Einführung in Neuronale Netze

Sophia Thamm

Seminar Maschinelles Lernen

Dr. Zoran Nikolić

Ziel

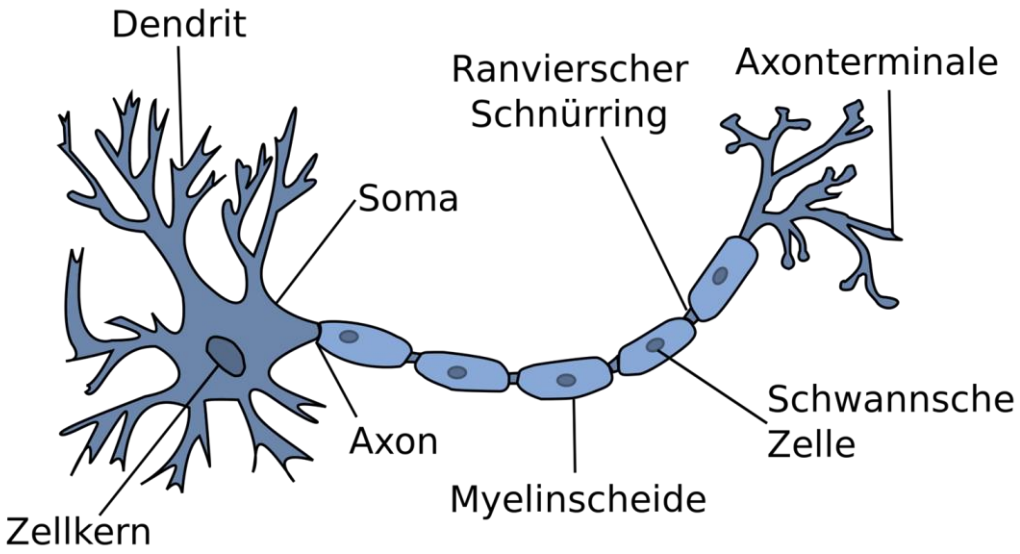
Grundlagen Neuronaler Netze verstehen

- Was sind Neuronale Netze?
- Welche Anwendungen gibt es für Neuronale Netze?
- Wie sind sie aufgebaut?
- Wie lernen Neuronale Netze?
- Welche Schwierigkeiten gibt es und wie löst man diese?
- Wie konfiguriert man ein Neuronales Netz?

Idee der Neuronalen Netze

Modell des menschlichen Nervensystems

Abbildung eines Neurons:



- Dendriten empfangen Signale und leiten elektrisches Signal ins Soma
- Falls Signale insgesamt größer als Schwellenwert, Weitergabe eines Impulses über das Axon (**Alles-oder-Nichts-Prinzip**)
- Impuls wird über Synapse an weiteres Neuron oder an Muskelzelle weitergegeben

Künstliche Neuronale Netze

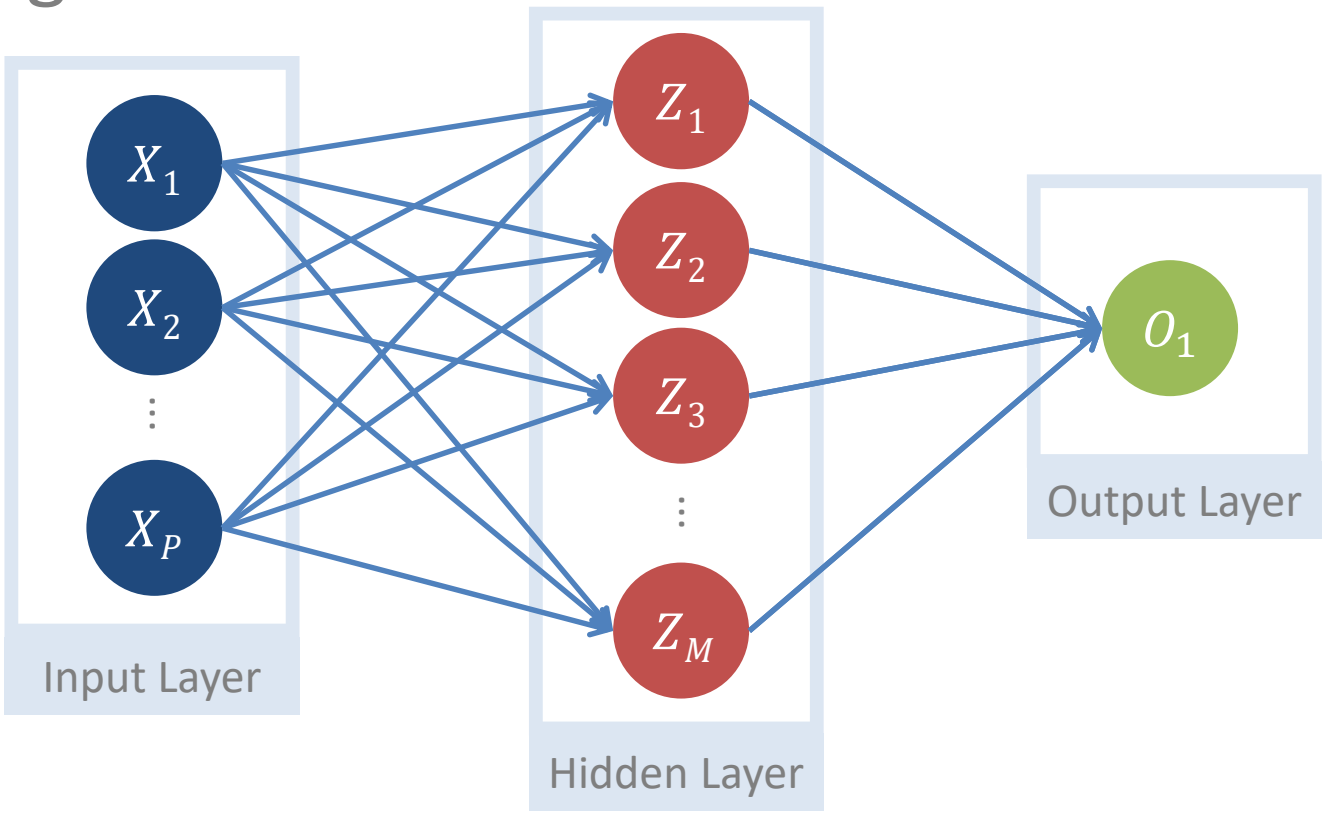
Modelle zur Informationsverarbeitung

- **Nicht-lineare statistische Modelle zur Informationsverarbeitung**
- Informationsverarbeitung umfasst hierbei unter anderem:
  - Datenklassifizierung
  - Prozessimulation
  - Prognosenerstellung

- Units der Neuronalen Netze angelehnt an Neuronen
  - Inputs zusammenfassen
  - Mit Schwellenwert vergleichen bzw. aktivieren
- Verbindungen zwischen Units angelehnt an Synapsen
  - Gewichtung mit verstärkender oder schwächender Wirkung

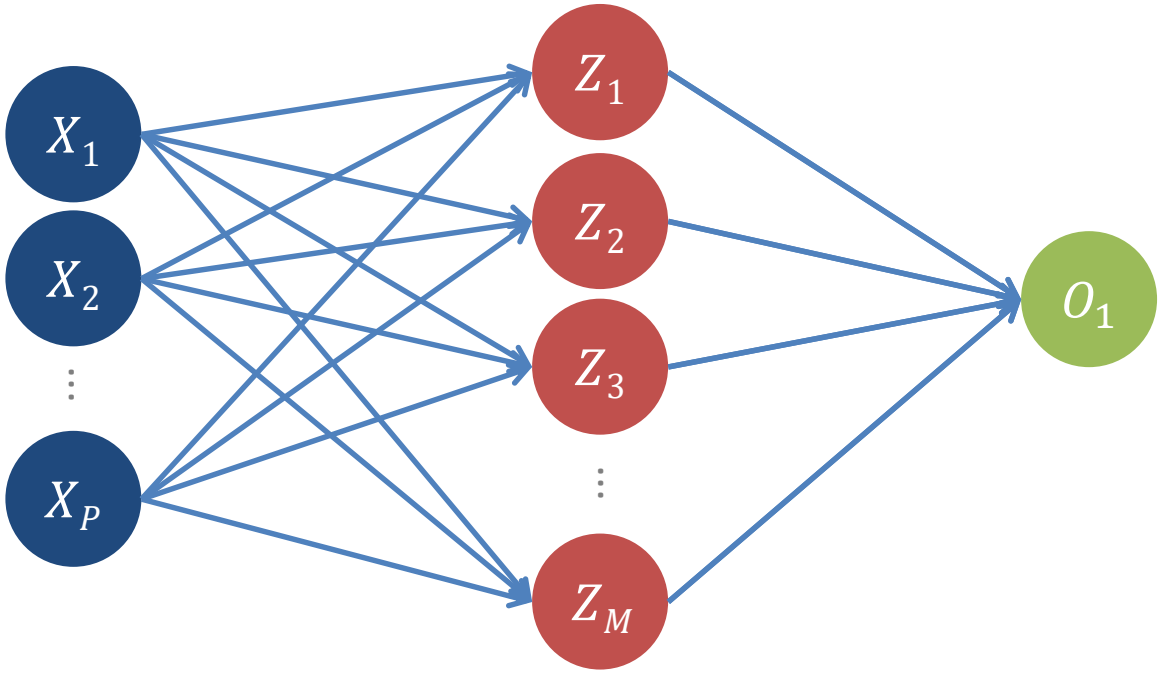
Topologie Neuronales Netz

Regression



Topologie Neuronales Netz

Regression

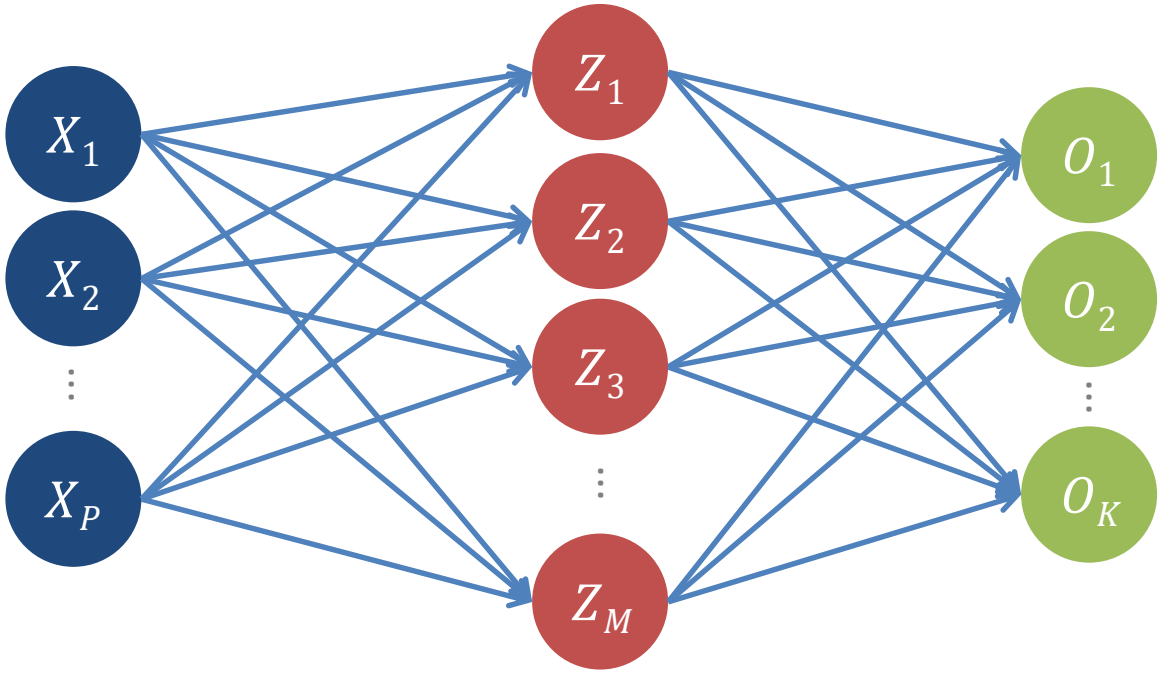


- Einheiten eines Layers: Input, Hidden, bzw. Output Unit
- Mehrere Hidden Layer möglich
- Hier: Single Hidden Layer, Feed Forward Netzwerk

Beispiel Regression: Prognose eines Aktienkurses

Topologie Neuronales Netz

Klassifizierung

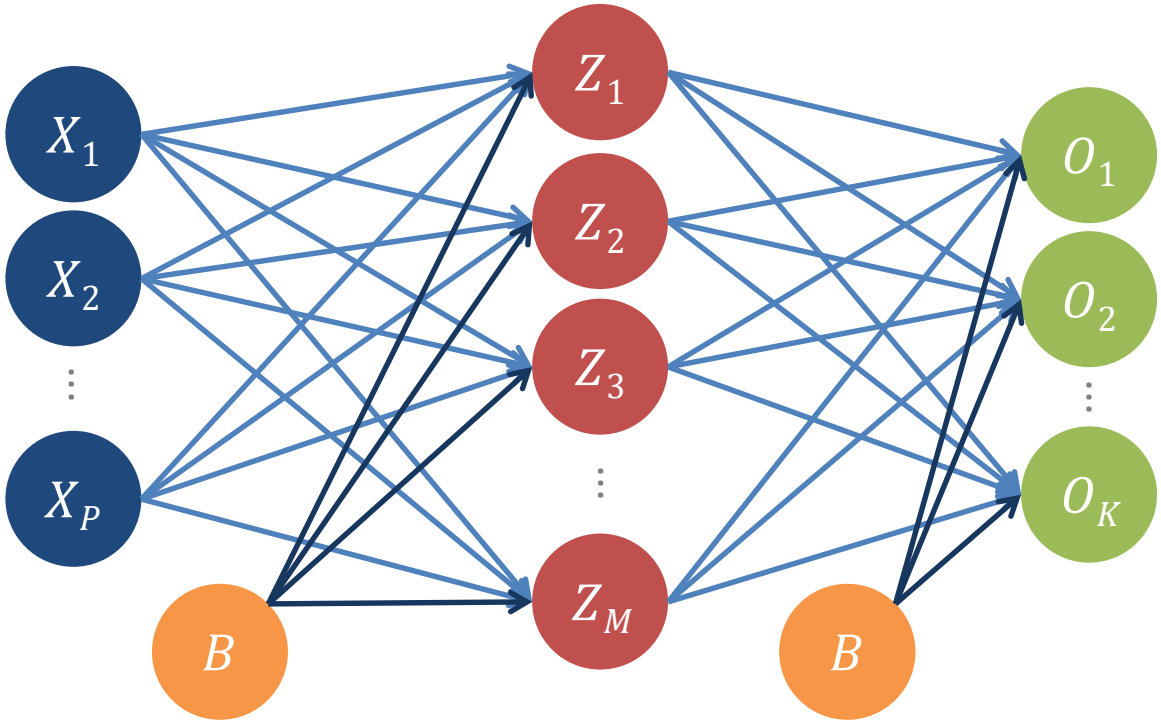


- Mehrdimensionales Output Layer: Klassifizierung
- k-te Unit modelliert Wahrscheinlichkeit, dass Input in Klasse k liegt

Beispiel Klassifizierung: Handschrifterkennung Ziffern

Zusatz

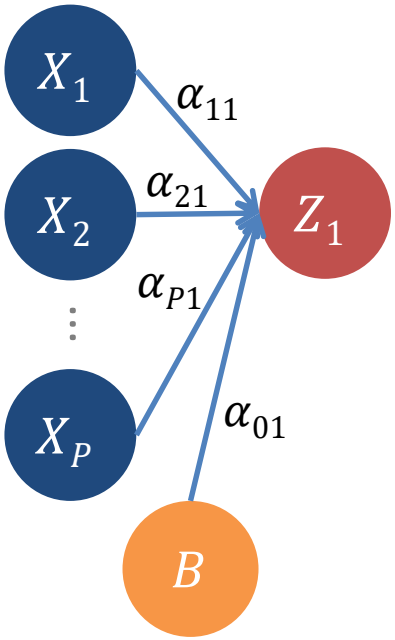
Bias Unit



- Bias Units zur Modellierung der Konstanten bzw. Schwellenwerte des linearen Modells
- Input der Bias Units ist $+1$

Mathematisches Modell

Unit Betrachtung

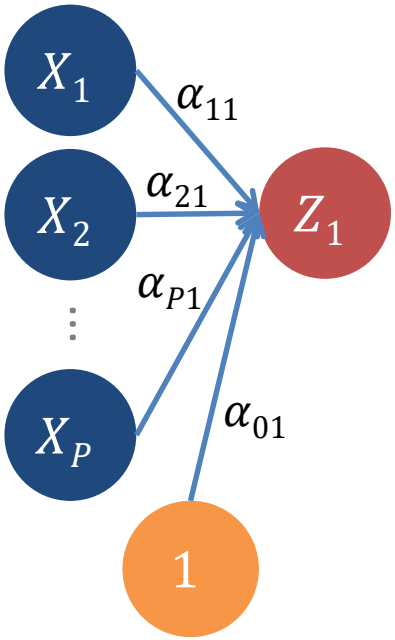


$$\alpha_1^* \left\{ \begin{bmatrix} \alpha_{01} \\ \alpha_{11} \\ \vdots \\ \\ \alpha_{P1} \end{bmatrix} \right. \quad \alpha_1 \quad {}^T \begin{bmatrix} B \\ X_1 \\ X_2 \\ \vdots \\ X_P \end{bmatrix} = \hat{Z}_1$$

- Anwendung der **gewichteten Summe** auf Ausgabe der Input Units zur Ermittlung der **Netzeingabe** $\hat{Z}_1$: $\hat{Z}_1 = {\alpha_1}^{*T} X^*$

Mathematisches Modell

Unit Betrachtung



$$\alpha_1^* \left\{ \begin{bmatrix} \alpha_{01} \\ \alpha_{11} \\ \vdots \\ \\ \alpha_{P1} \end{bmatrix} \right. \quad \alpha_1 \quad {}^T \begin{bmatrix} 1 \\ X_1 \\ X_2 \\ \vdots \\ X_P \end{bmatrix} = \hat{Z}_1$$

- Anwendung der **gewichteten Summe** auf Ausgabe der Input Units zur Ermittlung der **Netzeingabe** $\hat{Z}_1$: $\hat{Z}_1 = {\alpha_1}^{*T} X^* = \alpha_{01} + {\alpha_1}^T X$
- Anwendung einer nicht-linearen **Aktivierungsfunktion** auf $\hat{Z}_1$ zur Ermittlung der Ausgabe Z_1: $Z_1 = \sigma(\hat{Z}_1)$

Mathematisches Modell

Matrixschreibweise mit Bias Unit

$$\begin{pmatrix} \alpha_{01} & \alpha_{02} & \cdots & \alpha_{0M} \\ \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1M} \\ \vdots & \ddots & & \\ & & & \\ \alpha_{P1} & \cdots & & \alpha_{PM} \end{pmatrix}^T \begin{pmatrix} B_1 \\ X_1 \\ X_2 \\ \vdots \\ X_P \end{pmatrix} = \begin{pmatrix} \hat{Z}_1 \\ \hat{Z}_2 \\ \vdots \\ \hat{Z}_M \end{pmatrix}$$

$$Z = \sigma(\alpha^{*T} X^*)$$

Mathematisches Modell

Matrixschreibweise mit Bias Unit

$$\begin{bmatrix} \alpha_{01} & \alpha_{02} & \cdots & \alpha_{0M} \\ \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1M} \\ \vdots & \ddots & & \\ & & & \\ \alpha_{P1} & \cdots & & \alpha_{PM} \end{bmatrix}^T \begin{bmatrix} 1 \\ X_1 \\ X_2 \\ \vdots \\ X_P \end{bmatrix} = \begin{bmatrix} \hat{Z}_1 \\ \hat{Z}_2 \\ \vdots \\ \hat{Z}_M \end{bmatrix}$$

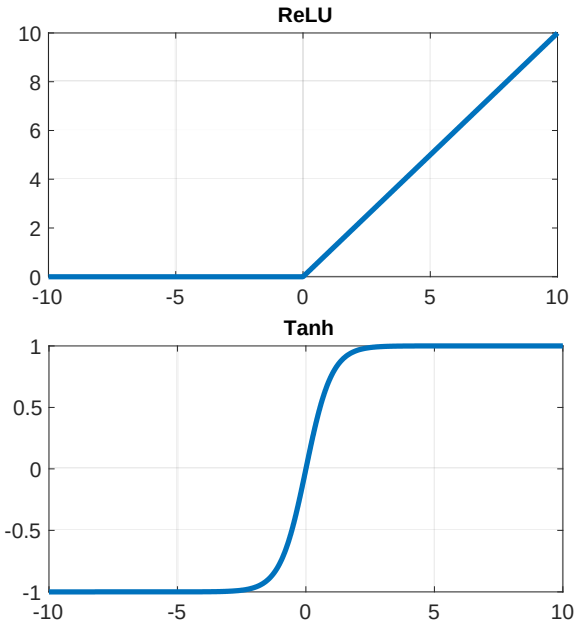
$$Z = \sigma(\alpha_0 + \alpha^T X)$$

Aktivierungsfunktion

Essenzielle Arbeit einer Unit

- Aktivierungsfunktion verarbeitet die Netzeingabe zu einer Ausgabe
- Aktivierungsfunktion ist meist global für alle Neuronen definiert

- Verschiedene geeignete Funktionstypen
  - Binäre Schwellenwertfunktion
  - Stückweise lineare Funktion
  - ReLU Funktion
  - Sigmoid Funktion
  - Tangens Hyperbolicus

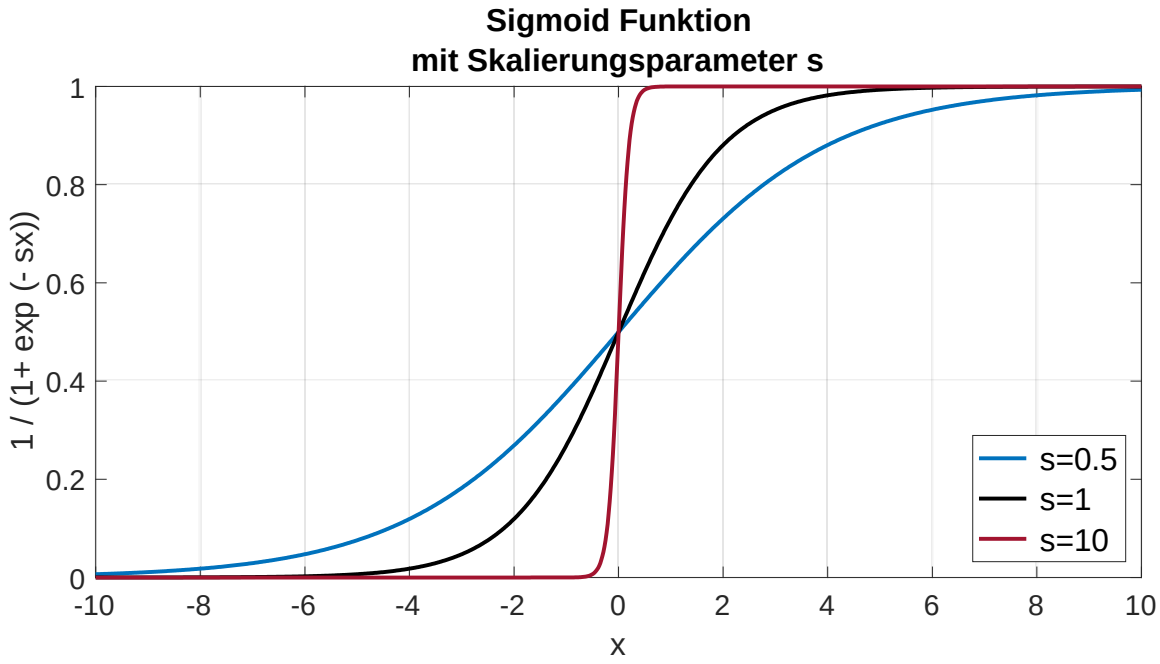


- Lineare Funktionen sind nicht als Aktivierungsfunktion geeignet

Aktivierungsfunktion

Hidden Layer

- **Sigmoid Funktion** $\sigma(x) = \frac{1}{(1+ e^{-x})}$ als praktisch häufig angewandte Aktivierungsfunktion
- $Z_1 = \sigma({\alpha_1}^{*T} X^*)$ Gewichte bestimmen die „Linearität“ des Sigmoids



Aktivierungsfunktion

Output Layer

Regression

- **Identität** als Aktivierungsfunktion im Output Layer

Klassifikation

- **Softmax-Funktion** transformiert finale Werte in eine kategoriale Wahrscheinlichkeitsverteilung
  - Summe der Outputs ist 1
  - O_k ist Wahrscheinlichkeit, dass Input X der Kategorie k entspricht

- Finale Transformation der Netzeingabe $\hat{O} := \beta_0 + \beta^T Z$ mittels Softmax:

$$O = g(\hat{O}) = \frac{e^{\hat{O}}}{\sum_{k=1}^{K} e^{\hat{O}_k}}$$

Lernprozess

Definitionen und Methoden

Überwachtes Lernen: Training mit Input-Output-Paaren

- Netz wird nach Ausgabe ein genauer Fehlervektor zurückgegeben
- Netzgewichte werden mittels Lernregel angepasst
- **Lernregel:** Algorithmus, der festlegt, welche Gewichte wie stark verändert werden und dem Netz beibringt, für eine vorgegebene Eingabe eine gewünschte Ausgabe zu produzieren

Wie kann ein Neuronales Netz lernen?

- Entwicklung/Löschung von Verbindungen
- ➔ Änderung der Gewichte
- Änderung der Schwellenwerte von Neuronen
- Entwicklung/Löschung von Units

Online oder Offline?

- Offline: mehrere Trainingsbeispiele gleichzeitig, Betrachtung des kumulierten Fehlers
- ➔ Online: jedes Trainingsbeispiel einzeln, Betrachtung des Fehlers nach jedem Beispiel

Delta-Lernregel

Gewichtsanpassung im Neuronalen Netz

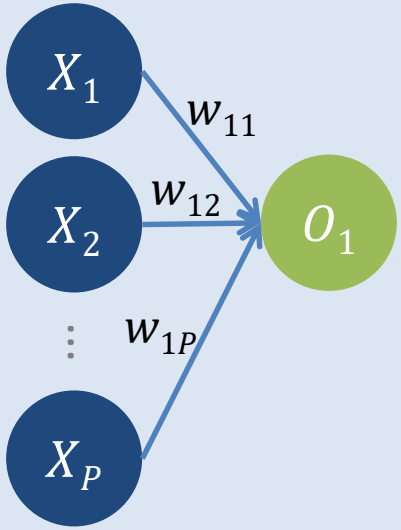
Delta-Regel

Vergleicht gewünschten und beobachteten Output:

$$\delta_j = Y_j - O_j$$

Anpassung der Gewichte mittels:

$$\Delta w_{ij} = \eta \, \delta_j X_i, \text{ wobei } \eta \text{ Lernrate}$$



Vorteil

- Gewichtsveränderung proportional zur Größe des Fehlers
- Einflussreiche Inputs werden stärker gewichtet

Nachteil

- Nur auf Neuronale Netze ohne Hidden Units anwendbar

Lernrate

Erfolgsentscheidender Parameter

- **Lernrate** η bestimmt Geschwindigkeit und Genauigkeit des Lernverfahrens
- η groß
  - Sprünge auf der Fehlerfläche zu groß, Überspringen gute Werte
  - Unkontrollierte Bewegung über die Fehlerfläche
- η klein
  - Hoher Zeit-/Rechenaufwand
- Gute Werte für η liegen erfahrungsgemäß zwischen 0,01 und 0,9
  - Hängt von Problem, Netz und Trainingsdaten ab
  - Trial-and-Error
- **Strategie**: Training mit relativ großem η starten und stufenweise verringern

Backpropagation

Algorithmus zur Gewichtsmodifikation

- **Backpropagation**: Gewichtsmodifikation durch Rückwärtsausbreitung der Fehlerterme

Backpropagation Algorithmus

Gegeben seien N Trainingsdatenpaare $\left(X^{(i)}, Y^{(i)}\right)$ mit $i = 1, \ldots, N$

1. **Forward-Pass**
   Output O des Neuronalen Netzes mit initialen/neuen Gewichten berechnen
2. **Fehlerbestimmung**
   Delta δ_j bestimmen und mit gewählter Fehlertoleranz vergleichen
3. **Backward-Pass**
   Fehlerterme werden mittels **Gradientenabstiegsverfahren** rückwärts-gerichtet an die Gewichte weitergegeben

Backpropagation

Herleitung Gradientenabstiegsverfahren

- Sei $\theta = \{\alpha_{0m}, \alpha_m ; m = 1, \dots, M\} \cup \{\beta_{0k}, \beta_k ; k = 1, \dots, K\}$ die Menge aller Gewichte im Netz
- Wollen die Fehlerfunktion $R(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{K} ({Y_k}^{(i)} - {O_k}^{(i)})^2$ minimieren

Betrachten $R_i(\theta) = \frac{1}{2} \sum_{k=1}^{K} ({Y_k}^{(i)} - {O_k}^{(i)})^2$ und leiten nach α_{pm} bzw. β_{mk} ab:

$$\frac{\partial R_i(\theta)}{\partial \beta_{mk}} = \underbrace{-\left({Y_k}^{(i)} - {O_k}^{(i)}\right) g'\left(\beta_{0k} + \beta_k^T Z^{(i)}\right)}_{\delta_k^{(i)}} Z_m^{(i)} = \delta_k^{(i)} Z_m^{(i)}$$

Analog:

$$\frac{\partial R_i(\theta)}{\partial \alpha_{pm}} = \underbrace{-\left({Y_k}^{(i)} - {O_k}^{(i)}\right) g'\left(\beta_{0k} + \beta_k^T Z^{(i)}\right) \beta_{mk}\, \sigma'\left(\alpha_{0m} + \alpha_m^T X^{(i)}\right)}_{\varphi_m^{(i)}} X_p^{(i)} = \varphi_m^{(i)} X_p^{(i)}$$

Backpropagation

Herleitung Gradientenabstiegsverfahren

- Sei $\theta = \{\alpha_{0m}, \alpha_m ; m = 1, \dots, M\} \cup \{\beta_{0k}, \beta_k ; k = 1, \dots, K\}$ die Menge aller Gewichte im Netz
- Wollen die Fehlerfunktion $R(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{K} ({Y_k}^{(i)} - {O_k}^{(i)})^2$ minimieren

Betrachten $R_i(\theta) = \frac{1}{2} \sum_{k=1}^{K} ({Y_k}^{(i)} - {O_k}^{(i)})^2$ und leiten nach α_{pm} bzw. β_{mk} ab:

$$\frac{\partial R_i(\theta)}{\partial \beta_{mk}} = \delta_k^{(i)} \, Z_m^{(i)}$$

$$\beta_{mk}^{(i+1)} = \beta_{mk}^{(i)} - \eta \, \delta_k^{(i)} \, Z_m^{(i)}$$

Analog:

$$\frac{\partial R_i(\theta)}{\partial \alpha_{pm}} = \varphi_m^{(i)} \, X_p^{(i)}$$

$$\alpha_{pm}^{(i+1)} = \alpha_{pm}^{(i)} - \eta \, \varphi_m^{(i)} \, X_p^{(i)}$$

Backpropagation

Probleme des Gradientenabstiegs

- Konvergenz gegen **suboptimale Minima**
  - Verfahren bleibt in einem lokalen Minimum „hängen“
- **Flache Plateaus** verlangsamen Training
  - Viele Schritte benötigt bei sehr kleinem Gradienten
- **Verlassen guter Minima**
  - Großer Gradient bedingt große Schritte, sodass gute Minima übersprungen werden können
- **Verschwindende Gradienten**
  - Bei multiplen Layers geringer Lerneffekt der Gewichte der vorderen Schichten

Backpropagation

Modifikationen

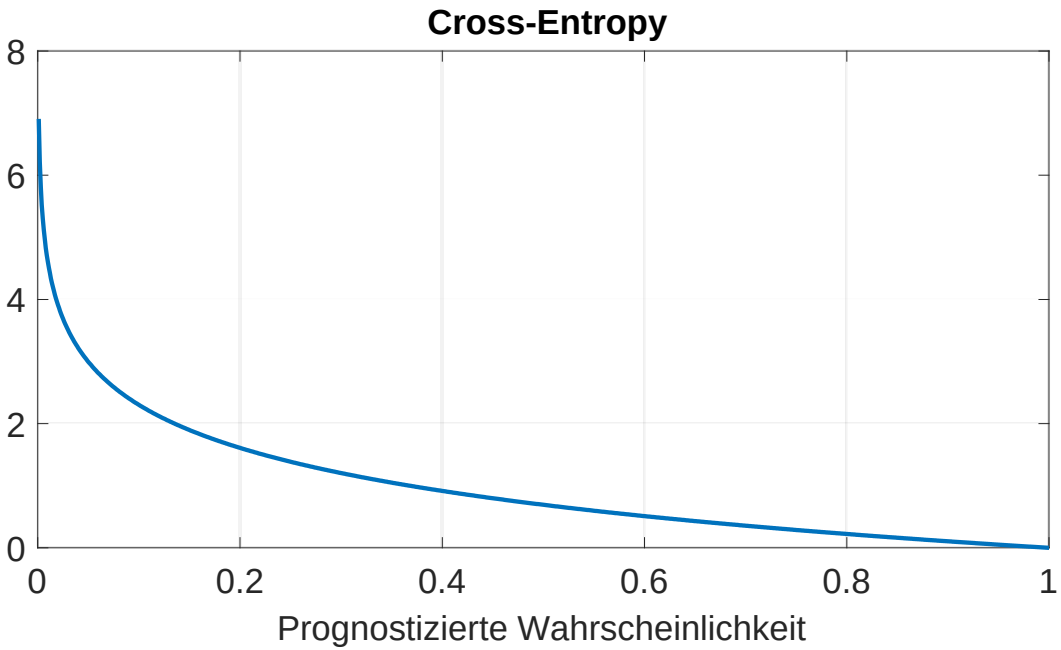
- Addition eines **Momentum-Terms**, der vorherige Gewichtsmodifikation miteinbezieht: $\Delta w_{mk}^{(i)} = (1-\varepsilon)\,\eta\,\delta_k^{(i)}\,Z_m^{(i)} - \varepsilon\,\Delta w_{mk}^{(i-1)}$
  - Beschleunigung auf flachen Plateaus, Verhinderung von Oszillation
  - Trägheit durch Vorfaktor ε
  - Nachteil: schnelleres Verlassen guter Minima
- Einbeziehung der **zweiten Ableitung**
  - Genauere Schätzungen der Gewichtskorrekturen
  - Weniger Trainingszyklen, aber höherer Rechenaufwand
- **Pruning** („Stutzen“)
  - Entfernung von Neuronen mit geringem Einfluss verhindert Auswendiglernen

Cross-Entropy

Fehlerfunktion bei Klassifikationsmodellen

- Cross-Entropy als **alternative Fehlerfunktion** zur Summe der quadratischen Abweichungen in Klassifikationsmodellen

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} Y_k^{(i)} \log(O_k^{(i)})$$



Cross-Entropy

Fehlerfunktion bei Klassifikationsmodellen

- Cross-Entropy als **alternative Fehlerfunktion** zur Summe der quadratischen Abweichungen in Klassifikationsmodellen

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} Y_k^{(i)} \log(O_k^{(i)})$$

- Mit Softmax Aktivierungsfunktion und Cross-Entropy Fehlermessung
  - **lineares logistisches Regressionsmodell** in den Hidden Units
  - Gewichte werden durch **Maximum-Likelihood** Schätzung bestimmt

Konfiguration eines Neuronalen Netzes

Topologie

- **Keine Patentformel** für die Initialkonfiguration eines Neuronalen Netzes
- Bestes Netz für jedes Problem unterschiedlich
- **Trial-and-Error** Methode

- Anzahl Hidden Units?
  - Mit relativ hoher Anzahl Units starten, Units mit geringem Einfluss ausschließen (Pruning)
  - Bei zu wenigen Units: fehlende Flexibilität, um Nichtlinearitäten abzubilden
- Anzahl Hidden Layer?
  - Jedes Layer extrahiert Eigenschaften aus dem Input, die der Klassifikation/Regression dienen
  - Multiple Layer erlauben Konstruktion von hierarchischen Eigenschaften mit unterschiedlicher Granularität
  - Initialisierung mittels Hintergrundwissen über das Problem und Trial-and-Error

Konfiguration eines Neuronalen Netzes

Initiale Gewichte

- Initiale Gewichte gleich Null?
  - Ableitungen gleich Null → kein Lerneffekt durch Backpropagation

- Initiale Gewichte **zufällige Werte nahe bei Null**
  - Sigmoid nahezu linear, daher auch Modell nahezu linear
  - Mit absolut größeren Gewichten wird das Modell zunehmend nicht-linear

- Mehrfache Wiederholung des Trainings mit verschiedenen Startwerten

Neuronale Netze

Vor- & Nachteile

Vorteile

- Kein konstruiertes Modell (Formalisierung) notwendig
- Großer Anwendungsbereich
- Widerstandsfähigkeit (Rauschen, fehlender Input)
- Generalisierungsfähigkeit
- Lösungen auch für stark nicht-lineare Probleme

Nachteile

- „Black Box“ Hidden Units
- Unsicherheit, ob systematischer Fehler (Bias) im Netz
- Hoher Trainingsdatenbedarf
- Hoher Rechenaufwand
- Trial-and-Error zur Optimierung

Literatur

[1] T. Hastie, R. Tibshirani, J.H. Friedman: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics, 2. Aufl.: Springer 2009.

[2] D. Kriesel: Ein kleiner Überblick über Neuronale Netze. URL: dkriesel.com [25.05.2019].