Übung zur Vorlesung

Maschinelles Lernen 1

Wintersemester 2010/2011

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und
theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Blatt 12

Abgabe bis Mittwoch, den 2. Februar 2011 um 12:00

Die Lösungen bitte im **Postfach** von Dr. Konrad Rieck abgeben und
elektronisch an `konrad+lehre@mlsec.org` senden.

Neuronale Netze und Backpropagation

In dieser Übung soll der Backpropagation-Algorithmus implementiert werden, um ein neuronales Netz zu trainieren. Das Netz soll eine Eingabeschicht mit n Neuronen, eine versteckte Schicht mit k Neuronen und eine Ausgabeschicht m Neuronen besitzen. Die Gewichte des Netzes sollen durch einen stochastischen Gradientenabstieg optimiert werden (siehe Programmskelett).

Aufgaben

1. Implementiere die Funktion `net_init`, die das Netz initialisiert (6 Punkte). Die Gewichte sollten zufällig aus einer uniformen Verteilung mit $\mu = 0$ und $\sigma = m^{-1/2}$ gezogen werden, wobei m die Anzahl der Eingänge jedes Neurons in einer Schicht ist.

2. Implementiere eine Funktion `net_forward`, die einen Eingabevektor $x \in \mathbb{R}^n$ vorwärts durch das Netz propagiert (6 Punkte). Verwende als Schwellwertfunktion

$$s(x) = \tanh(x).$$

3. Implementiere eine Funktion `net_backward`, die den Fehler einer Vorhersage rückwärts durch das Netz propagiert und partielle Ableitungen berechnet (6 Punkte). Nutze hierfür die Ableitung der Schwellwertfunktion

$$s'(x) = 1 - s(x)^2.$$

4. Implementiere eine Funktion `net_update`, die die Gewichte des Netzes anpasst (6 Punkte). Die Anpassung wird über die Lernrate η gesteuert.

5. Teste Deine Implementierung mit dem XOR-Problem (6 Punkte). Untersuche den Einfluß von Lernrate und Anzahl von versteckten Neuronen auf das Lernen. Was passiert, wenn die Lernrate zu hoch oder zu klein ist?

Tipps

Die Funktionen sollen eine Matlab-Struktur `N` nutzen, um das neuronale Netz zu repräsentieren und Zwischenergebnisse zu speichern. Es ist hilfreich, wenn Ihr folgende Felder verwendet

- `N.W1` Gewichtsmatrix zwischen Eingabeschicht und versteckter Schicht mit $n \times k$.
- `N.W2` Gewichtsmatrix zwischen versteckter Schicht und Ausgabeschicht mit $k \times m$.
- `N.Y0` Eingabevektor mit n Einträgen.
- `N.Y1` Ausgabe des Netzes nach der versteckten Schicht mit k Einträgen.
- `N.Y2` Ausgabe des Netzes nach der Ausgabeschicht mit m Einträgen.
- `N.D1` Partielle Ableitung an der versteckten Schicht mit k Einträgen.
- `N.D2` Partielle Ableitung an der Ausgabeschicht mit m Einträgen.

```
function sheet12

% Learning rate
rate = 0.05;

% Iterations of learning
iterations = 1000;

% Create XOR problem (2 dimensions + 1 bias dimension)
[X,Y] = create_data();

% Init network with 3 input, 5 hidden and 1 output neuron
N = net_init(3, 5, 1);

% Stochastic learning
for i = 1:iterations
    % Select random point
    j = ceil(rand() * length(Y));

    % Forward/backward pass
    N = net_forward(N, X(:,j));
    N = net_backward(N, Y(j));
    N = net_update(N, rate);

    % Store error at point j
    e(i) = mean((N.Y2 - Y(j)).^2);
end

clf; plot(e);
ylabel('Iterations');
xlabel('Mean squared error');

function [X,Y] = create_data()
X = [ 1 1; -1 -1; 1 -1; -1 1]';
Y = [ 1  1  -1 -1];
X = [X ; ones(size(Y))];

%%
% Initialize neural network
% Input:
%    n     Number of input neurons
%    k     Number of hidden neurons
%    m     Number of output neurons
% Output:
%    N.W1  Weight matrix (n x k)
%    N.W2  Weight matrix (k x m)
function N = net_init(n,k,m)
...

%%
% Forward pass through network
% Input:
%    N     Network structure
%    x     Input vector (1 x n)
% Output:
%    N.Y0  Input vector (1 x n)
%    N.Y1  Output at hidden layer (1 x k)
%    N.Y2  Output at output layer (1 x m)
function N = net_forward(N, x)
...
```

```
%%
% Backward pass through network
% Input:
%    N     Network structure
%    y     Target vector (1 x m)
% Output:
%    N.D2  Partial derivative at output layer (1 x m)
%    N.D1  Partial derivative at hidden layer (1 x k)
function N = net_backward(N, y)
...


%%
% Update the network weights
% Input:
%    N     Network structure
%    r     Learning rate
% Output:
%    N.W1  Updated weights
%    N.W2  Updated weights
function N = net_update(N, r)
...
```

Für Fragen zum Übungsblatte bitte in der Google Group http://groups.google.com/group/ml-tu registrieren und die Frage an die Mailingliste stellen.