COMP3221 Assignment 2 Report

Implementation.

The classification model used in this program is MCLR, Multi-Class Logistic Regression. The softmax function:

$$h_w(X) = p(y = k|X; w_1, \dots, w_k) = \frac{\exp(w_k^T X)}{\sum_{k=1}^{K} \exp(w_k^T X)}$$

is used to model the posterior probability, and the class with the maximum probability of a particular data sample is the class we predict. The cross-entropy error function:

$$f(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log \frac{\exp(w_k^T x_n)}{\sum_{k=1}^{K} \exp(w_k^T x_n)}$$

is used to measure the loss and hence, the gradient is given by:

$$\nabla f(w) = \sum_{n=1}^{N} (h_w(w^T x^n) - y^n) x^n$$

In each iteration, we update the model by:

$$w_{j+1} \leftarrow w_j - \eta \left(\sum_{n=1}^{N} \left(h_{w_j}(x^n) - y^n \right) x^n \right)$$

In the formula above, η is the learning rate. We tune this parameter along with the local epoch E and batch size B to optimise the performance. The Federated Learning is a fast-developing decentralised machine learning technique where there is only one global model and many local models in a network, which can quickly incorporate new data and preserve privacy. The techniques and methodology of implementing the federated learning algorithm is as follows.

The server firstly waits for the handshake from the clients. It maintains a counter of the total number of the clients. Then it will send its first global model that is randomly generated to all its clients. And it will receive the models from the clients and aggregate the new global model. The server can either aggregate all the models, or randomly two models. It will calculate the global model from the clients' local model based on their sample size. The client with a larger sample size has more weight on the global model's calculation. The server will iterate this process 100 times, and the global model that is generated by the last iteration will be the final model. **The clients** send the client ID, sample size and number of features to the server as the handshake message. Then they will listen and wait for the server to send the global model to them, and evaluate the performance (loss and accuracy) to the log file and then update the local model based on GD or mini-batch GD. They output the information and send the new local model to the server to be aggregated.

Techniques and unique features

After finishing all the requirements, we notice that when the loss of the global model is too large, the function `np.exp` in the softmax function may cause an overflow, resulting in a local model full of NaN values, which will further affect the global model in the future iterations. However, we find that not all clients have the NaN issue at the same time and this

issue only means that the loss of the global model is relatively too large for the particular client, hence, if the server only aggregates the local models without the ones containing NaN, the new global model will have a much lower probability of causing a NaN problem for the same client. By ignoring the local models with NaN in the aggregation part, we are able to tune the hyperparameters in a larger range and optimise the performance even better.

In addition, we have conducted a series of trials of tuning the local epoch E, batch size B and learning rate η, and successfully reach an accuracy above 90% for both GD and mini-batch GD methods. **Figure 1 - 4** in the appendix show that in mini-batch GD, by increasing E and other things equal, the performance cannot be improved remarkably, but the running time is increased significantly. The same observation is found in GD as well, a large E only shorten the time for convergence but not necessary improve the accuracy, hence, we set E = 2 for both methods. **Figure 5 – 10** in the appendix show that by decreasing the mini-batch GD's batch size, the convergence will be faster, and the accuracy will be higher. We therefore use a batch size B = 5 in mini-batch GD.

After trying the fixed learning rate and the dynamic learning rate, we decide to use $\eta_i = \frac{\eta_0}{n\sqrt{i}}$ as the learning rate at i-th iteration for mini-batch GD, where i is the iteration step, n is the number of training data samples, η_0 is a constant step which is chosen to be 0.04. By using this learning rate, we can reduce the size of the step over time, improving the performance. Similarly, we chose a dynamic time-based learning schedule for GD, which is of the form: $\eta_{i+1} = \frac{\eta_i}{1+d \times i}$, where η is the learning rate, i is the iteration step, η_0 is 4, and d is a decay parameter which is set to be 0.00067 after a series of testing. By tuning the hyperparameters as stated above, it is guaranteed that the performance of the global model is greater than 90%.

Furthermore, we have implemented to deal with the **client failure** when connecting to the server and the **late connection** of clients. The server uses different sockets to receive the handshakes and local models, if there are less than five clients connected to the server, the program can still process normally with the rest of the clients. And if the client connects to the server after the federated learning algorithm starts, the server will add the new client to the system and send the global model to it and aggregates its local model in the following iterations.

Comparison between GD and mini-batch GD

As mentioned above, the larger the batch size, the faster the process takes and the lower the accuracy as shown in figure 10, 8 and 6. The best performance using mini-batch GD is given by **figure 10** in appendix, where we get an accuracy of global model of 92% with a run time of 35s. If set the batch size to the full sample size, the performance will fall to 25.5% as shown in **figure 13**, but the running time decreases significantly, to 13s. Hence, we need to tune the learning rate and local epoch to optimise the performance. The best performance using GD is given by **figure 12**, with an accuracy of 90.5% and running time of 15s, which is still significantly faster than the mini-batch GD. This is because in the GD, there is only one iteration (local model update) per local epoch per global iteration while there are N/B iterations (local model updates) per local epoch per global iteration. Hence, **in our federated learning**

system, mini-batch GD gives us a better performance while scarifying the runtime; GD gives us a better runtime while scarifying the optimal performance.

Comparison between subsample clients and non-subsample client

During the first several epochs, it is highly possible that the random two clients' models are not-a-number (This is caused by the softmax function, the randomly generated model becomes too large after the normalized exponential process). In this case, we will not clear the previous global model, but use the last epoch's global model for the next iteration (otherwise we lost the previous work). If only one of the random two clients' models is not-a-number, we only aggregate the other one that is working.

Figure 14, 15 is a set of images of a subsample client model's learning process using mini-batch GD. It is obvious that the learning process is much more unstable and inaccurate than the non-subsample client model.

Figure 16, 17 is another set of images of a subsample client model's learning process using GD. The learning process is even more unstable and inaccurate than the subsample client model with mini-batch GD. But the fluctuation of loss decreases over the global iteration. In addition, the increased volatility of loss and accuracy of GD sub-client is not controlled by the local epoch E. If increasing E to 10 as shown in **figure 18, 19**, the volatility in loss and accuracy has no obvious change.

The non-subsample client model is much more accurate and stable than the subsample-clients model for both GD and mini-batch GD. Because sub-clients model only randomly aggregates two models in each epoch, and the dataset for each client varies a lot, the model that is suitable for the randomly two clients is not always suitable for all the clients.

Appendix

- Appendix A. Tuning local epoch E, learning rate and batch size (B = 5)

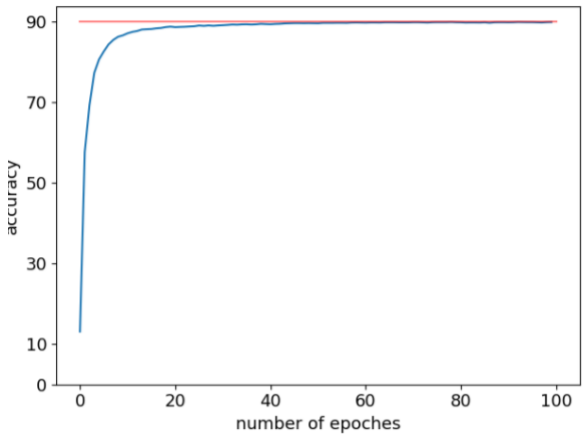

Figure 1 Mini-batch, E = 1, t = 21.48s

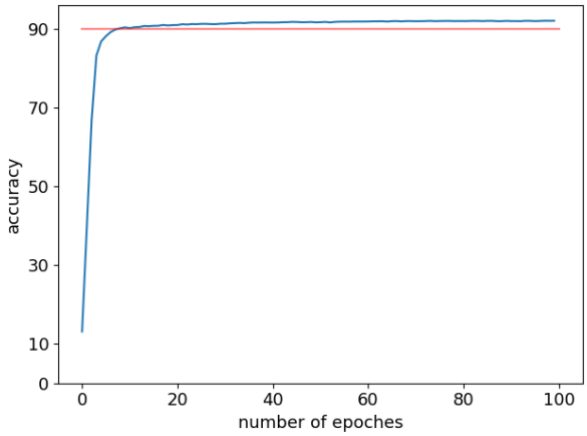

Figure 2 Mini-batch, E = 2, t = 35.64s

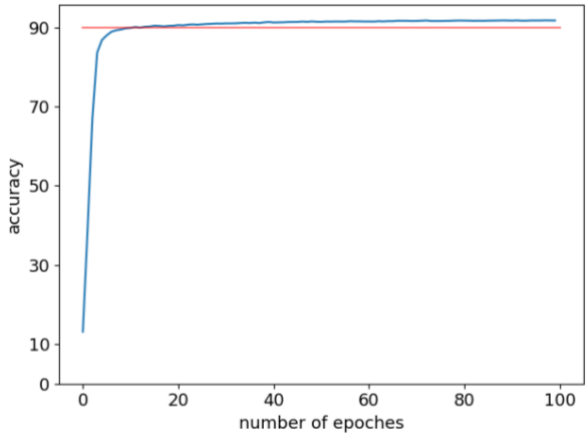

Figure 3 Mini-batch, E = 3, t = 47.24s

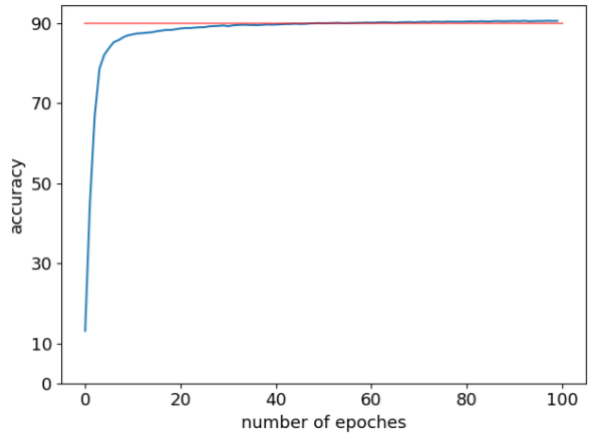

Figure 4 Mini-batch, E = 10, t = 99.69s

- Appendix B. Tuning batch size B, learning rate $\eta_i = \frac{0.04}{n\sqrt{i}}$ and local epoch E = 2

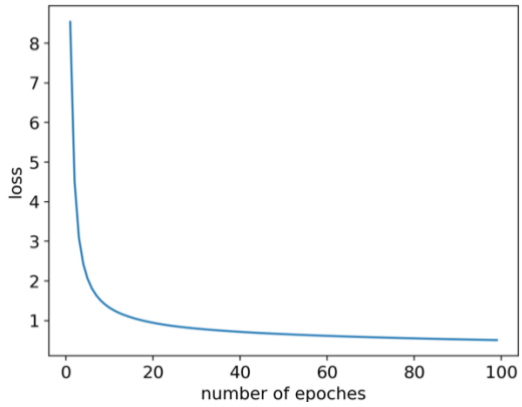

Figure 5 Mini-batch loss, B = 20, t = 19s

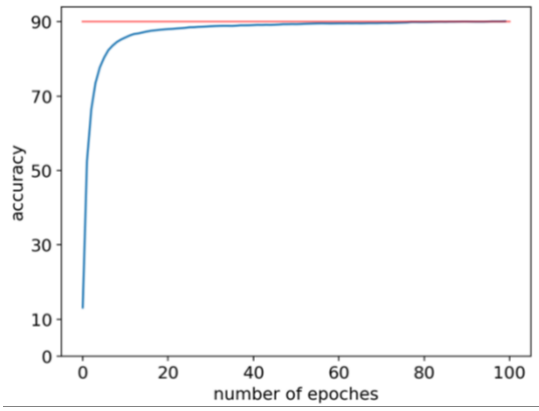

Figure 6 Mini-batch accuracy, B = 20, t = 19s

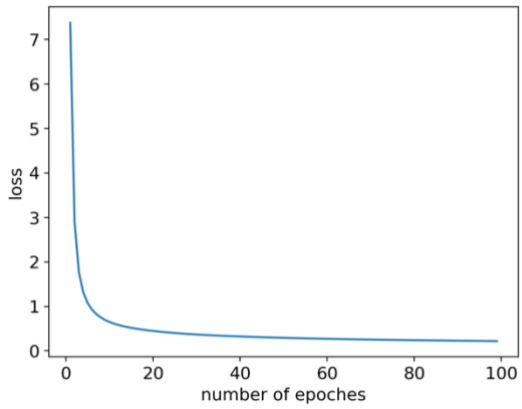

Figure 7 Mini-batch loss, B = 10, t = 25s

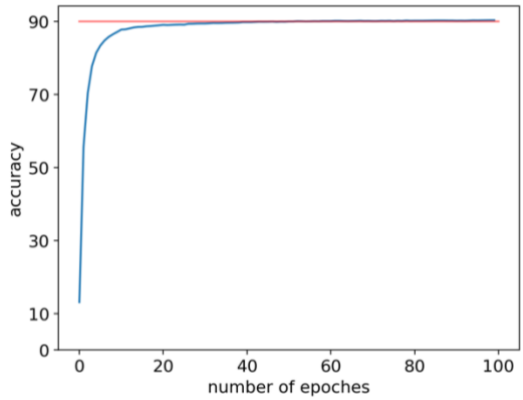

Figure 8 Mini-batch accuracy, B = 10, t = 25s

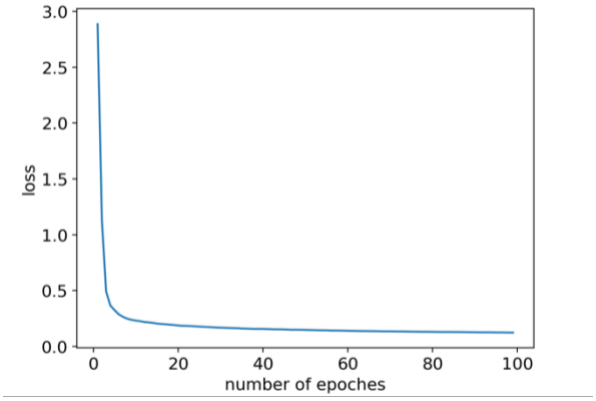

Figure 9 Mini-batch loss, B = 5, t = 35s

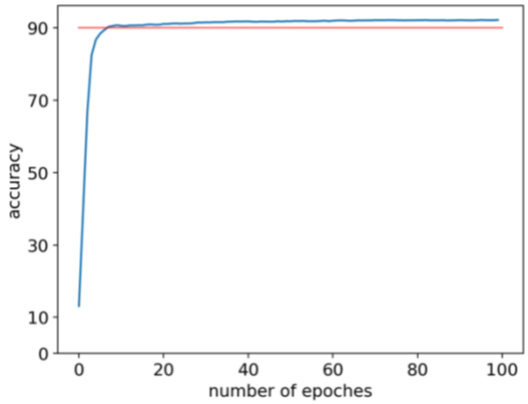

Figure 10 Mini-batch accuracy, B = 5, t = 35s

- Appendix C. GD, learning rate $\eta_{i+1} = \frac{\eta_i}{1+0.00067 \times i}$, $\eta_0 = 4$, local epoch E = 2,

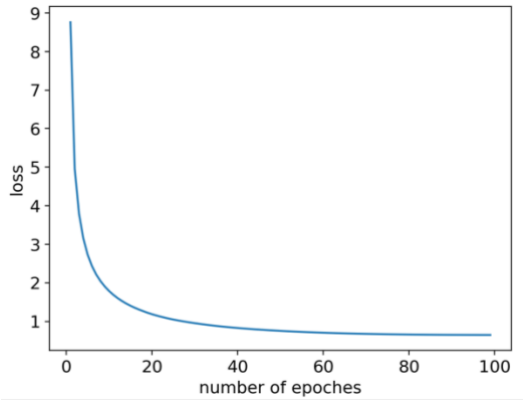

Figure 11 GD loss, t = 15s

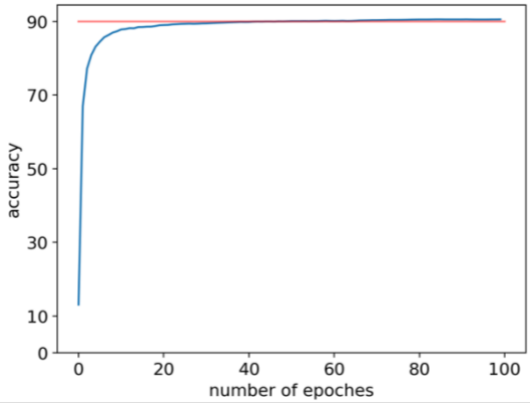

Figure 12 GD accuracy, t = 15s

- Appendix D. GD, learning rate $\eta_i = \frac{0.04}{n\sqrt{i}}$ (same as mini batch), local epoch E = 2

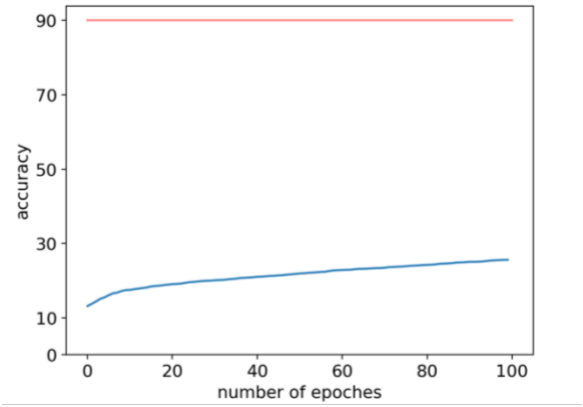

Figure 13 GD accuracy, t = 13s

- Appendix E. Sub-clients, M = 2,

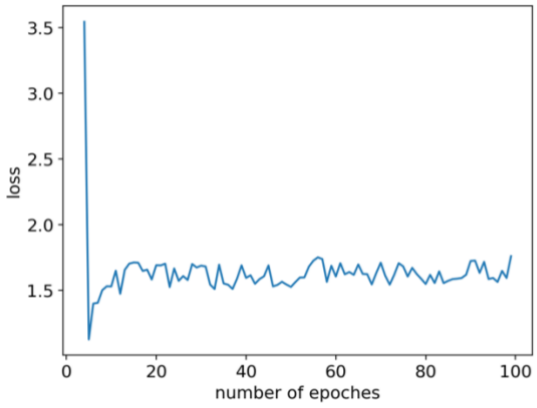

Figure 14 Mini-batch loss, B = 5, E = 2, η = Appendix B, t = 32s

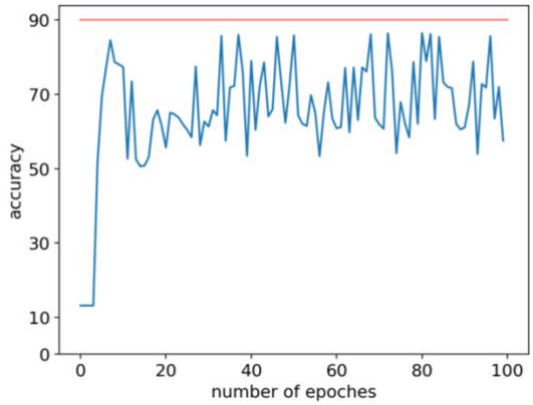

Figure 15 Mini-batch accuracy, B = 5, E = 2, η = Appendix B, t = 32s

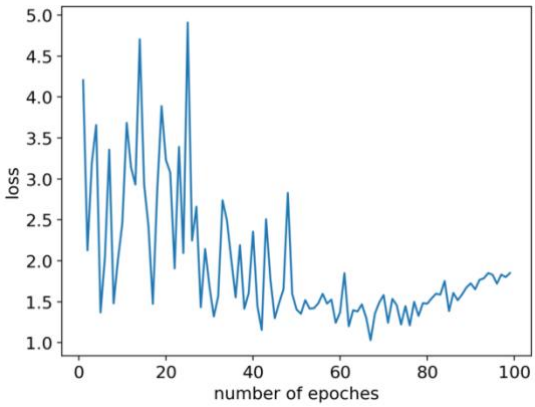

Figure 16 GD loss, E = 2, η = Appendix C, t = 10s

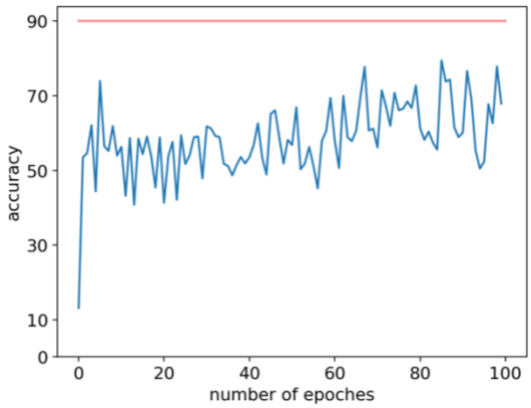

Figure 17 GD accuracy, E = 2, η = Appendix C, t = 10s

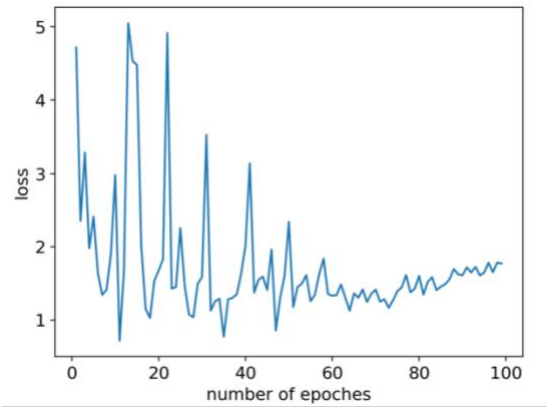

Figure 18 GD loss, E = 10, η = Appendix C, t = 82s

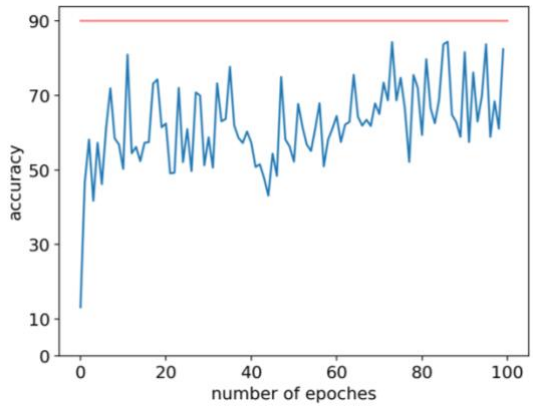

Figure 19 GD loss, E = 10, η = Appendix C, t = 82s