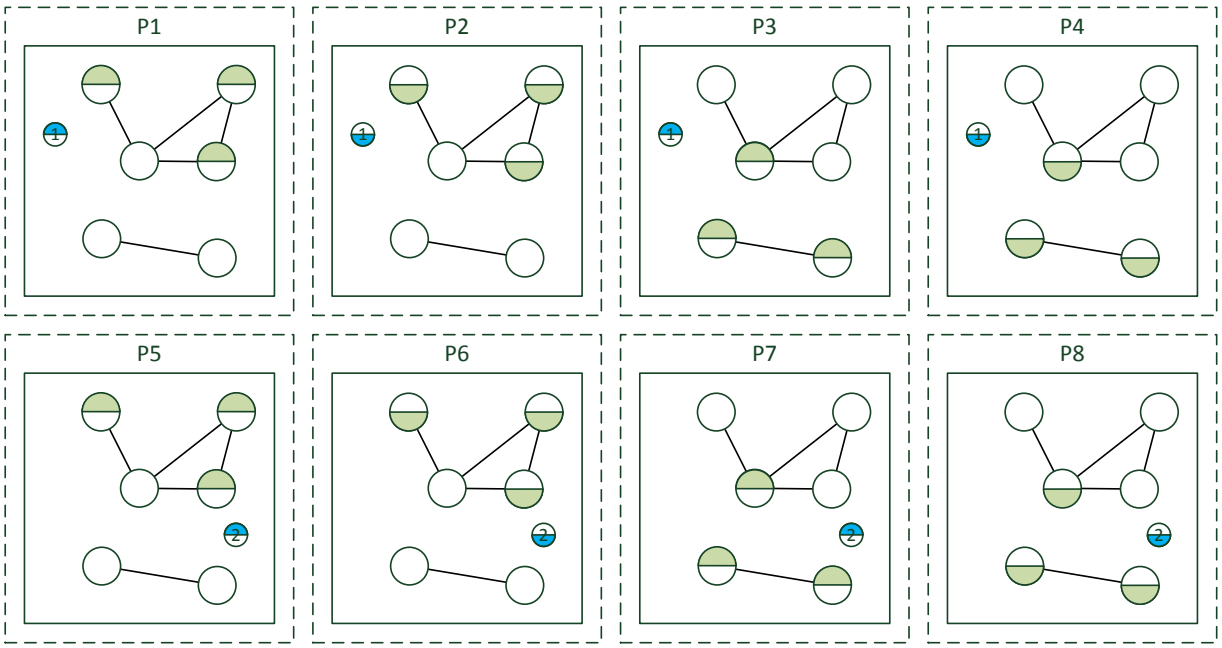
# Dokumentation – paralleles GNG-Training

## Inhalt

# Einleitung

Das Growing-Neural-Gas (GNG) eignet sich gut zum Clustern hochdimensionaler Daten. Dazu wird ein vektorbasiertes neuronales Netz trainiert. Dieses Training ist sehr rechen- und damit zeitintensiv. Um die Laufzeit zu reduzieren, wird versucht die Berechnungen auf mehrere Recheneinheiten aufzuteilen. Diese Anwendung setzt die Verteilung von Daten, Neuronen und Vektorkomponenten praktisch um. Als Recheneinheiten werden dabei Multiprozessorsysteme und CUDA-fähige GPUs unterstützt. Getestet wurde die Anwendung auf Windows (XP und Vista) und Linux.



# Endnutzeranleitung

Es wurden 3 Anwendungen entwickelt. Eine um eine Datenmenge zu clustern, eine um zwei Clusterergebnisse miteinander zu vergleichen und eine um ein Trainingsergebnis zu visualisieren.

Mit der Konsolen-Anwendung „gng“ kann man ein GNG-Netz mit beliebigen Daten trainieren, dabei die Parameter für das Training und die Verteilung beliebig festlegen und erhält neben dem trainierten Netz eine Clusterzuordnung aller Eingabedaten.

Will man 2 Clusterzuordnungen miteinander vergleichen, steht die Konsolen-Anwendung „compare_cluster“ zur Verfügung. Diese gibt aus, wie Eingabedaten eines Clusters in den Clustern des zweiten Clusterergebnisses verteilt sind.

Zur grafischen Darstellungen des trainierten Netzes kann man die Ausgabe von „gng“ mit der Konsolen-Anwendung „net2plot“ in Daten umwandeln, welche als Eingabedaten für eine Plotting-Anwendung wie Gnuplot dienen können.

## Anwendung „gng“ bzw. „gng.exe“

### Aufruf

```
gng [eingabedatendatei [parameterdatei [netzdatei [clusterdatei]]]]
```

- `eingabedatendatei`: Pfad zur Eingabedatendatei (Eingabe). Standard: input.csv
- `parameterdatei`: Pfad zur Eingabedatendatei (Eingabe). Standard: settings.ini
- `netzdatei`: Pfad zur Netzdatei (Ausgabe). Standard: output.csv
- `clusterdatei`: Pfad zur Clusterdatei (Ausgabe). Standard: cluster.csv

### Format der Eingabedatendatei

Die Eingabedaten müssen als Vektoren im CSV-Format vorliegen. Als Trennzeichen zwischen den Datensätzen dient der Zeilenumbruch, die Vektorkomponenten trennen Semikolons. Als Dezimaltrennzeichen dient der Punkt. Anführungszeichen sind innerhalb der CSV-Daten nicht zulässig. Weiterhin wird davon ausgegangen, dass alle Zeilen die gleiche Anzahl an Vektorkomponenten enthalten. Zeilen mit weniger Komponenten werden nicht importiert, gibt es in einer Zeile mehr Komponenten als vorgesehen, werden die zusätzlichen Einträge ignoriert. Die Mindestanzahl an Vektoren ist drei, die Höchstanzahl ist durch den verfügbaren Arbeitsspeicher begrenzt. Die Anzahl an Dimensionen ist nicht begrenzt, jedoch können im GPU-Modus „`shared`“ maximal 96 Dimensionen verwendet werden.

Beispiel einer CSV-Eingabedatendatei:

```
2.345;1.231;4.454
3.221;7.987;1.342
2.123;3.547;6.435
```

### Format der Parameterdatei

In der Parameterdatei können beliebig viele Trainingseinstellungen getroffen werden. Die Parameter sind durch Zeilenumbrüche getrennt. Ein Parameter besteht aus einem Parametername, einem Gleichheitszeichen und dem Wert. Leerzeichen vor, zwischen und nach den Elementen sind nicht zulässig. Wird ein Trainingsparameter nicht gesetzt, so wird seine Standardeinstellung verwendet. Das Dezimaltrennzeichen ist bei gebrochenen Zahlen der Punkt.

Beispiel einer Parameterdatei:

```
max_number_of_neurons=32
device=gpu
gpu_mode=shared
datapartition_mode=median
p_n=16
p_v=16
p_d=16
k=10000
lambda_ins=10
lambda_del=10
number_of_gpus=1
gpus=1
```

Mögliche Parameter:

| Name | Beschreibung | Standardwert |
|---|---|---|
| p_v | Anzahl Prozesse auf Vektorebene (1-…)[1][2] | 1 |
| p_n | Anzahl Prozesse auf Netzebene (1-…)[1][2] | 1 |
| p_d | Anzahl Prozesse auf Datenebene (1-…) | 1 |
| device | Recheneinheit (cpu \| gpu) | cpu |
| number_of_gpus | Anzahl zu verwendender GPUs (1-32) | 1 |
| gpus | GPU-Nummern (welche GPUs sollen für die Berechnung verwendet werden, Nummern werden durch Komma getrennt) | 0,1,2, … ,30,31 |
| gpu_mode | (shared \|con \| flex) | flex |
| max_number_of_neurons | Maximale Anzahl Neuronen (2-…)[1][3] | 10 |
| alpha | Parameter zum Anpassen der lokalen Fehlervariablen beim Normieren (0.0-1.0) | 0.001 |
| beta | Wert, um den die Kantenzähler erhöht werden sollen (0.0-1.0)[1] | 0.001 |
| beta_min | Mindestwert für die Kantenzähler beim Löschen (0.0-…)[1] | 0.05 |
| epsilon_b | Koeffizient zum Adaptieren des Gewinnerneurins (0.0-1.0)[1] | 0.1 |
| epsilon_n | Koeffizient zum Adaptieren der Nachbarn des Gewinnerneurins (0.0-1.0)[1] | 0.002 |
| epsilon_k | Koeffizient zum Adaptieren der Neuronen beim Batch-Lernen (0.0-1.0) | 0.2 |
| lambda_ins | Einfügeschrittintervallgröße (1-…) | 10 |
| lambda_del | Löschschrittintervallgröße (1-…) | 10 |
| k | Anzahl zu verarbeitender Eingabevektor je Prozess bis zur Synchronisation bei Verteilung auf Datenebene (1-…) | 100 |
| datapartition_mode | (median \| batch \| gng) | median |
| show_cindex | C-Index berechnen und ausgeben (0 \| 1) | 1 (bis 100 EV) |
| show_dindex | Dunn-Index berechnen und ausgeben (0 \| 1) | 1 (bis 1000 EV) |
| show_gkindex | Goodman-Kruskal-Index berechnen und ausgeben (0 \| 1) | 1 (bis 100 EV) |

[1] Im GPU-Modus „shared“ und „con“ können diese Werte nur gesetzt werden, wenn die Anwendung nicht mit den Compiler-Konstanten erstellt wurde (USE_CONSTANTS).

[2] Im GPU-Modus „shared“ können auf Netz- und Vektorebene jeweils maximal 16 Prozesse verwendet werden.

[3] Im GPU-Modus „shared“ ist die maximale Anzahl an Neuronen auf 32 begrenzt.

### Format der Netzausgabedatei

Die Ausgabedatei für das Netz ist im CSV-Format. Die erste Zeile enthält die Überschrift und die folgenden Zeilen jeweils die Daten für ein Neuron. In der ersten Spalte steht die Neuronennummer, die zweite Spalte enthält den Referenzvektor, wobei die einzelnen Komponenten durch Komma getrennt sind. Das Dezimaltrennzeichen ist der Punkt. Die dritte Spalte enthält die Nummern all jener Neuronen mit denen das Neuron durch eine Kante verbunden ist. Die Neuronennummern sind dabei durch Komma getrennt.

Beispiel einer Netzdatei:

```
number;vector;edges
0;1.633179,2.518127,5.910339;1
1;1.036316,4.748627,2.006821;0,2
2;3.185608,3.518127,3.412476;1
```

### Format der Clusterausgabedatei

Die Ausgabedatei für die Clusterzuordnung ist im CSV-Format. Die erste Zeile enthält die Überschrift und die folgenden Zeilen jeweils die Daten für einen Eingabevektor. In der ersten Spalte steht die Eingabevektornummer, die zweite Spalte enthält den Vektor, wobei die einzelnen Komponenten durch Komma getrennt sind. Das Dezimaltrennzeichen ist der Punkt. Die dritte Spalte enthält die Nummer des Clusters zu dem der Vektor gehört.

Beispiel einer Clusterdatei:

```
number;vector;cluster
0;2.345,1.231,4.454;1
1;3.221,7.987,1.342;2
2;2.123,3.547,6.435;1
```

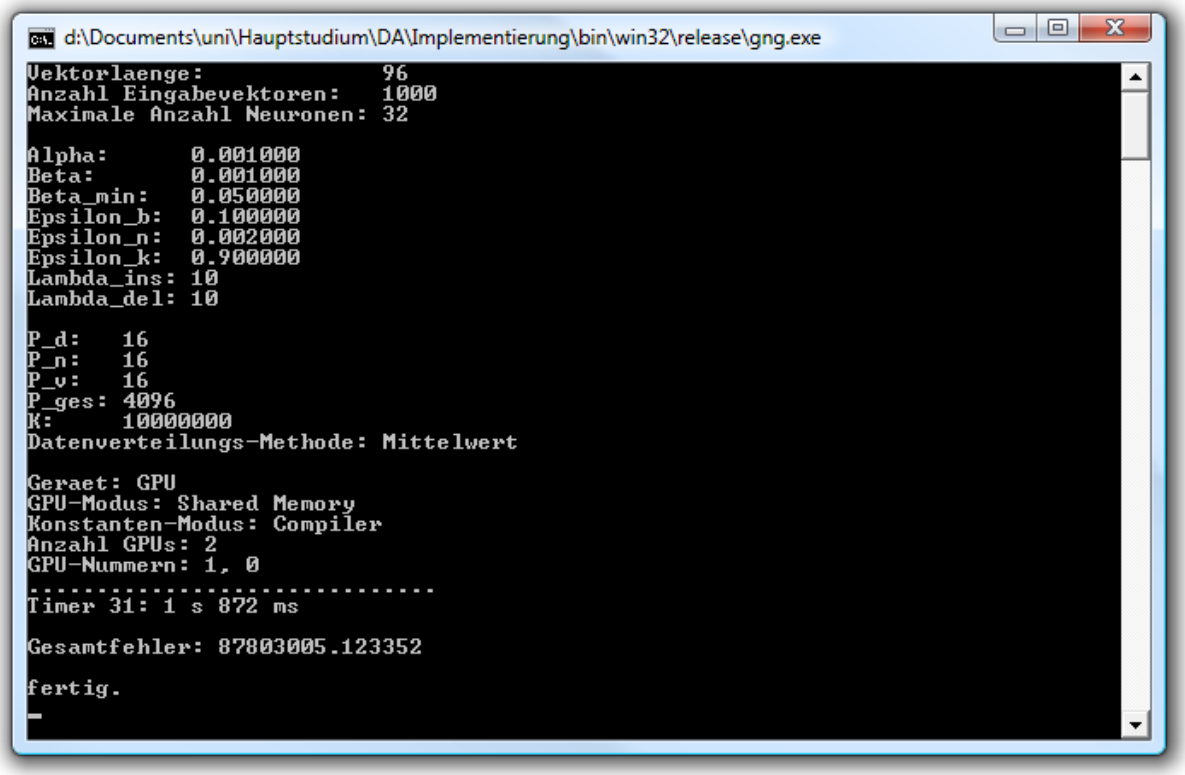
### Parallelisierungsvarianten und Synchronisationsmethoden

**Mittelwertmethode:** Die parallelen Prozesse trainieren wie bei der seriellen Variante ihr Netz. Bei der Synchronisation wird der Mittelwert aller Referenzvektoren, lokalen Fehlervariablen und Kantenzähler der parallelen Netze gebildet und bei allen Netzen eingetragen.

**Batch-Methode:** Während des parallelen Trainings ändern sich die Werte der Netze nicht. Erst nach einer Trainingsepoche bzw. k * p_d Trainingsschritten werden Änderungen mit den Batch-Regeln übernommen. Mithilfe des Parameters epsilon_k kann die Referenzvektoradaption beeinflusst werden.

**GNG-Methode:** Hierbei wird der aktuelle Zustand des Netzes gesichert und die parallelen Prozesse trainieren ihre Netze wie in der seriellen Variante. Bei der Synchronisation wird ein Trainingszyklus auf das gesicherte Netz ausgeführt. Dabei dienen die Referenzvektoren der Neuronen der parallelen Netze als Eingabevektoren. Das Ergebnis dieses Trainings wird dann in die parallelen Netze eingetragen.

### Interpretation der Konsolenausgabe




Neben den Traingsparametern wird die Trainingslaufzeit (Timer 31) und der Gesamtfehler ausgegeben. Der Gesamtfehler ist die Summe der quadratischen Abstände aller Eingabevektoren zu ihren Gewinnerneuronen. Dieser Wert kann als Qualitätskriterium für die Neuronenpositionen dienen. Weiterhin werden evtl. noch diverse Cluster-Indexe ausgegeben. Die Interpretation derselben kann man in der Diplomarbeit nachlesen.

## Anwendung „compare_cluster“ bzw. „compare_cluster.exe“

Diese Anwendung dient dem Vergleich mehrerer Clusterergebnisse miteinander. Hierbei werden 2 Clusterausgabedateien benötigt, die dieselben Eingabevektoren enthalten.

### Aufruf

```
compare_cluster [clusterdatei1 [clusterdatei2]]
```

- clusterdatei1: Pfad zur Clusterausgabedatei 1 (Eingabe). Standard: cluster_org.csv
- clusterdatei2: Pfad zur Clusterausgabedatei 2 (Eingabe). Standard: cluster.csv

### Format der Clusterdateien

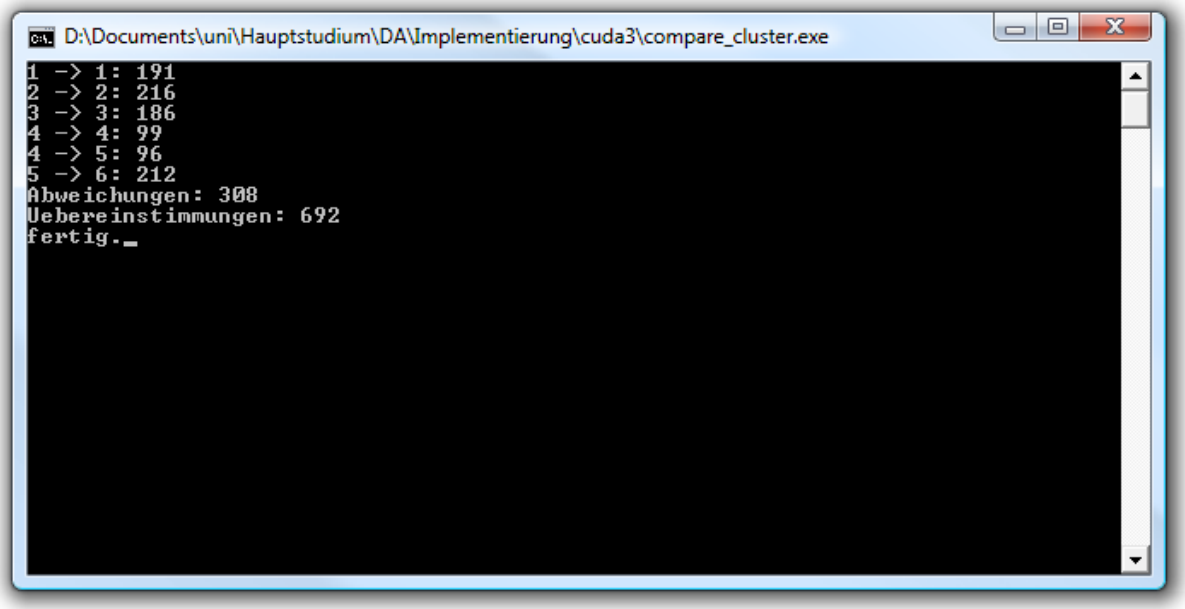
Die Dateien für die Clusterzuordnung sind im CSV-Format. Die erste Zeile enthält die Überschrift und die folgenden Zeilen jeweils die Daten für einen Eingabevektor. In der ersten Spalte steht die Eingabevektornummer, die zweite Spalte enthält den Vektor, wobei die einzelnen Komponenten durch Komma getrennt sind. Das Dezimaltrennzeichen ist der Punkt. Die dritte Spalte enthält die Nummer des Clusters zu dem der Vektor gehört.

Beispiel einer Clusterdatei:

```
number;vector;cluster
0;2.345,1.231,4.454;1
```

```
1;3.221,7.987,1.342;2
2;2.123,3.547,6.435;1
```

### Interpretation der Ausgabe




Die Anwendung gibt auf der Konsole die Clusterbewegungen aus. Diese werden in folgender Form dagestellt:

Ursprungsclusternummer -> Zielclusternummer: Anzahl Eingabevektoren

Beispiel:

```
1 -> 1: 191
2 -> 2: 216
3 -> 3: 186
4 -> 4: 99
4 -> 5: 96
5 -> 6: 212
Abweichungen: 308
Uebereinstimmungen: 692
```

Im obigen Beispiel ist zu erkennen, dass der Cluster Nummer 4 sich in 2 Cluster. Dabei bleiben 99 Vektoren im Cluster 4, 96 wandern in Cluster 5. Cluster 5 erhält lediglich die neue Nummer 6. Die Cluster 1 bis 3 bleiben unverändert.

## Anwendung „net2plot“ bzw. „net2plt.exe“

Diese Anwendung erstellt aus der Trainingsergebnisausgabe (output.csv) und der Datenmenge (input.csv) von „gng“ zwei Datendateien, welche zum Plotten des Netzes geeignet sind. Für Gnuplot stehen hier vorgefertigte Scripte zur Verfügung.

### Aufruf

```
net2plot [traingsergebnis [neuronendatei [kantendatei]]]
```

- trainingsergebnis: Pfad zur Netzausgabedatei (Eingabe). Standard: output.csv
- `neuronendatei`: Pfad zur Neuronendatei (Ausgabe). Standard: neurons.dat

- `kantendatei`: Pfad zur Kantendatei (Ausgabe). Standard: edges.dat

### Format der Netzausgabedatei

Die Ausgabedatei für das Netz ist im CSV-Format. Die erste Zeile enthält die Überschrift und die folgenden Zeilen jeweils die Daten für ein Neuron. In der ersten Spalte steht die Neuronennummer, die zweite Spalte enthält den Referenzvektor, wobei die einzelnen Komponenten durch Komma getrennt sind. Das Dezimaltrennzeichen ist der Punkt. Die dritte Spalte enthält die Nummern all jener Neuronen mit denen das Neuron durch eine Kante verbunden ist. Die Neuronennummern sind dabei durch Komma getrennt.

Beispiel einer Netzdatei:

```
number;vector;edges
0;1.633179,2.518127,5.910339;1
1;1.036316,4.748627,2.006821;0,2
2;3.185608,3.518127,3.412476;1
```

### Format der Neuronendatei

Die Neuronendatei enthält die Koordinaten (bis zu 3) der Neuronenpositionen. Das Dezimaltrennzeichen ist der Punkt und die Koordinaten sind durch Leerzeichen getrennt. Jedes Neuron hat eine Zeile.

Beispiel einer Neuronendatei:

```
1.633179 2.518127 5.910339
1.036316 4.748627 2.006821
3.185608 3.518127 3.412476
```

### Format der Kantendatei

Die Kantendatei enthält die Startkoordinaten (bis zu 3) und Bewegungsrichtungen (auch bis zu 3) der Kanten. Das Dezimaltrennzeichen ist der Punkt und die Koordinaten und Bewegungsrichtungen sind durch Leerzeichen getrennt. Jede Kante hat eine Zeile.

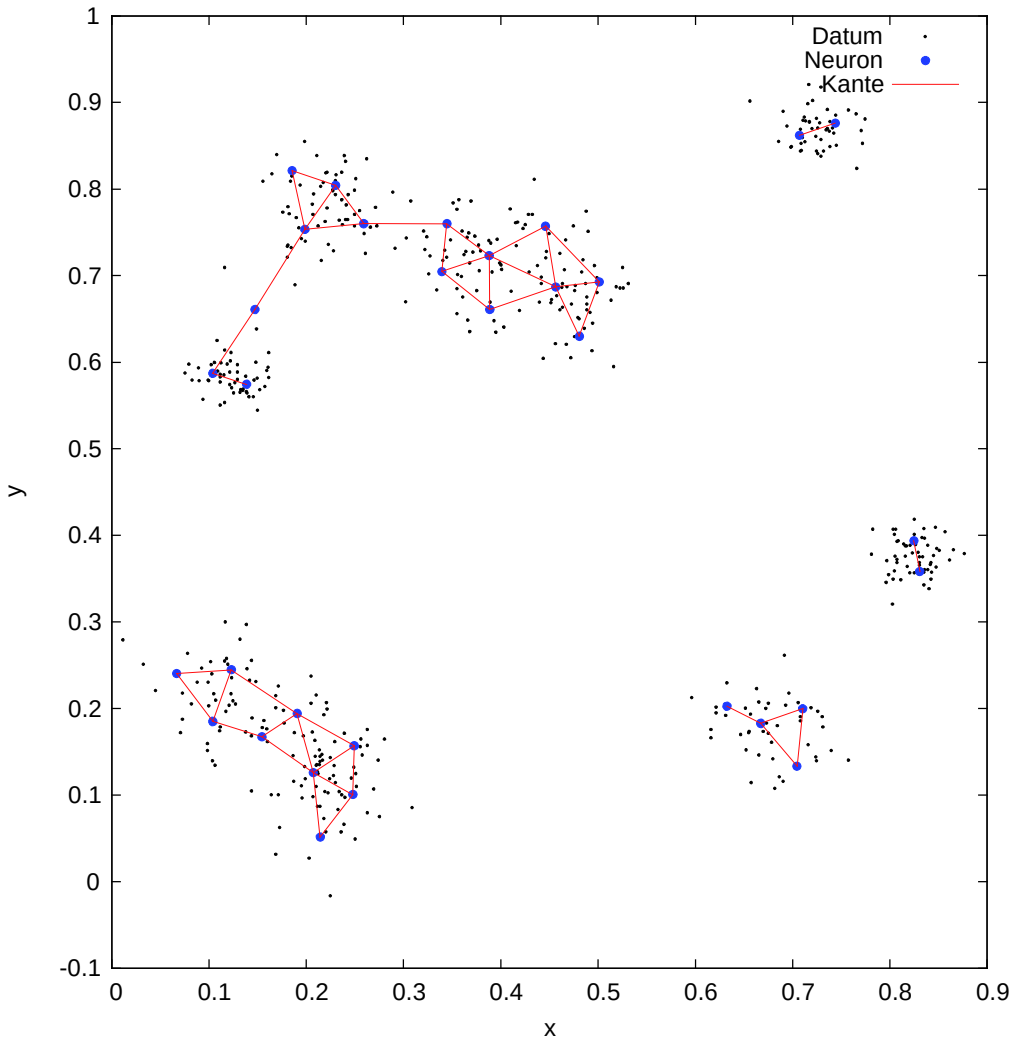
Beispiel einer Kantendatei:

```
71.274597 88.107872 86.533371 61.136368 -15.527916 -6.151215
107.354904 94.327278 82.457748 25.056061 -21.747322 -2.075592
137.196243 88.527069 97.657784 -4.785278 -15.947113 -17.275627
86.468269 105.004890 120.128975 45.942696 -32.424934 -39.746819
75.830505 64.436394 82.508484 56.580460 8.143562 -2.126328
```

### Gnuplot-Scripte

- show_net2D.plt: Erstellt EPS-Datei mit 2D-Ausgabe. Dabei müssen Netz und Eingabendaten ebenfalls zweidimensional sein.
- show_net2D_plus.plt: Erstellt EPS-Datei mit 2D-Ausgabe. Das Netz und die Eingabendaten müssen mehr als 2 Dimensionen haben.
- show_net3D.plt: Erstellt EPS-Datei mit 3D-Ausgabe. Dabei müssen Netz und Eingabendaten ebenfalls dreidimensional sein.
- show_net3D_plus.plt: Erstellt EPS-Datei mit 3D-Ausgabe. Dabei muss das dreidimensional und die Eingabedaten mehr als dreidimensional sein.

## Beispielausgabe



# Entwickleranleitung

## Voraussetzungen und Erstellen

### Windows

Benötigt werden die Pthreads-Bibliotheken für Windows (http://sourceware.org/pthreads-win32/) und die CUDA-Bibliotheken. Kompiliert werden kann die Anwendung u.a. mit Visual Studio 2005.

Getestet unter Windows Vista x64 und Windows XP i386 mit 4 Prozessorkernen und CUDA 2.3. Es kann nur die 32-Bit-Version erstellt werden, da Pthreads für Windows in der Regel kein 64-Bit unterstützt.

### Linux

Benötigte Pakete:

- g++
- libc6-dev
- CUDA-Toolkit
- CUDA-Treiber

Kompilieren:

```
xyz/> make
```

Ausführen:

```
xyz/> ./gng
```

Getestet unter opensuse 11.1 i386 mit 4 Prozessorkernen und CUDA 2.3.

## Dateien

### gng.cpp

Test-Programm zum Nutzen der GNG-Bibliothek.

### gng.sln

Visual Studio 2005 Solution Datei.

### gng.vcproj

Visual Studio 2005 Projekt Datei. Enthält u.a. alle Compiler- und Build-Einstellungen.

### gng_lib.cpp

Datei für Implementierungen der Basis-C++-Funktionen der GNG-Bibliothek.

### gng_lib.h

Header-Datei für Basis-C++-Funktionen der GNG-Bibliothek und diverser Makros und Typen.

### gng_lib_cuda.cu

Datei für Implementierung der Host-CUDA-Funktionen und der gemeinsam genutzten Kernel-Funktionen.

### gng_lib_cuda.h
Header-Datei für C++-kompatible CUDA-Funktionen der GNG-Bibliothek.

### gng_lib_cuda_con.cu
Datei für die Implementierungen der CUDA-Funktionen der GNG-Bibliothek, die für die Parameter Konstanten nutzen (im Speicher oder Compiler).

### gng_lib_cuda_con.h
Header-Datei für CUDA-Funktionen der GNG-Bibliothek, die für die Parameter Konstanten nutzen (im Speicher oder Compiler).

### gng_lib_cuda_flex.cu
Datei für die Implementierungen der CUDA-Funktionen der GNG-Bibliothek, die die maximale Flexibilität bieten.

### gng_lib_cuda_flex.h
Header-Datei für CUDA-Funktionen der GNG-Bibliothek, die die maximale Flexibilität bieten.

### gng_lib_cuda_general.h
Header-Datei für Nicht-C++-kompatible CUDA-Funktionen der GNG-Bibliothek.

### gng_lib_cuda_shared.cu
Datei für Implementierung der CUDA-Funktionen der GNG-Bibliothek, die Shared-Memory nutzen.

### gng_lib_cuda_shared.h
Header-Datei für CUDA-Funktionen der GNG-Bibliothek, die Shared-Memory nutzen.

### gng_lib_func_pthreads.cpp
Datei für Implementierungen der Multithreading-Funktionen der GNG-Bibliothek mit Pthreads.

### gng_lib_func_pthreads.h
Header-Datei für Multithreading-Funktionen der GNG-Bibliothek mit Pthreads.

### gng_settings.h
Header-Datei für Einstellungen, die vor dem Kompilieren bekannt sein müssen.

### makefile
Makefile zum Compilieren und Erstellen der Anwendung unter Linux.

## Datenstrukturen

### Struktur „nettype“
Diese Struktur enthält alle Parameter und Variablen des GNG-Netzes.

| Variable | Beschreibung |
|---|---|
| `int p_d` | Anzahl Prozesse auf Datenebene |
| `int p_n` | Anzahl Prozesse auf Netzebene |
| `int p_v` | Anzahl Prozesse auf Vektorebene |
| `int p_ges` | Gesamtzahl an Prozessen (Threads) |
| `int size_of_vector` | Anzahl Dimensionen gesamt |
| `int number_of_neurons` | Anzahl Neuronen gesamt |
| `int vector_part_size` | Anzahl Dimensionen pro Prozess |

| | |
|---|---|
| `int net_part_size` | Anzahl Neuronen pro Prozess |
| `int max_number_of_neurons` | Maximale Anzahl Neuronen |
| `int max_number_of_vectors` | Anzahl Eingabe-Vektoren |
| `int t` | Zeitparameter |
| `int silent` | Nichts ausgeben |
| `float alpha` | Parameter zur Normierung des lokalen Fehlermaßes |
| `float beta` | Parameter zum Erhöhen des Kantenzählers |
| `float beta_min` | Mindestwert, dass Kanten nicht gelöscht werden |
| `int lambda_ins` | Einfügeintervallgröße |
| `int lambda_del` | Löschintervallgröße |
| `float epsilon_b` | Gewinneradaptionskoeffizient |
| `float epsilon_n` | Nachbaradaptionskoeffizient |
| `float epsilon_k` | Batchadaptionskoeffizient |
| `int k` | Anzahl Vektoren, die auf Datenebene je Prozess verarbeitet werden bis eine Synchronisierung stattfindet |
| `float q_max` | Wert für Abbruchkriterium, wenn alle lokalen Fehlervariablen kleiner sind |
| `int device` | Gerät auf dem parallelisiert gerechnet werden soll (DEVICE_CPU oder DEVICE_GPU) |
| `int gpu_mode` | Modus der GPU-Verarbeitung (GPU_MODE_FLEX, GPU_MODE_CON, GPU_MODE_SHARED) |
| `int datapartition_mode` | Modus für die Datenverteilung (DATAPARTITION_MODE_MEDIAN, DATAPARTITION_MODE_BATCH, DATAPARTITION_MODE_GNG) |
| `int number_of_gpus` | Anzahl GPUs |
| `int gpus[32]` | Feld mit Nummern der GPUs |
| `size_t size_input_vector` | Größe des Eingabevektoren-Speichers |
| `size_t size_neuron_local_error` | Größe des Neuonenfehler-Speichers |
| `size_t size_neuron_vector` | Größe des Neuronenvektoren-Speichers |
| `size_t size_edge_age` | Größe des Kantenalter-Speichers |
| `size_t size_neuron_vector_batch1` | Größe des Batch-Neuronenvektoren-Speichers 1 |
| `size_t size_neuron_vector_batch2` | Größe des Batch-Neuronenvektoren-Speichers 2 |
| `size_t size_neuron_local_error_batch` | Größe des Batch-Neuronenvektoren-Speichers 2 |
| `size_t size_edge_age_batch` | Größe des Batch-Kantenalter-Speichers |
| `size_t size_distances` | Größe des Abstand-Speichers |
| `size_t size_distances_reduction` | Größe des Abstandzwischenergebnis-Speichers |
| `size_t size_winner` | Größe des Gewinner-Speichers |
| `size_t size_second` | Größe des Zweiter-Speichers |
| `float *h_input_vector` | Zeiger auf Eingabevektor-Speicher im Host-RAM |
| `float *h_neuron_local_error` | Zeiger auf Fehlervariablen-Speicher im Host-RAM |
| `float *h_neuron_vector` | Zeiger auf Referenzvektor-Speicher im Host-RAM |
| `float *h_edge_age` | Zeiger auf Kantenzähler-Speicher im Host-RAM |
| `float *h_distances` | Zeiger auf Abstände-Speicher im Host-RAM |
| `float *h_distances_reduction` | Zeiger auf Abständezwischenergebnisse-Speicher im Host-RAM |
| `int *h_winner` | Zeiger auf Gewinner-Speicher im Host-RAM |
| `int *h_second` | Zeiger auf Zweiter-Speicher im Host-RAM |

| | |
|---|---|
| `float *h_neuron_vector_batch1` | Zeiger auf Batch-Eingabevektor-Speicher 1 im Host-RAM |
| `float *h_neuron_vector_batch2` | Zeiger auf Batch-Eingabevektor-Speicher 2 im Host-RAM |
| `float *h_neuron_local_error_batch` | Zeiger auf Batch-Fehlervariablen-Speicher im Host-RAM |
| `float *h_edge_age_batch` | Zeiger auf Batch-Kantenzähler-Speicher im Host-RAM |
| `float *h_input_vector_gng` | Zeiger auf Eingabevektor-Speicher im Host-RAM für Synchronisations-GNG |
| `float *h_neuron_local_error_gng` | Zeiger auf Fehlervariablen-Speicher im Host-RAM für Synchronisations-GNG |
| `float *h_neuron_vector_gng` | Zeiger auf Referenzvektor-Speicher im Host-RAM für Synchronisations-GNG |
| `float *h_edge_age_gng` | Zeiger auf Kantenzähler-Speicher im Host-RAM für Synchronisations-GNG |
| `float *h_distances_gng` | Zeiger auf Abstände-Speicher im Host-RAM für Synchronisations-GNG |
| `float *h_distances_reduction_gng` | Zeiger auf Abständezwischenergebnisse-Speicher im Host-RAM für Synchronisations-GNG |
| `int *h_winner_gng` | Zeiger auf Gewinner-Speicher im Host-RAM für Synchronisations-GNG |
| `int *h_second_gng` | Zeiger auf Zweiter-Speicher im Host-RAM für Synchronisations-GNG |
| `pthread_barrier_t *barriers` | Zeiger auf die Pthreads-Barriers |
| `struct timeval timer_start[32]` | Feld für Startzeiten. |
| `long timer_seconds[32]` | Feld für Gesamtzeiten (Sekunden) |
| `long timer_microseconds[32]` | Feld für Gesamtzeiten (Mikrosekunden) |
| `int show_dindex` | Dunn-Index berechnen |
| `int show_cindex` | C-Index berechnen |
| `int show_gkindex` | Goodman-Kruskal-Berechnung |

### Eingabevektor-Speicher

In diesen Speicher sind die Komponenten der Eingabevektoren hintereinander eingetragen. Der Werte der i-ten Komponenten des j-ten Vektors steht also an Position (j * Anzahl Dimensionen + i).

### Fehlervariablen-Speicher

In diesem Speicher sind die lokalen Fehlervariablen aller möglichen Neuronen aller Netze eingetragen. Der Wert des j-ten Neurons des p-ten Netzes steht also an Position (p * Maximale Anzahl Neuronen + j).

### Referenzvektor-Speicher

In diesem Speicher sind die Referenzvektoren aller möglichen Neuronen aller Netze eingetragen. Die Werte der Komponenten stehen dabei hintereinander. Der Wert der i-ten Komponente des j-ten Neurons des p-ten Netzes steht also an Position (p * Maximale Anzahl Neuronen * Anzahl Dimensionen + j * Anzahl Dimensionen + i).

### Kantenzähler-Speicher

In diesem Speicher sind die Kantenzähler aller möglichen Kanten aller Netze eingetragen. Der Wert der Kante (k,j) bzw. (j,k) des p-ten Netzes steht also an Position (p * Maximale Anzahl Neuronen * Maximale Anzahl Neuronen + j * Maximale Anzahl Neuronen + k).

### Abstände-Speicher

Die Abstände aller möglichen Neuronen aller Netze zu dem aktuellen Eingabevektor werden hier hintereinander gespeichert. Der Abstand des j-ten Neurons des p-ten Netzes steht also an Position (p * Maximale Anzahl Neuronen + j).

### Abständezwischenergebnisse-Speicher

Die Zwischenergebnisse der Abstände aller möglichen Neuronen aller Netze zu dem aktuellen Eingabevektor werden hier hintereinander gespeichert. Dabei gibt es p_v Zwischenergebnisse je Neuron. Das Zwischenergebnis des v-ten Prozesses des j-ten Neurons des p-ten Netzes steht also an Position (p * Maximale Anzahl Neuronen * p_v + j * p_v + v).

### Gewinner-Speicher

Die Zwischen- und Endergebnisse für die Gewinnerneuronen aller Netze werden hier hintereinander eingetragen. Je Netz gibt es p_n Ergebnisse. Die Position des Ergebnisses des v-ten Prozesses des p-ten Netzes steht an Position (p * p_n + v).

### Zweiter-Speicher

Die Zwischen- und Endergebnisse für die zweitnächsten Neuronen aller Netze werden hier hintereinander eingetragen. Je Netz gibt es p_n Ergebnisse. Die Position des Ergebnisses des v-ten Prozesses des p-ten Netzes steht an Position (p * p_n + v).

### Struktur „learn_net_cpu_thread_args“

Datenstruktur für die Übergabe der Parameter an die Threads beim CPU-Lernen.

| Variable | Beschreibung |
|---|---|
| `int p` | Threadnummer |
| `nettype *net` | Netz |

### Struktur „learn_net_gpu_thread_args“

Datenstruktur für die Übergabe der Parameter an die Threads beim GPU-Lernen.

| Variable | Beschreibung |
|---|---|
| `int p` | Threadnummer |
| `nettype *net` | Netz |

## Funktionen

### learn_net_cpu_thread

Statische Funktion, die von den einzelnen CPU-Threads beim CPU-Lernen ausführt wird.

### learn_net_gpu_thread

Statische Funktion, die von den einzelnen CPU

### create_net

Füllt die Variablen des Netzes mit den Standardwerten

### alloc_net

Reserviert den Speicher für das Netz, nachdem alle Parameter angegeben wurden.

### init_net

Initialisiert das Netz mit 2 Neuronen und einer Kante.

### learn_net
Führt den kompletten Trainingsprozess aus.

### output_net
Speichert das Trainingsergebnis (Netz) in eine Datei.

### output_net_p
Speichert den aktuellen Netzzustand eines Prozesses auf Datenebene in eine Datei. Diese Funktion dient nur zu Debug-Zwecken.

### cluster_net
Ermittelt zu allen Eingabevektoren den Cluster und gibt diese Zuordnung in eine Datei aus. Weiterhin werden die Cluster-Indexe berechnet.

### input2file_net
Speichert die Eingabedaten in eine Datei, wenn diese zum Beispiel mit geninput_net generiert wurden.

### file2input_net
liest die Eingabedaten aus einer CSV-Datei aus.

### geninput_net
Generiert beliebig viele Eingabedaten mit beliebiger Länge, die sich in bis zu 5 Clustern befinden.

### geninput2_net
Generiert eine Datenmenge mit komplexer Cluster-Verteilung. Die Daten werden nicht in den Eingabevektor-Speicher, sondern in eine Eingabe- und eine Original-Cluster-Datei gespeichert.

### getsettings_net
Die Trainingsparameter aus einer Parameterdatei einlesen.

### outputsettings_net
Die Trainingsparameter auf der Konsole ausgeben.

### delete_net
Den reservierten Speicher des Netzes freigeben.

### start_timer
Startet einen Timer bzw. setzt ihn fort (timerid = 0..31).

### stop_timer
Stoppt einen Timer.

### output_timer
Gibt alle Timer auf der Konsole aus.

### getcluster
Ermittelt die Zusammenhangskomponenten rekursiv (Tiefensuche).

### copynet2gngsync
Netzzustand in den GNG-Sync-Speicher kopieren, wenn die Synchronisationsmethode „GNG“ ist.

### ptmo_barrier_init
Funktion zum Initialisieren von Barriers. Nutzt unter Windows die Pthreads-Bibliothek, auf anderen Systemen werden andere Pthreads-Konstrukte verwendet, welche schneller sind.

### ptmo_barrier_destroy
Funktion zum Freigeben von Barriers.

### ptmo_barrier_wait
Funktion zum Warten an einer Barrier.

### checkCUDAError
Fehlerprüfung für CUDA: gibt die Fehlernachricht aus und wartet auf Eingabe.

### learn_net_gpu_universal
Lernt auf der GPU. Bei MultiGPU wird diese Funktion von mehreren Threads aufgerufen.

### learn_net_vector_gpu_kernel_con_caller
Kopiert die Parameter in den Konstantenspeicher und ruft den Kernel auf. GPU-Modus: „con“.

### learn_net_vector_gpu_kernel_con
Kernel des GPU-Modus: „con“.

### learn_net_vector_gpu_con
Trainiert einen Eingabevektor mit GPU-Modus: „con“.

### calc_distances_con
Berechnet die Abstandszwischenergebnisse. GPU-Modus: „con“.

### calc_distances_reduce_con
Fasst die Abstandszwischenergebnisse zusammen. GPU-Modus: „con“.

### compare_distances_con
Vergleicht die Abstände und berechnet dafür Zwischenergebnisse. GPU-Modus: „con“.

### compare_distances_reduce_con
Fasst die Zwischenergebnisse der Abstandsvergleiche zusammen. GPU-Modus: „con“.

### update_edge_con
Aktualisiert den Kantenzähler. GPU-Modus: „con“.

### update_local_error_con
Aktualisiert die lokale Fehlervariable. GPU-Modus: „con“.

### update_winner_vector_con
Adaptiert den Referenzvektor des Gewinnerneurons. GPU-Modus: „con“.

### update_neighbors_vector_con
Adaptiert die Referenzvektoren der Nachbarn des Gewinnerneurons. GPU-Modus: „con“.

### learn_net_vector_gpu_kernel_flex_caller
Ruft den Kernel auf. GPU-Modus: „flex“.

### learn_net_vector_gpu_kernel_flex
Kernel des GPU-Modus: „flex“.

### learn_net_vector_gpu
Trainiert einen Eingabevektor mit GPU-Modus: „flex“.

### calc_distances
Berechnet die Abstandszwischenergebnisse. GPU-Modus: „flex“.

### calc_distances_reduce
Fasst die Abstandszwischenergebnisse zusammen. GPU-Modus: „flex“.

### compare_distances
Vergleicht die Abstände und berechnet dafür Zwischenergebnisse. GPU-Modus: „flex“.

### compare_distances_reduce
Fasst die Zwischenergebnisse der Abstandsvergleiche zusammen. GPU-Modus: „flex“.

### update_edge
Aktualisiert den Kantenzähler. GPU-Modus: „flex“.

### update_local_error
Aktualisiert die lokale Fehlervariable. GPU-Modus: „flex“.

### update_winner_vector
Adaptiert den Referenzvektor des Gewinnerneurons. GPU-Modus: „flex“.

### update_neighbors_vector
Adaptiert die Referenzvektoren der Nachbarn des Gewinnerneurons. GPU-Modus: „flex“.

### sync_nets_kernel
Kernel, welche die Netze-synchronisieren-Funktion aufruft. Alle GPU-Modi.

### insert_neuron_kernel
Kernel, welche die Neuron-einfügen-Funktion aufruft. Alle GPU-Modi.

### delete_edges_kernel
Kernel, welche die Kanten-löschen-Funktion aufruft. Alle GPU-Modi.

### insert_neuron
Fügt ein neues Neuron ein.

### delete_edges
Löscht unbedeutende Kanten.

### sync_nets
Synchronisiert die Netze nach Mittelwert- oder Batch-Methode. Alle GPU-Modi.

### learn_net_vector_gpu_kernel_shared_caller
Kopiert die Parameter in den Konstantenspeicher und ruft den Kernel auf. GPU-Modus: „shared“.

### learn_net_vector_gpu_kernel_shared
Kernel, der alle Lernvorgänge für die bestimmte Anzahl an Eingabevektoren durchführt.

### get_edge
Gibt eine Kantenexistenz zurück.

### set_edge
Setzt beziehungsweise löscht eine Kantenexistenz.

### barrier
Warteschranke für eine bestimmte Thread-Gruppe.

### barrier_init
Warteschranke für eine bestimmte Thread-Gruppe initialisieren.

### barrier_destroy
Warteschranke für eine bestimmte Thread-Gruppe freigeben.

### learn_net_vector_cpu
Lernt einen Eingabevektor auf der CPU.

### calc_distances_cpu
Berechnet die Abstandszwischenergebnisse auf der CPU.

### calc_distances_reduce_cpu
Fasst die Abstandszwischenergebnisse auf der CPU zusammen.

### compare_distances_cpu
Vergleicht die Abstände und berechnet dafür Zwischenergebnisse auf der CPU.

### compare_distances_reduce_cpu
Fasst die Zwischenergebnisse der Abstandsvergleiche auf der CPU zusammen.

### update_edge_cpu
Aktualisiert den Kantenzähler auf der CPU.

### update_local_error_cpu
Aktualisiert die lokale Fehlervariable auf der CPU.

### update_winner_vector_cpu
Adaptiert den Referenzvektor des Gewinnerneurons auf der CPU.

### update_neighbors_vector_cpu
Adaptiert die Referenzvektoren der Nachbarn des Gewinnerneurons auf der CPU.

### insert_neuron_cpu
Fügt ein neues Neuron auf der CPU ein.

### delete_edges_cpu
Löscht unbedeutende Kanten auf der CPU.

### sync_nets_cpu

Synchronisiert die Netze auf der CPU nach Mittelwert-, Batch- oder GNG-Methode.