Multiclass classification with backpropagation using student dataset

Introduction and Initial analysis:
This report intends to implement ANN back propagation model for multi class classification problem on the dataset, 'Predict student dropout and academic success'. It is done by using Matlab and Python and comparing the results of the two implementations, to gain valuable insights into the respective strengths and weaknesses of each.The dataset has 4424 records with 35 attributes and no missing and duplicate values. The problem is a three-category classification task where the majority class is Graduate, Dropout and Enrolled representing 50%, 32% and 18% respectively. Since the distribution is not heavily imbalanced, we have not balanced the dataset, as in some cases balancing the data may also introduce bias towards the minority class.

Data preprocessing:
Since the dataset contains features with different units and varying scales, by scaling the data using MinMaxScaler method, we ensure that all the features are on the same scale and do not introduce bias in the learning process. This scaled dataset was used in both Python and Matlab.

We have implemented MLP with one hidden-layer and standard back propagation algorithm in both Python and Matlab, where sigmoid activation function is used in the hidden layer and softmax in the output layer. The loss is calculated using CrossEntropy Loss function in both. The optimization algorithm used in Python is Adam optimizer while in Matlab, the default 'trainscg' (Scaled conjugate gradient backpropagation) is used.

Comparison and critical evaluation of the results:
In Python(Table 1):

No. of Neurons	Learning Rate	Training Loss	Approx.Epoch (Stable)	Time to train	Test Accuracy
10	0.001	1.25 - 0.80	500	1.32	69.6
10	0.01	1.14 - 0.54	400	1.34	76.49
10	0.8	1.05 - 0.54	Not stable	1.36	73.22
100	0.001	1.16 - 0.65	Not converged	3.21	75.48
100	0.01	1.08 - 0.46	280	3.45	75.07
100	0.8	Very unstable	Very unstable	3.14	48.14
300	0.001	1.14 - 0.60	Not converged	4.12	75.36
300	0.01	1.10 - 0.50	300	4.18	76.95
300	0.8	Not stable	Not stable	3.58	59.55

In Matlab(Table 2):

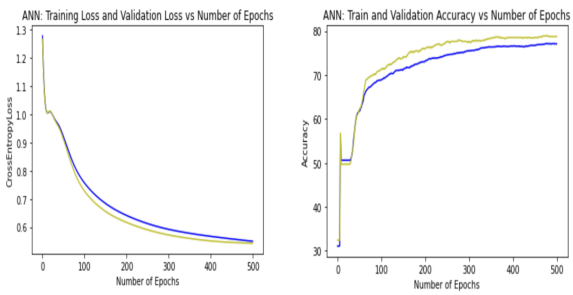
Neurons	Learning Rate	Training Loss	Epoch	Time to train(sec)	Test Accuracy
10	Optimized	0.608 - 0.179	53	4.03	75.59
100	Optimized	1.09 - 0.155	49	6.67	75.25
300	Optimized	1.82 - 0.182	45	9.22	74.01

We are comparing the performance of Python and Matlab models by passing different sets of hidden layer neurons and learning rates (please refer to tables 1 and 2). As per the Matlab functionality, it adjusts the optimum learning rate during the training process and we do not have the provision to change it manually. Both the models are tested against 3 different numbers of neurons, 10, 100 and 300 respectively and in Python, it is further coupled with 3 different learning rates, 0.001, 0.01 and 0.8 respectively.
With the learning rate 0.001, the training and validation loss were not converging within the given maximum 500 epochs for all three sets of neurons and also they were higher with 0.8, 0.65 and 0.6 respectively. However, the test accuracies were better for 100 and 300 neurons with around 75% than for 10 neurons where it was 69.6%. With a high learning rate of 0.8, the training loss and accuracy were not stable in any of the 3 sets of neurons and therefore not suitable for the model to train. Therefore, the learning rate 0.01 seems to be optimum for training the model in Python.
From the tables, we can infer that the time taken to train the python model was lesser compared to the Matlab model and also the testing accuracy seems to be better in Python than Matlab. On the other hand, the Matlab model was trained with less number of epoch iterations than its Python counterpart and also, the training loss was very less comparatively.

Lessons learned and future work
Increasing the number of neurons can improve performance but may lead to overfitting and longer training times. Higher learning rates can cause loss oscillation and lower rates can result in slower convergence. Lower training loss and higher test accuracy indicate better models. Stable training processes are preferable for effective learning. Shorter training times are desirable, but hardware constraints may affect this. Future work should explore changing other parameters and implementing advanced techniques such as regularization and dropout for more robust models with better performance.

In Matlab:

In Python:



References:

[1] “Predict students' dropout and academic success” The Devastator. (2020). Higher Education Predictors of Student Retention. Retrieved from https://www.kaggle.com/datasets/thedevastator/higher-education-predictors-of-student-retention.

[2] Rodríguez-Hernández, C. F., Musso, M., Kyndt, E., & Cascallar, E. (2020). ‘Artificial neural networks in academic performance prediction: Systematic implementation and predictor evaluation’. Computers & Education, 150, 103859. doi: 10.1016/j.compedu.2020.103859.

[3] Realinho, V., Machado, J., Baptista, L., & Martins, M. V. (2020). Data Descriptor: Predicting Student Dropout and Academic Success. Scientific Data, 7(1), 1-12. https://doi.org/10.1038/s41597-020-0486-1