Comparison between stochastic gradient decent and Resilient propagation training algorithms.

D.S. Geldenhuys 18564321

dylangeldenhuys@sun.ac.za

Abstract

Stochastic gradient descent is a popular learning algorithm which employs backpropagation to compute weight updates. There have been many algorithms used to explore alternative or adaptions of this algorithm to bypass some of its known disadvantages which include instability, slow convergence, and the problem dependent learning parameters which need to be optimised for each problem. This report provides a qualitative comparative analysis of stochastic gradient descent ("SGD") and the alternative learning algorithm called Resilient propagation ("Rprop") which is known to bypass some of the problems experienced with SGD. To evaluate the algorithms, we used a simple single layer Neural network whereby we optimise the number of hidden units for each algorithm as well as the learning parameters for SGD. We use three classification tasks and three regression tasks of varying complexity to compare the algorithms. The results of this study show that Rprop outperforms SGD on all learning tasks. It also proves to be efficient and easily implemented without the need to choose learning parameters.

Introduction

Background

SGD is one of the most widely used training algorithms. To achieve gradient descent, it makes use of partial derivatives of the error term with respect to each weight and uses a learning rate to adapt the step size. Using only learning rate and the partial derivatives for weight update steps is often very unstable. One solution proposed to solve this problem was momentum which considers previous steps which results in a more stable descent [1]. SGD is known to have slow convergence due to the partial derivatives (gradients) having small values and can stagnate around local optima. There have been other adaptive techniques that attempt to scale the learning rate using local weight specific information. Most of these methods including SGD are severely affected by the partial derivative with respect to the weight which can unforeseeably affect the step size. Martin Riedmiller *et al*. explains this effect as 'blurred adaptivity'[1]. SGD requires that one optimises each learning parameter since the optimal learning parameters are very problem dependent. Alternatively, Rprop provides a new weight update routine to remove some of these problems relating to the 'blurred adaptivity' and the need for parameter optimization [1]. Rprop does not consider the value of the gradient derivative at each weight step, instead it considers the sign of the error gradient to indicate the direction of the weight update. The update value slightly increased for every consecutive step that the sign of the partial derivative remains the same. When the sign changes it means the last update was too big and so the update is decreased by some factor. Rprop attempts to remove the effect of the small partial derivatives and 'blurred adaptivity' and also removes the necessity of finding

optimal learning parameters. The Rprop algorithm is said to have very little variation in performance when varying its hyper parameters, and so it is rare that they need to be changed from their default values [2].

Implementation

The models implemented in this report were done so using the python Pytorch library. The datasets used to evaluate the training algorithms where taken from the Sklearn built in databases. To compare the two training algorithms, a model was built for each data set of varying complexity. The models contained sigmoid activation functions for each hidden unit and a linear layer for each output where cross entropy loss was used for the classification tasks and mean squared error loss was used for regression problems. Pytorch implements SoftMax as part of the cross entropy loss during training and so the actual model for both regression and classification are identical apart from the datatypes used for the target variables (which tell Pytorch how to use them in the criterion).

For each dataset we create to two training algorithms: Rprop and SGD. The training loss and validation loss were plotted for varying learning parameters to decide on the optimal hyperparameters. The number of hidden units were chosen by observing the variation in accuracy for the classification problems and mean square error for regression problems. Once the hyperparameters had been chosen for each model, the SGD and Rprop algorithms were compared by plotting them together using accuracy for classification tasks and root mean square error for regression tasks.

Empirical procedure

The classification datasets used in the experiments are given below in table 1 along with the specifications of the data set.

The data sets were imported from Sklearns built in datasets and are described in their documentation [3].

Classification Data sets	Number of instances	Number of attributes	Number of classes	Class distribution
Iris	150	3	3	Uniform
Wine	178	13	3	59|71|48
Digits	5620	64	9	Uniform

Regression Data sets	Number of instances	Number of attributes	Number of outputs
Diabetes	442	10	1

Boston house prices	506	13	1
Linnerrud	20	3	3

Table 1

One of the key benefits of the Rprop algorithm is the quick convergence [2]. The hypothesis is thus that the Rprop algorithm will converge faster than SGD. To observe this empirically, we plot the performance of the algorithms after each epoch using accuracy metric for classification problems and root mean square error for regression tasks. Before the algorithms can be compared, the hyperparameters for SGD must be optimised. Since Rprop is said to be constant for most hyperparamters and rarely problem specific we will not optimise any hyper parameters for Rprop. For SGD, the parameters that need to be optimised are the *learning rate* and the *momentum.* To do so, we use an empirical trial and error approach where the final converged loss value was plotted for varying hyperparameters and the parameters which gave the lowest error where chosen. There was a total of twelve runs to achieve parameter optimisation for the SGD learning parameters; one run to estimate the range of optimal parameters for a given task and then another to decide on the best parameter in this range.

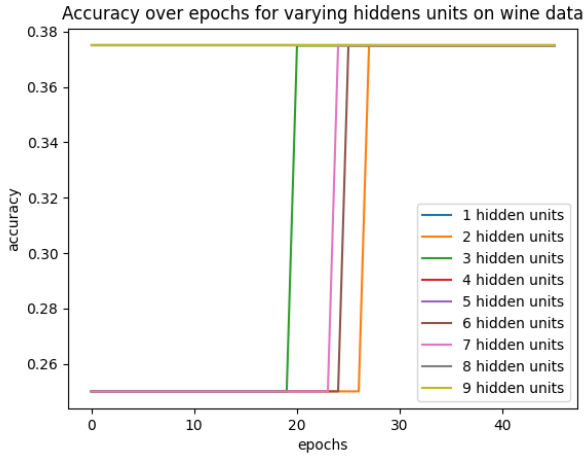
Most of the problems have even or close to even class distributions which makes accuracy an appropriate metric to evaluate performance for classification tasks. For regression tasks we use root mean square error which provides an indication of the spread of the datapoints around the function mapping.

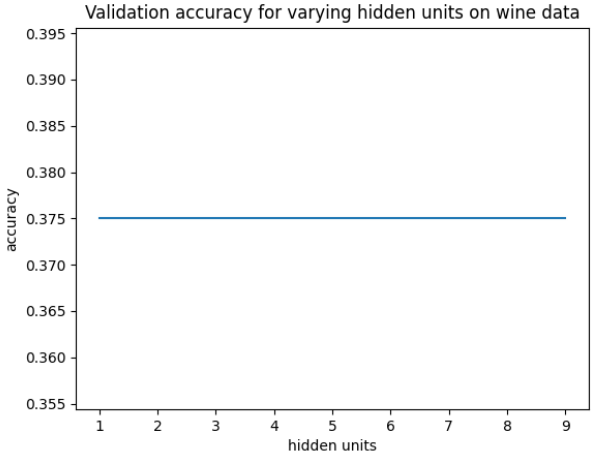
Research results

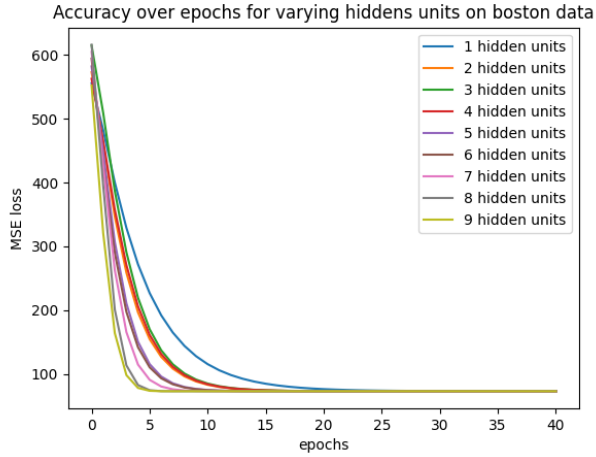
Parameter optimisation

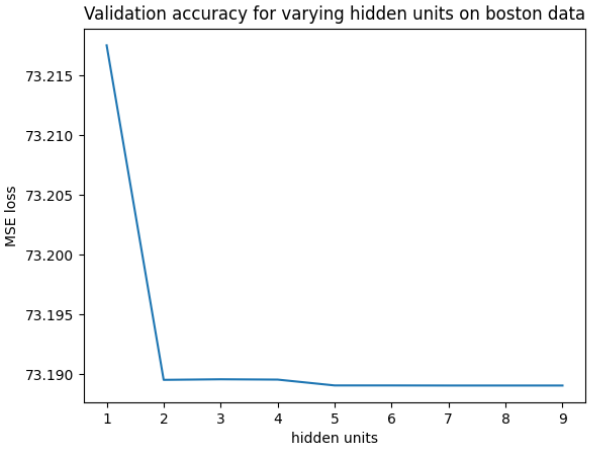
To obtain the optimal number of hidden units, we empirically decide on the number of hidden units for each training algorithm by plotting the accuracy per hidden unit. These results are given below.

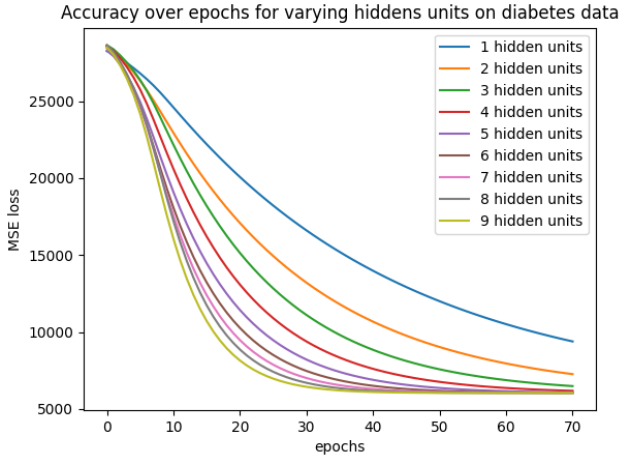
Hidden unit optimization for SGD

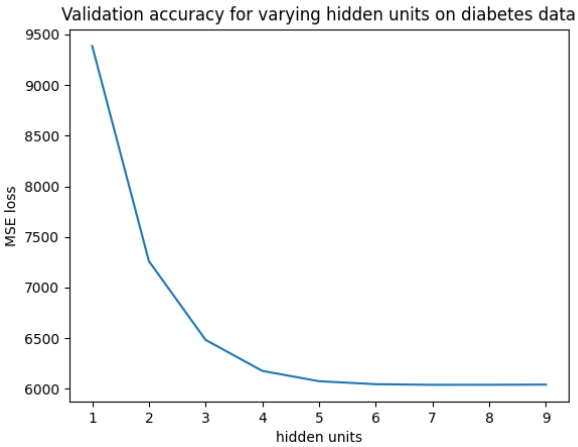


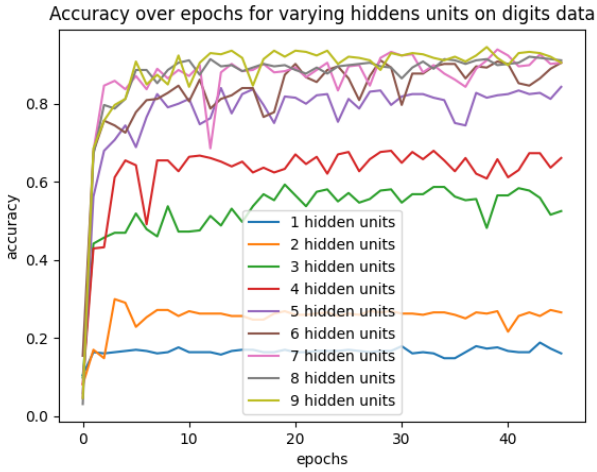


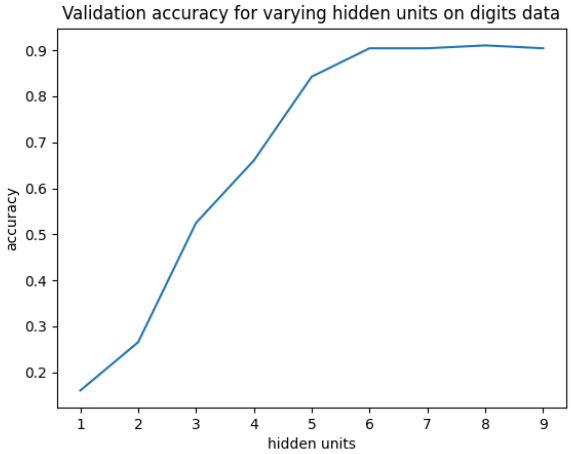




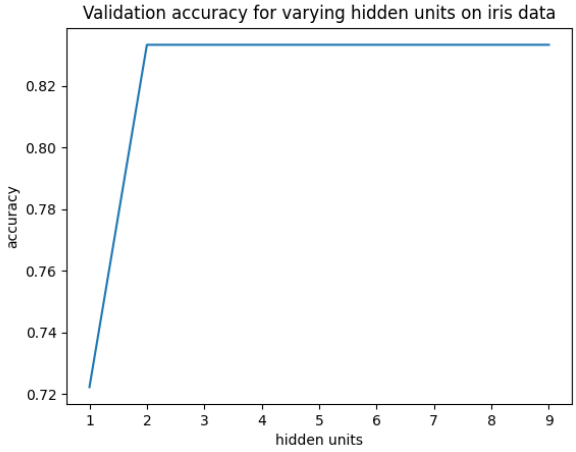


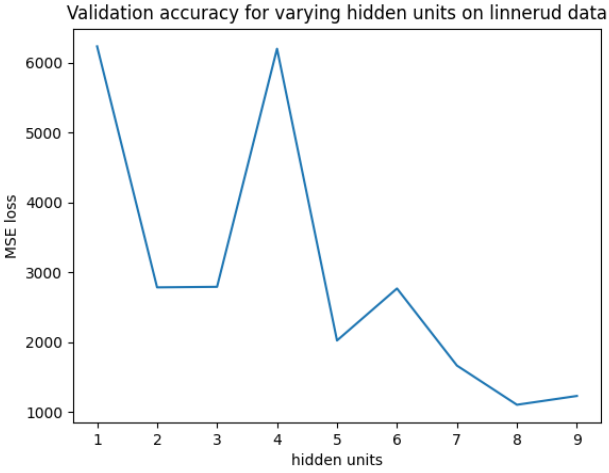


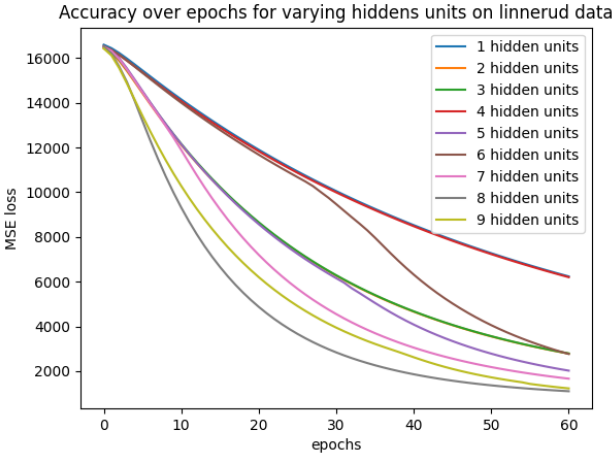

Accuracy over epochs for varying hiddens units on digits data
accuracy
epochs
1 hidden units
2 hidden units
3 hidden units
4 hidden units
5 hidden units
6 hidden units
7 hidden units
8 hidden units
9 hidden units


Validation accuracy for varying hidden units on digits data
accuracy
hidden units

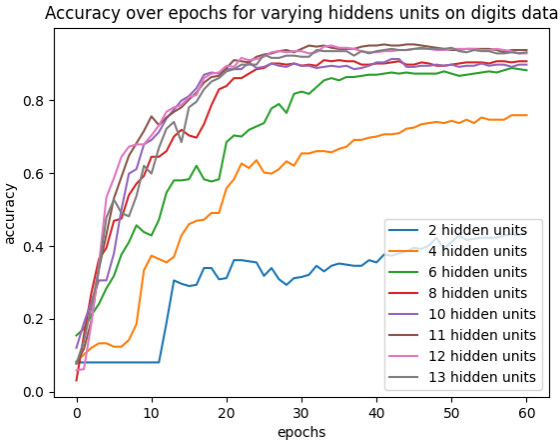

Accuracy over epochs for varying hiddens units on iris data
accuracy
epochs
1 hidden units
2 hidden units
3 hidden units
4 hidden units
5 hidden units
6 hidden units
7 hidden units
8 hidden units
9 hidden units

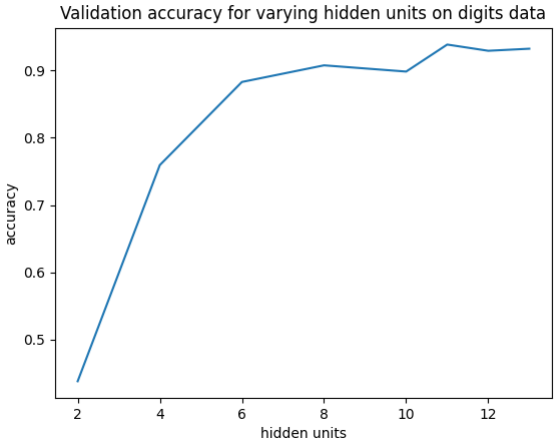

Validation accuracy for varying hidden units on iris data
accuracy
hidden units

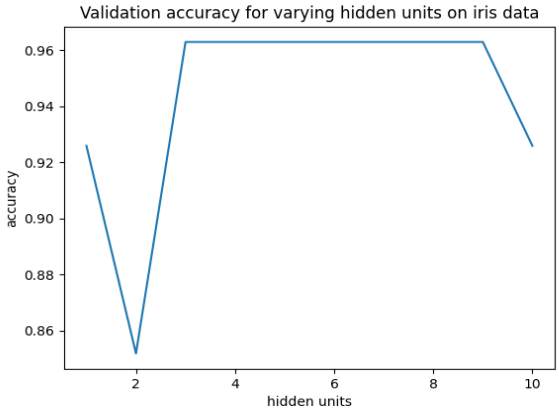

Validation accuracy for varying hidden units on linnerud data
MSE loss
hidden units

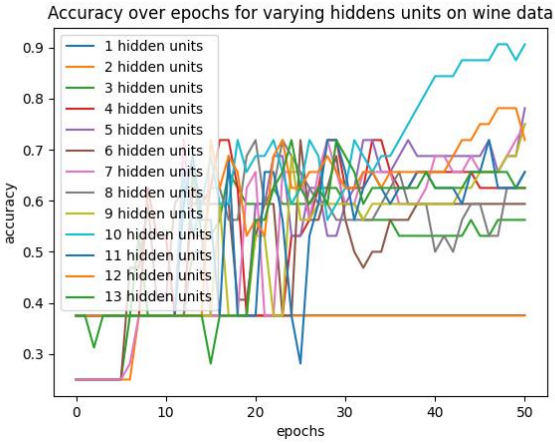

Accuracy over epochs for varying hiddens units on linnerud data
MSE loss
epochs
1 hidden units
2 hidden units
3 hidden units
4 hidden units
5 hidden units
6 hidden units
7 hidden units
8 hidden units
9 hidden units

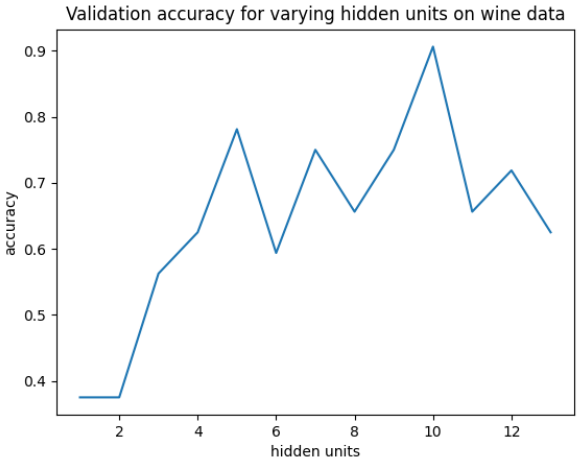
Hidden unit optimization for Rprop

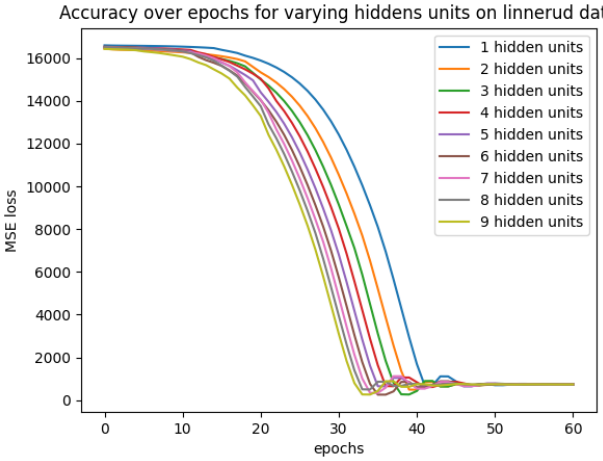


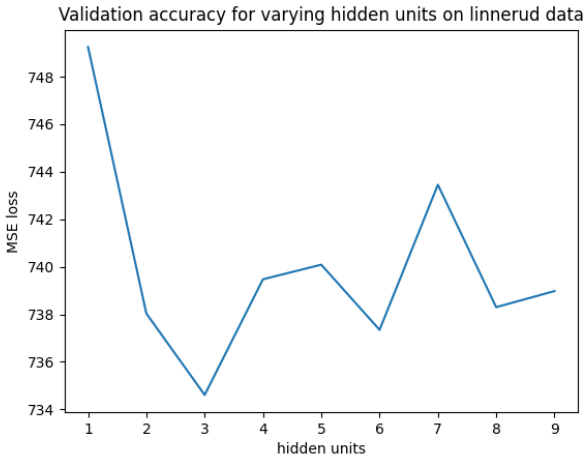


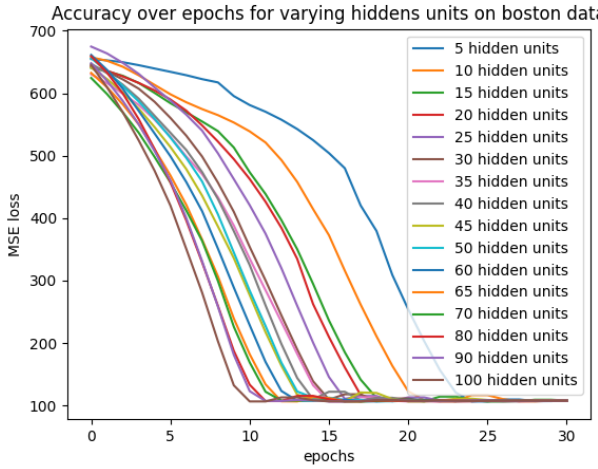


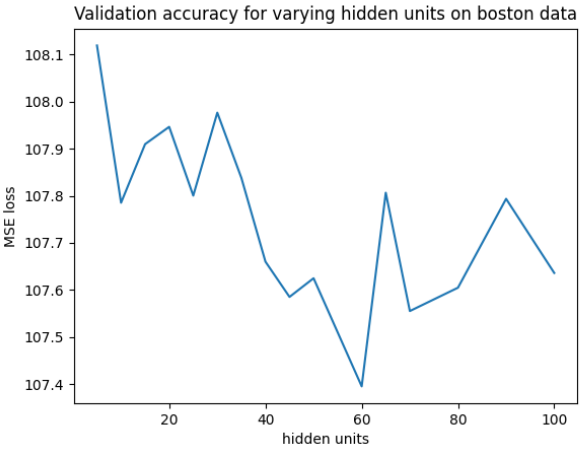


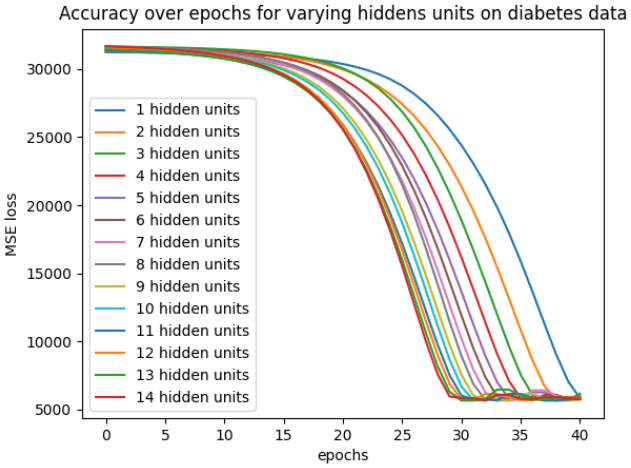


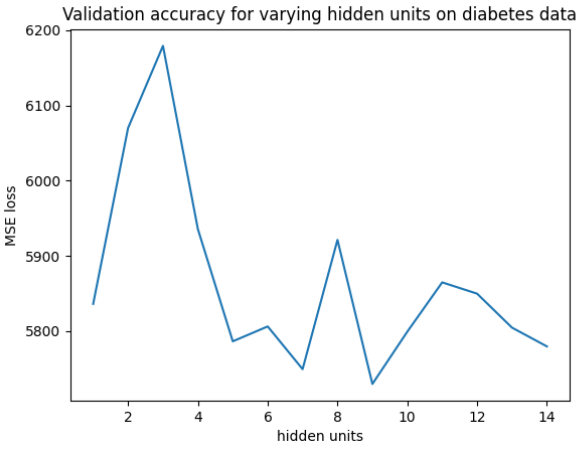

Accuracy over epochs for varying hiddens units on linnerud data
MSE loss
epochs
1 hidden units
2 hidden units
3 hidden units
4 hidden units
5 hidden units
6 hidden units
7 hidden units
8 hidden units
9 hidden units


Validation accuracy for varying hidden units on linnerud data
MSE loss
hidden units


Accuracy over epochs for varying hiddens units on boston data
MSE loss
epochs
5 hidden units
10 hidden units
15 hidden units
20 hidden units
25 hidden units
30 hidden units
35 hidden units
40 hidden units
45 hidden units
50 hidden units
60 hidden units
65 hidden units
70 hidden units
80 hidden units
90 hidden units
100 hidden units


Validation accuracy for varying hidden units on boston data
MSE loss
hidden units


Accuracy over epochs for varying hiddens units on diabetes data
MSE loss
epochs
1 hidden units
2 hidden units
3 hidden units
4 hidden units
5 hidden units
6 hidden units
7 hidden units
8 hidden units
9 hidden units
10 hidden units
11 hidden units
12 hidden units
13 hidden units
14 hidden units


Validation accuracy for varying hidden units on diabetes data
MSE loss
hidden units

The number of hidden units chosen for each model are given in table 2 and the learning parameters for SGD for each model are given in table 3.

Model	Number of hidden units SGD	Number of hidden units Rprop
Iris (classification)	2	3
Wine (classification)	2	10
Digits (classification)	7	12
Diabetes (regression)	7	7
Boston (regression)	2	60
Linnerrud (regression)	8	3

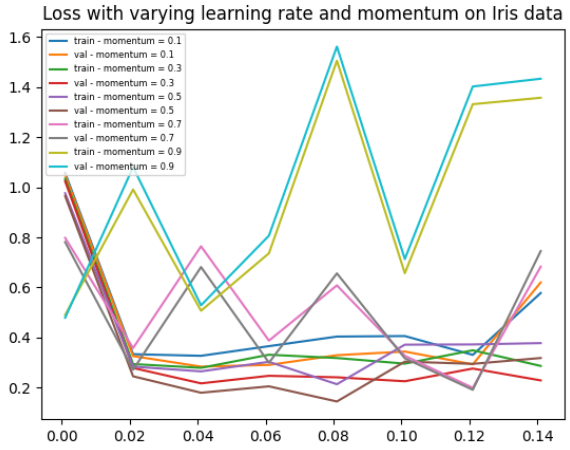
Table 2

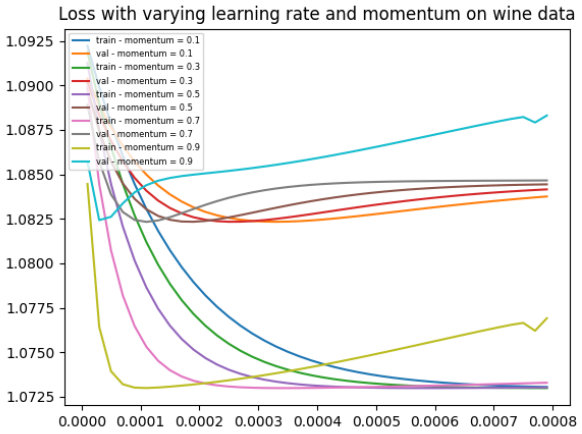
	Learning rate (lr)	*Momentum*
iris	0.08	0.5
wine	0.0001	0.7
digits	0.007	0.5
Diabetes	0.005	0.3
Boston	0.000112	0.5
linnerrud	0.004	0.9

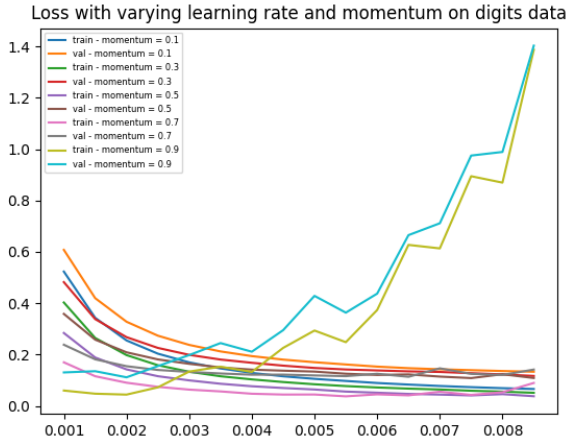
Table 3

Hyperparameter tuning for SGD

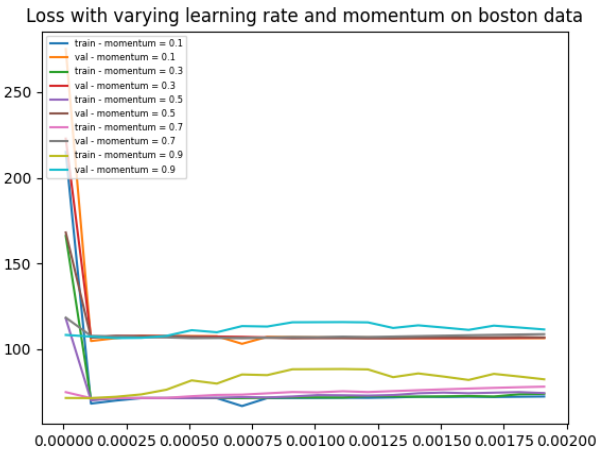
Classification

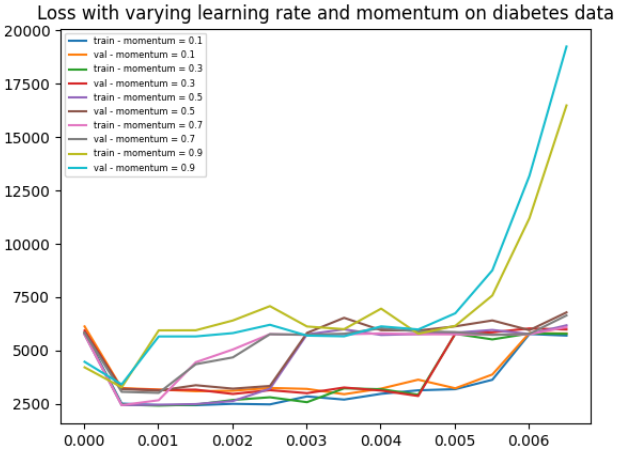


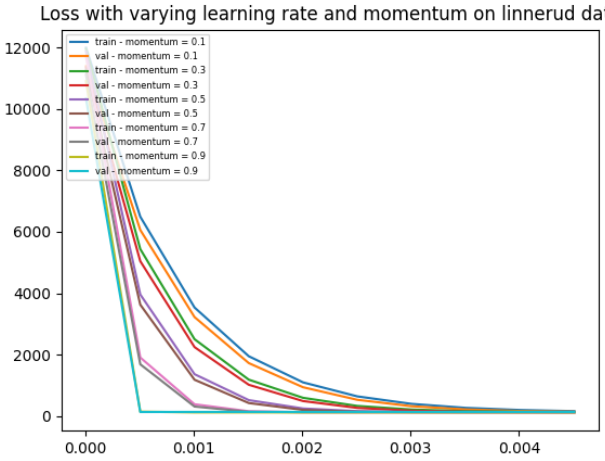




Regression

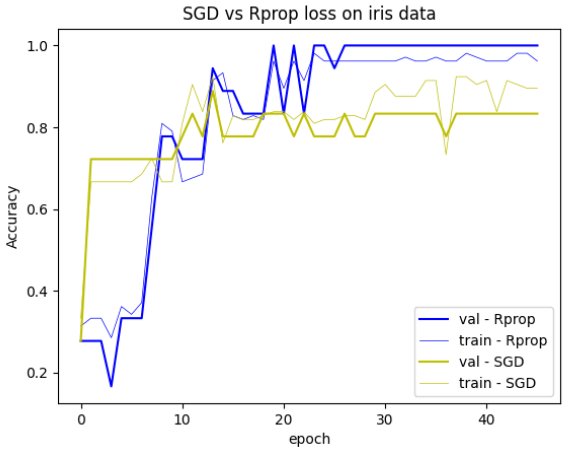


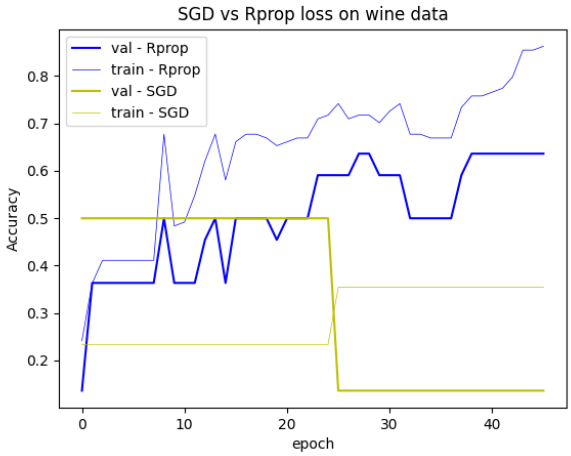


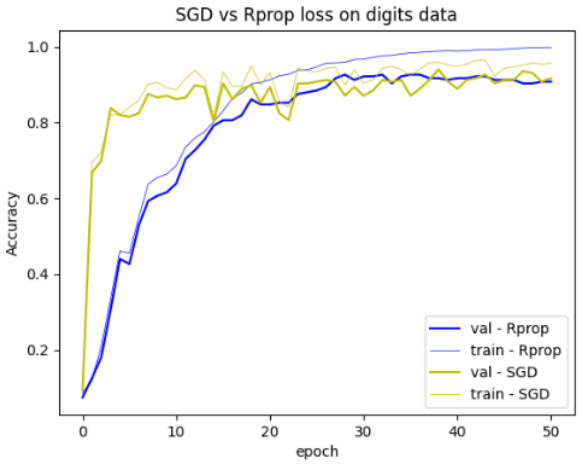


SGG vs Rprop

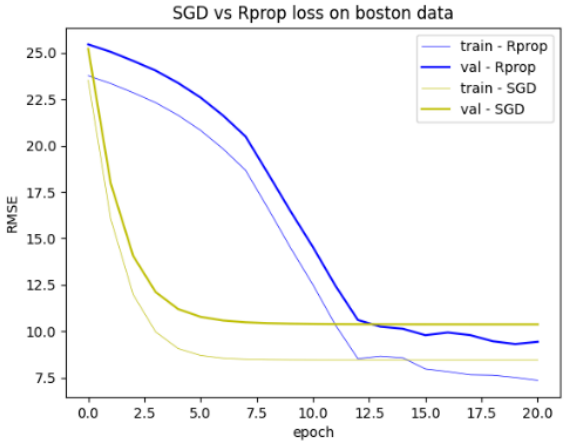
Classification

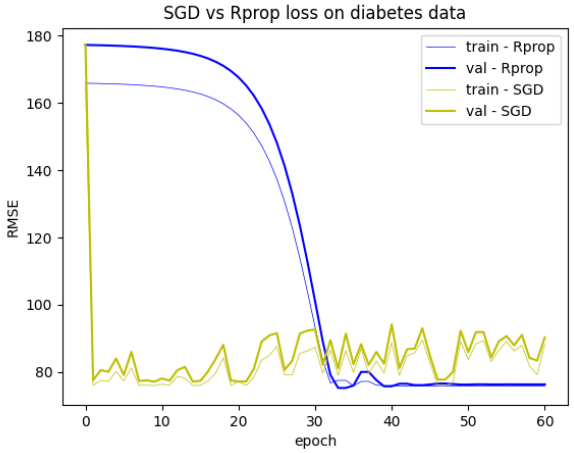


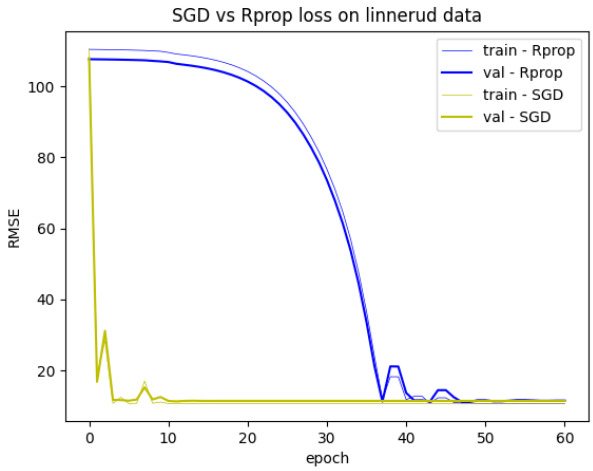




Regression

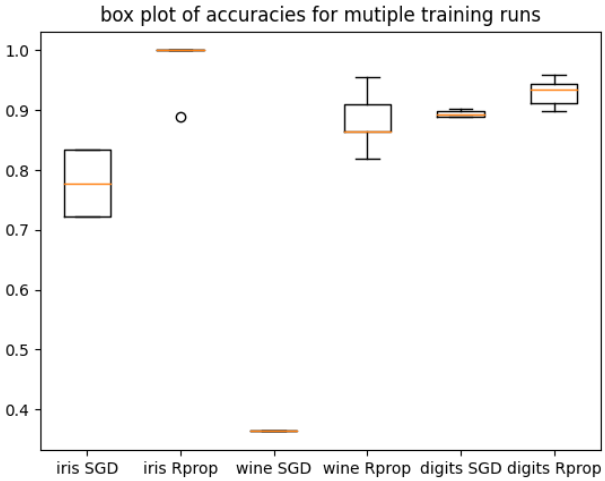




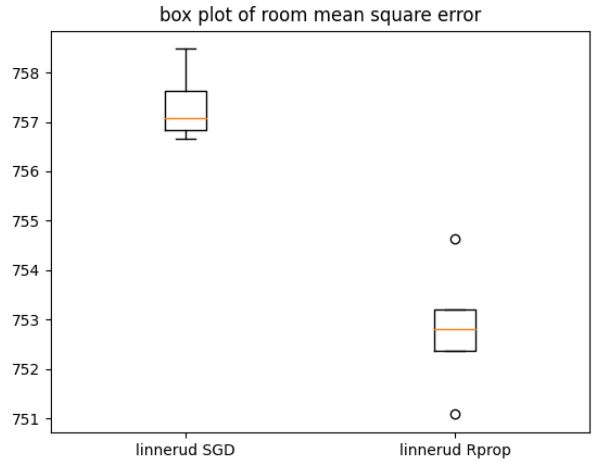


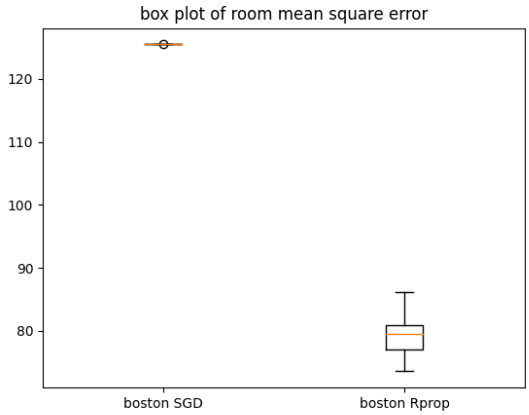
Classification results (accuracy)

Classification tasks



Regression results (RMSE)





In general, the Rprop algorithm more easily deployable, because it does not require many hyperparameters to be tuned. According to the literature [2], Rprop should converge faster than SGD. From the results we do not clearly observe this, but we do in general observe a higher score for Rprop. Rprop shows a slow initial descent, this may be because the algorithm progressively increases its step size with every correct step (i.e. no sign change between steps) which is what we observe.

For classification tasks it is evident from the "SGD vs Rprop" plots that Rprop performs better in all cases and shows more stability in is gradient descent than SGD. The wine dataset proves to be the most challenging dataset for both algorithms. SGD is not able to learn this data and the results are not conclusive. Rprop also struggles with this dataset, but it very clearly shows that it is performing gradient descent and its accuracy increases over epochs.

For regression, the overall result is that Rprop performs gradient descent more optimally than SGD and reaches a lower root mean square error in most cases. All the plots show that SGD is quick to step to a low loss value, but struggles to continue its descent. Rprop is slow to descend initially, but increases its rate of descent and ultimately reaches a more optimal result than SGD.

The box plots provide the results of multiple training iterations. It is clear from the classification tasks that Rprop performs better in general on all classification problems.

The regression problems were plotted separately as the target values are in different ranges. From the regression boxplots we see that Rprop outperforms SGD considerably on all problems.

Conclusion

RProp proves to be a more efficient algorithm than SGD on the datasets tested in this report. The Rprop algorithm provides an effective adaptive learning scheme that allows the gradient descent to increase over time and descend to a lower global minimum that SGD. In comparison, SGD converges quickly to local optima, whereas Rprop can bypass local optima and potentially reaches global optima through its adaptive

weight update rule. The added benefit of easily implementation of this algorithm without the need to optimize hyperparameters makes it very effective and efficient.

References

[1] Riedmiller M, Braun H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. InIEEE internationa1 conference on neural networks 1993 Mar 28 (pp. 586-591). IEEE.

[2] Chen L, Huang JF, Wang FM, Tang YL. Comparison between back propagation neural network and regression models for the estimation of pigment content in rice leaves and panicles using hyperspectral data. International Journal of remote sensing. 2007 Aug 20;28(16):3457-78.

[3] 1. 5. Dataset loading utilities — scikit-learn 0.16.1 documentation [Internet]. Scikit-learn.org. 2020 [cited 4 October 2020]. Available from: https://scikit-learn.org/0.16/datasets/index.html