run time for the simplest neural network in each experiment was at best hundreds of times higher than the run time for logistic regression, making it appear favorable to choose logistic regression whenever possible. However, it should be noted that the neural network used in this work was implemented by the authors themselves, and cannot compete with optimized code such as the one found in the R-package `neuralnet`.

6.3 Interpretability

Oftentimes a researcher will not only be intrested in predicting the class for new observations, but also in understanding the effect of the predictor variables on the outcome. This is especially the case for studies in the field of medicine, where membership in a certain class might correspond to the presence of a disease. It is clearly of great interest to not only be able to predict the onset of said disease, but also get insight on the effect of the predicting variables on the risk of developing it. The coefficients β_i in logistic regression are easily interpreted as the increase in conditional log-odds for class one when x_i is increased by one unit while the other x_j are held fixed [9]. Neural networks offer no such direct insight into the effect of the predictors on the outcome. On the contrary, there exist several different sets of parameters corresponding to certain minimums of the loss function [1]. Thus, there is no uniquely determinable optimal set of parameters, and even if there were, its interpretation would not be as straightforward as that of the logistic regression parameters.

6.4 Ground for Improvements

There are several ways to expand and/or improve this study, which were not implemented due to restrictions on time and experience. Here are some of them.
Due to limited programming experience, the neural network used in this

study results in unreasonably long run times. One clear way to improve this work would be to improve the model by means of optimizing the code. Some structural changes to the neural network model could also be implemented in order to improve its performance: these include the use of the back-propagation algorithm for computation of the gradient updates, and exchanging the sigmoid activation function for the ReLU, or *rectified linear unit* activation function. According to Ian Goodfellow, "using a rectifying nonlinearity is the single most important factor in improving the performance of a recognition system" [2].

Automatic early stopping with cross-validation was introduced in order to automate the model. Further automation can be achieved by performing *node pruning*, i.e. starting with a number of nodes and gradually removing nodes whose weights are near zero, thus making the model slimmer and faster. This corresponds to eliminating the derived features appearing to have a low influence on the outcome, much like performing a kind of backward stepwise selection of the nodes. This way, we could have included one single neural network in each experiment, instead of three different NNs.

A natural extension of this work would also be to manually add the appropriate derived features to the logistic regression models in the experiment, e.g. $x_3 = x_1^2 + x_2^2$ in the third experiment, and compare the performance of this adjusted logistic regression with that of a neural network. Alternatively, the study could be expanded to include more general linear regression models starting with a large set of derived features and making use of feature selection methods, such as the LASSO, discussed in Section 2.1.2.

Finally, more reliable results could be achieved by simulating a number of data sets from each given distribution, i.e. by simulating several different training data sets for each experiment. The models of interest would then be fitted to each data set, tested on a number of simulated validation sets, and the averaged results would be presented. This way, a more general analysis could be carried out, with a lowered risk of the conclusion being dependent on particular qualities of the current data set. In the case of this study, however, we do not expect this to affect the conclusion to a noticeable extent. The data sets at hand range from a size of 160 to 200 data points, which — in two dimensions — is large enough to guard us from a strong dependence of the results on the nature of the simulated data set. Moreover, all inputs were standardized before model fitting, resulting in a shrinkage of the possible differences between simulated data sets from the same distribution. Finally, the differences in performance highlighted by this study were so pronounced that it is highly unlikely they would depend solely on particular qualities of the simulated data sets.

7 Conclusion

Our study suggests that neural networks offer much greater flexibility than logistic regression, and therefore can yield better — or at least equally good — results in most scenarios. This conclusion is supported by the theory for the models, as well as by the wide popularity of neural networks in areas where prediction accuracy is the main concern, such as the field of artificial intelligence. Classification tasks play a crucial role in the automation of many processes, and thus the usefulness of a powerful classifier cannot be overstated, even when it comes at the price of low interpretability. This also applies to risk assessment tasks, where a higher predictive power can save lives. For instance, in a study about landslide risk in the Hendek region in Turkey, neural networks were found to yield more credible results than logistic regression [10].

Logistic regression is, however, the clear choice when the primary goal of model development is to look for possible causal relationships between independent and dependent variables [12]. This has led to a conflict of opinions regarding the use of neural networks and other machine learning models, as opposed to the use of older, more easily interpreted models from the field of statistics. Stephan Dreiseitl and Lucila Ohno-Machado found, in their 2002 methodological review of logistic regression and neural networks for biomedical data classification, that "there was a 5:2 ratio of cases in which it was not significantly better to use neural networks" [7]. At the same time, neural networks are being used to analyze images of patients and correctly identify malignant tumors and recognize solid nodules [11] and many similar tasks, showing that the predictive power of neural networks can be of great value even in the medical field, as a precious aid in diagnosing disease.

8 References

[1] Bishop, C.M. (2006) *Pattern Recognition and Machine Learning*, Springer.

[2] Goodfellow, I., Bengio Y., & Courville A. (2016) *Deep Learning*, MIT Press. http://www.deeplearningbook.org

[3] Hastie, T., Tibshirani, R., & Friedman, J. (2017) *The Elements of Statistical Learning*, Second Edition, Springer.

[4] Hornik, K., Stinchcombe, M., & White, A. (1989) Multilayer feedforward networks are universal approximators *Neural Networks*, Volume 2, Issue 5, 1989, Pages 359-366.
https://doi.org/10.1016/0893-6080(89)90020-8

[5] Golik, P., Doetsch, P., & Ney, H. (2013) Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison. Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH. 1756-1760.
https://www.researchgate.net/publication/266030536_Cross-Entropy_vs_Squared_Error_Training_a_Theoretical_and_Experimental_Comparison

[6] Wilson, D.R., & Martinez,T.R. (2003) The general inefficiency of batch training for gradient descent learning. *Neural Networks*, Volume 16, Issue 10, December 2003, Pages 1429-1451.
https://doi.org/10.1016/S0893-6080(03)00138-2

[7] Dreiseitl, S., & Ohno-Machado, L. (2002) Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, Volume 35, Issues 5-6, October 2002, Pages 352-359.
https://doi.org/10.1016/S1532-0464(03)00034-0

[8] Prechelt, L. (1998) Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, Volume 11, Issue 4, June 1998, Pages 761-767.
https://doi.org/10.1016/S0893-6080(98)00010-0

[9] Nöu, A. (2018) Logistic Regression versus Support Vector Machines Bachelor Thesis in Mathematical Statistics at Stockholm University, 2018.
kurser.math.su.se/pluginfile.php/20130/mod_folder/content/0/Kandidat/2018/2018_20_report.pdf?forcedownload=1

[10] Yesilnacar, E., & Topal, T. (2005) Landslide susceptibility mapping: A comparison of logistic regression and neural networks methods in a medium scale study, Hendek region (Turkey) *Engineering Geology*, Volume 79, Issues 3-4, 11 July 2005, Pages 251-266.
https://doi.org/10.1016/j.enggeo.2005.02.002

[11] Qi, X., Zhang, L., Chen, Y., Pi, Y., Chen, Y., Lv, Q., & Yi, Z. (2019) Automated diagnosis of breast ultrasonography images using deep neural networks *Medical Image Analysis*, Volume 52, February 2019, Pages 185-198.
https://doi.org/10.1016/j.media.2018.12.006

[12] Tu, J. V. (1996) Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, Volume 49, Issue 11, November 1996, Pages 1225-1231.
https://doi.org/10.1016/S0895-4356(96)00002-9