DEEP LEARNING ASSIGN-1

Priya Savithiri Baskaran(25232590)
Github repo : DeepNN-A1

ABSTRACT:

This assignment implements deep neural networks from first principles using Python in a Jupyter Notebook, closely following the material covered in lectures. Only low-level functionality from NumPy is used for array operations, no built-in implementations of core learning algorithms (such as optimizers or gradient descent routines) are used. The work progresses in a structured way from simpler to more complex models. First, logistic regression is implemented as a neural network with no hidden layers and a single output node, trained using stochastic gradient descent (SGD). This implementation is then evaluated on two small datasets: one that is largely linearly separable and another that requires a non-linear decision boundary, highlighting the limitations of linear classifiers. Next, the model is extended to a shallow neural network with one hidden layer, enabling non-linear representations and improved performance on non-linearly separable data. This same shallow network is then applied to a real handwritten letter recognition task using the EMNIST Letters dataset, restricted to two classes (F vs L) as specified. Finally, the implementation is generalized to support a deep network with an arbitrary number of layers and an arbitrary number of nodes per layer. A lecture discussed method to reduce overfitting on small datasets (L2 regularization / weight decay) is included and experiments compare shallow versus deeper architectures and training with and without regularization. Performance is reported using training curves, accuracy metrics and classification analysis tools such as confusion matrices and ROC curves where appropriate.

1. TASK 1 - Logistic Regression Implemented from First Principles:

Task 1 builds a complete binary logistic regressor from scratch, written in a neural-network style (no hidden layers, one output node). The model takes input features, produces a probability between 0 and 1, and then learns the best weights by repeatedly improving itself using stochastic gradient descent (updating the weights using one randomly chosen training example at a time). This task created the basic building blocks that are reused later: the forward prediction step, the loss calculation, and the training loop that updates parameters. To make sure the implementation was correct, training was tested on a small setup and monitored using training graphs. The most important check was that the model's cost consistently goes down as training continues. In the FIG1.1, the cost drops quickly at the beginning and then slowly flattens, which shows the model is learning the pattern and then stabilising near its best solution.

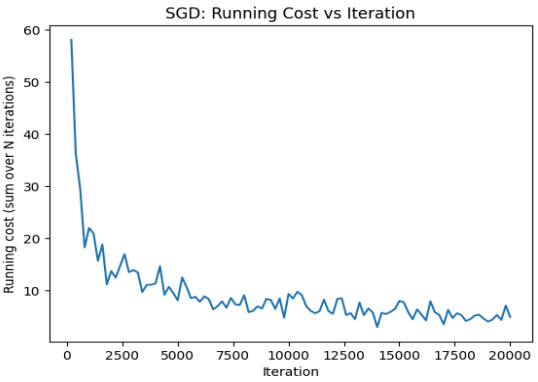


FIG 1.1 Running cost vs iteration plot

FIG 1.2 showing the change in running cost between checkpoints becomes smaller over time, which is another sign that training is reaching convergence. Small bumps in both graphs are normal because SGD updates are based on random samples, so the cost does not decrease perfectly smoothly at every checkpoint.

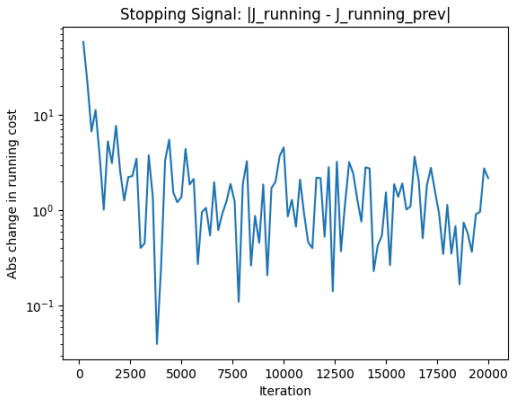


FIG 1.1 Stopping signal plot

For this task, the key training setting was the learning rate (alpha), which controls how big each weight update is. A reasonable learning rate leads to a stable decrease in cost without divergence, and the training curves confirm that the chosen value worked properly. Overall, Task 1 successfully produced a clean baseline logistic regression implementation and verified that it trains as expected using the included loss and convergence plots.

2. TASK 2 - Logistic Regression on Two Small Datasets

Task 2 evaluates the logistic regression model from Task 1 on two small datasets provided for the assignment. The main aim is to follow a proper workflow (train/validation/test split, feature scaling, hyperparameter tuning) and then compare how logistic regression behaves on a dataset that is close to linearly separable versus a dataset that is clearly non-linear. For both datasets, the same steps are used: the data is randomly split into training (70 percent), validation (15 percent), and test (15 percent); inputs are standardised using only training-set statistics; multiple learning rates are tried; the best learning rate is selected using validation accuracy; and the final performance is reported on the test set along with visual plots.

2.1. Dataset 1: blobs600 (3 input features, 600 samples)

The blobs600 dataset contains 600 total samples, each with 3 input attributes. After splitting, the dataset sizes were: Train (420 samples), Validation (90 samples), and Test (90 samples). Logistic regression performed extremely well on this dataset. Across the tested learning rates (alpha values from 0.001 to 0.2), the model achieved almost identical results, with training accuracy around 0.995 to 0.998, validation accuracy around 0.989, and test accuracy reaching 1.000. The fact that the scores are consistently high for many learning rates suggests two things: first, the dataset is easy for logistic regression, and second, training is stable and not very sensitive to the learning rate. In practical terms, blobs600 is structured in a way that a linear decision boundary can separate the classes effectively.

The scatter plot (Fig2.1.1) for blobs600 supports this conclusion visually. The points form two clear clusters in feature space, and the predicted class colours match the cluster structure almost perfectly. Very few (or zero) points appear incorrectly coloured, which matches the perfect test accuracy. If a confusion matrix heatmap is included for blobs600, it is expected to show very large values on the diagonal (true negatives and true positives) and values close to zero off the diagonal (false positives and false negatives), indicating that the classifier is almost never confusing the classes.

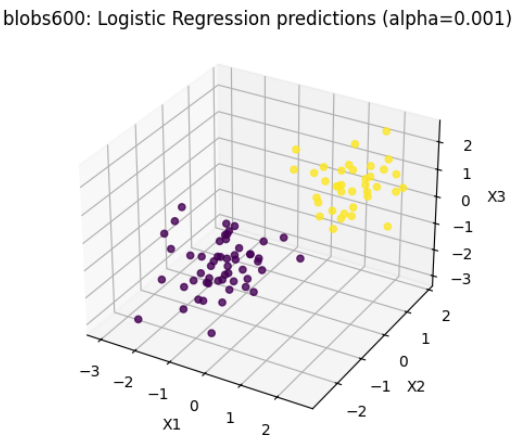


FIG 2.1.1 Blobs600 LR predictions

2.2. Dataset 2: circles500 (2 input features, 500 samples)

The circles500 dataset contains 500 total samples, each with 2 input attributes. After splitting, the dataset sizes were: Train (350 samples), Validation (75 samples), and Test (75 samples). Unlike blobs600, logistic regression struggles on circles500. A learning-rate search was performed using alpha values from 0.001 to 0.2. The best model was selected using validation accuracy, and the highest validation score achieved was about 0.52, with test accuracy around 0.51. Since random guessing in a balanced binary task would give about 0.50 accuracy, these results show that logistic regression is not learning a strong useful decision boundary for this dataset. The reason becomes clear when looking at the structure of circles500. The data forms an inner ring and an outer ring (a curved separation), but logistic regression is a linear classifier, meaning it can only draw a single straight boundary in the original input feature space. No matter how long it trains or which learning rate is chosen, a straight boundary cannot cleanly separate an inner circle from an outer circle, so performance remains low. The learning rate alpha = 0.1 was selected because it gave the best validation accuracy (tied with 0.2), but even this best setting cannot overcome the model limitation.

The scatter plot(Fig 2.2.1) for circles500 makes this failure visible. Instead of producing a clean separation between inner and outer circles, the predicted class colours appear mixed across the rings, showing that the model is essentially drawing an incorrect linear split through a problem that needs a curved split. This visual evidence aligns with the poor accuracy values.

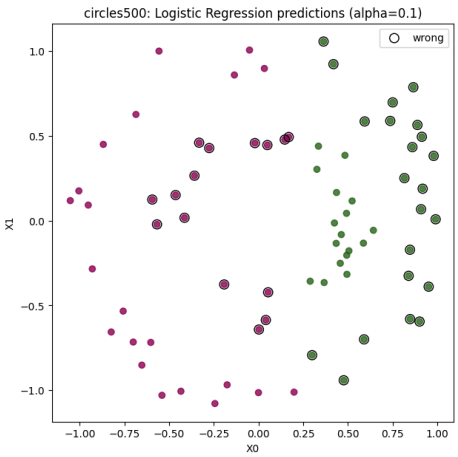


Fig 2.2.1 Circles500 LR predictions

The confusion matrix(Fig 2.2.2) for circles500 provides a detailed numeric breakdown of the errors. In the results observed, the test set confusion matrix values were: true negatives equal to 21, false positives equal to 22, false negatives equal to 15, and true positives equal to 17. This means only 38 out of 75 test samples were classified correctly, giving a test accuracy of 38 divided by 75, which is approximately 0.507. The off-diagonal values (false positives and false negatives) are large, meaning the classifier frequently swaps the two classes. In a confusion matrix heatmap, this appears as significant colour intensity away from the diagonal, confirming that the model is not reliably separating the classes.

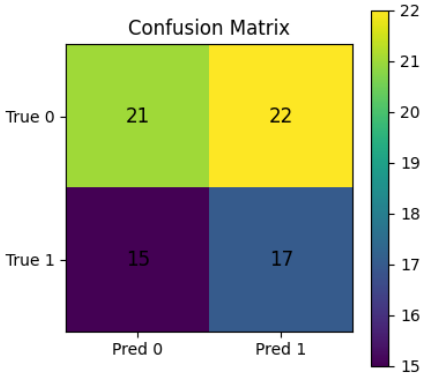


Fig 2.2.2 Confusion matrix

2.3 Training curve and convergence behaviour

For both datasets, training curves were plotted using the running cost recorded during SGD training. For blobs600, the running cost drops quickly and then becomes small and stable, which matches the high accuracy and indicates the model converges easily. For circles500, the cost may decrease somewhat, but it typically does not lead to strong accuracy improvements because the model class is fundamentally mismatched to the data's shape. Any small bumps in the training curve are expected because SGD uses random samples per update, which naturally causes noisy progress rather than a perfectly smooth curve.

Fig 2.3 Training Curve

Task 2 demonstrates the main strength and weakness of logistic regression. When the dataset is close to linearly separable (blobs600), logistic regression achieves near-perfect results and generalises extremely well. When the dataset requires a non-linear separation (circles500), logistic regression performs close to random guessing, and both the scatter plot and confusion matrix clearly show frequent misclassification. This is an important motivation for Task 3: adding a hidden layer introduces non-linear transformations, allowing a neural network to learn curved decision boundaries that logistic regression cannot represent

5. TASK 3 - Shallow Neural Network (One Hidden Layer) on the Two Small Datasets

Task 3 upgrades the logistic regression model from Task 1 into a shallow neural network by adding exactly one hidden layer. The main purpose of this task is to show that introducing a hidden layer allows the model to learn non-linear patterns that logistic regression cannot. The same training workflow from Task 2 is kept: the earlier train/validation/test splits are reused, training is done using the same SGD-style loop, different hidden-layer sizes are tested, and results are compared using validation accuracy (with test accuracy used only for final reporting). This task also checks that the implementation supports different numbers of input features (2D for circles and 3D for blobs) and different numbers of hidden nodes, as required.

3.1 circles500 (2 input features, non-linear dataset)

The circles500 dataset is the key test here because it is not linearly separable. Logistic regression in Task 2 was close to random guessing on this dataset, so a big improvement is expected if the shallow neural network is working correctly. The shallow network was tested with multiple hidden-layer sizes, and the results show a very clear pattern:

1. With h = 2, the model improves compared to logistic regression, but performance is still limited: training accuracy is 0.757, validation accuracy is 0.667, and test accuracy is 0.667. This suggests that with only 2 hidden nodes, the network has some non-linear ability, but not enough capacity to fully learn the circular boundary.
2. With h = 4, performance jumps dramatically: training accuracy becomes 0.994, validation accuracy becomes 0.973, and test accuracy reaches 1.000. This indicates the network has enough hidden capacity to form a strong non-linear decision boundary.
3. With h = 8, the model reaches perfect performance: training accuracy is 1.000, validation accuracy is 1.000, and test accuracy is 1.000. This was selected as the best configuration because it achieved the highest validation accuracy.
4. With h = 16, performance remains extremely strong, with training accuracy at 1.000 and test accuracy at 1.000, but validation accuracy slightly drops to 0.987. This small drop is a sign that adding more hidden nodes is not necessarily helpful and can start to introduce mild overfitting or unnecessary complexity.

Overall, circles500 shows exactly why hidden layers matter. Once the network has enough hidden nodes (h = 4 or higher), it can learn a curved boundary and separate the inner and outer rings correctly, which logistic regression could not do. The training curve plots(Fig 3.1.1) for circles500 also support this. The running cost decreases over time and eventually flattens out near zero, meaning the model is learning successfully and reaching a stable solution. The "stopping signal" plot (Fig 3.1.2) shows that improvements become small as training continues, which indicates convergence. Some noise in the curve is expected because SGD uses random samples at each update. The circles500 scatter plot(Fig 3.1.3) of predictions is the strongest visual proof. With the best hidden size, the plot shows a clean separation: one class occupies the inner ring and the other occupies the outer ring, with almost no incorrectly coloured points. This matches the perfect test accuracy reported.

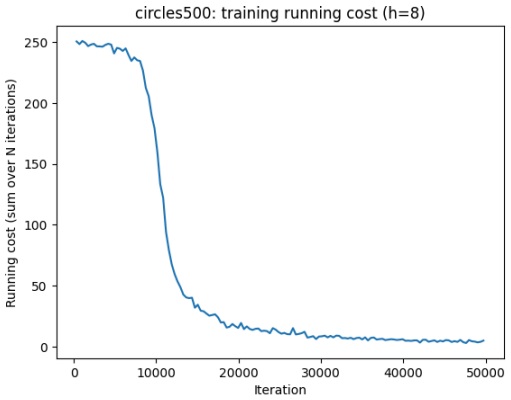


Fig 3.1.1 Training curve-circles

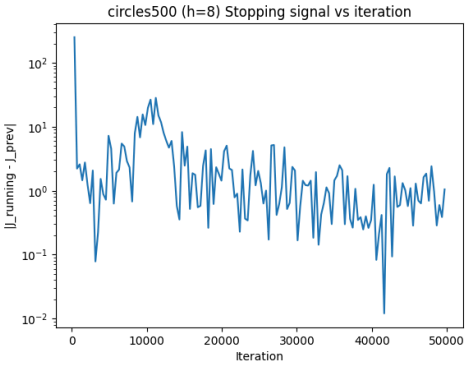


Fig 3.1.2 Stopping signal

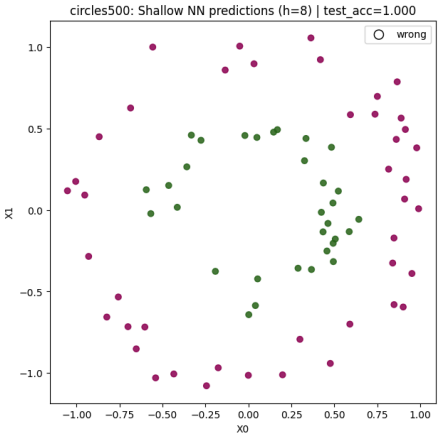


Fig 3.1.3 Shallow NN predictions on circles500

3.2 blobs600 (3 input features, mostly linearly separable dataset)

The blobs600 dataset is much easier. It is already close to linearly separable, so logistic regression in Task 2 was already near-perfect. The shallow neural network was still tested with different hidden sizes to satisfy the task requirement and to observe whether increasing hidden nodes changes performance. The results show that hidden size does not matter much here because the dataset is already easy:

1. With h = 2, training accuracy is 0.998, validation accuracy is 0.989, and test accuracy is 1.000.

2. With h = 4, the results are essentially the same: training 0.998, validation 0.989, test 1.000.
3. With h = 8, again the same pattern: training 0.998, validation 0.989, test 1.000.
4. With h = 16, still the same: training 0.998, validation 0.989, test 1.000.

The best hidden size was chosen as h = 2 by validation accuracy, but in practice all hidden sizes performed equally well. This indicates that the dataset does not require extra model complexity; even a simple non-linear model behaves like a strong classifier here because the clusters are already well separated. The blobs600 training curve(Fig 3.2.1) typically drops fast and stabilises early, which matches the idea that the model finds a good solution quickly. The 3D scatter plot(Fig 3.2.2) of predictions also matches the results: the clusters are cleanly separated, and the predicted colours align strongly with the true class structure. Because blobs600 is a 3D dataset, the scatter plot is shown in 3D, but the conclusion is the same: almost no points appear misclassified, which matches the perfect test accuracy.

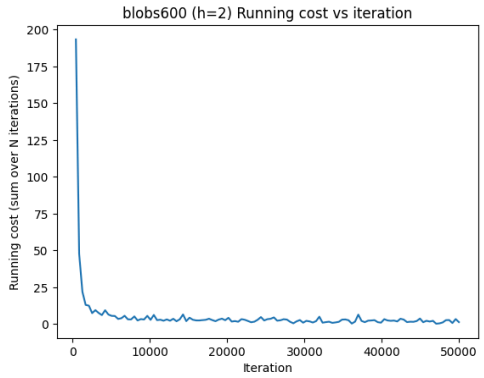


Fig 3.2.1 Training curve- blobs

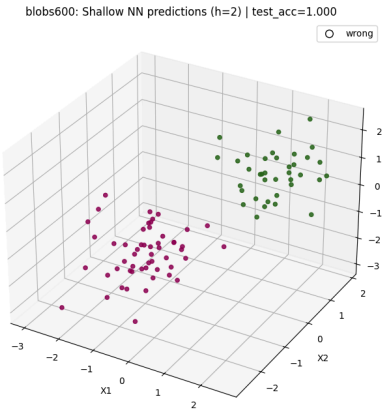
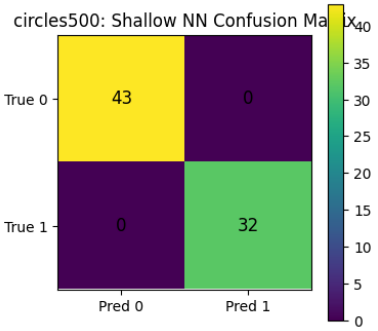
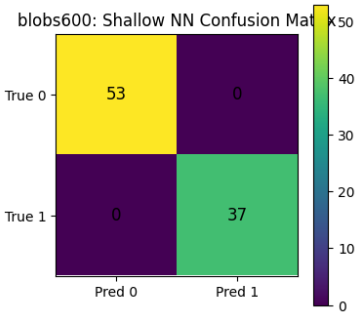


Fig 3.2.2 Shallow NN predictions on circles500





Task 3 confirms that adding one hidden layer dramatically increases what the model can learn. On circles500, which requires a non-linear boundary, the shallow network improves from near-random performance (Task 2 logistic regression) to near-perfect performance once enough hidden nodes are used. On blobs600, which is already easy and close to linearly separable, the shallow network also performs extremely well, but increasing hidden nodes does not change results because the dataset does not need extra complexity. This task demonstrates that hidden layers are especially valuable when the data contains non-linear structure, and it provides a strong foundation for Task 4 where the same shallow network is applied to high-dimensional handwritten image inputs.

4. TASK 4 - Handwritten Letter Recognition (EMNIST Letters: F vs L)

Task 4 applies the shallow neural network from Task 3 to a real-world classification problem using handwritten letter images from the EMNIST Letters dataset. The dataset originally contains 26 classes (letters), but the assignment requirement is to classify only two specific classes. The given class pair for this work was F and L. Each image is 28 by 28 pixels and already scaled to the range 0 to 1, so every image is flattened into a single vector of 784 input values before training. Compared to Task 3, this is a much higher-dimensional input space, so a larger hidden layer is needed to capture meaningful patterns from pixel data. The workflow in Task 4 follows the same structure used earlier: first the dataset is loaded using the provided starter code, then only the samples belonging to the two target letters are selected, and labels are converted into a binary format. In this

notebook, class 0 represents F (Fig 4.1) and class 1 represents L(Fig 4.2). After extracting these two classes, the data is split into training, validation, and test sets, following the same idea as before: the training set is used to learn weights, the validation set is used to pick good settings, and the test set is used only for the final performance evaluation. Feature standardisation is then applied based on training-set statistics, and the same transformation is applied to validation and test data for consistency.

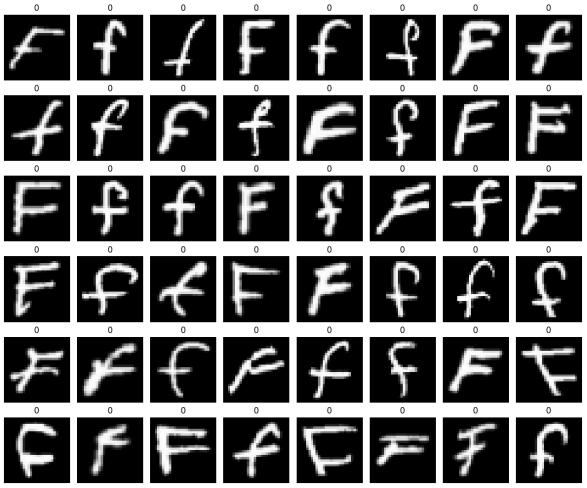


Fig 4.1 F class

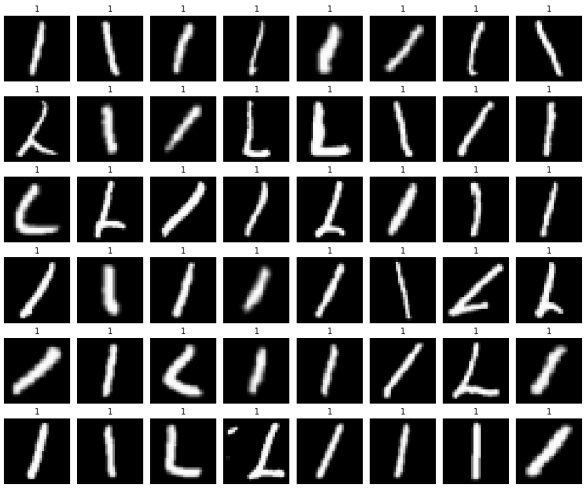


Fig 4.2 L class

To achieve strong performance, multiple combinations of hidden layer size and learning rate were tested. Hidden sizes of 32, 64, and 128 were evaluated, along with learning rates of 0.05, 0.1, and 0.2. Validation accuracy was used as the selection rule. The results showed that performance was consistently high across many settings, but the best validation score came from a hidden size of 128 with learning rate 0.1, producing a validation accuracy of about 0.988. This model was chosen as the final Task 4 model. The final reported performance for the selected model was very strong: training accuracy reached 1.000, validation accuracy was about 0.988, and test accuracy was about 0.993. The confusion matrix(Fig 4.3) provides a more detailed breakdown of errors. For the test set, true negatives were 497 and true positives were 486, meaning most samples were correctly classified. Errors were very small: only 4 cases where an F was incorrectly predicted as L (false positives) and 3 cases where an L was incorrectly predicted as F (false negatives). In other words, only 7 test samples were misclassified in total, which explains the near-99 percent test accuracy.

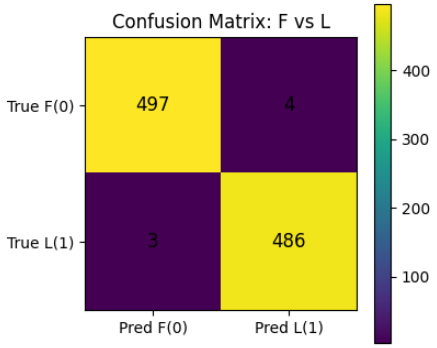


Fig 4.3 Confusion matrix

The figures included in Task 4 support these results clearly. The training curve(Fig 4.4) shows the cost dropping quickly and stabilising near a very small value, indicating that the model has converged. A prediction-confidence plot (probability histogram) Fig 4.5 shows that the model typically outputs values close to 0 for F and close to 1 for L, meaning it is not only accurate but also confident for most samples. The ROC curve(Fig 4.6) lies near the top-left corner, far above the random-guess diagonal, which indicates excellent separation between the two classes across different thresholds. Finally, the visual display of misclassified images shows that the few mistakes come from ambiguous handwriting cases where letter shapes are unclear or incomplete, which is expected in handwritten datasets.

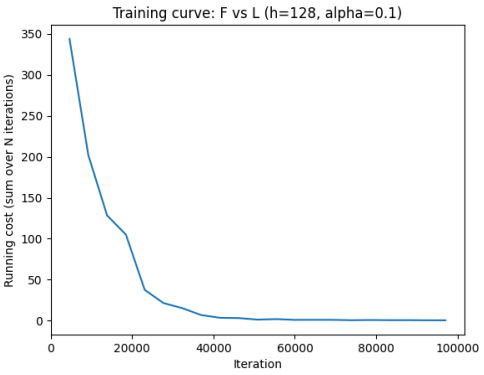


Fig 4.4 Training curve

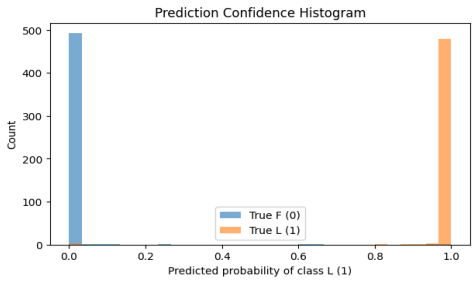


Fig 4.5 Prediction Confidence

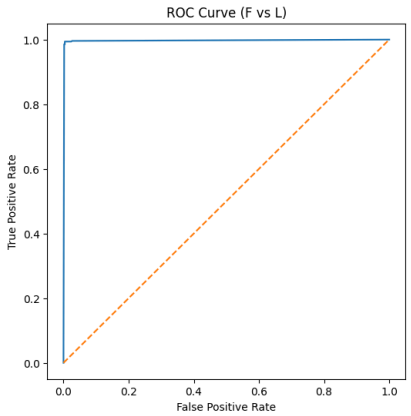


Fig 4.5 ROC curve

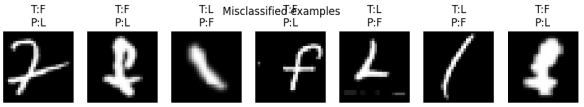


Fig 4.7 Misclassified examples

Overall, Task 4 demonstrates that the one-hidden-layer neural network implemented in Task 3 generalises well to real image data when given a sufficiently large hidden layer. The model successfully learns pixel-level patterns that distinguish F from L, achieves high validation and test accuracy, and produces supporting evaluation visuals (training curve, confusion matrix, ROC curve, and error examples) that confirm the quality of the classifier.

5. TASK 5 - DeepNetwork (Multiple layers) + Overfitting control

Task 5 extends the neural network implementation so that it can support an arbitrary number of layers, with an arbitrary number of nodes in each layer. In Task 3 and Task 4, the network had exactly one hidden layer. In this task, the same idea is generalised so that the architecture can be defined by a simple list of layer sizes, for example: 784, 128, 1 for a one-hidden-layer model, or 784, 128, 64, 1 for a two-hidden-layer model. This makes the code flexible, because deeper models can be tested without rewriting the training logic. Along with this extension, Task 5 also introduces a method to reduce overfitting, selected from lecture-discussed options. The chosen method here is L2 regularisation (weight decay), which discourages very large weights and typically improves generalisation when a model is overly complex for a small dataset. The evaluation requirement of Task 5 is to test the deep network on the same dataset used in Task 4 (EMNIST F vs L) and then compare results. This means the training, validation, and test splits remain the same so that performance comparisons are fair. The task specifically asks for a comparison between shallow versus deeper architectures, and also between training with and without the overfitting control method. For that reason, four main experiments were run: one hidden layer without L2, one hidden layer with L2, two hidden layers without L2, and two hidden layers with L2. All experiments used the same training procedure as before (SGD, training curve tracking) and were compared using validation accuracy as the selection rule, with test accuracy reported afterward.

The results show that the one-hidden-layer model without L2 performed best overall. It achieved training accuracy of 1.000, validation accuracy of 0.9879, and test accuracy of 0.9929. Adding L2 regularisation reduced training accuracy to 0.9881 and also slightly reduced validation and test accuracy, giving validation accuracy of 0.9818 and test accuracy of 0.9879. This indicates that the original model was already generalising well, so adding weight constraints made the model slightly less flexible than needed for this particular task. When increasing depth to two hidden layers, the model still performed very strongly, but did not improve on validation or test. The two-hidden-layer model without L2 reached training accuracy of 1.000, validation accuracy of 0.9828, and test accuracy of 0.9919, while the two-hidden-layer model with L2 achieved training accuracy of 0.9874, validation accuracy of 0.9808, and test accuracy of 0.9899. In short, deeper models did not provide a benefit on this dataset, and regularisation slightly reduced accuracy in all tested settings. The training-curve(Fig 5.1) comparison plot supports these findings. The models trained without L2 show the running cost dropping rapidly and reaching very small values, which aligns with perfect training accuracy.

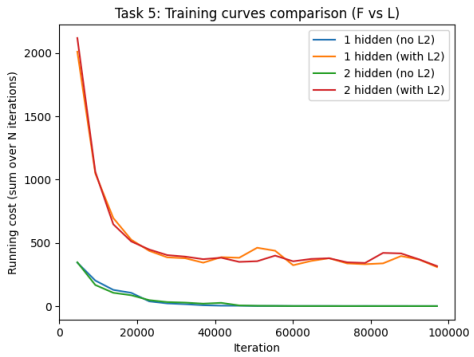


Fig 5.1 Training curve

The L2-regularised models show higher training cost curves and do not drop as close to zero, which is expected because regularisation deliberately prevents the model from fitting the training data too perfectly. Despite this, validation performance did not improve with L2 here, suggesting that overfitting was not the main issue for this dataset and architecture, especially given that the baseline model already achieved strong generalisation.

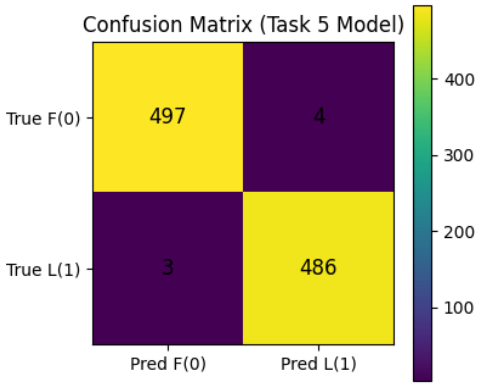


Fig 5.2 Confusion Matrix

Overall, Task 5 confirms that the deep-network implementation works correctly and allows experiments with different depths and layer sizes. However, for the specific F vs L classification task, the best performing model remained the simpler one-hidden-layer network without regularisation. This outcome is reasonable because the task appears highly separable in the chosen feature representation, so additional depth adds complexity without improving generalisation. The required comparison is therefore satisfied: one hidden layer versus two hidden layers is reported, and with-versus-without regularisation is reported, all evaluated on the Task 4 dataset with training curves and accuracy metrics included.

6. CONCLUSION

This assignment successfully demonstrates a complete end-to-end implementation of neural networks from first principles, progressing from a simple baseline model to deeper architectures. Logistic regression (Task 1) provided a solid foundation and worked well when the underlying class structure was close to linear. When evaluated on small datasets (Task 2), it achieved near-perfect performance on the linearly separable dataset (blobs600) but performed close to random guessing on the non-linear dataset (circles500), clearly showing the limitation of linear decision boundaries. Introducing one hidden layer (Task 3) significantly increased model capability. The shallow neural network learned a non-linear boundary and achieved perfect or near-perfect classification on circles500 once the hidden layer contained enough nodes, while blobs600 remained easy regardless of hidden size. The same shallow network was then applied to a higher-dimensional real dataset (Task 4) using EMNIST handwritten letters restricted to F versus L. With an appropriately larger hidden layer, the model achieved very strong generalisation, reaching approximately 99 percent test accuracy with only a small number of misclassifications, supported by the training curve, confusion matrix, ROC curve, and error examples.

Finally, the implementation was extended to support an arbitrary number of layers and an overfitting control method (Task 5). Experiments comparing one hidden layer versus two hidden layers, and training with versus without L2 regularisation, showed that the simpler one-hidden-layer model without regularisation performed best on validation and test for this particular task. This indicates that additional depth and regularisation were not necessary here because the F versus L problem was already well captured by a single hidden layer and the model generalised strongly without extra constraints.

Overall, the results across all tasks highlight an important practical lesson: increasing model complexity helps when the data requires non-linear structure (as in circles500), but extra layers or regularisation do not automatically improve performance when the problem is already separable and the baseline model generalises well.

7. REFERENCES

[1] W3Schools. NumPy Array Reshape - Flattening Arrays. Available at: https://www.w3schools.com/python/numpy/numpy_array_reshape.asp

[2] W3Schools. NumPy Data Types - Datatype Conversion. Available at: https://www.w3schools.com/python/numpy/numpy_data_types.asp

[3] Towards Data Science. 'Let's Code a Neural Network in Plain NumPy' - Activation Functions. Available at: https://medium.com/data-science/lets-code-a-neural-network-in-plain-numpy-ae7e74410795

[4] GeeksForGeeks. Implementation of Logistic Regression from Scratch using Python. Available at: https://www.geeksforgeeks.org/implementation-of-logistic-regression-from-scratch-using-python/

[5] GeeksForGeeks. Binary Cross-Entropy / Log Loss for Binary Classification. Available at: https://www.geeksforgeeks.org/binary-cross-entropy-log-loss-for-binary-classification/

[6] xbeat / Machine-Learning. Explaining Log Loss Using Python — Clipping in Log-Loss. Available at: https://github.com/xbeat/Machine-Learning/blob/main/Explaining%20Log%20Loss%20Using%20Python.md

[7] Analytics Vidhya. Gradient Descent vs SGD vs Mini-Batch SGD. Available at: https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4

[8] Real Python. Gradient Descent Algorithm in Python — SGD. Available at: https://realpython.com/gradient-descent-algorithm-python/

[9] KoshurAI. Understanding and Implementing Standardisation in Python. Available at: https://koshurai.medium.com/understanding-and-implementing-standardization-in-python-86214518da8d

[10] Data Science 365. One Hidden Layer Shallow Neural Network Architecture. Available at: https://medium.com/data-science-365/one-hidden-layer-shallow-neural-network-architecture-d45097f649e6

[11] GeeksForGeeks. Backpropagation in Neural Networks. Available at: https://www.geeksforgeeks.org/backpropagation-in-neural-network/

[12] CodeSignal. Confusion Matrix, Precision and Recall in Classification Metrics. Available at: https://codesignal.com/learn/courses/classification-algorithms-and-metrics/lessons/understanding-the-confusion-matrix-precision-and-recall-in-classification-metrics

[13] Medium. Optimal Number of Hidden Layers in a Neural Network. Available at: https://medium.com/@mohamedgele2000/optimal-number-of-hidden-layers-in-neural-network-should-we-prefer-a-large-hidden-layer-or-a-small-25aec51f101b

[14] Machine Learning Mastery. Ridge Regression (L2 Penalty) with Python. Available at: https://machinelearningmastery.com/ridge-regression-with-python/

[15] GeeksForGeeks. How to Split a Dataset into Train and Test Sets using Python. Available at: https://www.geeksforgeeks.org/how-to-split-a-dataset-into-train-and-test-sets-using-python/