Draft version November 10, 2019
Preprint typeset using LaTeX style emulateapj v. 12/16/11

PROJECT 2 ON MACHINE LEARNING: CLASSIFICATION AND REGRESSION, FROM LINEAR AND LOGISTIC REGRESSION TO NEURAL NETWORKS

Bruce Chappell
email b.a.chappell@fys.uio.no
Francesco Anello
email f.anello1@campus.unimib.it/francean@student.matnat.uio.no

Draft version November 10, 2019

ABSTRACT

In this project we study neural networks implemented in two different problems types: first, a classification problem is analysed using a credit card default payment dataset; then a linear regression problem is studied with data generated from the Franke function. Our aim is to compare our neural network algorithm against logistic regression and SciKit-Learn for classification, and against Ordinary Least Squares and SkiKit-Learn for linear regression. The results obtained by using our own code match SciKit-Learn for classification, and best SciKit-Learn for regression. A 10×10 neural network turns out to be the best predictive model for classifying the credit card data and a $100 \times 50 \times 20$ network just barely outperforms OLS for the Franke Function data. Due to computational complexity, OLS is taken to be the preferred regression method.

1. INTRODUCTION

In recent years, the popularity of neural networks has exploded. Many problems within business, science, and engineering require predicting the class or type of an output based on given inputs. Neural networks have shown significant strengths when solving these problems which has contributed to their popularity increase. They have also been shown to be effective when solving classic linear regression problems.

In this first part of this paper, we will consider a classification problem applied to the credit card data set from UCI. This data set documents 23 variables related to credit card usage and whether a customer defaulted on their next bill payment. The objective is to develop a neural network to predict whether or not a customer defaults on their next payment based on the 23 input features. Our network will be constructed using gradient descent for backpropogation for learning between the layers. To ensure the neural network is working efficiently, we will compare its predictions against those of the more basic Logistic Regression. Due to the high number of predictors, Principal Component Analysis is used too.

In the second part we will explore using neural networks to make linear regression fits, specifically on the Franke function. Fitting this function using traditional regression methods was studied in detail in Project 1 and we will benchmark our new results against those from Project 1.

In the following section we presents a theoretical and mathematical overview of logistic regression, gradient descent methods and neural networks. Then a description containing a short summary of our specific implementations is shown. At the end, the analysis of results are displayed with final conclusions.

2. THEORY AND METHODS

This section describes the different methods studied and applied to obtain our final results. In particular, Logistic regression, gradient descent, Neural networks and Principal Component Analysis are explained. For the linear regression section, reference is made to the previous project.

2.1. *Logistic Regression*

Logistic Regression is a statistical method used to study and quantify the relations between one or more independent variables $X_1, ..., X_p$ and a dichotomous dependent variable Y.

This method is better considered as a form of classification, not of regression. Let Y be a binary response, where:

$$\begin{cases} Y = 1 \text{ represents "success" with } Pr(Y = 1) = \pi \\ Y = 0 \text{ represents "failure" with } Pr(Y = 0) = 1 - \pi \end{cases} \tag{1}$$

Y is a Bernoulli random variable with parameter $p = \pi$ with probability function:

$$f(y; \pi) = \pi^y (1 - \pi)^{1-y};\ y = 0, 1;\ 0 \leq \pi \leq 1 \tag{2}$$

Hence, its expected value $E(Y)$ is:

$$E(Y) = \sum_{y=0}^{1} y f(y; \pi) = 0 \cdot f(0; \pi) + 1 \cdot f(1; \pi) = \pi \tag{3}$$

Let Y be a linear combination of the inputs variables $Y = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \varepsilon$. Thus:

$$E(Y|x) = p(x) = \pi(x) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p = \boldsymbol{X}\beta \tag{4}$$

So, since by definition $0 \leq \pi(x) \leq 1\ \forall x$, using a linear regression model would not be appropriate: it would give values in the set of real numbers $\mathbb{R}$. To guarantee that π stays between 0 and 1, we require a positive monotone function that maps the 'linear predictor' into the unit interval. Commonly the logistic (or sigmoid) function is

adopted for this reason:

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z} \tag{5}$$

So in this case,

$$E(Y|x) = \pi(x) = \frac{e^{\boldsymbol{X}\hat{\beta}}}{1+e^{\boldsymbol{X}\hat{\beta}}} \tag{6}$$

This function is not linear but we can make it linear considering its *logit* transformation :

$$logit(\pi(x)) = ln(\frac{\pi(x)}{1-\pi(x)}) = ln(\frac{Pr(Y_i=1)}{Pr(Y_i=0)}) = \boldsymbol{X}\hat{\beta} \tag{7}$$

Logistic regression can be consider as linear regression applied on the logit transform. The function $\frac{\pi(x)}{1-\pi(x)}$ is called odds ratio and represents how much the success is more plausible to the failure. It assumes real positive values, 0 included.
So, we have that:

$$Pr(y_i = 1|x_i, \hat{\beta}) = \sigma(x_i\hat{\beta}) \tag{8}$$

and

$$Pr(y_i = 0|x_i, \hat{\beta}) = 1 - Pr(y_i = 1|x_i, \hat{\beta}) \tag{9}$$

The prediction is set to 1 if $Pr(y_i = 1|x_i, \hat{\beta}) \geq 0.5$, otherwise to 0. In order to estimate the parameters of logistic regression, we use the Maximum likelihood function. Since the n units are independent, the Maximum likelihood function is given by the product of n probability functions:

$$L(\hat{\beta}) = \prod_{i=1}^{n} \Big[p(y_i = 1|x_i, \hat{\beta})\Big]^{y_i} \Big[1 - p(y_i = 1|x_i, \hat{\beta}))\Big]^{1-y_i}$$

So, the log-likelihood function for logistic regression is:

$$\sum_{i=1}^{n} \Big(y_i \log p(y_i = 1|x_i, \hat{\beta}) + (1-y_i) \log \Big[1 - p(y_i = 1|x_i, \hat{\beta}))\Big]\Big) \tag{10}$$

Now, we take the cost function, $\mathcal{C}(\hat{\beta})$, to be the negative log-likelihood. This is the equation that needs to be minimized to find optimal parameters $\hat{\beta}$ for the model. The derivatives with respect to $\hat{\beta}$ are as follows:

$$\frac{\partial \mathcal{C}(\hat{\beta})}{\partial \hat{\beta}} = -\hat{X}^T (\hat{y} - \hat{p}) \tag{11}$$

$$\frac{\partial^2 \mathcal{C}(\hat{\beta})}{\partial \hat{\beta} \partial \hat{\beta}^T} = \hat{X}^T \hat{W} \hat{X} = H \tag{12}$$

H is positive definite, so the function considered is convex and has a unique global minimum. The optimal set of parameters to find it can be found using gradient descent. The performance of a logistic regression model can be evaluated by calculating the accuracy score:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \tag{13}$$

2.2. *Gradient descent*

Gradient descent (also known as steepest descent) is one of the most preferred first-order optimization methods for finding minima of a given multivariate function which, in machine learning, is the cost function. The procedure iteratively performs updates on the parameters β in the direction where the gradient of the cost function is large and negative. Parameters are initialized to some value $\boldsymbol{\beta}_0$ and iteratively updated according to the equation

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \nabla C(\mathbf{x}_k), \tag{14}$$

where $\nabla C(\mathbf{x}_k)$ is the gradient or first derivative of the cost function $C(\mathbf{x}_k)$ at the k-th step and γ_k is the learning rate which determines how big a step would be in the direction of the gradient at time iteration k. If γ_k is of sufficient size, this method will converge to a local minimum of the cost function. However, if γ_k is too small, the method will converge very slowly which comes at a large computational cost. On the contrary, if γ_k is too large, the algorithm may fail to converge.

This method is very sensitive to choices of the learning rates and also of GD's initial conditions. Depending on where one initiates the algorithm, one can become stuck in a local minimum, missing the true global minimum of the cost function. The cost functions studied in Machine Learning often present many local minima, making it important to include stochasticity in our method.

2.2.1. *Stochastic gradient descent with mini-batches*

Stochasticity is introduced in the training process by approximating the gradient on a subset of the data, called a mini-batch. If there are n data-points and the mini-batch size is M, there will be $B_k = n/M$ mini-batches where $k = 1, ..., n/M$. The idea is to perform gradient descent over random mini-batches of points and update the parameters accordingly.

$$\nabla_\beta C(\beta) = \sum_{i=1}^{n} \nabla_\beta c_i(\mathbf{x}_i, \beta) \to \sum_{i \in B_k}^{n} \nabla_\beta c_i(\mathbf{x}_i, \beta) \tag{15}$$

Thus a gradient descent step can be rewritten as:

$$\beta_{j+1} = \beta_j - \gamma_j \sum_{i \in B_k}^{n} \nabla_\beta c_i(\mathbf{x}_i, \beta) \tag{16}$$

A cycle over all $k = 1, ..., n/M$ mini-batches is commonly called an epoch.

2.3. *Neural Networks*

In Machine Learning, a neural network is a computational model composed by artificial "neurons", inspired by the simplification of a neural circuit in a brain. The neurons are organized into multiple layers, where the output of one layer is the input for the next. In particular, the first layer is called the input layer, the middle layers are "hidden layers", and the final layer is the output layer. Each "neuron" i takes a vector of d input features $x = (x_1, x_2, ..., x_d)$ and produces an output $a_i(x)$ which depends on the type of the non-linear function considered. However, $a^{(i)}$ can be decomposed into a linear transformation $z^{(i)}$ that weighs the importance

of various inputs, and a non-linear activation function $f(z^{(i)})$.
The linear transformation can be described in the following way:

$$\mathbf{z}^{(i)} = \mathbf{w}^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)} \tag{17}$$

where $\mathbf{w}^{(i)}$ is the vector containing the assigned weights to each neuron and $b^{(i)}$ is a vector of biases associated with each neuron. This feed forward process can be generalized by the following steps:

$$\mathbf{z}^{(0)} = \mathbf{w}^{(0)}\mathbf{X}_{inputs} + \mathbf{b}^{(0)} \tag{18}$$

$$\mathbf{a}^{(i)} = f^{(i)}(\mathbf{z}^{(i)}) \tag{19}$$

$$\mathbf{z}^{(i)} = \mathbf{w}^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)} \tag{20}$$

$$\mathbf{z}^{(Last)} = \mathbf{w}^{(Last)}\mathbf{a}^{(Last-1)} + \mathbf{b}^{(Last)} \tag{21}$$

$$\mathbf{a}^{(Last)} = f^{(Last)}(\mathbf{z}^{(Last)}) \tag{22}$$

$\mathbf{a}^{(Last)}$ is the output of the neural network. After a feed forward run, we want to backpropogate through the network updating the weights and biases in a way that minimizes the cost function. We do this by calculating how much each layer of weights and biases contributed to the overall error in the prediction. We start with the error in the last layer L or the prediction layer.

$$\delta_j^L = f'(z_j^L)\frac{\partial \mathcal{C}}{\partial(a_j^L)} \tag{23}$$

where $f'(z_j^L)$ is the derivative of the activation function of the j-th neuron with respect to its input and $\frac{\partial \mathcal{C}}{\partial(a_j^L)}$ is the derivative of the cost function with respect to the output of the network. For regression problems, δ_j^L is simply:

$$\delta_j^L = f'(z_j^L)(a_j^L - t_j) \tag{24}$$

where t_j is the target vector. For classification:

$$\delta_j^L = (a_j^L - y_j) \tag{25}$$

where y_j is a vector of categorical data.
We now find the errors in the previous layers using the following equation:

$$\delta_j^{l-1} = \sum_k \delta_k^l w_{kj}^l f'(z_j^{l-1}) \tag{26}$$

These calculated errors are used to update the corresponding weights and biases by the following gradient descent method:

$$w_{jk}^l \leftarrow w_{jk}^l(1-\lambda) - \eta\delta_j^l a_k^{l-1} \tag{27}$$

$$b_j^l \leftarrow b_j^l - \eta\delta_j^l \tag{28}$$

where η is the learning rate and λ is a regularization parameter.
To prevent runaway values in the hidden layers, it is convenient to feed the z weighted sums into some function that squishes the real number line into the range between 0 and 1. Two commonly used functions are the Sigmoid function and the Hyperbolic Tangent.

Using a neural network is a flexible method since it creates good regression fits and can be used for multiclass classification problems. However, it presents also some disadvantages. For example, there are no standard and strict criteria with which to choose the number hidden layers and nodes. Optimizing a network for a data set often requires a bit of guessing and checking to arrive at a suitable structure.

2.4. *Principal Component Analysis*

Given p variables $X_1, ..., X_j, ...X_p$, the aim of the PCA is to determine k new variables not directly observable, called PC, such that $k << p$ with the following properties:

1. They are uncorrelated: $\rho(PC_s, PC_m) = 0, \forall s \neq m$

2. They reproduce the larger possible amount of total variance remaining after the construction of the first $s-1$ Principal Components:
   $Var(PC_s) \geq Var(PC_m)\ \forall s < m$ and
   $Var(PC_s) \leq Var(PC_m)\ \forall s > m$

3. Principal components are built in a non-increasing order with the respect to the amount of total variance that they reproduce:
   $Var_{tot} = tr(\Sigma) = \sum_{j=1}^{p} Var(X_j)$

4. They have expected values equal to zero:
   $E(PC_s) = 0\ \forall s = 1, ..., k \leq p$

To guarantee the latter, PCA assumes that the considered data is centered around the origin: hence, PC are built by using centred or standardized variables as their appropriate linear combination:

$$PC_s = v_{s1}X_1 + ... + v_{sj}X_j + v_{sp}X_p \forall s = 1, ..., k \leq p \tag{29}$$

Hence, Principal Components can be considered as intrinsic variables that approximately describe the data. From a geometrical point of view, PCA identifies the hyperplane that is located closest to the data, and therefore it projects the data onto it. It can be shown the following relation:

$$Var(PC_1) = v_{s1}^T \Sigma v_{s1} = \lambda_1 \tag{30}$$

with $\lambda_1 = max_{s=1,...,p}(\lambda_s)$. v_1 is considered the normalized eigenvector associated with λ_1. Generally, in matrix notation we have that: $\mathrm{y}_s = \boldsymbol{X}v_s$ where $v_s = [v_{s1}, ..., v_{sj}, ..., v_{sp}]^T$ is the normalized eigenvector ($v_s^T v_s = 1$) associated with the eigenvalue $\lambda_s = Var(PC_s)$ and it is orthogonal to all the other s-1 eigenvectors v_m associated to the first $s-1$ Principal Components PC_m with $m = 1, ..., s-1$.

3. DATA EXPLORATION AND IMPLEMENTATION

3.1. *Taiwan Credit Card Dataset*

This dataset contains 30000 instances on default payments (described by a binary variable which assumes value equal to 1 for "Yes" and 0 for "No") and demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.
The data set comes from the UCI Machine Learning Repository Irvine, CA: University of California, School of Information and Computer Science.

There are 24 variables, among which 9 are categorical. More details about the nature and the meaning of the variables are described in the Appendix (Table 2).

Before running the proper analysis, we do pre-processing on the considered dataset. First we check if there are missing data and if the values assumed by the categorical variables are in agreement with the documentation provided for the data. This is possible using the *value_counts*() function of pandas which calculates frequency distribution for categorical variables. There are no missing values but many anomalous values are detected as shown in the following table:

TABLE 1
Undocumented values for categorical variables

Variable	Undocumented label	Observed values
EDUCATION	0	14
MARRIAGE	0	54
PAY_1	-2	2579
PAY_1	0	14737
PAY_2	-2	3927
PAY_2	0	15730
PAY_3	-2	4085
PAY_3	0	15764
PAY_4	-2	4348
PAY_4	0	16455
PAY_5	-2	4546
PAY_5	0	16947
PAY_6	-2	4895
PAY_6	0	16286

We have undocumented values for EDUCATION, MARRIAGE and all the repayment status variables PAY. We decide to delete the observations with undocumented values for EDUCATION and MARRIAGE. If we removed all the observations with unlabelled values for these features, we would have only 4030 observations left from the original 30000. Thus we change the value of PAY variables with -2 to 0 to consider them as irrelevant. After this first step of data cleaning, 29601 observations are left.

Afterwards, we use pandas.DataFrame.describe function to get all the most important descriptive statistics (central tendency and dispersion) to start understanding the distribution of the variables and check if there are any anomalies or outliers. We observe very wide ranges of values in our continuous variables that could cause problems in our study. We created the function *find_outliers* to return the dataset without outliers to a given tolerance. Generally a value is considered as an outlier if it is bigger than $Q3 + 1.5 \times IQR$ or less than $Q1 - 1.5 \times IQR$ where Q1, Q3, IQR are respectively the first quartile, the third quartile and the interquantile range.
Since only 14299 observation would remain using the standard 1.5, we choose to use 3 as our threshold. Using this alternative threshold, 21708 units are left. These units will represent the first data set we focus on. We now split the data set into test and training sets and standardize the continuous and PAY features by removing the mean and scaling to unit variance since the features present different measuring scales. This is advantageous from a computational point of view because large input values can saturate activation functions (sigmoid and tanh) and lead to a slowdown in the training.

The correlation matrix (Figure 24 in Appendix) shows that while features associated with bill amount ('$BILL\_AMT$') present high correlations to each other, they do not contribute significantly to the target variable. As expected, the variables related to payment status are correlated to each other and show higher contribution to the outcome variable. They are not so high (between 0.23 and 0.35) but significantly more important than the correlations observed for all the other features. We observe negative correlations that in absolute value are smaller than 0.01 for all the $BILL\_AMT$ and PAY_ATM variables and correlation for "SEX", "AGE", "MARRIAGE","EDUCATION" are also negative. Hence, many of our variables do not appear to provide significant information for our purposes. We then work to reduce the dimensionality of the problem using Principal Component Analysis (PCA) which is described in 2.4. We observe the following scree plot for the standardized training data:

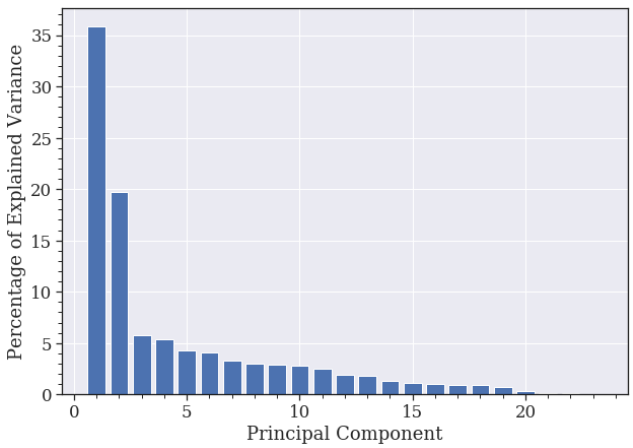

Fig. 1.— This plot reports the percentage of reproduced variance by each principal component(in ordinate) with the respect to its order number (in abscissa). A common approach is to keep the PC before the "elbow" for analysis. In this case we observe that the "elbow" is located after the two first PC.

We choose to keep the number of PC that explains 95% of the variance. For our training data set this is the first 15 PC. We now have a data set comprised of 15 new variables instead of the original 23 variables. This will constitute our second dataset.

We then perform a grid search over the

hyperparameters η and λ to calculate the accuracy scores of the neural network and logistic regression methods when predicting payment defaults. We use k-fold cross-validation with $k = 5$ to create train and validation sets from our training set. After finding ideal η and λ values we explore the relationship between batch size, epochs, and accuracy score. This is done for the original dataset and the PC dataset.

3.2. *Franke's Function*

We also use the neural network algorithm to study a linear regression case from our previous project, the Franke function. For comprehensive descriptions and further details regarding the latter, reference is made to the Section 3 of Project 1.

For this regression analysis, a random 300×300 mesh grid in the domain $[0, 1]$ is created and fed into the Franke Function producing a 300×300 mesh grid of z values. We then add noise to the z values of the form $z_i = f(x_i, y_i) + \epsilon$ where $\epsilon \sim N(0, \sigma = 0.05)$.

The same hyperparamter analysis performed on the credit card data is performed on the Franke Function but MSE and R^2 are calculated for each set of parameters. These two error metrics are illustrated in Section 2 of Project 1.

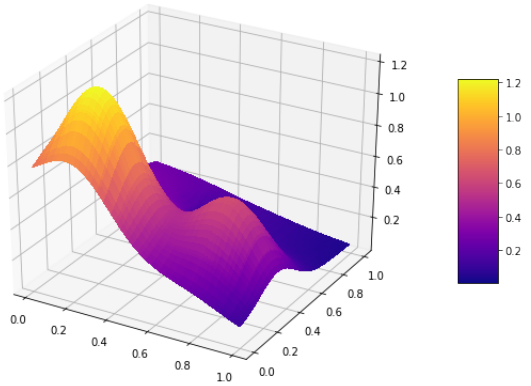

FIG. 2.— Franke's function: particular function used in interpolation problems. The function is evaluated on the square $x_i \in [0, 1]$, $\forall i = 1, 2$.

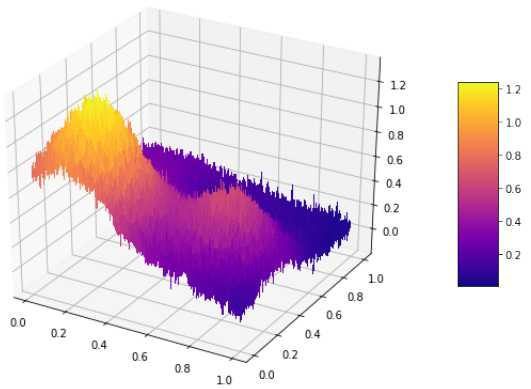

FIG. 3.— Franke's function with added noise.

4. RESULTS

In this section all the results are obtained using our own original codes for logistic regression and neural network classification and regression.

The results are presented in sections corresponding to the solving method used to obtain them and the data set.

4.1. *Taiwan Credit Card Dataset*

4.1.1. *Logistic Regression*

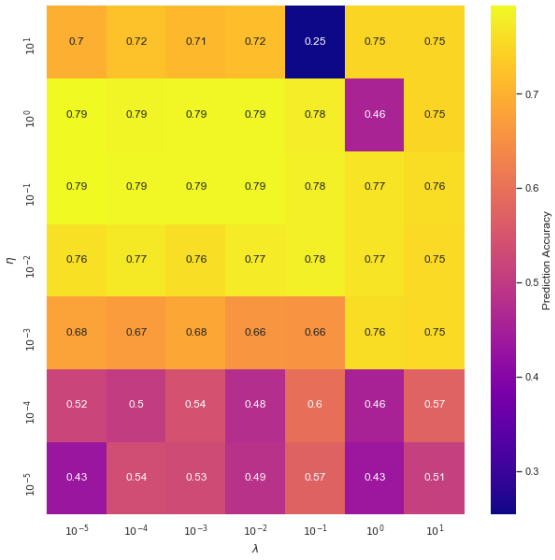

FIG. 4.— Accuracy score for different regularization λ and learning rate η for logistic regression using Stochiastic Gradient Descent. The Sigmoid function was used as the activation function and the problem was solved using 20 epochs and a batch size of 100.

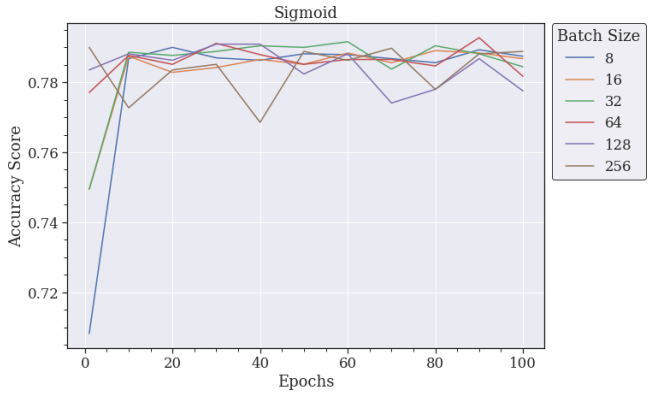

FIG. 5.— Accuracy score for Logistic regression plotted against the number of epochs and batch sizes using Sigmoid activation. For this plot, $\eta = 5 \times 10^{-1}$ and $\lambda = 5 \times 10^{-3}$.

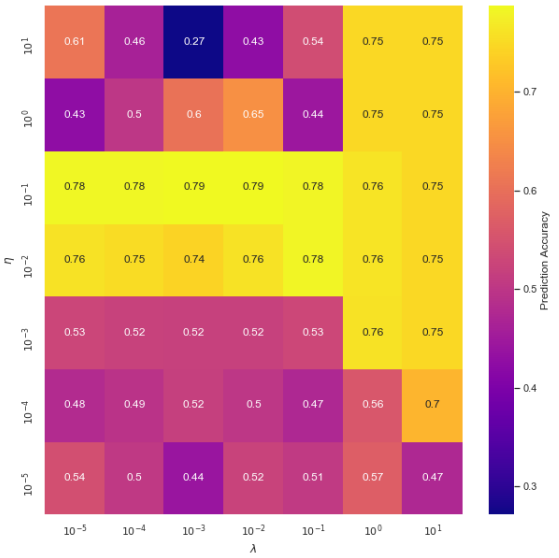

FIG. 6.— Accuracy score for different regularization λ and learning rate η for logistic regression using Stochiastic Gradient Descent. The Tanh function was used as the activation function and the problem was solved using 20 epochs and a batch size of 100.

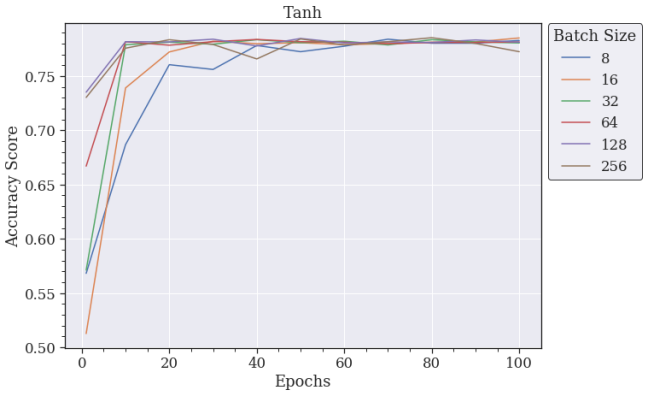

FIG. 7.— Accuracy score for Logistic regression plotted against the number of epochs and batch sizes using Tanh activation. For this plot, $\eta = 1 \times 10^{-1}$ and $\lambda = 5 \times 10^{-3}$.

4.1.2. *Logistic Regression with PC Analysis*

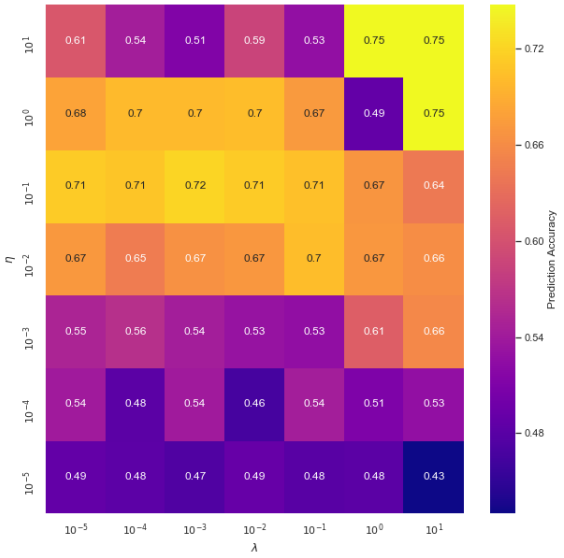

FIG. 8.— Accuracy score for different regularization λ and learning rate η for logistic regression using Stochiastic Gradient Descent. The Sigmoid function was used as the activation function and the problem was solved using 20 epochs, a batch size of 100, and 15 PC variables.

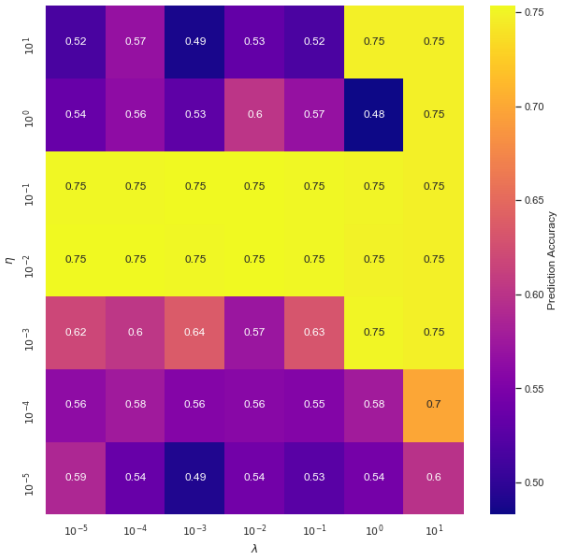

FIG. 9.— Accuracy score for different regularization λ and learning rate η for logistic regression using Stochiastic Gradient Descent. The Tanh function was used as the activation function and the problem was solved using 20 epochs, a batch size of 100, and 15 PC variables.

4.1.3. *Neural Network Algorithm*

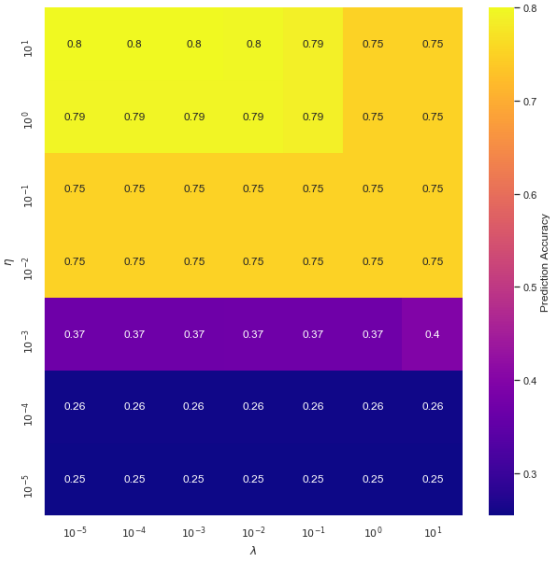

FIG. 10.— Accuracy score for different regularization λ and learning rate η for the Neural Network algorithm. The Sigmoid function was used as the activation function and the problem was solved using 10 epochs, a batch size of 100, and two hidden layers with 10 neurons each.

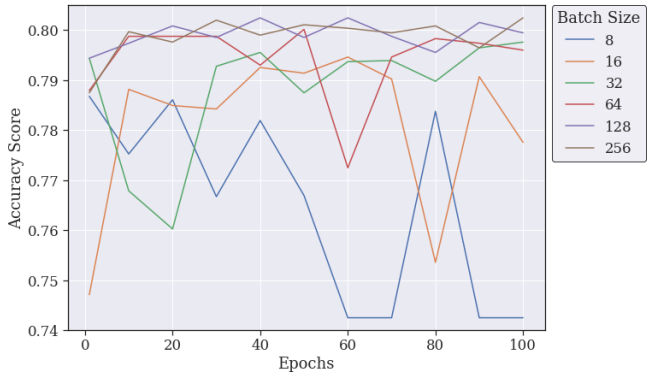

FIG. 12.— Accuracy score for the Neural Network using Sigmoid activation plotted against the number of epochs and batches. Two hidden layers with 10 neurons each, $\eta = 1 \times 10^{1}$, and $\lambda = 1 \times 10^{-2}$ were used when solving.

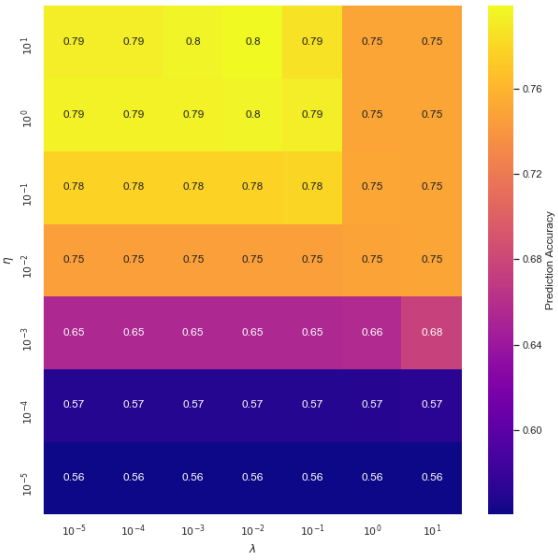

FIG. 11.— Accuracy score for different regularization λ and learning rate η for the Neural Network algorithm. The Sigmoid function was used as the activation function and the problem was solved using 10 epochs, a batch size of 100, and two hidden layers with 80 and 20 neurons respectively.

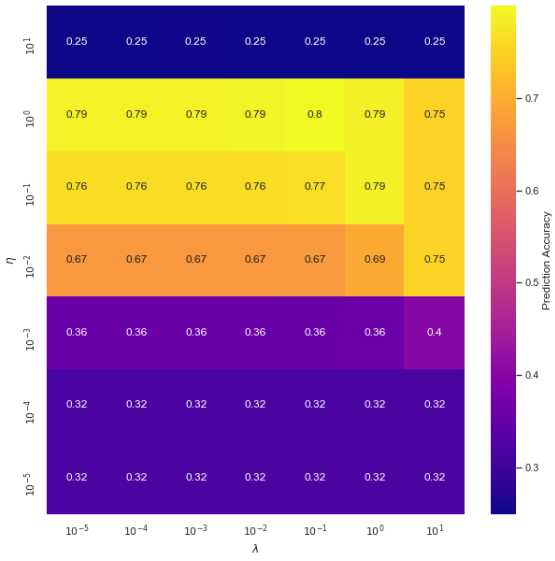

FIG. 13.— Accuracy score for different regularization λ and learning rate η for the Neural Network algorithm. The Tanh function was used as the activation function and the problem was solved using 10 epochs, a batch size of 100, and two hidden layers with 10 neurons each.

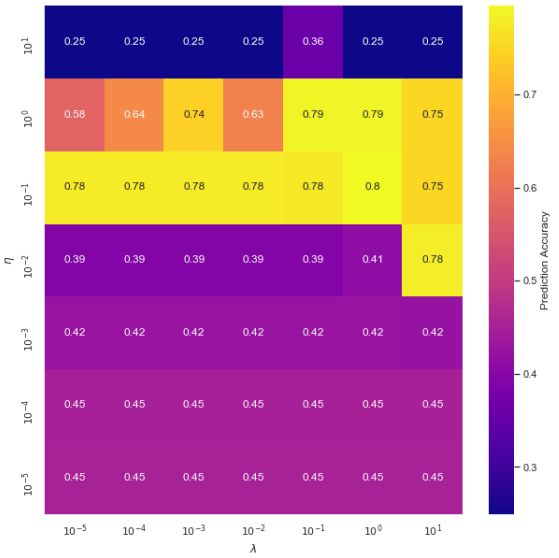

FIG. 14.— Accuracy score for different regularization λ and learning rate η for the Neural Network algorithm. The Tanh function was used as the activation function and the problem was solved using 10 epochs, a batch size of 100, and two hidden layers with 80 and 20 neurons respectively.

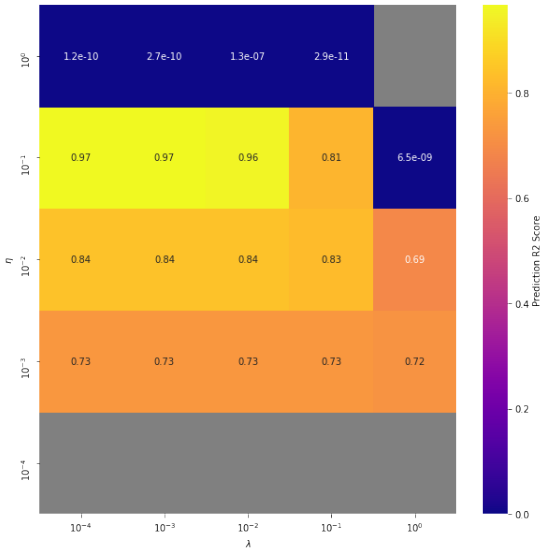

FIG. 16.— R^2-score for different regularization λ and learning rate η for the Franke function. These results were obtained using Sigmoid activation, 3 layers of 100, 50, and 20 neurons respectively, 50 epochs, and a batch size of 200. The grey squares indicate invalid values that resulted form numerical instability.

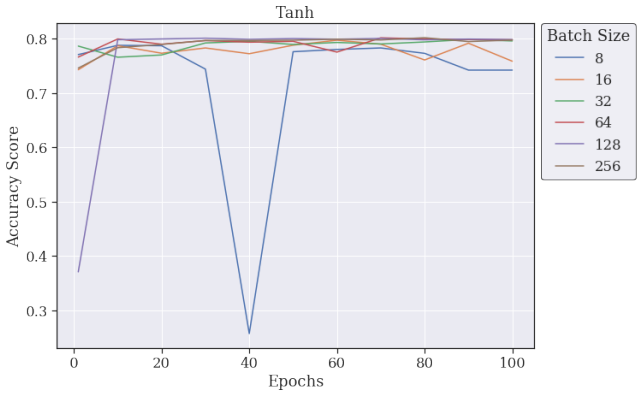

FIG. 15.— Accuracy score for the Neural Network using Tanh activation plotted against the number of epochs and batches. Two hidden layers with 10 neurons each, $\eta = 1 \times 10^{0}$, and $\lambda = 1 \times 10^{-1}$ were used when solving.

4.2. *The Franke Function*

4.2.1. *Neural Network*

For The Franke Function, results are obtained using different combinations of regularization parameters λ and learning rates η as in the previous classification case. All results are obtained using our own neural network code.

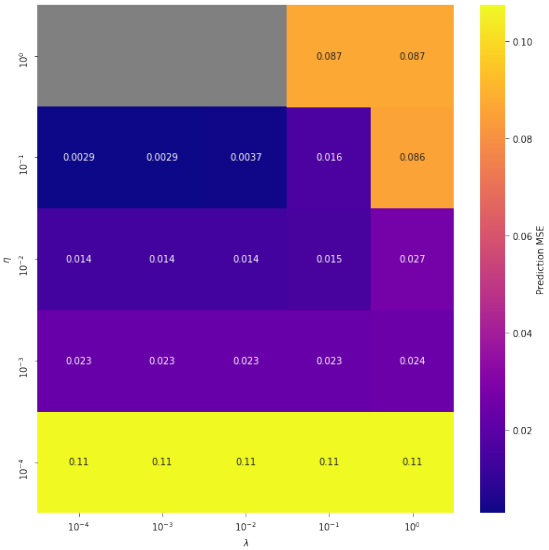

FIG. 17.— MSE for different regularization λ and learning rate η for the Franke function. These results were obtained using Sigmoid activation, 3 layers of 100, 50, and 20 neurons respectively, 50 epochs, and a batch size of 200. The presence of grey squares is explained in 16

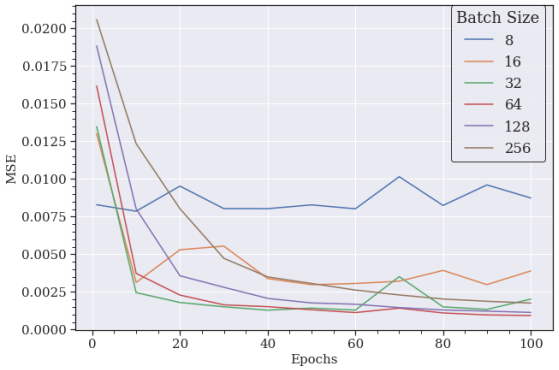

FIG. 18.— *MSE* plotted against the number of epochs and batch sizes. These results were obtained using Sigmoid activation, 3 layers of 100, 50, and 20 neurons respectively, $\eta = 1 \times 10^{-1}$, and $\lambda = 1 \times 10^{-3}$.

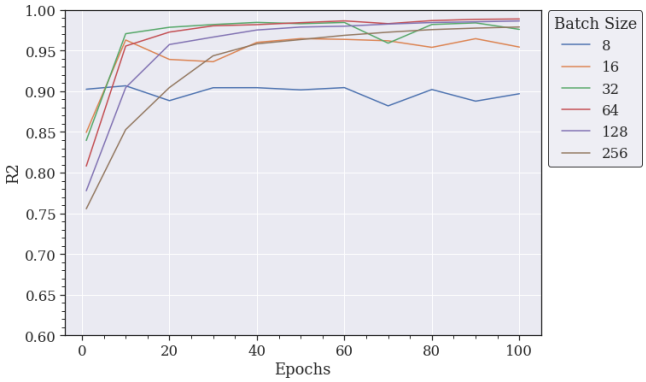

FIG. 19.— R^2-score plotted against the number of epochs and batch sizes. These results were obtained using Sigmoid activation, 3 layers of 100, 50, and 20 neurons respectively, $\eta = 1 \times 10^{-1}$, and $\lambda = 1 \times 10^{-3}$.

5. DISCUSSION

5.1. *Logistic Regression on credit card data*

We begin our analysis with Logistic Regression applied to the full credit card data set. In 4 we find optimal the optimal hyperparameters to be $\eta = 5 \times 10^{-1}$ and $\lambda = 5 \times 10^{-3}$ when using Sigmoid activation for a fixed amount of epochs and batch sizes. We then perform an analysis of the accuracy score versus batch size and epochs for these optimal parameters which is shown in 5. This process is repeated for the Tanh activation in 6 and 7. We see that model predicts the test data more accurately when the Sigmoid function is used. Due to the high dimensionality of the problem, we then explore the performance of the model when using the first 15 Principle Components of the the training data. The first 15 PCs contain 95% of the explained variance of the data. We again perform a hyperparameter grid search using the Tanh and Sigmoid functions and find that we lose quite a bit of accuracy by reducing dimensionality. 8 and 9 both show that the maximum accuracy of the 15 dimensional model is 75% compared with 79% for the full model.

5.2. *Neural Network on credit card data*

We being our analysis with a hyperparameter grid search using both the Sigmoid and Tanh activation as shown in 10 and 13. Here we have used 2 layers of 10 neurons each and both yield a maximum accuracy score of 80%. Due to the somewhat mysterious nature of neural networks we tried various numbers of layer and neurons through a process of trial and error. We found that networks with 2 layers of 80 and 20 (11 and 14) neurons respectively produced similar results to the networks with 2 10 neuron layers. The added complexity did not contribute to increased accuracy and we therefore continued the study with the simpler models. 15 and 12 show the evolution of the accuracy score of the simpler models with respect to epochs and batch size. As was the case in logistic regression, we see that a larger batch size and number of epochs tends to give a better accuracy score. To ensure our neural network is providing reasonable results, we then compare our results against those of the SciKit-Learn MLPClassifier class.

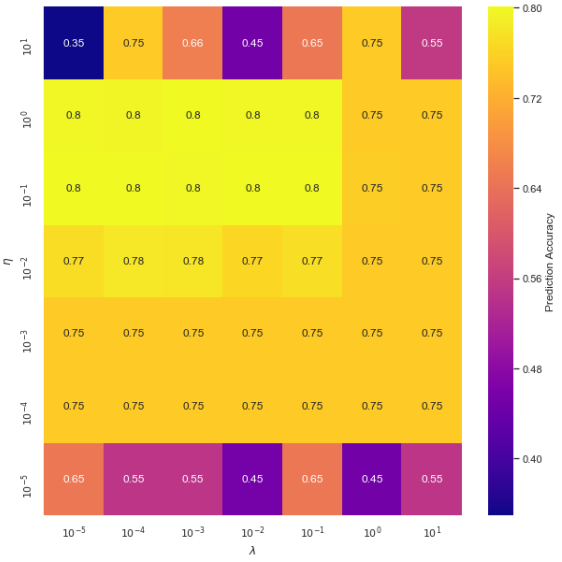

FIG. 20.— Accuracy score for different regularization λ and learning rate η for Neural Network trainand considering 10 epochs, batchsize equal to 100, two hidden layers with 10 neurons each using SciKit-learn's MLPClassifier algorithm.

FIG. 21.— Accuracy score of SciKit-Learn's neural network using Sigmoid activation and plotted against the number of epochs and batch sizes with two hidden layers of 10 neurons each.

20 and 21 show good agreement with our maximum accuracy score of 80% for the network explored in 10 and 12.

5.3. *Neural Network on Franke Function*

Our neural network performed quite well when fitting the Franke Function data. After a bit of trial and error we found that 3 hidden layers with 100, 50, and 20 neurons provided good results. As seen in 16 and 17 for 50 epochs and a batch size of 200 we obtain a maximum R^2-score of .97 and a minimum MSE of 0.0029. 18 and 19 both show that with a larger batch size and number of epochs, the fit becomes better. From Project 1, our best fit to the Franke Function data was found when using OLS with a polynomial of degree 11. We obtained a minimum MSE value of 2×10^{-3} for this fit. Using the neural network with 100 epochs and a batch size of 64, we obtain a minimum MSE of 1×10^{-3}. We also fitted the Franke function using SciKit-Learn's MLPRegressor class. The results are shown below in 22 and 23

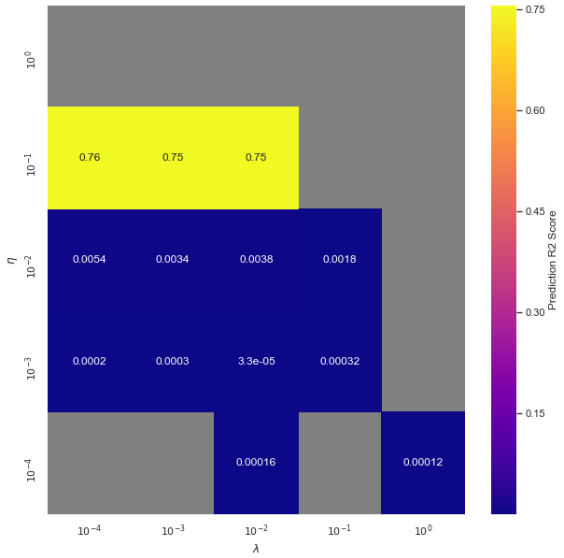

FIG. 22.— R^2-score for different regularization λ and learning rate η for the Franke function. These results were obtained using Sigmoid activation, 3 layers of 100, 50, and 20 neurons respectively, 50 epochs, and a batch size of 200 with SciKit-Learn's MLPRegressor. The grey squares indicate invalid values that resulted form numerical instability.

Surprisingly, both our neural network and OLS code outperformed SciKit-Learn's neural network regressor algorithm. From all the plots in this section, it is notable that for a fixed learning rate, there tends to be small changes in performance when using different regularization parameters. This highlights the learning rate's dominance in developing an accurate model.

6. CONCLUSIONS

In our analysis, we have shown that our neural network code performs well both when solving classification problems and linear regression problems. For the credit card data, our logistic regression code performed well when the whole data set was analyzed using both Tanh and Sigmoid activation functions. Our maximum accuracy score here was 79%.

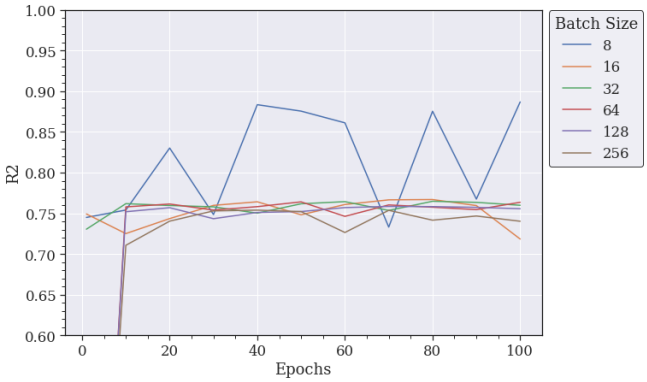

FIG. 23.— R^2-score plotted against the number of epochs and batch sizes. These results were obtained using Sigmoid activation, 3 layers of 100, 50, and 20 neurons respectively, $\eta = 1 \times 10^{-1}$, and $\lambda = 1 \times 10^{-3}$ with SciKit-Learn's MLPRegressor.

While reducing dimensionality using PCA resulted in a computationally simpler problem, we considered the loss of accuracy to be too great. Our simple 10×10 neural network performed the best for both Sigmoid and Tanh activations with maximum accuracy sores of 80%. It was interesting to find that adding complexity in the layers did not result in noticeably better accuracy scores. We therefore assert that using the simple 10×10 neural network is ideal when analyzing the credit card data. Our neural network slightly outperformed our OLS code from Project 1 with $MSE = 1 \times 10^{-3}$ compared to $MSE = 2 \times 10^{-3}$ for OLS. However, due to the computational simplicity of OLS and the marginal MSE reduction when using our neural network, we assert that our OLS code is preferred when considering the Franke Function data.

6.1. *Further research*

These two data sets both present many potential options moving forwards. One area to explore would be a more empirical way of determining optimal hidden layers and neurons. We took a bit of a cavalier guess and check approach due to time constraints, but see potential in exploring these parameters more thoroughly. The Sigmoid and Tanh both behaved similarly, as is expected due to their similar shape. It would therefore be interesting to explore using the ReLU activation function in the hidden layers as it has a different behavior than the two activations we used. This study would be more complete if our neural network results would have been compared to other techniques like convolution neural networks and decision trees.

REFERENCES

[1]Hjorth-Jensen, Morten Lectures notes in FYS-STK4155.Data analysis and machine learning: Optimization and Gradient Methods,LogisticRegression,Neural networks https://github.com/CompPhysics/MachineLearning
[2]Pankaj Mehta, A high-bias, low-variance introduction to Machine Learning for physicists, May 29, 2019
[3]Trevor Hastie, Robert Tibshirani, and JH Friedman. The elements of statistical learning: data mining, inference, and prediction. 2009.
[4]Piccolo Domenico, Statistica, 21 Oct 2010.
[5]Azzalini, Scarpa. Data Analysis and Data Mining: an introduction, 2012-04-23.
[6]Zani S., Cerioli A. Analisi dei dati e data mining per le decisioni aziendali, Giuffrè Editore, Milano, 2007.

7. APPENDIX

All source code, data and figures can be found at the github repository: https://github.com/bruce-chappell/FYS4155/tree/master/project2

TABLE 2
Variables of the dataset with possible values

Variables	***Description***	***Type (with documented values)***
LIMITBAL	Amount of given credit in NT dollars	Continuous
SEX	Gender	Categorical-binary (1=male, 2=female)
EDUCATION	Level of education	Categorical (1=graduate school, 2=university, 3=high school, 4=others)
MARRIAGE	Marital status	Categorical (1=married, 2=single, 3=others)
AGE	Age in years	Discrete
PAY1-PAY6	Repayment status respectively in September, August, July, June, May, April	Categorical (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, . . . , 8=payment delay for eight months, 9=payment delay for nine months and above)
BILLAMT1-BILLAMT6	Amount of bill statement respectively in September, August, July, June, May, April (NT dollar)	Continuous
PAYAMT1-PAYATM6	Amount of previous payment respectively in September, August, July, June, May, April (NT dollar)	Continuous
default.payment.next.month	Default payment in June	Categorical-binary (1=yes, 0=no)

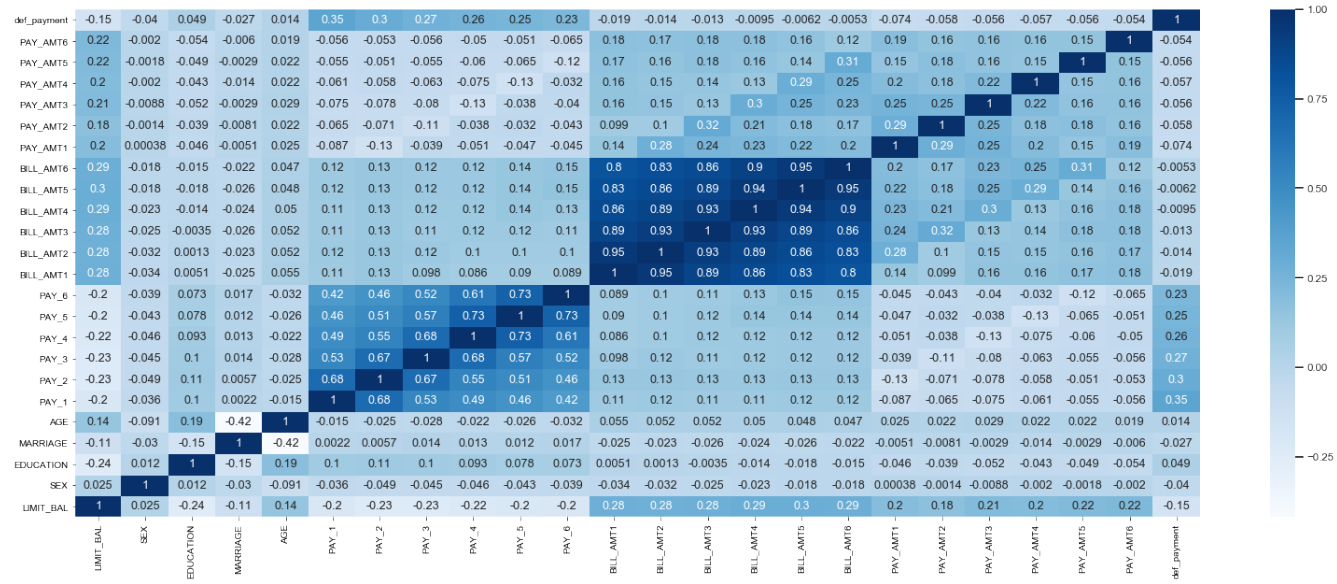

FIG. 24.— Correlation matrix for Taiwan Credit Card Dataset