Groundwater quality classification using PSO and Spatial CNN

1 Introduction

This document discusses the problem of classifying groundwater quality and presents a solution that leverages a Spatial Convolutional Neural Network (SCNN) optimized with Particle Swarm Optimization (PSO).

1.1 The Challenge

Groundwater quality assessment involves categorizing water samples into various quality classes, such as **Excellent, Good, Poor, or Very Poor**. Achieving accurate and timely classification is a complex task that depends on the **selection of hyperparameters** for the machine learning model.

The hyperparameters of an SCNN, including the **number of filters, filter size, stride, and the number of neurons in a hidden layer,** play a critical role in determining the model's performance. Manually tuning these hyperparameters is a labor-intensive process, and an optimal configuration might be elusive.

Our approach to addressing this challenge involves the following steps:

2 Data Preprocessing

- Raw groundwater data is preprocessed.
- Each sample is represented as a multidimensional array, typically in the form of a matrix.

3 Spatial Convolutional Neural Network (SCNN)

- The SCNN architecture is designed for spatial data analysis.
- It consists of convolutional layers to extract relevant features from the groundwater samples.
- Pooling layers are used for spatial down-sampling.
- Fully connected layers, including a hidden layer, are incorporated for classification.

3.1 SCNN Architecture

- Input layer: Raw spatial data
- Convolutional layers: Feature extraction
- Pooling layers: Spatial down-sampling
- Fully connected layers: Classification
- Output layer: Groundwater quality classification

4 Particle Swarm Optimization (PSO)

- PSO is employed to find the optimal set of hyperparameters for the SCNN.
- ***Each particle in the swarm represents a hyperparameter configuration for the SCNN.***
- The optimization process aims to **minimize the categorical cross-entropy loss**, the fitness function.

4.1 Design variables (Parameters to be updated)

The primary hyperparameters for an SCNN typically include:

1. **Number of Filters (N)**: This parameter determines the number of filters or convolutional kernels in each convolutional layer of the SCNN. It controls the depth of feature extraction.

2. **Filter Size (F):** The filter size specifies the dimensions of the convolutional kernels. It influences the spatial extent over which the SCNN extracts features from the input data.

3. **Stride (S)**: The stride determines how the convolutional kernels move across the input data. It affects the spatial downsampling of feature maps.

4. **Number of Neurons in Hidden Layer (H)**: The number of neurons in the hidden layer of the SCNN, often present before the output layer, can significantly impact the network's capacity and representational power.

During the PSO optimization process, we will update these hyperparameters (N, F, S, H) for each particle in the swarm. The position and velocity updates of particles are used to explore and exploit the search space of these hyperparameters. Particles explore the space to find promising hyperparameter combinations and converge towards optimal solutions based on their individual and global best positions.

4.2 Encoding Function

In our case a particle represents a potential solution or a set of hyperparameters in the search space. In the context of our problem, a particle represents a specific combination of hyperparameters for the SCNN model. For example, a particle might represent a set of hyperparameters like (N, F, S, H), where:

- N: Number of filters in the convolutional layers.
- F: Filter size in the convolutional layers.
- S: Stride in the convolutional layers.
- H: Number of neurons in the hidden layer of the fully connected layer.

So, we represent a particle by an array or vector structure in which each element corresponds to one of the design variables (N, F, S, H) for SCNN.

4.3 Objective Function

The objective function, denoted as "f(N, F, S, H)," represents the mathematical expression that measures the classification accuracy of the groundwater quality classification model based on the given hyperparameters for the Spatial Convolutional Neural Network (SCNN).

The objective function quantifies how well the SCNN, with a specific set of hyperparameters (N, F, S, H), is performing on the dataset. we have a multi-class classification problem, one commonly used objective function is the categorical cross-entropy loss.

The categorical cross-entropy loss measures the dissimilarity between the true class labels and the predicted class probabilities for all classes. Here's the mathematical representation of the categorical cross-entropy loss:

$$f(N,F,S,H) = \frac{-1}{m}\sum_{i=1}^{m}\sum_{c=1}^{4} y_{i,c}\cdot\log(SCNN(N,F,S,H)_{i,c}$$

Where:

- m is the number of data samples.
- $y_{i,c}$ is an indicator that is 1 if the true label of the ith sample is in class c (e.g., 1 for "Excellent" if the true label is "Excellent"), and 0 otherwise.
- $SCNN(N,F,S,H)_{i,c}$: use the $SCNN_{i,c}$ model trained using **(N,F,S,H)** hyperparameters to predict the correspondent labels of the sample ith in dataset.

The objective function calculates the average cross-entropy loss over all training data samples and all classes. The PSO algorithm aims to find the hyperparameters (N, F, S, H) that minimize this objective function, leading to the highest classification accuracy for the four water quality categories.

4.4 Fitness Function

- The fitness function is synonymous with the objective function, as it quantifies the quality of a solution (particle) in the optimization process.
- It is the same as the categorical cross-entropy loss.

4.5 inertia weight (w)

To calculate the inertia weight (w) in Particle Swarm Optimization (PSO), we can use a formula that balances exploration and exploitation. The choice of constants in this formula plays a crucial role in determining the algorithm's behavior. The most common method to calculate w is as follows:

$$w = \frac{w_{\text{max}} - w_{\text{min}}}{\text{MaxIter}} \times (\text{MaxIter} - \text{CurrentIteration}) + w_{\text{min}}$$

where :

- **w_max**: This is the maximum inertia weight. It determines the balance between exploration and exploitation. A larger value of **w_max** emphasizes exploration, allowing particles to move faster and explore a larger search space. However, too much exploration may hinder convergence. Common values for **w_max** are in the range of [0.9, 0.5].
- **w_min**: This is the minimum inertia weight. It ensures that as the optimization progresses (i.e., as the number of iterations increases), particles focus more on exploitation. Smaller values of **w_min** encourage particles to converge towards the global best solution. Common values for **w_min** are in the range of [0.4, 0.1].
- **MaxIter**: This is the total number of iterations or generations the PSO algorithm will run for.
- **CurrentIteration**: This is the current iteration number within the range [1, **MaxIter**].

The formula gradually decreases the inertia weight from **w_max** to **w_min** as the algorithm progresses through the iterations. This allows PSO to start with more exploration and gradually shift towards exploitation as it converges towards optimal solutions.

4.6 Initialize a swarm of particles & The iterative optimization

Initializing a swarm of particles in Particle Swarm Optimization (PSO) typically involves randomizing their positions and velocities within the defined search space. Here's how we can initialize a swarm of particles for our problem, including the mathematical formulas for position and velocity initialization:

1. **Define PSO Parameters**: Before initializing the swarm, we need to specify some PSO parameters, such as :
   - the number of particles (P),
   - maximum iterations (MaxIter),
   - inertia weight (w),
   - cognitive coefficient (c1), and social coefficient (c2).

   These parameters govern the behavior of the PSO algorithm.

2. **Initialize Positions and Velocities**: For each particle $i = 1, 2, \ldots, P$, do the following:

   (a) ***Position Initialization***: Initialize the position of the particle randomly within the defined search space for each design variable (N, F, S, H).

   $$x_i^{(0)} = [N_i^{(0)}, F_i^{(0)}, S_i^{(0)}, H_i^{(0)}]$$

   Where $x_i^{(0)}$ represents the initial position of particle i, and $N_i^{(0)}, F_i^{(0)}, S_i^{(0)}, H_i^{(0)}$ are the initial values for each design variable. Make sure these values are within their respective search space ranges.

   (b) ***Velocity Initialization***: Initialize the velocity of the particle with random

values within a specified range. You can use a range like [-1, 1] for each design variable.

$$v_i^{(0)} = [V_{N_i}^{(0)}, V_{F_i}^{(0)}, V_{S_i}^{(0)}, V_{H_i}^{(0)}]$$

Where $v_i^{(0)}$ represents the initial velocity of particle i, and $V_{N_i}^{(0)}, V_{F_i}^{(0)}, V_{S_i}^{(0)}, V_{H_i}^{(0)}$ are the initial values for each design variable's velocity.

3. **Iterative Optimization**: The PSO algorithm then proceeds with iterative optimization, where particles' positions and velocities are updated based on their personal best (pbest) and global best (gbest) positions, and the objective function's value is calculated for each particle.

   The iterative optimization process in Particle Swarm Optimization (PSO) involves updating the positions and velocities of particles to find the optimal solution. In our case, the goal is to find the best set of hyperparameters (N, F, S, H) for our Spatial Convolutional Neural Network (SCNN) to maximize the classification accuracy for groundwater quality categories. Here's how the iterative optimization works:

   (a) **Evaluate Fitness**: For each particle i, evaluate its fitness (objective function) by configuring the SCNN with the hyperparameters in its current position:

   $$\text{Fitness}_i^{(t)} = f(N_i^{(t)}, F_i^{(t)}, S_i^{(t)}, H_i^{(t)})$$

   Where:

   - t is the current iteration.
   - i represents the particle index.
   - $N_i^{(t)}, F_i^{(t)}, S_i^{(t)}, H_i^{(t)}$ are the hyperparameters for particle i at iteration t.

   (b) **Update Personal Best (pbest)**: For each particle i, compare its fitness to its personal best fitness ($pbest_i$). If the fitness of the current position is better than the previous personal best, update $pbest_i$ and $pbest_i^{(t)}$ with the current position and fitness.

   $$\text{If Fitness}_i^{(t)} < pbest_i^{(t)}, \text{ then}$$

   $$pbest_i^{(t)} = \text{Fitness}_i^{(t)}$$

   $$pbest_i = [N_i^{(t)}, F_i^{(t)}, S_i^{(t)}, H_i^{(t)}]$$

   (c) **Update Global Best (gbest):** Determine the particle with the best fitness among all particles in the current iteration t. If this particle has a better fitness than the current global best fitness ($gbest^{(t)}$), update $gbest^{(t)}$ with the fitness and position of the best particle.

   $$\text{If } \min(\text{Fitness}_i^{(t)}) < gbest^{(t)}, \text{ then}$$

   $$gbest^{(t)} = \min(\text{Fitness}_i^{(t)})$$

   $$gbest = [N_i^{(t)}, F_i^{(t)}, S_i^{(t)}, H_i^{(t)}]$$

   (d) **Update Velocities and Positions**: Update the velocity and position of each particle for the next iteration $(t + 1)$ using the following formulas:

   $$V_{N_i}^{(t+1)} = w \cdot V_{N_i}^{(t)} + c1 \cdot r1 \cdot (pbest_{N_i}^{(t)} - N_i^{(t)}) + c2 \cdot r2 \cdot (gbest_N^{(t)} - N_i^{(t)}) \quad (1)$$

   $$N_i^{(t+1)} = N_i^{(t)} + V_{N_i}^{(t+1)}$$

   (Repeat the same update process for all design variables: F, S, H)

   Where:

   - $V_{N_i}^{(t+1)}$ is the updated velocity for the x hyperparameter of particle i at iteration $t + 1$.
   - w is the inertia weight.
   - $c1$ and $c2$ are the cognitive and social coefficients, respectively.
   - $r1$ and $r2$ are random values between 0 and 1.

   (e) **Convergence Check**:

- If Iteration $>$ ConsecutiveIterations (example 20):
  - Calculate Validation Loss for $gbest$:
    - Train $SCNN$ on $X_{\text{train}}, y_{\text{train}}$ using (**gbest = [N,F,S,H]**)
    - ValidationLoss = Fitness$_{gbest}$ using $SCNN_{gbest}$ on $X_{\text{validation}}, y_{\text{validation}}$
- If $(ValidationLoss$ doesn't improve by more than $ValidationLossThreshold$ over $ConsecutiveIterations$ iterations): **Terminate PSO**

4. **Termination**: Terminate the algorithm when the convergence criteria are met or the **(maxIter)** is reached.

This process is repeated iteratively until the PSO algorithm converges to a solution or reaches the maximum number of iterations. The PSO algorithm guides particles to explore and exploit the search space to find the optimal set of hyperparameters that maximize the classification accuracy for groundwater quality categories.