Performance-Evaluation der potenziellen Engpässe der neuronalen Netzimplementierung in SetlX

(network.stlx durch sigmoid_timing.stlx oder zip_timing.stlx ersetzen zum Testen)

1. Aufruf der Funktion `zip(l1, l2)`. Bildung jeweils einer Liste für `l1` und `l2` (durch `toList(v)`) bedeutet zusätzlichen Rechenaufwand.

Codeabschnitte mit Zeitmessungspunkten:

```
sgd := procedure(training_data, epochs, mini_batch_size, eta, test_data) {
    if(test_data != null) {
        n_test := #test_data;
    }
    n := #training_data;

    for(j in {0..epochs}) {
        s1 := now();
        training_data := shuffle(training_data);
        // Get mini-batches from the training data to train the network
        mini_batches := [ training_data[k..k+mini_batch_size-1] : k in [1,mini_batch_size..n] ];

        for(mini_batch in mini_batches) {
            update_mini_batch(mini_batch, eta);
        }
        epoche_time := now() - s1;
        // Visual output
        if(test_data != null) {
            ev := evaluate(test_data);
            //print("Epoch $j$: $ev$ / $n_test$");
            print("Zipping-time:\t" + zip_time);
            print("Epoche-time:\t" + epoche_time);
            print("--> " + 100.0 * zip_time/epoche_time + "%");
            this.zip_time := 0;
        }
        else {
            print("Epoch $j$ complete");
        }
    }
};
```

```
/* Computes the Cartesian product of two matrices or vectors */
zip := procedure(l1, l2) {
    s1 := now();
    res := toList(l1) >< toList(l2);
    this.zip_time += (now() - s1);

    return res;
};
/* casts vector to list */
toList := procedure(v) {
    return [v[i] : i in [1..#v]];
};
```

Anzahl Datensätze: 10.000 Testsätze, 10.000 Trainingssätze
Rechnerdaten: Intel Core i7-4720HQ, 16GB RAM

Ergebnisse 1.:

```
Start SGD
Zipping-time:   7154
Epoche-time:    23136
--> 30.921507607192254%
Zipping-time:   6451
Epoche-time:    19665
--> 32.8044749555047%
Zipping-time:   6006
Epoche-time:    20937
--> 28.68605817452357%
Zipping-time:   6371
Epoche-time:    19349
--> 32.926766241149416%
Zipping-time:   6550
Epoche-time:    20508
--> 31.938755607567778%
Zipping-time:   6229
Epoche-time:    18742
--> 33.23551381922954%
Zipping-time:   6248
Epoche-time:    18913
--> 33.03547824247872%
```

2. `sigmoid_prime(z)` und `sigmoid_vector(z)`

Codeabschnitte mit Zeitmessungspunkten:

```
sgd := procedure(training_data, epochs, mini_batch_size, eta, test_data) {
    if(test_data != null) {
        n_test := #test_data;
    }
    n := #training_data;

    for(j in {0..epochs}) {
        s1 := now();
        training_data := shuffle(training_data);
        // Get mini-batches from the training data to train the network
        mini_batches := [ training_data[k..k+mini_batch_size-1] : k in [1,mini_batch_size..n] ];

        for(mini_batch in mini_batches) {
            update_mini_batch(mini_batch, eta);
        }
        epoche_time := now() - s1;
        // Visual output
        if(test_data != null) {
            ev := evaluate(test_data);
            //print("Epoch $j$: $ev$ / $n_test$");
            print("Sigmoid-time:\t" + sigmoid_time);
            print("Epoche-time:\t" + epoche_time);
            print("--> " + 100.0 * sigmoid_time/epoche_time + "%");
            this.sigmoid_time := 0;
        }
        else {
            print("Epoch $j$ complete");
        }
    }
};
```

```
// Sigmoid function for vectors
// 1.0/(1.0+np.exp(-z))
sigmoid_vector := procedure(z) {
    // z is a vector, so the function has to be used on every part of it
    s1 := now();
    res := la_vector([ 1.0/(1.0 + exp(- z[i] )) : i in [1..#z] ]);
    this.sigmoid_time += (now() - s1);
    return res;
};

// Derivative of the sigmoid function, when z is a vector
// sigmoid(z)*(1-sigmoid(z))
sigmoid_prime := procedure(z) {
    s := sigmoid_vector(z);
    s1 := now();
    res := la_matrix([ [ s[i] * (1 - s[i]) ] : i in [1..#s] ]);
    this.sigmoid_time += (now() - s1);
    return res;
};
```

Ergebnisse 1.:

```
Start SGD
Sigmoid-time:   1423
Epoche-time:    23065
--> 6.169520919141556%
Sigmoid-time:   1391
Epoche-time:    20955
--> 6.63803388212837%
Sigmoid-time:   1328
Epoche-time:    24388
--> 5.445300967689027%
Sigmoid-time:   1220
Epoche-time:    21434
--> 5.691891387515163%
Sigmoid-time:   1583
Epoche-time:    22733
--> 6.963445211806625%
Sigmoid-time:   1337
Epoche-time:    20903
--> 6.396211070181313%
```