Weight Optimization for a Neural Network using Particle Swarm Optimization (PSO)

Stefanie Peters

October 27, 2006
Prof. Dr.-Ing. Andreas König
Neurocomputing

Prof. Dr.-Ing. Andreas König
Institute of Integrated Sensor Systems
Dept. of Electrical Engineering and Information Technology
University of Kaiserslautern

Fall Semester 2006
What did we learn?

Back Propagation

Digital Neural Network Hardware

Analog Neural Network Hardware

Neural Network Visualization

Technical Real World Problems
Neurocomputing Project

Weight Optimization for a Neural Network using Particle Swarm Optimization (PSO)

Silimann Evaluation Board

implementer of offline training of the weights for the Silmann neural network

// Initialize from File
COptimizationPSO pso(4, 3, 75, 40);
pso.LoadTrainingData("Iris_train.txt");
pso.SetActivityFunctionInput("linear");
pso.SetActivityFunctionHiddenNeurons("tanhyp");
pso.SetActivityFunctionOutput("saltus", 0);

Silimann Trainer Software

Neurocomputer

Silimann Trainer 2.3

SAMPLE PATTERNS
PERFORM NEW TRAINING
RUN PATTERN TEST
RUN TASK MANAGER
BOARD TEST
EXIT PROGRAM

Presenter: Stefanie Peters
Prof. Dr.-Ing. Andreas König
Silimann 120cx evaluation board

- Neurocomputer
- Max: 10-6-10 network
  => (10 input neurons, 6 hidden neurons and 10 output neurons)
- Feed forward neural network
- Output:

\[ O = \left[ I^T \cdot W_1^T \right] \cdot W_2^T \]
Particle Swarm Optimization (PSO)

- Population based search algorithm
- Developed to simulate the behavior of birds in search for food on a cornfield or fish school
- Evolutionary Computation Technique
- Each individual (here: particle) has a randomly initialized position $X$ and velocity $V$ in the search space
Depending on
• its actual position and velocity (1)
• its own previous best position (2)
• and the previous best position of a particle in a defined neighborhood (e.g. the complete swarm) (2)
the new velocity (3) and position (4) in the search space is found.
Particle Swarm Optimization (PSO)

1. Actual position and velocity
2. Local and global best position
3. New velocity
4. New position

Particles 🌟 🌟 🌟 Velocity ➔ Previous best position 🌟 Global best position 🌟
**Particle Swarm Optimization (PSO)**

\[ V_{t+1}^i = V_t^i + C_1 R_1 \cdot (XLB_t^{i'} - X_t^i) + C_2 R_2 \cdot (XGB_t^n - X_t^i) \]

\[ X_{t+1}^i = X_t^i + V_{t+1}^i \]

V: velocity of particle \( i \) at time \( t/t+1 \),
X: position in the search space
XLB: best previous position of the particle \( i \) with \( t' < t \) (X Local Best)
XGB: best previous position of a particle in the \( n \)-neighborhood (e.g., the complete swarm) of the particle. (X Global Best)

\( C_1 \) and \( C_2 \) are positive constants, usually in the range \([1,4]\)

\( R_1 \) and \( R_2 \) are two random functions taking values in the range \([0,1]\).
Software Implementation PSO

Hardware restrains:

• Max: 10-6-10 network
  => 10 input neurons, 6 hidden neurons and 10 output neurons
  => max. \((10+1)\times6 + (6+1)\times10 = 136\) weights (= parameters) for the PSO
  => Those network parameters can be easily changed in the program

• Input (data) and output files (basically: weights of the NN) of the software must be compatible to the “Silimann Trainer” files.
  • Those values must lie in the range \([-1,1]\).
Software Implementation PSO

Software design:
- Implementation of a standard PSO
- The number of features, classes and samples must be known (Constructor) before loading any data from a file.
- Adjustable PSO Parameters:
  - Swarm size (Constructor),
  - Constants $C_1$ and $C_2$ (Constructor),
  - Maximum/minimum velocity
  - Boundaries of the search space

```cpp
COptimizationPSO pso( nNoInputParam , nNoClasses, nNoTrainSamples , nNoParticles, fConst1,
                      fConst2);
```  
Constructor call for a pso run.
Note: each parameter in the constructor has a default value. COptimizationPSO pso( ) will call pso( 10, 10, 1, 20, 2, 2)

```cpp
pso.LoadTrainingData(„iris_train.txt“)
```
Loading training data.

```cpp
pso.SetVRange(0.6);
```
Set minimum/maximum velocity.

```cpp
pso.SetXMinMax (-1, 1);
```
Set boundaries of the search space.
• Each layer of the NN has its own activity function (e.g. to limit the output to a defined range).
  • **Linear**: no restriction of the NN outputs
  • **Tanhyp**: hyperbolic tangent, -> restriction to values in the range \([-1, 1]\).
  • **Saltus**: step function, global threshold s, -> restriction to values \([-1, 1]\).
  • More activity functions can be easily implemented.

```csharp
psosetActivityFunctionInput(“saltus”, s);
```
Set activity function input layer.

```csharp
psosetActivityFunctionHiddenNeuron(“tanhyp”);
```
Set activity function hidden neurons.

```csharp
psosetActivityFunctionOutput(“linear”);
```
Set activity function output layer.
Software Implementation PSO

- Evaluation of the fitness of a swarm particle

- **PMSE**: percentage of mean square error. PMSE of all N neural network outputs \( o_i \) is calculated for all P pattern samples (\( t_{pi} \): desired result).

\[
PMSE = 100 \times \frac{1}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2
\]

PMSE for fitness evaluation.

- **EC**: Sum of classification errors \( EC_p \) for all pattern samples. If the maximum neural network output \( \max[o_{pi}] \) corresponds to \( t_{pi} = 1 \) \( \rightarrow EC_p = 0 \), else \( EC_p = 1 \).
Software Implementation PSO

• Evaluation of the fitness of a swarm particle

  • **EC2**: like EC, but the minimum difference $\text{mdiff} = \max[O_{pi}] - O_{pj}$ ($j \neq i$) must be greater than a predefined threshold to generate an error $\text{EC}_p = 0$.

  • Further fitness computation functions are implemented as well. See the source code for more information.

```
psosetfitnessfunction("PMSE");
```

Set fitness function.

Note: At the current state, only one of these parameters (PMSE, EC, ...) can be applied to optimize the weights of the neural network (the other parameters are computed for each particle and displayed during the optimization as well). The PSO minimizes the selected fitness parameter.
Further important Functions

- **InitializePSO( Key ):** Sets the seed point for all used random numbers. (Key == 0: the system time is used as key).

Call these function for every generation:

- **ComputeFitness():** Computes the fitness for all particles.
- **UpdateVelocity():** Computes and sets the new velocity and position for all particles.

Output of the fitness parameters during the optimization.
Multiple debugging options increase the comprehensibility of the software

- For example: Output of the hidden and output neurons (to screen or to file)
**Iris Data**

**Iris Plants Database**
- The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.
  - 4 Parameters: sepal length and width, petal length and width
- => 4 – 6 – 3 Neural Network
- => Training set: 25 instances of each class
- => Test set: 25 different instances of each class
- Sources:
  (a) Creator: R.A. Fisher
  (b) Donor: Michael Marshall
  (c) Date: July, 1988

*Evolutionary Computation Technique*

4 – 6 – 3 Feed forward Neural Network
A Resulting Net-File weights:
(Optimization: 40 Particles, PMSE, hyptan-Activity for hidden and output neurons, 
\(C_1, C_2 = 2\))
Results Optimization Offline:
Training 75/75 pattern, Test: 73/75 pattern correctly classified.

\[
\begin{align*}
W_1_{\text{TRAIN}} & = 0.1361;0.5477;0.3449;0.0754;-0.2446 \\
& -0.2446;0.3370;-0.8200;0.9504;0.2757 \\
& 0.2757;0.4560;-0.4385;-0.1748;-0.0970 \\
& -0.0970;0.4672;0.3002;0.0073;0.8531 \\
& 0.8531;0.5733;-0.5808;-0.0668;0.1156 \\
& 0.1156;0.9243;-0.1257;-0.9057;0.8236 \\
W_2_{\text{TRAIN}} & = 0.1290;-0.7530;0.1069;0.9800;0.3467;-0.7851;0.9652 \\
& -0.8735;-0.0558;-0.3678;0.2691;0.3834;0.5740;-0.2593 \\
& 0.1578;-0.3466;0.2025;0.1710;-0.7851;0.9503;-0.0766
\end{align*}
\]
Results: Iris Data 1

- Uploading of the trained weights to the Silimann Evaluation Board and testing of Training and Test pattern samples:

75 of 75 training patterns are correctly classified by the Silimann evaluation board.
Results: Iris Data 1

- Uploading of the trained weights to the Silimann Evaluation Board and testing of Training and Test pattern samples:

75 of 75 training patterns are correctly classified by the Silimann evaluation board.
Results: Iris Data 1

- Uploading of the trained weights to the Silimann Evaluation Board and testing of Training and Test pattern samples:

71 of 75 test patterns are correctly classified by the Silimann evaluation board
Questions