Einführung in die Neuroinformatik

Tim Luchterhand, Paul Nykiel (Gruppe P)

23. Mai 2018

1 Backpropagation [Pen and Paper]

1. Forwärts propagieren:

$$\begin{aligned} u_1^{(1)} &= x_1 \cdot w_{11}^{(1)} + x_2 \cdot w_{21}^{(1)} + b_1^{(1)} \\ y_1^{(1)} &= f_1(u_1^{(1)}) \\ u_1^{(2)} &= y_1^{(1)} \cdot w_{11}^{(2)} + y_2^{(1)} \cdot w_{21}^{(2)} + b_1^{(2)} \\ y_1^{(2)} &= f_2(u_1^{(2)}) \end{aligned}$$

2. Fehler in der Ausgabeschicht bestimmen:

$$\delta_1^{(2)} = y_1^{(2)} - T_1$$

3. Backpropagation

$$\delta_1^{(1)} = \delta_1^{(2)} w_{11}^{(2)} f' \left(u_i^{(1)} \right)$$

4. Gewichte adaptieren

$$\tilde{w_{11}^{(1)}} = w_{11}^{(1)} + \eta x_1 \delta_1^{(1)}$$

2 Backpropagation [Matlab]

2.1

```
function [weights] = initWeights(inputDimensions, hiddenNeurons, outputDimensions)
%initWeights initializes the weights of the network
%   Arguments:
%       - inputDimensions: number of input neurons
%       - hiddenNeurons: number of hidden neurons
%       - outputDimensyions: number of output neurons
%
%   Returns:
%       - weights: struct with the parameters w1, w2, theta1 and theta2
%
    %rng(1337, 'combRecursive');
    weights.w1 = rand(hiddenNeurons,inputDimensions) .- 0.5;
    weights.w2 = rand(outputDimensions, hiddenNeurons) .- 0.5;
    weights.theta1 = rand(hiddenNeurons,1) .-0.5;
    weights.theta2 = rand(outputDimensions,1) .-0.5;
end
```

2.2

```
function [y2, u2, y1, u1] = forward(inputX, weights, trans)
%forward calculates the network output
%   Arguments:
%       - inputX: input data organized as samples x dimensions (each row denotes a point)
%       - weights: struct with the parameters w1, w2, theta1 and theta2
%       - trans: activation function f(x) of the hidden layer
%
    u1 = weights.w1 * inputX + weights.theta1;
    y1 = trans(u1);
    u2 = weights.w2 * y1 + weights.theta2;
    y2 = u2;
end
```

2.3

```
function [delta1, delta2] = propagateError(T, y2, w2, u1Diff)
%propagateError calculates the error of the network (delta1 and delta2)
%   Arguments:
%       - T: teacher signal
%       - y2: output of the last neuron
```

```
%           - w2: weights matrix of the second layer
%           - u1Diff: f'(u1)
%
    delta2 = norm(T-y2)^2;
    delta1 = delta2 * (transpose(w2) .* u1Diff);
end
```

2.4

```
function [weights, errors] = train(hiddenNeurons, learnRate, inputX, outputT, epochs, trans, transDiff)
%train trains the neural network
%   Arguments:
%           - hiddenNeurons: number of hidden neurons
%           - learnRate: learning rate \eta
%           - inputX: input data organized as samples x dimensions (each row denotes a point)
%           - outputT: teacher signal as column vector
%           - epochs: number of epochs to train the network
%           - trans: transfer function to use in the hidden layer (activation function)
%           - transDiff: derivative of the transfer function
%
    assert(iscolumn(outputT), 'T must be a column vector');
    assert(size(inputX, 1) == size(outputT, 1), 'Each data point must have an associated label');
    %rng(1337, 'combRecursive'); % For reproducibility (does also work with parfor: http://de.mathworks.com/help/distcomp/control-random-number-streams.html#btms9o_)
    weights = initWeights(size(inputX,2),hiddenNeurons,size(outputT,2));
    E = 0;
    for e=1:epochs
        indexSet = randperm(size(inputX,1));
        for c=indexSet
            currentInput = transpose(inputX(c,:));
            trainerOutput = transpose(outputT(c,:));
            [mlpOutput,u2,hiddenOutput,u1] = forward(currentInput, weights,trans);
            [delta1,delta2] = propagateError(trainerOutput, mlpOutput,weights.w2,transDiff(u1));

            weights.w1 += learnRate * (delta1 * transpose(currentInput));
```

```
            weights.w2 += learnRate * delta2 * transpose(hiddenOutput);
            weights.theta1 -= learnRate * delta1;
            weights.theta2 -= learnRate * delta2;
        end;

        E = 0;
        %Fehler berechnen
        for c=1:size(inputX,1)
            currentInput = transpose(inputX(c,:));
            trainerOutput = transpose(outputT(c,:));
            [mlpOutput,u2,hiddenOutput,u1] = forward(currentInput, weights,trans);
            E += norm(mlpOutput-trainerOutput)^2;
        end;
        E
    end;
end
```