Logistic Regression and Multi-Layer Perceptron Classifiers

Logistic Regression Analysis

Accuracy:
The logistic regression model demonstrates decent accuracy across both datasets. By correctly classifying instances, it exhibits a satisfactory level of performance (an accuracy of >60% on both training and validation sets for both datasets). Further optimization and fine-tuning may push accuracy even further. Rescaling the data around (0, 1) before training helped increase the accuracy for both datasets. On average, the training accuracy for “water” was around 94.04% and the validation accuracy was around 69.05%, the wide gap indicating overfitting. The training accuracy for “loans” was around 71.70% and the validation accuracy was around 69.68%.

Generalization:
Utilizing the validation data, the model's generalization performance was assessed. Logistic regression exhibits a notable ability to generalize to unseen data. For the relatively large dataset “loans”, the difference between training accuracy and validation accuracy wasn’t great, indicating well generalization. On the other hand, for the smaller dataset “water”, the model seems to overfit, since the training accuracy is much higher than validation accuracy. This can be solved by using more training data and applying generalization techniques (L1/L2 regularization, etc).

Compute Requirements:
One of logistic regression's key advantages lies in its minimal compute requirements. Leveraging simple matrix operations, it efficiently processes data, making it suitable for large-scale datasets and resource-constrained environments. On average, training the model on “water” took 0.3276 seconds. For “loans”, the model took around 26.2378 seconds to train.

Reliability:
The reliability of logistic regression is evident in its consistent performance across multiple runs. With straightforward interpretation, it is a dependable choice for classification tasks. However, there is a trade off between speed and accuracy. More complex models like Neural Networks will provide better results but will take longer to train.

<u>Accuracy vs. Compute Plot:</u>
The plots below illustrate the relationship between accuracy and time for the best configuration of logistic regression. It provides insights into the model's convergence behavior, and how accuracy evolves with increasing computational investment.

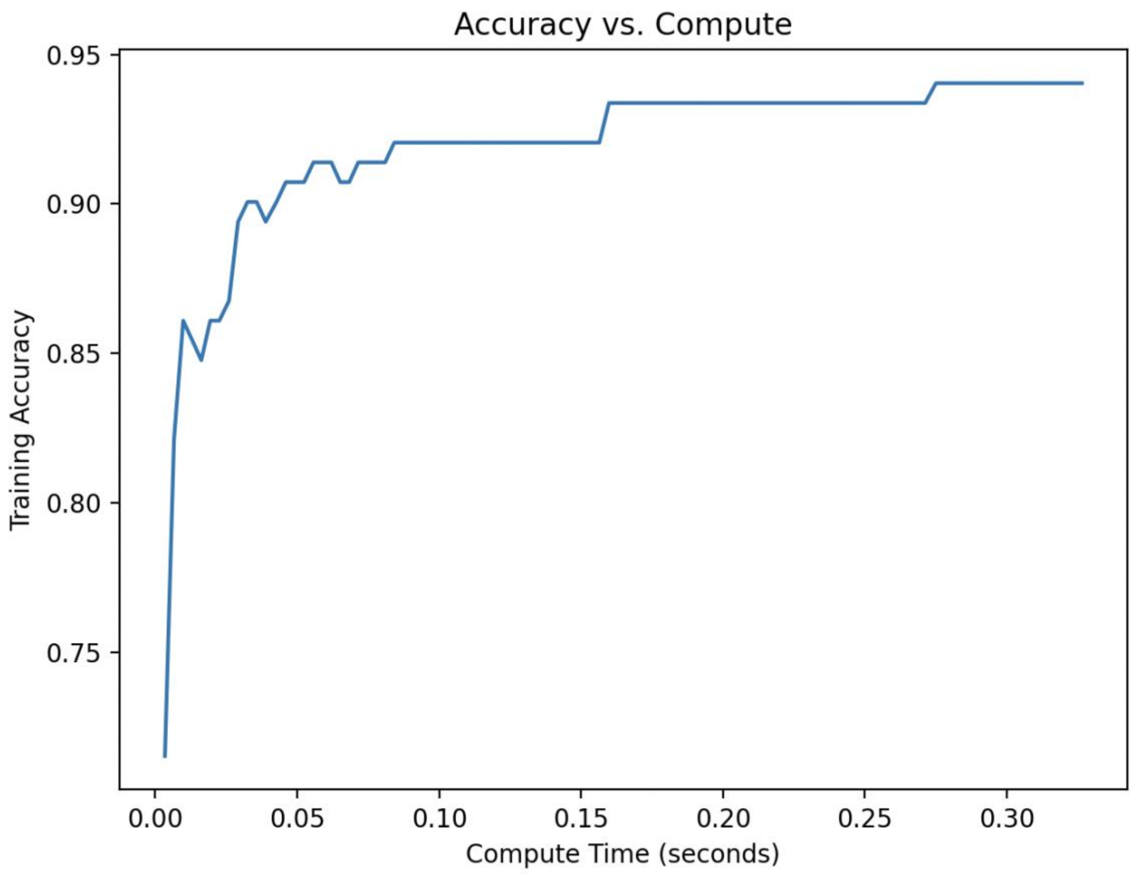


Figure 1 "Water" Dataset on Logistic Regression

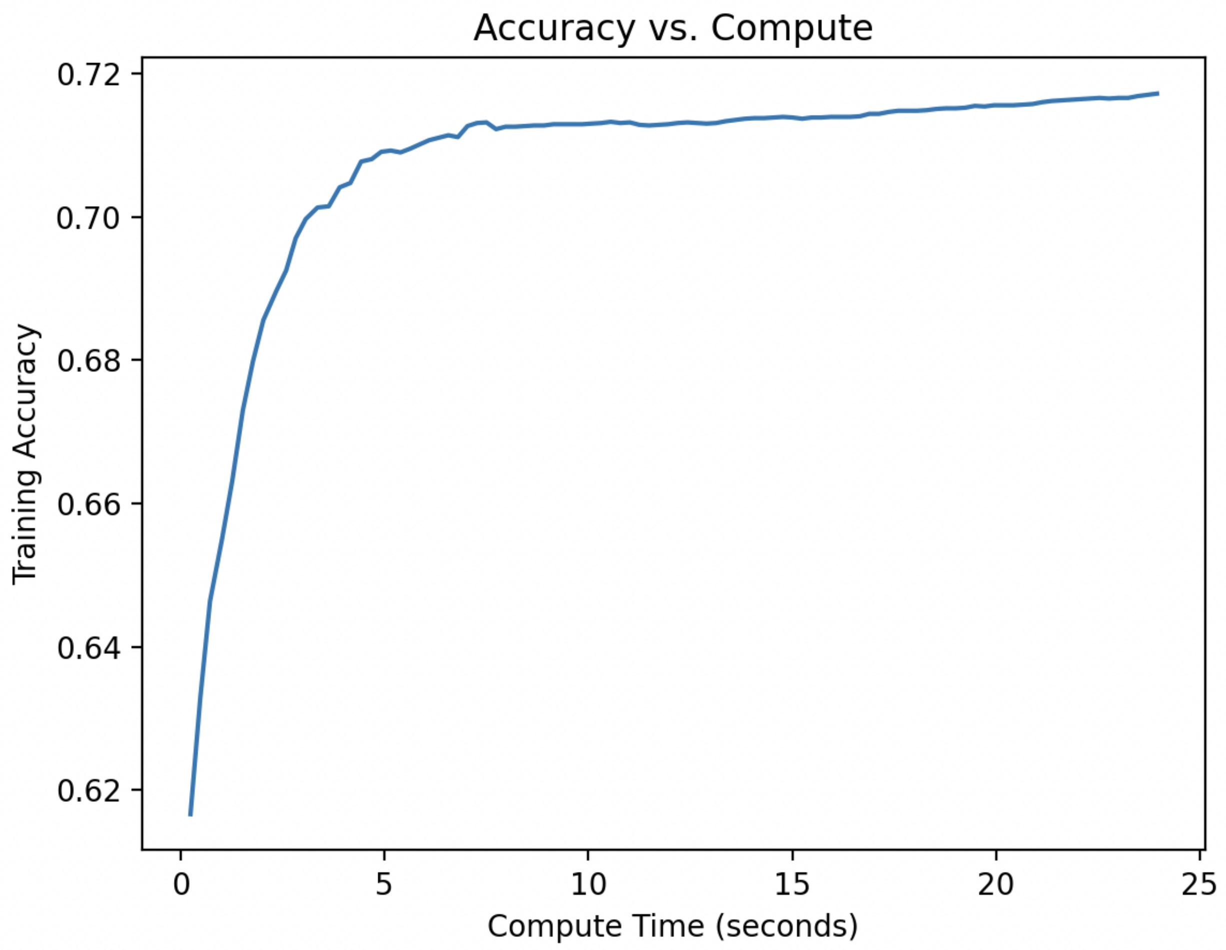


Figure 2 "Loans" Dataset on Logistic regression

Multilayer Perceptron Analysis

Model Description:
The MLP consists of an input layer, a hidden layer with 10 tanh activation units, and an output layer. The weights are initialized using a zero-mean unit variance Gaussian distribution. Stochastic gradient descent (SGD) is employed for training, with a softmax activation function in the output layer for multi-class classification tasks.

Training Methodology:
The training data is preprocessed by rescaling features to a range between 0 and 1 to facilitate convergence. The model is trained using SGD for a maximum of 1000 epochs, with a learning rate of 0.1. Regularization was also implemented (with a constant of 0.015) to prevent overfitting. During each epoch, the training data is shuffled to prevent the model from memorizing the order of samples. Backpropagation is utilized for gradient computation, and weights are updated using the computed gradients.

Performance Evaluation

<u>Accuracy:</u>
The MLP demonstrates good levels of accuracy on the datasets. Detailed accuracy metrics, including both training and validation accuracies, are computed to assess model performance. The training accuracy for “water” was around 99% and the validation accuracy was around 71.43%. The training accuracy for “loans” was around 74.88% and the validation accuracy was around 73.30%. The MLP model is more powerful compared to the Logistic Regression model, and hence had better results.

<u>Generalization:</u>
The generalization performance of the MLP is evaluated using validation data. Discrepancies between training and validation accuracies are examined to determine the model's generalization capability. The model seems to generalize well for “loans”, with very close accuracy values for training and validation. On the other hand, it’s overfitting for “water”. This could be solved by using more training data, data augmentation, regularization, etc.

<u>Compute Requirements:</u>
MLPs entail higher computational demands compared to logistic regression due to their more complex architecture (layers and hidden units) and training process. On average, training the model on “water” took 0.5265 seconds. For “loans”, the model took around 40.2321 seconds to train.

<u>Reliability:</u>
The reliability of the MLP is assessed based on its consistency across multiple training runs. There were no observed fluctuations in performance across runs.

<u>Accuracy vs. Compute Plot:</u>
The plots below illustrate the relationship between accuracy and time for the best configuration of MLP.

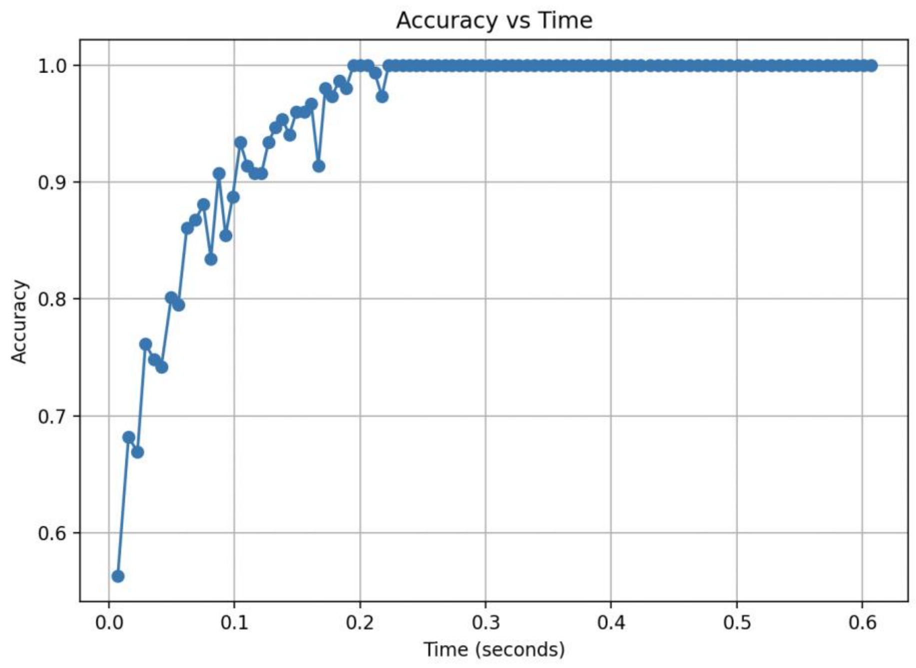


Figure 3 Water Dataset Plot using MLP

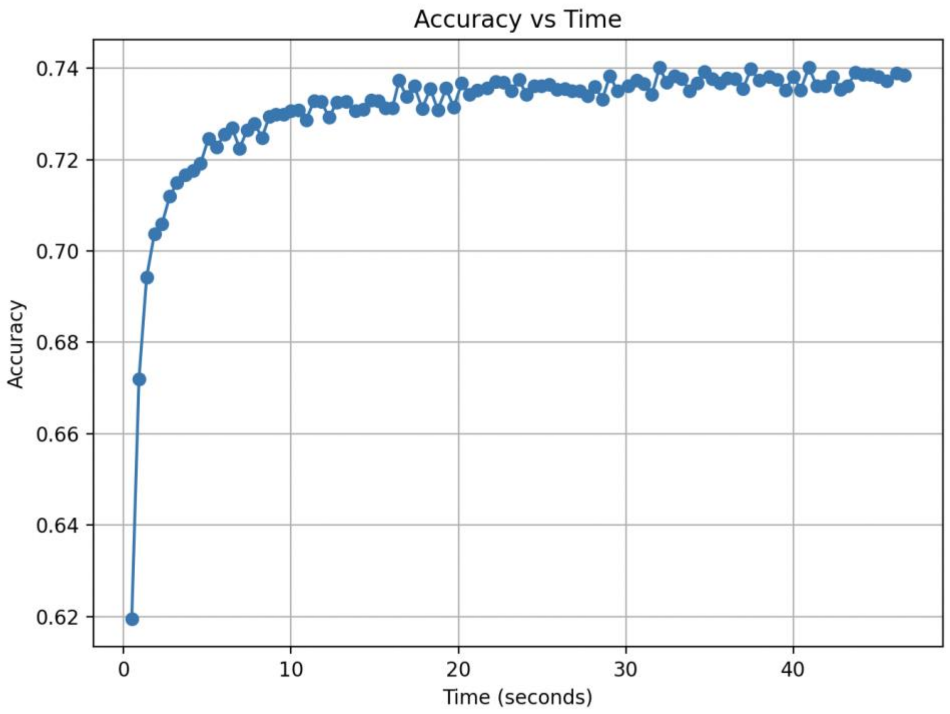


Figure 4 Loans Dataset Plot using MLP

Research Question Response

How important are good initialization? (Compare different initialization schemes.)

Initialization is a critical aspect of neural network training, as it establishes the starting point for optimization algorithms and significantly influences convergence speed, stability, and final model performance. Improper initialization can lead to issues such as exploding or vanishing gradients, hampering effective training processes. In this section of the report, we will investigate the importance of good initialization by comparing various initialization schemes[1].

One commonly used initialization method, popularized by AlexNet, the winner of the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), involves initializing weights with Gaussian (normal) noise[2]. This method sets the mean to zero and the standard deviation to 0.01, with bias set to one for specific layers. However, this approach encounters challenges, especially in training deep networks with ReLU activation functions, due to the aforementioned gradient issues.

To address these challenges, Xavier and Bengio (2010) proposed the "Xavier" initialization technique[3]. This method takes into account the network's architecture, including the number of input and output units, during weight initialization. By adjusting weights to be inversely proportional to the square root of the number of units in the previous layer (referred to as fan-in), Xavier initialization aims to maintain weights within an appropriate range.

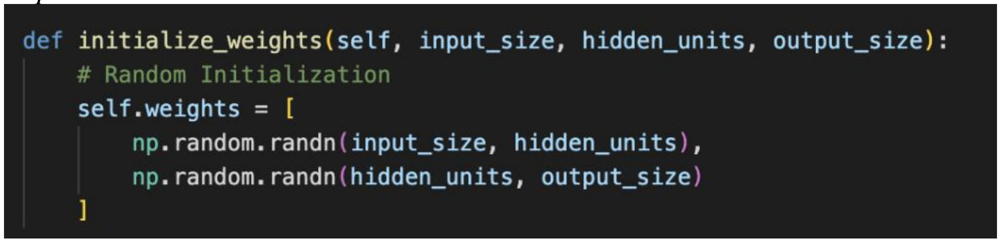
The choice of activation function is pivotal in determining the effectiveness of the initialization method. Kaiming He et al. (2015) introduced an enhanced weight initialization technique tailored for popular activation functions like ReLU, considering their asymmetry[4]. This method, detailed in He et al.'s paper, shares a similar theoretical foundation with Xavier initialization. Both approaches determine an appropriate variance for the distribution from which initial parameters are sampled, adapting to the specific activation function without explicitly considering the distribution type.

Despite significant advancements, weight initialization remains a dynamic field of research. Recent endeavors have explored various avenues, including data-dependent initializations, sparse weight matrices, and random orthogonal matrix initializations. This ongoing research underscores the importance of continual exploration and refinement in the realm of weight initialization techniques.

Experiments

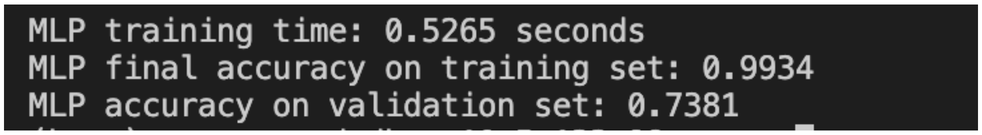
a. <u>Using random initialization to train an MLP on “water” dataset</u>

- *Implementation:*



```
def initialize_weights(self, input_size, hidden_units, output_size):
    # Random Initialization
    self.weights = [
        np.random.randn(input_size, hidden_units),
        np.random.randn(hidden_units, output_size)
    ]
```

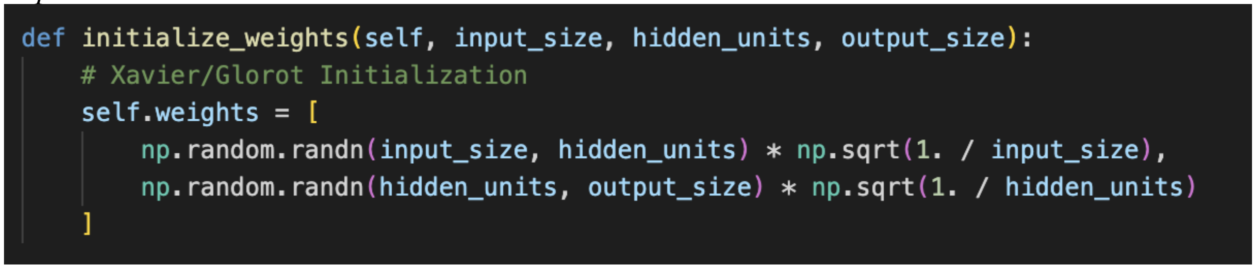
- *Results:*



```
MLP training time: 0.5265 seconds
MLP final accuracy on training set: 0.9934
MLP accuracy on validation set: 0.7381
```

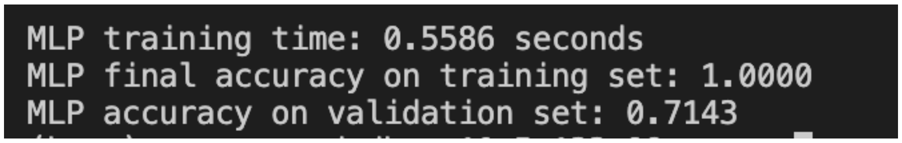
b. <u>Using Xavier/Glorot initialization to train an MLP on “water” dataset</u>

- *Implementation:*



```
def initialize_weights(self, input_size, hidden_units, output_size):
    # Xavier/Glorot Initialization
    self.weights = [
        np.random.randn(input_size, hidden_units) * np.sqrt(1. / input_size),
        np.random.randn(hidden_units, output_size) * np.sqrt(1. / hidden_units)
    ]
```

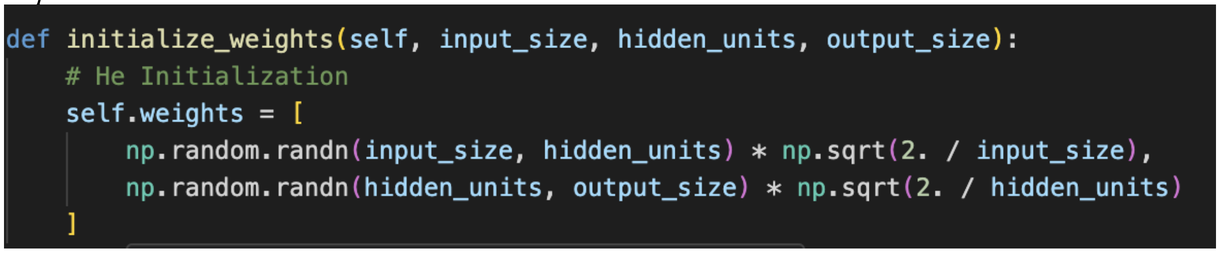
- *Results:*



```
MLP training time: 0.5586 seconds
MLP final accuracy on training set: 1.0000
MLP accuracy on validation set: 0.7143
```

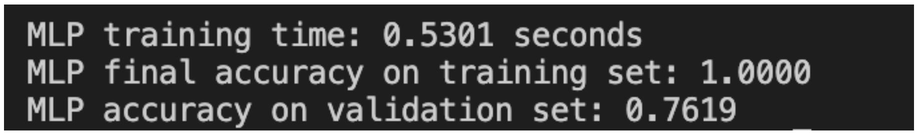
c. <u>Using He Initialization to train an MLP on “water” dataset</u>

- *Implementation:*



```
def initialize_weights(self, input_size, hidden_units, output_size):
    # He Initialization
    self.weights = [
        np.random.randn(input_size, hidden_units) * np.sqrt(2. / input_size),
        np.random.randn(hidden_units, output_size) * np.sqrt(2. / hidden_units)
    ]
```

- *Results:*



```
MLP training time: 0.5301 seconds
MLP final accuracy on training set: 1.0000
MLP accuracy on validation set: 0.7619
```

Works Cited

1. Guo, J. (n.d.). *AI Notes: Initializing neural networks - deeplearning.ai*. deeplearning.ai. https://www.deeplearning.ai/ai-notes/initialization/index.html#I

2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (n.d.). ImageNet Classification with Deep Convolutional Neural Networks. *ImageNet Classification With Deep Convolutional Neural Networks*. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

3. Glorot, X., Bengio, Y., & DIRO, Universit´e de Montr´eal, Montr´eal, Qu´ebec, Canada. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010* (Vol. 9). https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

4. He, K., Zhang, X., Ren, S., & Sun, J. (2015, February 6). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. arXiv.org. https://arxiv.org/abs/1502.01852