Seminar - KE und Lernen in Spielen

„Shooting“

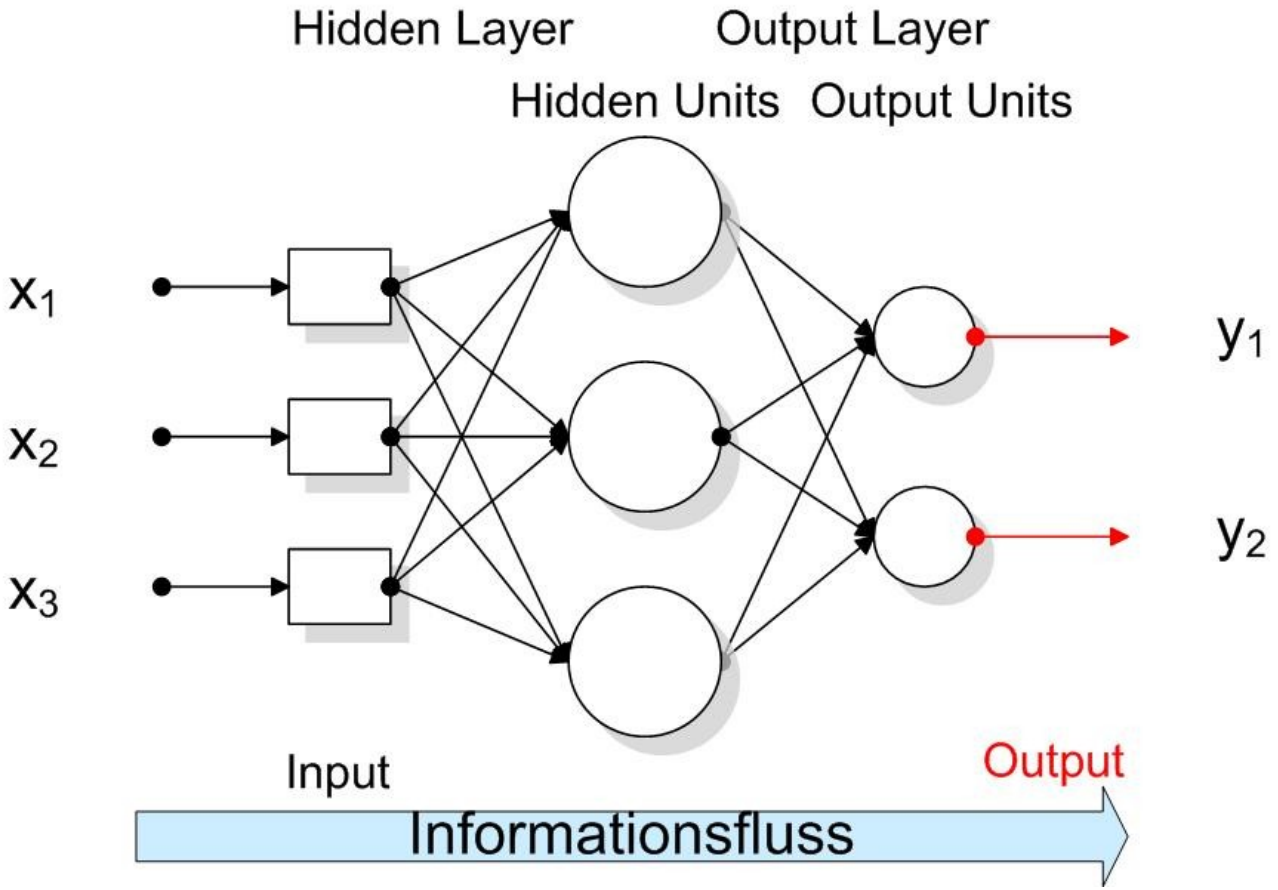
Felix Gliesche
SS 06

Fachgebiet Knowledge Engineering
Prof. Dr. Fürnkranz

Gliederung

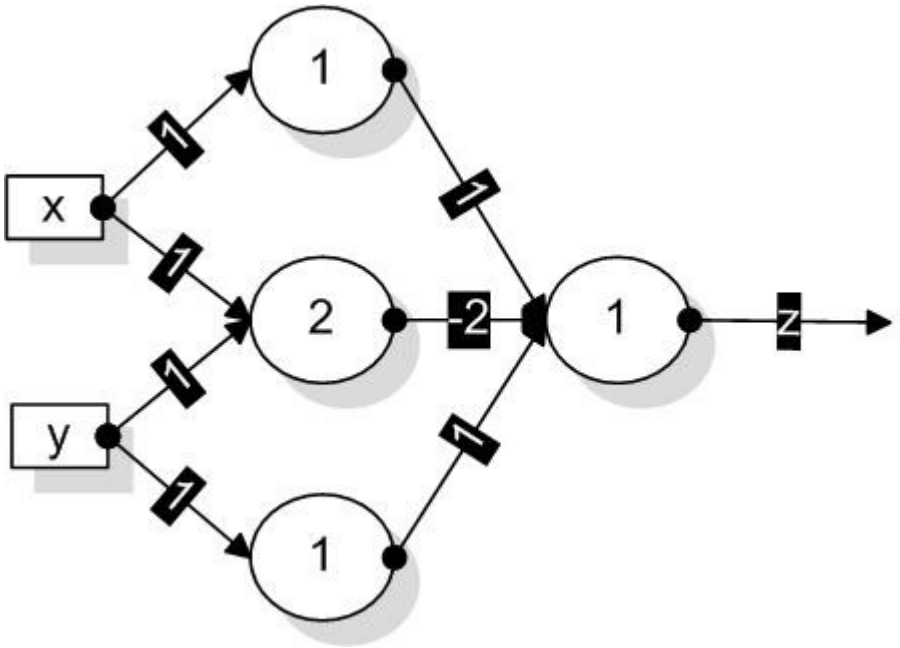
- Multilayer Perceptrons (MLPs)
- Illustration: Shooting realisiert durch MLPs
- Auswertung
- Quellen
- Demo: Quake II

MLPs – Modell


Hidden Layer
Output Layer
Hidden Units
Output Units
x_1
x_2
x_3
y_1
y_2
Input
Output
Informationsfluss

MLPs – Einfaches Beispiel

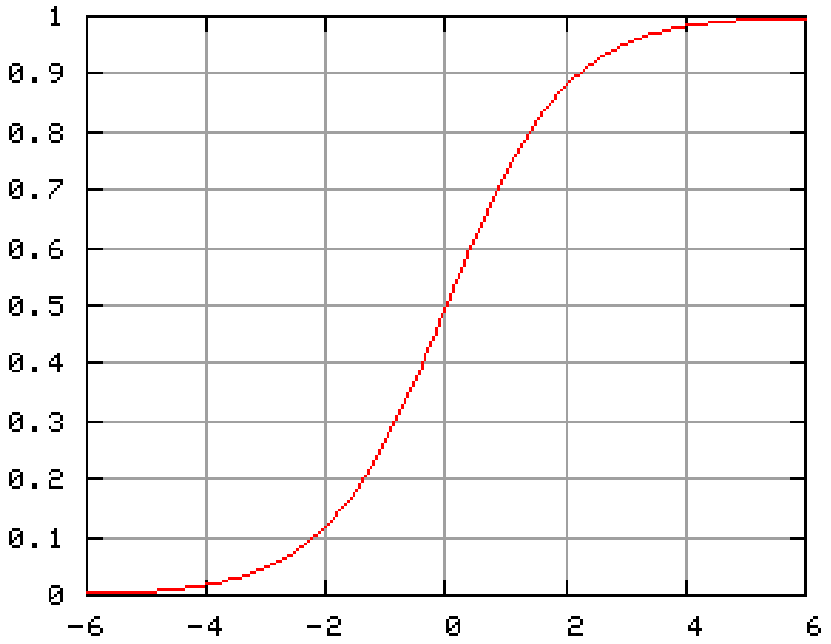
- XOR(x,y) = z



x/y	0	1
0	0	1
1	1	0

MLPs – Unterschiede zu einfachen Perceptrons

- Mehrere Schichten
- Aktivierungsfunktion:



$$sig(x) = \frac{1}{1 + e^{-\beta x}}$$

MLPs - Simulation

```
// zunächst Verarbeitung des Input Arrays
current = input
for layer from first to last
    // output jedes Neurons berechnen
    for each i in [1..neurons] from layer
        // netSum berechnen
        s = netSum(neuron[i].weights, current)
        // Ausgabe speichern
        output[i] = activate(s)
    end for
    // die nächste Schicht bekommt diesen output als input
    current = output
end for
```

MLPs - Training

- Problem: Fehlerberechnung des Outputs der Hidden Units
- Hauptsächlich wird Back Propagation (Backprob) verwendet
- Weitere Algorithmen
  - Quick Propagation
  - Resilient Propagation

MLPs - Training

- Back Propagation Algorithmus (1)
  - Wie beim einfachen Perceptron kann der Fehler für den Output berechnet werden.
  - Fehler wird auf die Vorgänger “propagiert“
  - Fehler jeder Hidden Unit ist die gewichtete Summe der Fehler der Output Units

MLPs - Training

- Back Propagation Algorithmus (2)
  - Berrechnung des Fehlers (delta)

$$\delta_j = \begin{cases} \sigma'(\varsigma_j)(t_j - y_j), \text{ falls j output unit} \\ \sigma'(\varsigma_j)\sum_k \delta_k w_{jk}, \text{ falls j hidden unit} \end{cases}$$

  - Generalized delta rule um die Gewichte anzupassen

```
for each unit j
    for each input i
        // Anpassen der Gewichte
        weight[i][j] += learning_rate * delta[j] * output[i]
    end for
end for
```

Illustration – Target Selection

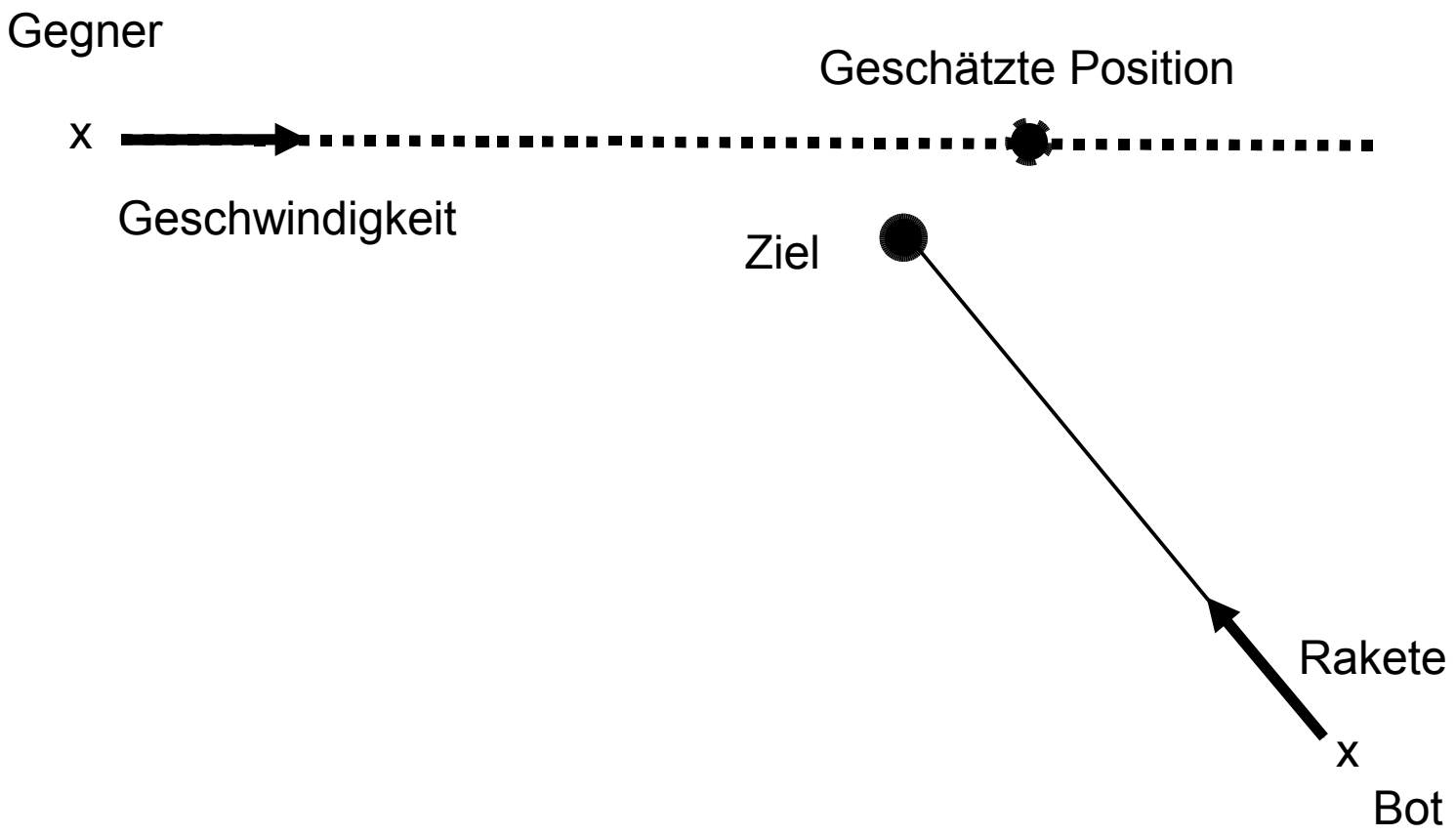


Illustration – Target Selection

- Ziel ist es, die Effektivität eines Schusses vorhersagen zu können
- Dazu muss zunächst ein Ziel ausgewählt werden
- Algorithmus:
  - Generieren von zufälligen Punkten um die geschätzte Positions des Gegners
  - Suchen des ersten Hindernisses auf einer Linie
  - Voraussagen des Schadens
  - Falls Schaden hoch genug → Wähle Ziel

Illustration – Target Selection

- Probleme beim Informationsgewinn des Schusses
  - Hat der Schuss Schaden angerichtet?
  - War der Schuss von mir?
- Störungen in den Daten
  - Gleiche Situationen führen zu unterschiedlichen Ergebnissen
  - Limitierte Datenerfassung

Illustration – Target Selection

- Input
  - Geschätzte Position des Gegners / Ziel
  - Bot / Ziel
  - Gegner / Ziel
- Output
  - Hit
  - Miss

Illustration – Target Selection

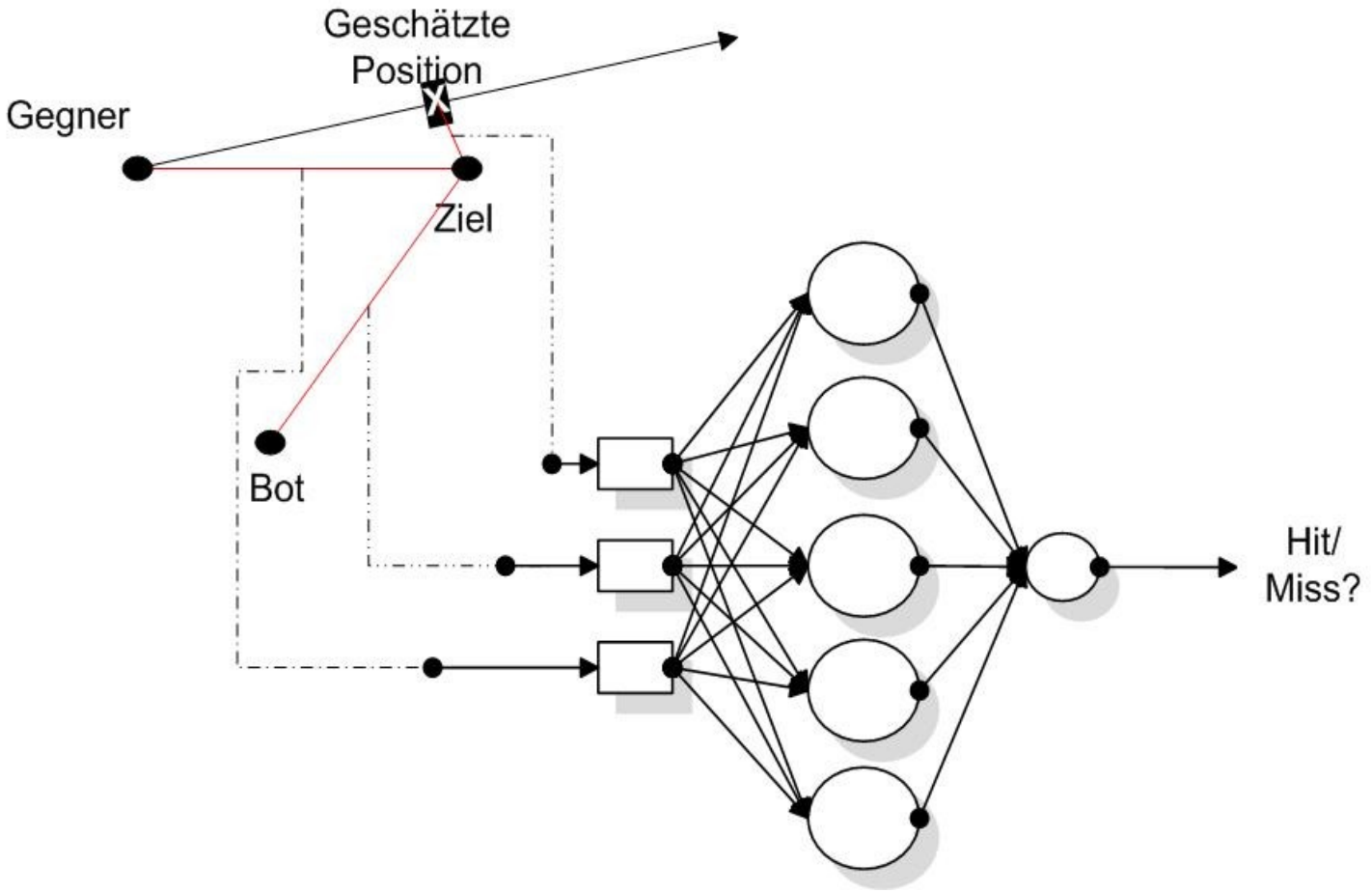
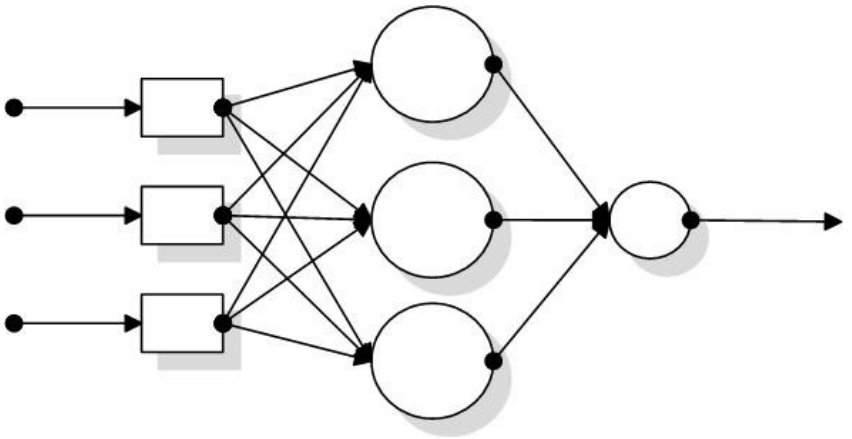


Illustration – Target Selection

- XML

```
<Perceptron>
        <layer inputs="3" units="3" />
        <layer units="1" />
</Perceptron>
```

- Modell



Auswertung

- Durchschnittlicher Fehler beträgt 25%
- Dennoch sichtbare Verbesserung der Zielauswahl
- Trends sind zu erkennen
- MLP erkennt schlechte Vorschläge bei Zielauswahl
- Kann auf andere NPCs einfach übernommen werden

Quellen

- Alex J. Champandard: AI Game Development, New Riders Publishing, 2003.
- http://aigamedev.com/
- http://library.thinkquest.org/18242/perceptron.shtml
- http://www.ai-junkie.com/ann/evolved/nnt1.html
- http://de.wikipedia.org/wiki/Perzeptron

Demo - Quake II