

Mustererkennung: Übungsblatt 9

von

Naja v. Schmude (4127652), Lisa Dohrmann (4130066), Adrian Neumann (4140810)

Aufgabe 1

Bei dieser Aufgabe mussten wir mittels eines neuronalen Netzes Misch- und Unmischmatrizen finden, die ein Signal möglichst so trennen, dass es gut linear mit Werten aus der nahen Vergangenheit approximierbar ist.

Dazu haben wir die im Tutorium vorgestellte Architektur verwendet. Für die Backpropagation benutzen wir den RPROP Algorithmus.

Wie üblich bei Lernverfahren mit neuronalen Netzen gibt es drei Teile.

0.1 Feedforward

Im Feedforward Schritt entmischen wir die Eingangssignale mit einer Matrix und mischen sie darauf wieder. Im einem anderen Teil des Netzes berechnen wir eine Linearkombination k ($=5$) alter Werte der Signale. Desweiteren bestimmen wir den quadratischen Fehler und seine Ableitung.

```

1  Matrix x = mixed.getVector(t);
2  Matrix y = unmix(x);
3  Matrix x_schlange = mix(y);
4
5  //vorläufige Rekonstruktion merken
6  rec.setSignal1Value(t, y.get(0,0));
7  rec.setSignal2Value(t, y.get(1,0));
8
9  //Fehlervektor, erster Teil
10 Matrix error1_deriv = x_schlange.minus(x);
11 Matrix error1 = error1_deriv.arrayTimes(error1_deriv).times(0.5);
12
13 Matrix y_i = new Matrix(k,2);
14
15 //die letzten k Werte der Mischung betrachten
16 for (int l = t-1; l >= t-k; l--) {
17     int i = 0;
18     Matrix y_l = new Matrix(2,1);
19     if (l >= 0) {
20         Matrix x_l = mixed.getVector(l); //2 x 1
21         y_l = unmix(x_l); //2 x 1
22     }
23     //y_l in die Matrix ,bertragen
24     y_i.set(i, 0, y_l.get(0,0));
25     y_i.set(i, 1, y_l.get(1,0));
26     ++i;
27 }
28
29 //h?lt alle 1. Komponenten
30 Matrix y_i1 = y_i.getMatrix(0, y_i.getRowDimension()-1, 0, 0);
31 //h?lt alle 2. Komponenten
32 Matrix y_i2 = y_i.getMatrix(0, y_i.getRowDimension()-1, 1, 1);
33
34 //beide Spalten von Theta holen
35 Matrix theta_i = Theta.getMatrix(0, Theta.getRowDimension()-1, 0, 0); //k
    x 1 erste Spalte
36 Matrix theta_j = Theta.getMatrix(0, Theta.getRowDimension()-1, 1, 1); //k
    x 1 zweite Spalte
37
38 Matrix y_i_schlange = new Matrix(2,1); // 2 x 1

```

```

39 y_i_schlange.set(0, 0, theta_i.transpose().times(y_i1).get(0,0));
40 y_i_schlange.set(1, 0, theta_j.transpose().times(y_i2).get(0,0));
41
42 //das y aus der ersten Gruppe vom Output der zweiten abziehen
43 y_i_schlange.minusEquals(y);
44 //Fehlervektor, zweiter Teil
45 Matrix error2_deriv = y_i_schlange.minus(y);
46 Matrix error2 = error2_deriv.arrayTimes(error2_deriv).times(0.5);
47
48 //Globaler Error
49 double globalError = error1.normF() + error2.normF();
50 System.out.println(globalError);

```

0.2 Ableitungen Berechnen (Backprop)

Um die partiellen Ableitungen nach den einzelnen Komponenten der drei Matrizen zu berechnen, lassen wir das Netz „rückwärts“ laufen.

```

1 //hier die Ableitungen merken
2 Matrix deltaM = new Matrix(2,2);
3 Matrix deltaW = new Matrix(2,2);
4 Matrix deltaTheta = new Matrix(k,2);
5
6 //jetzt tatsächlich die Ableitungen ausrechnen
7
8 //dM (Formel siehe Tutorium)
9 for (int i = 0; i < deltaM.getRowDimension(); i++)
10     for (int j = 0; j < deltaM.getColumnDimension(); j++) {
11         double value = y.get(i,0) * error1_deriv.get(j,0);
12         deltaM.set(i,j, value);
13     }
14
15 //dTheta (Formel siehe Tutorium)
16 for (int i = 0; i < deltaTheta.getRowDimension(); i++)
17     for (int j = 0; j < deltaTheta.getColumnDimension(); j++) {
18         double value = y_i.get(i,j) * error2_deriv.get(j,0);
19         deltaTheta.set(i,j, value);
20     }
21
22 //dW (Formel siehe Tutorium)
23 for (int i = 0; i < deltaW.getRowDimension(); i++)
24     for (int j = 0; j < deltaW.getColumnDimension(); j++) {
25         //Netz oben
26         double sum1 = 0.0;
27         for (int l = 0; l < x.getRowDimension(); l++) {
28             sum1 += M.get(j,l) * error1_deriv.get(l,0) -
29                 error2_deriv.get(j, 0);
30         }
31         //Netz unten
32         double sum2 = 0.0;
33         int p = 0;
34         for (int l = t-1; l >= Math.max(t-k, 0); l--) {
35             sum2 += mixed.getVector(l).get(i,0) * Theta.get(p++,j) *
36                 error2_deriv.get(j,0);
37         }
38         deltaW.set(i, j, x.get(i,0) * (sum1) + sum2);
39     }

```

0.3 Update mit RPROP

Schlußendlich kucken wir in welche Richtung die Ableitung zeigt, vergleichen sie mit dem früheren Wert und laufen je nachdem, ob sich die Richtung geändert hat, schneller oder langsamer in die umgekehrte Gradientenrichtung. Dabei lassen wir die gröÙe des Gradienten außer Acht und betrachten nur das Vorzeichen.

```

1 //Update per RPROP
2 updateWeights(M, deltaM, deltaM_old, gammaM);
3 updateWeights(W, deltaW, deltaW_old, gammaW);

```

```

4 updateWeights(Theta, deltaTheta, deltaTheta_old, gammaTheta);
5
6 deltaM_old = deltaM;
7 deltaW_old = deltaW;
8 deltaTheta_old = deltaTheta;

```

Die Funktion updateWeights ist so definiert:

```

1 /**
2  * RPROP
3  * @param delta
4  * @param deltaOld
5  * @param gamma
6  */
7 private static void updateWeights(Matrix weights, Matrix delta, Matrix
  deltaOld, Matrix gamma) {
8     for (int i = 0; i < delta.getRowDimension(); i++)
9         for (int j = 0; j < delta.getColumnDimension(); j++) {
10             double gammaNew = getNewGamma(delta.get(i,j),
11                 deltaOld.get(i,j), gamma.get(i,j));
12             gamma.set(i,j, gammaNew);
13
14             double temp = -gammaNew * Math.signum(delta.get(i,j));
15             weights.set(i,j, weights.get(i,j) + temp);
16         }
17 }
18
19 private static double getNewGamma(double delta, double deltaOld, double
  gammaOld) {
20     double gammaTemp;
21     if (delta * deltaOld >= 0)
22         gammaTemp = Math.min(gammaOld * UP, GAMMAMAX);
23     else
24         gammaTemp = Math.max(gammaOld * DOWN, GAMMAMIN);
25     return gammaTemp;
26 }

```

Die Lernkonstanten sind so gewählt:

```

1 static final double UP = 1.2;
2 static final double DOWN = 0.8;
3 static final double GAMMAMAX = 1;
4 static final double GAMMAMIN = 1E-6;
5 static final double GAMMASTART = 0.001;

```

0.4 Schicke Bilder

Wir malen die Ausgabe unseres Algorithmus auf. Für einen Testlauf sieht das ungefähr so aus:

