Particle Swarm Optimisation for Neural Networks

Project Proposal: 4G4 Coursework 1

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

Introduction

Lots of work has been done into specific variants of PSO, tuning parameters and making specializations to particular problems. Such work is important and relevant to the field, as it expands the borders of our knowledge and provides direction for the high-performance applications tackled by our society.

An issue with specialist research such as this is that the search space for individuals trying to understand the problem becomes very large, and unapproachable for newcomers in the field. It requires understanding of complex statistics and knowledge of the best optimisation algorithms for different net structures. At the higher level, it also requires knowledge on the best net structures to use, how to tune the hyperparameters.

One of the major advantages of bio-inspired algorithms is that they draw off concepts that we are already familiar with. They present their own natural metaphors for the algorithms being used, and this means they are much easier to understand without a strong mathematical background. Making something easy to understand for the lay user is an incredibly important goal, as it opens up the opportunity for individuals to take advantage of the power of modern computing and machine learning in other areas. In these applications, "high performance" as defined in the rigorous mathematical sense are often of lesser importance - just take any consumer product (@fitbit, @glucose monitors). A kid learning to code doesn't require 99.9% efficiency in it's facial recognition: it just needs to work.

Swarm intelligence and PSO

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, schools of fish and many more animal groups. Artificial SI has been created to mimic the principles of swarming behaviour, in applications ranging from simulation and artistic purposes, to swarm robotics and the creation of optimisation algorithms.

This project will investigate the application of PSO to the optimisation of the weights and biases of artificial neural networks (ANNs) to best perform an image processing task. This choice seemed relevant for three reasons. Firstly, ANNs are loosely inspired by our understanding of the structure and learning mechanisms of biological neural networks (BNNs), and are thus a product of the field of biomimetics. Secondly, research on ANNs have enabled biologists to develop more accurate models of the behaviour of BNNs. ANNs therefore also contribute to biomimetics by

using engineering to develop a better understanding of biology. Finally, this choice was motivated by ANNs' relevance, given they are the most popular machine learning algorithms today. Their applications range from image processing and speech recognition to medical diagnosis and various types of forecasting.

1. Aims

In this project we wish to investigate PSO as a general-use algorithm for training neural networks. This algorithm is based on models of natural swarming behaviour, and follows a simple set of rules. This algorithm is good because it makes no assumptions about cost function, has good performance and doesn't require a differentiable cost function [citation]. It also has the major advantage that it can be visualized, which we will do in the first part of our project. We have chosen neural networks because, in recent years, these have emerged as one of the most powerful machine learning tools available to us, and there is significant public interest in their use [citation]. Based on review of literature, we theorize that we will be able to achieve decent performance across a range of tasks with minimal alterations to the core PSO algorithm. Using the simple, we aim to show that PSO is able to produce competitive results to other best-in-class algorithms for different data sets, and utilising different architectures.

2. Problem Statement

The hypothesis we would like to test is the following:

"For the classic machine learning problems, of classifying handwritten digits from the MNIST dataset, does a neural network, trained using particle swarm optimisation, perform better than the same neural network, trained with backpropagation?"

Milestones for the project are:

(i) Literature review
(ii) Algorithms for PSO-NN and current state of the art
(iii) Codes to apply these algorithms to our chosen data set
(iv) Compare the optimisation performance with the original method for optimisation.
(v) Conclude on the use of PSO for NN optimisation.

3. Literature Review

3.1. Particle Swarm Optimisation

The PSO algorithm is inspired by the behaviour of flocks of birds. Flocking behaviour introduces several an evolutionary advantages including increasing chances of survival against predators and aerodynamic gains from energy reduction. 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 [32]. He discovered this behaviour could largely governed by three rules: cohesion (birds steer towards centroid of its neighbours), separation (birds maintain relative position to neighbours), and alignment (birds match velocity to their neighbours). These rules define local interactions which are usually based on the "nearest neighbour principle" where adjustments are made only via visual observations of a bird's nearest neighbour. The aggregate of these local interactions, is an overall fluid flock movement of discrete elements.

In 1995, during the creation of an algorithm to extend on Reynolds work of flock simulation, Kennedy and Eberhart found a simplified version of their algorithm also had optimisation capabilities. This lead to the introduction of the PSO algorithm, as a non-linear continuous optimisation solver [20]. Kennedy and Eberhart later published a book in 2001 [19], 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. For example, it has been successfully implemented for discrete optimisation problems [47, 45] and multi-objective problems [44]. Another notable variation is changing the neighbourhood topology of the algorithm [18, 13]. A recent overview of these different variations on the algorithm was conducted by Zhang in 2015 [48].

The first practical use of PSO was by Kennedy and Eberhart which proved to train the weights of a NN as effectively as backpropagation [20]. Since this, PSO has been applied to a diverse range of problems such as anomaly detection [40], computer vision [38, 1] and robotics [30]. A recent survey of the publications on PSO applications was conducted by Poli [29].

The use of PSO has shown promising results in a range of different optimisation tasks. As it is a metaheuristic, it requires little knowledge about the optimisation problem. Its strength lies in the fact it can 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 [17].

However, PSO will not always guarantee finding the global optimum. For example, using original 1995 PSO algorithm [20], there is a chance the algorithm gets trapped in a local area due to premature convergence, known as stagnation. PSO also fails where the problem fails to uniquely and exactly define the next and previous particle positions in the solutions space [28]. Additionally, PSO is not useful in time critical applications as pathways are not pre-defined by emergent. Finally, its parameters must also be tuned very carefully for any given problem. This parameter tuning is very problem-dependent and often parameters are empirically selected by trial-and-error [**?**] or can be adaptively adjusted whilst running as in fuzzy adaptive PSO [37]. A current open research question is how to tune the PSO parameters effectively? How local should knowledge of the environment be? Should particles remain homogeneous?

3.2. Neural Networks

Early NN architectures go back at least to the early 1800s and were essentially variants of linear regression methods [10]. The first ideas about unsupervised learning (UL) were published by Hebb in 1949 [14]. UL algorithms take unlabelled data and cluster it by identifying patterns. Supervised Learning (SL) algorithms use labelled input data to learn a link between inputs and the outputs they should be associated to. In the following decades, NN algorithms were developed, trained by SL [33] and UL [11]. Following this, in the 1960s and 1970s NNs, with several successive nonlinear layers of neurons, constituted perhaps the first Deep Learning (DL) systems of the Feedforward Multilayer Perceptron type [16].

In 1979, Fukushima's Neocognitron [6] was the first NN to display supervised, feedforward, gradient-based Deep Learners with alternating convolutional and downsampling layers [36]. 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 [36]. A downsampling layer generates outputs with smaller dimensions than the inputs it receives. Differences with today's DL algorithms include that Fukushima set weights by winner-takes-all (WTA)-based unsupervised learning rules [9], rather than supervised backpropagation (BP), and used Spatial Averaging [7] [8] instead of Max-Pooling (MP) for downsampling.

BP is an efficient gradient descent method for teacher-based SL 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 [42] [43]. 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 [36].

In 1991, Hochreiter [15] 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 [2]. 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 [35] 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 (LSTM)-like 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 about a million since the early 1990s. This allows for propagating errors through more layers without encountering the long time lag problem [4]. 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 [27] [26].

As previously mentioned, today's DL algorithms use Max-Pooling (MP) for downsampling.

MP layers were first introduced with the Cresceptron [41]. MP is done by applying a max filter to, usually non-overlapping, sub-regions of the initial representation. This reduces the dimensionality of an input representation, while retaining the information required to make assumptions about features contained in the aforementioned sub-regions. 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 [31]. BP-trained MPCNNs have become central to many modern, competition-winning, feedforward, visual Deep Learners [36]. 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.

3.3. The PSO-NN Algorithm

There have been many papers on the use of particle swarm optimisation for neural network applications: querying the Web Of Science database with the search term "TI= "particle swarm" optimi?ation neural net*" gave 716 hits. Broadly, the applications of PSO within neural nets can be split into two part: the optimization of parameters, and hyperparameters. Parameters are typically defined to be the weights and biases used within the neural network. These are the targets of optimization for algorithms using BP. Hyperparameters refer to variables such as the network architecture and neuron transfer functions.

3.3.1 Optimizing the parameters

One of the first applications of the algorithm was in the optimisation of neural network parameters, presented in the original paper on PSO, o [20]. In this work, the authors used PSO to train the weights and biases of a feedforward multilayer perceptron to solve the XOR problem. They showed that the PSO algorithm produced results that were as effective as error back-propagation (BP). In the 2001 book by the same authors[21], they make the statement "Evolving artificial neural netwoks, including not only the network weights but also the netwok structure, is one of the most exciting early applications of particle swarm optimization. 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 [12], it was shown that feedforward neural network weights converged faster with PSO than when using gradient descent with back-propagation (BP), the *de facto* standard method in neural network training. Later, improved convergence speed of PSO over BP was again shown in a paper by Lin and Hsieh (2009), where they used PSO-trained neural networks for classification of mental tasks using EEG data [24]. In 2010, parameter of a radial basis function neural net (RBFNN) were optimised using PSO and used for ECG beat classification, giving the best classification performance with the smallest size of network [23]. More recently, it has been shown that a PSO-optimized convoluational neural network (CNN) could outperform similar best-in-class algorithms in terms of training time and performance [39].

3.3.2 Optimizing the hyperparameters

Optimization of neural network hyperparameters is one of the major problems currently facing the field of neural networks[5]. Currently, good performance of neural networks requires manual tweaking of the hyperparameters by machine learning experts. This can be time-consuming, and is also makes the utilisation of neural networks inaccessible to those without extensive machine learning research.

Recently, it has been proposed to use PSO for automating this hyperparameter optimization. Lorenzo et al. [25] showed that PSO effectively traversed the solution space and delivered high quality results on the MNIST and CIFAR-10 datasets, "clearly surpassing human expertise when optimizing an existent DNN architecture designed by experts". PSO has also been used to train both parameters and architectures, in a paper using recurrent neural nets to understand the nonlinear dynamics of the gene expression time series [34].

Kiranyaz et al. [22] proposed a technique for neural network 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.

3.3.3 Comparison of PSO to other algorithms in neural network optimization

With regard to PSO's advantages over other algorithms in training parameters, it is particularly effective when the function has multiple local optima[21], and can converge faster than BP for certain functions[12]. In particular, it has been found that PSO is particularly effective in the early stages of training [46]. 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 it's performance, but its simplicity, producing competitive results without requiring an extensive understanding of the methodology involved or an expertise in the tweaking of network parameters.

3.4. Summary

Bleep blap blo. P.S.O.

4. Mathematical Model

4.1. The Original PSO Algorithm

The original PSO algorithm features a global topology. A group of M particles are first initiated with random positions and velocities within the search space drawn from uniform distributions. We specify the limits of which these uniform distributions with 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.

We consider the update equations for each particle, i, in an n-dimensional space. Each particle is capable of storing its own historically best position according to the desired fitness function, which we denote $\mathbf{p_i}$. Each particle also has access to information on the global historically best position of any particle in 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 $\mathbf{R_2}$ refer to 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 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 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 best positions. The relative values of c_1 and c_2 also allow us 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] with the constraint $c_1 + c_2 = 4$

We refer to the velocity at the previous time-step $\mathbf{v_i}(t)$ as the inertia of the particle. This ensures 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 this case $\mathbf{x_i} = \mathbf{p_i} = \mathbf{p_g}$ and the social and cognitive terms will both be equal to zero. This means 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., maximum number of iterations, minimum fitness value reached) or until the algorithm converges. The full implementation for PSO is outlined in Algorithm 1 below:

Algorithm 1: The Original PSO algorithm

for *each particle* $i = 1 \dots m$ **do**
  Set particle position with value drawn from a uniform distribution: $\mathbf{x_i} \sim U[\mathbf{x}_{\min}, \mathbf{x}_{\max}]$
  Set each particles best known position as its initial position: $\mathbf{p_i} \leftarrow \mathbf{x_i}$
  if $\mathbf{f}(\mathbf{p_i}) > \mathbf{f}(\mathbf{p_g})$ **then**
    update global best position $\mathbf{p_g} \leftarrow \mathbf{p_i}$
  end
  Set particle velocity with value drawn from a uniform distribution: $\mathbf{v_i} \sim U[\mathbf{v}_{\min}, \mathbf{v}_{\max}]$
end
while *termination criterion not satisfied* **do**
  for *each particle* $i = 1 \dots m$ **do**
    Draw diagonal entries r of $\mathbf{R_1}$ and $\mathbf{R_2}$ from $r \sim U[0, 1]$
    Use (1) and (2) to update the position velocity
    if $f(\mathbf{x_i}) > f(\mathbf{p_i})$ **then**
      $\mathbf{p_i} \leftarrow \mathbf{x_i}$
      if $f(\mathbf{p_i}) > f(\mathbf{p_g})$ **then**
        $\mathbf{p_g} \leftarrow \mathbf{p_i}$
  end
end

The original PSO algorithm is designed to optimise real-value continuous problems. Other variations exist with different topologies and weightings. It has even been adapted for use in discrete optimisation problems.

4.2. Applying PSO to NN

To apply PSO to a problem, two things must be defined: the variables to be optimized, and the objective function. In the problem of training a simple feed-forward neural net, the variables to be optimized are the weights and biases linking layers together. If there are $N_{\text{weights}} + N_{\text{biases}} = M$ parameters to be optimized, then a particles position will be defined as a location in 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 net will be the similarity between the output vectors $\hat{y}$ and the true vectors 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 [3]. This is defined as:

$$f(\{y_i\}, \{\hat{y}_i\}) = -\sum_{i \in S} \boldsymbol{y}_i \log \bar{\boldsymbol{y}}_i \tag{3}$$

Where S represents the sum over all samples presented to the data-set. The outputs $\{\hat{y}_i\}$ are generated from the calculations within the neural net, 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 3 to get a numerical value out.

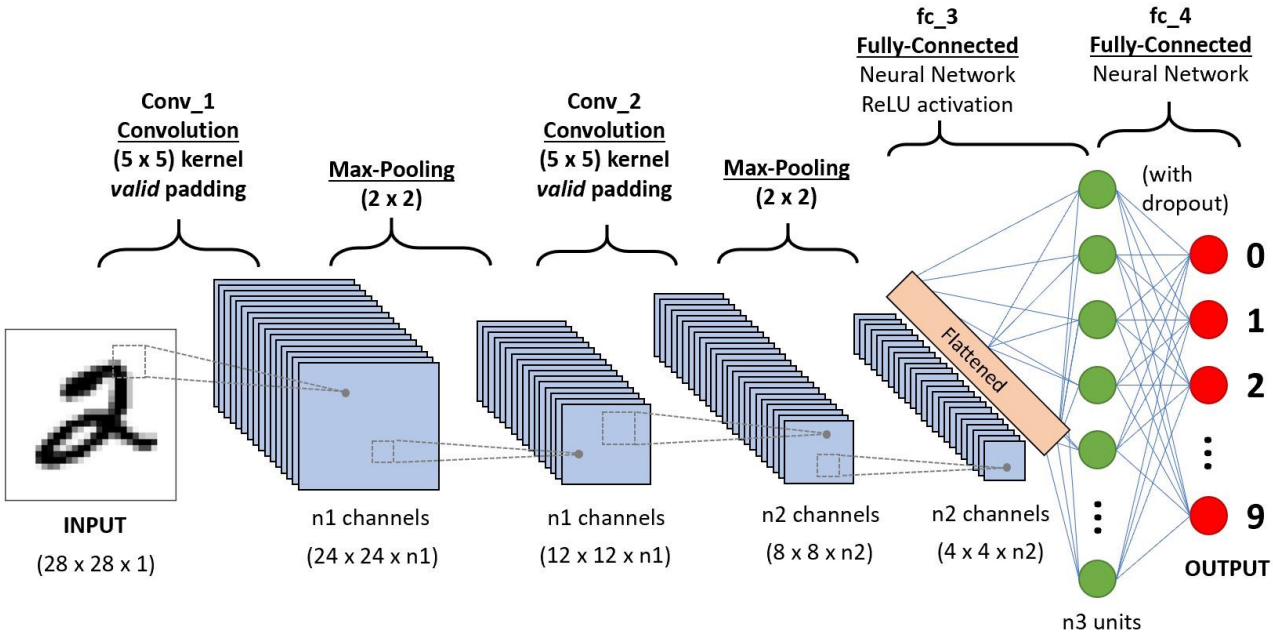

Figure 1: *An MPCNN sequence to classify handwritten digits*

4.3. Architectures to investigate

We have decided to focus on image classification. This is an important topic and a widely-used aim in neural network literature.

BRIEFLY RUN THROUGH THREE PROBLEMS THAT WE WOULD LIKE TO SOLVE: IRIS DATA SET, MNIST,

The algorithms this project will be based upon are MPCNNs. Figure 1, shows a typical MPCNN structure, with an input layer, alternating convolutional and max-pooling layers, fully connected layers and an output layer.

The convolutional layer's parameters consist of a set of learnable filters, or kernels, which are convolved with the input data. This produces 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. The MP layer then selects as an output only the most relevant data from its input. Then the rectified linear unit (ReLU) layer sets the negative values from its input activation map to zero to increase the nonlinear properties of the network. Finally, before obtaining outputs, high-level reasoning occurs in the fully connected layers. Neurons 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.

5. Project Plan

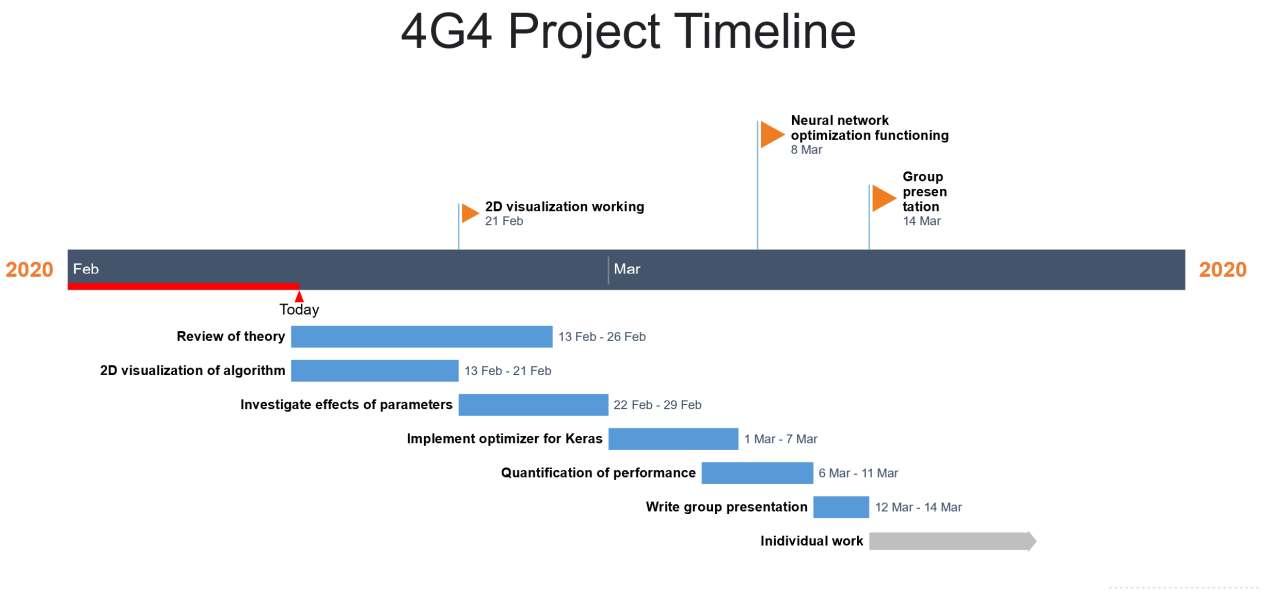

Figure 2: *Project plan for the group component of the 4G4 project*

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] 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.

[12] 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.

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

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

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

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

[23] 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.

[24] 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.

[25] 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.

[26] . 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.

[27] 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.

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

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

[30] 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.

[31] 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.

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

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

[34] 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.

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

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

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

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.