Hybrid Learning in Neural Networks for Almond Classification

*COS711 Assignment 2

1st Reuben Jooste (u21457060)
Computer Science Dept.
University of Pretoria
Pretoria, South Africa
u21457060@tuks.co.za

***Abstract*—In this study, we investigate the utilisation of various optimisation algorithms within the context of a neural network aimed at addressing an almond classification challenge. Our study will include the following optimisation algorithms: Resilient Backpropagation (RProp), Stochastic Gradient Descent (SGD), and Adaptive Moment Estimation (Adam). Furthermore, we engage in a thorough comparative examination of the accuracy and loss (error) associated with each algorithm across the training and testing periods. This facilitates our assessment of how effectively each algorithm optimises the performance of the Neural Network (NN). In addition, we propose a hybrid learning methodology that involves the integration of the three algorithms through the averaging of their weight updates, with the intention of potentially improving the NN's generalisation capabilities. Lastly, we elucidate the strengths and limitations inherent to each algorithm by scrutinising the experimental results and identifying areas that warrant improvement to enhance classification accuracy. This investigation aims to contribute to the advancement of more robust learning strategies in NN optimisation by providing insights into practical implementations relevant to agricultural applications, such as almond classification.**

***Keywords*—Machine Learning, Neural Networks, Hybrid Learning, Optimisation Algorithms, Classification**

I. Introduction

In previous years, a lot of work has been done to assess the impact of utilising optimisation algorithms to help enhance Neural Network (NN) performance [1], [2]. More specifically, machine learning solutions and neural networks have increasingly been used to enhance crop quality and classification tasks such as Almond classification. With the help of NNs we can accurately classify Almonds based on a number of attributes/features which could lead to increased product quality and optimised supply-chain operations [3]–[5]. However, achieving high classification accuracy in a NN depends on the optimisation algorithm used during the training phase. Our study focuses on evaluating three optimisation algorithms – Stochastic Gradient Descent (SGD), Resilient Back Propagation (RProp), Adaptive Moment Estimation (Adam) – in the context of the almond classification problem.

A. Proposed Optimisation Algorithms

- *SGD:* This algorithm is widely adopted as it is regarded as a simple algorithm. However, due to the batch size being very small (one record per weight update), the algorithm is prone to slow convergence.
- *Adam:* Another variant of the gradient-based methods is the Adam optimiser. This variant uses adaptive and momentum learning rates which offers faster convergence allowing the model to find optimal results more quickly.
- *RProp:* This algorithm improves the learning speeds by adapting step sizes without the need of gradient magnitudes, meaning that the update direction for each weight is only controlled by the sign of the derivative of the performance function used [6].

Optimisation algorithms are a crucial component for a NN as it has a big influence on the NNs ability to minimise the loss and improve the accuracy. These algorithms have become more adopted over the past years as they increase the effectiveness and reliability of NN models to solve complex problems [1], [7]. Furthermore, NN architectures have become more complex and bigger hence optimisation algorithms have been proposed as well as hybrid learning approaches [8].

This study introduces a hybrid learning approach in which the three optimisation algorithms are integrated to enhance the NN performance. We integrate these optimisation algorithms by averaging the weight updates between the three algorithms and using it to update the NN's weights, thereby creating a unified learning mechanism. The motivation behind this hybrid learning approach is that we combine the unique strengths of each algorithm to potentially increase the model's accuracy and minimize the error [7].

The objectives of this study are two fold: we will first conduct a thorough comparative study of the three optimisation algorithms in terms of their training, validation, and testing performance. Secondly, we will assess the proposed hybrid learning approach and discuss whether it achieved an improved performance over any of the three single optimisation algorithms. This study aims to contribute to further development of NN models for classification tasks in the agricultural industry. In the following sections we present background information on the discussed research area, present our methodology and experimental setup, and finally compare and evaluate our algorithms based on the observed results.

II. Background Information

Over the years, the use of machine learning have become widely adopted due to revolutionising many fields, including the agricultural field. In this paper we address the problem with almond classification by employing NNs to identify whether an object is an almond or not based on several attributes. By using NNs we try to enhance the classification accuracy as it can directly influence factors such as the product quality control and supply-chain operations [3]–[5]. However, careful planning must be done to ensure that an appropriate optimisation algorithm is used to update the model weights during training since the success of a NN hinges on the optimisation algorithm.

NNs are simply mathematical models where the architecture is designed in such a way to mimic the learning processes of the human brain. The network consists of interconnected layers where each layer is made up of a number of neurons where each neuron is connected to another layer's neuron through a weight. The goal of a NN is to pass the input information from one layer to the next in order to make a prediction. The NN uses an optimization algorithm to iteratively update the weights of the network to minimise the loss between the actual label and the predicted label. Figure 1 shows an illustration of a simple NN architecture. As mentioned earlier, the NN goes through two main phases: training and testing. During the training phase we train the model for a number of iterations on a large subset of the original dataset, called the training set. These iterations are referred to as epochs and during each epoch the algorithm (optimiser) evaluates the performance of the model using the entire training dataset. With each epoch the algorithm also calculates the weight updates and updates the weights of the neural network either after every single record (SGD) or after a batch of records (RProp and Adam). Single batches are used by the SGD optimising algorithm since we want to train the NN faster however, due to only using a single record per weight update the algorithm will converge very slowly. The other two algorithms use what is called mini-batches meaning that a certain number of records are first evaluated before we update the weights. In other words, we accumulate the gradients for each record evaluated and then use the sum of the gradients of the batch to calculate the weight updates. By using mini-batches the algorithm can reach optimum minima which the single batch cannot reach [9].

For this study, we utilise three widely adopted optimisation algorithms: Stochastic Gradient Descent (SGD), Adaptive Moment Estimation, and Resilient Backpropagation (RProp). SGD is one of the simplest optimisation algorithms which uses a batch size of 1, meaning it updates the weights after each record was propagated forward through the NN. This obviously makes the training noisy and fast but the algorithm suffers from slow convergence, meaning that SGD is more unstable in local areas [10].

To address the issues with SGD, the Adam optimiser algorithm was introduced. Adam achieves a smoother gradient as it achieves a lighter gradient tail than SGD [10]. One advantage

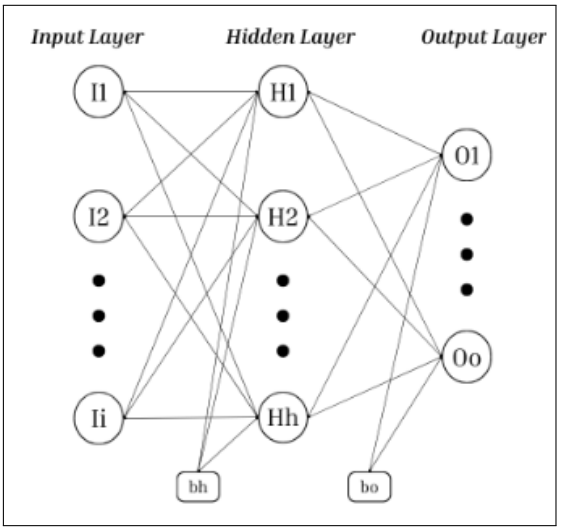

Fig. 1: Example of a simple Neural Network architecture

of the Adam algorithm is that it "wakes up" dead neurons that are affected by activation saturation in the network allowing the NN model to achieve greater generalisation capabilities [11].

On the other hand, the RProp algorithm optimises the performance of normal backpropagation by using the local gradient information (obtained from learning iterations) to update the weights and biases of the NN model. This means that when updating the direction of each weight, the direction is only affected by the sign of the derivative of the performance function used and not affected by the gradient of the magnitudes [6], [12], [13].

While each algorithm has its own strengths, no single approach is universally optimal. Therefore, we propose a hybrid learning technique which integrates these three algorithms thereby creating a new algorithm which combines the unique strengths of the three optimisers. By using this hybrid algorithm, we are able to address the issues associated with complex NN architectures and enhance the model's classification accuracy and ability to minimize the error/loss [7], [8]. The hybrid learning approach works by averaging the weight updates of all three algorithms and using it to update the weights of the NN.

III. Experimental Setup

In this section we cover details explaining the dataset that was used for the almond classification task, the architecture of the neural network, the training parameters used, the two hyperparameters and their possible values, and finally the specific experimental conditions under which the four algorithms were tested. Additionally, a *Jupyter Notebook*[1] was used to implement our research using the **Python** programming language. We used a **Global Seed value of 99** to allow for replicating our results.

[1]The Jupyter Notebook can be found on this GitHub repo: https://github.com/JsteReubsSoftware/COS711-Assignment-2

A. Dataset Preparation

In this research we used the Almond Classification Dataset available on **Kaggle**[2]. This dataset consists of 14 features and 2803 unique records. The target class variable includes three different categories: SANORA, MAMRA, and REGULAR. Based on the other 13 attributes our NN model will classify the almond according to one of the three almond types.

However, as with any dataset we first had to preprocess the dataset to ensure that it was free of missing values, outliers, and duplicates. We first plotted the descriptive statistics (mean, median, mode, std.) for all features in the dataset. Additionally, we plotted Box-plots and histograms to visualise the distribution of each feature. This allowed us to understand the dataset's structure and content better as well as identify whether the dataset contained any missing values, outliers, and/or duplicate records. Figure 2 shows a visualisation of the distribution for each feature.

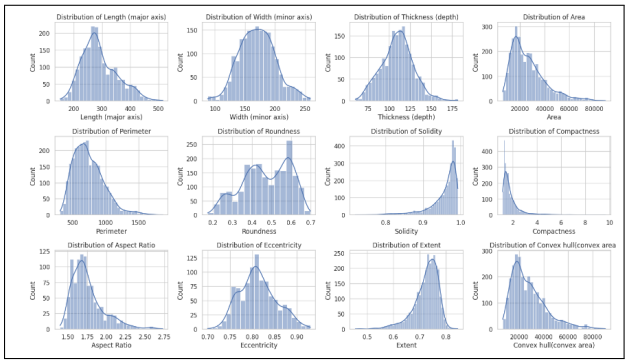

Fig. 2: Feature Distributions before cleaning

1) Handling Outliers: From the visualisations we plotted, we identified some outliers among a few features. We first handled these outliers before handling the missing values since outliers will have an negative effect when we e.g. replace a missing value with the mean value of the respective feature. We decided to replace all the outliers with the median value of the respective feature due to the median being less sensitive to outliers compared to the mean. Furthermore, when replacing outliers we should take note of the distribution of the respective feature since the Z-Score is sensitive to data with a skewed distribution. Fortunately, from the visualisations we could easily identify those features with skewed distributions and those having a normal distribution. We then used the Z-Score to extract the outlier data (those with z-score ¿ 3) from the normally distributed features, respectively and used the interquartile range (IQR) to extract outliers (those outside the IQR range) for the features with skewed distributions. Finally, we replaced all the outlier data with the median value for each respective feature that have outliers.

2) Handling Missing Values: Now that our data was cleaned from any outliers we could move forward to replacing all the missing values. Replacing missing values is a critical step during data preprocessing as missing values can cause biased results or lead to misinformed analysis [14]. For our implementation we decided to replace the missing values using a K-Nearest Neighbours (KNN) approach. This method works by replacing the missing value using the k-nearest data points. The KNN algorithm uses all the other available features' values to make a prediction for what the value of the missing data value should be i.e. it regards the feature with the missing value as the target variable. This of course would only work effectively if there are only one single missing value per record [15].

Based on our understanding of the dataset from the descriptive statistics and what each feature meant, we found that some missing values are only mark as NA due to it being dependent on another feature with a missing value. For example the **Aspect Ratio** feature is dependent on both the **Length** and **Width** features. Therefore, if either Length or Width have a missing value, Aspect Ratio would also have a missing value. We therefore, first replaced all of the missing values for the features **Length**, **Width**, **Thickness** using the KNN imputer. Once these *dependent* features had values we replaced the other features using the respective feature's formula e.g.

$$\text{Aspect Ratio} = \text{Length}/\text{Width}$$

However, if the majority of a record's features included missing values, we simply remove it as it would be of no value to the model during the training phase. By following this approach we cleaned the dataset from all missing values. This also ensured that we eliminated any bias in our data that could affect our model's performance during the training phase.

3) Label Encoding: For our classification problem we are required to encode our target class variable to numerical values. We could have used any method such as One-hot encoding or Label encoding but we opted to use **Label Encoding** as our **Cross Entropy loss function** expects integer labels (e.g. 0, 1, 2) for the three categories of our dataset. We therefore, the categories as follows:

- *MAMRA* is encoded as **0**
- *REGULAR* is encoded as **1**
- *SANORA* is encoded as **2**

Now that all of the dataset's features are in a numerical representation, we can apply standardisation techniques.

4) Standardisation / Normalisation: Once our data was free of any errors and all categorical features encoded to a numerical format, we applied the well-known **Min-Max scaling** technique to convert all the data points to a similar range. It is known that applying the Min-Max scaling technique can have a great affect on the model's performance [16]. By standardising all features to a similar range we ensure that our NN model will treat all features equally and that no single feature will be treated as being more important than another i.e. having more influence than another. By doing this we ensure that our NN model does not create a bias towards a single feature.

[2]The Almond Classification dataset can be found at: https://www.kaggle.com/datasets/sohaibmoradi/almond-types-classification

B. Hyperparameters

The main goal of our research is to find an optimal NN model for all three of our optimisation algorithms. We chose two specific parameters to fine-tune: **number of hidden layers** and **learning rate**. The other parameters used by our implementation include the following:

- *Number of input neurons:* This is defined by the number of input features used to train our NN model.
- *Number of output neurons:* This is defined by the number of output classes. Due to our classification problem having three different Almond types, our NN has three output neurons on the Output Layer.
- *Batch Size:* SGD always uses a batch size of 1 and we decided to use a batch size of 32 for the RProp and Adam optimisers as it is recommended as the default value for achieving high performance results in previous work [9], [17].
- *Number of epochs:* We simply used 200 epochs as this is a large enough value for all three algorithms to converge. Using a smaller value might result in stopping the algorithm too early before it reached the optimal result.
- *Activation Function:* We opted to only use the well-known activation function **ReLU**. This function is more effective than others as certain neurons are activated at a time rather than activating all neurons at the same time [18].
- *Number of Independent Runs:* We opted to only use 5 independent runs.

We opted to experiment with the following values for our chosen two hyperparameters:

- *Number of Hidden Layers*: [64, 32], [128, 64, 32], [256, 128, 64, 32]
- *Learning Rate*: 0.001, 0.01, 0.015

The values for the number of hidden layers represent a vector of layer sizes. Meaning that the length of the vector indicates the number of hidden layers to use while the numerical values indicate the number of hidden neurons to use on the respective layer. Furthermore, we will experiment with small values for the learning rate hyperparameter since using large learning rates will cause the algorithm to jump around rather than slowly move towards the optimal result.

To find the most optimal combination for hyperparameters, we will use the well-known technique of a **grid search**. Furthermore, we will visualise this using a heatmap to help indicate the most effective combination in terms of accuracy performance. Based on our grid search results and hypothesis testing, we can evaluate and make conclusions on which hyperparameter values are optimal.

C. Neural Network Architecture

For each of the optimisation algorithms we trained a separate Feed-forward Neural Network model for our Almond classification task. Each NN model was made up using the following layers:

- *Input Layer:* This layer consisted of 13 input neurons since our dataset comprised of 13 variables that were not regarded as the target class variable.
- *Hidden Layers:* Due to the number of hidden layers being one of our chosen hyperparameters, the NN could potentially have the following structure for its hidden layers:
  - ⋆ **Two hidden layers** made up of 64 and 32 hidden neurons, respectively.
  - ⋆ **Three hidden layers** made up of 128, 64, and 32 hidden neurons, respectively.
  - ⋆ **Four hidden layers** made up of 256, 128, 64, and 32 hidden neurons, respectively.
- *Output Layer:* The final layer of our NN model was made up of only **three output neurons** since the classification problem only has three distinct categories for the Almond type.

Our NN architecture also have an activation function on each layer, excluding the input layer. For every hidden layer we used the **Rectified Linear Unit (ReLU)** activation function while using the widely used **Softmax** activation function on the output layer.

$$\text{ReLU}(x) = max(0, x)$$

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

The ReLU function takes in the output value from a hidden neuron and either returns the same value of if it is negative it returns the value 0. On the other hand, the Softmax function normalises all the input values (i.e. all output values from the output neurons) to a range between 0 and 1. This then makes it easier for classification problems as the predictions for each class (or category) are represented as probabilities, allowing us to identify the prediction made by the NN model by simply taking the class with the highest predicted probability.

D. Optimisation Algorithms

To allow for a fair performance comparison between the three optimisation algorithms as well as the hybrid learning algorithm, each algorithm was tested under the same experimental conditions. This means that for each algorithm we used the same experimental values for the hyperparameters and kept the other parameter values fixed. The SGD obviously had to use a batch size of 1 while the other algorithms used a batch size of 32.

During training all the optimisation algorithms, we apply the well-known **SMOTE** technique which is used to balanced the number of samples for each class. This ensures that our model will not assign a bias toward a majority category during the training process [19]. Additionally, each training process uses the **K-Fold Cross Validation** technique to perform the hyperparameter optimisation. We decided to use five fold (**k=5**) since it has been proven to be an effective number of folds to use [20]. Each model was also trained on **200 epochs** and we also used a similar **stopping criteria** for each

algorithm: if for 10 consecutive epochs the model did not increase the accuracy with at least 1% then the algorithm halts. All the algorithms also used the exact same **Cross Entropy loss function** for calculating the error of the NN model. We also used **Accuracy** as the performance metric for all optimisation algorithms to measure the NN model's performance on both the training and testing sets using a **80-20** split for these sets, respectively.

In terms of how we **implemented our Hybrid Learning algorithm**, we simply used all three of the other optimisation algorithms to build a universal NN model. The hybrid algorithm works by separately doing forward and backward propagation on each of the three algorithms' NN models. We then accumulate each model's gradients for a single epoch. After one epoch, the accumulated gradients are averaged such that we can use the average gradient to update one of the three algorithms' NN model's weights. The training process is then repeated for a number of epochs until it converges or reaches the stopping criterion. We also did a grid search to optimise the hyperparameters for Hybrid Learning NN model. After the training phase, we use the trained hybrid NN model to make predictions on unseen data (i.e. the testing set) and evaluate its performance.

Finally, we used **five independent runs** to test different seed values for every combination of the two hyperparameters for every optimisation algorithm (including the Hybrid Learning approach).

IV. Research Results

We first ran the four optimisation algorithms (SGD, Adam, RProp, Hybrid) to evaluate each algorithm's performance for solving our almond classification task. We then obtained a grid of average results across the number of independent runs and used a heatmap to highlight the performance of each combination of hyperparameter values. Figures 3, 4, 5, and 6 shows the heatmaps for all four optimisation algorithms.

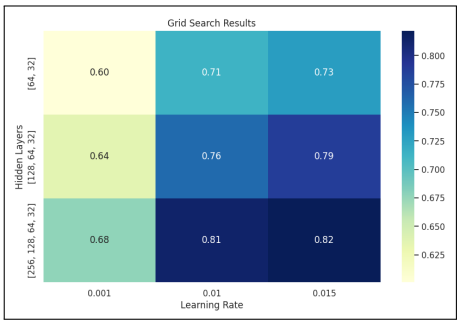

Fig. 3: SGD Grid Search

From our grid search results we note that both there is a common factor between all the optimisation algorithms due to all of the plots having the highest *heat colour* on the lowest level/row. The SGD achieved an accuracy of **82%**, the Adam and the Hybrid algorithm achieved an accuracy of **85%**, and the RProp achieved a low accuracy of **62%** This then indicates that regardless of what the learning rate is, using a higher

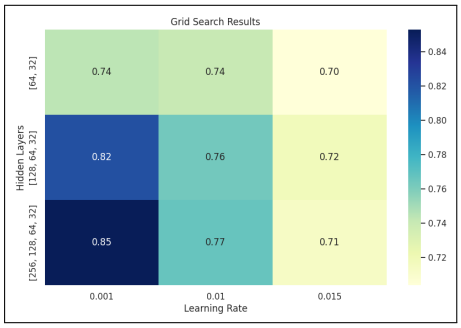

Fig. 4: Adam Grid Search

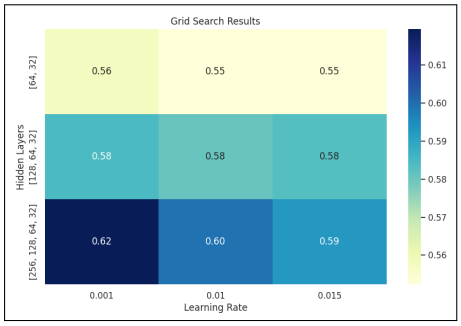

Fig. 5: RProp Grid Search

number of hidden layers will automatically increase the NN model's performance. This is because when using a larger number of hidden layers, the NN has the ability to learn more complex and abstract features from the input data, allowing it to capture more complex relationships or patterns between certain features.

We further note that both the Adam optimiser and the RProp optimiser appear to be more sensitive towards smaller learning rates while the SGD optimiser appears to perform better when the learning rate is higher. This could potentially be due to the batch size that was used during the training phase since the Adam and RProp optimisers used a different batch size than the SGD optimiser. When using a larger batch size, the model is able to prove a more accurate estimate of the gradients which leads to more consistent and smaller updates. Therefore, when using a larger learning rate the model is more likely to

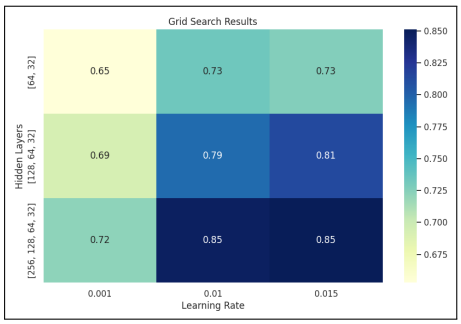

Fig. 6: Hybrid Grid Search

overshoot optimal results. In contrast, when using a smaller batch size there appears to be more noise in the gradient. Therefore, when using a larger learning rate the model is able to explore the loss regions more effectively and thus performs better when using the larger learning rate. Due to these differences between the first three optimisation algorithms, we opted to implement a Hybrid Learning approach. From the grid search results we note that the Hybrid Learning algorithm performed better than any of the other three algorithms for a learning rate of either 0.01 or 0.015. We also note that the heatmap for the Hybrid algorithm is darker when using a larger number of hidden layers which is expected since we used the other three algorithms to implement our Hybrid algorithm.

We also conducted simple **hypothesis tests** by using **ttests** to determine whether one hyperparameter value is more performant than another. We found that the learning values of 0.01 and 0.015 is more performant (i.e. there is a statistically significant difference) than the value of 0.001 for the SGD algorithm since the **p-value** $<$ **0.05**. However, for the Adam, RProp, and Hybrid algorithms we found that there is no significant difference between the learning rate values but there is a statistically significant difference between using the hidden layers [64, 32] and [256, 128, 64, 32].

We did a further comparison between the algorithms by comparing their training performance over a number of epochs. As mentioned earlier, we use a K-Fold Cross validation approach meaning that for each fold there would be a vector of performance scores for a number of epochs. Therefore, we had to take the average across all folds to create a new linear vector for each algorithm. This then allowed us to plot line charts for both the accuracies and losses of each algorithm over the 200 epochs. Figures 7 and 8 shows a visual comparison between all four algorithms in terms of their training accuracy and loss performance, respectively.

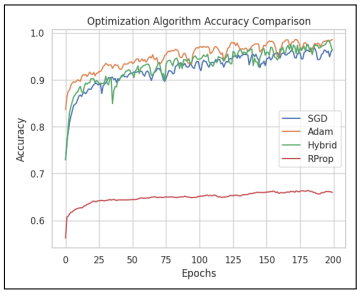

Fig. 7: Training Accuracy performance over 200 epochs

From Figure 7 we note that both the Adam and SGD optimising algorithms performed much better than the RProp algorithm. This indicates that the RProp algorithm failed to accurately predict the type of an almond. Furthermore, the NN model that used the RProp algorithm as its optimiser failed to learn the complex patterns and relationships of the different features in the dataset, hence achieving low accuracy and high loss scores. We also note that while the Adam and SGD algorithms took close to 200 epochs to converge, the RProp algorithm appears to have converged much earlier

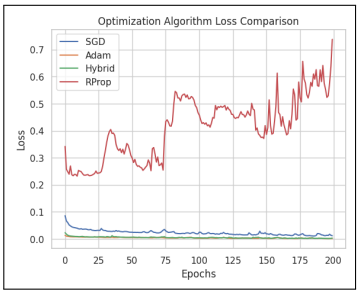

Fig. 8: Training Loss performance over 200 epochs

(around 25 epochs) which further highlights that the RProp algorithm achieved a bad performance. In terms of the best optimiser, it appears that the Adam optimiser achieved the best performance (highest training accuracy). From previous work we expected the Adam optimiser to achieve a better performance than the SGD algorithm since the Adam optimiser if often regarded as an advancement over the SGD algorithm [10], [21]–[23]. To do a more accurate comparison between the Hybrid algorithm and the others, we decided to plot the same line chart, this time excluding the RProp algorithm's performance. Figures 9 and 10 shows a zoomed-in graph of Figures 7 and 8.

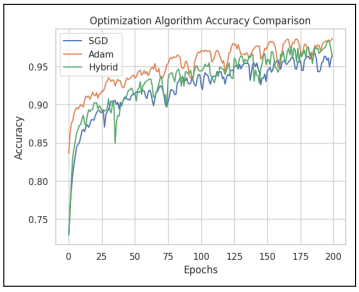

Fig. 9: Training Accuracy performance over 200 epochs

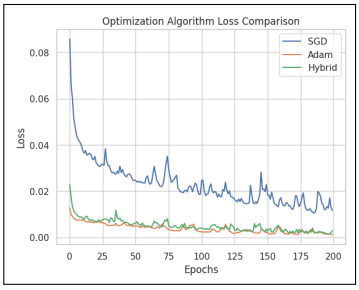

Fig. 10: Training Loss performance over 200 epochs

We observe that the Adam optimiser and Hybrid Learning algorithm achieved very similar final performances in terms of both accuracy and loss even though the Adam optimiser appears to start off better than the Hybrid Learning algorithm. This is somewhat expected as our Hybrid learning algorithm used the Adam optimiser as its base algorithm and as the training process progresses the other two algorithms' (SGD and RProp) updates are integrated to achieve the hybrid learn-

ing approach. To further compare the algorithms, we report on the means and standard deviations across the independent runs for each combination of hyperparameters for each algorithm. The independent runs each used different seed values:

- Run 1 – Seed value is 7204
- Run 2 – Seed value is 3241
- Run 3 – Seed value is 9930
- Run 4 – Seed value is 1769
- Run 5 – Seed value is 7891

Hidden Layers, LR	SGD	Adam	RProp	Hybrid
[64, 32], 0.001	60.09	74.37	55.77	65.29
[64, 32], 0.01	71.30	74.04	55.23	73.35
[64, 32], 0.015	73.02	70.37	55.27	72.57
[128, 64, 32], 0.001	63.57	82.16	58.44	69.29
[128, 64, 32], 0.01	75.97	76.32	58.09	79.47
[128, 64, 32], 0.015	78.71	71.80	58.32	81.38
[256, 128, 64, 32], 0.001	67.69	85.29	61.95	72.24
[256, 128, 64, 32], 0.01	81.37	76.72	59.95	84.73
[256, 128, 64, 32], 0.015	82.16	71.32	59.27	85.11

TABLE I: Mean Accuracy for Optimisers

Hidden Layers, LR	SGD	Adam	RProp	Hybrid
[64, 32], 0.001	0.0240	0.0331	0.0216	0.0308
[64, 32], 0.01	0.0268	0.0304	0.0127	0.0407
[64, 32], 0.015	0.0340	0.0225	0.0235	0.0494
[128, 64, 32], 0.001	0.0184	0.0551	0.0139	0.0310
[128, 64, 32], 0.01	0.0493	0.0397	0.0179	0.0683
[128, 64, 32], 0.015	0.0627	0.0373	0.0161	0.0760
[256, 128, 64, 32], 0.001	0.0308	0.0663	0.0185	0.0532
[256, 128, 64, 32], 0.01	0.0877	0.0511	0.0180	0.0969
[256, 128, 64, 32], 0.015	0.1003	0.0312	0.0237	0.0946

TABLE II: Standard Deviation Accuracies for Optimisers

Hidden Layers, LR	SGD	Adam	RProp	Hybrid
[64, 32], 0.001	0.8403	0.6106	0.9930	0.7589
[64, 32], 0.01	0.6703	0.8444	0.9229	0.7629
[64, 32], 0.015	0.6845	0.8077	0.9215	0.8798
[128, 64, 32], 0.001	0.7804	0.6413	2.6302	0.6852
[128, 64, 32], 0.01	0.7681	0.9207	1.0007	0.7719
[128, 64, 32], 0.015	0.7359	0.8612	0.9282	0.7096
[256, 128, 64, 32], 0.001	0.7123	0.6241	64.9347	0.6678
[256, 128, 64, 32], 0.01	0.6867	0.9622	2.3685	0.5907
[256, 128, 64, 32], 0.015	0.7300	0.8402	172.0483	0.6062

TABLE III: Mean Loss for Optimisers

Hidden Layers, LR	SGD	Adam	RProp	Hybrid
[64, 32], 0.001	0.0359	0.0784	0.1021	0.0369
[64, 32], 0.01	0.0393	0.1072	0.0295	0.0752
[64, 32], 0.015	0.0576	0.0720	0.0343	0.1926
[128, 64, 32], 0.001	0.0318	0.1760	1.2516	0.0476
[128, 64, 32], 0.01	0.1238	0.1737	0.1449	0.2413
[128, 64, 32], 0.015	0.2096	0.1448	0.0929	0.3109
[256, 128, 64, 32], 0.001	0.0533	0.3662	85.1919	0.0788
[256, 128, 64, 32], 0.01	0.3236	0.1927	2.2109	0.4007
[256, 128, 64, 32], 0.015	0.4604	0.0902	333.9557	0.4210

TABLE IV: Standard Deviation Losses for Optimisers

Table I presents the results of the mean accuracy scores for each optimiser across the five independent runs. We can easily analise that the RProp algorithm performed poorly for all hyperparameter combinations which is likely due to it not being designed to handle NNs with large architectures as well as the fact that it only considers the sign of the gradient while ignoring the magnitude [6], [12], [13]. On the other hand, the Adam optimiser algorithm performed the best across all hyperparameter combinations and we note that using a larger NN architecture and smaller learning rate helped optimise the NN model. As mentioned earlier, we expected that this algorithm would perform the best between the three single algorithms since this algorithm was designed to handle complex NN architectures and has the ability to wake-up saturated neurons [11], [23]. When analysing the standard deviations of the accuracies (Table II) we note that all the algorithms show a **consistent performance** across all hyperparameter combinations.

Despite the fact that three of the four algorithms showed a good enough performance in terms of there mean accuracy, all the algorithms did not perform that well in terms of the mean loss across the different hyperparameter combinations as indicated by Table III. This indicates that although each algorithm made a high number of correct predictions, it diverged from the optimal results. This is likely due to our implementation not having used any regularisation techniques.

However, our goal was to develop a more optimised, hybrid algorithm that would outperform all three algorithms but from our results we analise that our Hybrid implementation did not satisfy our expectations. This should however have been expected since all three algorithms use different weight-update strategies which (when combined) could introduce conflicting updates. For example, the SGD applies a global learning rate across all weights, the Adam optimiser adjusts the learning rate adaptively, and the RProp algorithm uses only the sign of the gradient to update the learning rates. Additionally, averaging different weight update directions could also be a compromise that leads to slower convergence.

We also tabulated each algorithms testing results to evaluate the generalisation abilities of each algorithm. From Table V we note that the RProp algorithm performed the worst on unseen data (as expected after analysing its training performance) indicating that it failed to generalise well on new data. On the other hand, the Adam optimiser performed the best out of all four algorithms with the SGD and Hybrid algorithms falling just short of its performance.

Algorithm	Accuracy (%)	Loss
SGD	81.9964	1.2924
Adam	81.2834	1.2701
RProp	60.4278	142.6118
Hybrid	80.9269	1.4610

TABLE V: Algorithm testing performance comparison

Despite this, the best model for all three of these algorithms achieved above 80% which indicates to us that these algorithms performed relatively well during the testing phase. However, when comparing the testing accuracies with the training accuracies of the best models, we observe that there is a big difference between the results as the training accuracies

for these three algorithms were close to or more than 95% (see Figure 9). This highlights the fact that our models suffered from the well-known problem called **overfitting** since they performed much better on seen data compared to unseen data. This problem is likely due to the algorithms over-optmising on the training data causing the NN model to fit very small, noisy patterns. To counter these problems we could consider applying regularisation techniques to help prevent the NN model from learning specific patterns in the seen data, thereby reducing the chances of overfitting.

V. Conclusion

In this paper we conducted a study to evaluate three different optimisation algorithms: SGD, Adam, and Rprop. Additionally, we implemented a Hybird Learning approach which combined the strenghts of all three algorithms to try and build a more optimiserd NN model for our Almond classification problem. Furthermore, each of the four algorithms were fine-tuned using a grid search to help find the most optimal values for our two chosen hyperparameters: number of hidden layers, and learning rate. Based on our results, we found that the Adam optimiser performed the best across all hyperparameter combinations over a number of independent runs. We found that the Hybrid learning approach did not perform as expected due to the three algorithms having conflicting weight-update strategies which caused the hybrid NN model not to outperform any of the other algorithms.

Finally, to build on this research one can consider choosing a set of optimisation algorithms. The algorithms should use a similar weight-update strategy such that when implementing a hybrid learning approach it would not be affected by conflicting weight-update strategies. Furthermore, consider implenting additional regularisation techniques to prevent the models from overfitting.

References

[1] M. G. M. Abdolrasol, S. M. S. Hussain, T. S. Ustun, M. R. Sarker, M. A. Hannan, R. Mohamed, J. A. Ali, S. Mekhilef, and A. Milad, "Artificial neural networks based optimization techniques: A review," *Electronics*, vol. 10, no. 21, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/21/2689

[2] Z. Li, T. Lin, X. Shang, and C. Wu, "Revisiting weighted aggregation in federated learning with neural networks," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 19 767–19 788. [Online]. Available: https://proceedings.mlr.press/v202/li23s.html

[3] A. Priyangka and I. Kumara, "Classification of rice plant diseases using the convolutional neural network method," *Lontar Komputer: Jurnal Ilmiah Teknologi Informasi*, vol. 12, no. 2, p. 123, 2021.

[4] F. Saeed, M. A. Khan, M. Sharif, M. Mittal, L. M. Goyal, and S. Roy, "Deep neural network features fusion and selection based on pls regression with an application for crops diseases classification," *Applied Soft Computing*, vol. 103, p. 107164, 2021.

[5] Y. Gulzar, Y. Hamid, A. B. Soomro, A. A. Alwan, and L. Journaux, "A convolution neural network-based seed classification system," *Symmetry*, vol. 12, no. 12, p. 2018, 2020.

[6] H. Zhu, J. Leandro, and Q. Lin, "Optimization of artificial neural network (ann) for maximum flood inundation forecasts," *Water*, vol. 13, no. 16, 2021. [Online]. Available: https://www.mdpi.com/2073-4441/13/16/2252

[7] M. Thirunavukkarasu, Y. Sawle, and H. Lala, "A comprehensive review on optimization of hybrid renewable energy systems using various optimization techniques," *Renewable and Sustainable Energy Reviews*, vol. 176, p. 113192, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032123000485

[8] D. Soydaner, "A comparison of optimization algorithms for deep learning," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 34, no. 13, p. 2052013, 2020. [Online]. Available: https://doi.org/10.1142/S0218001420520138

[9] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959519303455

[10] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, and W. E, "Towards theoretically understanding why sgd generalizes better than adam in deep learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 21 285–21 296. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/f3f27a324736617f20abbf2ffd806f6d-Paper.pdf

[11] Z. Liu, Z. Shen, S. Li, K. Helwegen, D. Huang, and K.-T. Cheng, "How do adam and training strategies help bnns optimization," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 6936–6946. [Online]. Available: https://proceedings.mlr.press/v139/liu21t.html

[12] W. Saputra, A. P. Windarto, and A. Wanto, "Analysis of the resilient method in training and accuracy in the backpropagation method," *International Journal of Informatics and Computer Science (IJICS)*, vol. 5, no. 1, pp. 11–15, 2021.

[13] L. M. Saini, "Peak load forecasting using bayesian regularization, resilient and adaptive backpropagation learning based artificial neural networks," *Electric Power Systems Research*, vol. 78, no. 7, pp. 1302–1310, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378779607002258

[14] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona, "A survey on missing data in machine learning," *Journal of Big Data*, vol. 8, no. 1, p. 140, Oct 2021. [Online]. Available: https://doi.org/10.1186/s40537-021-00516-9

[15] A. F. Sallaby and A. Azlan, "Analysis of missing value imputation application with k-nearest neighbor (k-nn) algorithm in dataset," *International Journal of Informatics and Computer Science (IJICS)*, vol. 5, no. 2, pp. 123–130, 2021.

[16] A. Ambarwari, Q. Jafar Adrian, and Y. Herdiyeni, "Analysis of the effect of data scaling on the performance of the machine learning algorithm for plant identification," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 4, no. 1, pp. 117 – 122, Feb. 2020. [Online]. Available: http://jurnal.iaii.or.id/index.php/RESTI/article/view/1517

[17] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade: Second edition*. Springer, 2012, pp. 437–478.

[18] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci*, vol. 6, no. 12, pp. 310–316, 2017.

[19] G. A. Pradipta, R. Wardoyo, A. Musdholifah, I. N. H. Sanjaya, and M. Ismail, "Smote for handling imbalanced data problem: A review," in *2021 sixth international conference on informatics and computing (ICIC)*. IEEE, 2021, pp. 1–8.

[20] B. G. Marcot and A. M. Hanea, "What is an optimal value of k in k-fold cross-validation in discrete bayesian network analysis?" *Computational Statistics*, vol. 36, no. 3, pp. 2009–2031, Sep 2021. [Online]. Available: https://doi.org/10.1007/s00180-020-00999-9

[21] A. Gupta, R. Ramanath, J. Shi, and S. S. Keerthi, "Adam vs. sgd: Closing the generalization gap on image classification," in *OPT2021: 13th Annual Workshop on Optimization for Machine Learning*, 2021.

[22] F. Kunstner, J. Chen, J. W. Lavington, and M. Schmidt, "Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be," 2023. [Online]. Available: https://arxiv.org/abs/2304.13960

[23] Y. Pan and Y. Li, "Toward understanding why adam converges faster than sgd for transformers," 2023. [Online]. Available: https://arxiv.org/abs/2306.00204