Particle Swarm Optimisation for Neural Networks

Project Proposal: 4G4 Coursework 1

Group 8: Kimberly Barker (kb619), Louise Aumont (la403), William Halfpenny (wh305)

Introduction

Swarm intelligence (SI) is the name given to the phenomenon where a group of simple homogeneous entities interact to perform a complex task together, without any central control system. These entities, known as "agents", individually follow simple rules, unaware of the global intelligent system created by the local, and partially random, coordination between them. SI is observed in nature in the collective behaviour of ant colonies, flocks of birds and many more animal groups. Artificial SI is an area of biomimetics, which takes inspiration from the principles of swarming behaviour, in applications ranging from simulation and artistic purposes, to swarm robotics and optimisation algorithms.

This research project will focus on the use of the particle swarm optimisation (PSO) algorithm. PSO is just one example of an optimisation algorithm inspired by swarm behaviour, while others include algorithms based on ant colonies and bee hives. The PSO algorithm iteratively searches for the global minimum of a function in an multi-dimensional space. It was derived from the behaviour of flocks of birds and schools of fish. The algorithm is initiated by randomly placing many particles, with random velocities, in the search space. At each step, every particle's velocity is updated according to its previous velocity, its historically best position and the global optimum amongst all the particles. Over time, the particles will search the space and converge to the optimal point evaluated, using a given fitness criterion.

One of the major advantages of bio-inspired algorithms is that they take inspiration from concepts that we are already familiar with. They present their own natural metaphors for the algorithms being used, and this means that they are much easier to understand without a strong mathematical background. One application of PSO is in the training of artificial neural networks, henceforth referred to as neural networks (NNs). A NN is a nonlinear statistical model, which uses a complex relationship between the inputs and outputs to identify patterns and cluster input data. It is one of the most popular machine learning methods used today. NN applications range from image processing and speech recognition to medical diagnosis and various types of forecasting. Currently, NNs are trained using complex mathematical methods, such as back-propagation (BP), which beginners will often struggle to grasp. Switching to the more intuitive method of PSO for this NN training could open up the possibility for more people to take advantage of the opportunities machine learning has to offer. In this research project, we will evaluate whether the original PSO algorithm can train NNs as well as conventional methods. If we find that PSO in its basic form gives comparable performance, it could be used to provide a simpler explanation for the optimisation of neural networks, making this complex field more accessible to beginners.

This report first outlines the problem statement and aims for this research project. Then, a literature review presents an understanding of the current research in this field and how this project contributes to the field. Next, the relevant mathematical models for the project are outlined. Finally, the project plan, detailing how these aims will be addressed over the next 10 weeks, is presented.

1. Problem Statement

We believe that PSO could provide a more accessible explanation of the NN training process. We would therefore like to investigate whether the original PSO algorithm is a viable alternative to conventional NN training algorithms such as BP. We will apply a NN to an image processing task, as the results give intuitive outputs to visualise performance. Thus, the problem statement we would like to explore is the following:

"How does the original PSO algorithm compare to conventional methods for training NNs for image processing?"

The key milestones for this project are the following:

(i) Conduct a literature review to assess current uses of PSO for NN training.
(ii) Create our own implementation of the original PSO algorithm for a standard 2D mathematical function, with a visualisation of the model.
(iii) Train a NN first using the original PSO method, then the BP method, for a simple dataset (e.g., IRIS dataset).
(iv) Test these implementations with standard image datasets of increasing difficulty (e.g., MNIST, facial recognition).
(v) For each dataset, compare the optimisation performance of the original PSO with that of BP.

2. Literature Review

2.1. Particle Swarm Optimisation

The PSO algorithm is inspired by the behaviour of flocks of birds. Flocking behaviour introduces several evolutionary advantages, including increasing the chances of survival against predators and improving the aerodynamic performance of the swarm. Such flocks are observed to move together, with a shared group objective. In 1986, Reynolds was the first to attempt to simulate this flock movement computationally [37]. He discovered that this behaviour was largely governed by the following three rules:

(i) Cohesion: birds steer towards the centroid of their neighbours,
(ii) Separation: birds maintain relative position to neighbours,
(iii) Alignment: birds match their velocity to match that of their neighbours.

These rules define local interactions, which are usually based on the "nearest neighbour principle". This dictates that adjustments are made only via visual observations of surrounding neighbours. The aggregate of these local interactions creates an overall fluid flock movement of discrete elements.

In 1995, whilst working to extend on Reynolds' work on flock simulation, Kennedy and Eberhart found that a simplified version of their algorithm also had optimisation capabilities. This lead to the introduction of the original PSO algorithm, as a non-linear continuous optimisation solver [22]. Kennedy and Eberhart later published a book in 2001 [21], which gives results from a range of PSO experiments and also considers where PSO fits into the larger context of evolutionary computing.

Since then, PSO has been developed in many different variations. It has, for example, been successfully implemented for discrete optimisation problems [51,53] and multi-objective problems [50]. Another notable variation entails changing the neighbourhood topology of the algorithm [15,20]. A recent overview of these different variations of the algorithm was conducted by Zhang in 2015 [54].

In 1995, Kennedy and Eberhart first used PSO in practice for training a NN. Since then, PSO has been applied to a diverse range of problems, such as anomaly detection [46], computer vision [2,44] and robotics [35]. A recent survey of the publications on PSO applications was conducted by Poli in 2008 [34].

The use of PSO has shown promising results in a range of different optimisation tasks. It is defined as a metaheuristic, meaning it is able to generate a satisfactory optimisation solution with limited computation and little knowledge about the optimisation problem. Another strength lies in its ability to optimise large non-differentiable functions. Indeed, because it does not rely on a gradient based approach to find the minimum, the functions it can optimise do not need to be differentiable [19].

However, PSO will not always guarantee finding the global optimum. For example, using original 1995 PSO algorithm [22], if the parameters are not carefully adjusted to the problem, there is a high probability that the algorithm will get trapped in a local area. This occurs due to premature convergence, known as stagnation. PSO also fails for problems where we cannot uniquely define the next and previous particle positions in the search space [32]. Additionally, PSO is not effective in time critical applications, as pathways emerge rather than being pre-defined, and hence it works on unpredictable time scales.

Parameter tuning still poses a large issue in the field of PSO. This parameter tuning is very problem-dependent and often parameters are empirically selected by trial-and-error and previous knowledge. A possible solution to this, is the fuzzy adaptive PSO variation [43], which attempts to tune parameters adaptively as the algorithm runs. Current open research questions in the field include: How can the PSO parameters be tuned effectively? How local should awareness of other particles be? Should agents remain homogeneous?

2.2. Neural Networks

NNs are inspired by our understanding of the structure and learning mechanisms of biological neural networks. The structure of NNs can be described as a collection of nodes, analogous to neurons, connected by edges, analogous to synapses. All edges initially get assigned variable weights, which are adjusted during the learning process in order to vary the importance of the information carried by a given connection. The nodes are usually arranged into a number of layers, through which the signal passes in order. In general, each layer represents a different transformation. Neural networks with hidden layers are known as deep neural networks, which allows for the mapping of complex relationships between inputs and outputs.

A typical NN structure would generally include an input layer, in which nodes receive the input data, one or several hidden layers, in which nodes perform complex non-linear functions on the sum of their inputs, and an output layer, which communicates the patterns that the input data was identified to follow. Feedforward neural networks (FNNs) are ideally used for image processing and follow the aforementioned typical structure, in which the connections between the nodes do not form a cycle. In recurrent neural networks (RNNs) on the other hand, the connections between

the neurons do form loops, allowing information to be stored within the network to inform future reasoning in upcoming events. Their use of internal memory and time-series information makes them ideal for text and speech analysis.

Interestingly, research on NNs has enabled biologists to develop more accurate models of the behaviour of biological neural networks. Therefore, as well as fundamentally being biomimetic by drawing inspiration from the brain, NNs also help us gain a better understanding of biology.

2.2.1. The origins of neural networks

Early NN architectures go back at least to the early 1800s and were essentially variants of linear regression methods [12]. The first ideas about unsupervised learning were published by Hebb in 1949 [16]. Unsupervised learning algorithms take unlabelled data and cluster it by identifying patterns. Supervised learning algorithms use labelled input data to learn links between inputs and the outputs they should be associated to. In the following decades, NN algorithms were developed, trained by supervised [38] and unsupervised [13] learning. Following this, in the 1960s and 1970s NNs, with several successive non-linear layers of neurons, constituted perhaps the first deep learning systems of the feedforward multi-layer perceptron type [18]. A neuron whose activation function is a Heaviside step function is called a perceptron.

2.2.2. The Neocognitron

In 1979, Fukushima's Neocognitron [8] was the first NN to display supervised, feedforward, gradient-based deep learners, with alternating convolutional and downsampling layers [41]. The receptive field of a convolutional unit is assigned a filter and shifted across a 2D array of input values. The resulting activation events of this unit can then provide inputs to the next layer of units [41]. A downsampling layer generates outputs with smaller dimensions than the inputs it receives, by pooling pixels with a given rule e.g., by max-pooling (MP). Differences with today's deep learning algorithms include that Fukushima set weights by winner-takes-all-based unsupervised learning rules [11], rather than supervised BP, and used spatial averaging [9,10] instead of MP for downsampling.

BP is an efficient gradient-descent method for teacher-based supervised learning in discrete, differentiable networks of arbitrary depth. It is used for fine-tuning the weights of a NN, based on the error rate obtained in the previous iteration, to ensure lower error rates, and make the model reliable by increasing its generalisation. The first NN-specific application of efficient BP was described in 1981 [48,49]. This initiated, in the late 1980s, the popularisation of NNs relying on BP-based training. However, in practice, BP-based training of deep NNs was unsuccessful and BP only seemed to work for shallow problems [41].

2.2.3. The fundamental deep learning problem

In 1991, Hochreiter [17] identified why deep NNs were hard to train by BP: they typically suffer from vanishing or exploding gradients. Indeed, with standard activation functions, cumulative BP error signals either decay exponentially in the number of layers, or are not robust to noise and explode out of bounds [3]. This is known as the "Fundamental Deep Learning Problem", or the long time lag problem.

Since 1991, several methods were developed to address this problem. First, in 1991, the "History Compressor" [40] addressed the issue through unsupervised pre-training of each RNN. Thanks to this, only unexpected inputs got fed to the next layer of RNN, and the data got compressed in both time and space, allowing for better stability. Secondly, long short term memory networks used an architecture that is unaffected by the issue. This architecture is typically composed, for each unit, of an input gate, a memory component, a forget gate, ensuring only relevant data is conserved, and an output gate. Another source of progress is that desktop machines' computational power was multiplied by a million since the early 1990s. This allows for propagating errors through more layers without encountering the long time lag problem [6]. A more direct approach to mitigate the problem, is to use Hessian-free optimisation. This improves the NN's generality, while reducing its computational cost and required storage space [30,31]. Finally, the space of NN weight matrices can be searched without relying on error gradients. Successful alternatives include random weight guessing [29], Universal Search [26], combinations of linear methods, and evolving weights of connections [42].

2.2.4. The state of the art and remaining challenges

As previously mentioned, today's deep learning algorithms use MP for downsampling. MP layers were first introduced with the Cresceptron in 1992 [47]. When CNNs are combined with MP, they become Cresceptron-like MPCNNs, with alternating convolutional and max-pooling layers. However, unlike Cresceptron, MPCNNs are trained by BP [36]. BP-trained MPCNNs have become central to many modern, competition-winning, feedforward, visual deep learners [41]. The training of MPCNNs is still a core challenge in the field. This project will contribute to addressing it, by evaluating the relevance of using PSO algorithms, as opposed to BP, to perform MPCNNs' training.

2.3. Applying PSO to NN training

There has been much research surrounding the use of PSO for NN applications: querying the Web Of Science database with the search term "TI= "particle swarm" optimisation neural net*" returned 716 results. Broadly, the applications of PSO to NNs can be split into two parts: the optimisation of parameters and the optimisation of hyperparameters. Parameters are typically defined to be the weights and biases used within the NN. These are the targets of optimisation for algorithms using BP. Hyperparameters refer to variables such as the network architecture and neuron activation functions.

2.3.1. Optimising the parameters

One of the first applications of the algorithm was in the optimisation of NN parameters, presented in the 1995 original paper on PSO [22]. In this work, the authors used PSO to train the weights and biases of a feedforward multi-layer perceptron. They proved that the PSO algorithm produced results that were as effective as error back-propagation (BP), the *de facto* standard method in neural network training. In the 2001 book by the same authors [23], they make the statement: "Evolving artificial neural networks, including not only the network weights but also the network structure, is one of the most exciting early applications of particle swarm optimisation. The method is so simple and efficient that we have almost completely ceased using traditional neural network training paradigms such as backpropagation."

In a highly-cited 2003 paper by Gudise and Venayagomoorthy [14], it was shown that feed-forward NN weights converged faster with PSO than when using gradient descent with BP. Later, improved convergence speed of PSO over BP was again shown in a paper by Lin and Hsieh (2009), where PSO-trained NNs were used for classification tasks [27]. In 2010, parameters of a radial basis function NN were optimised using PSO, giving the best classification performance with the smallest network [25]. More recently, it has been shown that a PSO-optimised CNN could outperform similar best-in-class algorithms in terms of training time and performance [45].

2.3.2. Optimising the Hyperparameters

Optimisation of hyperparameters is one of the major problems currently facing the field of NNs [7]. Currently, good performance of NNs requires manual tweaking of the hyperparameters by machine learning experts. This can be time-consuming, and is also makes the utilisation of NNs inaccessible to those without extensive machine learning education.

Recently PSO has been proposed as an automated method of tuning these hyperparameters. Lorenzo et al. [28] showed that PSO effectively explored the solution space and delivered high quality results on the MNIST and CIFAR-10 datasets, "clearly surpassing human expertise when optimising an existent deep NN architecture designed by experts". PSO has also been used to train both parameters and architectures, in a paper using RNNs to understand the nonlinear dynamics of the gene expression time series [39].

Kiranyaz et al. [24] proposed a technique for NN design that involved a modified version of the PSO that removed the requirement for dimensions to be fixed *a priori*, and produced a ranked list of network configurations. The technique they used evolved to optimum or near-optimum in general, and could effectively generalise.

2.3.3. Comparison of PSO to other algorithms in NN Optimisation

With regard to PSO's advantages over other algorithms in training parameters, it is particularly effective when the function has multiple local optima [23], and can converge faster than BP for certain functions [14]. In particular, it has been found that PSO is particularly effective in the early stages of training [52]. With regard to parameters, PSO benefits from the fact that it does not rely on having a differentiable, or even continuous function. However, one of the main advantages of PSO in this instance is not its performance, but its simplicity, producing competitive results without requiring an extensive understanding of the methodology involved or expertise in the tweaking of network parameters.

3. Mathematical Model

3.1. The Original PSO Algorithm

The original PSO algorithm features a global topology. A group of M particles are first initiated, in the search space, with random positions and velocities, drawn from uniform distributions. We specify the limits of these uniform distributions to be the minimum and maximum positions ($\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$), and velocities ($\mathbf{v}_{\min}$ and $\mathbf{v}_{\max}$) respectively. We must define a desired fitness function $f(\cdot)$ which we wish to maximise over this space, and which can be evaluated at every point.

Algorithm 1: The Original PSO algorithm

```
for each particle i = 1...M do
    Set particle position with value drawn from a uniform distribution: x_i ~ U[x_min, x_max]
    Set each particle's best known position as its initial position: p_i ← x_i
    if f(p_i) > f(p_g) then
        Update global best position p_g ← p_i
    end
    Set particle velocity with value drawn from a uniform distribution: v_i ~ U[v_min, v_max]
end
while termination criterion not satisfied do
    for each particle i = 1...M do
        Draw diagonal entries r of R_1 and R_2 from r ~ U[0,1]
        Use (1) and (2) to update the position velocity
        if f(x_i) > f(p_i) then
            p_i ← x_i
            if f(p_i) > f(p_g) then
                p_g ← p_i
    end
end
```

We then consider the update equations for each particle, i, in an n-dimensional space. Each particle is capable of storing its own historically best position $\mathbf{p_i}$ according to the fitness function $f(\cdot)$. Each particle also has access to information on the global historically best position of any particle in the swarm, $\mathbf{p_g}$. This allows particles to take advantage of knowledge from the entire swarm. The position $\mathbf{x_i}$ and velocity $\mathbf{v_i}$ of each particle i are updated at each iteration time-step, t, according to the following two equations:

$$\mathbf{v_i}(t+1) = \mathbf{v_i}(t) + c_1\mathbf{R_1}\left(\mathbf{p_i}(t) - \mathbf{x_i}(t)\right) + c_2\mathbf{R_2}\left(\mathbf{p_g}(t) - \mathbf{x_g}(t)\right) \tag{1}$$

$$\mathbf{x_i}(t+1) = \mathbf{x_i}(t) + \mathbf{v_i}(t+1) \tag{2}$$

where $\mathbf{R_1}$ and $\mathbf{R_2}$ are two diagonal matrices, and c_1 and c_2 are constants, referred to as the cognitive and social weighting parameters respectively. The matrices $\mathbf{R_1}$ and $\mathbf{R_2}$ allow us to introduce randomness into the system. The entries of the diagonal are uniformly sampled numbers in the range [0, 1]. These entries are updated at every iteration such that there is a random weighting of the update parameters for each dimension and time-step.

The constants c_1 and c_2, are used to balance the exploration and exploitation of positions by the particles. Higher values of c_1 and c_2 lead to a more exploratory nature, with particles moving greater distances in each iteration and hence exploring further through the search space. Lower values lead to a more refined search near the current best positions. The relative values of c_1 and c_2 also allow us to balance the relative importance of the particle's individual history with that of the entire swarm. We refer to $(\mathbf{p_i}(t) - \mathbf{x_i}(t))$ as the cognitive term representing local memory, and $\left(\mathbf{p_g}(t) - \mathbf{x_i}(t)\right)$ as the social term representing the group knowledge. For $c_1 > c_2$, the swarm will be more biased to the cognitive term, whereas for $c_2 > c_1$, the bias is towards the social term. In the original PSO algorithm, $c_1 = c_2 = 2$, giving an equal balance between each. This value of 2 is given to account for the average of the diagonal entries in $\mathbf{R_1}$ and $\mathbf{R_2}$ being 0.5. Recent versions of

PSO have varied the relative values of c_1 and c_2 to improve control of the search ability. The values typically fall in the $[0, 4]$ range with the constraint $c_1 + c_2 = 4$ [33].

We refer to the velocity at the previous time-step $\mathbf{v_i}(t)$ as the inertia of the particle. This ensures that a particle's velocity changes gradually, and it is able to maintain some momentum in its trajectory. This term is particularly. important where a particle reaches a position that is the global best, as in the case where $\mathbf{x_i} = \mathbf{p_i} = \mathbf{p_g}$, and the social and cognitive terms will both be equal to zero. This means that this particle's velocity will remain constant, and it will continue with its current momentum. However if we neglected to take momentum into account the particle would no longer move and remain "trapped" in the same position until the next global optimum is found.

These update steps occur until some termination criterion is satisfied (e.g., convergence, maximum number of iterations, minimum fitness value reached). The full implementation for PSO is outlined in Algorithm 1.

3.2. Modelling Neural Networks

A standard neural network (NN) can be mathematically defined as a computing system made up of a number of simple, highly interconnected processing elements, which map an input x to an output y [5]. The width of a NN refers to the number of neurons, or nodes, in each layer, while its depth refers to the number of layers. These nodes are activated through weighted connections from previously active neurons in previous layers, and pass on their outputs via connections to nodes in the following layers. The training is the learning process of assigning appropriate weights $\mathbf{w_i}$ to the aforementioned connections, to produce the mapping that minimises a given error criterion.

For each layer k, $\mathbf{x}^{(k)}$ denotes the input to the layer and $\mathbf{y}^{(k)} = \mathbf{x}^{(k+1)}$ denotes the output. Each layer transforms the aggregate activation of the network, often through non-linear operations. For each node i, the form of output y_i from layer k is as follows:

$$y_i^{(k)} = \phi\left(\mathbf{w_i}^{\mathrm{T}}\mathbf{x}^{(\mathbf{k})} + b_i\right) \tag{3}$$

where ϕ is some activation function e.g., tanh, sigmoid, and b_i refers to the bias term, which allows us to create an offset in the output.

The algorithms this project will be based upon are MPCNNs (max-pooling convolutional NNs), as these are the most suitable for image based tasks. Fig 1, shows a typical MPCNN structure which combines alternating convolutional and max-pooling layers, followed by two fully connected layers between the input and output layer. The different types of layers are explained below as follows:

- **Convolutional layer:** parameters consist of a set of learnable filters which are convolved with input data, producing a 2D activation map of each filter. This enables the network to learn filters that activate, when it detects a given feature, at a certain spatial position in the input.
- **Maximum pooling layer:** selects the maximum pixel from a group of usually non-overlapping, sub-regions of the initial image. This reduces the dimensionality of an input representation, while retaining the information needed to analyse features contained in these sub-regions.
- **Fully connected layer:** Nodes in fully connected layers have connections to all activations in the previous layer and their activation functions consist in an affine transformation, with matrix multiplication followed by a bias offset. These layers use a rectified linear unit activation function to allow the network to capture nonlinear properties of the mapping.

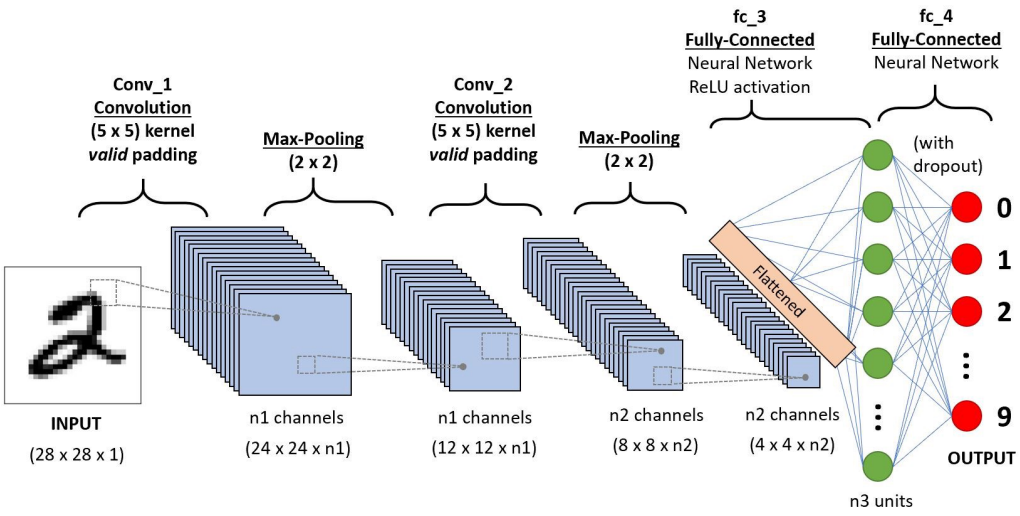

Figure 1: *An MPCNN sequence to classify handwritten digits [1]*

3.3. PSO-NN

To apply PSO to a problem, two things must be defined: the variables to be optimised, and the objective function. In the problem of training a simple feed-forward neural network, the variables to be optimised are the weights and biases linking layers together. If there are $N_{\text{weights}} + N_{\text{biases}} = M$ parameters to be optimised, then a particle's position will be defined as a location in the M-dimensional space, defined by a flattened vector containing all of these values.

The objective function, in our case, will be the accuracy of the classifications. The classifications for our task will be given using one-hot encoding. The performance of the NN will be the similarity between the output vectors $\hat{\boldsymbol{y}}$ and the true vectors $\boldsymbol{y}$, across the training data set. There are more than one ways to define the performance of a neural network on a multi-class classification task, but a common one is cross-entropy loss [4]. This is defined as follows:

$$f(\{\boldsymbol{y}_i\}, \{\hat{\boldsymbol{y}}_i\}) = -\sum_{i \in S} \boldsymbol{y}_i \log \bar{\boldsymbol{y}}_i \tag{4}$$

where S represents the sum over all samples presented in the data-set. The outputs $\{\hat{y}_i\}$ are generated from the calculations within the neural network, using the set of weights and biases specified by a particle's location. Each function evaluation in algorithm 1 corresponds to calculating the output vector for each image in the training data set, and then using equation 4 to get a numerical output.

4. Project Plan

We have a defined a clear workflow for investigating the key question in our project. These tasks are detailed below, with the proposed timings for each given in the Gantt chart (Fig 2). The project is structured to progressively increase the difficulty of the tasks we are trying to solve. We start by solving simple 2D optimisation problems, allowing simultaneous visualisation of the process. We then move on to image classification, training first on simple black-and-white images (MNIST), then progressing onto colour images (CIFAR-10) and finally testing the PSO-NN on dataset of faces for facial recognition. At each stage, the performance of PSO in NN training will be investigated and compared to other standards. In order, we will tackle the following tasks:

1. **Writing the algorithm and visualising the process.** We will implement Algorithm 1 as a function in Python. We will then visualise the optimisation of a simple two-dimensional problem, and investigate the effects of altering parameters.
2. **Using the algorithm to optimise a simple FNN for the Iris flower data set.** This problem is tackled Kennedy and Eberhart [21], and will primarily be used for solving the problem of integrating an optimizer into Keras.
3. **Using the algorithm for classification of MNIST dataset.** This is a benchmark image recognition task, with black-and-white handwritten digits.
4. **Using the algorithm for classification of CIFAR-10 dataset.** This is a slightly larger, colour, benchmark data set for image classification.
5. **Using the algorithm for facial recognition.** Investigating the ability of a PSO-trained neural network to learn to classify images of faces. This will be an instructive real-life test of the functionality of the PSO algorithm for training a CNN.
6. **Further work.** The progression of this project determines the activites of this phase. This may include writing a guide for easy NN optimisation, or testing out different PSO performance on different problems, but also leaves room for unforeseen divergences from the prior plan.

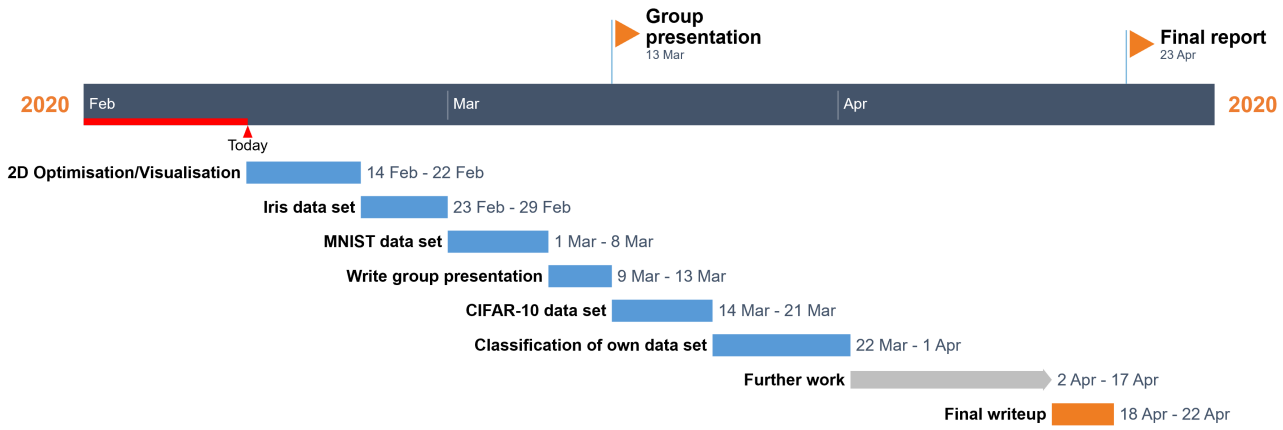

Figure 2: *Gantt chart for our 10-week research plan*

This project is ambitious in scope, given that we are writing our own algorithm and aim to test it on at least four databases, using two training methods for the neural network each time. However, we are mitigating the risks of not reaching our aims within the next 10 weeks by having thoroughly researched these topics already and gradually increasing the level of difficulty of the databases we work with over time. Furthermore, conclusions can be drawn to have an interesting, relevant and novel research project at any point if we fail to achieve the most ambitious of our aims. Research always entails uncertainty and risk but these measures should mitigate their foreseeable consequences.

References

[1] A comprehensive guide to convolutional neural networks — the eli5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (Accessed on 02/13/2020).

[2] L. Anton-Canalis, M. Hernandez-Tejera, and E. Nielsen. Particle swarms as video sequence inhabitants for object tracking in computer vision. *Intelligent Systems Design and Applications, International Conference on*, 2:604–609, 10 2006.

[3] S. P. . F. P. Bengio, Y. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks,*, 5(2):157–166, 1994.

[4] C. M. Bishop and C. M. *Pattern recognition and machine learning*. Springer, 2006.

[5] M. Caudill. *Neural Network Primer: Part I*. AI Expert, 02 1989.

[6] M. U. G. L. M. . S. J. Ciresan, D. C. Deep big simple neural nets for handwritten digit recogntion. *Neural Computation*, 22(12):3207–3220, 2010.

[7] G. I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, and H. Samulowitz. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development*, 61(4/5):9:1–9:11, jul 2017.

[8] K. Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position—neocognitron. *Transactions of the IECE*, J62-A(10):658–665, 1979.

[9] K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[10] K. Fukushima. Increasing robustness against background noise: visual pattern recognition by a neocognitron. *Neural Networks*, 24(7):767–778, 2011.

[11] K. Fukushima. Training multi-layered neural network neocognitron. *Neural Networks*, 40:18–31, 2013.

[12] C. F. Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium.* 1809.

[13] S. Grossberg. Some networks that can learn, remember, and reproduce any number of complicated space–time patterns. *I. Journal of Mathematics and Mechanics*, 19:53–91, 1969.

[14] V. Gudise and G. Venayagamoorthy. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, pages 110–117, Indianapolis, IN, USA, 2003. IEEE.

[15] M. Günther and V. Nissen. A comparison of neighbourhood topologies for staff scheduling with particle swarm optimisation. In *KI 2009: Advances in Artificial Intelligence*, pages 185–192. Springer Berlin Heidelberg, 2009.

[16] D. O. Hebb. *The organization of behavior.* New York: Wiley., 1809.

[17] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen (diploma thesis). *In Automatic differentiation: applications, theory, and implementations*, 1991.

[18] . L. V. G. Ivakhnenko, A. G. Cybernetic predicting devices. *CCM Information Corporation.*, 1965.

[19] Y. Jiang, T. Hu, C. Huang, and X. Wu. An improved particle swarm optimization algorithm. *Applied Mathematics and Computation*, 193(1):231–239, Oct. 2007.

[20] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. IEEE.

[21] J. Kennedy. *Swarm intelligence*. Morgan Kaufmann Publishers, San Francisco, 2001.

[22] J. Kennedy and R. Eberhart. *Particle swarm optimization*. IEEE, 02 1995.

[23] J. F. Kennedy and R. C. Eberhart. *Swarm intelligence*. Morgan Kaufmann Publishers, 2001.

[24] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj. Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. 2009.

[25] M. Korürek and B. Doğan. ECG beat classification using particle swarm optimization and radial basis function neural network. *Expert Systems with Applications*, 37(12):7563–7569, dec 2010.

[26] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[27] C.-J. Lin and M.-H. Hsieh. Classification of mental task from EEG data using neural networks based on particle swarm optimization. *Neurocomputing*, 72(4-6):1121–1130, jan 2009.

[28] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. Ranilla. Particle Swarm Optimization for Hyper-Parameter Selection in Deep Neural Networks. 8, 2017.

[29] . S. I. Martens, J. Bridging long time lags by weight guessing and long short-term memory. *In F. L. Silva, J. C. Principe, L. B. Almeida (Eds.), Frontiers in artificial intelligence and applications: Vol. 37. Spatiotemporal models in biological and artificial systems*, page 65–72, 1996.

[30] . S. I. Martens, J. Learning recurrent neural networks with hessian-free optimization. *In Proceedings of the 28th international conference on machine learning*, page 1033–1040, 2011.

[31] J. Martens. Deep learning via hessian-free optimization. *In J. Fürnkranz, T. Joachims (Eds.), Proceedings of the 27th international conference on machine learning*, page 735–742, 2010.

[32] P. Melin, J. Kacprzyk, and W. Pedrycz, editors. *Soft Computing for Recognition Based on Biometrics*. Springer Berlin Heidelberg, 2010.

[33] K. E. Parsopoulos and M. N. Vrahatis. *Particle Swarm Optimization and Intelligence*. IGI Global, 2010.

[34] R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008:1–10, 2008.

[35] J. Pugh, A. Martinoli, and Y. Zhang. Particle swarm optimization for unsupervised robotic learning. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* IEEE.

[36] H. F. B. Y. . L. Y. Ranzato, M. A. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *In Proc. computer vision and pattern recognition conference*, page 1–8, 2007.

[37] C. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21:25–34, 07 1987.

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

[39] Rui Xu, D. Wunsch, and R. Frank. Inference of Genetic Regulatory Networks with Recurrent Neural Network Models Using Particle Swarm Optimization. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):681–692, oct 2007.

[40] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[41] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 6:85–117, Oct. 2015.

[42] W. D. G. M. . G. F. J. Schmidhuber, J. Training recurrent networks by evolino. *Neural Computation*, 19(3):757–779, 2007.

[43] Y. Shi and R. Eberhart. Fuzzy adaptive particle swarm optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. IEEE.

[44] O. Sjahputera and J. Keller. Particle swarm over scene matching. volume 2005, pages 108 – 115, 07 2005.

[45] A. R. Syulistyo, D. M. Jati Purnomo, M. F. Rachmadi, and A. Wibowo. PARTICLE SWARM OPTIMIZATION (PSO) FOR TRAINING OPTIMIZATION ON CONVOLUTIONAL NEURAL NETWORK (CNN). *Jurnal Ilmu Komputer dan Informasi*, 9(1):52, feb 2016.

[46] D.-M. Tsai, P.-C. Lin, and C.-J. Lu. An independent component analysis-based filter design for defect detection in low-contrast surface images. *Pattern Recognition*, 39(9):1679–1694, Sept. 2006.

[47] A. N. . H. T. S. Weng, J. Cresceptron: a self-organizing neural network which grows adaptively. *In International joint conference on neural networks*, vol. 1:576–581, 1992.

[48] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. *In Proceedings of the 10th IFIP conference, 31.8-4.9, NYC*, page 762–770, 1981.

[49] P. J. Werbos. Backwards differentiation in ad and neural nets: Past links and new opportunities. *In Automatic differentiation: applications, theory, and implementations*, page 15–34, 2006.

[50] B. Xue, M. Zhang, and W. N. Browne. Multi-objective particle swarm optimisation (PSO) for feature selection. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO 12*. ACM Press, 2012.

[51] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, 2000.

[52] J.-R. Zhang, J. Zhang, T.-M. Lok, and M. R. Lyu. A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. *Applied Mathematics and Computation*, 185(2):1026–1037, feb 2007.

[53] W.-J. Zhang and X.-F. Xie. DEPSO: hybrid particle swarm with differential evolution operator. In *SMC03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)*. IEEE.

[54] Y. Zhang, S. Wang, and G. Ji. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, 2015:1–38, 2015.