Natural Computing Assessment

Exam Nos. B156771, [insert exam number here]

November 16, 2020

1 Neural Network Training with PSO

1.1 Choosing the Fitness Function

This task is interpreted to mean that we are to design a fitness function to evaluate the fitness of the network as a whole after it has been trained. Let the training error e_T, and the testing error e_τ. Taking the average of the two would be a sensible, if not standard course of action. However, a fitness function would need to incorporate a measure of how overfit the model is to the training data in order to be effective. In this sense, adding the difference between the two values is simple and effective. Ultimately, the fitness function takes the following form:

$$f(e_T, e_\tau) = 1 - \left(\frac{e_T + e_\tau}{2} + (e_\tau - e_T)\right), \tag{1}$$

where the error is calculated using the Mean Squared Error function. We subtract the average and distance from 1 to ensure that a greater fitness value implies a better model.

1.2 Defining the Search Space

The search space of a PSO is given by $[a, b]^D$, where $a, b \in \mathbb{R}$, and $D \in \mathbb{N}$ is the dimension of the space. We define the shape of a neural network by a sequence $(a_n)_{n=1}^{l}$ where $l \in \mathbb{N}$ defines the number of layers, and each $a_n \in \mathbb{N}$ gives the number of neurons in layer n. The dimension of the weight vector for a would be the number of edges connecting each sequential pair of layers, plus the biases, given by the number of neurons in the latter of the pair of layers. Succinctly, the dimension D of the search space is given by,

$$D = \left(\sum_{i=1}^{l-1} a_{n+1}\left(a_n + 1\right)\right) - a_l, \tag{2}$$

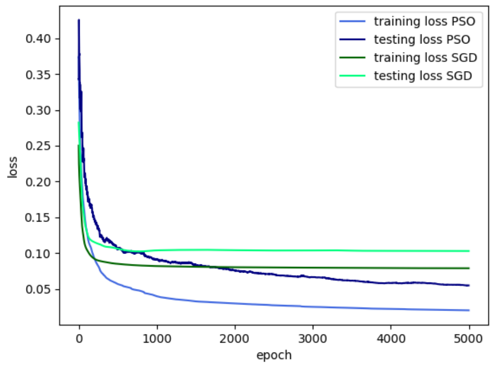
where the final decrement addresses non-existent bias terms in the output layer that were counted in the summation.

For this task, we set the shape to be $(6, 8, 1)$, meaning that the dimension of the search space $D = 43$. To ensure that we are notified of a divergent swarm, we set $-a = b = 20$, meaning that the search space for the PSO will be $[-20, 20]^{43}$.

1.3 Training the Model

For this task we trained two neural networks: one using Particle Swarm as its optimiser, and another using Stochastic Gradient Descent. The model using SGD provides a fairly standard baseline on which to test the PSO model. In the code provided, the PSO neural net does not use any third party machine learning libraries as due to the nature of PSO, these neural nets only needed to consider forward propagation, not backwards. For the SGD optimised net, Pytorch was used due to the myriad of material online explaining how to code it.

Each network was trained with 10-fold cross validation using the cross-validate program provided (refer to the README in the task-1 directory on how to use it). The parameters for the PSO were as follows: $n = 10$, $\omega = 0.6$, $\alpha_1 = \alpha_2 = 1.61$. The parameters for the SGD were as follows: learning rate = 0.03, momentum = 0.4. The figure below shows the training and testing loss of both over time:



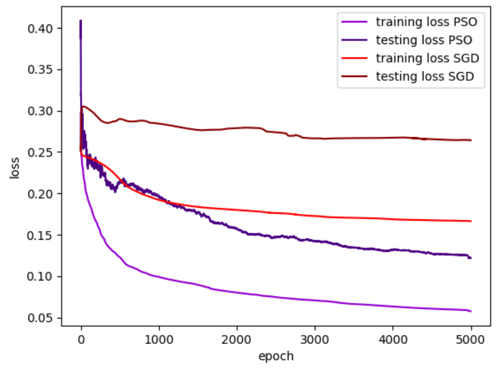
The table below shows the final average training and testing losses, as well as the average fitness using equation (1):

Optimiser	Average Training Loss	Average Testing Loss	Average Fitness
PSO	0.0159	0.0378	0.0489
SGD	0.0828	0.1164	0.1332

By these results it would appear that the model trained using PSO as its optimiser performed significantly better than the model trained with SGD. In a way, this could almost be expected. Gradient Descent follows the single path of a point gradually moving along the steepest descent in the search space. In contrast, Particle Swarm has several particles searching for the (ideally) global, if not various local minima. This means that PSO is more likely to find a global minimum than SGD is, though how much more likely is still to be determined. Nevertheless, it is clear that for our purposes the PSO is superior. While it was slower to train, the low loss on both the training and testing sets for each fold of the cross validation demonstrates that it is well suited for neural network training. Furthermore, by eliminating backpropagation as a consideration in training, developing the model is conceptually simpler as well.

1.4 Restriction to Linear Inputs

Imposing a restriction to linear inputs, we decided to change the architecture of the models. For this part of the task, the models shape will be $(2, 6, 5, 1)$. This yielded the following results:



Optimiser	Average Training Loss	Average Testing Loss	Average Fitness
PSO	0.0657	0.1248	0.1574
SGD	0.1639	0.2654	0.3162

The restriction to linear inputs was clearly a terrible idea. These models performed significantly worse in all aspects when compared to their nonlinear counterparts, which is to be expected. Linear inputs do not lend themselves well to the curvature of the spiral dataset. By allowing nonlinear transformations to the data point, we are able to introduce curved aspects to the output space which is able to better fit the data.

1.5 The Effect of Particle Swarm Parameters

For this part of the task, we took a look at how changing various parameters affected the final training and testing errors, as well as the overall fitness of the model. First, we took a look at varying ω, considering $\omega = \{0.1, 0.2, \dots, 0.9\}$. The table showing these results is in the appendix. Here it is helpful to remember that ω indicates what percentage of the previous iteration's velocity is still present in the particle. Too little and the particles would have trouble moving, being guided primarily by the personal and global bests. Too much and the particle would have a hard time changing direction when the swarm dictates it to. One would expect the optimal ω to be somewhere near the middle then, and indeed our results show that the best $\omega = 0.7$, earning the network a fitness of 0.9208.

While ω concerns itself with the velocity term, the parameters α_1 and α_2 help determine the weight of the personal and global bests in comparison to the particles current position. We ran a similar experiment varying the value of α between 1.2 and 2.1. We decided for the sake of simplicity to follow convention and keep $\alpha_1 = \alpha_2$. The table with this data is also in the appendix.

Observe that the difference between the fitness of each model as we change the α parameter is much smaller than when we changed ω. More notable; however, is that while changes in ω appeared to provide a smoother change in the fitness, changes in the α term do not, meaning that it would be much harder to choose properly. It stands to reason that the higher the value of α, the more volatile the particle's movements will be, unless the particle is close to the optimal position. So a higher α would allow the possibility for a particle to overshoot the optimal position. In contrast, too low an α and the particle would be guided only by its intertia. Again, logically the best place to start looking then would be a median point between the two extremes. The table in appendix A shows that the α value with the best fitness is $\alpha = 1.5$, which does fit with our hypothesis. This, of course, is a heuristic for where to start looking rather than a definitive guide. From this, one could draw the conclusion that extreme values for these parameters would almost never provide an optimal solution, and that it would be better to start with a more middle value for each and adjust accordingly (or even better, use a MHO algorithm to find the optimal parameters).

A Tables and Figures

Results of experiments with varying ω ($\alpha_1 = \alpha_2 = 1.6$):

ω	e_T	e_τ	f
0.1	0.2372	0.2975	0.6723
0.2	0.1946	0.2733	0.6873
0.3	0.1467	0.195	0.7808
0.4	0.1823	0.252	0.7131
0.5	0.0928	0.1714	0.7893
0.6	0.0129	0.0629	0.9121
0.7	0.0097	0.056	0.9208
0.8	0.1976	0.2541	0.7176
0.9	0.4023	0.2909	0.542

Results of experiments with varying α ($\omega = 0.7$):

α	e_T	e_τ	f
1.2	0.027	0.0987	0.8654
1.3	0.0297	0.0683	0.9124
1.4	0.0163	0.1015	0.8559
1.5	0.0064	0.0214	0.9711
1.6	0.024	0.1602	0.7718
1.7	0.029	0.1351	0.8118
1.8	0.0158	0.0395	0.9481
1.9	0.1669	0.2009	0.7821
2.0	0.2168	0.2032	0.8036
2.1	0.2691	0.122	0.9515