Large Scale Optimisation with Adaptive Strategy Particle Swarms

Computational Intelligence COM3013

Ali Shihab
Department of Computer Science
University of Surrey
as02795@surrey.ac.uk
6564898

***Abstract*—The training and development of Artificial Neural Networks (ANNs) is contingent on techniques in optimisation to minimise residual error. While metaheuristic optimisation is well-researched, gradient-based optimisation continues to dominate the space due to its superior computational efficiency and ability to find sufficiently optimum solutions. Despite this, gradient-based techniques maintain a vulnerability to premature convergence to local optima. Further, there is a necessity for robust metaheuristic techniques in contexts where the solution cannot be found analytically. In this paper, we compare canonical state-of-the-art gradient-based optimisation to a metaheuristic evolutionary algorithm in the optimisation of a Convolutional Neural Network (CNN) to classify a dataset of images. Further, we propose a novel adaptive-memetic swarm optimisation algorithm to avoid premature convergence to local optima. The proposed algorithm shows super performance to both of the alternatives, both in convergence speed and ability to escape local optima. The algorithm shows strong potential for viability with further computational improvements and parallelism.**

***Index Terms*—Computational Intelligence, Machine Learning, Deep Learning, Metaheuristic Optimisation, Memetic Algorithms, Baldwinian Learning, Lamarckian Learning, Gradient Descent, Evolutionary Algorithms, Particle Swarm**

I. Introduction

One of the most notable research developments is that of the early model of a neuron, termed the M-P perceptron [1] in 1943. In this architecture, a rudimentary system analogous to the neurological composition and function of a neuron in a brain was proposed. It roughly modelled synaptic processes and nervous activity using logical calculus. Later, the ability of multi-layered feed-forward networks [2] (misnomered "Multi-layer Perceptron", "MLP") to identify non-linearly separable patterns was identified in 1958. The key development in the MLP was the development of a "hidden layer", which consisted of perceptrons to process the input after the input layer and prior to the output layer, each with non-linear activation functions. This set the foundation for the developments that followed in Deep Learning (DL) and the efforts to develop systems for specific task-execution and function approximation, inspired by natural intelligence. One such problem-area is that which falls under the umbrella term "Computer Vision (CV)". One of the most notable focuses of CV is in image classification and pattern recognition, which aims to identify the class to which an image belongs, from a set of predetermined classes (termed "labels").

When processing image data, despite the MLP's ability to generalise well to new input data and to learn non-linear relationships, it lacked translation invariance and equivariance. In these weaknesses, the Convolutional Neural Network (CNN) - first introduced in its foundational form in 1989 [3] - successfully improved. Translation invariance, in the context of image processing, is the ability to produce the same output despite the linear translations (horizontal translations, lighting, etc.) applied to the input pixels of the image. This is achieved by the pooling layers, which substitute feature maps produced by preceding convolutional layers with representative statistics, thus disregarding the location of features. Translation equivariance refers to the ability to produce an output equal of the translation to the input. In a CNN, this is achieved by the convolutional layers, which produce feature maps translated by the same magnitude in the same direction as the translation to the input. Despite this, CNNs do not naturally maintain invariance and equivariance to rotation, scaling and affine transforms. In these cases, data augmentation techniques are required to make the CNN robust to these transformations.

The overarching contribution of CNNs was the improvement of ANNs at handling image recognition tasks by considering spatial hierarchies through backpropagation. The introduction of CNNs marked a pivotal shift in neural network research and application, showcasing how layer-wise, locally connected structures in a network could lead to more efficient and effective learning. In subsequent works, CNNs, through their translation invariance and equivariance, showed superiority in handwritten digit recognition [4], document recognition [5] and were even shown to be applicable with temporally structured data such as speech signals and time series [6]. With the advent of new technology with increased efficiency and processing power, CNNs and DL, in general, have drastically developed in architecture and increased in complexity and computational demand. LeNet-5 [5] introduced a standard template for modern CNNs, consisting of 2 convolutional and 3 fully connected layers, with a "sub-sampling" (pooling)

layer, for a total of 60,000 parameters. AlexNet [7], in 2012, increased the scale to 60 million parameters and introduced the use of ReLu activation, marking a breakthrough in image classification on ImageNet. Later models like VGG-16 [8] demonstrated the efficiency of deeper CNN architectures with 138 million parameters and ResNet-50 [9] then introduced skip connections to address accuracy degradation in deep networks, allowing for even deeper CNNs without compromising performance. Other models like Xception [10] (based on its predecessor, Inception-v4 [11]) solely used depth-wise separable convolutional layers, while Inception-ResNets [11] combined the architecture of Inception-v4 and ResNet-50 to scale up cardinality within a module to learn more complex features and representations from the input data, improving its ability to recognize patterns.

On the topic of optimisation, [12] set the foundations for optimising ANN architecture by proposing the method involving a recurrent neural network (RNN) that is trained with reinforcement learning to generate the architecture of a neural network. Other techniques for optimisation of ANNs in the macro-sense involve a computationally intensive "Grid Search", where a "Grid" (limited subset) of hyperparameter values is iterated through as inputs for training a model, of which the best performing model is selected. In the micro-sense, metaheuristic optimisation consists of many approaches to problems with no analytical solution, or where such solution is too computationally expensive to obtain. One such approach is through framing of the solutions in a population-based manner, as in Evolutionary Algorithms (EAs) [13]. EAs are stochastic direct search umbrella-field under population-based metaheuristics that mimic different aspects of natural intelligence. Notably, Genetic Algorithms (GAs) [14] seek to mimic natural intelligence modelled by Charles Darwin's theory of evolution. Individuals are encoded as binary-valued or real-valued, with "chromosomes", each consisting of "genes" representing its "genotype" which determine the individual's "phenotype". The general school of thought maintains the following in its approach:

1) Causality requires that changes in genotype result in proportionate changes in phenotype;
2) Evolvability requires that heritable new phenotypes can be produced to be adaptable to environmental changes;
3) Robustness requires that critical phenotypes should be insensitive to minor changes in the genotype.

GA operators consist of mutation and recombination to produce new candidate solutions, termed "offspring". Selection occurs according to "fitness functions" under conditions that impose "selection pressure". The process is often repeated for iterations, known as "generations". Other population-based metaheuristic EAs, like Particle Swarm Optimisation (PSO) [15], model candidate solutions as "particles" moving around a search or decision space. Inspired by swarm-like organisms of nature, such as a school of fish or an ant colony, production of new candidate solutions consists of deterministic or non-deterministic (or a combination of both) changes to their "velocities" dependant on their positions. In cases where two or more objectives are to be optimised for, the problem becomes a "multi-objective" or "many-objective" optimisation problem (hereafter referred to by the former) respectively. One algorithm suited for multi-objective is NSGA-II [16]. NSGA-II is an algorithm that offers increased efficiency (complexity) over its predecessor, NSGA [17], offering $O(MN^2)$ time complexity as opposed to its predecessor's $O(MN^3)$, where M is the number of objectives and N is the size of the dataset. In both algorithms, the solution is no longer a single optimal candidate, but instead the Pareto-front (set) of optimal solutions. Where in single-objective optimisation the main indices of performance are accuracy and efficiency, multi-objective optimisation additionally considers:

1) The distribution of Pareto optimal solutions
2) The spread of Pareto optimal solutions

The improvement that NSGA-II offers over NSGA, in this context, is attained through its reduced number of comparisons of fitness by considering the intra-set elitism.

In this paper, we compare a proposed hybrid memetic PSO algorithm with the canonical GA, PSO and Gradient Descent (GD) in optimising the weights and biases of a CNN image classifier, operating on the CIFAR-10 [18] dataset. The comparison involves a stand-alone comparison of the three major Gradient Descent algorithms, Adam [19], SGD [20] and RMSprop [21], before moving onto standard GA and PSO. Finally, we compare the approach of the memetic algorithm, which consists of an an adaptive Lamarckian "search guide" guiding a population of Baldwinian search groups.

A. Literature Review

In the following, we provide the details of CNNs and their components, canonical algorithms of the analytical and metaheuristic optimisation technique variants, including Gradient Descent (GD), GA, PSO and finally, the proposed hybrid Memetic algorithm.

1) Convolutional Neural Network: In a CNN, the most basic layers are that of the feed-forward MLP layers previously mentioned. The input layer consists of in-take neurons. In the forward pass, at the input of each layer, the dot product of the input and the weights leading to the next layer is computed and summed with the vector containing the biases for each neuron. The activation function at each neuron is computed on the value of the previous computation, thus producing the inputs to the next layer. Example activation functions are the Sigmoid [22] and ReLu [23] functions. In each layer, the weights and biases are the learnable parameters of the network, adjusted via a process of backpropagation. During backpropagation, the partial derivatives (Jacobian) of a loss (error) function, with respect to each parameter, are computed according to the error of the output of the final layer - example loss functions are the Cross-Entropy Loss [24] and Mean Absolute Error (MAE) functions. The appropriate adjustments to each parameter is then made. The key distinction of CNNs to the standard MLP is in its additional "receptive" architectural components, mimicking the human ability to visually percieve geometric

structures. These additions include convolutional layers and pooling layers - additionally, the Fully-connected layer. Each convolutional layer operates by way of a convolutional-filter - termed "kernel" - which traverses the dimensions of each input image, including the "layer-wise" channels, responsible for extracting the saptial and temporal structures to generate feature maps as well as reducing dimensionality. The convolutional layers extract increasingly abstract features, not necessarily interpretable to a human. Further, the pooling layers operate on each feature map by substituting the relative kernel space using some representative statistic, further reducing dimensionality and introducing translation equivariance. Example pooling functions include Min pooling and Max pooling, among others, which substitute the kernel space with the minimum or maximum of that space. After iteratively extracting features, the resulting outputs are fed to the final layer of the CNN, the Fully-connected layer. The loss is then measured by measuring the disparity in the predicted label of the input compared to the actual label of the input via a variety of means. For example, using Cross-entropy Loss computes the logarithm of the loss to assign low loss values to high confidence correct predictions and high loss values to high confidence wrong predictions. This not only encourages correct predictions, but also encourages confidence in those predictions. To encourage better performance on a test set through better generalisation, regularisation maybe used.

2) Regularisation: Overly complex models are prone to overfitting to training data. Oftentimes, it is beneficial to approach the optimisation of a CNN as a multi-objective scenario, where the objectives consist of the model's loss on the training set, as well as its loss on the validation set. In such a case, a variety of regularisation techniques can be employed to approach the optimal of this multi-objective problem. One technique involves evaluating the fitness of the candidate solutions using a second "fitness function", the L2 norm. Minimising for the L2 norm (1), termed "L2 regularization" (Gaussian Regularisation) stabilises training by assuming a Gaussian prior on the model parameters.

$$L = \lambda \sum_{i}^{n} w_i^2 \tag{1}$$

The term is analogous to the negative log-likelihood of the Gaussian. The parameter λ determines the strength of the minimisation of the weights - a larger λ implies a simpler model. The two objectives can then be optimised for using NSGA-II. Further techniques of regularisation include dropout [25], which includes spontaneously setting the values of weights from neurons to 0, effectively "dropping" them. This sporadically collapses the search space and forces the model to develop more robustly. The idea is that training with dropout encourages behaviour similar to an electric device working despite operating with a few faulty connections - such a device is considered robust. However, a major obstacle with this approach is that the optimal dropout rate (probability) must be known a priori in order to utilise this to full effect. Reference [26] sought to overcome this with GADropout - an encoding of dropout rate as a GA in a multi-objective problem, where the first objective is the reduction of error due to the dropout configuration, shown in (2), and the second objective was the Chebychev scalarisation of the fitness using an ideal (configurable) dropout mid-value of 0.5, shown in (3). The variable dropout configuration was shown to yield superior results over standard configurations.

$$f_1 = e_1 - e_2 \tag{2}$$

Where e_1 is the error on obtained on the validation set prior to the encoding, e_2 is the error after the encoding of the dropout rates into individual $Ind_{i,*}$ and f_1 therefore represents the improvement in validation error gained after the encoding.

$$f_2 = \frac{1}{L} \sum_{j=0}^{L} |Ind_{i,j} - t| \tag{3}$$

The second objective that limits the dropout rates encoding from being too extreme.

Given the common loss functions of CNNs, the learnable parameters can be optimised using analytical means due to the loss function's continuity and differentiability. The canonical analytical approach is that of Gradient Descent (GD).

3) Gradient Descent: GD emerged from optimization theory through Cauchy's finding of the most rudimentary model of GD in 1847 [27]. GD is the by far the most standard choice of optimisation algorithm for ANNs, as a standalone algorithm or otherwise. In its most basic form, GD consists of calculating the Jacobian of the loss function with respect to the learnable parameters and updating the current weights with the Jacobian according to some determined step-size, specific to the type of GD used. The most used GD variations include Adaptive moment estimation (Adam), Stochastic Gradient Descent (SGD) and Root Mean Square Propagation (RMSprop). Of the three, Adam seems most applicable due to its adaptability to non-stationary objectives, efficiency, performance in high dimensions and ability to navigate noisy and/or sparse gradients.

4) Genetic Algorithms: GA is a population-based metaheuristic optimisation algorithm. Candidate solutions are encoded as real or binary valued chromosomes. At each generation (iteration), the population is recombined and mutated before being re-evaluated according to the objective function. Selection of offspring and survivors for the next generation can be conducted using several techniques, namely fitness proportionate selection, rank proportionate selection and tournament selection, among others.

5) Particle Swarm Optimisation: In standard PSO, candidate solutions are encoded as particles with positions and velocities. New solutions are generated by iteratively updating current solutions using the positions of the global and individual best particles. The stochastisity of PSO is encoded in the use of uniformly generated random scalars that determine the influence of each component on its new speed. The equation for the update of velocities and particles in standard PSO is provided in (4) and (5):

$$v_i{}^{t+1} = wv_i{}^t + c_1 r_{i1}{}^t (p_i{}^{best} - x_i{}^t) + c_2 r_{i2}{}^t (g^{best} - x_i{}^t) \quad (4)$$

$$x_i{}^{t+1} = x_i{}^t + v_i{}^{t+1} \quad (5)$$

where w, c, r denote the inertia weight, constant parameter and uniformly distributed random number between 0 and 1, and t denotes the current time step. v denotes the velocity of the particle, x denotes the position of the particle, g^{best} denotes the global best particle and p^{best} denotes the current particle's personal best.

In [28], a pair-wise competition-based variant of PSO, CSO, was introduced to address the tendency of vanilla PSO to stagnate. CSO was shown to be able to perform well on problem of dimensionality up to 5000D with its ability to balance both exploration and explotiation. Reference [29] introduced a variant of PSO, Social Learning PSO (SL-PSO), which addressed the until-then shortfall of reliance on historical information by updating particles based on a randomly selected and better-performing demonstrator. The work showed the superior ability of PSO to optimise in low dimensional spaces, while showing promise in higher dimensional spaces (100D-1000D). Moreover, it was shown that SL-PSO had a strong ability to optimise for non-stationary and dynamic optima. In the context of batch-trained CNNs where statistical properties may vary slightly between batches, this characteristic is particularly useful. Another approach to addressing the tendency of vanilla PSO to stagnate was used in [30] in which a two-stage approach, consisting of SGD gradient descent followed by PSO, was shown to effectively optimise the parameters of CNNs on CIFAR-10 and CIFAR-100 [31] by using a random-variable step-size approach to utilise both "fast particles" and "slow particles" to explore and exploit, respectively. Another two-stage approach is used in [32] which introduced TLPSO to address the high-diversity of CSO and high-exploitative capabilities of SL-PSO and others. It was shown to be able to outperform state-of-the-art cooperative PSO variants in most partially separable and non-separable and multi-modal functions of up to 1000D. The approach consists of a "study group" phase which operates analogously to a local-CSO. The groups are then filtered for elite members, who are then also updated according to one-another to further their search for optimal solutions.

Despite this, [33] recognised that in most surrogate-assisted PSO algorithms performed well in problems generally under 100D, but still suffered from the curse of dimensionality in large-scale optimization. It proposed a multi-strategy MSL-PSO, in which demonstrators are identified in two separate "sub-nets" and then learned from. MSL-PSO demonstrated competitive performance 20 CEC2008 functions with dimensionalities ranging from 100D to 1000D. Of all approaches, the most robust of them generally tend to be those that employ a hybridised approach, with dedicated mechanisms for exploration and exploitation.

Memetic Algorithms: Memetic algorithms [34] show similarities to EAs in that they model a propagation of change through a population. The distinction is that, while Evolutionary algorithms aim to model natural intelligence for evolution, memetic algorithms model evolution as a sort of cultural influence. Further Memetic algorithms can be categorised into two factions; Lamarckian algorithms operate on the logic that local-search individual improvements propagate on to an individual's offspring - that is, changes in the parent also manifest in the offspring. On the other hand, local improvements in individuals of Baldwinian algorithms do not manifest in their offspring - instead, the improvements is viewed as a "potential" to learn, where those with higher potential are viewed as fitter, but do not have their fitness evaluated as their potential. The idea is that the population will evolve more slowly over time to a more globally-optimum solution and has the advantage of maintaining diversity in the population for longer. The disadvantage is that convergence often occurs after a longer period of time.

6) The Proposed Algorithm: Given the related and aforementioned works, a number of issues persist in proposed algorithms:

1) The curse of dimensionality persists in most approaches. Those which largely overcome this issue do so at a compromise to computational efficiency;
2) Multi-modality is still a major issue in most approaches - those that are able to generalise well enough to a given subset of multi-modal functions often fail to perform well on the complement set;
3) Convergence to local optima persists. Those that are able to escape local optima often do not have a generalised ability to do so.

The algorithm proposed in this paper aims to solve these issues by employing an adaptive hybrid-strategy approach, encoding the weights and biases of the final layer of the neural network as a vector. The algorithm utilises a hybrid-memetic logic, exploting Adam optimser's momentum characteristic to navigate unstable domains, its exploitability and superior ability to converge to non-stationary optima, as well as SL-PSO's explorative nature in high dimensional spaces. The algorithm operates on "influence", determined by a parameter which varies by considering the Baldwinian characteristics of each particle and adjusting the influence of the Adam global best's influence on the rest of the population by means of a weighted moving average of past influence factors. Through this, we observe superior speed and stability in convergence to a more optimal solution.

II. The Architecture

In this section, the data pre-processing pipeline is detailed, followed by a description and justification of the CNN architecture.

A. Data Preprocessing

The data (CIFAR-10) consists of 60,000 RGB images belonging to 10 classes, including animals and vehicles. The data was initially split into stratified training and test sets, with an 80-20 split ratio of 48,000 images in the stratified

training set and 12,000 images in the stratified test set. Of those in the training set, five stratified "folds" were generated in order to perform k-fold cross-validation for a robust model that can generalise well. This meant that each stratified validation fold consistuted 20% of the training set, amounting to 9,600 images withheld for validation. Prior to loading, the three RGB channels of test the images were normalised by z-score using means and standard deviations calculated ad-hoc on the training set. Further, during the training process, the validation fold was normalised using the means and standard deviations calculated only using the train data. The training data was then augmented to ensure a robust and generalised model in the following way:

- A random horizontal flip with 25% probability of 25
- A random vertical flip with 25% probability
- A random conversion to grey-scale with 25% probability
- A random crop, with a padding of 4, was applied with 25% probability

A batch size of 128 was used as it was the best compromise between performance and memory consumption.

B. CNN Architecture

Fig. 1 shows the CNN architecture. Given the time constraints of the project, a relatively simple architecture was selected. Five convolutional layers, each followed by a batch normalisation to stabilise training and a ReLU activation layer, were succeeded by a Max pooling layer. With a standard square kernel size of three and a stride of one, the convolutional layers maintained dimensionality while maximising feature extraction. In sequence, the convolutional layers outputted tensors of 64, 128, 128, 64 and 32 channels respectively. Each Max pooling layer halved the height and width of the preceding tensor, with a square kernel size of two and stride two. In all feature extraction layers, no padding was used. Three fully connected layers followed; the first outputted a vector of 64 and was followed by a dropout layer with 30% probability, the second reduced the representative vector to half the size at 32 outputs, and the final outputted the probabilistic predictions of each class as a vector of size 10. Cross-entropy loss was used for its aforementioned strengths in encouraging confidence in correct predictions. Using this architecture, the decision space for the proposed algorithm was reduced to $1/16^{th}$ the original proposed architecture, from 5130 parameters at the last layer to 330, at a cost of only 4% of the previous accuracy. This resulted in a 16x speed up at run-time of the algorithm.

III. The Proposed Algorithm

A. Overview

The proposed algorithm augments standard local and social learning PSO. It employs a hybridised Baldwinian-Lamarckian approach. A Lamarckian Adam gradient-descent "search guide" particle descends while guiding a population a of Baldwinian-learning local SL-PSO "search parties". SL-PSO is augmented in that an additional parameter, γ, determines, for each Baldwinian particle, the influence of the global Lamarckian search-lead over its local SL-PSO demonstrator. At each discovery of a new optimal value, the Adam search-lead is re-assigned to the particle that made that discovery. The size of the local search groups are determined arbitrarily at 10 particles, but it is highly probable that an optimal search-party size would be determined with consideration of the population size and dimensionality of the search space. By guiding the population with an Adam Lamarckian particle, the over-all convergence of the population is generally stabilised, on average, by Adam's use of momentum. This allowed for navigation of unstable geometry in the fitness landscape, as well as reduced sensitivity to local optima.

B. Addressing Shortfalls of Previous Approaches

To further stabilise the convergence and decrease the sensitivity to local optima, the following logic was applied to the influence-factor, γ:

1) Both a global γ as well as a local γ, specific to each particle, were maintained. The local γ was calculated using slightly differing logic, to provide discretion for preference of its local Baldwinian demonstrator, if it showed superior potential relative to that of the global lead. This allowed for local search groups to move towards a (geometrically close) local optima, over the one being pursued by the global search lead;
2) The global γ was updated used a sliding, weighting moving average of the current and past gammas. This acts almost analogously to "momentum" in that influence is transferred after a sustained period of worse performancer. This meant that short term, temporary increases in loss (decreases in fitness) due to the introduction of a new slightly, statistically different batch, would not cause sporadic and unstable shifts of influence, leading to oscillation or divergence;

Finally, the issue that particles with the best fitness in their perceived local neighbourhood of 10 would not have a demonstrator was addressed by allowing those particles to use a demonstrator which manifests their "potential" - that is, rather than picking an existing particle as their demonstrator, they follow a "virtual" particle that is in the position that they would have been in after one step of Adam gradient descent. In this way, they maintain their Baldwinian nature, but are sporadically influenced by the "better version" of themselves to lead neighbouring particles towards potential convergence.

C. Mathematical Description

The formulae for the determination and update of the global γ is shown in (6) and (7).

$$\gamma^g = 1 - \frac{p^g}{f^g} \tag{6}$$

In a context where the objective function is being maximised, the potential, p^g, would divide the fitness, f^g. Once the threshold number of iterations, determined by the look-back window of accumulated global γ^g, is reached, the momentum-driven equation for the calculation of γ^g is used:

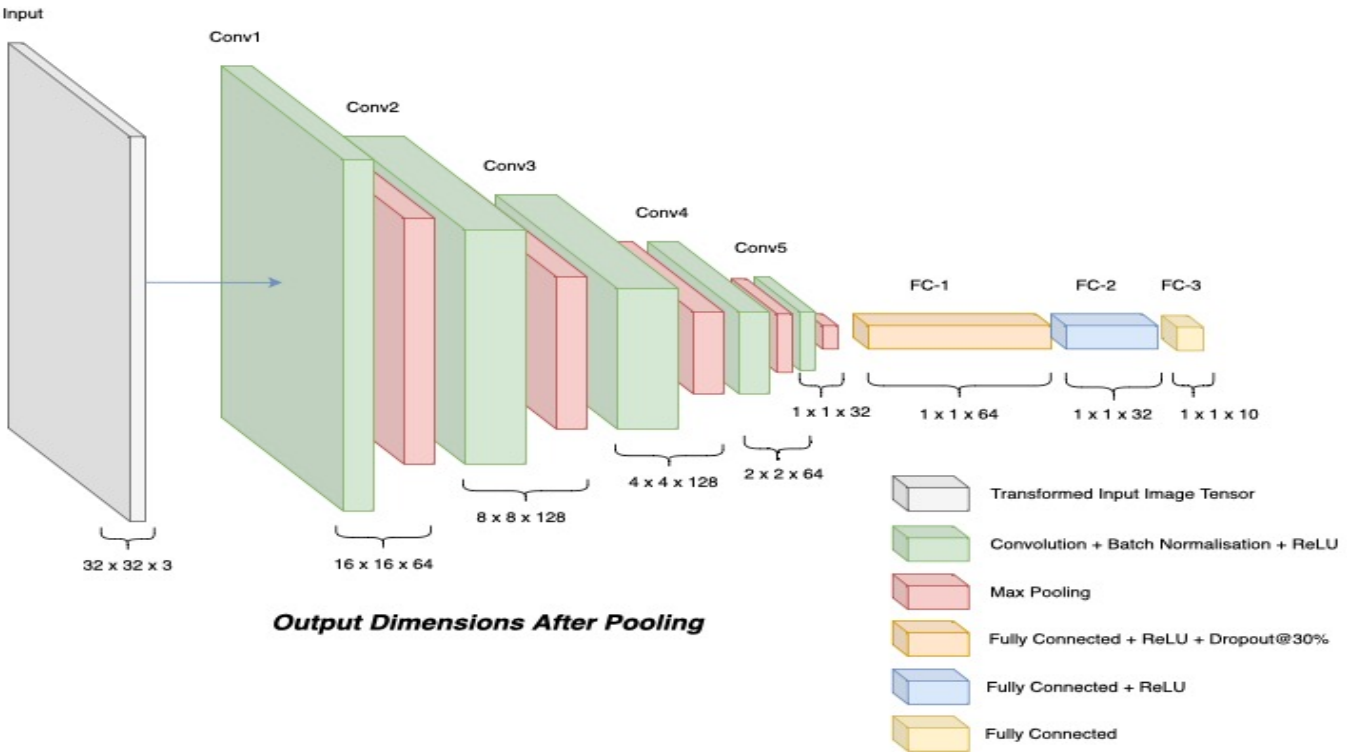

Fig. 1. The model architecture

$$\gamma_t^g = 0.5\gamma_t^g + 0.25\gamma_{t-1}^g + 0.10\gamma_{t-2}^g + 0.075\gamma_{t-3}^g + 0.075\gamma_{t-4}^g \tag{7}$$

Use of both a global and local influence factor allows for "communication" between parties and the lead. This means that inter-party independence is maintained - not every party will follow the global search-lead to the same extent, which sustains diversity. The local influence factor γ^l is determined using formula (8) and the final influence factor γ^f is determined via (9):

$$\gamma^l = \frac{p^d}{f^g} \tag{8}$$

where p^d is the Baldwinian potential of the local demonstrator and f^g is the fitness of the global search-lead. Using this, we use an elitist approach to select the final influence factor γ^f:

$$\gamma^f = min(\gamma^l, \gamma^g) \tag{9}$$

The algorithm maintains computational efficiency by considering a local neighbourhood as the 10 closest particles, rather than imposing geometric constraints on the neighbourhood search. This means that the number of particles searching by Adam gradient-descent is significantly less than the size of the population, and less than the number of those employing a local-search in many Memetic algorithms. This results in a relatively trivial increase in the base complexity of canonical SL-PSO of $O(m^2)$ in the worst case during the sorting phase and $O(mn)$ during the learning phase, where m is the number of particles and n is the number of dimensions. Using this final influence factor γ^f, each Baldwinian particle's position x_b and speed v_b is updated via (10) and (11), whereas the global particle $x_g$search lead is updated via vanilla Adam gradient-descent.

$$v_{ib} = r_{i1}v_{ib} + r_{i2}\mu_i + r_{i3}(\gamma^f x_g + (1-\gamma^f)x_{id}) \tag{10}$$

$$x_{ib} = x_{ib} + v_{ib} \tag{11}$$

Where v_{ib} denotes the i^{th} baldwinian-learner particle's velocity; r_{i*} denote randomly distributed scalars bounded between $[0, 1]$; μ_i denotes the mean of the i^{th} particle's perceived neighbourhood; x_g denotes the position of the global Lamarckian lead particle and x_{id} denotes the i^{th} particle's perceived neighbourhood demonstrator.

Only in the first iteration is the population of n randomly generated individuals limited to a search space between [-1,1], with the fitness and potential of all particles evaluated at the beginning, after which the general scheme of logic for the algorithm consists of:

1) Sort by fitness to find the global lead;
2) Sort all other particles by potential;
3) Calculate γ^g and store it;
4) Local SL-PSO behaviour learning on all Baldwinian learners using a γ^f calculated on an individual basis;

5) A single descent on the global Lamarckian lead;
6) Sort by fitness to find the global lead;
7) Sort all other particles by potential;
8) Returning the best particle to be evaluated on the batch of data at the caller function.

Given the pseudo code for SL-PSO provided in [29], with only the minor additon of the calculation of a local neighbourhood according to the euclidean distance, the calculation of γ^f as mentioned before, and the explicit bounding of particle coordinates to the [-1.1] search space, the general Memetic logic is presented in Fig. 1.

Algorithm 1 Adaptive Strategy Memetic Swarm Optimisation

Initialize a swarm of N particles with random positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$ in $[U(-1,1)]^D$
Evaluate the fitness $f(\mathbf{x}_i^t)$ and potential $f(\mathbf{x}_i^{t+1})$ for each particle i
Sort N by fitness $f(\mathbf{x}_i^t)$
Sort $\{N \backslash x_1\} = \{x_2, x_3, ..., x_n\}$ by potential $f(\mathbf{x}_i^{t+1})$
Initialise $\gamma_t^g = 1 - \frac{f(\mathbf{x_1^t})}{f(\mathbf{x_1^{t+1}})}$
Initialise store for γ_t^g as $G \leftarrow \gamma_t^g$
while termination criteria not met **do**
  Initialise $\gamma_t^g = 1 - \frac{f(\mathbf{x_1^t})}{f(\mathbf{x_1^{t+1}})}$
if length of $G \geq$ length of look-back window **then**
  $\gamma_t^g = 0.5\gamma_t^g + 0.25\gamma_{t-1}^g + 0.10\gamma_{t-2}^g + 0.075\gamma_{t-3}^g + 0.075\gamma_{t-4}^g$
end if
  for each particle $i = 2$ to N **do**
    Find local neighbourhood by euclidean distance
    Find local demonstrator x_d
    Calculate $\gamma^l = \frac{f(\mathbf{x_d^t})}{f(\mathbf{x_d^{t+1}})}$
    Calculate $\gamma^f = min(\gamma^l, \gamma_t^g)$
    Update velocity: $\mathbf{v}_i \leftarrow r_1\mathbf{v}_i + r_2\mu_i + r_3(\gamma^f\mathbf{g} + (1 - \gamma^f)(\mathbf{x}_{id})$
    Update position: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$
    Evaluate the new fitness $f(\mathbf{x}_i^t)$ and potential $f(\mathbf{x}_i^{t+1})$
  end for
  Update position and velocity of global best $\mathbf{g}^t$ by Adam gradient descent
  Evaluate the new fitness $f(\mathbf{g}_i^{t+1})$ and potential $f(\mathbf{g}_i^{t+2})$
  Sort N by fitness $f(\mathbf{x}_i^t)$
  Sort $\{N \backslash x_1\} = \{x_2, x_3, ..., x_n\}$ by potential $f(\mathbf{x}_i^{t+1})$
  Update global best $\mathbf{g}^{t+1} = \mathbf{x_1^{t+1}}$ if necessary
end while=0

IV. Results

A. Training

Given the time constraints of the project, the CNN was pretrained for 20 epochs, after which, the parameters of the last fully connected layer were erased and randomised. All algorithms were used to train only the last layer of the CNN for five epochs. With 320 weights and 10 biases of the final fully connected layer, a standard population size of 100 particles or individuals was chosen for both population based algorithms. Only one of the five stratified validation folds was used during the comparison stage of the training of the three algorithms, due to the time constraints. Since a batch size of 128 was used, this meant that there were 300 batches per epoch and therefore 1500 evaluations of each algorithm across the 5 epochs.

1) GD Parameters:: In the original training of all layers of the CNN for 20 epochs, a learning rate of 0.001 for all gradient descent methods - Adam, SGD and RMSprop - was used. In the second phase involving the comparative training of Adam against GA and the Memetic algorithm, the same standard learning rate of 0.001 was used for Adam, leaving all other parameters to their default values in given in PyTorch.

2) GA Parameters:: In the GA approach, binary grey-coding was used, with a chromosome length of 30 bits was used to represent the individuals. This necessitated the conversion to and from the continuous space when evaluating the individuals. A cross-over probability of 0.8 was used in a two-point cross-over mechanism, with a mutation probability of 0.05. K-elitism was used, with 10 elites of the previous generation carried forward. Tournament selection of the remaining population of 90 was used to select the offspring, with a tournament size of 2.

3) Memetic Parameters:: All default parameters of standard SL-PSO were used for those parameters that were used in this approach. Settings for the additional parameters were as follows: A look-back window of 5, with weights for each gamma value, in reverse order, being (0.5, 0.25, 0.10, 0.75, 0.75). A neighbourhood size of 10 was used.

B. Evaluation

Evaluation involved a single pass through the entire test set of 12,000 images, normalised using the mean and standard deviation values calculated on the training set. Fig. 3 shows the training performance of the three GD based approaches.

Due to the clearly superior performance of Adam, it was chosen to be the baseline gradient-descent optimiser for comparison. Fig. I shows the comparison of accuracy and loss for Adam and the population-based algorithms on the test set. It shows that the proposed algorithm strongly outperformed both Adam and Genetic Algorithms. It was able to attain an almost 30% relative improvement in accuracy and loss over Adam, and about a 10% relative improvement in accuracy over GA, with over 80% relative improvement in loss on the test set. Further, it was able to out perform both optimisers within one epoch. To illustrate the convergence speed and performance trend, Fig. 2 shows the training performance of Adam against the population-based algorithms. The trend shows that, while both Adam and GA seemed to plateau indicating a local optimum, the proposed algorithm, on average, was able to make consistent improvements on each epoch. This supports the possibility that the proposed algorithm maintains a lesser sensitivity to local optima.

Notably, GA and the proposed algorithm were both achieving almost 75% less in relative performance in accuracy in comparison to Adam when in the first evaluation. Despite all being approaches being initialised with a newly randomised

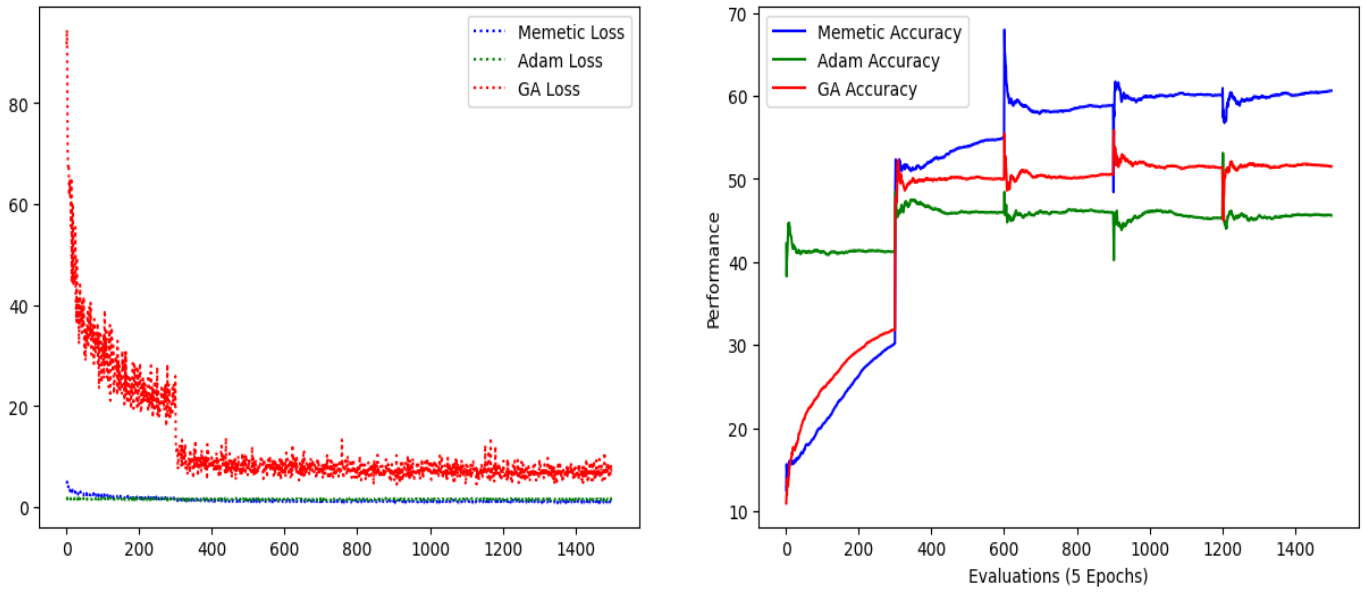

Fig. 2. Training Performance of All GD, GA Memetic

set of weights, there seems to be a correlation with the landscape of both GA and the proposed with respect to their ability to navigate the fitness landscape. Despite re-running the comparisons a number of times, similar results were obtained on every run. All algorithms show a sudden deviation in performance at the beginning of a new epoch before a rapid recovery, at the same instance, indicating that this is a characteristic of the dataset or the stratification process. Moreover, while the proposed algorithm and Adam both exhibited similar characteristic loss curves, GA mimicked the trend but in two orders of magnitude above the aforementioned. Further investigation is required to draw concrete conclusions, but a potential reason might be due to the tendency for the population to hyper-specialise toward a local minimum, producing probability distributions with sub-optimal statistical properties for the log function used in cross-entropy loss, but with non-trivial accuracy. In all cases, the exact same train, validation and test sets were used, which eliminates suspicions regarding coincidental differences in distributions between the algorithms.

The non-trivial increase in performance between training and test sets indicate that perhaps the drop-out probability was set too high, causing the model to under perform on training set. However, this also suggests that the model did not over fit in any case and thus the characteristic plateau exhibited by GA and Adam cannot be explained by this. Further testing on larger and more diverse tests, as well as in higher dimensions over more iterations of the entire data set is required to gain solid insight to the robustness of all three algorithms. In cases where an analytical solution is possible, optimal hybridisation may involve utilising pure Adam for its 4x computational efficiency to reach a preliminary solution, followed by further tuning via the proposed algorithm to minimise error and runtime. In contexts where an analytical solution is not present, GA shows promise in obtaining at least a non-trivial solution - otherwise, Adam descent may potentially be substituted for Adaptive Pattern Search methods, such as in [35], at a compromise to computational efficiency.

TABLE I
LOSS AND ACCURACY AFTER 5 EPOCHS

	Adam	*GA*	*Memetic*
Train Loss	1.52	6.24	1.00
Train Accuracy(%)	45.61	51.51	63.64
Test Loss	1.48	5.97	0.97
Test Accuracy(%)	51.74	60.13	65.43

V. CONCLUSION

In conclusion, an adaptive-strategy Particle Swarm Optimisation with gradient-descent search was used to optimise the trainable parameters of the final feed-forward layer of a Convolutional Neural Network. Results showed that, while gradient-descent maintained superiority in computational complexity and efficiency, the proposed algorithm showed competitive ability to reach a more optimal solution in 20% of the allocated number of evaluations. Further, the proposed algorithm showed superior results in loss and accuracy, though further testing is required to inspect the premature convergence issue of gradient descent. Further directions of research and improvements include further computational optimisation and multi-core adaptation of the proposed algorithm, an investigation of the application of the algorithm for a wider variety of datasets, decision space dimensionalities and neural network tasks, as well as investigation into the robustness of the algorithm over all feed-forward layers of the neural network.

APPENDIX

This work was completed in whole by Ali Shihab, due to lack of completion of tasks (or lack of effort or standard in those attempted) by the rest of the assigned group and subsequent breakdown four days before the project deadline. All work is exclusively Ali's and all measures have been taken to ensure complete independence and uniqueness of work.

REFERENCES

[1] Warren S. Mcculloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Journal of Symbolic Logic*, 9(2):49–50, 1943.

[2] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[3] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[4] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[6] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, pages 255–258, 1995.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. IEEE, 2016.

[10] François Chollet. Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1800–1807, 2017.

[11] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[12] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[13] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[14] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, Cambridge, MA, USA, 1992.

[15] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, 1995. IEEE.

[16] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[17] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[18] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[20] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[21] Geoffrey Hinton. Lecture 6e – rmsprop: Divide the gradient by a running average of its recent magnitude. Coursera: Neural Networks for Machine Learning, 2012. Available online.

[22] David E. Rumelhart, James L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, 1986.

[23] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010.

[24] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[25] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[26] Pengcheng Jiang, Yu Xue, and Ferrante Neri. Continuously evolving dropout with multi-objective evolutionary optimisation. *Engineering Applications of Artificial Intelligence*, 124:106504, 2023.

[27] Augustin-Louis Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes rendus des séances de l'Académie des Sciences*, 25:536–538, 1847. Available at Gallica Digital Library.

[28] Ran Cheng and Yaochu Jin. A competitive swarm optimizer for large scale optimization. *IEEE Transactions on Cybernetics*, 45(2):191–204, 2015.

[29] Ran Cheng and Yaochu Jin. A social learning particle swarm optimization algorithm for scalable optimization. *Information Sciences*, 291:43–60, 2015.

[30] Nguyen Huu Phong, Augusto Santos, and Bernardete Ribeiro. Psoconvolutional neural networks with heterogeneous learning rate. *IEEE Access*, 10:89970 – 89988, 08 2022.

[31] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[32] Rushi Lan, Yu Zhu, Huimin Lu, Zhenbing Liu, and Xiaonan Luo. A two-phase learning-based swarm optimizer for large-scale optimization. *IEEE Transactions on Cybernetics*, 51(12):6284–6293, 2021.

[33] Hao Wang, Mengnan Liang, Chaoli Sun, Guochen Zhang, and Liping Xie. Multiple-strategy learning particle swarm optimization for large-scale optimization problems. *Complex Intelligent Systems*, 7, 05 2020.

[34] Ferrante Neri, Carlos Cotta, and Pablo Moscato, editors. *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*. Springer, 2012.

[35] Ferrante Neri. *Adaptive Covariance Pattern Search*, pages 178–193. 04 2021.

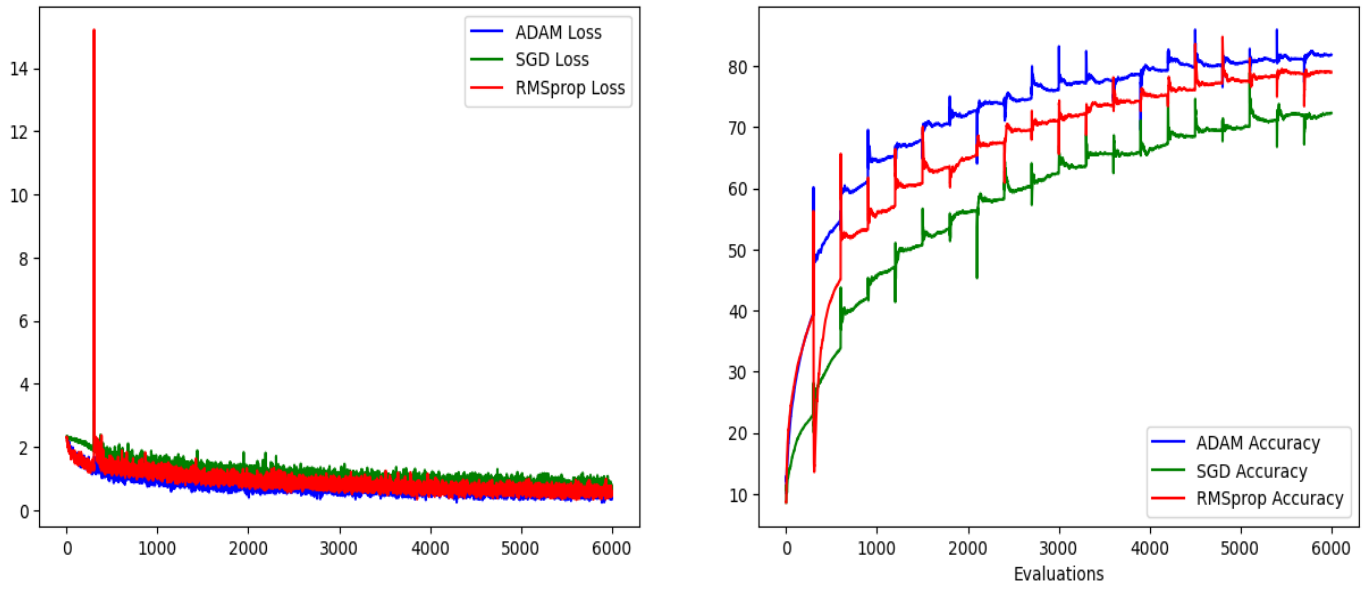

Fig. 3. Training Performance of All GD Optimisers (in 10s of Evaluations)