Künstliche neuronale Netze

Sebastian Morr

4. Juni 2008

Worum geht es?

„Die Natur hat uns 3,7 Milliarden Jahre Forschungszeit voraus.“ – Toby Simpson

- Vorbild: Strukturen des Gehirns
- Ziel: Lernfähige Künstliche Intelligenz
- Netz künstlicher Nervenzellen
- erzeugt aus Eingabe eine passende Ausgabe

Worum geht es?

„Die Natur hat uns 3,7 Milliarden Jahre Forschungszeit voraus.“ – Toby Simpson

- Vorbild: Strukturen des Gehirns
- Ziel: Lernfähige Künstliche Intelligenz
- Netz künstlicher Nervenzellen
- erzeugt aus Eingabe eine passende Ausgabe

Worum geht es?

„Die Natur hat uns 3,7 Milliarden Jahre Forschungszeit voraus.“ – Toby Simpson

- Vorbild: Strukturen des Gehirns
- Ziel: Lernfähige Künstliche Intelligenz
- Netz künstlicher Nervenzellen
- erzeugt aus Eingabe eine passende Ausgabe

Worum geht es?

„Die Natur hat uns 3,7 Milliarden Jahre Forschungszeit voraus.“ – Toby Simpson

- Vorbild: Strukturen des Gehirns
- Ziel: Lernfähige Künstliche Intelligenz
- Netz künstlicher Nervenzellen
- erzeugt aus Eingabe eine passende Ausgabe

Worum geht es?

„Die Natur hat uns 3,7 Milliarden Jahre Forschungszeit voraus.“ – Toby Simpson

- Vorbild: Strukturen des Gehirns
- Ziel: Lernfähige Künstliche Intelligenz
- Netz künstlicher Nervenzellen
- erzeugt aus Eingabe eine passende Ausgabe

Gliederung

Gliederung

Gliederung

Gliederung

Gliederung

Geschichte

- erste Idee 1943: Warren McCulloch
  - McCulloch-Pitts-Neuron
- Frank Rosenblatt: **Perzeptron** zur Buchstabenerkennung
- John Hopfield: Hopfield-Netz
- Paul Werbos: Lernverfahren Backpropagation
- Inzwischen unüberschaubar. Computereinsatz!

Geschichte

- erste Idee 1943: Warren McCulloch
  - McCulloch-Pitts-Neuron
- Frank Rosenblatt: **Perzeptron** zur Buchstabenerkennung
- John Hopfield: Hopfield-Netz
- Paul Werbos: Lernverfahren Backpropagation
- Inzwischen unüberschaubar. Computereinsatz!

Geschichte

- erste Idee 1943: Warren McCulloch
  - McCulloch-Pitts-Neuron
- Frank Rosenblatt: **Perzeptron** zur Buchstabenerkennung
- John Hopfield: Hopfield-Netz
- Paul Werbos: Lernverfahren Backpropagation
- Inzwischen unüberschaubar. Computereinsatz!

Geschichte

- erste Idee 1943: Warren McCulloch
  - McCulloch-Pitts-Neuron
- Frank Rosenblatt: **Perzeptron** zur Buchstabenerkennung
- John Hopfield: Hopfield-Netz
- Paul Werbos: Lernverfahren Backpropagation
- Inzwischen unüberschaubar. Computereinsatz!

Geschichte

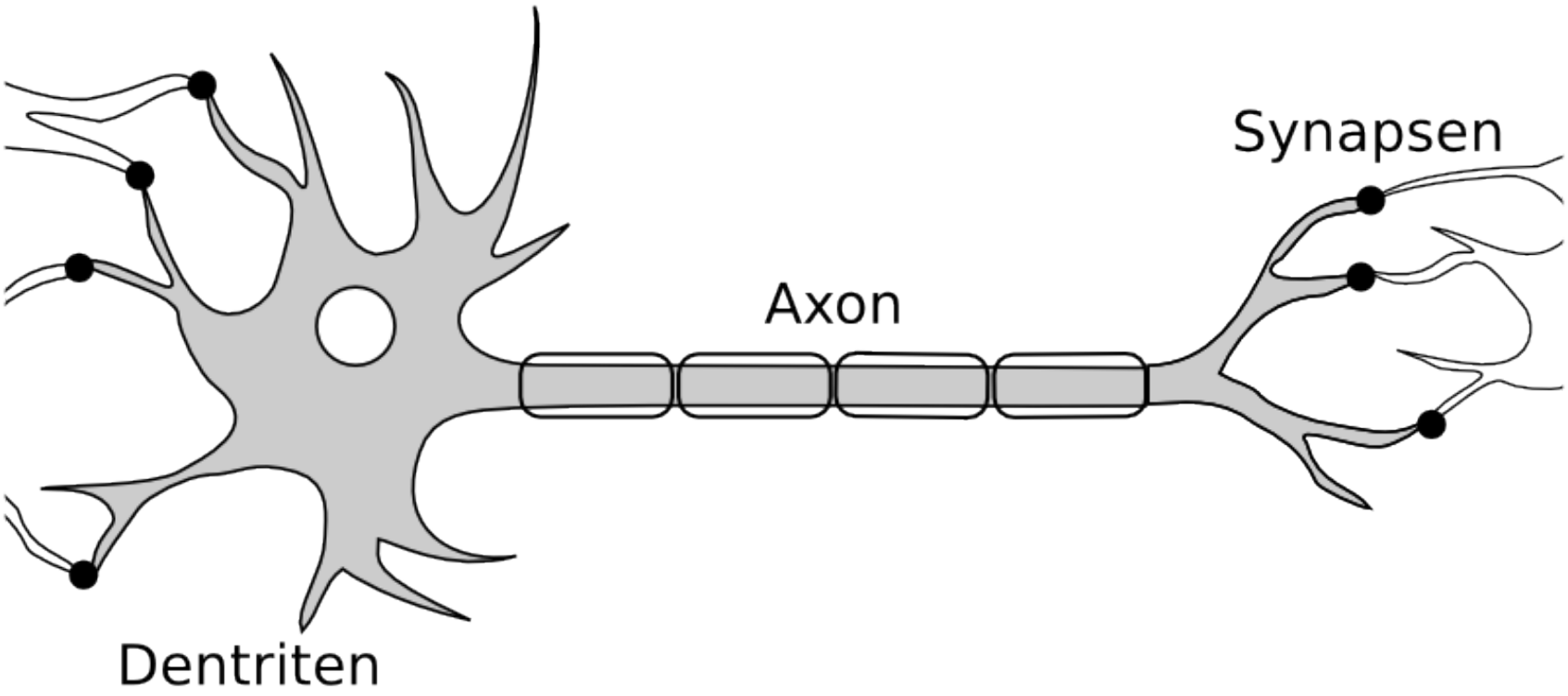
- erste Idee 1943: Warren McCulloch
  - McCulloch-Pitts-Neuron
- Frank Rosenblatt: **Perzeptron** zur Buchstabenerkennung
- John Hopfield: Hopfield-Netz
- Paul Werbos: Lernverfahren Backpropagation
- Inzwischen unüberschaubar. Computereinsatz!

Geschichte

- erste Idee 1943: Warren McCulloch
  - McCulloch-Pitts-Neuron
- Frank Rosenblatt: **Perzeptron** zur Buchstabenerkennung
- John Hopfield: Hopfield-Netz
- Paul Werbos: Lernverfahren Backpropagation
- Inzwischen unüberschaubar. Computereinsatz!

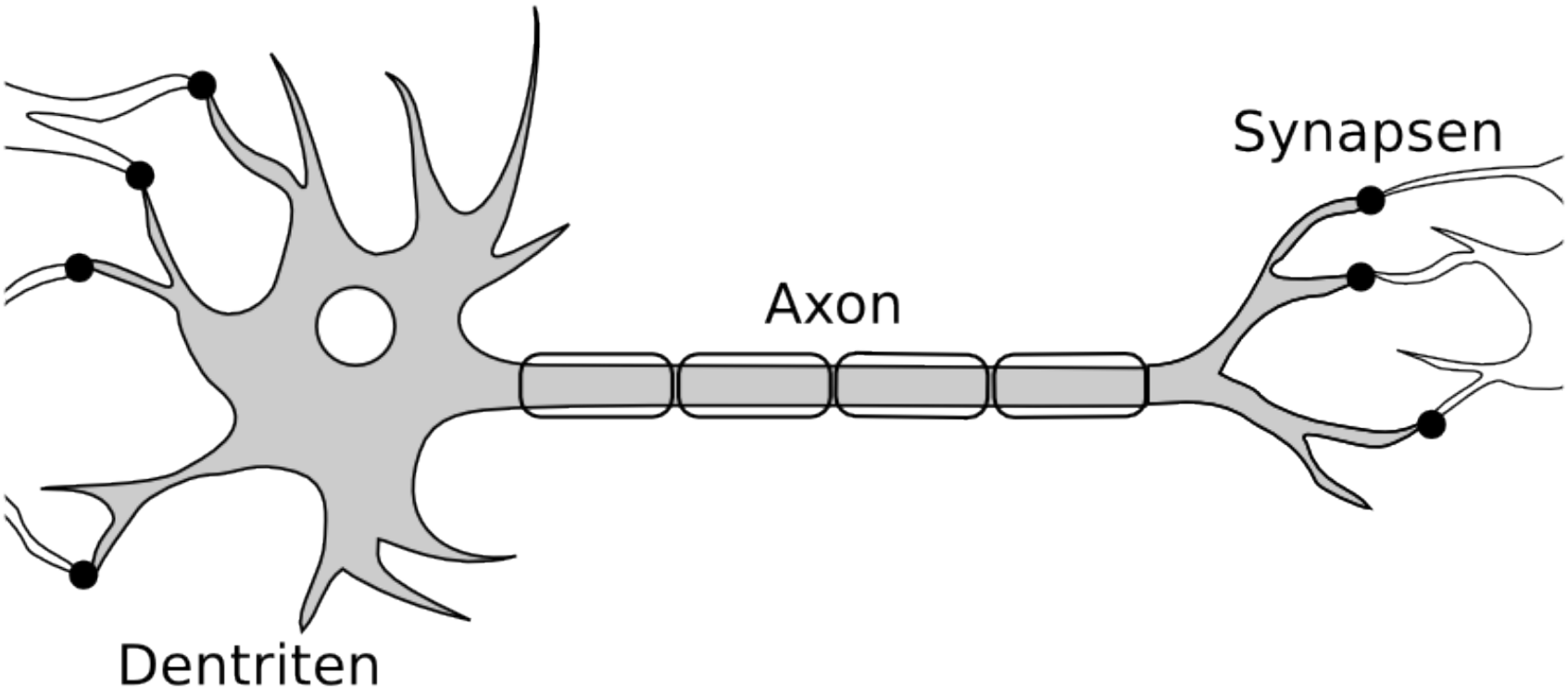
Gliederung

Biologisches Neuron



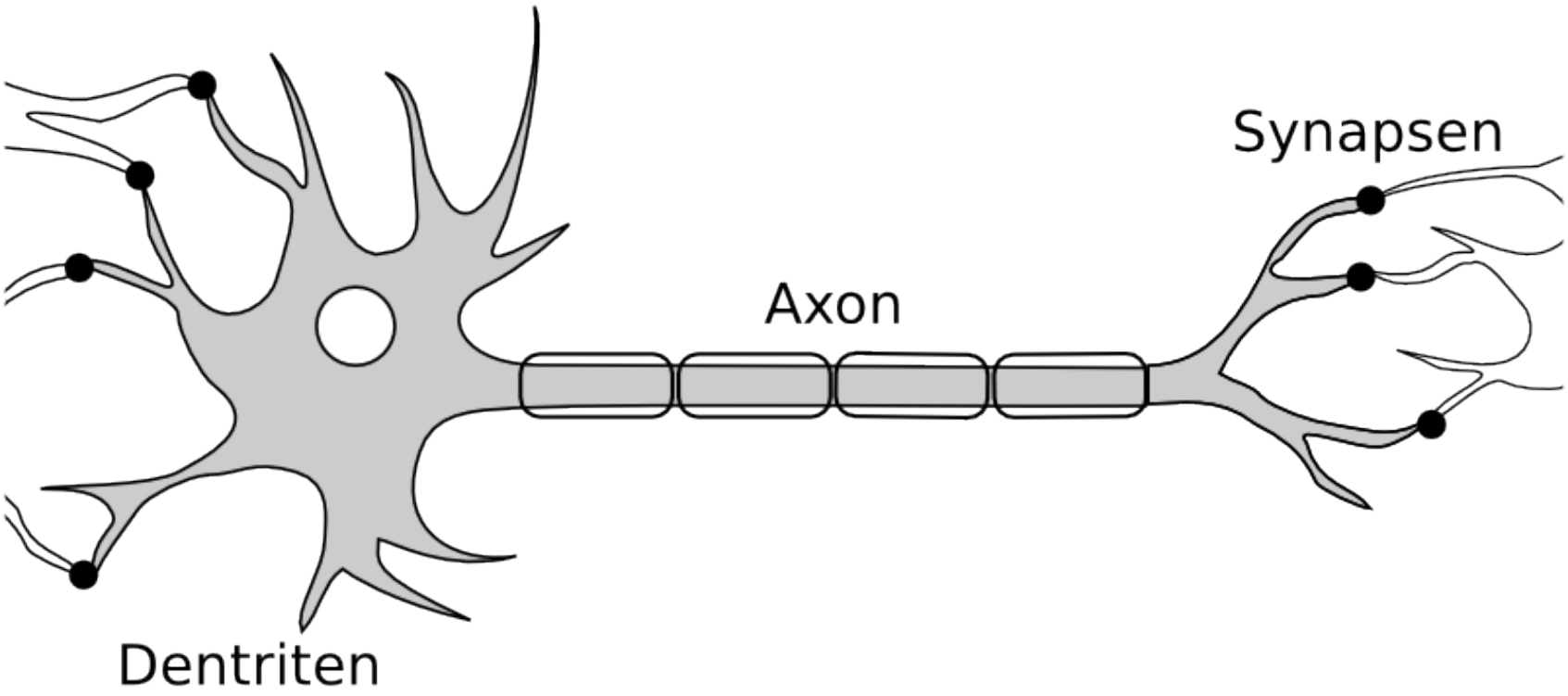
- Grundeinheit eines Nervensystems
- Aufnahme und Weitergabe von Erregungen
- in Sinnesorganen, Rückenmark und Gehirn

Biologisches Neuron



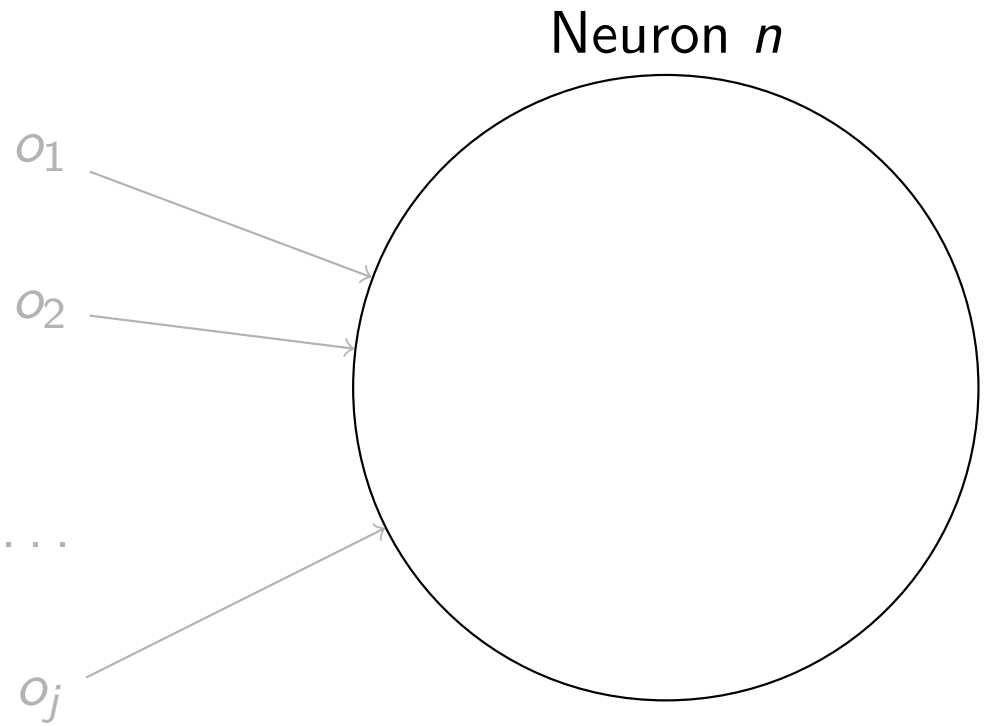
- Grundeinheit eines Nervensystems
- Aufnahme und Weitergabe von Erregungen
- in Sinnesorganen, Rückenmark und Gehirn

Biologisches Neuron



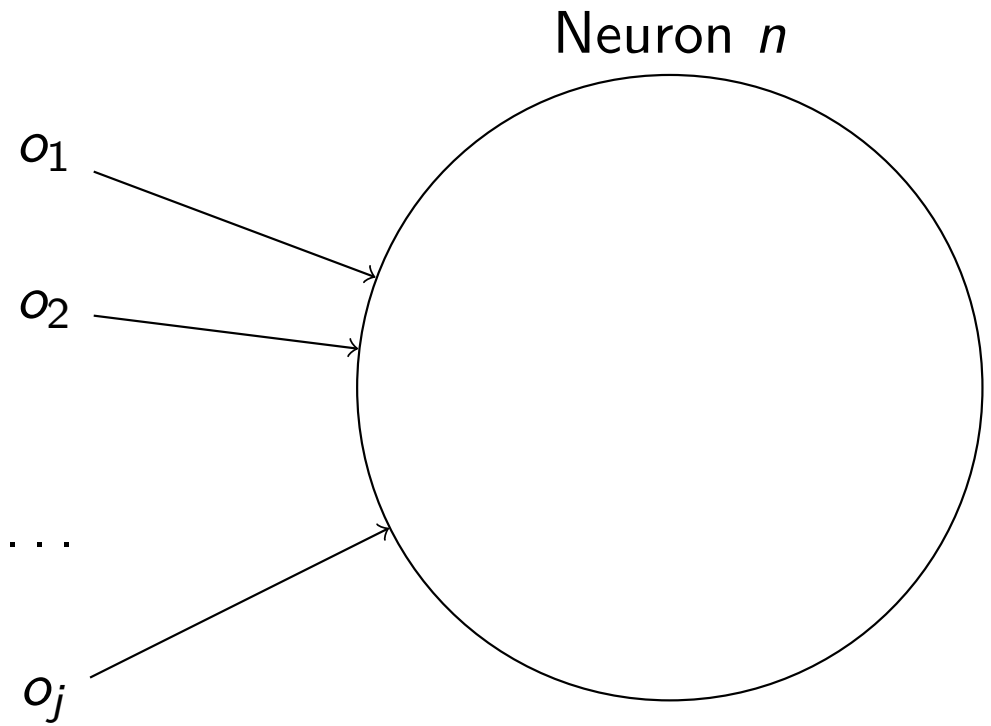
- Grundeinheit eines Nervensystems
- Aufnahme und Weitergabe von Erregungen
- in Sinnesorganen, Rückenmark und Gehirn

Künstliches Neuron



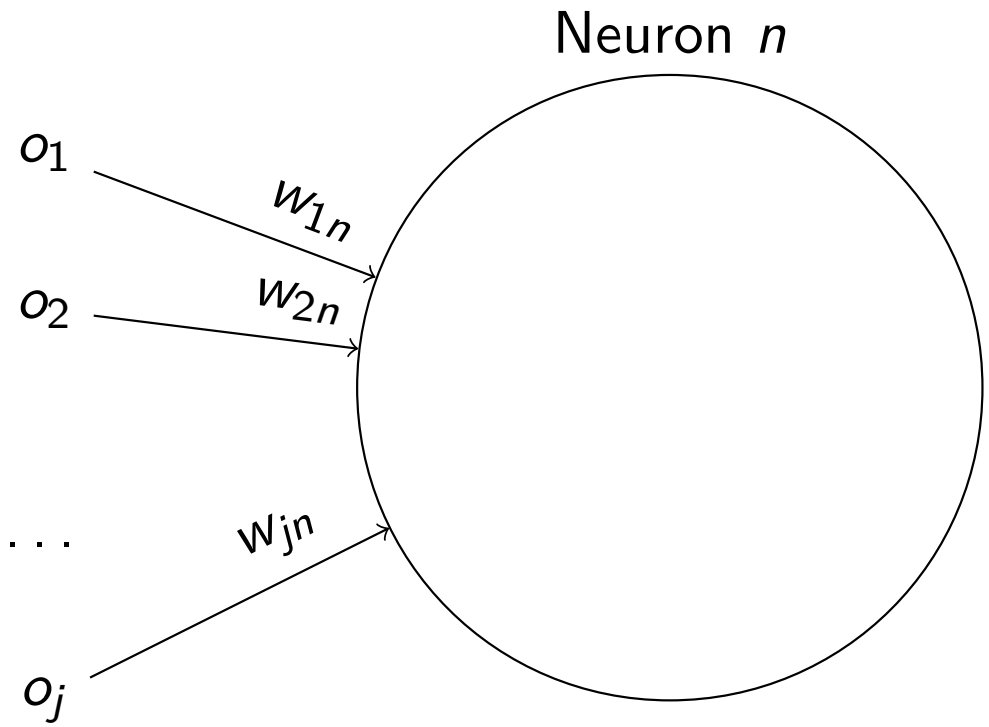
- Eingabeinformationen o_1, o_2, ...
- Gewichte w_{1n}, w_{2n}, ...
- Propagierungsfunktion net_n addiert Eingaben
- Aktivierungsfunktion f_{act}

Künstliches Neuron



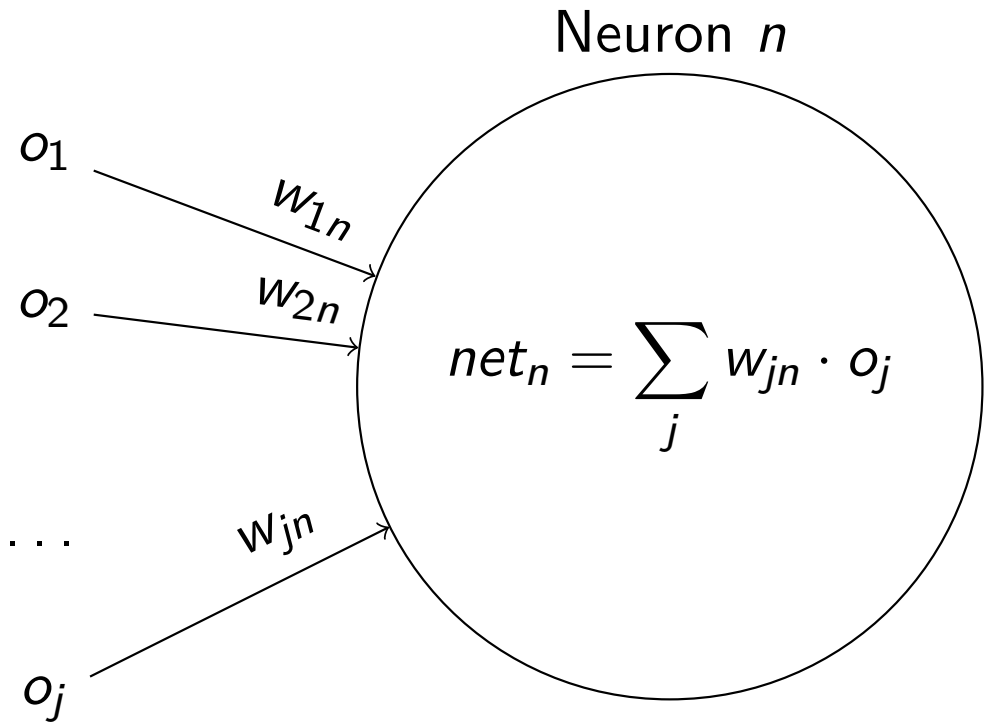
- Eingabeinformationen o_1, o_2, ...
- Gewichte w_{1n}, w_{2n}, ...
- Propagierungsfunktion net_n addiert Eingaben
- Aktivierungsfunktion f_{act}

Künstliches Neuron



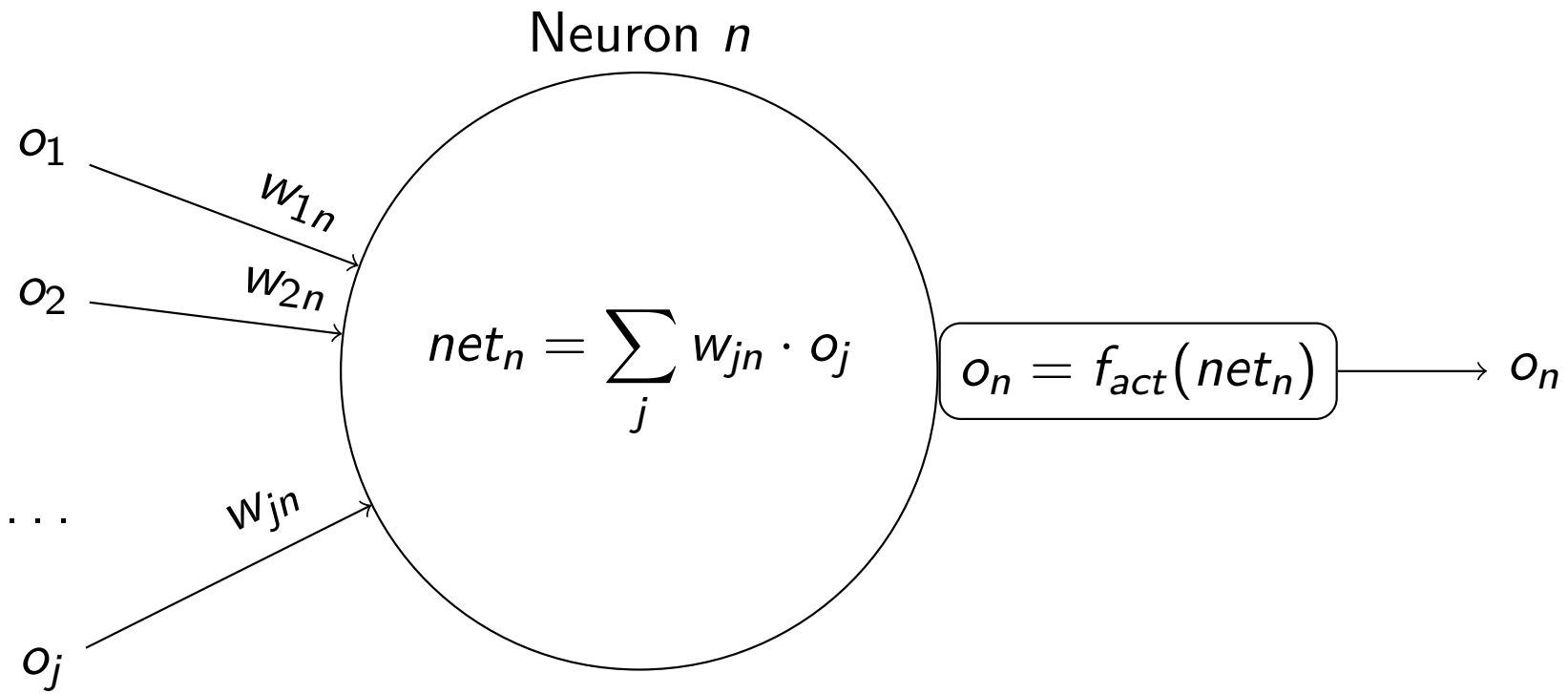
- Eingabeinformationen o_1, o_2, ...
- Gewichte w_{1n}, w_{2n}, ...
- Propagierungsfunktion net_n addiert Eingaben
- Aktivierungsfunktion f_{act}

Künstliches Neuron



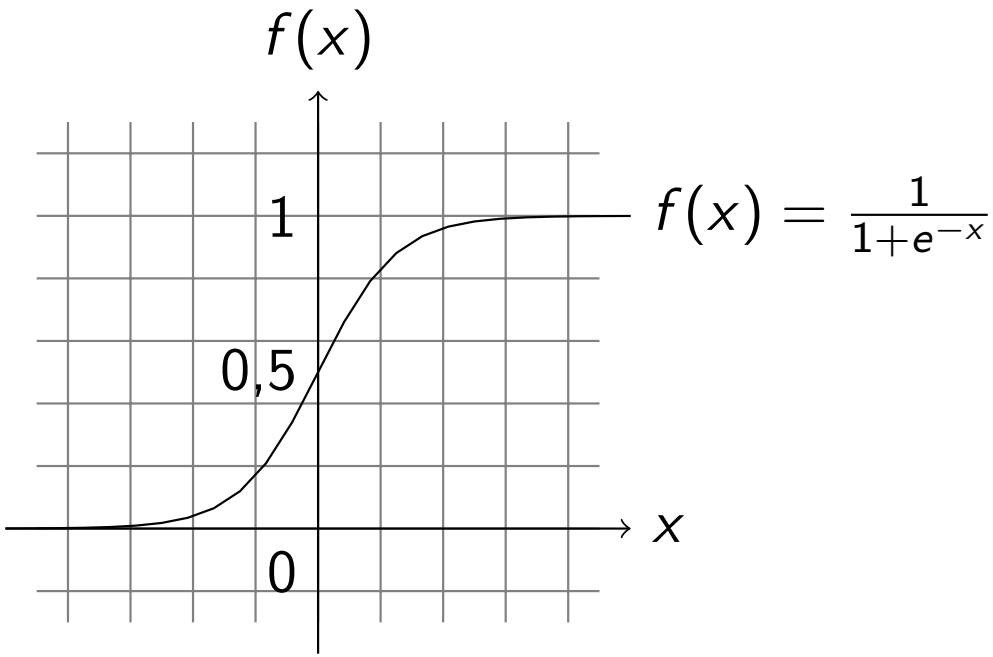
- Eingabeinformationen o_1, o_2, ...
- Gewichte w_{1n}, w_{2n}, ...
- Propagierungsfunktion net_n addiert Eingaben
- Aktivierungsfunktion f_{act}

Künstliches Neuron



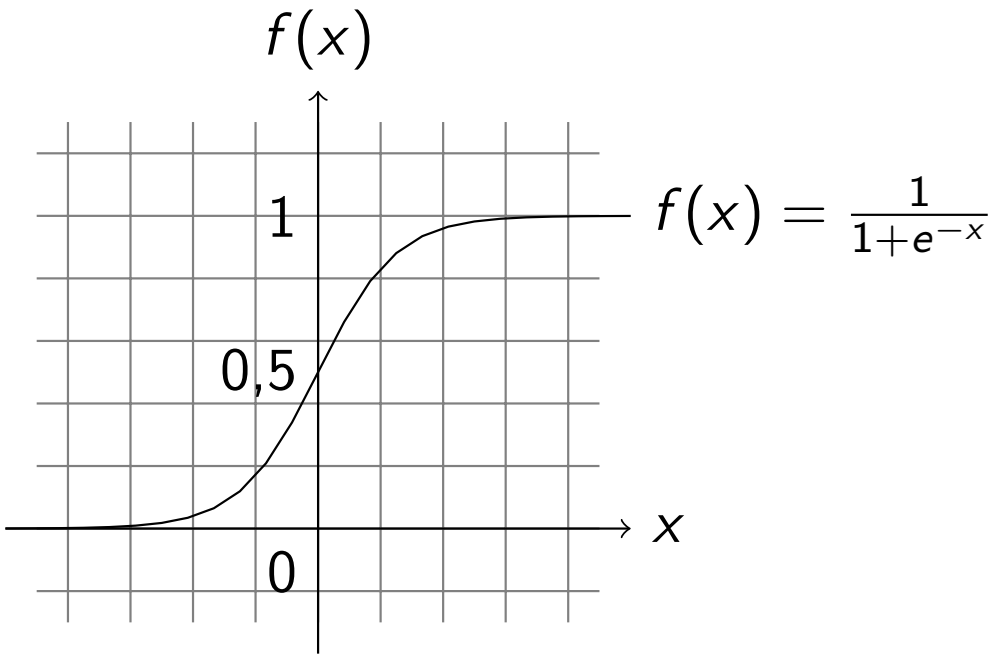
- Eingabeinformationen o_1, o_2, ...
- Gewichte w_{1n}, w_{2n}, ...
- Propagierungsfunktion net_n addiert Eingaben
- Aktivierungsfunktion f_{act}

Künstliches Neuron



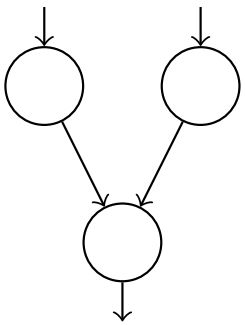
- **sigmoide** Aktivierungsfunktion
- z.B.: Logistische Funktion oder Tangens Hyperbolicus

Künstliches Neuron



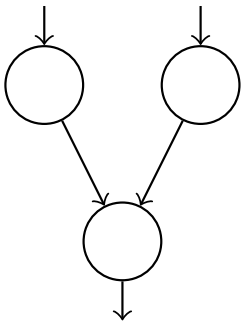
- **sigmoide** Aktivierungsfunktion
- z.B.: Logistische Funktion oder Tangens Hyperbolicus

Allgemeines



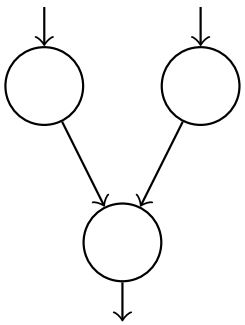
- Verknüpfung der Neuronen zum Netz
- Eingabe
- Verarbeitung
- Ausgabe

Allgemeines



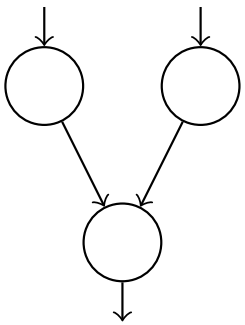
- Verknüpfung der Neuronen zum Netz
- Eingabe
- Verarbeitung
- Ausgabe

Allgemeines



- Verknüpfung der Neuronen zum Netz
- Eingabe
- Verarbeitung
- Ausgabe

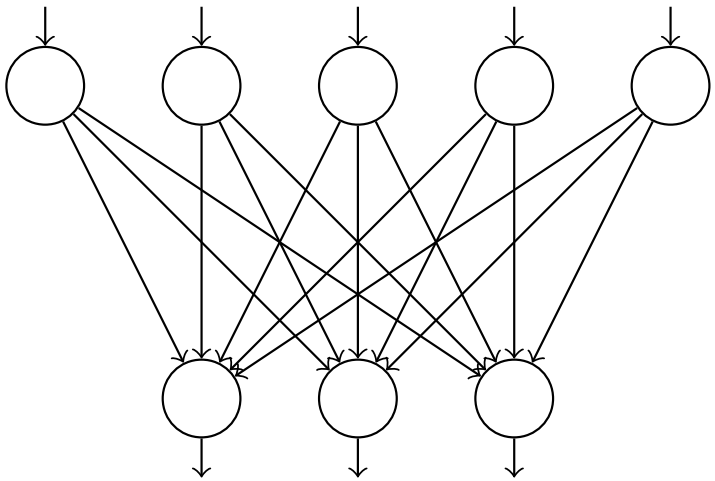
Allgemeines



- Verknüpfung der Neuronen zum Netz
- Eingabe
- Verarbeitung
- Ausgabe

Perzeptron

Eingabeschicht

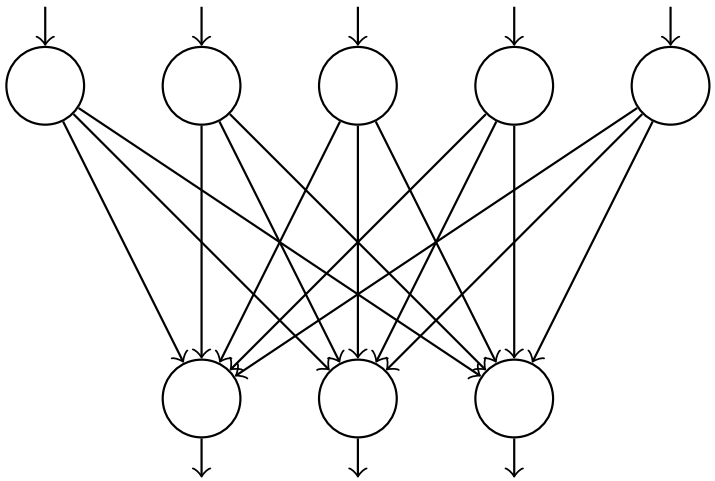


Ausgabeschicht

- einfachste Netzarchitektur
- zwei vollvernetzte Schichten
- von *percept*, „wahrnehmen“

Perzeptron

Eingabeschicht

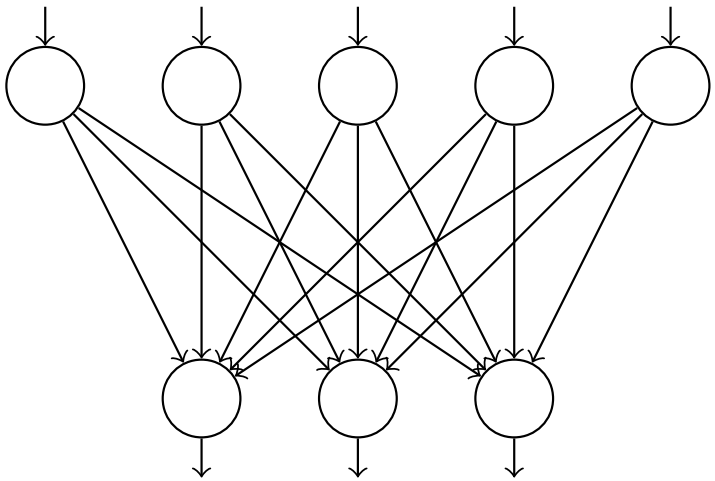


Ausgabeschicht

- einfachste Netzarchitektur
- zwei vollvernetzte Schichten
- von *percept*, „wahrnehmen“

Perzeptron

Eingabeschicht



Ausgabeschicht

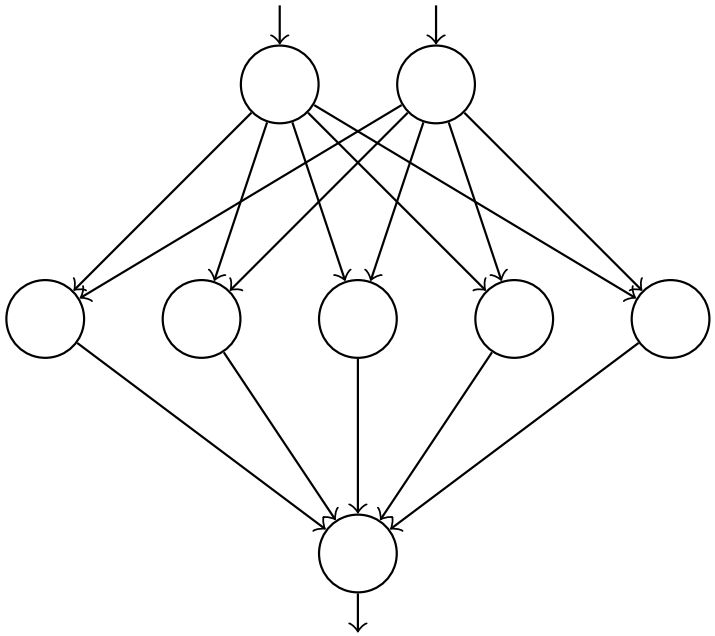
- einfachste Netzarchitektur
- zwei vollvernetzte Schichten
- von *percept*, „wahrnehmen“

Mehrschichtige vorwärts verkettete Netze

Eingabeschicht

Innere Schicht

Ausgabeschicht



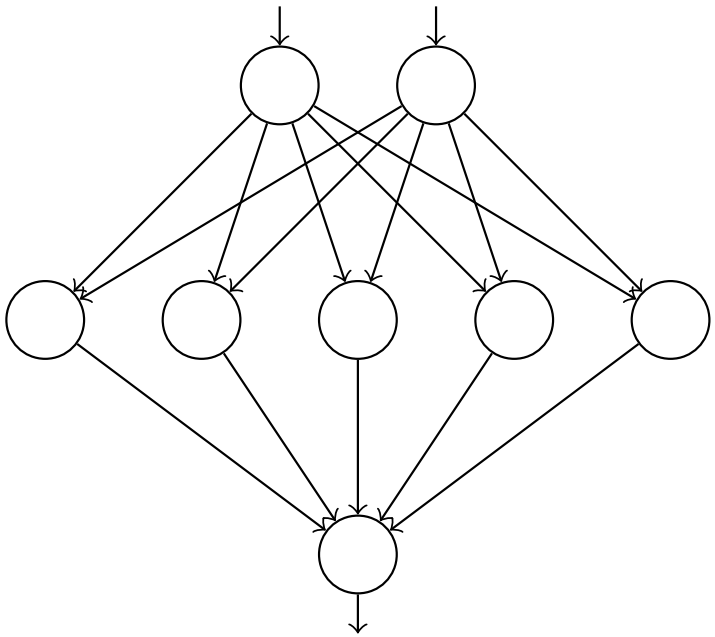
- für komplexere Aufgaben
- zusätzliche **innere Schicht**

Mehrschichtige vorwärts verkettete Netze

Eingabeschicht

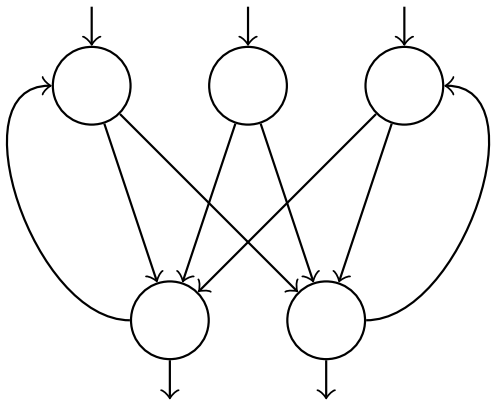
Innere Schicht

Ausgabeschicht



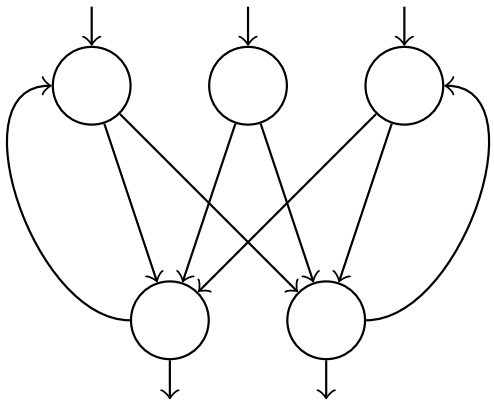
- für komplexere Aufgaben
- zusätzliche **innere Schicht**

Sonstige Netztypen



- **Rückgekoppelte Netze** mit „Erinnerungsvermögen“
- **Selbstorganisierende Karten** lernen eigenständig

Sonstige Netztypen



- **Rückgekoppelte Netze** mit „Erinnerungsvermögen“
- **Selbstorganisierende Karten** lernen eigenständig

Unterteilung

- **überwachtes Lernen**: Vergleich mit Trainingsdaten
- **bestärkendes Lernen**: Unterscheidung richtig/falsch
- **unüberwachtes Lernen**: eigenständiges Erarbeiten von Regeln

Unterteilung

- **überwachtes Lernen**: Vergleich mit Trainingsdaten
- **bestärkendes Lernen**: Unterscheidung richtig/falsch
- **unüberwachtes Lernen**: eigenständiges Erarbeiten von Regeln

Unterteilung

- **überwachtes Lernen**: Vergleich mit Trainingsdaten
- **bestärkendes Lernen**: Unterscheidung richtig/falsch
- **unüberwachtes Lernen**: eigenständiges Erarbeiten von Regeln

Backpropagation

- verbreiteter Algorithmus zum überwachten Lernen
- zwei Durchläufe: Bestimmung der Ausgabe, dann Vergleich und Anpassung
- Fehlersignal:

$$\delta_n = (t_n - o_n) \cdot f'_{act}(net_n)$$

bzw.

$$\delta_n = \sum_o (\delta_o \cdot w_{no}) \cdot f'_{act}(net_n)$$

- Anpassung der Gewichte:

$$\Delta w_{mn} = \delta_n \cdot o_n \cdot \eta$$

Backpropagation

- verbreiteter Algorithmus zum überwachten Lernen
- zwei Durchläufe: Bestimmung der Ausgabe, dann Vergleich und Anpassung
- Fehlersignal:

$$\delta_n = (t_n - o_n) \cdot f'_{act}(net_n)$$

bzw.

$$\delta_n = \sum_o (\delta_o \cdot w_{no}) \cdot f'_{act}(net_n)$$

- Anpassung der Gewichte:

$$\Delta w_{mn} = \delta_n \cdot o_n \cdot \eta$$

Backpropagation

- verbreiteter Algorithmus zum überwachten Lernen
- zwei Durchläufe: Bestimmung der Ausgabe, dann Vergleich und Anpassung
- Fehlersignal:

$$\delta_n = (t_n - o_n) \cdot f'_{act}(net_n)$$

bzw.

$$\delta_n = \sum_o (\delta_o \cdot w_{no}) \cdot f'_{act}(net_n)$$

- Anpassung der Gewichte:

$$\Delta w_{mn} = \delta_n \cdot o_n \cdot \eta$$

Backpropagation

- verbreiteter Algorithmus zum überwachten Lernen
- zwei Durchläufe: Bestimmung der Ausgabe, dann Vergleich und Anpassung
- Fehlersignal:

$$\delta_n = (t_n - o_n) \cdot f'_{act}(net_n)$$

bzw.

$$\delta_n = \sum_o (\delta_o \cdot w_{no}) \cdot f'_{act}(net_n)$$

- Anpassung der Gewichte:

$$\Delta w_{mn} = \delta_n \cdot o_n \cdot \eta$$

Gliederung

Anwendungsbereiche

- Mustererkennung und -zuordnung (Schrift, Sprache, Bilder)
- Sensoren (Geruchssensoren)
- Vorhersage (Wetter, Aktien, Medizin)
- Steuerung (Roboter, Autopiloten)
- Computerspiele (*Creatures*)

Anwendungsbereiche

- Mustererkennung und -zuordnung (Schrift, Sprache, Bilder)
- Sensoren (Geruchssensoren)
- Vorhersage (Wetter, Aktien, Medizin)
- Steuerung (Roboter, Autopiloten)
- Computerspiele (*Creatures*)

Anwendungsbereiche

- Mustererkennung und -zuordnung (Schrift, Sprache, Bilder)
- Sensoren (Geruchssensoren)
- Vorhersage (Wetter, Aktien, Medizin)
- Steuerung (Roboter, Autopiloten)
- Computerspiele (*Creatures*)

Anwendungsbereiche

- Mustererkennung und -zuordnung (Schrift, Sprache, Bilder)
- Sensoren (Geruchssensoren)
- Vorhersage (Wetter, Aktien, Medizin)
- Steuerung (Roboter, Autopiloten)
- Computerspiele (*Creatures*)

Anwendungsbereiche

- Mustererkennung und -zuordnung (Schrift, Sprache, Bilder)
- Sensoren (Geruchssensoren)
- Vorhersage (Wetter, Aktien, Medizin)
- Steuerung (Roboter, Autopiloten)
- Computerspiele (*Creatures*)

Vorteile

- Lernfähigkeit
- Generalisierungsfähigkeit
- Fehlertoleranz
- Anpassungsfähigkeit
- Modellfreiheit

Vorteile

- Lernfähigkeit
- Generalisierungsfähigkeit
- Fehlertoleranz
- Anpassungsfähigkeit
- Modellfreiheit

Vorteile

- Lernfähigkeit
- Generalisierungsfähigkeit
- Fehlertoleranz
- Anpassungsfähigkeit
- Modellfreiheit

Vorteile

- Lernfähigkeit
- Generalisierungsfähigkeit
- Fehlertoleranz
- Anpassungsfähigkeit
- Modellfreiheit

Vorteile

- Lernfähigkeit
- Generalisierungsfähigkeit
- Fehlertoleranz
- Anpassungsfähigkeit
- Modellfreiheit

Nachteile

- Überanpassung
- Ineffektivität
- keine Garantie auf das bestmögliche Ergebnis

Nachteile

- Überanpassung
- Ineffektivität
- keine Garantie auf das bestmögliche Ergebnis

Nachteile

- Überanpassung
- Ineffektivität
- keine Garantie auf das bestmögliche Ergebnis

Gliederung

Ziel

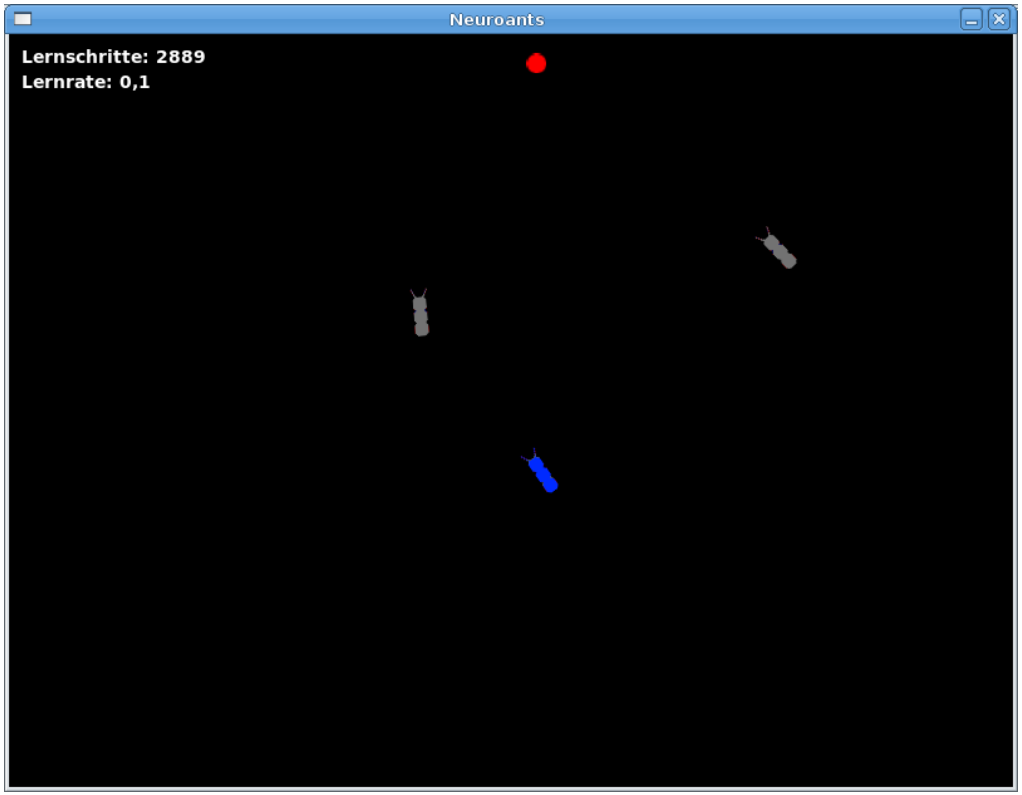
- Bibliothek + Beispielanwendung
- trainierbare virtuelle Agenten
- $\longrightarrow$ Ameisensimulation!

Ziel

- Bibliothek + Beispielanwendung
- trainierbare virtuelle Agenten
- $\longrightarrow$ Ameisensimulation!

Ziel

- Bibliothek + Beispielanwendung
- trainierbare virtuelle Agenten
- $\longrightarrow$ Ameisensimulation!



Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:

```
net = Net.new [2, 4, 3]
net.train [3.14, 42], [0, 0, 1]
output = net.eval [1, 2]
=> [0.168, 0.171, 0.829]
```

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:
  - `net = Net.new [2, 4, 3]`
  - `net.train [3.14, 42], [0, 0, 1]`
  - `output = net.eval [1, 2]`
  - `=> [0.168, 0.171, 0.829]`

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:

```
net = Net.new [2, 4, 3]
net.train [3.14, 42], [0, 0, 1]
output = net.eval [1, 2]
=> [0.168, 0.171, 0.829]
```

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:
  - `net = Net.new [2, 4, 3]`
  - `net.train [3.14, 42], [0, 0, 1]`
  - `output = net.eval [1, 2]`
  - `=> [0.168, 0.171, 0.829]`

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:

```
net = Net.new [2, 4, 3]
net.train [3.14, 42], [0, 0, 1]
output = net.eval [1, 2]
=> [0.168, 0.171, 0.829]
```

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:
  - `net = Net.new [2, 4, 3]`
  - `net.train [3.14, 42], [0, 0, 1]`
  - `output = net.eval [1, 2]`
  - `=> [0.168, 0.171, 0.829]`

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:
  - `net = Net.new [2, 4, 3]`
  - `net.train [3.14, 42], [0, 0, 1]`
  - `output = net.eval [1, 2]`
  - `=> [0.168, 0.171, 0.829]`

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:
  - `net = Net.new [2, 4, 3]`
  - `net.train [3.14, 42], [0, 0, 1]`
  - `output = net.eval [1, 2]`
  - `=> [0.168, 0.171, 0.829]`

Bibliothek

- programmiert in **Ruby**
  - objektorientiert
  - leicht schreib- und lesbar
  - freie Lizenz („Open Source“)
- Anwenung in 3 Schritten:
  - `net = Net.new [2, 4, 3]`
  - `net.train [3.14, 42], [0, 0, 1]`
  - `output = net.eval [1, 2]`
  - `=> [0.168, 0.171, 0.829]`

Neuroants

- Netz für jede „Ameise“
- Eingabe: Position des Zieles (2 Neuronen)
- Ausgabe: Bewegungsgeschwindigkeit und Drehung (2 Neuronen)
- innere Schicht mit 5 Neuronen
- Lernmechanismus: Backpropagation-Verfahren
- Steuerung des Benutzers gibt gewünschtes Verhalten vor

Neuroants

- Netz für jede „Ameise“
- Eingabe: Position des Zieles (2 Neuronen)
- Ausgabe: Bewegungsgeschwindigkeit und Drehung (2 Neuronen)
- innere Schicht mit 5 Neuronen
- Lernmechanismus: Backpropagation-Verfahren
- Steuerung des Benutzers gibt gewünschtes Verhalten vor

Neuroants

- Netz für jede „Ameise“
- Eingabe: Position des Zieles (2 Neuronen)
- Ausgabe: Bewegungsgeschwindigkeit und Drehung (2 Neuronen)
- innere Schicht mit 5 Neuronen
- Lernmechanismus: Backpropagation-Verfahren
- Steuerung des Benutzers gibt gewünschtes Verhalten vor

Neuroants

- Netz für jede „Ameise“
- Eingabe: Position des Zieles (2 Neuronen)
- Ausgabe: Bewegungsgeschwindigkeit und Drehung (2 Neuronen)
- innere Schicht mit 5 Neuronen
- Lernmechanismus: Backpropagation-Verfahren
- Steuerung des Benutzers gibt gewünschtes Verhalten vor

Neuroants

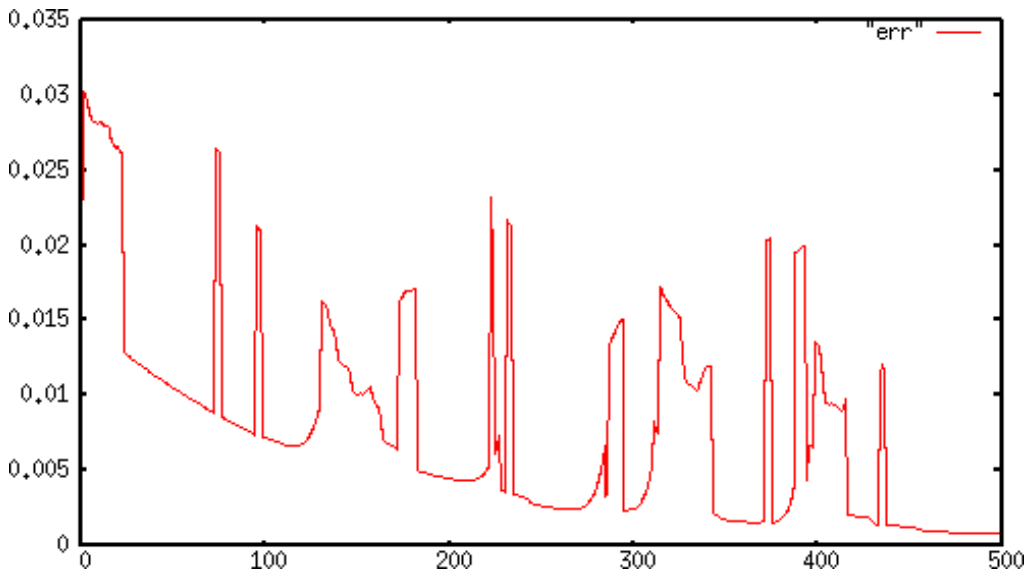
- Netz für jede „Ameise“
- Eingabe: Position des Zieles (2 Neuronen)
- Ausgabe: Bewegungsgeschwindigkeit und Drehung (2 Neuronen)
- innere Schicht mit 5 Neuronen
- Lernmechanismus: Backpropagation-Verfahren
- Steuerung des Benutzers gibt gewünschtes Verhalten vor

Neuroants

- Netz für jede „Ameise“
- Eingabe: Position des Zieles (2 Neuronen)
- Ausgabe: Bewegungsgeschwindigkeit und Drehung (2 Neuronen)
- innere Schicht mit 5 Neuronen
- Lernmechanismus: Backpropagation-Verfahren
- Steuerung des Benutzers gibt gewünschtes Verhalten vor

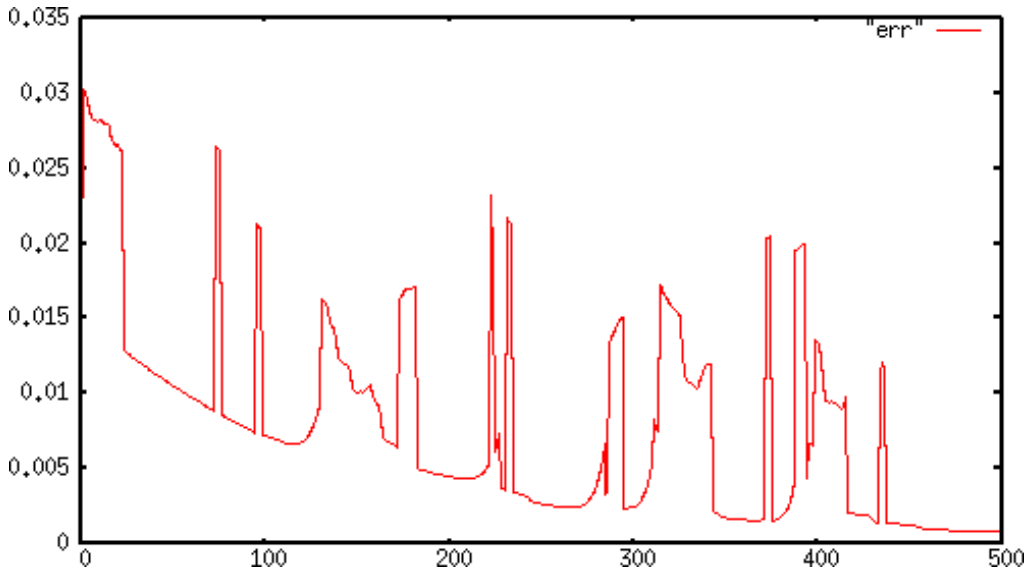
Resultat

- Trainigsmöglichkeiten:
  - Futter vorwärts- oder rückwärts einsammeln
  - Futter umkreisen
  - auf der Stelle drehen
  - nur Rechtskurven benutzen
  - Bewegung in Schleifen
- *ausprobieren!*



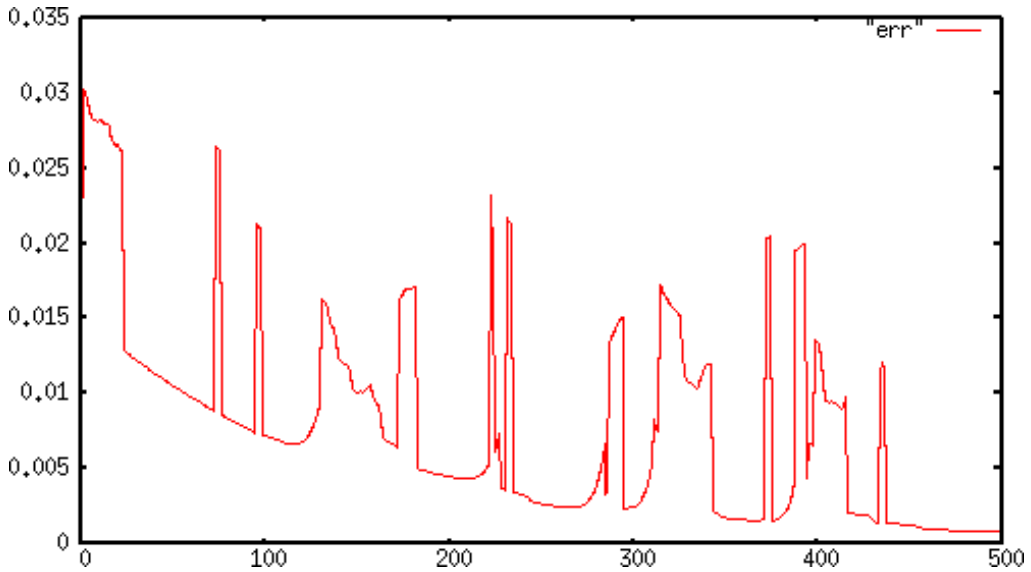
Resultat

- Trainigsmöglichkeiten:
  - Futter vorwärts- oder rückwärts einsammeln
  - Futter umkreisen
  - auf der Stelle drehen
  - nur Rechtskurven benutzen
  - Bewegung in Schleifen
- *ausprobieren!*



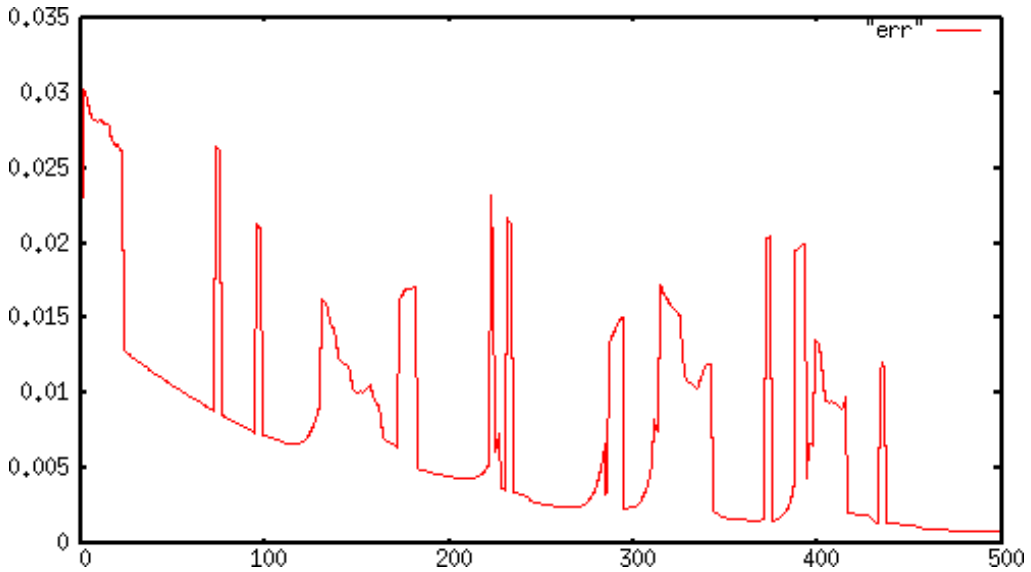
Resultat

- Trainigsmöglichkeiten:
  - Futter vorwärts- oder rückwärts einsammeln
  - Futter umkreisen
  - auf der Stelle drehen
  - nur Rechtskurven benutzen
  - Bewegung in Schleifen
- *ausprobieren!*



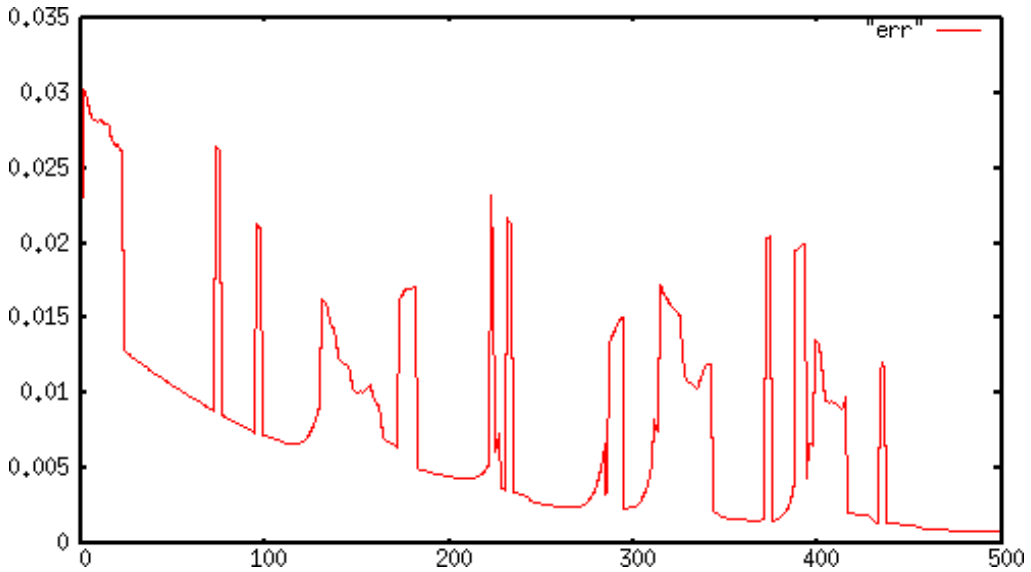
Resultat

- Trainigsmöglichkeiten:
  - Futter vorwärts- oder rückwärts einsammeln
  - Futter umkreisen
  - auf der Stelle drehen
  - nur Rechtskurven benutzen
  - Bewegung in Schleifen
- *ausprobieren!*



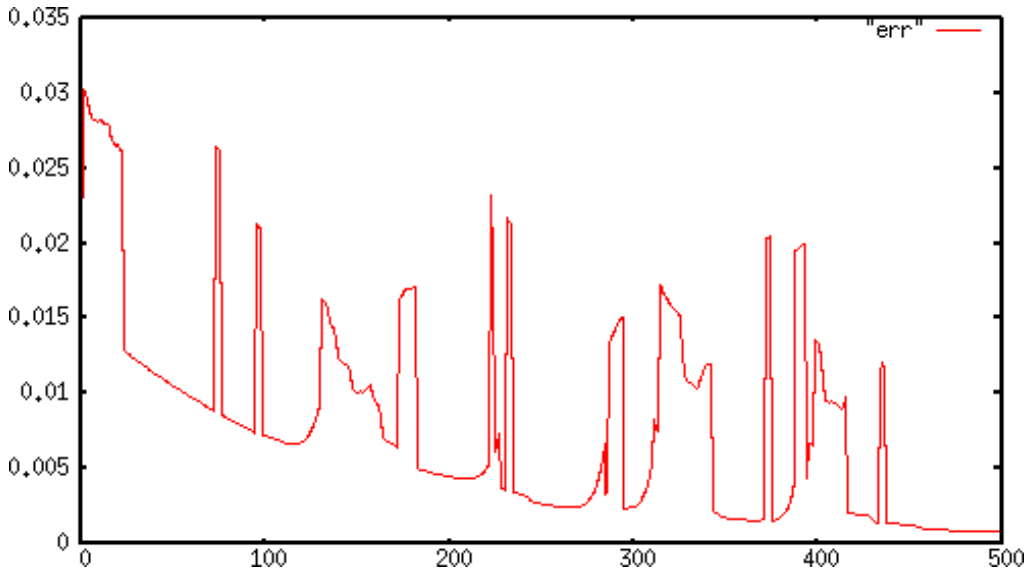
Resultat

- Trainigsmöglichkeiten:
  - Futter vorwärts- oder rückwärts einsammeln
  - Futter umkreisen
  - auf der Stelle drehen
  - nur Rechtskurven benutzen
  - Bewegung in Schleifen
- *ausprobieren!*



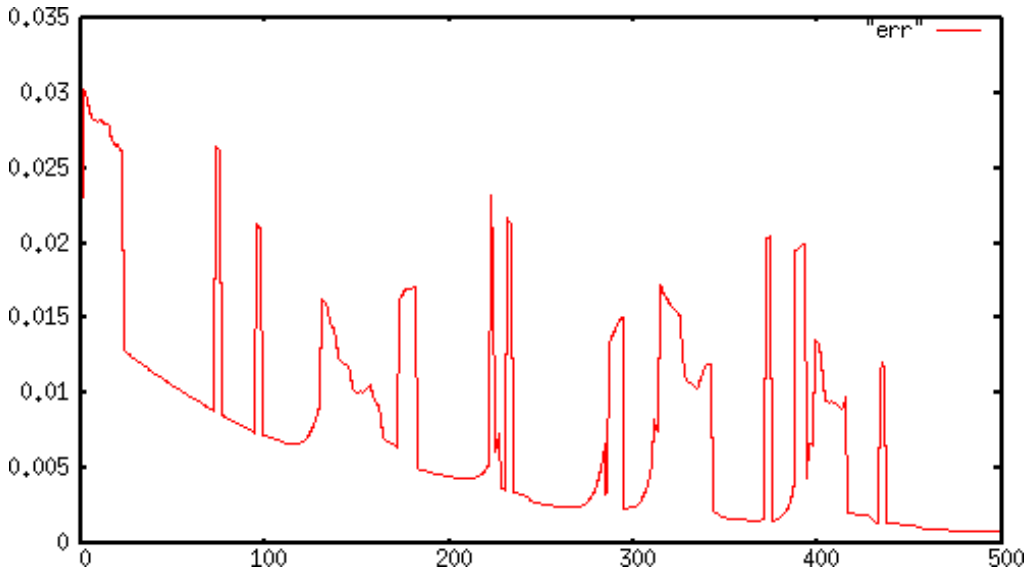
Resultat

- Trainigsmöglichkeiten:
  - Futter vorwärts- oder rückwärts einsammeln
  - Futter umkreisen
  - auf der Stelle drehen
  - nur Rechtskurven benutzen
  - Bewegung in Schleifen
- *ausprobieren!*



Resultat

- Trainigsmöglichkeiten:
  - Futter vorwärts- oder rückwärts einsammeln
  - Futter umkreisen
  - auf der Stelle drehen
  - nur Rechtskurven benutzen
  - Bewegung in Schleifen
- *ausprobieren!*



Fazit

Künstliche neuronale Netze:

- Erzeugung Künstlicher Intelligenz
- flexibel, lernfähig
- zahlreiche Anwendungsgebiete

Ausblick

- Erweiterung des Programms
  - Steuerung von Robotern
  - Ausweichen
- technische Grenzen bald überwindbar?

Fazit

Künstliche neuronale Netze:

- Erzeugung Künstlicher Intelligenz
- flexibel, lernfähig
- zahlreiche Anwendungsgebiete

Ausblick

- Erweiterung des Programms
  - Steuerung von Robotern
  - Ausweichen
- technische Grenzen bald überwindbar?

Fazit

Künstliche neuronale Netze:

- Erzeugung Künstlicher Intelligenz
- flexibel, lernfähig
- zahlreiche Anwendungsgebiete

Ausblick

- Erweiterung des Programms
  - Steuerung von Robotern
  - Ausweichen
- technische Grenzen bald überwindbar?

Fazit

Künstliche neuronale Netze:

- Erzeugung Künstlicher Intelligenz
- flexibel, lernfähig
- zahlreiche Anwendungsgebiete

Ausblick

- Erweiterung des Programms
  - Steuerung von Robotern
  - Ausweichen
- technische Grenzen bald überwindbar?

Fazit

Künstliche neuronale Netze:

- Erzeugung Künstlicher Intelligenz
- flexibel, lernfähig
- zahlreiche Anwendungsgebiete

Ausblick

- Erweiterung des Programms
  - Steuerung von Robotern
  - Ausweichen
- technische Grenzen bald überwindbar?

Fazit

Künstliche neuronale Netze:

- Erzeugung Künstlicher Intelligenz
- flexibel, lernfähig
- zahlreiche Anwendungsgebiete

Ausblick

- Erweiterung des Programms
  - Steuerung von Robotern
  - Ausweichen
- technische Grenzen bald überwindbar?

Fazit

Künstliche neuronale Netze:

- Erzeugung Künstlicher Intelligenz
- flexibel, lernfähig
- zahlreiche Anwendungsgebiete

Ausblick

- Erweiterung des Programms
  - Steuerung von Robotern
  - Ausweichen
- technische Grenzen bald überwindbar?

Diskussion

- Fragt mich aus!

Ende!

- Danke für eure Aufmerksamkeit!
- Diese Präsentation ist frei verfügbar unter der Lizenz **CC-by-nc-sa-2.0**