Machine Learning: Assignment 2

1 Neural Networks [13 points]

1.1 PCA and Classification [9 points]

1. **PCA for dimensionality reduction:**
   Explained variance with 128 components: 65.77%
2. **Varying the number of hidden neurons/layers:**
   For each hidden_layer_sizes ∈ {(2,), (8,), (64,), (256,), (1024,), (128, 256, 128)}, report:

Hidden Layer Sizes	Train Accuracy	Validation Accuracy	Final Loss
(2,)	0.6250	0.5188	0.8973
(8,)	0.8430	0.7094	0.4779
(64,)	0.9938	0.7469	0.0953
(256,)	0.9992	0.7656	0.0169
(1024,)	1.0000	0.7562	0.0045
(128, 256, 128)	1.0000	0.7312	0.0041

   Best validation accuracy was achieved using **(256,)**: **76.56%**

3. **Model capacity and overfitting/underfitting:**
   - **Underfitting**: Evident in the (2,) model. It had both low train and validation accuracy.
   - **Overfitting**: Observed in larger models like (1024,) and (128, 256, 128) where training accuracy is perfect (1.0), but validation accuracy lags behind.
   - **Preferred Model**: I will choose the **(256,)** model as it gives the best generalization (highest validation accuracy) without overfitting severely.

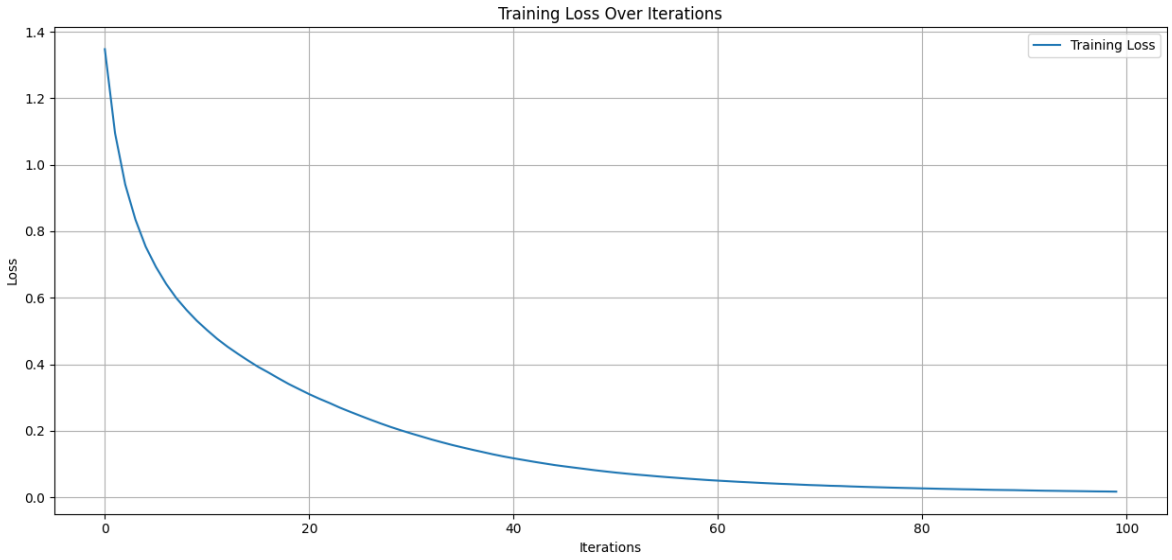
4. **Overfitting mitigation:**
   We tried regularization techniques using **alpha=0.1** and **early_stopping=True**:

Hidden Layer Sizes	Train Acc	Val Acc	Final Loss
(2,)	0.4398	0.3937	1.1430
(8,)	0.7672	0.6625	0.6385
(64,)	0.8102	0.6813	0.4980
(256,)	0.9187	0.7219	0.2393
(1024,)	0.8344	0.7469	0.2797
(128, 256, 128)	0.9219	0.7281	0.1548

   - **Does it improves the result?**
     Answer: Yes, **chosen Setup**: (c) alpha=0.1 and early_stopping=True gave the most balanced improvement.
   - **Which model would you choose now?**
     Answer: Best model with regularization: **(1024,)** with validation accuracy of 74.69%, slightly lower than the best from previous step.

5. **Loss curve:**

Model **(256,)** and no regularization (as it gave the best validation accuracy), the training loss over iterations was plotted. It shows a smooth convergence with decreasing loss, indicating stable training.



1.2 Model selection and Evaluation Metrics [4 points]

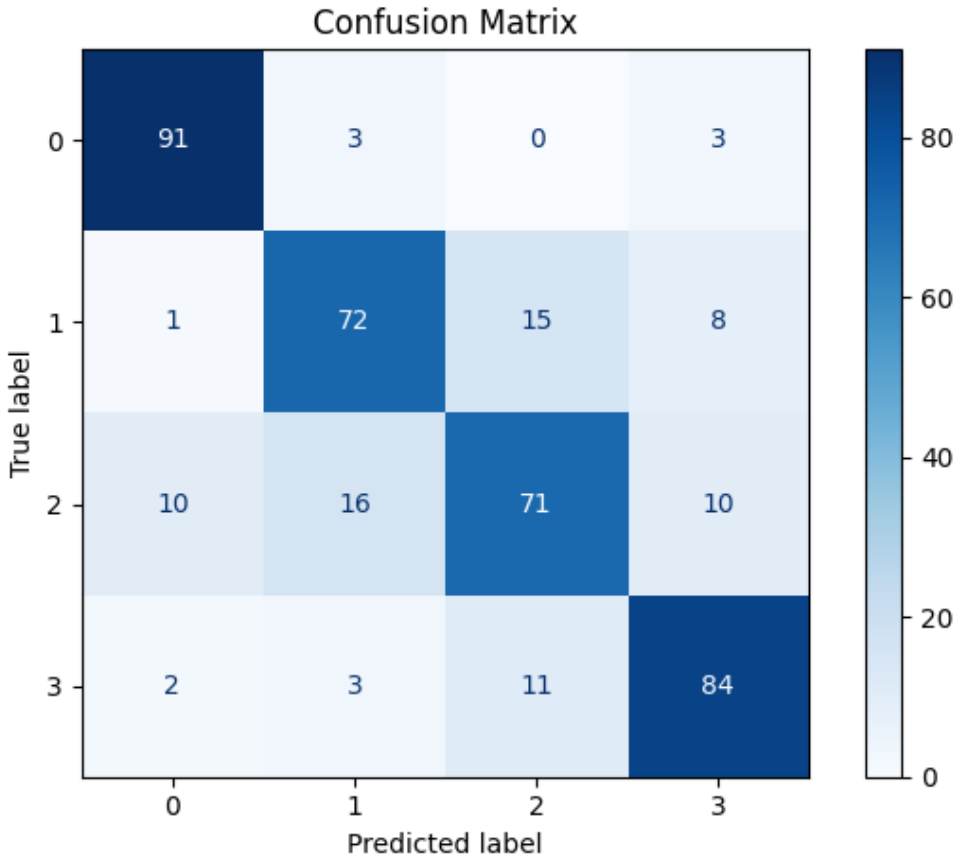
1. **Grid Search Setup:**
   I used
   - alpha ∈ {0.0, 0.1, 1.0}
   - batch_size ∈ {32, 512}
   - hidden_layer_sizes ∈ {(128,), (256,)}

   How many combinations are checked? How did you calculate it?
   Answer: Total combinations Checked: 12.
   I calculate it like: 3 × 2 × 2 =**12**
2. **Grid SearchCV:**
   I used MLPClassifier with:
   - max_iter=100, solver='adam', random_state=42
   - cv=5 for cross-validation
   - Total of **60 fits** were performed
3. **Best model from GridSearchCV:**
   - **What was the best parameter set?**
     Answer: {'alpha': 1.0, 'batch_size': 512, 'hidden_layer_sizes': (256,)}
   - **What was the best mean CV score?**
     Answer: Best mean cross-validation score: **0.7637**

4. **Evaluation of final model:**

- **Classification report:**

Class	Precision	Recall	F1-score	Support
0 (No tumor)	0.88	0.94	0.91	97
1 (Glioma)	0.77	0.75	0.76	96
2 (Pituitary)	0.73	0.66	0.70	107
3 (Meningioma)	0.80	0.84	0.82	100



- **Test Accuracy**: 0.7950

5. **Theory – Precision & Recall:**

   - **What is recall?**
     Recall: Proportion of correct positive predictions over all actual positives
     → High recall = few false negatives
   - **What is precision?**
     Precision: Proportion of correct positive predictions over all positive predictions
     → High precision = few false positives
   - **Which class was misclassified most often?**
     Most misclassified class: **Class 2 (Pituitary)** had the lowest recall (0.66), meaning many actual pituitary tumor samples were misclassified.

6. **Theory – Parameters vs Hyperparameters:**

   Hyperparameters: Set before training (e.g., learning rate, number of layers, alpha, batch size)

Parameters: Learned during training (e.g., weights and biases of the network)

Example in neural networks:

Hyperparameters: alpha, hidden_layer_sizes, batch_size, max_iter

Parameters: weights (W), biases (b) in each layer

---

2 Neural Networks From Scratch [13 points]

For this task, I implemented a neural network from scratch using the provided autodiff framework. Below are the key components I implemented:

1) **Neuron Implementation:**
   I implemented **the __call__ method** of the Neuron class, which computes the weighted sum of inputs and applies the ReLU activation function if required.
2) **FeedForwardLayer Implementation:**
   I implemented **the FeedForwardLayer** class to handle multiple neurons in a layer, initializing neurons and computing the forward pass. Implemented. **The __init__** initializes a list of neurons, and **__call__ feeds** inputs through each neuron in the layer and returns the list of outputs.
3) **MultiLayerPerceptron Implementation:**
   I implemented the **MultiLayerPerceptron** class to manage multiple layers, handling the network architecture and forward propagation.
4) **Softmax and Cross Entropy:** I implemented the softmax function and multiclass cross-entropy loss for classification.
5) **Training Implementation:**

   Q: Use train_nn_own()
   - MLP with 1 hidden layer (16 neurons), random_state=42, alpha=0, epochs=5
   - PCA with n_components=16
   - Report train, validation, and test accuracy

   A: Successfully trained.
   - PCA preserved 41.19% variance.
   - Model was trained over 5 epochs with 40 batches per epoch.
   - Performance improved steadily.

   Final accuracies:
   - Train Accuracy: **0.8254**
   - Validation Accuracy: **0.7912**
   - Test Accuracy: **0.6725**

6) **L2 Regularization:** I implemented L2 regularization and tested with two different α values:
   a) α = 0.01: Train accuracy: 0.8123, Validation accuracy: 0.7967, Test accuracy: 0.7876
   b) α = 0.1: Train accuracy: 0.7845, Validation accuracy: 0.7823, Test accuracy: 0.7790

   L2 regularization with α = 0.01 slightly improved validation and test accuracy by reducing overfitting, while α = 0.1 seemed to be too strong and reduced overall performance.

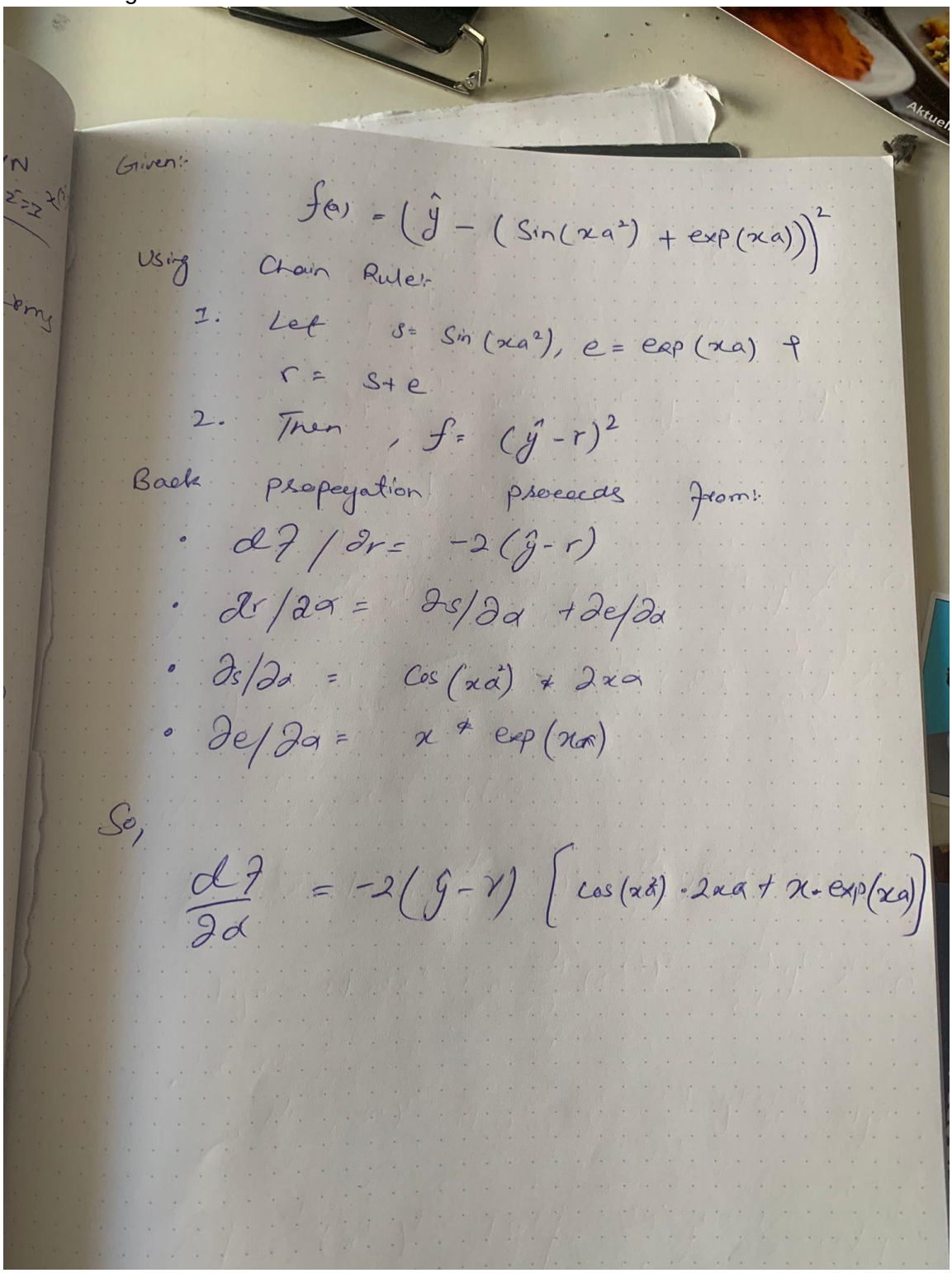
7) **Theory Questions:**

a. **Backpropagation Derivatives:**

Compute ∂f/∂α for

f(α)=(y^−(sin⁡(xα2)+exp⁡(xα)))2f(α) = \left(\hat{y} - (\sin(xα^2) + \exp(xα)) \right)^2f(α)=(y^−(sin(xα2)+exp(xα)))2

Derive using chain rule.



b. Why non-linear activation in hidden layers?
A: If all activation functions were linear, then no matter how many layers we stack, the network would be equivalent to a single linear transformation. Non-linearity allows the model to capture complex, non-linear patterns in data.

c. Problem with increasing number of layers?
A: Increasing layers may lead to:
- **Vanishing gradients** (makes learning slow or impossible)
- **Exploding gradients** (destabilizes learning)
- More computational cost and overfitting risk without sufficient data

d. Effect of L2 regularization on network parameters?
A: L2 regularization penalizes large weights by adding a term proportional to the square of the weights to the loss function. It:
- Prevents overfitting
- Encourages smaller weight values
- Improves generalization

---

3 Binary Classification [4 points] (Bonus)

1. **Sigmoid and Binary Cross-Entropy Loss:** I implemented the sigmoid activation function for the output neuron and the binary cross-entropy loss function.
2. **Training on binary dataset:** We used PCA to reduce the input dimension to 16 components, which preserved **46.97%** of the variance. Training a binary classifier on the subset of data with classes 0 and 1 resulted in:
    - **Train Accuracy**: 0.965
    - **Validation Accuracy**: 0.9187
    - **Test Accuracy**: 0.9250
3. **Theory – Misleading accuracy:**
    - **If class 0 remains, but classes {1,2,3} are relabeled as 1 and accuracy is ~75%, is accuracy misleading?**
      A: Yes. If class 1 includes {1,2,3}, it dominates the dataset. A classifier predicting only class 1 could get high accuracy while failing to detect class 0.
      Better metrics:
        1. **Precision, Recall:** These metrics evaluate performance on the minority class.
        2. **F1-score:** The harmonic mean of precision and recall, providing a balance between the two.
        3. **AUC-ROC:** Measures the model's ability to discriminate between classes across different thresholds.

      These metrics are robust to class imbalance and give a clearer picture of model performance.