F20BC : BIOLOGICALLY INSPIRED COMPUTATION

COURSEWORK REPORT

Iman Teh (H00422856)

Sweta Acharya (H00366051)

1. Implementation

We made classes for the Artificial Neural Network (ANN) and Particle Swarm Optimisation (PSO) .

1.1 Artificial Neural Network (ANN)

The ANN class stores the following:

- **layerSize**: We stored the architecture of the neural network as a list. Hence, each entry represents the number of nodes (input, hidden, and output layers).
- **activationFunction**: We stored the activation function that will be applied to the output of each layer. We used logistic, ReLU, leaky ReLU, and hyperbolic functions. For regression tasks, we used a linear activation function.
- **Weights**: A list of randomly initialised weight matrices for each layer.
- **Biases**: A list of randomly initialised bias vectors for each layer.

The neural network architecture is defined with a layer sizes of [8, 16, 8, 1]

- **Input Layer**: 8 nodes for 8 input features
- **Hidden Layer 1**: 16 neurons
- **Hidden Layer 2**: 8 neurons
- **Output Layer**: 1 node for the regression output.

1.2 Forward Propagation

At each layer, the input is multiplied by the weight matrix. Biases and the activation function are then added. The process continues iteratively through all layers until the final output gets a set of predicted values.

1.3 Loss Function

We applied the formula of :

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

This formula takes the average of the absolute difference between the predicted and true values.

1.4 Particle

- **particlePosition**: Stores the current position of the particle in the search space
- **particleVelocity**: Stores the current velocity of the particle, initialised randomly
- **bestPosition**: Tracks the best position of the particle in terms of fitness and will be returned back to the current position
- **Informants**: Stores a list of informant particles that influence this particle's movement.

1.4.1 particleToAnn

A function within the particle class that maps the particle's position in the search space to the weights and biases of the ANN.

1.4.2 assessFitness

This function computes the loss of the ANN by evaluating the fitness of a particle

1.5 Particle Swarm Optimisation

Class Attributes

- **swarmSize**: The number of particles in the swarm.
- **Alpha**: inertia weight (initial velocity)
- **Beta**: Coefficient for the particle's personal best position.
- **Delta**: Coefficient for the informant's global best position.
- **Gamma**: Coefficient for the best position among the particle's informants
- **jumpSize**: size of jump of the particle in the search space.
- **informantCount**: Number of informants for each particle.
- **vectorSize**: Dimensionality of the search space.
- **Global_best**: The best particle globally across the entire swarm.
- **Global_best_fitness**: The fitness value of the global best particle.

1.5.1 psoOptimisation()

We created a method for executing the PSO algorithm. We initialised the Particle by creating an array of Particle objects, each with random initial positions and velocities. We then evaluated the fitness of each particle in the swarm and updated the global best particle (best) based on fitness. After, the velocity is then updated.

Formula:
$vnew=\alpha \cdot vcurrent+\beta \cdot (pbest-pcurrent)+\gamma \cdot (nbest-pcurrent)+\delta \cdot (gbest-pcurrent)vnew=\alpha \cdot vcurrent+\beta \cdot (pbest-pcurrent)+\gamma \cdot (nbest-pcurrent)+\delta \cdot (gbest-pcurrent)$

Particle's position is then updated using the formula below

$Pnew = pcurrent+jumpSize \cdot vnewpnew=pcurrent+jumpSize \cdot vnew$

Output **:** returns the position of the particle with the best fitness after the optimisation process

2 Experiments and Results

While developing and testing the Artificial Neural Network (ANN), we experimented with various hyperparameters to evaluate how different configurations influenced the regression performance of our model. We have made adjustments to the number of hidden layers, the number of neurons per layer, activation functions, learning rates, and the parameter of the Particle Swarm Optimizations.

2.1 Hyperparameters Used

These are the PSO and ANN hyperparameters that we have used during testing:

- **Swarm Parameters**:
    - Swarm size: 10 particles.
    - Alpha (α): 0.9
    - Beta (β): 0.7
    - Gamma (γ): 0.5
    - Delta (δ): 0.5
    - Jump size: 0.5
    - Informant count: 2 informants (per particle)

    - Maximum iterations: 10.

2.2 Experimentation with Hyperparameters

Here are the key aspects of the experimentation

- **ANN Architecture**:

  We have tested architectures from 1 to 3 layers. To achieve this, we have experimented with neuron counts ranging from 10 to 50 per layer. We found that networks with more layers and neurons usually have better initial performance but were prone to overfitting, especially when working with a small dataset.

- **Learning Rate**:

  Learning rates between 0.001 and 0.1 were tested as well. We found that higher learning rates resulted in faster convergence but increased the risk of instability, while lower rates slowed down the training process but provided more consistent results.

- **PSO Parameters**:
    - Population sizes: 20, 50, 100.
    - Informant counts: 3, 5, 7.
    - Iterations: 50, 100, 200.
    - From these tests, the optimal results were achieved with a population size of 50, 5 informants, and 100 iterations.

2.3 Key Results

We found that before optimising and when running the ANN with default hyperparameters, we observed a high loss value of approximately **34.78**. This value shows a high prediction error. However, after adjusting the parameters of the PSO algorithm (after optimisation), the fitness value has improved to approximately **6.88**, thus showing a dramatic reduction in loss.

2.4 Challenges

- Overfitting was a recurring issue, especially with networks that use too many neurons per layer. To address this, we particularly adjusted the architecture to balance the complexity.

REFERENCES

Lones, M. (2011) 'Sean Luke: essentials of metaheuristics,' *Genetic Programming and Evolvable Machines*, 12(3), pp. 333–334. https://doi.org/10.1007/s10710-011-9139-0.

Training a Neural Network — PySwarms 1.3.0 documentation (no date). https://pyswarms.readthedocs.io/en/latest/examples/usecases/train_neural_network.html.