Karlsruher Institut für Technologie

Abschlusspräsentation der Java-Gruppe

Parallelisierung des Neuronale Netze Simulators Neuroph

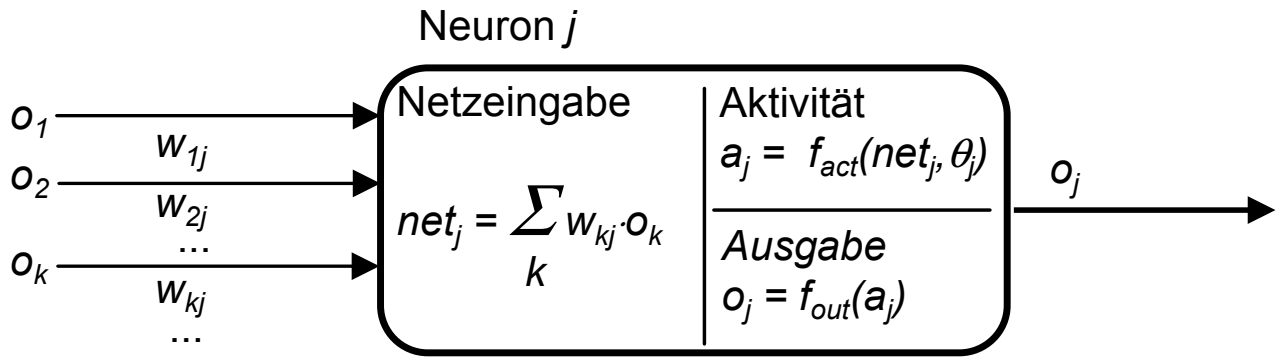
Markus Braun, Daniel Hammann, Dominik Messinger, Dominic Rausch | 12. Februar 2013

INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD), LEHRSTUHL FÜR PROGRAMMIERSYSTEME

Neuronale Netze

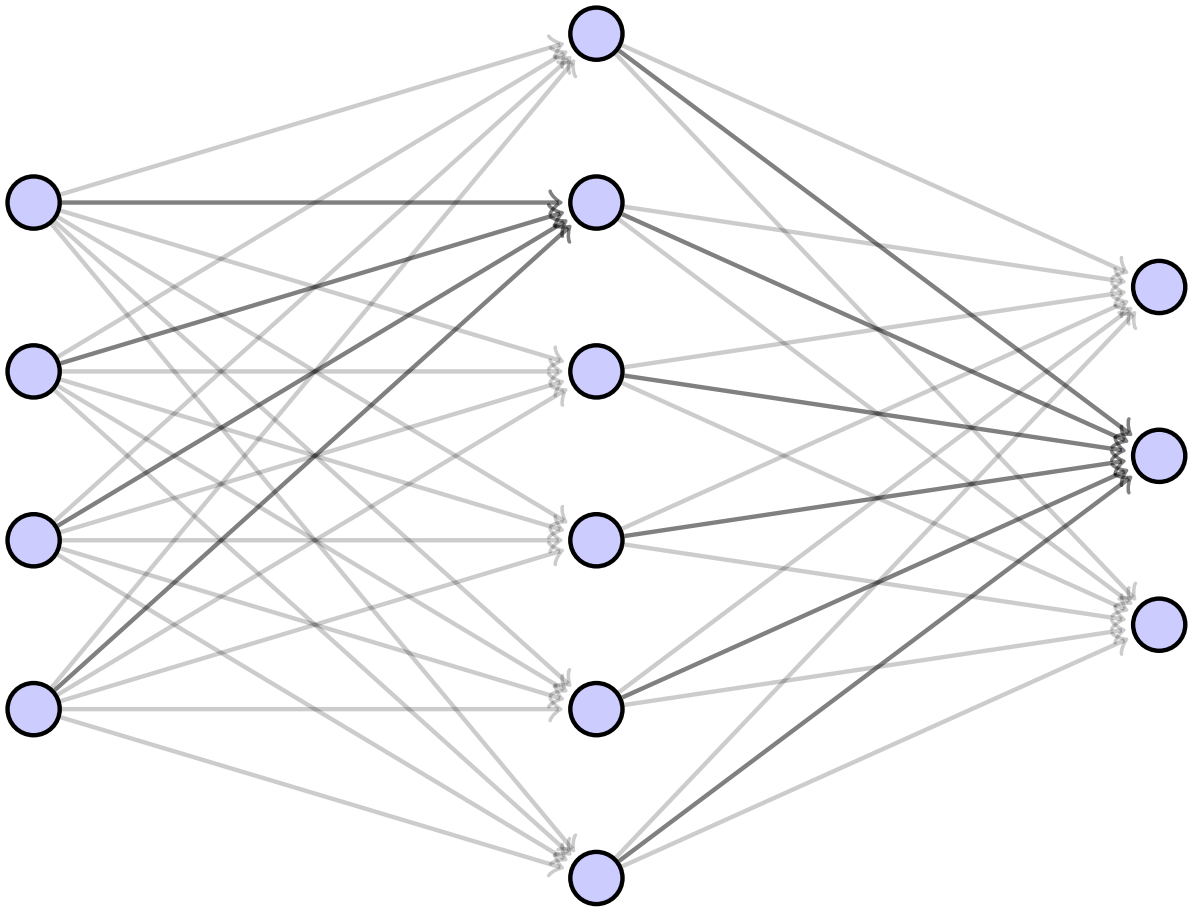
Einleitung

- Bestehen aus Neuronen (Verarbeitungseinheiten)
- Gewichtete Eingabeverbindungen werden zusammenfasst (Propagierungsfunktion)
- Aktivierungsfunktion mit Schwellwert
- Aus Aktivierung folgt mittels Ausgabefunktion die Ausgabe



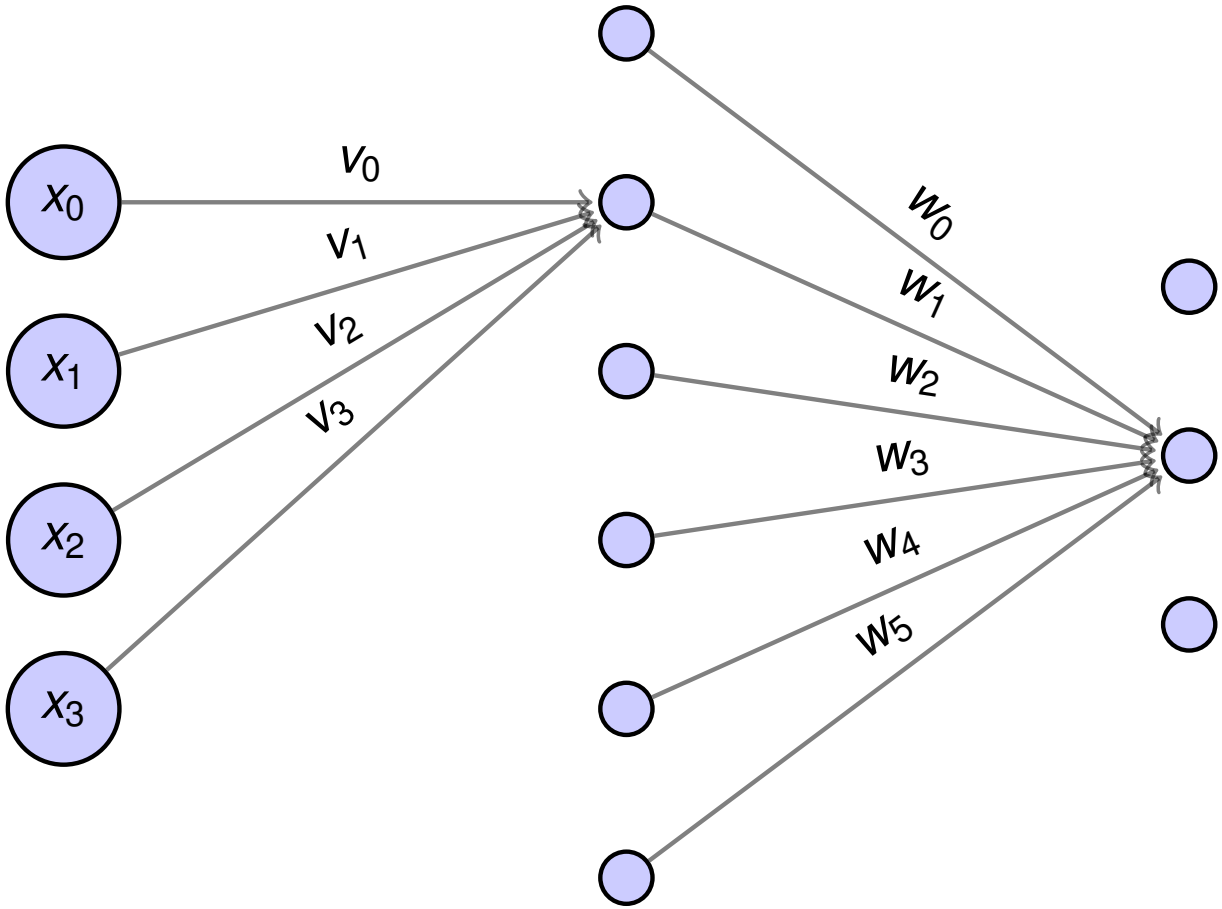
[Quelle: Lehr- und Übungsbuch künstliche Intelligenz; Lämmel, Cleve; 2012]

NN Auswertung



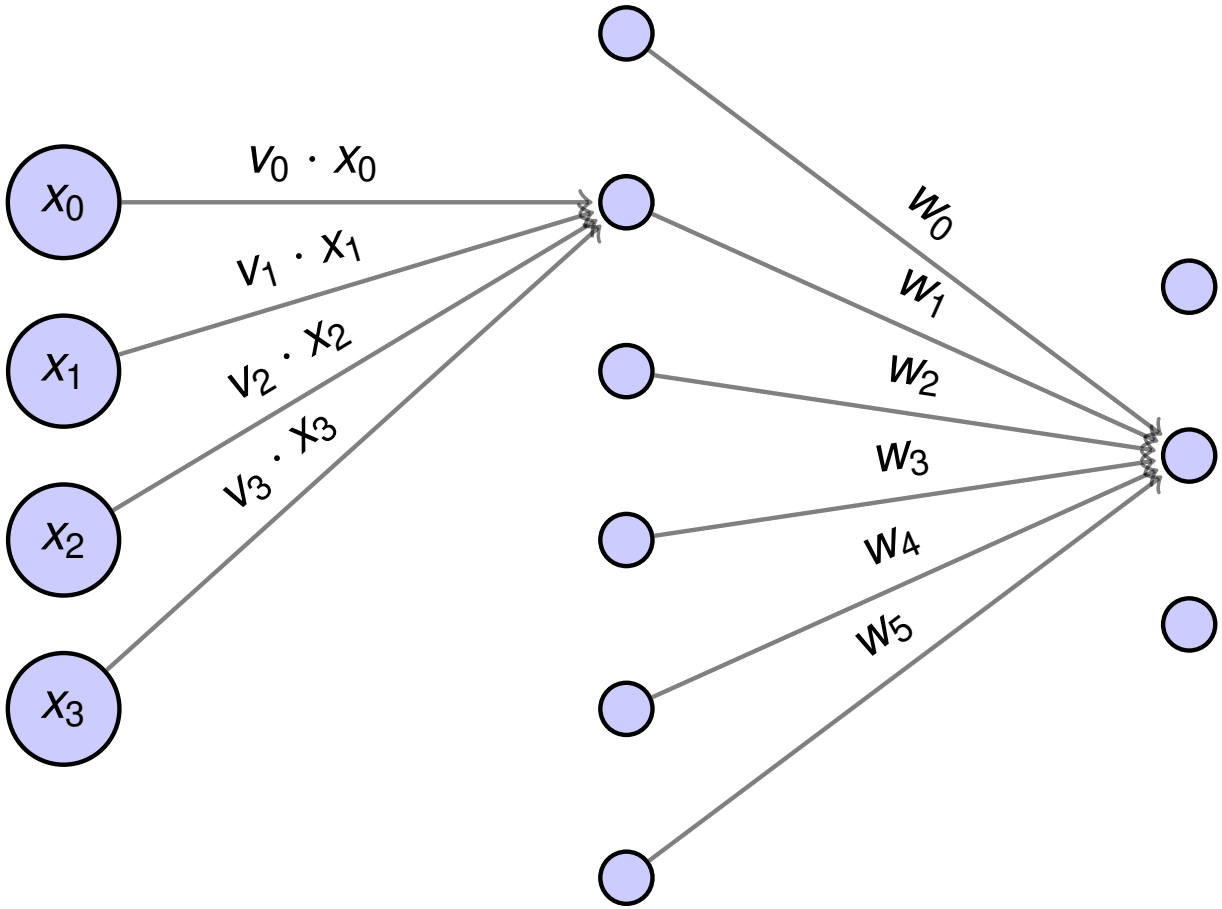
- Multilayerperceptron-Netz

NN Auswertung



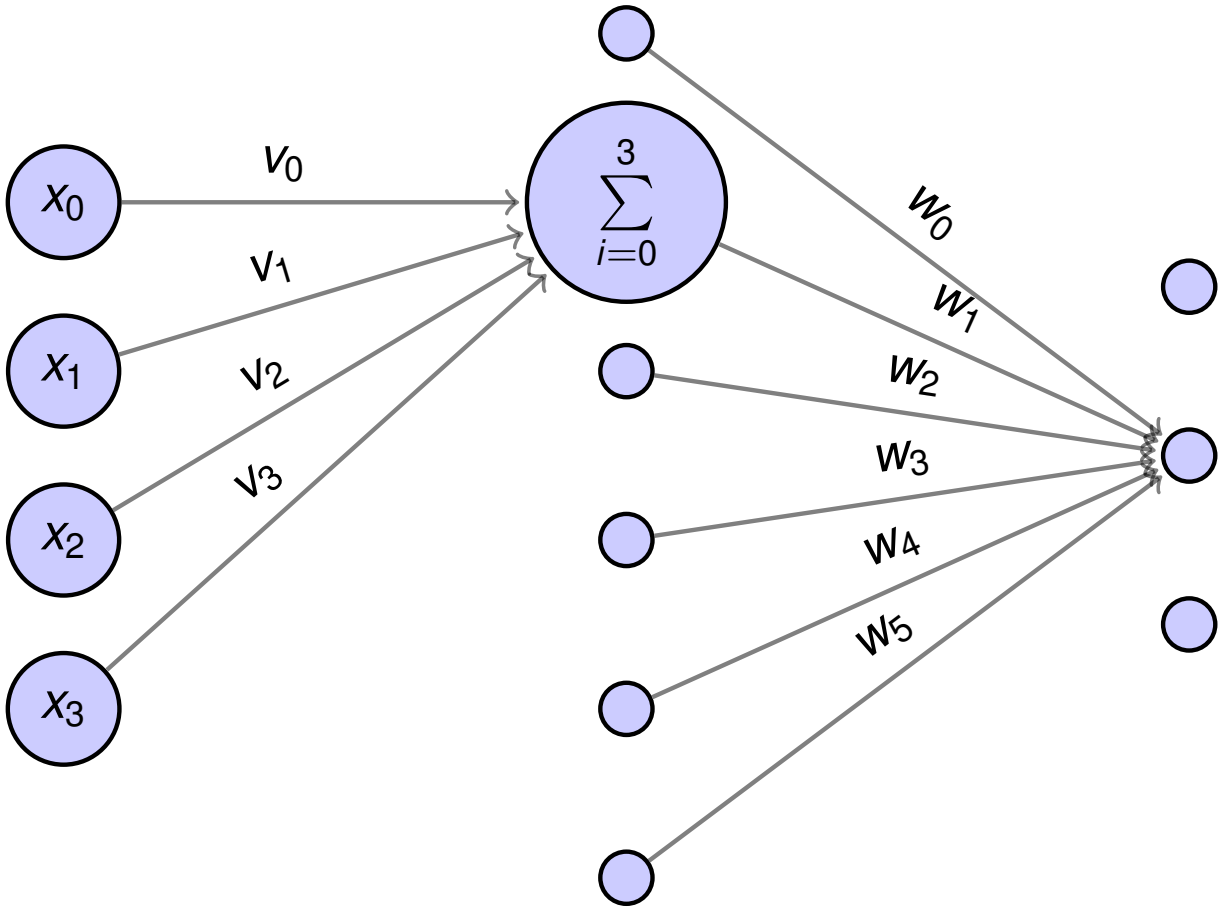
- Trainingseingabe und Kantengewichte

NN Auswertung



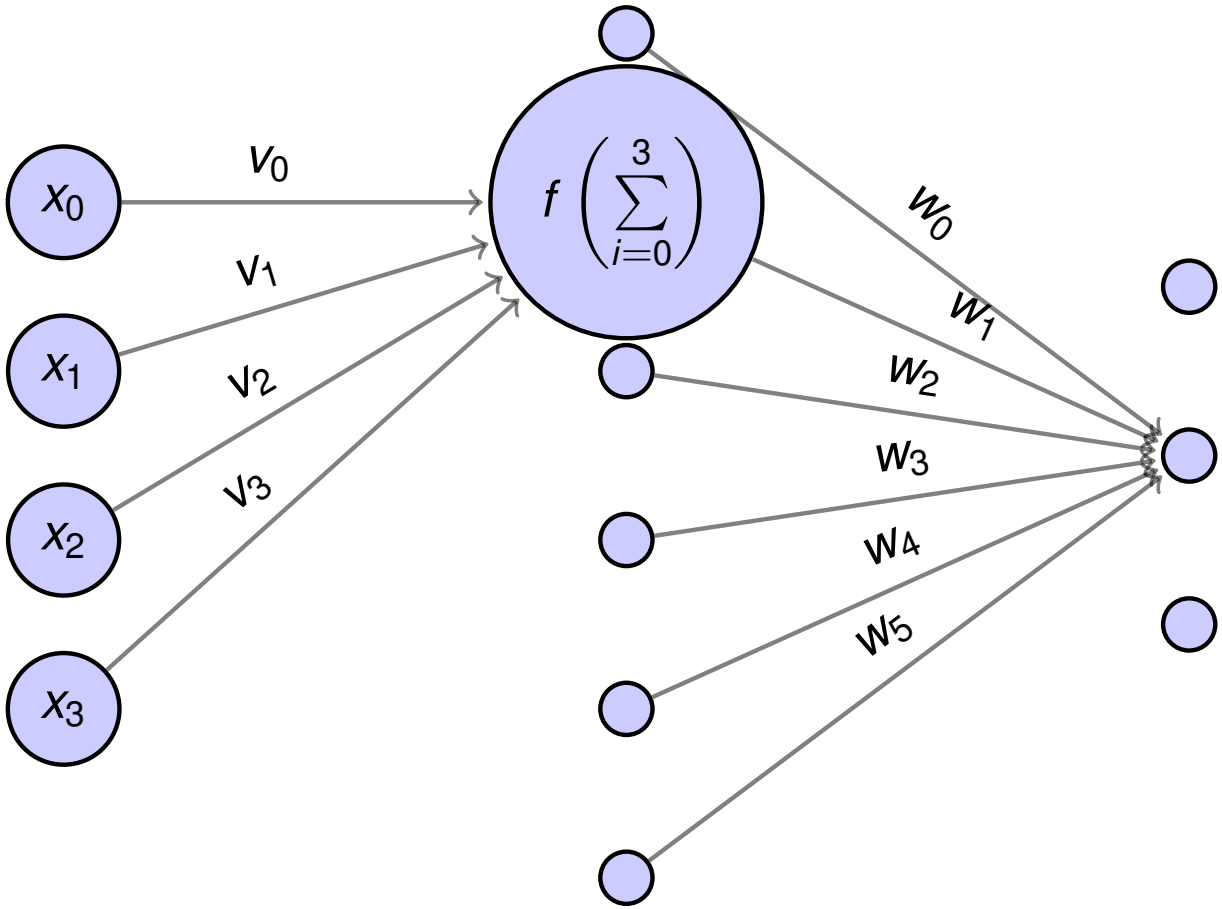
- Gewichtung der Trainingseingabe mit Kantengewicht

NN Auswertung



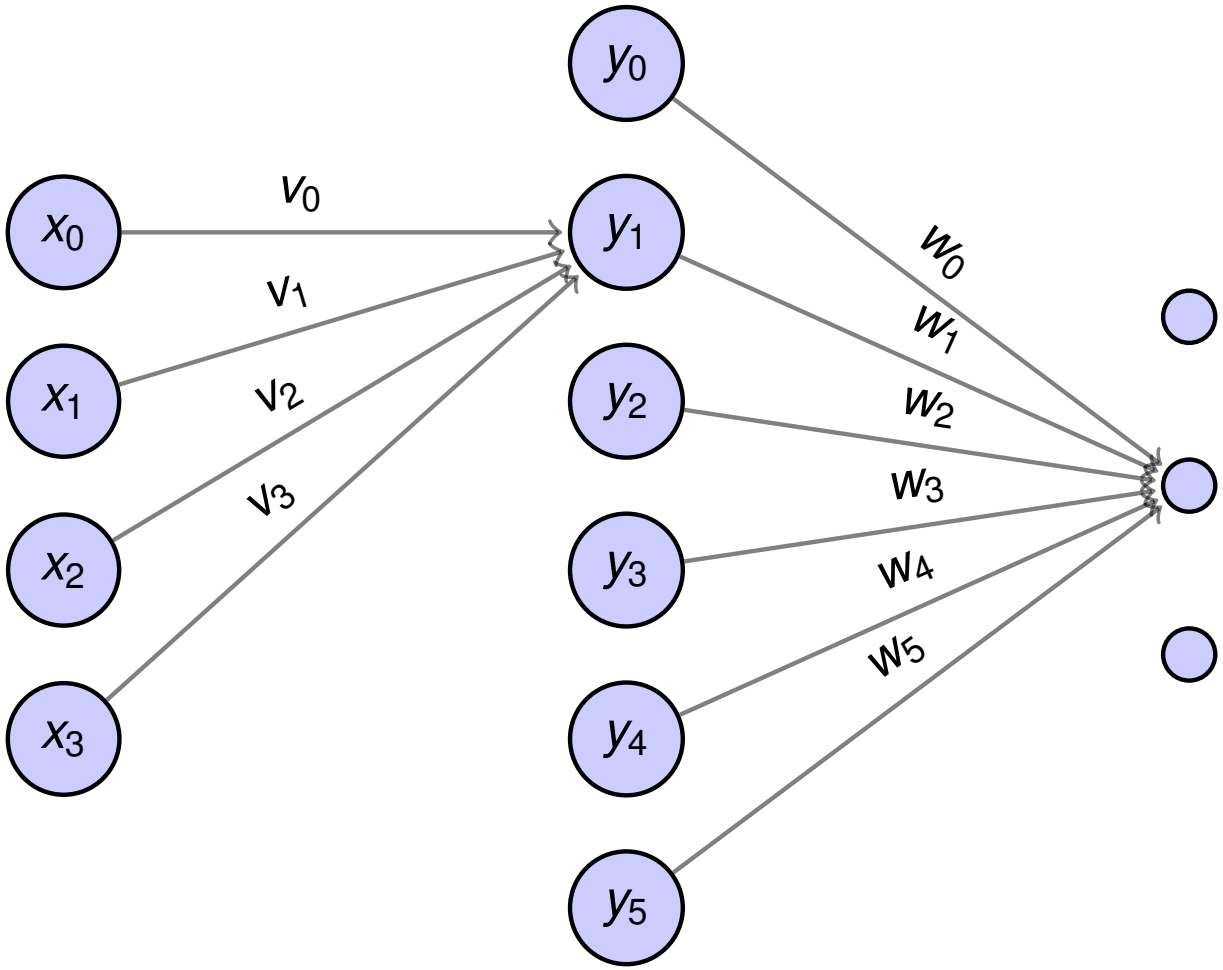
- Summe der Produkte ergibt Netzeingabe in der Hiddenschicht

NN Auswertung



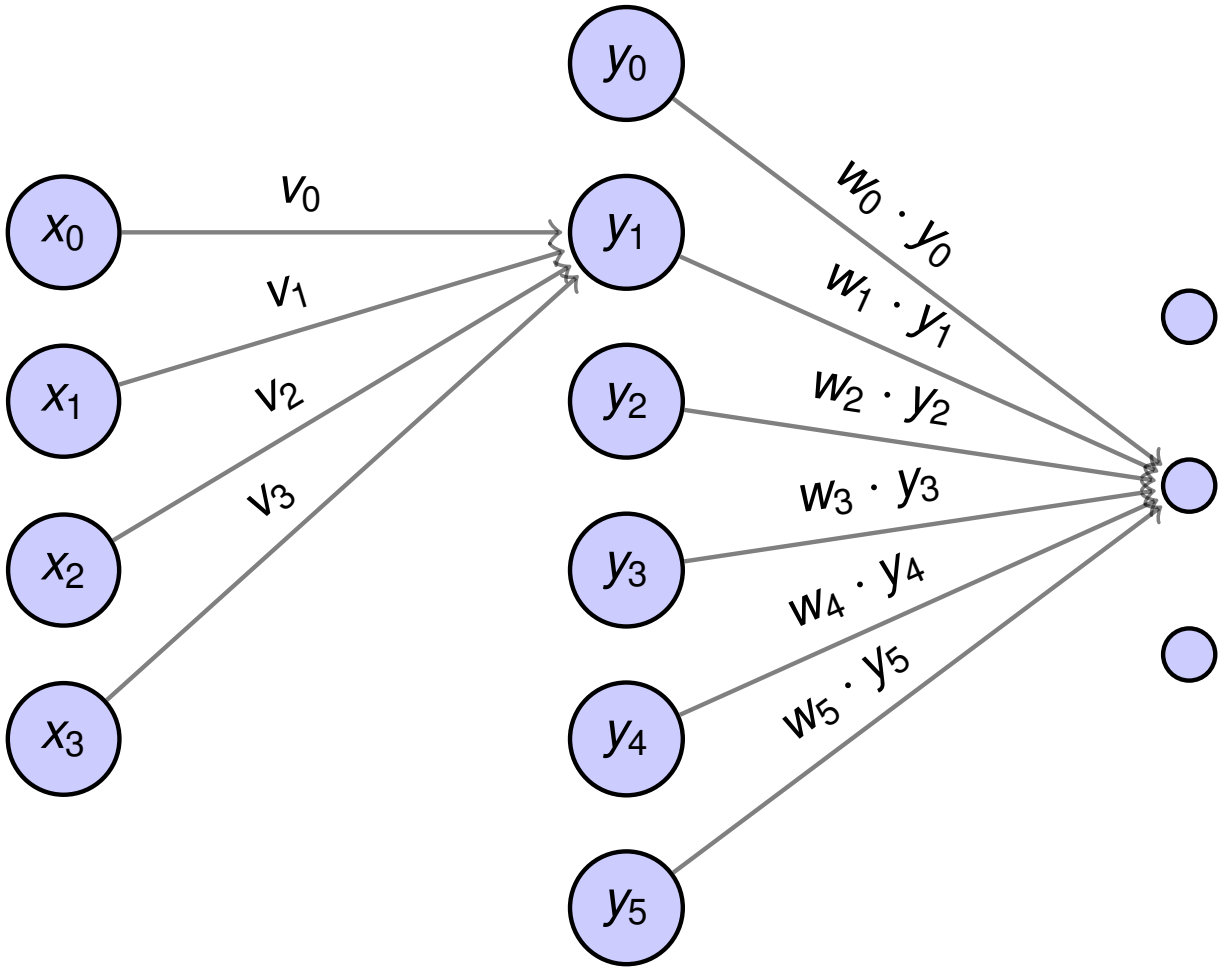
- Netzeingabe der restlichen Neuronen der Hiddenschicht

NN Auswertung



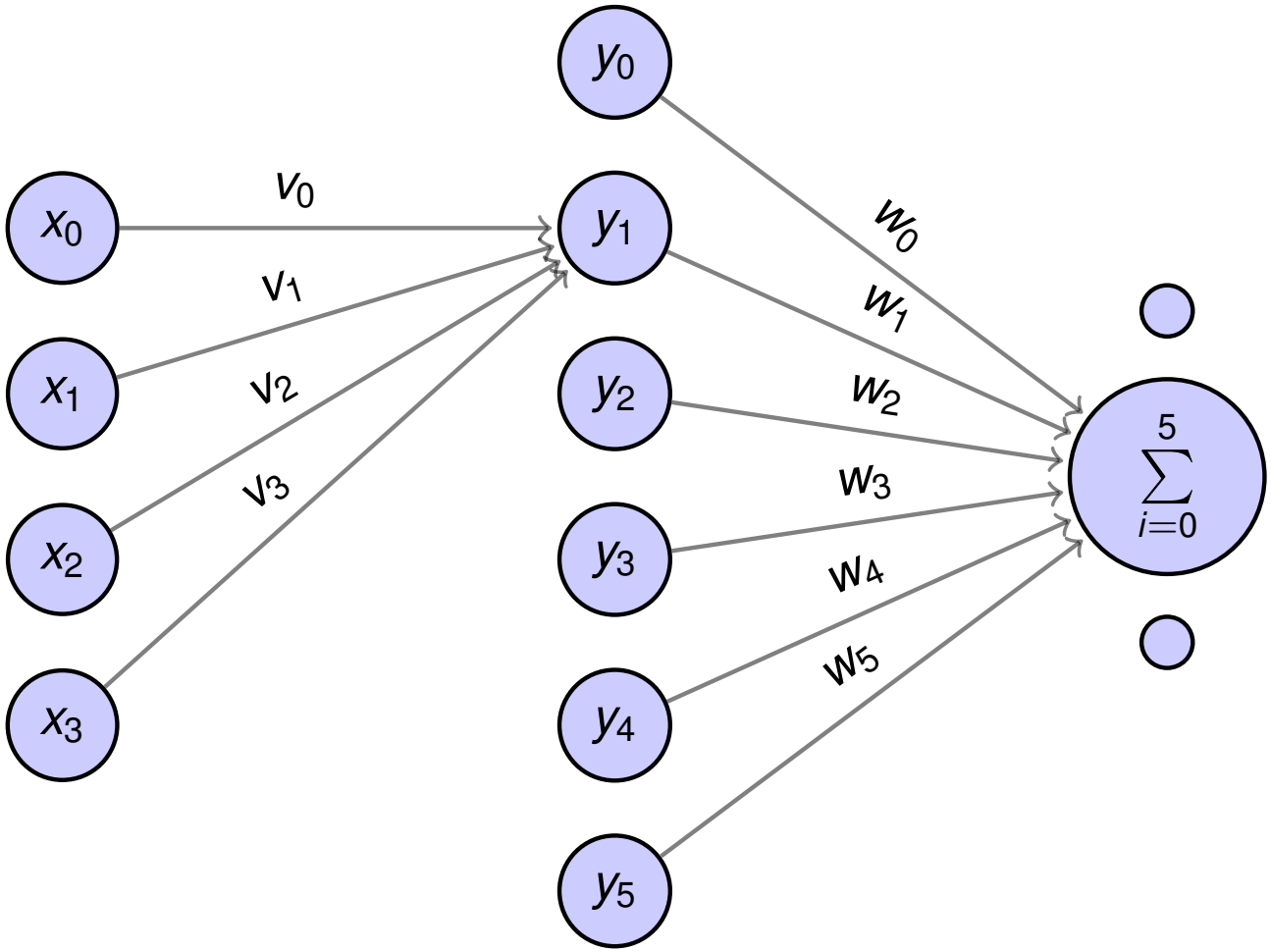
- Umrechnung der Netzeingabe mit der Übertragungsfunktion

NN Auswertung



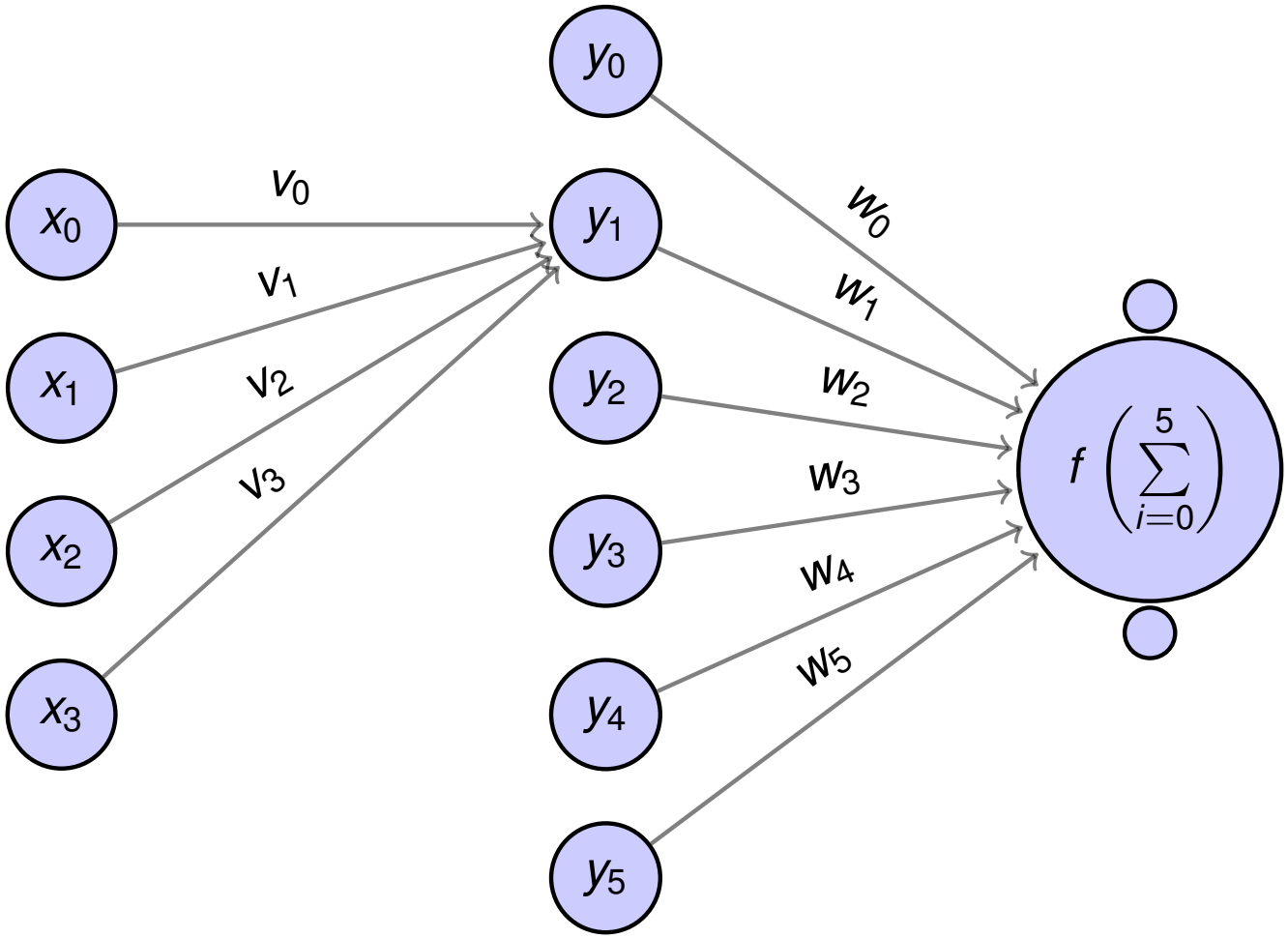
- Gewichtung der Ausgabewerte mit Kantengewicht

NN Auswertung



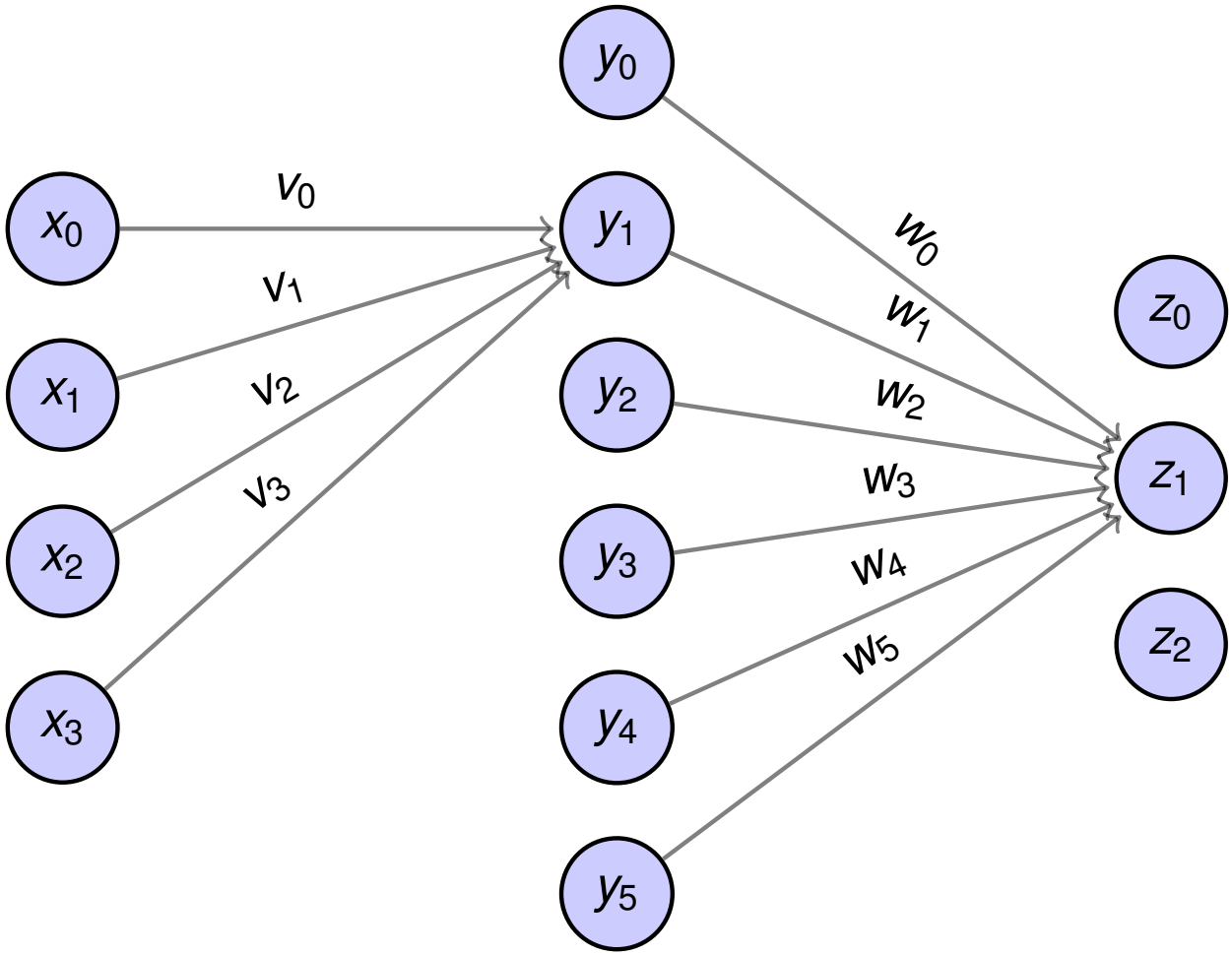
- Summe der Produkte ergibt Netzeingabe in der Ausgabeschicht

NN Auswertung



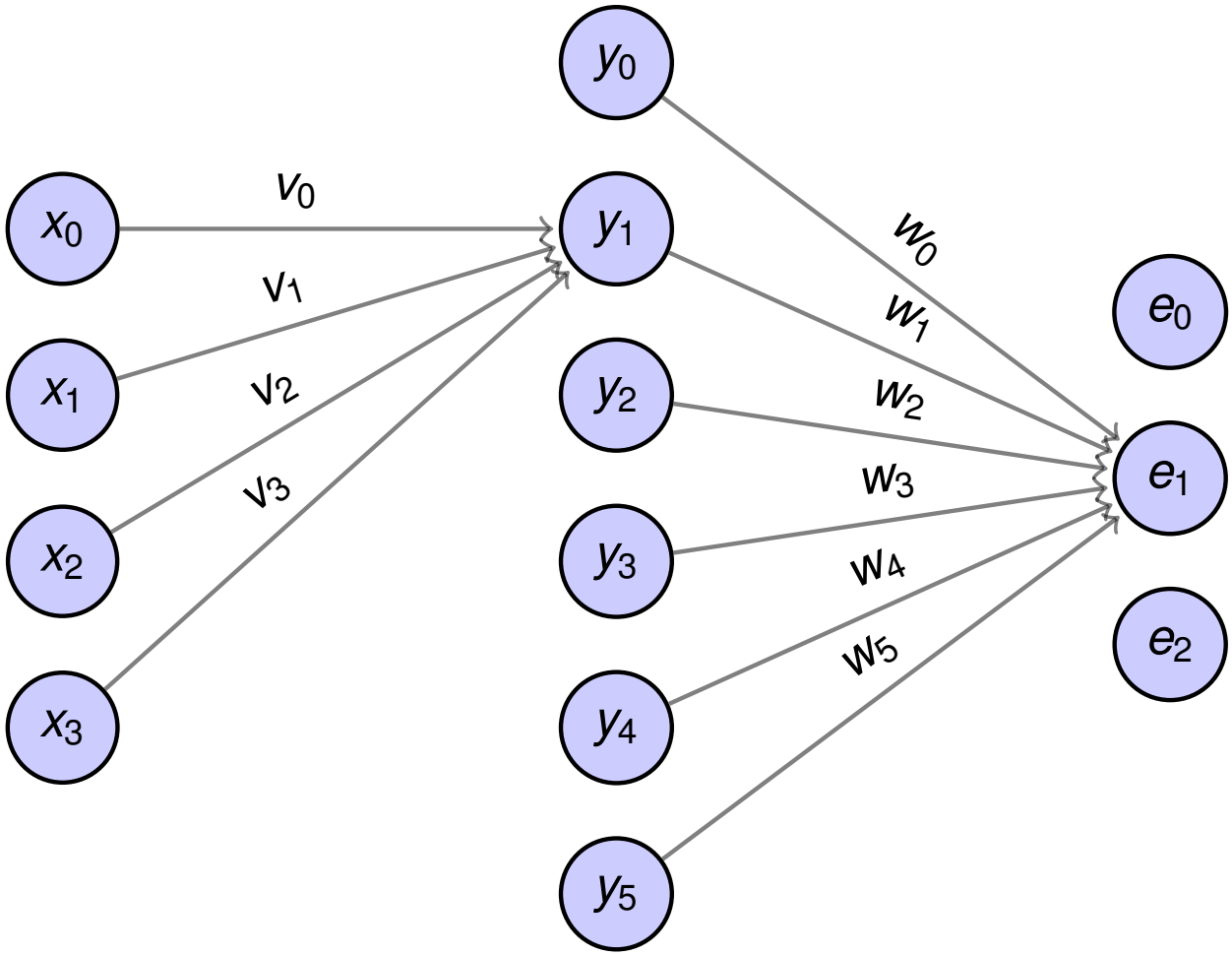
- Netzeingabe der restlichen Neuronen der Ausgabeschicht

NN Auswertung



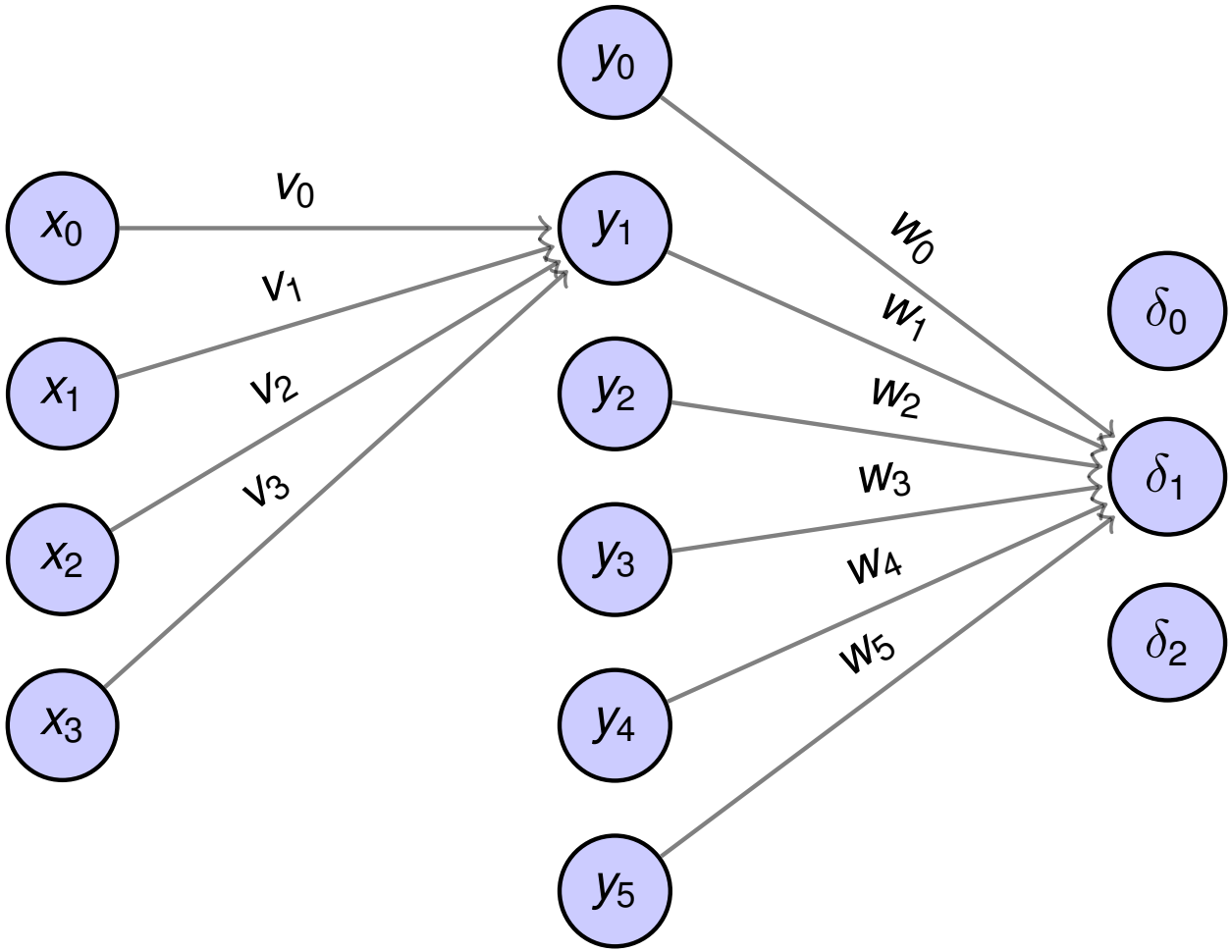
- Umrechnung der Netzeingabe mit der Übertragungsfunktion

NN Auswertung



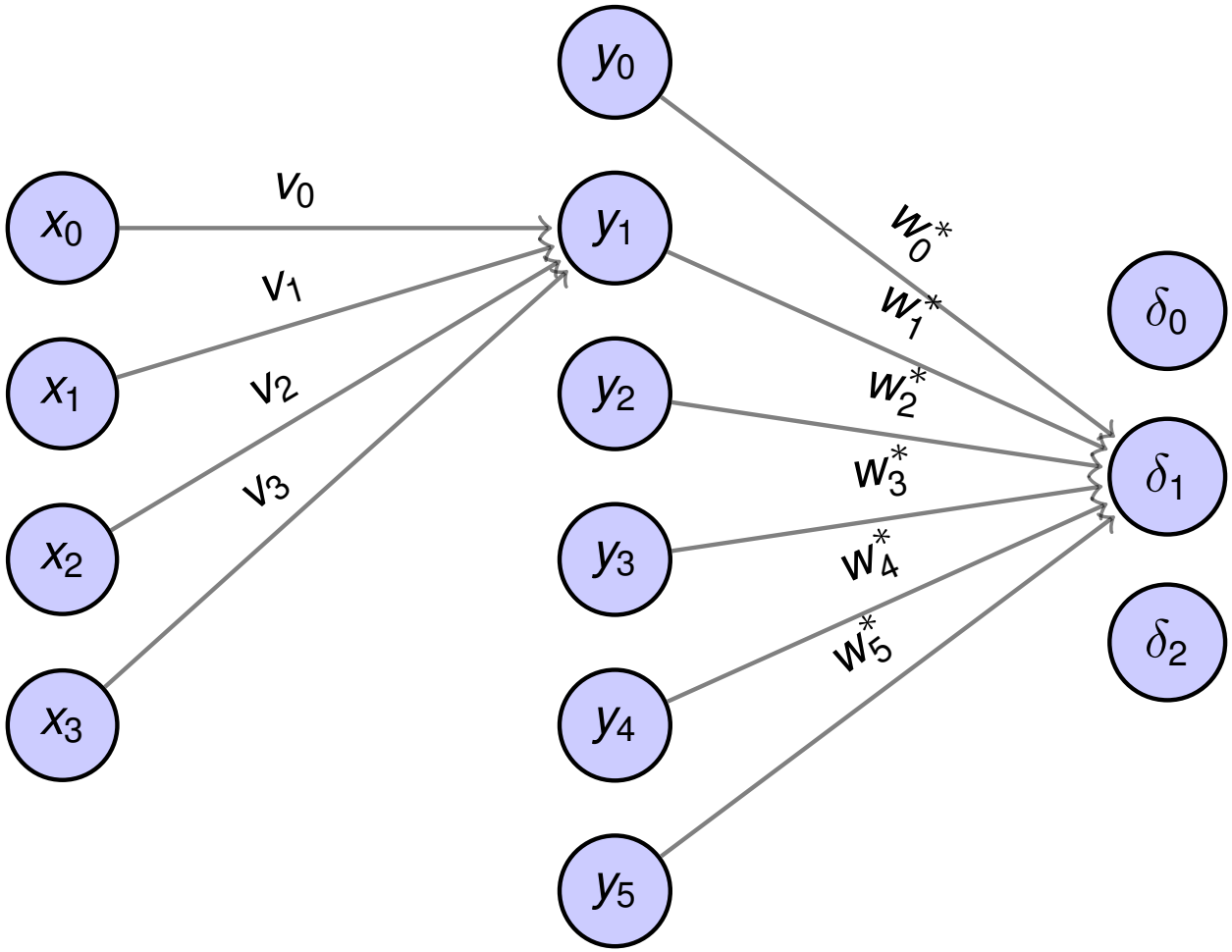
- Berechnung des Ausgabefehlers anhand Ausgabewert und gewünschtem Ausgabewert

NN Auswertung



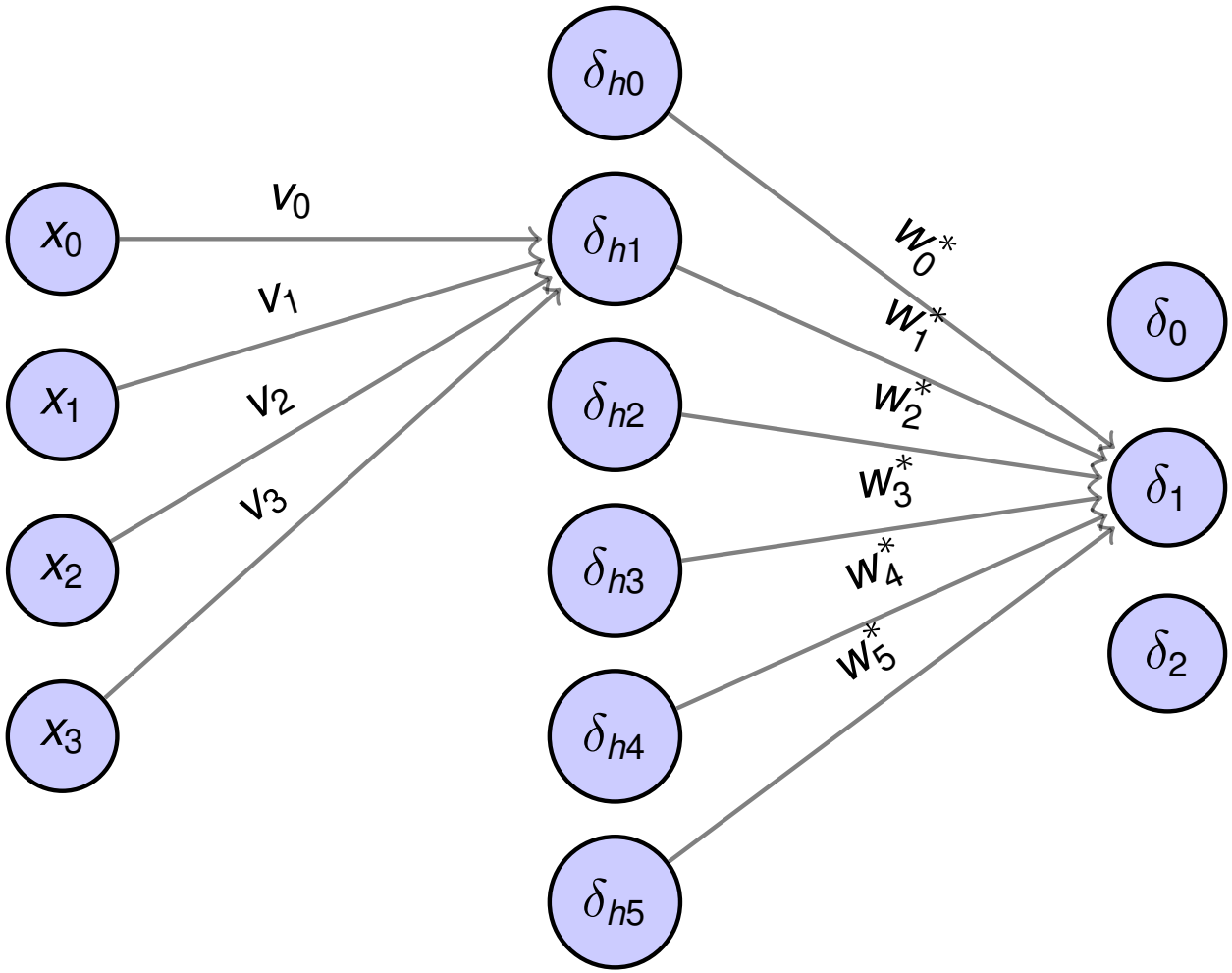
- Berechnung von delta-Werten anhand Netzeingabe, Übertragungsfunktion und Ausgabefehler

NN Auswertung



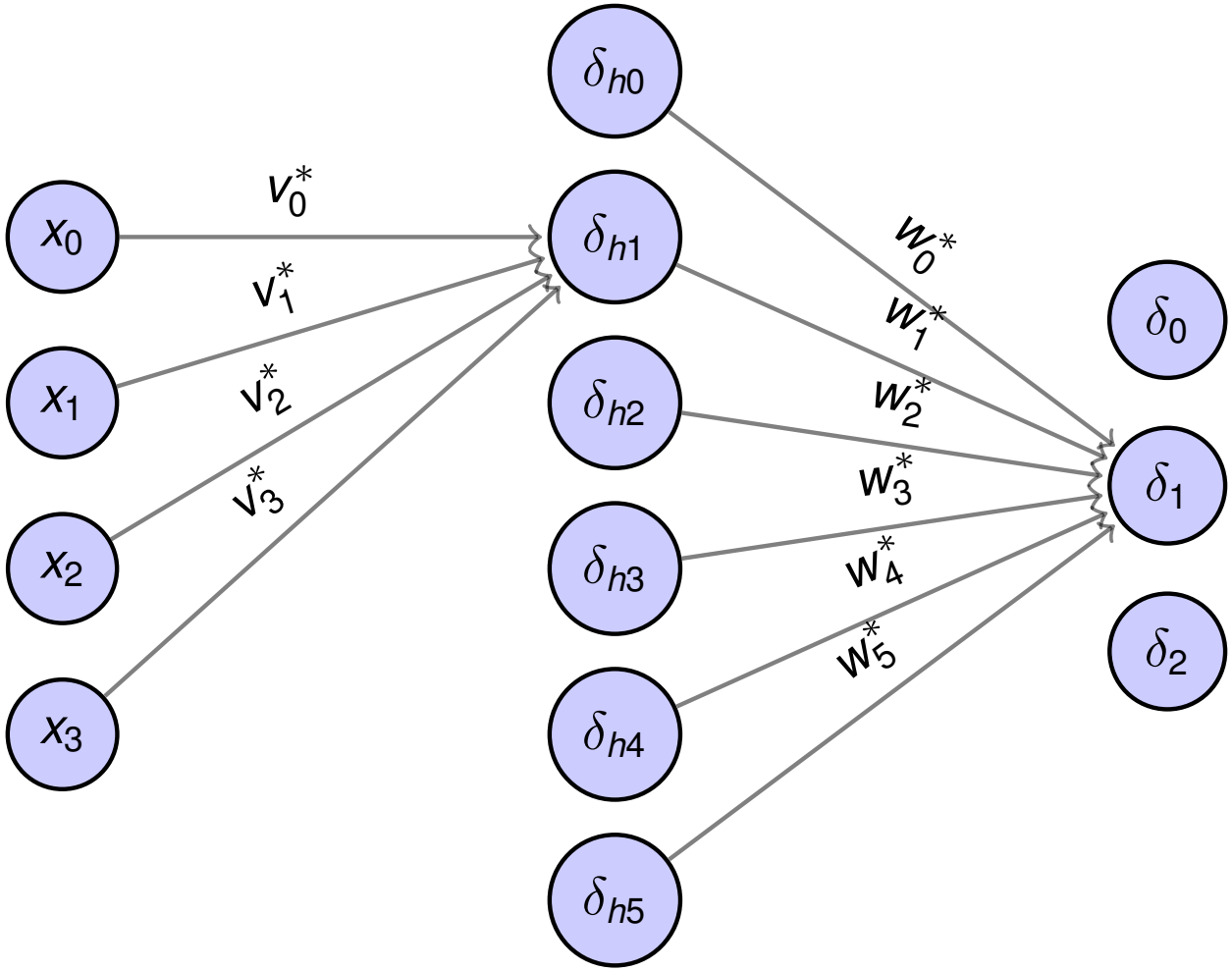
- Anpassung der Kantengewichte

NN Auswertung



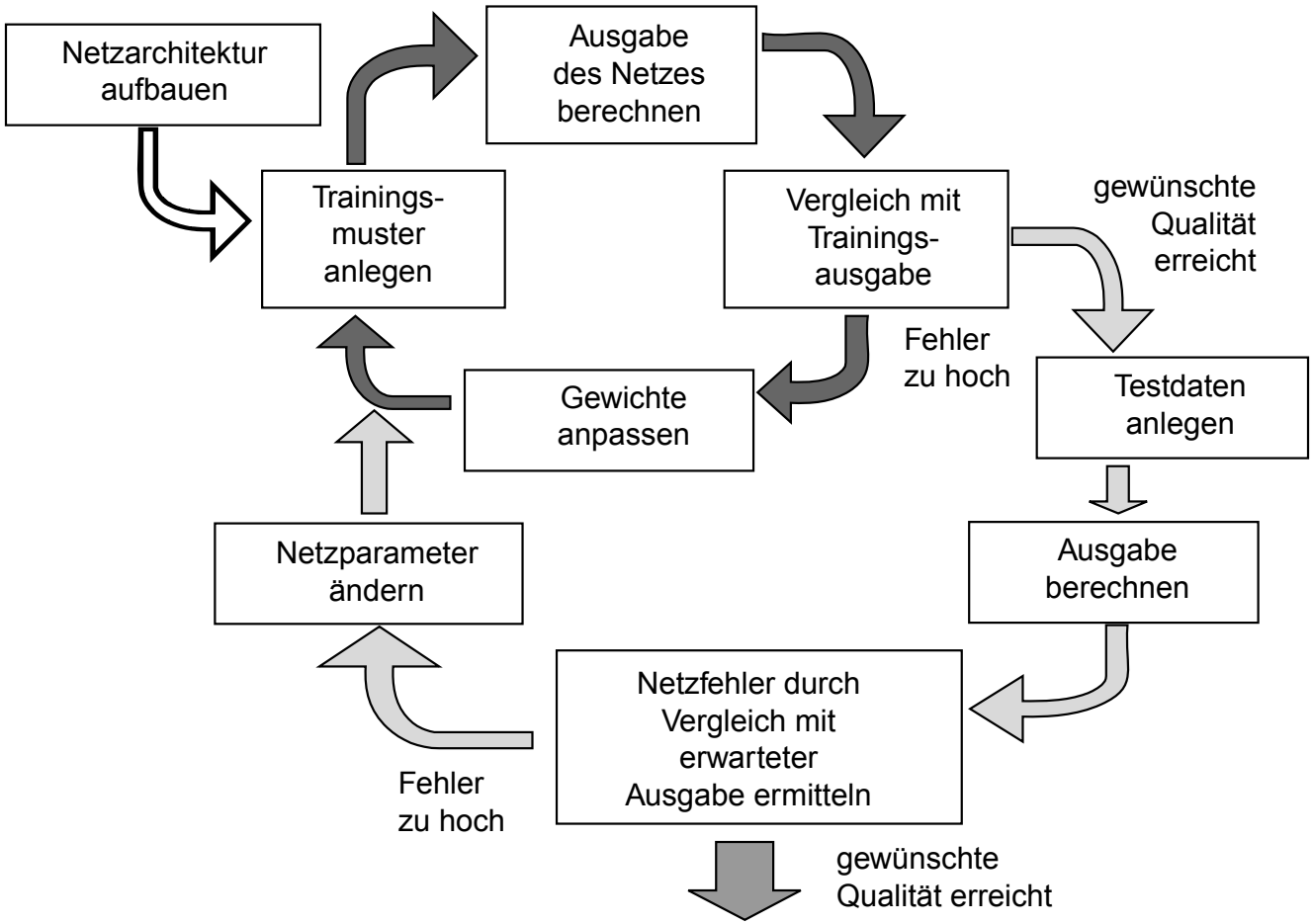
- Berechnung von delta-Werten in Hiddenschicht (benötigt u.a. ausgehende Kantengewichte und delta-Werte der Ausgabeschicht)

NN Auswertung



- Anpassung der Kantengewichte

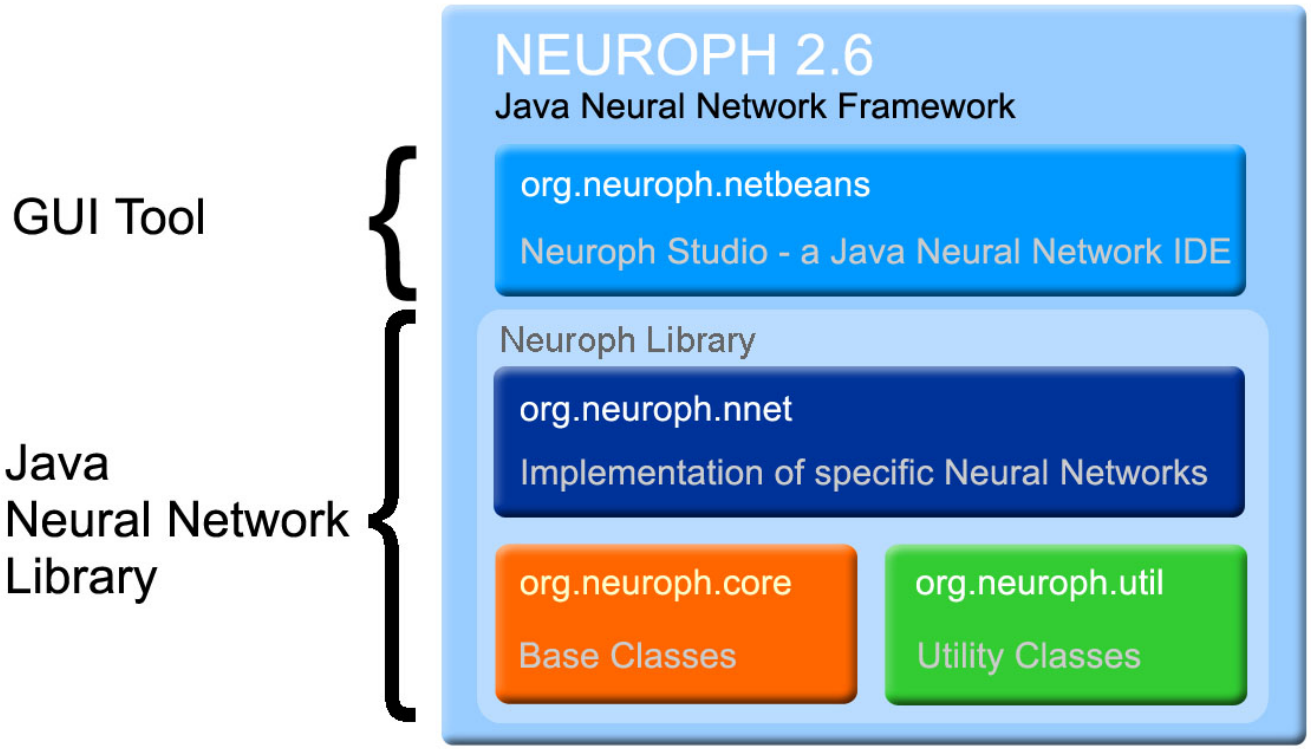
Neuronale Netze



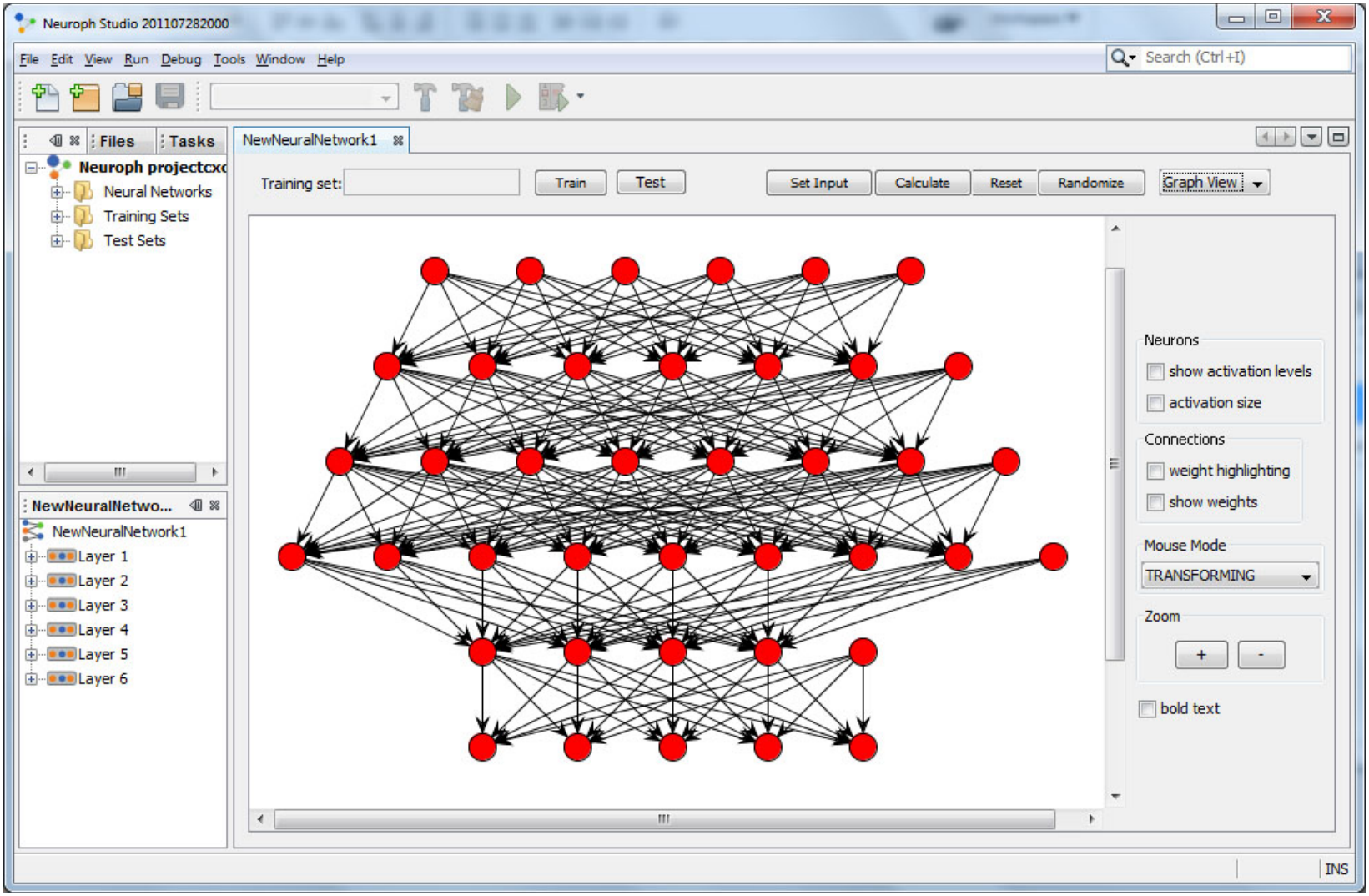
[Quelle: Lehr- und Übungsbuch künstliche Intelligenz; Lämmel, Cleve; 2012]

Neuroph

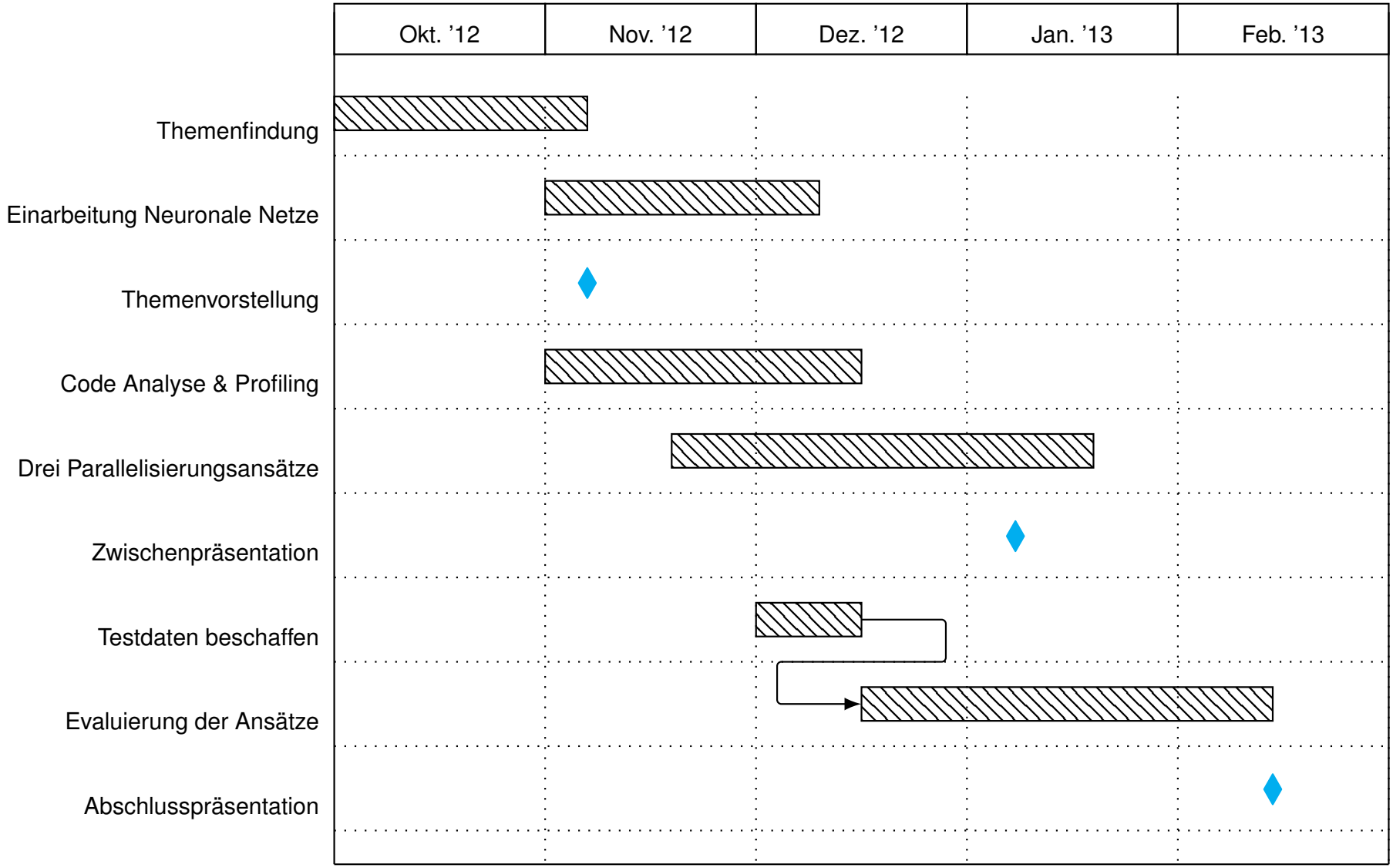
- Simulationsumgebung für Neuronale Netze
- 42751 LOC
- Apache 2.0 Lizenz
- `http://neuroph.sourceforge.net/`



Neuroph


Neuroph Studio 201107282000
File Edit View Run Debug Tools Window Help
Search (Ctrl+I)
Files
Tasks
Neuroph projectcxc
Neural Networks
Training Sets
Test Sets
NewNeuralNetwork1
Training set:
Train
Test
Set Input
Calculate
Reset
Randomize
Graph View
NewNeuralNetwo...
NewNeuralNetwork1
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Layer 6
Neurons
show activation levels
activation size
Connections
weight highlighting
show weights
Mouse Mode
TRANSFORMING
Zoom
+
-
bold text
INS

Überblick

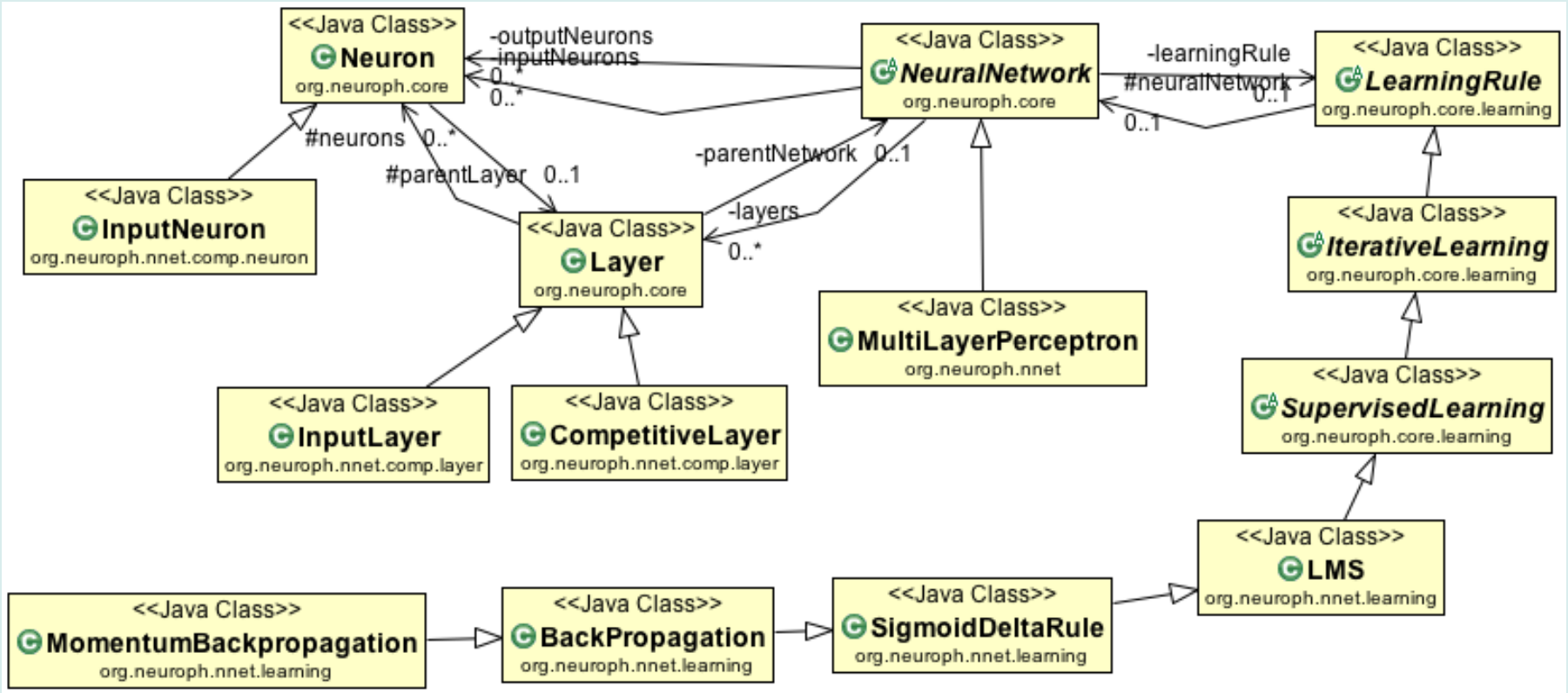

Okt. ’12
Nov. ’12
Dez. ’12
Jan. ’13
Feb. ’13
Themenfindung
Einarbeitung Neuronale Netze
Themenvorstellung
Code Analyse & Profiling
Drei Parallelisierungsansätze
Zwischenpräsentation
Testdaten beschaffen
Evaluierung der Ansätze
Abschlusspräsentation

Überblick

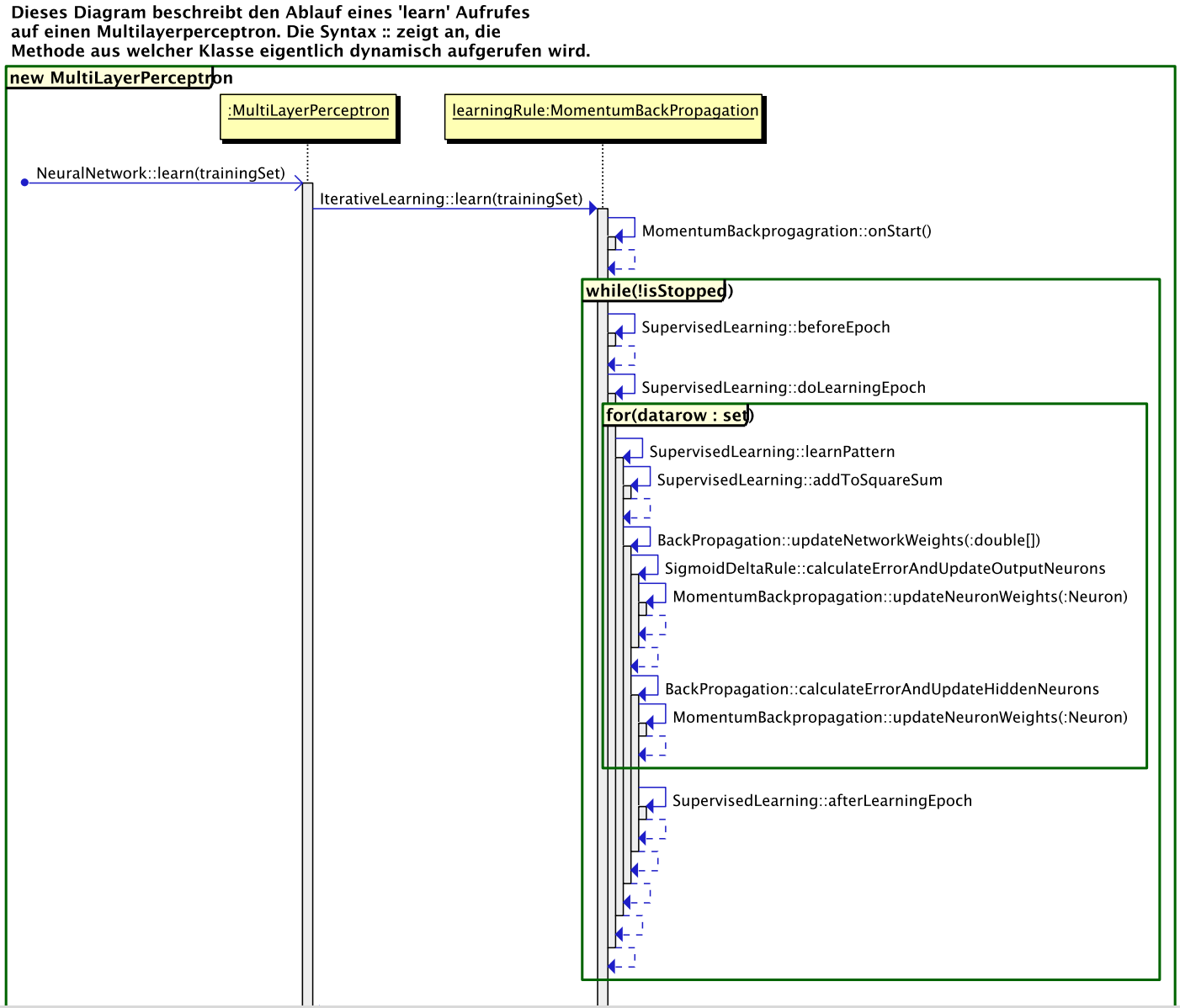
- Einleitung ✓
- Neuroph auf Code-Ebene
- Drei Parallelisierungsansätze
- Evaluierung
- Fazit

Code Analyse I

Was wir vorgefunden haben



Code Analyse II



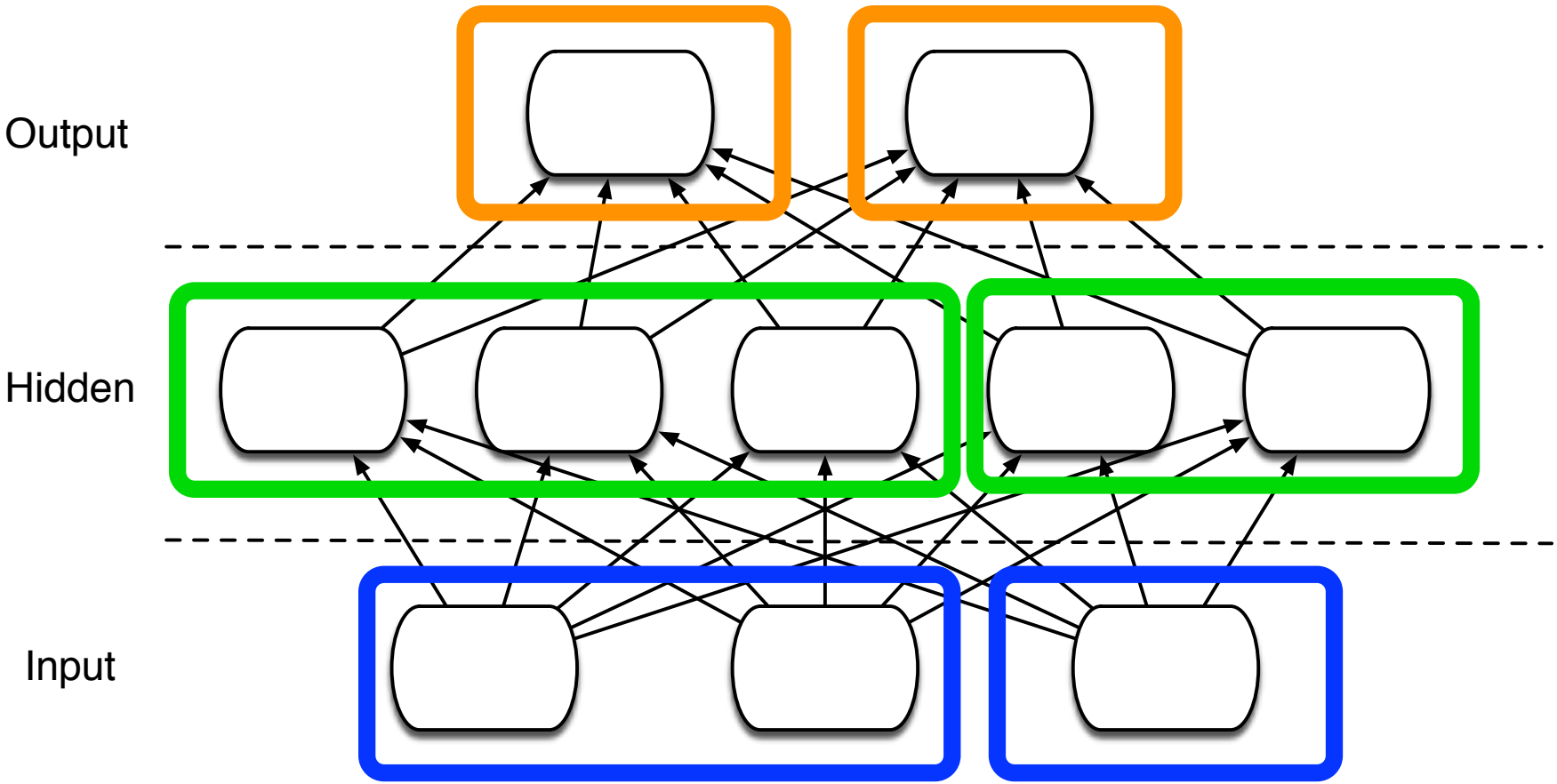
Überblick

- Einleitung ✓
- Neuroph auf Code-Ebene ✓
- Drei Parallelisierungsansätze
- Evaluierung
- Fazit

Parallelisierungsansätze

- Layer-Partitionierung
- Batch Learning Parallelisierung
- Clonebased Parallelisierung

Layer-Partitionierung I



Layer-Partitionierung II

Layer::calculate()

```
1 for (Neuron n : this.neurons)
2     n.calculate();
```

ParallelLayer::calculate()

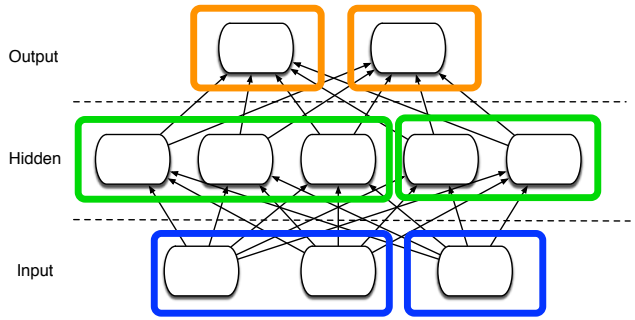
```
1 ExecutorService service = getExecutor();
2 Queue<Future<?>> futures = new LinkedList<>();
3
4 for (NeuronJob j : jobs)
5     futures.add(service.submit(j));
6
7 waitForAll(futures);
```

Layer-Partitionierung III

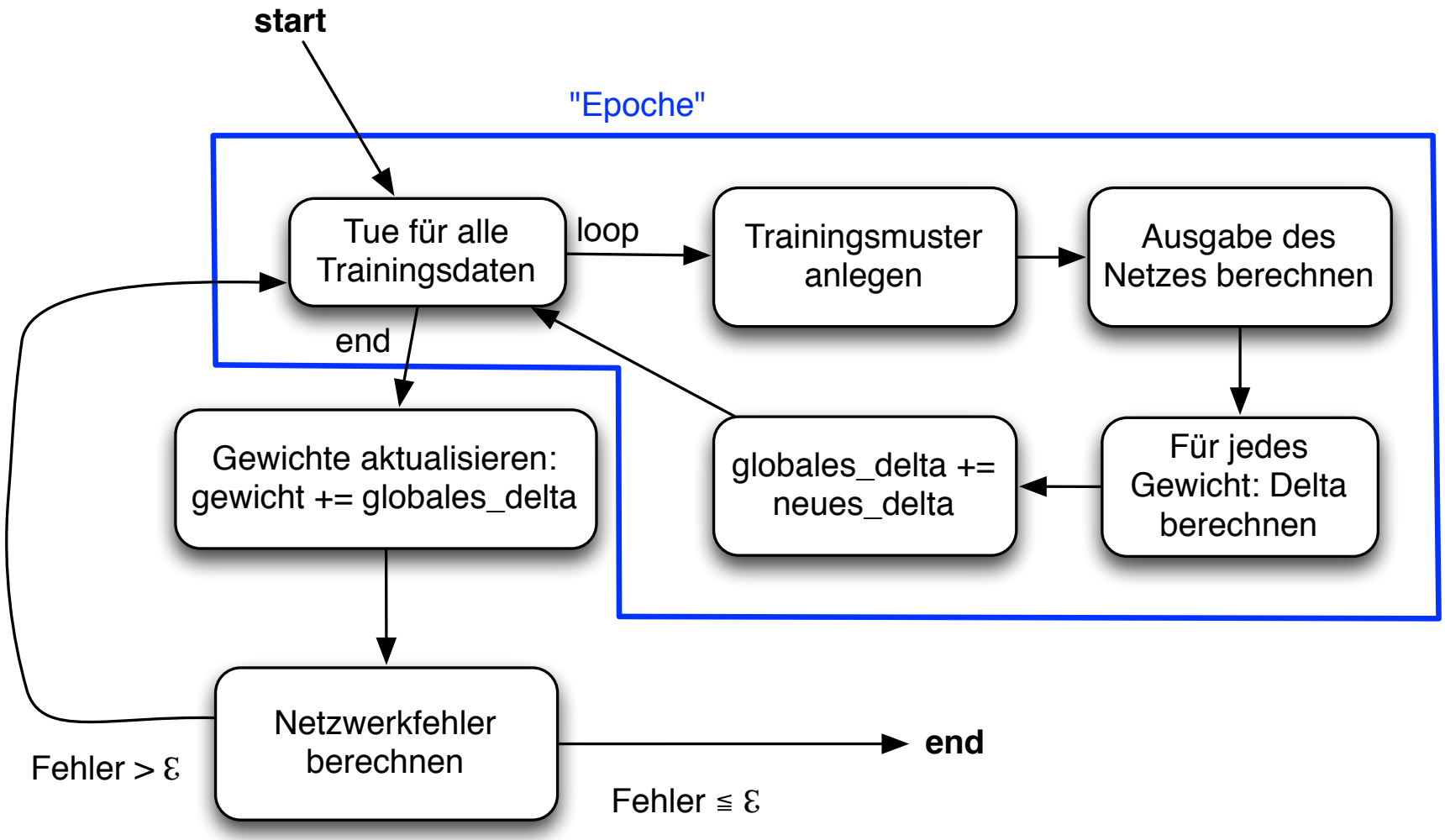
- Neuer Netzwerktyp: Paralleles neurales Netzwerk
  - Profitiert von großen Netzwerken
  - Nicht von großen Datenmengen
- Fast alle gemessenen Speedups < 1

Erster naiver Ansatz ineffektiv

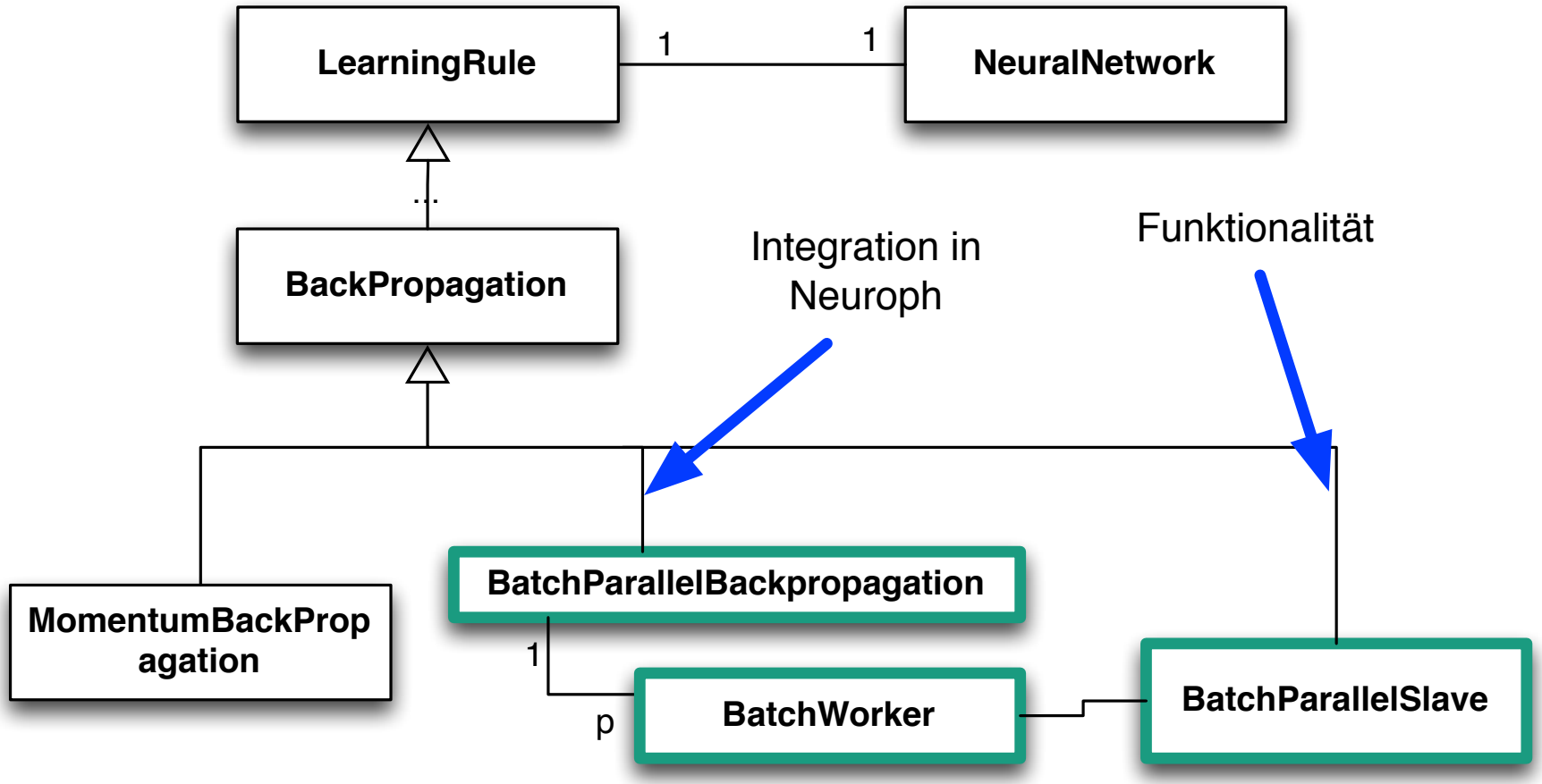
- Auswertung des NN sehr schnell
- Hotspot ist eine `get`-Methode!



Exkurs: Batch lernen



Batch Learning Parallelisierung I



Paralleles Lernen statt parallelem Netzwerk

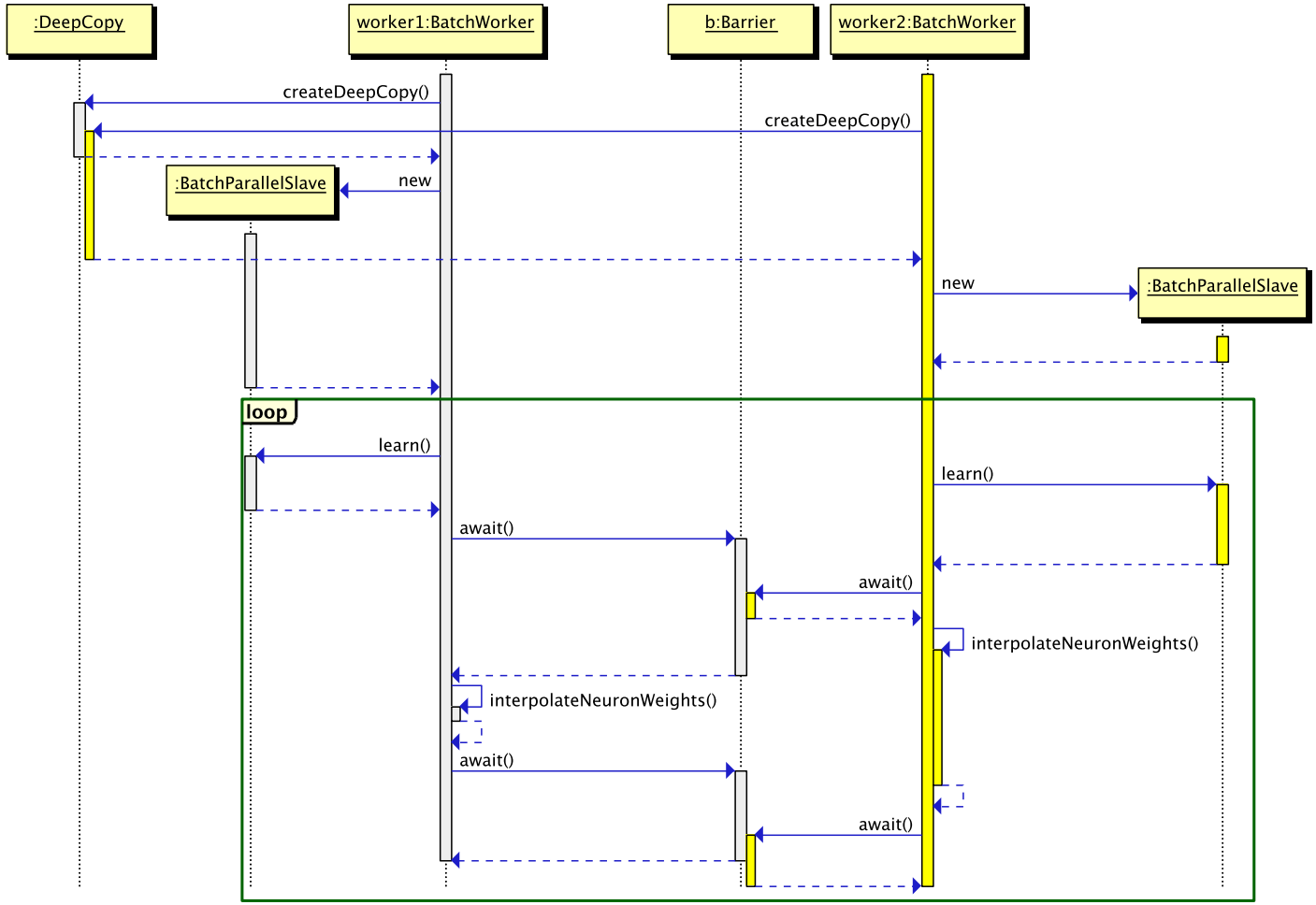
Batch Learning Parallelisierung II

- Paralleler Lernvorgang (Netzwerk sequentiell)
- Vorteil:
  - Saubere, konsistente Schnittstelle
  - Zugriff auf *protected* Funktionalität
- Nachteil:
  - Lernalgorithmus fest
    - Backpropagation
  - Softwaretechnisch fragwürdig
    - Beispiel: Override mit leerem Body

Batch Learning Parallelisierung III

- Trainingsdaten auf Worker verteilt
- Worker lernen unabhängig
- Gewichtänderungen aufaddieren
  - Jeder Thread bearbeitet Subset
  - Dadurch keine Abhängigkeiten

Batch Learning Parallelisierung IV



Clonebased Parallelisierung I

Charakteristika

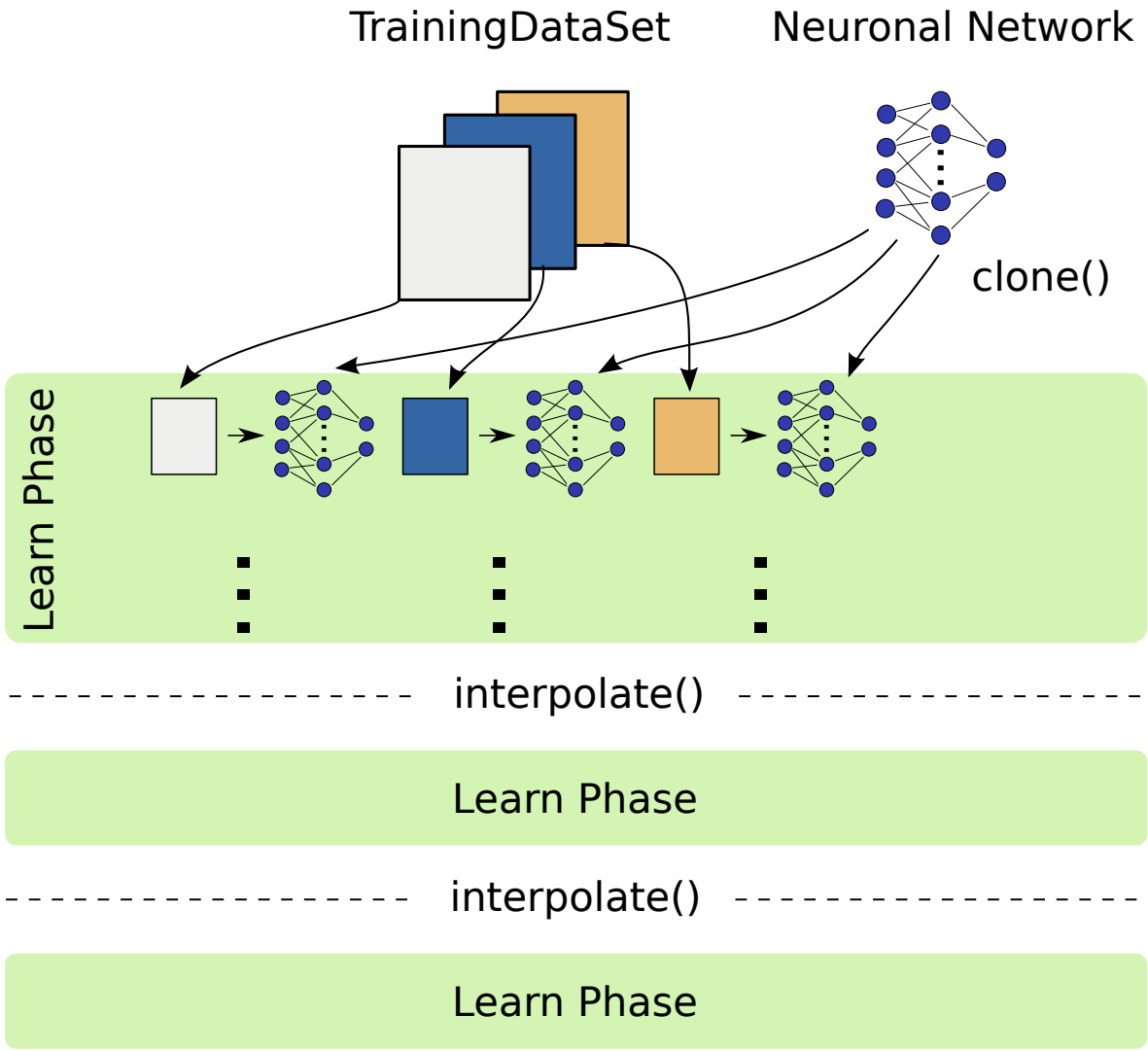
- Container für ein neuronales Netz mit überwachter Lernmethodik
- Unterteilt Trainingssatz und erlernt diese Teile parallel
- Abweichende Ergebnisgewichte verglichen mit sequentiellem Lernen

Clonebased Parallelisierung II

Vorgehen

1. Aufteilung des Trainingssatzes auf Worker-Threads
2. Jeder Worker klont sich das Original-Netz
3. Jeder Worker lernt eine Iteration
4. Gewichts-Interpolation unter allen Worker-Klonnetzen
5. *goto* (3) bis maximale Iteration erreicht oder Fehler klein genug

Clonebased Parallelisierung III



Clonebased Parallelisierung IV

Interpolationstypen

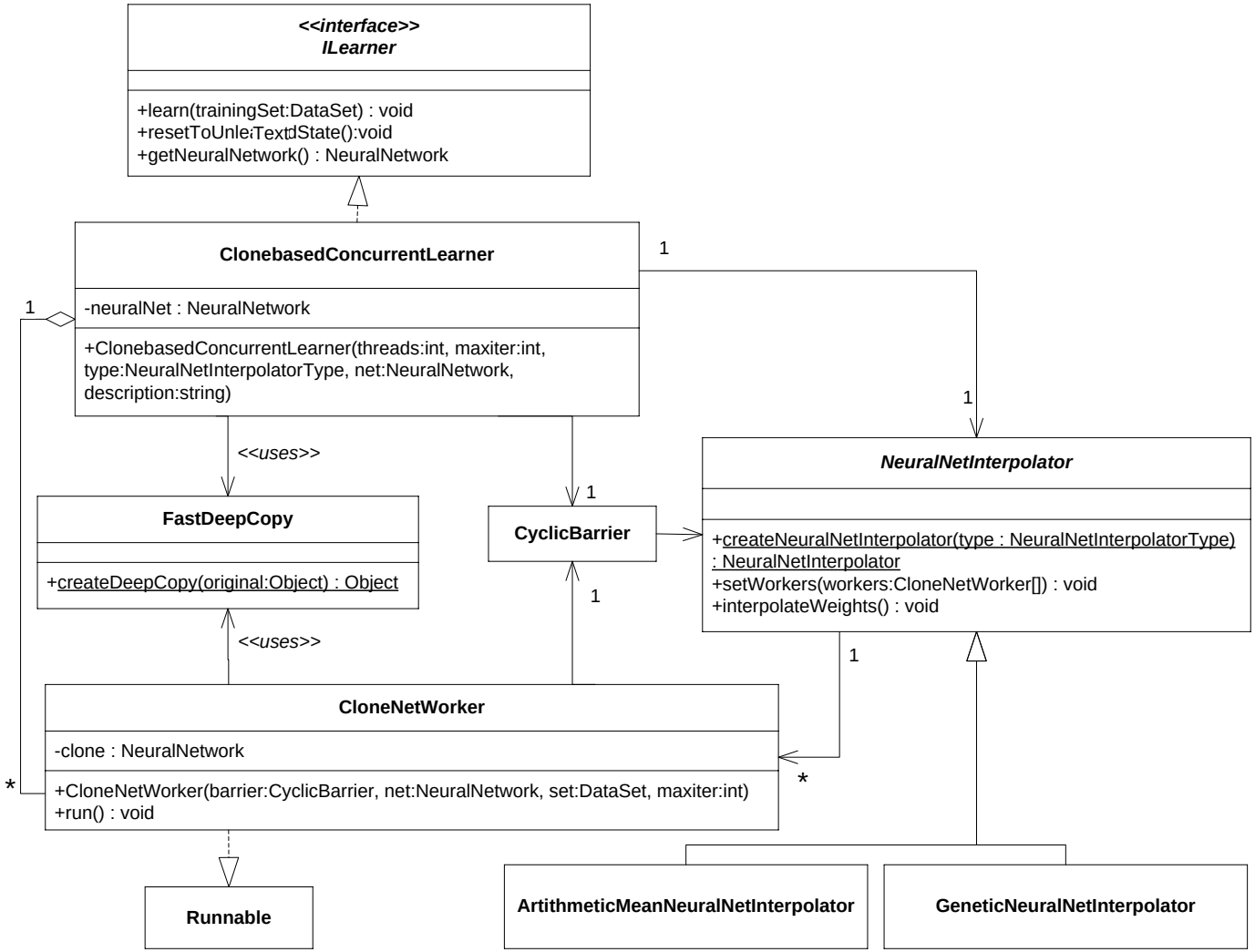
$w = f(w_1, w_2, w_3, \ldots, w_n)$

Gesucht: gute Interpolationsfunktion f

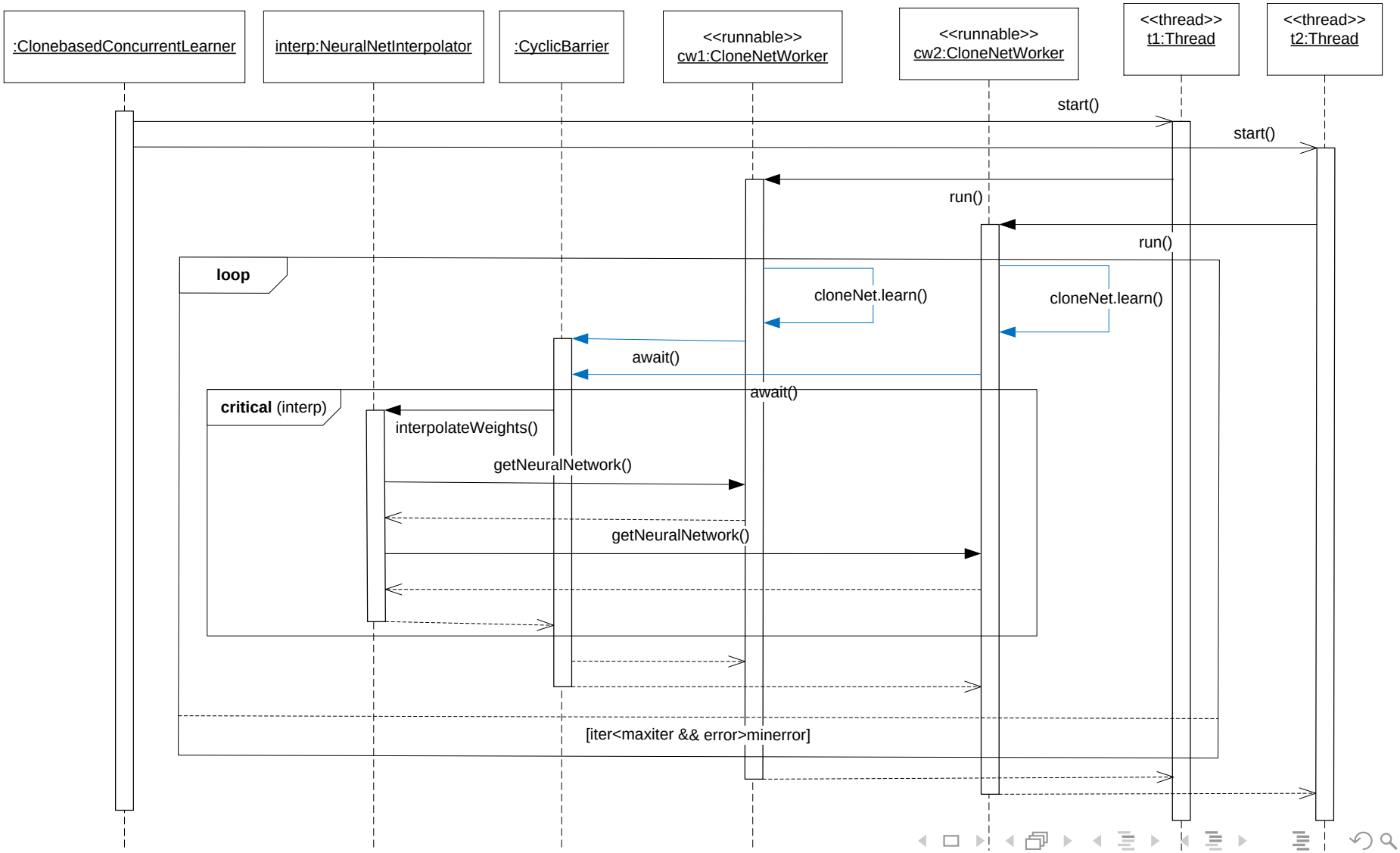
Kandidaten:

- Arithmetisches Mittel
- Minimum, Maximum
- Bestes bisheriges Gewicht *("Genetischer Ansatz")*

Clonebased Parallelisierung V



Clonebased Parallelisierung VI



Technischer Vergleich der Strategien

Klonbasiertes Lernen

- Kapselt vorhandene Funktionialität
  - Verwendung mit verschiedenen Lernstrategien und Netzen möglich

Paralleles Batchlernen

- Neue LearningRule in Vererbungshierarchie
  - saubere Schnittstelle, einfache Verwendung

Kategorie	Layerpartitionierung	Batch	Clonebased
Was ist parallel?	Netzwerk	Lernen	Lernen
Selbes Ergebnis?	ja	ja (batch)	nein
Integration	Modifikation	Erweiterung	Kapselung

Technischer Vergleich der Strategien

Klonbasiertes Lernen

- Kapselt vorhandene Funktionialität
  - Verwendung mit verschiedenen Lernstrategien und Netzen möglich

Paralleles Batchlernen

- Neue LearningRule in Vererbungshierarchie
  - saubere Schnittstelle, einfache Verwendung

Kategorie	Layerpartitionierung	Batch	Clonebased
Was ist parallel?	Netzwerk	Lernen	Lernen
Selbes Ergebnis?	ja	ja (batch)	nein
Integration	Modifikation	Erweiterung	Kapselung

Überblick

- Einleitung ✓
- Neuroph auf Code-Ebene ✓
- Drei Parallelisierungsansätze ✓
- Evaluierung
- Fazit

Testdaten

Erste Versuche

- StockExchange - Börsenvorhersage
- IrisScan Datensatz

Teilchenkollision (Cern)

- 15k Datensätze
- Eingabe: 2853 Sensorwerte
- Ausgabe: Ist das Ereignis interessant oder nicht?

Evaluationsframework I

Score

wird bestimmt durch

- Fehler (auf Testdaten)
- Laufzeit

Evaluationsframework II

Vorgehen

1. Parse Experiment-Konfigurationsdatei
2. Bereite Testläufe vor
3. *repeat*
   1. Permutation der Daten
   2. Aufteilung in Trainings- und Testdaten
   3. *foreach* ILearner L *do*
      1. Lerne Trainingsdaten und messe Ausführungszeiten
      2. Berechne Fehler auf Testdaten
4. *until* alle Durchgänge absolviert

Evaluationsframework III

```
java -jar runexperiment.jar -cf testconfig.txt -o results/ -v -d -csv
```

Experiment-Konfiguration

threads: 4
dataset: data/preparedData/15000rows.txt
training_to_test_ratio: 0.5
input_neurons: 2853
hidden_neurons: 100
output_neurons: 1
runs: 10
max_iteration: 2
sync_frequency:0.25
learners: clonebased-arithmeticmean, clonebased_revised-arithmeticmean, mlp

⇓ Evaluation

NAME;NUMTHREADS;RUNS;AVGER;CONFERLO;CONFERHI;AVGTIME;CONFTIMELO;CONFTIMEHI;SUMER;SUMTIME
ClonebasedConcurrentLearner;4;10;0,179106;0,089431;0,268781;211762;134939;288584;1,791057;2117624
ClonebasedConcurrentLearnerRevised;4;10;0,158935;0,105018;0,212852;223762;187061;260463;1,589349;2237627
NeuralNetworkWrapper;4;10;0,163642;0,064823;0,262460;528646;502374;554917;1,636419;5286460

Ergebnis als CSV

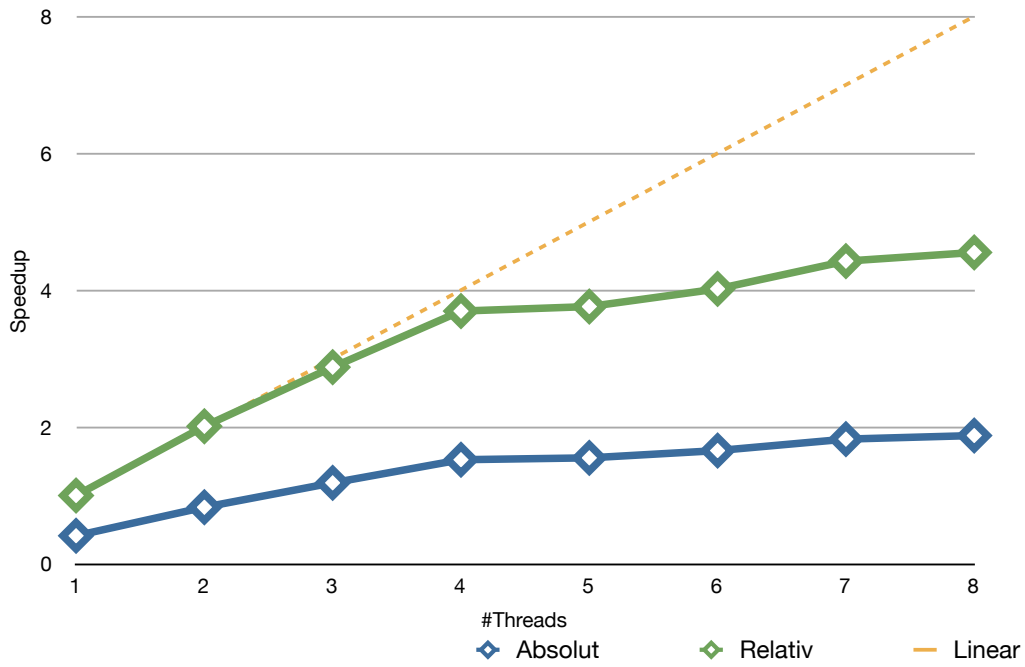
Testaufbau

Poolrechner

- Intel Core2Quad Q6600, 4 Kerne à 2,4 GHz, 8MB L2 Cache
- 8GB RAM
- JDK 7 (`-Xms4096m -Xmx8000m`)

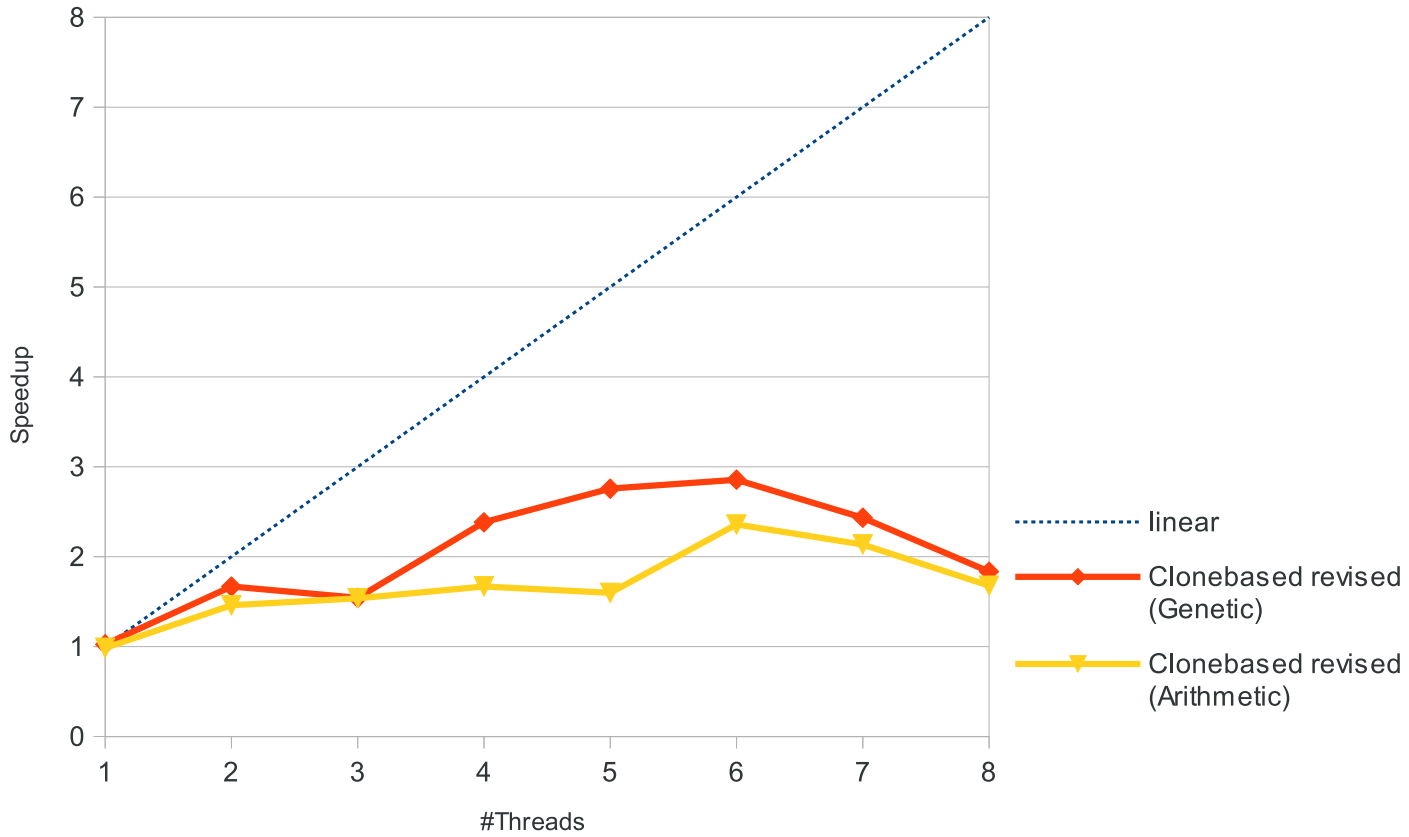
Messwerte: Speedup

- Ausführung des sequentiellen und des parallelen Batch-Learners
- 1000 Datenreihen aus CERN-Testdaten, 1:1 Trainings-/Testdaten, 10 Wiederholungen



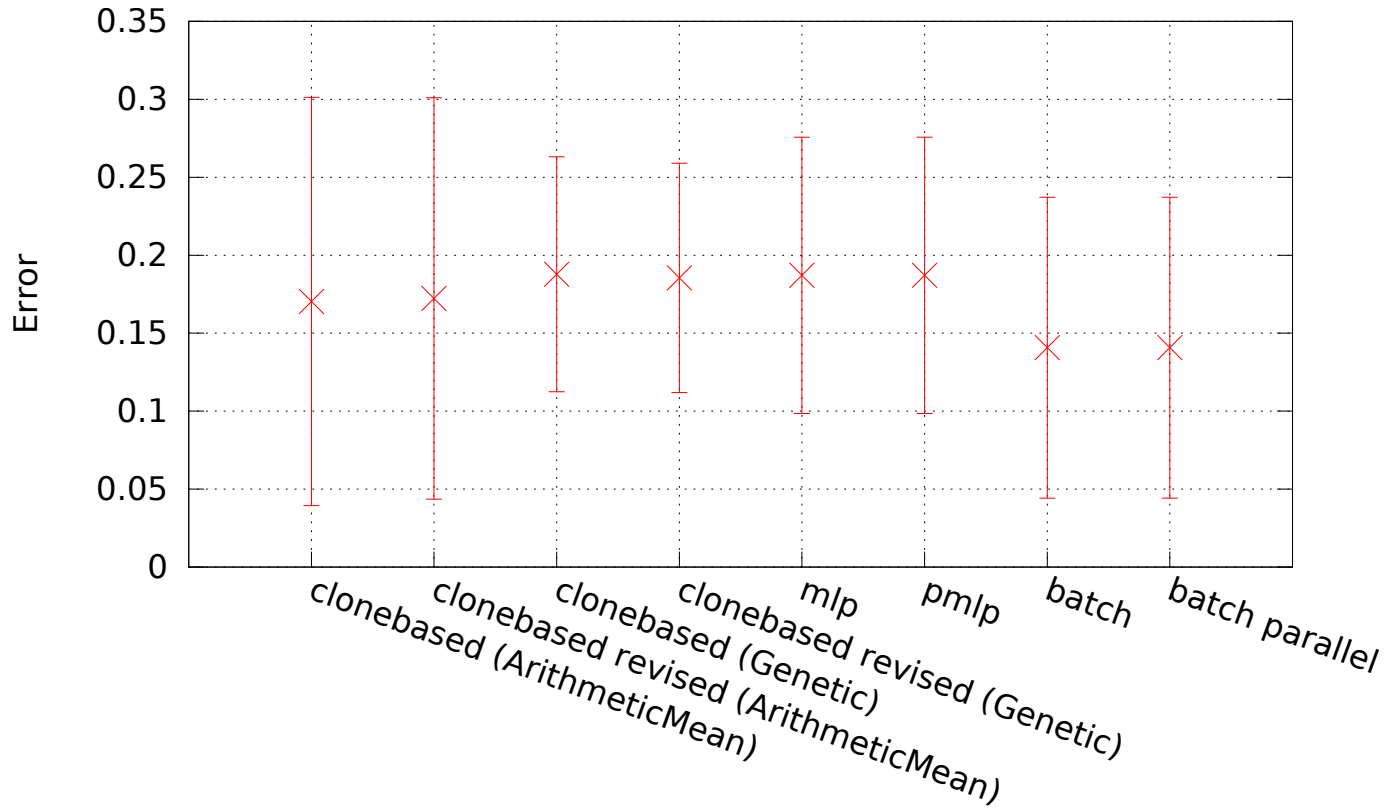
Messwerte: Speedup

- Ausführung des sequentiellen MLPs und der klonbasierten Ansätze
- 5000 Datenreihen aus CERN-Testdaten, 1:1 Trainings-/Testdaten, 3 Wiederholungen



Messwerte: Error

- 1000 Datenreihen aus CERN-Satz, 1:1 Trainings-/Testdaten, 10 Wiederholungen



Überblick

- Einleitung ✓
- Neuroph auf Code-Ebene ✓
- Drei Parallelisierungsansätze ✓
- Evaluierung ✓
- Fazit

Fazit

- Neuronale Netze geeignet für Parallelisierung
- Trivialer Lösungsansatz nicht möglich
- Codebasis (Neuroph) nicht vorbereitet
- Fremden Code bearbeiten sehr aufwändig