COS 711 Assignment 2 Report

Nathan Opperman

u21553832@tuks.co.za

github https://github.com/nathanMils/711_Ass1

***Abstract*—In order to compare and contrast different training algorithms I believe it is appropriate to,**

***Index Terms*—This probably is not necessary**

I. Introduction

In this assignment, I will be evaluating and comparing various hyperparameters and specifically training algorithms to understand the trade-offs of each approach. By exploring different configurations, such as learning rates, batch sizes, and network architectures, I aim to identify the optimal conditions to yield the best performance.

Before any training can be done, data must be prepared and so I will go over how the data has been investigated, analysed and transformed in order to ensure that data is correctly scaled, formatted and selected as to not hinder model performance.

Hyperparameter tuning can have a great affect, as the right choices can drastically enhance model performance while the wrong ones can lead to overfitting or underfitting amongst other issues. I will leverage techniques such as K-fold cross-validation and grid search to assess and compare the impact of these hyperparameters on model performance.

Moreover, I will investigate various gradient based training algorithms, including gradient descent variants and advanced optimization techniques like Adam and RMSprop. Each algorithm will be analyzed in terms of convergence speed and solution quality, allowing for a deep understanding of their applicability to different datasets and tasks.

The results of these experiments will then be presented as to highlight certain significant metrics such as accuracy, validation loss, and confusion matrices. By comparing these metrics across different configurations, I aim to provide insight into the design of a robust neural network.

Ultimately, this report will not only demonstrate the importance of careful hyperparameter selection and algorithm choice but also demonstrate my own understanding.

II. Background

A. Almonds

B. Multi-class Classification

C. Hyperparameter Optimization with K-Fold Cross Validation and Grid Search

D. Training Algorithms - Adam

E. Training Algorithms - Rprop

F. Training Algorithms - SGD

G. Hybrid Learning

A very high level discussion on the problem domain and the algorithms and/or approaches that you have used. This section is typically where the "base cases" of concepts that appear throughout the remainder of your report are discussed. It is also an ideal place to refer a reader to other sources containing relevant information on the topic which is outside the scope of your assignment. Remember to discuss very generally. After reading this section

the marker should be able to determine whether or not you understand the techniques that you are using. Try to limit this section to 1 page maximum. Make sure you reference the relevant sources when discussing the building blocks of your project.

III. Data Preparation

Data preparation is an important aspect of any learning process and is vital to enhancing the performance of any neural network models.

A. Dataset Description

The "Almond Types Classification" dataset was collected from Kaggle and comprises a total of 2,803 almonds. Each almond is described by fourteen attributes, including its ID and Type. The dataset features 2D images of almonds captured in three different orientations: upright, on its side, and laid on its back. The primary objective of this dataset is to classify each almond into one of three categories: SANORA, MAMRA, and REGULAR.

The attributes included are:

- **Id**: Unique identifier for each almond sample.
- **Length**: Length of the almond in the image (pixels).
- **Width**: Width of the almond (pixels).
- **Thickness**: Thickness of the almond (pixels).
- **Area**: Area of the almond region detected in the image (pixels).
- **Perimeter**: Total length of the almond boundary (pixels).
- **Roundness**: Measure of roundness calculated as $4 \times \text{Area}/(\pi \times \text{Length}^2)$.
- **Solidity**: Ratio of the area of the almond to its convex hull area, calculated as Area/area_hull.
- **Compactness**: Measure of compactness defined as $\text{Perimeter}^2/(4\pi \times \text{Area})$.
- **Aspect Ratio**: Ratio of the length to the width of the almond, calculated as Length/Width.
- **Eccentricity**: Measure defined as $\sqrt{1 - (\text{Width}/\text{Length})^2}$.
- **Extent**: Ratio of the area of the almond to the area of its bounding box, calculated as Area/area_bbox.
- **Convex hull (convex area, or area hull)**: The smallest convex set that contains the boundary points of the almond.
- **Type**: The almond type, which can be SANORA, MAMRA, or REGULAR.

Class Distribution

The almond types are distributed as follows:

- **SANORA**: 943 samples (33.64%)
- **MAMRA**: 933 samples (33.39%)
- **REGULAR**: 927 samples (33.07%)

This class distribution is rather fair so there should not be any issue regarding class bias at this point.

B. Data Cleaning - Missing Values

As you can see in Table I, one of the main issues of this dataset are the attributes: Length, Width and Thickness, whereby each almond is missing exactly one of these attributes. Specifically 30.57% are missing Length, 33.61% are missing Width and 35.82% are missing Thickness. This is clearly an extensive amount of missing information, and requires care consideration to ensure that information loss is kept to a minimal while without degrading the data.

To address this I decided to utilize the iterative imputer (IterativeImputer) from the scikit-learn library. This is a multivariate imputer that estimates each feature based on all the others. It uses a strategy for imputing values by modeling each feature with missing values as a function of others in a round-robin like fashion.

It is important to clarify that features like Roundness, Aspect Ratio, and Eccentricity, which depend on Length, Width, and Thickness, will be excluded from the imputation process. Including these dependent features would not provide valid imputations, as their values rely on the very attributes that are missing. Instead, I will compute these dependent attributes after imputation, ensuring accurate calculations based on the newly estimated values.

Furthermore, to enhance the reliability of

TABLE I
STATISTICAL SUMMARY OF FEATURES WITH MISSING VALUES

Feature	Count	Mean	Std Dev	Min	25%	50%	75%	Max	Missing Values
Length (major axis)	2803	1401.00	809.30	0.00	700.50	1401.00	2101.50	2802.00	857
Width (minor axis)	1946	290.61	62.72	151.34	245.97	279.88	330.51	515.35	942
Thickness (depth)	1799	109.71	18.94	59.49	97.09	110.28	121.39	181.85	1004
Area	2803	26511.12	13782.56	6037.00	16211.50	23440.50	33451.00	89282.50	0
Perimeter	2803	743.86	230.63	311.56	571.73	707.49	878.90	1864.95	0
Roundness	1946	0.47	0.12	0.17	0.38	0.47	0.58	0.70	857
Solidity	2803	0.96	0.04	0.72	0.94	0.97	0.98	0.99	0
Compactness	2803	1.83	0.79	1.16	1.36	1.58	1.97	9.66	0
Aspect Ratio	1004	1.75	0.21	1.40	1.61	1.71	1.83	2.73	1799
Eccentricity	1004	0.81	0.04	0.70	0.78	0.81	0.84	0.93	1799
Extent	2803	0.72	0.05	0.45	0.70	0.73	0.76	0.85	0
Convex hull (convex area)	2803	27696.22	14237.35	6355.00	17088.50	24589.00	34863.25	90642.50	0

computing Roundness, which is derived from both Length and Area. The issue here is that Area is dependent on the angle of the image, I introduced a temporary feature, Area_Length. The reason being is that Area is obviously dependent on the angle of the image taken, meaning it is highly dependent on weather length is present. So computing Roundness using Area that is not consistent with length will cause Roundness to be inconsistent. However Area_Length will retain the value of Area when Length is available, while remaining NaN when Length is absent. The IterativeImputer will then impute missing values for Area_Length, providing a more stable basis for accurately calculating Roundness.

C. Bias Handling

Neural networks are data-driven models, meaning their performance and predictions are highly dependent on the data they are trained on. This means that if our training data contains bias, the model will likely learn this bias and propagate it in its predictions on unknown data. This is a particularly important issue, luckily this dataset appears to represent each class equally with 943 (33.64%) SANORA almonds, 933 (33.39%) MAMRA almonds and 927 (33.07%) REGULAR almonds. Therefore each class is represented almost equally and so class bias should not be an issue.

D. Data Transformation - Encoding

In the dataset, the only non-numerical feature is the class attribute 'Type', which denotes the type of almond. Since neural networks require numerical input, it is essential to transform this categorical variable into a numerical format. For this purpose, I utilized label encoding, assigning a unique integer to each category. However, the integer values are treated as indices, effectively simulating one-hot encoding.

Label encoding involves converting each unique category in a nominal variable into a corresponding integer value. While this method is typically employed for ordinal data, in this case, the integer values are used as indices for the model. This allows the objective function to correctly interpret these values while maintaining the nominal nature of the data. For the 'Type' feature, which includes three unique categories: SANORA, MAMRA, and REGULAR, the encoding is illustrated in Table II.

TABLE II
ENCODING

Type	**1**	**2**	**3**	**encoding**
SANORA	1	0	0	1
MAMRA	0	1	0	2
REGULAR	0	0	1	3

E. Feature Selection

The feature ID (the unique identifier for each almond) wont be needed for our purposes, and hence it will be removed. This leaves us with Length, Width, Thickness, Area, Perimeter, Roundness, Solidity, Compactness, Aspect Ratio, Eccentricity, Extent and Convex hull.

F. Data Splitting

In order to train, optimize and evaluate the model I split the data into the standard 70:20:10 ratio, with 70% of the data allocated to training the model, 20% allocated to validation, and 10% allocated to testing. To ensure that the classes are represented equally in each set I used stratified sampling. With this approach the proportion of samples for each class remains near the same across the training, validation, and testing sets.

G. Data Transformation - Data Scaling

To prevent saturation for activation functions like sigmoid or tanh I utilized Z-Score normalization to scale all input features to zero mean and unit variance which helps avoid the issue of neurons becoming saturated. In order to attain the most realistic test error estimate, each set is scaled according to the training set.

IV. Experimental Set-Up

In this section, I will outline the approaches and strategies utilized throughout this assignment and provide rationale for the various choices made regarding data hyperparameters and hybrid learning.

A. Software Stack

To aid in this assignment, I used a Docker container to host and run JupyterLab locally. Specifically, I used the "jupyter/scipy-notebook_64-ubuntu-22.04" Docker image as a base. For the machine learning and neural network processes, I decided to use the PyTorch framework, known for its flexibility, efficiency and more importantly its simplicity. Additionally, I incorporated essential libraries such as Pandas and NumPy for data manipulation and preparation, along with scikit-learn (sklearn) for various preprocessing and modeling tasks. This software stack provided a comprehensive set of tool for addressing the task at hand.

B. NN model Architecture

There are four main elements that comprise a models architecture architecture:

- **The number of layers**: the more layers the more complex.
- **The number of neurons** (in each layer)
- **The activation function** (of each layer)
- **The training algorithm** (discussed later)

Since we already know that we will have an input layer and an output layer and the configuration (number of neurons, activation function) for such layers are essentially determined by the parameters and class variable, we only really need to investigate the hidden layers. There are many possible configurations that exist however I decided that this problem only required a simpler architecture.

1) Input Layer and Output Layer: As mentioned, the input and output layers require the least amount of discussion. Since we have twelve features, we will utilize twelve neurons in the input layer—one for each feature. These input neurons will accept the prepared feature values from our dataset and pass them to the subsequent layers. Since I am using Cross Entropy Loss we need an output neuron for each possible category. Since we have three categories (SANORA,MAMRA and REGULAR) we use three output neurons, each of which represent one category. Therefore the model will output three values, which will be passed through the Softmax activation function:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

This will convert them into probabilities which represent the likelihood of the input belonging to each class. However our implementation of Cross Entropy Loss (from PyTorch) actually does this internally so the output layer will technically not have an activation function.

Note: The softmax calculation is still included in the weight update calculation.

2) Hidden Layers: Activation Function - ReLU/Leaky ReLU, mention pros/cons and mention why fewer layers may be better here

C. Weight Initialization

- For ReLU - Kaiming Normal
- For Leaky ReLU - Kaiming Normal
- For PReLU - Kaiming Normal
- For Sigmoid - Xavier Normal
- For Swish - Xavier Normal

D. Loss Function - Multi-class Cross Entropy Loss

For this assignment, I believe the most suitable choice for my loss function is Multi-class Cross Entropy Loss with Softmax. Cross Entropy Loss is a popular and widely used loss function for multi-class classification. In this case, with the softmax function, it coincides with the multinomial logistic loss applied to the outputs of a neural network [1]. Cross Entropy Loss and Softmax are defined as follows:

Softmax Function: The softmax function transforms the outputs of the output layer (logits) into probabilities. It is defined as:

$$y_c = \text{softmax}(z_c) = \frac{e^{z_c}}{\sum_{k=1}^{C} e^{z_k}} \quad \text{for } c = 1, 2, \ldots, C$$

Multi-class Cross Entropy Loss:

$$E = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} [t_{cn} \ln y_{cn}]$$

- N: Number of samples in the dataset.
- C: Number of classes (3).
- t_{cn}: True label for class c of the n^{th} example, represented as a one-hot encoded vector.
- y_{cn}: Predicted probability of class c for the n^{th} example (computed by the softmax function).
- k: Index for the output neuron representing each class. ($k = 0, 1, 2$)

Cross-entropy loss quantifies the difference between the actual distribution and the models predicted distribution. Therefore, lower values indicate better model performance.

V. Hyperparameter Tuning

A. Learning Rate, Epochs and Batch Size

Emperically evaluated through grid search, mention k-fold cross validation. Add heat map for loss and accuracy

VI. Training Algorithms

Discuss

A. Training Algorithms - Adam

Adam (Adaptive Moment Estimation) is a modern training Implemented

B. Training Algorithms - RProp

Implemented

C. Training Algorithms - SGD

Implemented

D. Hybrid Training Algorithm

Implemented

VII. Research Results

A. Metrics

B. Results

C. Discussion

This is the section where you report your results obtained from running the experiments as discussed in the experimental set-up section. You have to give, at the very least, the averages and the standard deviations for all the experiments simulations. Graphing your results is advisable, and no conclusions regarding the superiority of one approach over another can be made without some form of statistical reasoning. Training and testing errors have to be reported. Thoroughly discuss the results that you have obtained, and reason about why you obtained the results that you have. Answer questions like “are these results to be expected?” and “why these results occurred?” and “would different circumstances lead to different results?”

VIII. Conclusions

Very general conclusions about the assignment that you have done. This section "answers" the questions and issues that you have raised and investigated. This section is, in general, a summary of what you have done, what the results were, and finally what you concluded from these results. This is the final section in your document, so be sure that all the issues raised up until now are answered here. This is also the perfect section to discuss what you have learnt in doing this assignment.

References

[1] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 23803–23828. PMLR, 23–29 Jul 2023.