Machine Learning: Using Logistic Regression and Neural Networks on Yellowstone Terrain and Credit Card Data

Claxton JB., Materne L., Steinnes L.

Institute of Physics, University of Oslo

j.b.claxton@fys.uio.no, lukasmat@student.matnat.uio.no, lassst@student.matnat.iuo.no

November 12, 2019

Abstract

Neural networks are heavily applied in Machine Learning for deep learning and perform like neurons in the human brain. This paper explores the implementation of logistic regression and neural networks for classifying credit card holders and predicting their default or not default. Neural networks achieve similar accuracy (0.81) compared to logistic regression. Neural networks are also applied to the regression case of predicting the landscape of Yellowstone Park. The neural network performs a little poorer (mse = 0.004) compared to the Ordinary Least Squares method in linear regression (mse = 0.001).

I Introduction

Artificial Neural Networks (ANNs) were first inspired by Warren S. McCulloch, a neurophysiologist, and by Walter Pitts, a mathematician. In their 1943 paper the two described how neurons activate and how signals propagate through networks of neurons. They described the neural activity as "all or none", either there is or is not a signal sent; there are no partial signals. McCulloch and Pitts hence said in their paper that this behaviour of the network can be treated as propositional logic or statement logic [4].

In the 1980's more research began applying neural networks to computing. Despite this research and significant developments, neural networks have encountered many problems. A few of the problems encountered were: the selection of the parameters and the structure of the ANN, the choice of initial values, the choice of learning samples and the

convergence of learning algorithms [1]. In recent years however, more research has been put into optimising ANN and using their advantages, especially with the introduction of activation functions other than the sigmoid function (CITE A PAPER). Throughtout this paper ANN will just be referred to as Neural Networks (NNs).

This papers' objective is to: apply NNs and Logistic Regression to classification problems, apply NNs and linear regression to a regression problem, and make comparisons between the models in each problem. The classification problem is predicting the default of credit card holders in Taiwan and the regression problem is predicting the terrain of Yellowstone Park (USA).

The paper is laid out by first introducing the theory behind Logistic Regression and NNs. There is a brief summary of the implementation of both Logistic Regression and NNs. The paper moves on to finding the best hyper parameters for making predictions and shows the confusion matrix for these best predictions. By finding the best predictions based on the tuned hyper-parameters, the paper moves on to discuss and compare the models within classification and regression problems. Finally conclusions are made on the advantages and disadvantages of the models and future improvements are suggested.

II Theory

II.I Classification Problems

The typical classification problem in machine learning is to infer a class label L_k from K-different classes given a set of features x. Two possible methods among others, are Neural Networks (NN) and logistic regression. The goal with these methods is to train the ability to predict class labels on a given labeled data set $\boldsymbol{X} \in \mathbb{R}^{N\times f}$ with N samples and f features and labeled target classes $\boldsymbol{t} \in \mathbb{R}^N$. Both methods are categorized as supervised learning methods and share the the same cost function

$$C(\boldsymbol{W},\boldsymbol{X}) = -\frac{1}{N}\sum_{i=1}^{N} t_i \log\left[p_{\mathrm{CL}(t_i)}(\boldsymbol{W},\boldsymbol{x}_i)\right]. \tag{1}$$

The cost function in eq. (1) is called cross-entropy and is related to the negative log-likelihood of the predicted probability $p_k(\boldsymbol{W},\boldsymbol{x})$ to be in a class k given a set of features $\boldsymbol{x}$. The function $\mathrm{CL}(t_i)$ in eq. (1) returns the true class label k of the target t_i. The prediction depends on the model weights $\boldsymbol{W}$ which are optimized by

$$\hat{\boldsymbol{W}} = \min_{\boldsymbol{W}} C(\boldsymbol{W},\boldsymbol{X}) = \nabla_{\boldsymbol{W}} C(\boldsymbol{W},\boldsymbol{X})\Big|_{\hat{\boldsymbol{W}}} = 0. \tag{2}$$

Using the softmax-function one can calculate the prediction

$$p_k(\boldsymbol{W},\boldsymbol{X}) = \frac{\exp(\boldsymbol{X}\boldsymbol{w}_k)}{\sum_{i=1}^{K}\exp(\boldsymbol{X}\boldsymbol{w}_i)}, \tag{3}$$

where $\boldsymbol{w}_k = [W_{k1}, W_{k2}, ..., W_{kf}]^T$ is the k-th row of the weight matrix $\boldsymbol{W}$. Note, that by construction $\sum p_k = 1$. The final prediction P is the the class with the maximal probability

$$P = \text{argmax}\, \boldsymbol{p}, \quad \text{where } \boldsymbol{p} = [p_1, p_2, ..., p_k, ..., p_K]^T. \tag{4}$$

The difference between logistic regression and NN are in the number of model weights. Whereas, for logistic regression $\boldsymbol{W} \in \mathbb{R}^{f\times K}$, a deep NN can have more weights due to the number of hidden layers. One should note here, that the biases are not listed separately since they can be included into the weight matrix as 0-th column and into the data matrix as 0-th feature which is always 1.

II.II Neural Networks

A neural network (NN) is a method that takes an input and processes it using connections between different layers (l) until the outputlayer L is reached. From one layer to the next, activation functions ($f(z_j^l = a_j^l)$ take the activation z^l (5), containing weights (w_{ij}^L) and biases(b_j^l), from the previous layer, process it and pass on the output to the next layer. M_{l-1} is the number of nodes in the previous layer. In this way, NNs are similar to the neural networks found in biological systems.

$$z_j^l = \sum_{i=1}^{M_{l_{1-1}}} w_{ij}^l a_j^{l-1} + b_j^l \tag{5}$$

There are several types of NNs suited to different analysis. However, in this article, a fully-connected feed forward network is used. This means that every node (i) in the network is connected to all the nodes in previous layers. If such a network have more than three layers, it is called a multilayer perceptron (MLP) (Fig. 1).

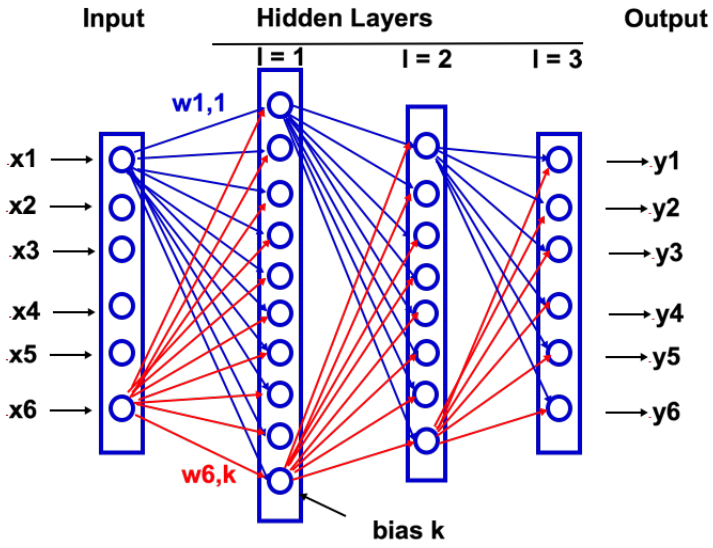


Figure 1: An overview of the multilayered perceptron model (MLP). Input data x_i, with i $= 0, 1, .., k^{l-1}$ are sent into the NN. The data are then activated and given weights (w_{ij}) and biases (b_j) with z_j^l, where j $= 0, 1, ..k^l$, and are passed to the next layer through an activation function, $f(z_j^l)$. Each layer consist of k nodes, and the number of nodes can differ between layers. In feedforward and backpropagation, all nodes between hidden layers are connected to each other.

The advantage of MLPs is due to theoretical mathematics, namely the universal approximation theorem. It states that given an activation function which is non-constant, continuous, bounded and monotonically increasing between hidden layers, any continuous function can be approximated with MLP to a certain accuracy with only one hidden layer and a finite number of nodes. Adding more hidden layers give even more universality to MLPs approximation potential.

Generally, MLPs consist of feed forward and backpropagation, which is then coupled with a standard or stochastic gradient descent method (SGD) to optimize the output model with respect to the weights. The feed forward is just the matrix and vector representation of eq. 5 for a dataset matrix X and a specific layer (l)

$$\hat{\boldsymbol{z}}^l = \hat{\boldsymbol{X}}\hat{\boldsymbol{W}}^l + \hat{\boldsymbol{b}}^l. \tag{6}$$

Applying the activation function over layers $l = (1, 2, ..., L-1, L)$ will in the last step yield

$$\hat{\boldsymbol{a}}^l = f(\hat{\boldsymbol{z}}^l), \tag{7}$$

$$\hat{\boldsymbol{z}}^L = \hat{\boldsymbol{a}}^L \hat{\boldsymbol{W}}^L + \hat{\boldsymbol{b}}^L. \tag{8}$$

In the case of classification, the output activation ($\boldsymbol{z}^l$) is then applied to the Softmax function (eq. 3), which determines the probability of a specific output neuron being in a particular class.

After the initial input is fed forward to the output layer, a backpropagation must be performed to reduce errors for the unknown weights and biases. The backpropagation updates $\hat{\boldsymbol{W}}^{\boldsymbol{l}}$ and $\hat{\boldsymbol{b}}^l$ in such a way that the cost-function $C(\boldsymbol{W}, \boldsymbol{X})$ is minimized. In the case case of classification, the cost-function the the cross entropy, while for regression, it is equal to the mean square error (MSE),

$$C(\boldsymbol{W}, \boldsymbol{X}) = \frac{1}{n} \sum_{i=1}^{n} (a_i^L + t_i)^2, \tag{9}$$

where t_i are the targets. Updating weights and biases is performed backwards from the output by calculating

$$\boldsymbol{\delta}^L = f'(\hat{\boldsymbol{z}}^L) \circ \frac{\partial C}{\partial \hat{\boldsymbol{a}}^L}, \tag{10}$$

where $\circ$ is the Hadamard product, which takes the multiplication between elements,ij, of two arrays with same dimensions. The errors ($\boldsymbol{\delta}^l$) for the hidden layers $\mathrm{l} = L-1, L-2, ..., 2$ then become,

$$\boldsymbol{\delta}^l = \delta^{l+1} \hat{\boldsymbol{W}}^L \circ f'(\hat{\boldsymbol{z}}^l). \tag{11}$$

$\hat{\boldsymbol{W}}^{\boldsymbol{l}}$ and $\hat{\boldsymbol{b}}^l$ can then be updated in the direction of the gradient of the cost-function,

$$\hat{\boldsymbol{W}}^{\boldsymbol{l}} \leftarrow = \hat{\boldsymbol{W}}^{\boldsymbol{l}} - \eta \boldsymbol{\delta}^l \hat{\boldsymbol{a}}^{l-1}, \tag{12}$$

and

$$\hat{\boldsymbol{b}}^l \leftarrow = \hat{\boldsymbol{b}}^l - \eta \frac{\partial C}{\partial \hat{\boldsymbol{b}}^l} = \hat{\boldsymbol{b}}^l - \eta \boldsymbol{\delta}^l. \tag{13}$$

The learning rate (η) is a parameter that determines how far along the gradient each iteration will take the weights and biases. As a concequence, a small learning rate will give a NN that learns slowly. However, if the learning rate is set too large, the NN might

learn poorly, due to missing the optimal weights and biases as a consequence of having too large step size.

After backpropagating to layer l= 2, the feedforward will pass trough all layers, and this cycle is repeated until the gradient have stopped decreasing, meaning a minima for the cost-function has been reached. The method of finding the optimal weights and biases are thus called Gradient Descent (GD).

Since there are many dimensions to keep track of, table 1 gives a summary of parameters with their respective dimensions.

Table 1: Parameters and dimensions used in the neural network.

Parameter	Symbol	Dimensions
Data	$\hat{\boldsymbol{X}}$	(n_{inputs}, $n_{features}$)
Weights	$\hat{\boldsymbol{W}}^{\boldsymbol{l}}$	($n_{features}$, $n_{neurons}$)
Bias	$\hat{\boldsymbol{b}}^{l}$	$n_{neurons}$
Activation	$\hat{\boldsymbol{a}}^{l}$	(n_{inputs},$n_{neurons}$)
Output	$f(\hat{\boldsymbol{z}}^{l})$	(n_{inputs},$n_{categories}$)

II.III Stochastic Gradient Descent

There are several shortcomings to performing a standard GD. Firstly, it is computationally costly to calculate the gradient of the cost-function. Moreover, there are no guarantees that the biases and weights used to initialize the NN will give a local or global minima of the cost-function. A way to deal with these drawbacks are to perform the gradient descent in batches and epochs, thus introducing stochasticity and enhancing computational efficiency.

The input data of size N is sorted randomly into subsets called minibatches, each containing M data points. Thus, the total number of batches will be N/M. Computing the gradient from each minibatch $B_{k=1,2...,N/M}$, and then taking the mean, instead of the whole dataset (see eq. 1), ensures both that the program run faster and the probability of stopping at a local minima is reduced.

III Methods

III.I Stochastic Gradient Descent

The stochastic gradient descent class `SGD` takes care of all parts of a SGD. It can be used to run a whole set of epochs and get the final optimal parameters, or it can be used as sub-parts, e.g. to create and run a mini-batch. Moreover a decaying learning rate and momentum terms for the gradient are supported. The learning rate decays by default after each epoch, but by controlling the `SGD.time` parameter this can be changed. Likewise, the momentum term can be added and the strength can be controlled. The gradient of the cost function is calculated using `autograd` with respect to the first cost function argument. This argument is the weights of logistic regression or the biases for NN. In the NN implementation, the `SGD.delta` class variable is of interest and the `SGD.weights` class variable is used as a `tuple` to provide all parameters of the cost function.

III.II Logistic Regression

The logistic regression is organized in a class `LogisticRegression`. The method `LogisticRegression.fit(`$\boldsymbol{X}$,$\boldsymbol{t}$`)` handles fitting of the decision model to an input data $\boldsymbol{X} \in \mathbb{R}^{N\times(f+1)}$ and targets $\boldsymbol{t} \in \mathbb{R}^{N}$. It is expected that in the data one column is filled with 1, otherwise no bias will be fitted. Because the class `LogisticRegression` supports fitting to multiple classification classes, a one-hot encoding is inferred from the unique labels in $\boldsymbol{t}$. In a one-hot encoding, each class gets it own column in $\boldsymbol{t}_{\text{one}-\text{hot}} \in \mathbb{R}^{N\times K}$ and is set to one if the target is labeled as this class otherwise its set to zero. If there is the chance, that one class is not present in the training date due to low occurrence of the class in the total data set, it is possible to pass the one-hot encoding dictionary in the constructor of the class. Moreover, the needed weights $\boldsymbol{W} \in \mathbb{R}^{(f+1)\times K}$ are inferred from the dimensions of the input data.

The cross-entropy is then minimized by calling the `SGD` method from section III.I. Additionally, l_1 or l_2 norm regularization on the weights can be imposed with a given strength. All relevant inputs for `SGD`, like learning rate, epochs and mini-batch size is passed to the constructor of `LogisitcRegression`. After the `SGD` finished, the performance is evaluated on the training data set. Likewise, the final model can be tested on a test data set with `LogisticRegression.evaluate(`$\boldsymbol{X}$,$\boldsymbol{t}$`)`. Besides the MSE and the R2 score, the

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \tag{14}$$

is computed, where I is the indicator function, 1 if $\boldsymbol{t} = \boldsymbol{y}$ and 0 otherwise evaluated element wise. Here $\boldsymbol{y} = \text{CL}\left[\text{argmax}(\boldsymbol{X}\hat{\boldsymbol{W}})\right] \in \mathbb{R}^{N}$ represent the outputs of the logistic regres-

sion.

A better evaluation score for unbalanced data sets, where the occurrence of the different classes is not balanced, is the confusion matrix [2]. Important concepts are the definitions of true positive tp (correctly classified), false positive fp (incorrectly classified), false negative fn (incorrectly unclassified) and true negative tn (correctly unclassified). Then, the models

$$\text{precision} = \frac{tp}{tp+fp}, \tag{15a}$$

$$\text{recall} = \frac{tp}{tp+fn} \quad \text{and} \tag{15b}$$

$$\text{specificity} = \frac{tn}{tn+fp}. \tag{15c}$$

Finally, the accuracy from eq. (14) is

$$\text{accuracy} = \frac{tp+tn}{tp+tn+fn+fp}.$$

This matrix can be constructed with the method `confusion_matrix(`$\boldsymbol{X}$`,`$\boldsymbol{t}$`)`. This matrix allows a deeper and more differentiated insight into the classification performance of the model.

Last but not least, the method `LogisticRegression.predict(`$\boldsymbol{X}$`)` can be used, to predict classes on new data. The user can choose between the raw class probabilities or the most likely class as output. In the authors opinion, it is good conduct of machine learning usage to at least review the class probabilities, not only the most likely class, in order to proceed with further decisions.

III.III Neural Networks

The neural network is organized in a class `NeuralNetwork`. `NeuralNetwork.training` performs feedforward and backpropagation for the set number of epochs, or if by reaching the tolerance condition. `NeuralNetwork.accuracytest` calculates the standard deviation of the last five accuracy scores. If the standard deviation is less than the tolerance value, the function returns True and the network stops training. A similar features is implemented for the MSE-score when running the NN with regression.

The biases are initialised as $b \in \mathbb{N}(\mu = 0, \sigma^2 = 1)$. The weights are initialised as $w \in \mathbb{N}(\mu = 0, \sigma^2 = 1)/\sqrt{n}$, where n is the number of input nodes or nodes in the previous layer. This initialisation by dividing by $\sqrt{n}$ results in the network learning faster and prevents the nodes from saturating to either 1 or 0. This can be show in the Appendix 14.

III.IV Data sets

III.IV.1 classification

Classifications uses the data set from [3]. This data set has 23 features which are closer described in [3] and [6]. In total it contains 30000 samples without any missing values [1]. The target is to predict weather a credit client will default '1' or not '0' on his payment. The relative frequency of default is 22.12% within the data set.

To investigate, how the target depends on the different features, it is useful to look at the correlation matrix in fig. 2. The blocks of high correlations are related to features which are the same measurement but at different times. The target is strongly correlated to the payment history and strongly anticorrelated to the credit card limit. Moreover, it is weakly anticorrelated with the history of previous amount of payments. It is expected, that the other features with low correlation do not contribute strongly to the classification accuracy.

Each feature in the credit card data set is scaled, such that the maximal value is one and the minimal value is zero. Moreover, there is the possibility to drop samples which correspond to non-defaulting targets, until the relative frequency of default is 50%.

III.IV.2 Regression

Terrain data from Yellowstone National Park was used to test how well the NN model regression cases with high complexity. The terrain data was obtained from `https://earthexplorer.usgs.gov/`, and preprocessed as described in our previous paper (cite our article). The data is normalized so that the mean value (μ) is equal to zero, with heights in the range of [-1,1].

The terrain being analyzed is highly varied, containing rivers, lakes, ridges and mountaintops (see Appendix VII.IV). Therefore, the challenge lies in finding a set of hyperparameters and a NN-structure that is able to describe such a varied set of data. The model is optimized by trial and error, choosing different hyperparameters (γ, λ, η) and trying a diverse set of batch sizes, layers and nodes for the NN for polynomial orders 50-100 (spacing 10). A final run is performed for polynomial orders 0-130 (spacing 5).

[1] Possibilities for wrong values cannot be ruled out. A detailed analysis of the data set can be found in [5].

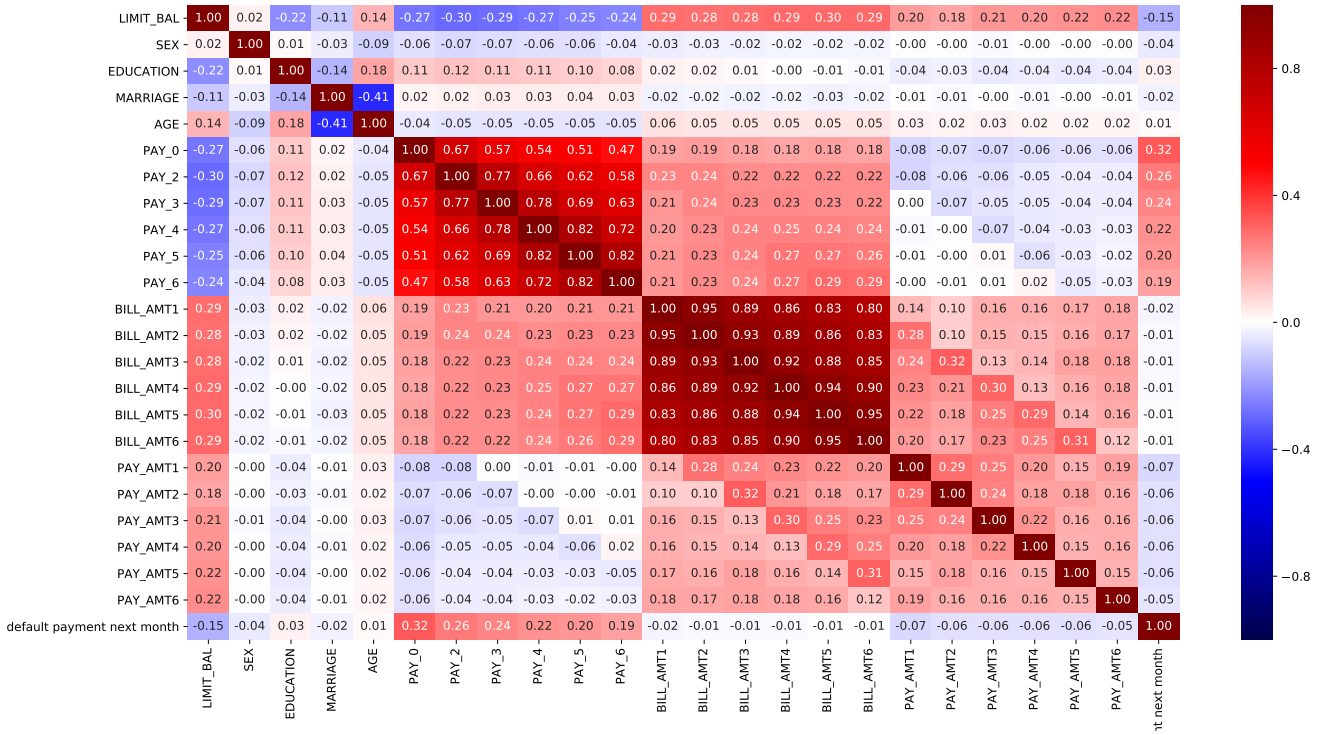


Figure 2: **Correlation Matrix**: The correlation matrix of all columns within the credit card data set are presented. The last line represents the target column.

IV Results

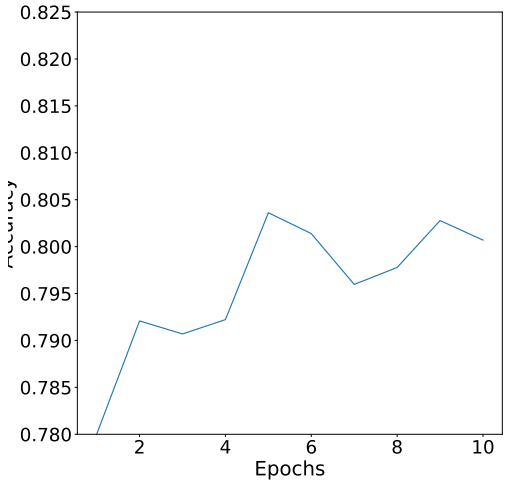
IV.I Stochastic Gradient Descent

The stochastic gradient descent is applied to the logistic regression of the credit card data set. The effects of a constant and decaying learning rate, as well as the effects of the momentum are studied in fig. 3. Because of the randomness in the SGD, the pictures are not directly comparable. However, the impact of the different options is visible.

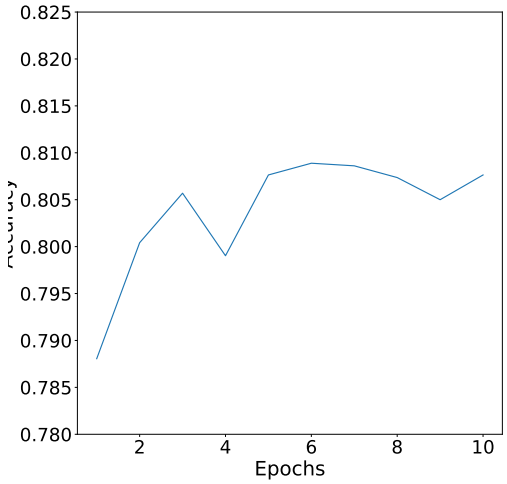
IV.II Classification

The logistic regression and neural network achieves in its best hyper parameter setting an overall accuracy of 80.6% and 81.3% respectively. A more in-depth evaluation of the logistic regression and neural network model is presented in tab. 2 and tab. 3 respectively. All results are obtained from a test subset of the credit card data which the model never has seen during training. Moreover, results are 10 times cross-validated with different training and test data subsets.

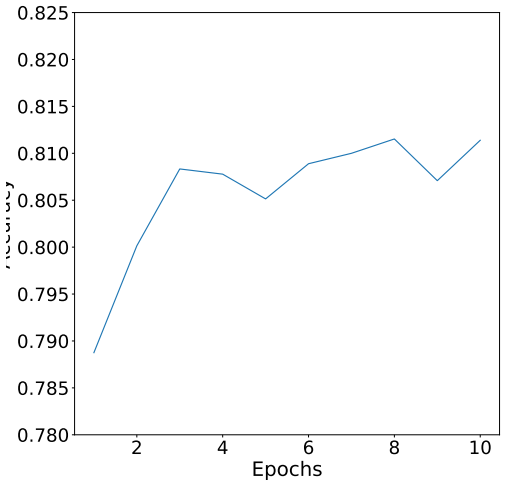
The impact of different hyperparameter settings for learning rate decay and momentum in the stochastic gradient descent can be seen in fig. 3. Details on the search and results of the hyperparameter optimization can be found in appendix VII.I for logistic regression, and in appendix VII.II for neural networks .



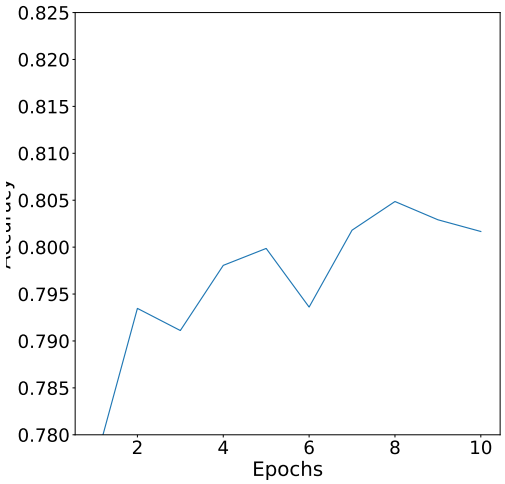
(a) constant learning rate, no momentum



(b) constant learning rate, with momentum



(c) decaying learning rate, no momentum



(d) decaying learning rate,with momentum

Figure 3: **Stochastic gradient descent learning rate and momentum**: Illustrated are the differences in stochastic gradient descent with constant and decaying learning rate with beginning amount of 0.05. Likewise, the momentum strength is varied between 0 (no momentum) and 0.1. The remaining parameters (except epoch) are set like in tab. 2b.

Table 2: **Confusion Matrix**: Comparison of the Confusion matrix between the original data (b) and the enhanced data (a). The hyperparameter are taken from an grid search (see App. VII.I) of the hyperparameter space. Results are 10-times cross validated.

(a) The confusion matrix for an balanced input data set. The learning rate is constant at 0.5. The model is trained in 40 epochs with a batch size of 10.L1 is used as regularization with a strength of 10^{-9}.The overall accuracy of the own Implementation (SGD) is 0.651, of the exact `scikit` Logisitc regression is 0.669 and of the SGD `scikit` implementation is 0.641, all on the balanced data set. The following confusion matrix is evaluated on the **original** data set.

actual class predicted class	0	1	precision	recall	specificity	accuracy
0	15073.1	2657.8	0.853	0.807	0.500	0.739
1	3613.6	2655.5	0.504	0.500	0.807	0.739
occurrence	0.779	0.221				

(b) The confusion matrix for an unbalanced input data set. The learning rate is constant at 0.5. The model is trained in 1 epochs with a batch size of 10.L2 regularization is used with a strength of 0.001.The overall accuracy of the own Implementation (SGD) is 0.806, of the exact `scikit` Logisitc regression is 0.808 and of the SGD `scikit` implementation is 0.807

actual class predicted class	0	1	precision	recall	specificity	accuracy
0	4481.6	971.9	0.823	0.958	0.266	0.806
1	194.4	352.1	0.664	0.266	0.958	0.806
occurrence	0.779	0.221				

Table 3: **Confusion Matrix II**: Comparison of the Confusion matrix between training the original data (b) and the enhanced data (a). The hyperparameters are taken from a grid search (see App. VII.II) of the hyperparameter space. Results are 10-times cross validated.

(a) The confusion matrix for a balanced input data set (There is an equal number of each class during training). The learning rate is decay from 0.3. The model is trained in 100 epochs with a batch size of 50. L2 is used as regularization with a strength of 10^{-2}. The following confusion matrix is evaluated on the **original** data set. The overall accuracy is 0.658. The network never predicts the credit card holder defaults. The `tensorflow` implementation reaches an accuracy of 67.2%.

actual class predicted class	0	1	precision	recall	specificity	accuracy
0	23364.0	6636.0	0.779	1.0	0.0	0.779
1	0.0	0.0	NaN	0.0	1.0	0.779
occurence	0.7788	0.2212				

(b) The confusion matrix for the whole (unbalanced) input data set. The learning rate is set to decay starting from 0.3. The model is trained in 100 epochs with a batch size of 50. L2 is used as regularization with a strength of 10^{-2}. The overall accuracy is 0.811. The `tensorflow` implementation with comparable hyper parameters achieves an average accuracy of 82.1%.

actual class predicted class	0	1	precision	recall	specificity	accuracy
0	23257.0	6462.0	0.783	0.995	0.026	0.781
1	107.0	174.0	0.619	0.026	0.995	0.781
occurence	0.7788	0.2212				

The neural network accuracy achieved (0.81) with the best hyper parameters is consistent with [6]. This network is also not over-fitting the training data as shown in Fig.4. Using the enhanced data set (balanced) the best accuracy (0.66) is much worse than the training on the orginial data set. As can be seen in tab.3a, the network never makes a prediction of class 1 (default).

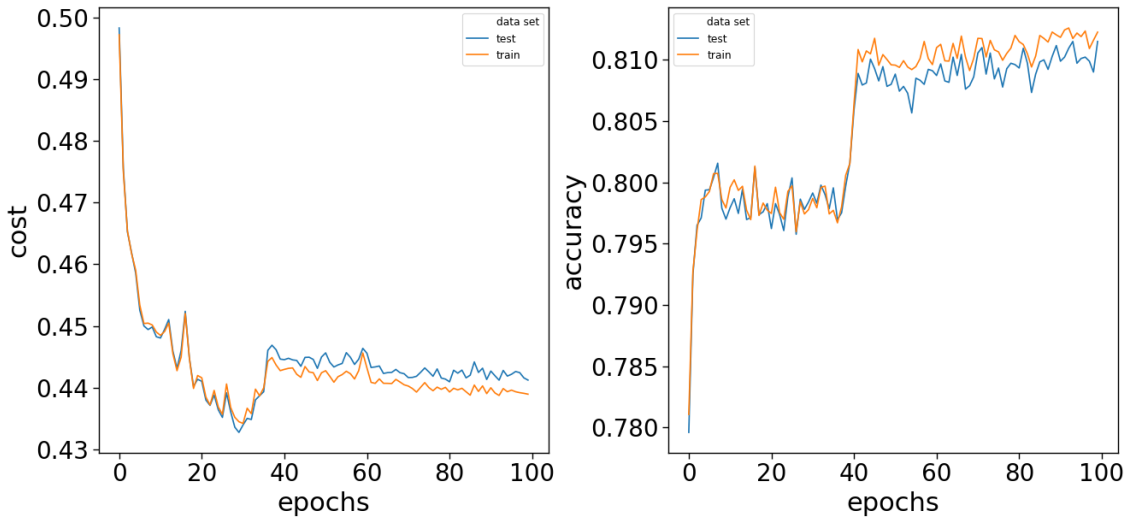
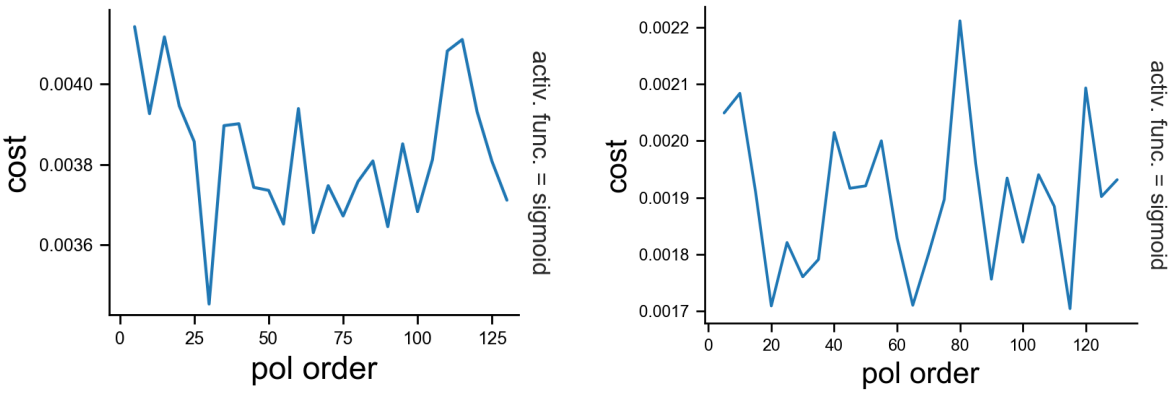


Figure 4: Accuracy and cost as a function of epochs with best hyper parameters. Test and train accuracy do not diverge showing that the network is not over-fitting. Testing sometimes performs better than training as seen in cost.

IV.III Neural network modelling of Yellowstone terrain data

After doing a grid search for the best hyperparameters and NN-structure, the optimal model was chosen, and run for polynomial orders between 0-130 (5 pol. order spacing). Among γ, η, and λ, the only hyperparameter having a significant impact on the MSE-score is the initial learning rate (η), with $\eta = 0.3 - 0.4$ giving the lowest MSE (Appendix: Fig. 15 and 16). Optimal batch sizes varies between polynomial order, for example p = 50, a batch size of 150 gives the best MSE-score, in contrast with p = 90, where a batch size of 50 give best cost scores.

The MSE-cost for test and validation sets is shown for the range of polynomial orders in fig. 5 for the sigmoid activation function. With tanh as activation function, the MSE-score displays the exact same behaviour, for both test and validation set (Appendix: fig. 18).



(a) Test

(b) Validation

Figure 5: Cost (MSE) for regression of Yellowstone terraindata as a function of polynomial order with best hyper parameters ($\eta = 0.4$, $\gamma = 0.9$, $\lambda = 1e-6$, batch size $= 50$) with sigmoid activation function. Structure of NN: (number of features in design matrix,40,1), i.e one hidden layer with 40 nodes.

The lowest cost score is obtained with polynomial order 115 (MSE:1.705E-3) for validation, while polynomial order 20 and 65 almost give the same cost score (MSE: 1.710E-3 and 1.711E-3 respectively) (5). However, lowest MSE-score for test set is 3.45E-3 with polynomial order 30. The ADAM-optimizer gave an average MSE-score of 4.5E-3 from 10 runs, each with 5 epochs, for the same NN-structure (one hidden layer).

The reduced terrain data and fitted model, using the lowest test MSE (pol order 30), is presented in fig. 6. The details in the heat map and the MSE-score indicate how well the optimized model from the NN fits the original terrain data.

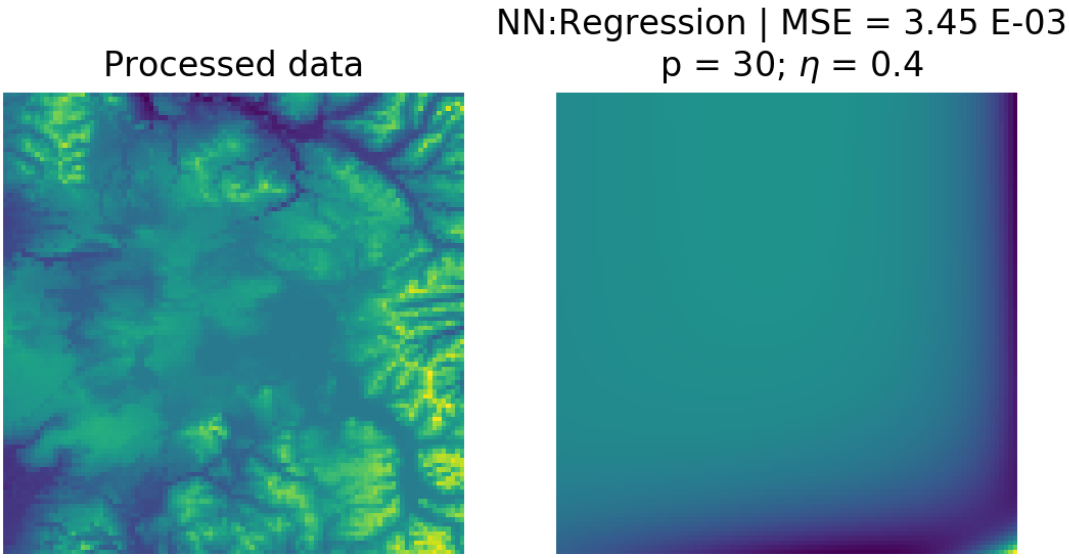


Figure 6: Heat map for elevations (blue (low) - orange(high)) in a real landscape from Yellowstone National Park (reduced with sel = 36) and from the best fit for the given data with neural network (η: Initial learning rate, p: Polynomial order, γ: Momentum = 0.9, λ: Weight penalty= 1e-6, K = 10). No cost regularisation.

V Discussion

V.I Stochastic Gradient Descent

In general the SGD is controlled mainly by the learning rate and momentum. In fig. 3 the differences becomes visible. A constant learning rate leads to more fluctuating results, but gives faster improvement in the beginning. Another way to speed up improvements, is to increase the magnitude of the learning rate. Then, on the other hand the magnitude of the fluctuation is expected to become larger (not shown here). Another way to speed up improvement is to use momentum. Especially, in combination with a decaying learning rate, which leads to a smoother improvement curve 3c, momentum can lead to fast and reliable convergence. This can be seen in fig. 3d.

V.II Classification

The 80.8% accuracy of the logistic regression model on the test subset of the credit card data seems to be acceptable, but not perfect. However, when the accuracy is set into perspective of the 77.88% percent of non defaulting credit card payment targets, it becomes clear that the logistic regression model performs not significantly better than randomly deciding whether a client will default or not based on the 77.88% chance of not defaulting.

Compared to the reported accuracy of 82% in[6], this results are slightly less accurate. The differences might arise from different optimization techniques for the logistic regression or a different data preparation. The paper does not describes all details of the methods to pin down the differences.

A more in depth insight on the performance of the model gives tab. 2b. The model can predict non defaulting clients with a precision of 82.3% and a recall of 95.8%. In other words, the model can identify non-defaulting clients acceptably, but performs worse when predicting payment defaults. This can be inferred from the low specificity of 26.6%.

The ad-hoc idea, is to reduce the number of non-defaulting clients in the data set until both cases of defaulting or not are equally present in the data. It is expected, that the model can achieve a better representation of defaulting clients when there are the same amount of defaulting and non-defaulting examples in the data.

This crude solution leads to an accuracy of 66.9% on the reduced data set. However, as tab. 2a indicates, the accuracy on the original data set is only 73.9%. This means, that the model does not have an accurate internal representation of defaulting clients which is transferable to the original data. Even though precision for predicting defaulting clients increase slightly, the recall of this class of clients drops by a large amount. Specificity almost doubles but in the same time the precision in predicting defaulting clients drops to 50%. To conclude, reducing the amount of non-defaulting clients in the training data gives worse predictions than training on an unbalanced training set. Maybe, artificially enriching the data set with defaulting clients is an option, but this was not tested here.

The 81.1% accuracy achieved by the neural network does not seem satisfactory, considering that 77.88% of credit card holders are not defaulting on their loans. As with the logistic regression, the model does not perform much better than randomly predicting default or not default.

The number of non-defaulting clients was reduced to train on a data set with equally occurring defaults and non-defaults. This resulted in 65.8% accuracy on the data set. How-

ever, as seen in tab.3a the model does not have a representation of the defaulting clients at all. The model did not predict correctly or incorrectly any clients would default on their loan. Reducing the number of non-defaulting clients has not improved the model prediction.

V.III Regression

Hyperparameters and NN-structure were tweaked to find the optimal model for the Yellowstone terrain data. Neither λ, γ or number of hidden layers impacted the MSE-score significantly, while there was a general trend that η and batch size had an impact on the MSE score (fig.15 and 16). Since the complexity is high in the terrain data, penalties on the weights and momentum to keep track of the direction wouldn't serve much purpose, which was verified during grid searches.

The two activation functions seem to perform equally well in fitting the model (Appendix: fig. 18), meaning that the small differences in the functions' behavior doesn't have any effect with this data set. That number of hidden layers doesn't impact the outcome, might indicate that either the implemented NN has problems with deep learning, or is unable to learn better with several layers when the activation function is the same between all hidden layers.

This might have consequences for the MSE-scores, since a lower validation cost than test cost, is a symptom of underfitting (might need a citation here). A possible solution would be to change activation functions between hidden layers, and increase the number of nodes and/or layers, due to the large size of the terrain data.

The general trend, that a low to medium learning rate gives the best MSE-scores can easily be explained by the higher fluctuation in predictive accuracy with a high learning rate ($\eta > 0.4$). However, what is not that obvious is why a batch size of 50 give lowest MSE-scores for polynomial order 90, whilst a batch size of 150 appears to be optimal for polynomial order 50.

Considering the central limit theorem (claiming that the expectation value of many experiments of independent identical events will approach the true value), the accuracy should increase with number of minibatches, which seem to not be the case. There is only a difference of 1E-4 in MSE-score between batch size 50 and 150 for $p = 50$ (Appendix: fig.16), but it is consistent among different NN-structures and activation functions, thus it should not be a consequence of the stochasticity in SGD.

Since the MSE varies between batch sizes for a specific η and polynomial order, it is difficult to know what batch size to choose for the final run. However, batch size 50 is cho-

sen, as it is most stable for different η values (Appendix; fig. 15 and 16) . From the test cost, a low polynomial order ($p = 30$) and no regularisation gives the lowest MSE (fig. 5), which is not in concordance with the results from linear regression on the same terrain data, in which high polynomial orders where favored (Appendix: fig. 17) (refer to our article).

In constrast to linear regression, where MSE-score generally decreased with polyonomial order (refer to our article), this seem not to be the case with NNs. The NN-method can thus not be easily compared to that of linear regression methods. Neither does NN seem better, or equally, suited to fitting models to complex two dimensional surfaces as linear regression, when comparing the two results (fig. 6 with fig. 17).

VI Conclusion

In this project, neural networks are used for regression and classification. Moreover, for classification, logistic regression is also used. The own implementations could produce similar results like state-of-the-art libraries like `tensorflow` and `scikitlearn`. However, these libraries seem to be a bit more reliable and faster, especially for neural networks.
Independent of the implementation, the challenge is not the implementation itself, but to find and tune the best model parameters. The conducted grid search seems rather inefficient and not satisfyingly cover the vast parameter space. Consequently, it is unclear if the chosen parameters are the optimal ones.

To test the classification abilities of logistic regression and neural networks, the credit card data set is used. Results strongly depend on the chosen data processing and accuracy on using only rescaled training data is around 81% for both neural networks and logistic regressions. However, the accuracy is mainly due to the models predicting non-defaulting credit card clients, which are the majority in the training data. This is unacceptable from a business perspective, because many defaulting clients are predicted as non-defaulting, and overall, the models predict defaulting clients seldom or never. Likewise, a potential business customer would not gain any insights about its credit clients, because the decision of the model is intransparent. This also applies to the researcher perspective, where it is unclear how the data influences the models decision.
To improve this situation, down-sampling techniques are tried, but are unsuccessful. Maybe bootstrapping defaulting clients could get better results, by getting a higher population of defaulting clients in the sample. In the same way, maybe data reduction techniques could improve results.

The neural network performing a regression task can compete with traditional regres-

sion methods like ordinary linear regression. For both approaches the MSE is roughly the same. However, visually the prediction of the neural network cannot reproduce the complex terrain. This, together with the fact that the best performing neural nets are not deep (only one or two layers), might be an indicator of a subtle problem in the implementation.

References

[1] Shifei Ding, Hui Li, Chunyang Su, Junzhao Yu, and Fengxiang Jin. Evolutionary artificial neural networks: a review. *Artificial Intelligence Review*, 39(3):251–260, Mar 2013.

[2] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.

[3] I-Cheng Yeh. default of credit card clients Data Set, 2016.

[4] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[5] Vladimir G. Drugov. Default Payments of Credit Card Clients in Taiwan from 2005.

[6] Yeh, I. C., Lien, C. H. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, pages 2473–2480, 2009.

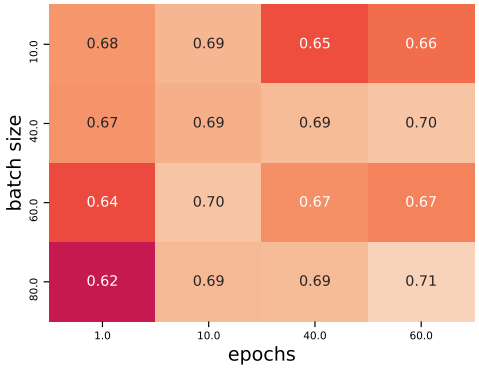
VII Appendix

VII.I Grid Search for Logistic Regression

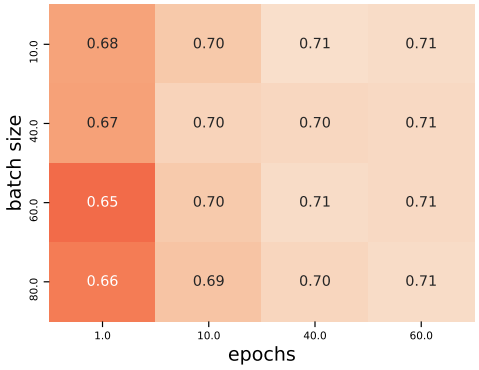
The search for the optimal hyperparameters for logistic regression is briefly described in the following. Additionally, some more results are presented inf fig. 7-8. For visual reasons, it is focused on the enhanced data set, but the same observations are found, although less visible, in the grid search on the raw data.

The grid search tries certain given values for each hyperparameter and their combinations. The maximal number of iterations per fit is set to 10^4 and the validation data size to 10%. Then the values 1, 0.5, 0.1 and 0.01 are set as the learning rate for a constant learning rate and as starting value for a decaying learning rate. The SGD runs between 1, 10, 40 and 60 epochs with batch sizes 10, 40, 60 and 80. Likewise, the regularization is changed between L1 and L2 regularization with a regularization strength λ between 10^{-9} and 10^{-3} in multiples of 1000.

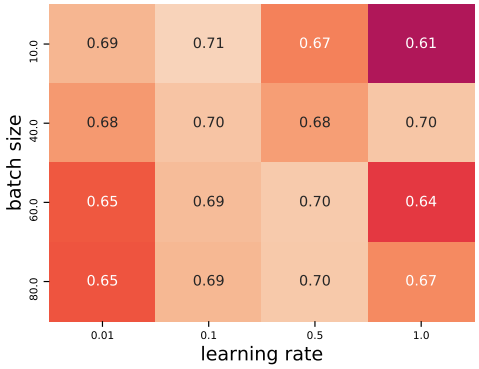
This method of finding hyperparameter is not satisfying, because the time consumption is enormous although the searched hyperparameter space is relatively small. Other methods, like random search, might be more effective. However, those methods are not used for the hyperparameters in logistic regression.



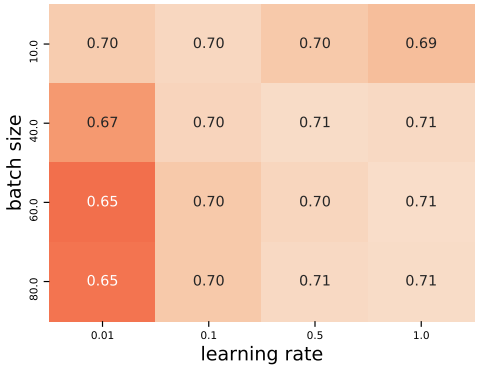
(a) Accuracy as function of epochs and batch size. Trained with constant learning rate.



(b) Accuracy as function of epochs and batch size. Trained with decaying learning rate.

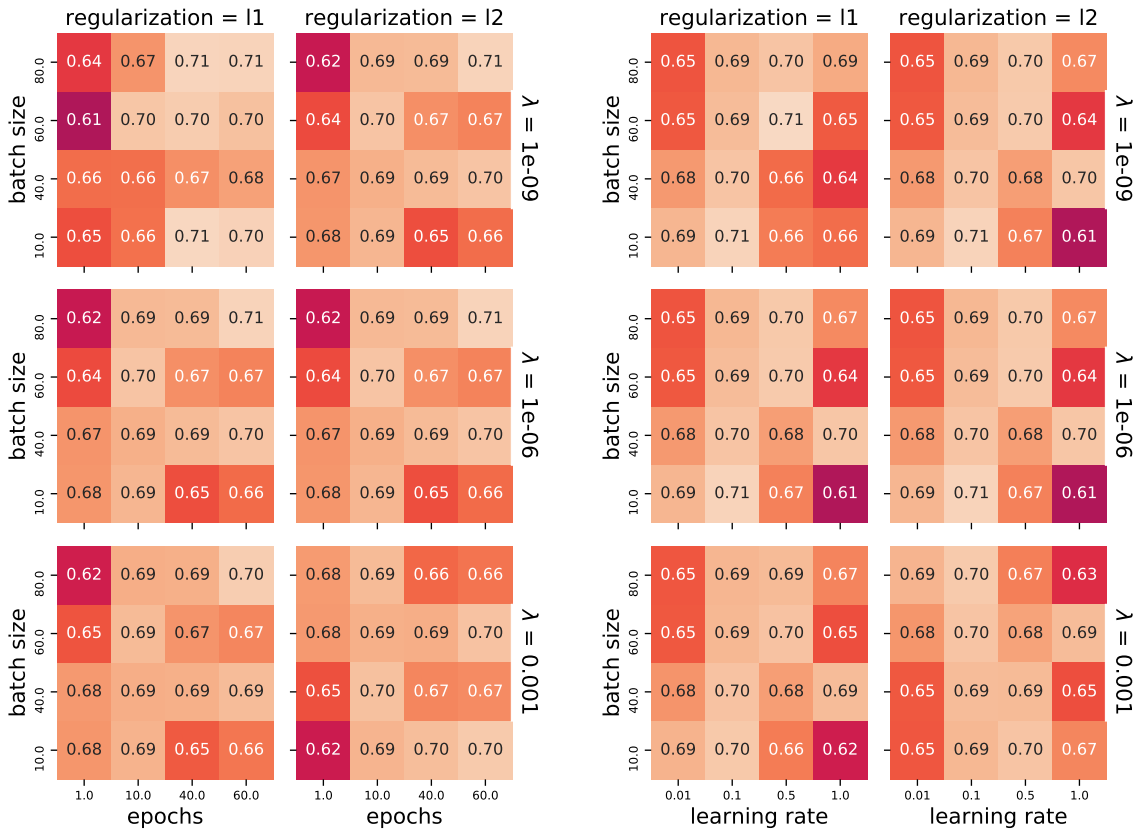


(c) Accuracy as function of learning rate and batch size. Trained with decaying learning rate.



(d) Accuracy as function of learning rate and batch size. Trained with decaying learning rate.

Figure 7: **Grid Search I**: No regularization is imposed on the parameters of the logistic regression. The enhanced data set is used. Over not shown hyperparameter dimensions is averaged.



(a) Accuracy as function of epochs, batchsize, regularization and regularization strength. The learning rate is constant.

(b) Accuracy as function of learning, batchsize, regularization and regularization strength. The learning rate is constant.

Figure 8: **Grid Search II**: Regularization is imposed on the parameters of the logistic regression. The enhanced data set is used. Over not shown hyperparameter dimensions is averaged.

VII.II Grid Search for Neural Network Classification

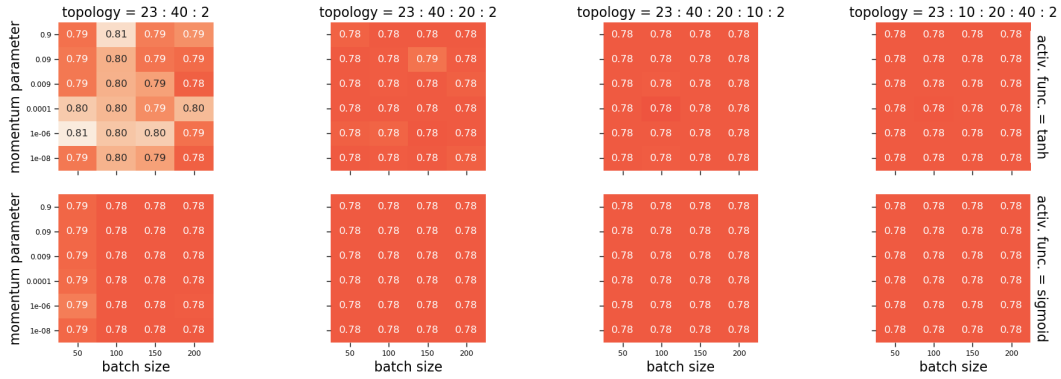


Figure 9: **Grid Search I**: Search for optimal parameters in neural network for classification. Accuracy as a function of batch size, topology of the network, activation function and momentum parameter. A single hidden layer using the tanh activation function yields greatest accuracy (0.81). The accuracy (0.81) resulted from batch sizes 50 and 100 and γ= 1e-6 and 0.9 respectively. Other parameters are fixed, λ=1e-4, learning rate η=0.3 is set to decay. Epochs ended after reaching the tolerance condition.

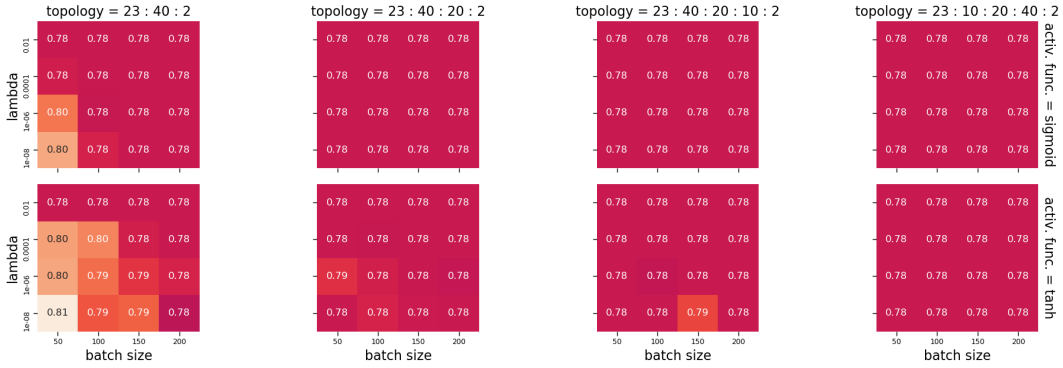


Figure 10: **Grid Search II**: Search for optimal parameters in neural network for classification. Accuracy as a function of batch size, topology of the network, activation function and regularisation parameter λ. The greatest accuracy (0.81) is achieved with: a single hidden layer, the tanh activation function, batch size 50 and λ = 1e-8. Other parameters are fixed, $\gamma = 0.9$, learning rate $\eta = 0.3$ is set to decay. Epochs ended after reaching the tolerance condition.

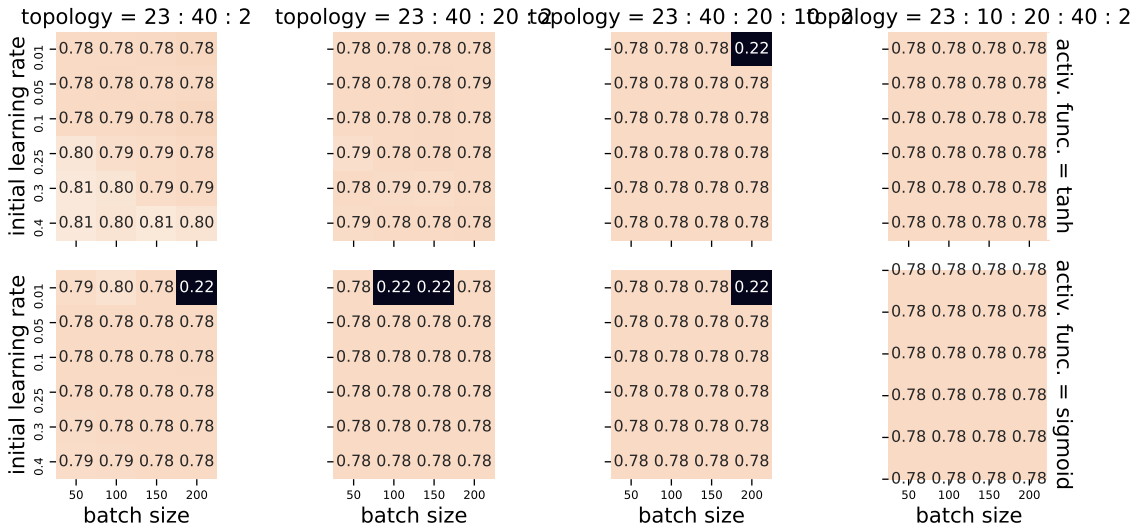


Figure 11: **Grid Search III**: Search for optimal parameters in neural network for classification. Accuracy as a function of batch size, topology of the network, activation function and initial learning rate η. For η's in the range $0.01 - 0.4$, the greatest accuracy (0.81) is achieved with: a single hidden layer, the the tanh activation function, batch sizes 50 and 150, $\eta = 0.8$. Be aware that only initial η's are shown in the graphs, and it is set to decrease with iterations. Other parameters are fixed, λ=1e-8 and momentum γ=0.9. Epochs ended after reaching the tolerance condition.

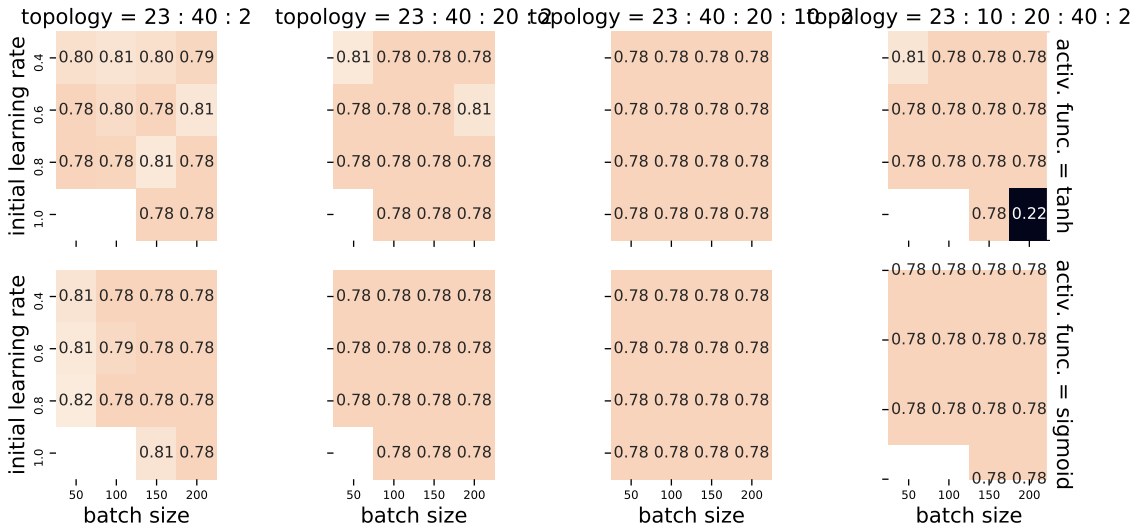


Figure 12: **Grid Search IV**: Search for optimal parameters in neural network. Accuracy as a function of batch size, topology of the network, activation function and initial learning rate η. The greatest accuracy (0.82) is achieved with: a single hidden layer, the sigmoid activation function, batch size 50, $\eta = 0.8$. However $\eta = 0.4$ gives an accuracy of $0.80 - 0.81$ for batch sizes 50, 100 and 150, with tanh as activation function. $\eta > 0.8$ results in no accuracy scores, revealing numerical instability for small batch sizes. Be aware that only initial η's are shown in the graphs, and it is set to decrease with iterations. Other parameters are fixed, λ=1e-8 and momentum γ=0.9. Epochs ended after reaching the tolerance condition.

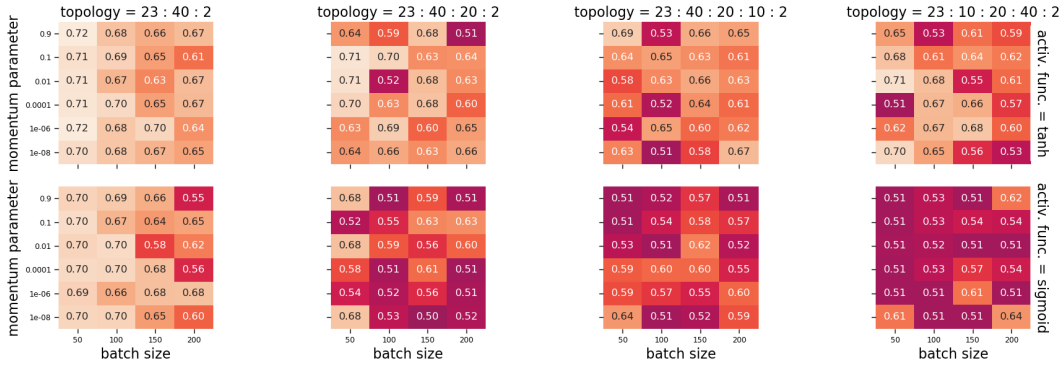


Figure 13: **Grid Search V**: Search for optimal parameters in neural network using an enhanced (balanced) data set. Accuracy as a function of batch size, topology of the network, activation function and momentum parameter γ. The prediction accuracy is worse than the whole (unbalanced) data set. The greatest accuracy (0.72) is achieved with: a single hidden layer, the tanh activation function, batch size 50, γ = 1e-6 and 0.9. Other parameters are fixed, λ=1e-8, learning rate η=0.3 is set to decay. Epochs ended after reaching the tolerance condition.

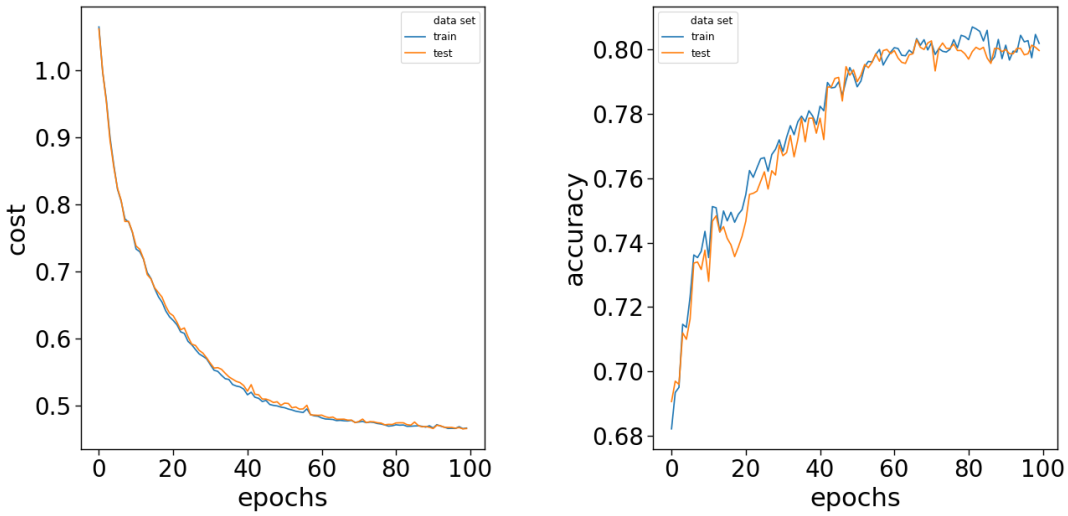


Figure 14: Cost and accuracy as a function of epochs. The weights have not been initialised by dividing by $\sqrt{n}$, where n is the number of nodes in the previous layer. The model trains more slowly than compared with Fig.4.

VII.III Grid Search for Neural Network Regression

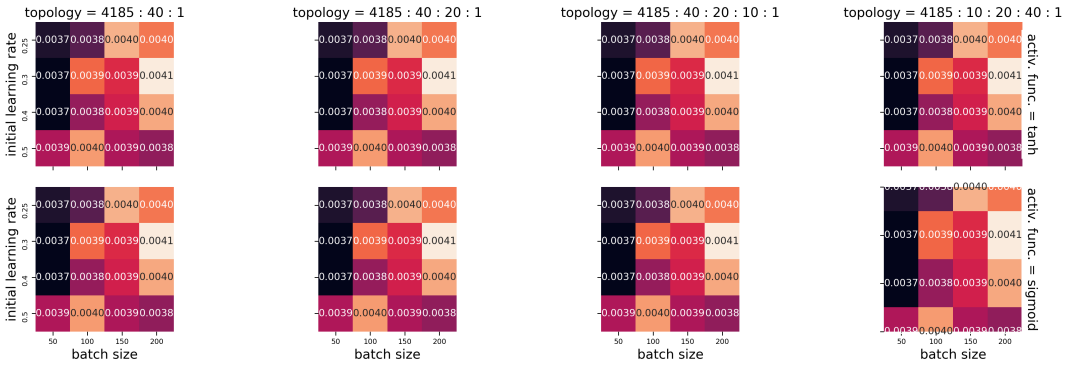


Figure 15: **Grid Search I**: Search for optimal parameters in neural network for regression. MSE as a function of batch size, topology of the network, activation function and initial learning rate η. The MSE is smaller than two decimal numbers. Judging from the heat map, the lowest MSE for polynomial order 90 is obtained by: a single hidden layer,batch size 50, η = 0.3 and 0.4. The activation functions, tanh and sigmoid, seem to perform approximately equally with regards to batch size and initial learning rate. Other parameters are fixed, λ=1e-8 and momentum γ=0.9. Be aware that learning rate is set to decay

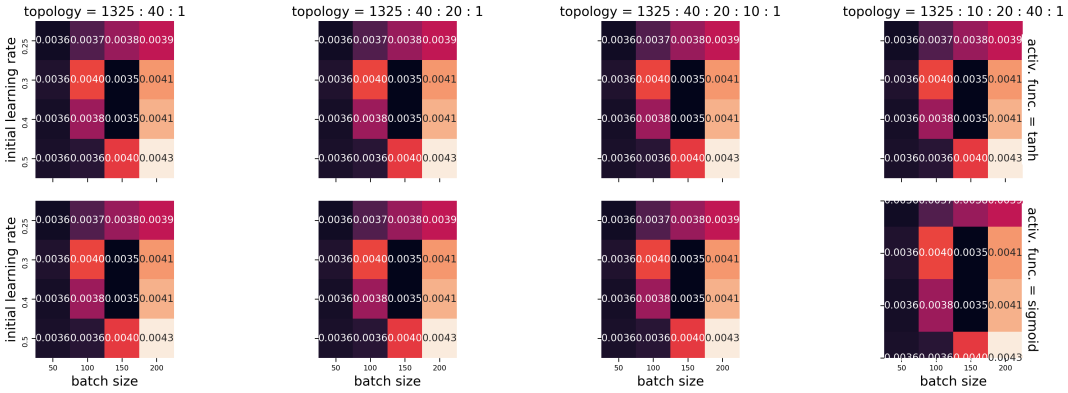


Figure 16: **Grid Search II**: Search for optimal parameters in neural network for regression. MSE as a function of batch size, topology of the network, activation function and learning rate η. The MSE is smaller than two decimal numbers. Judging from the heat map, the lowest MSE for polynomial order 50 is obtained by: a single hidden layer, batch size 150, η = 0.3 and 0.4. The activation functions, tanh and sigmoid, seem to perform approximately equally with regards to batch size and initial learning rate. Other parameters are fixed, λ=1e-8 and momentum γ=0.9 Be aware that learning rate is set to decay.

VII.IV Results of Yellowstone Terrain modelling with linear regression

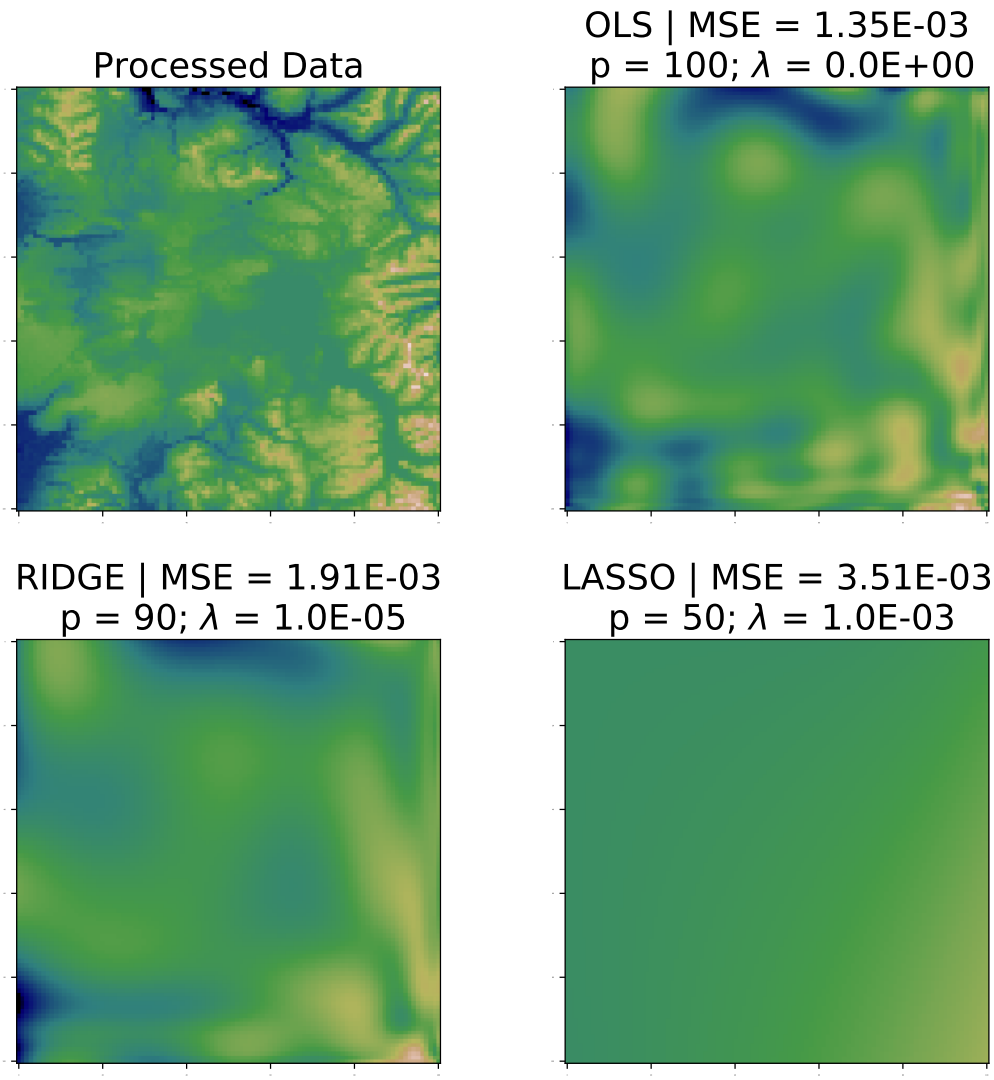


Figure 17: **Results from linear regression methods**: Heat map for elevations (blue (low) - orange(high)) in a real landscape from Yellowstone National Park (reduced with sel = 36), and for three regression models with best fit for the given data (λ:Regularization parameter, p: polynomial order, K = 5).

VII.V Results of Yellowstone Terrain modelling with Neural Network

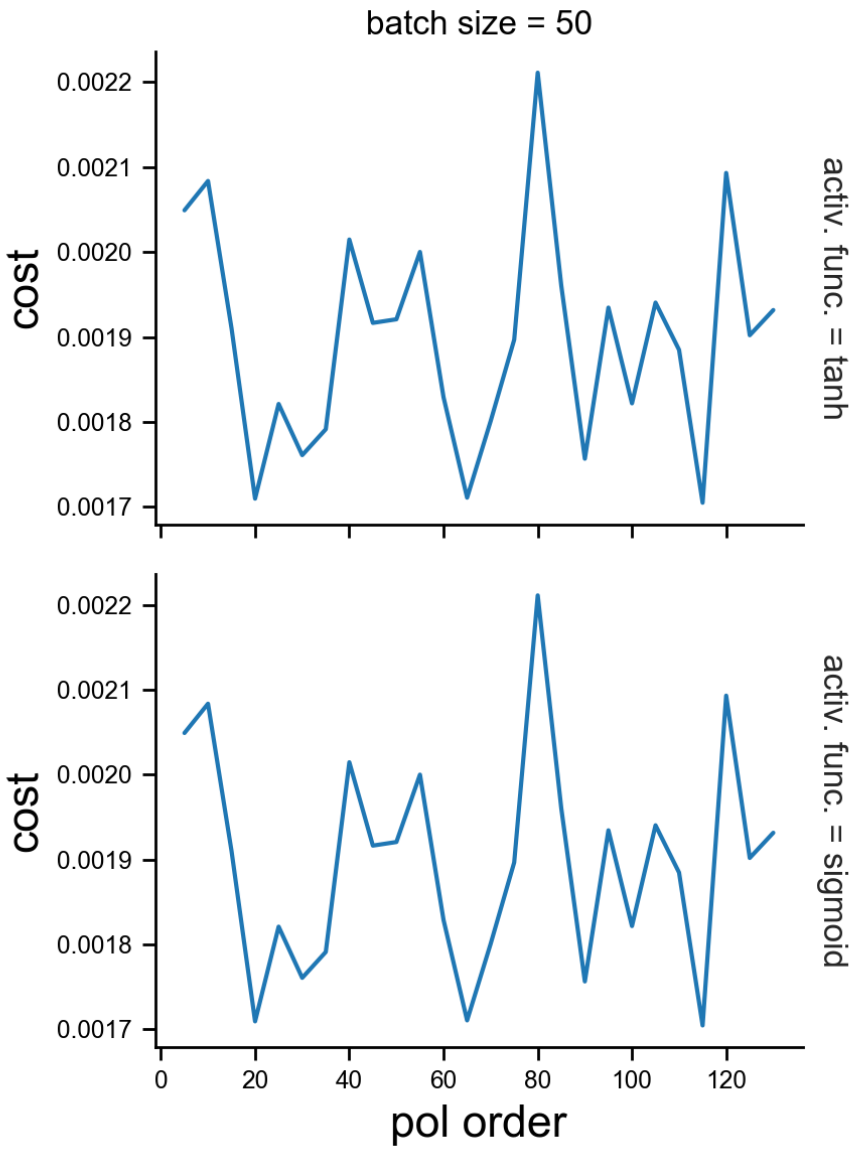


Figure 18: Cost (MSE) for validation set from regression of Yellowstone terraindata as a function of polynomial order with best hyper parameters ($\eta = 0.4$, $\gamma = 0.9$, $\lambda = 1e-6$, batch size $= 50$). Sigmoid and tanh activation functions gave the same cost-values.