PSO-Neural Network Implementation

As a second sub-project, we discovered the use of Particle Swarm Optimization as a method for optimizing the weights of neural networks. While most neural network implementations used these days update the weights during Backpropagation with the use of Stochastic gradient descent our goal is to solve this weights optimization task with an algorithm discussed in class. Unlike the feature selection algorithm, which is a discrete optimization problem, the Neural network weight optimization is a continuous optimization task to solve. Although there are some modifications that could be made to the genetic algorithm (real-value-/Gray-Coding representations, etc.) the ordinary form of Genetic Algorithm can only deal with discrete features. On the other hand, Particle Swarm Optimization due to its functioning that is described in the following can perfectly deal with continuous input variables.

How PSO work

Each Particle of the swarm has information about (1) the best position of all Particles combined, (2) the best position of the individual Particle, and (3) a vector describing the current direction of the Particle. With these three numbers, the velocity (exact movement of a Particle in a specific interation) can be calculated as shown in figure 1 in the Appendix. Please note that PSO due to the random component in this formula is a stochastic algorithm. Each Particles position will be updated until a termination criterion is reached.

PSO-Neural Network Implementation

In order to calculate the Fitness corresponding to a set of weights and biases, we implemented our own forward propagation using mainly numpy. Please note that our implementation allows the user to freely specify the number of hidden layers as well as the number of neurons per layer. In order to introduce non-linearity, we implemented relu activation functions. Since we would like to solve binary classification tasks exclusively, we introduce a sigmoid activation function with a default (but changeable) threshold of 0.5 in the last layer of the Neural Network. Please note that throughout this essay, the terms “weights/biases” and “Particle position” can be used interchangeably. Each Particle is moving in an n-dimensional space, where n is the total number of weights/biases of the neural network and each dimension of the position represents a certain weight/bias. Hence, the optimal combination of weights/biases is a specific point in the n-dimensional space, that we hope all Particles of the Swarm converge around.

Our implementation follows these steps:
1. Random Initialization of weights and biases (in other words: random initialization of one Particle’s position)
2. Using these weights to do the forward propagation
3. Obtain a prediction for each input data point
4. Calculating the fitness by comparing the predictions with the ground truth (metric either Accuracy or Cross-entropy

Updating the weights in the next iteration of the PSO algorithm (in other words: updating all Particle positions
6. Step 2: Calculate the fitness of the updated (presumably superior) weights using Forward
7. Continue steps 5 and 6 until the max_iterations is reached
8. Particle with the best fitness represents the optimized set of weights and biases

Please note that whether the metric Accuracy or Cross entropy is used, it can in both cases be seen as a minimization problem. In the case of accuracy, a value of 100% is as meaningful as a value of 0%. The only difference between these two cases is that the binary prediction that the neural network makes needs to be reversed from 1 (0) to 0 (1). Hence instead of bothering to implement a maximization heuristic, we simply minimize the accuracy and then use the opposite of the prediction as a result of the classification.

PSO Parameter Selection

The user of our implementation needs to specify the following PSO Parameter: Number of Particles, number of iterations, the inertia weight (w), and the two trust parameters c1 and c2. The optimal choice of these parameters heavily depends on the dataset and can be seen as hyperparameters that need to be tuned during the optimization process.

Tests on various Classification tasks

Dataset (binary Classification)	Average Accuracy of best Particle after 1000 iterations (10 runs)	Time per run (in sec.) on an M1 ARM CPU	Neural Network layers
2 features - centroids with stdev of 3 (2000 records)	92.13%	10.2	(2,16,1)
2 features - Spiral decision boundry (194 records)	68%	17.1	(2,16,16,1)
20 features - n_informative features = 15 n_redundant features = 2 (3000 records)	85.7%	50.4	(20,32,1)
All classifications were done with a constant inertia weight of 0.73 and c1=c2=1.49. By tuning these parameters it could be avoided to get stuck in a local minimum in some cases.			

Proposal for future work

In our work, we initialize the weights of the Particles at random (between -1 and 1). It would be interesting to investigate the effect of other Initialization methods such as Zero initialization of the Particle's Position or distributing the Particles equally across the Input space and compare the results to an initialization strategy at random. Finally, the fine tuning of the PSO Parameters is something that requires further research.