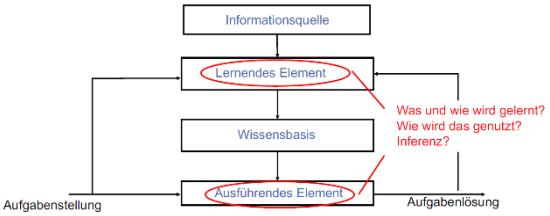
Maschinelles Lernen I

Lernzusammenfassung zur Vorlesung am KIT
Lukas Bach - lbach@outlook.de - lukasbach.com

1 Einführung

- Was ist Lernen?
  - Lernen von Entscheidungen, Aktionsfolgen, Beschreibungen, Modellen…
  - Lernen motorischer/kognitiver Fähigkeiten durch Anweisungen/Training
  - Neuorganisation/Transformation von Wissen
- Maschinelles Lernen
  - Ein System lernt aus Erfahrung E in Hinblick auf eine Klasse von Aufgaben T und einem Performancemaß P, wenn seine Leistungen bei Aufgaben aus T gemessen mit P durch Erfahrung aus E steigt.
    Ein lernendes System generiert eine oder mehrere Lösungshypothese(n) H um Aufgaben T zu lösen.
  - Beispiel: Schach spielen lernen.
    - T = Schach spielen
    - P = % der gewonnenen Spiele
    - E = Spiele gegen sich selbst
    - H = Modell um Spielzüge anhand der aktuellen Situation zu erzeugen
- Komponenten eines lernenden Systems



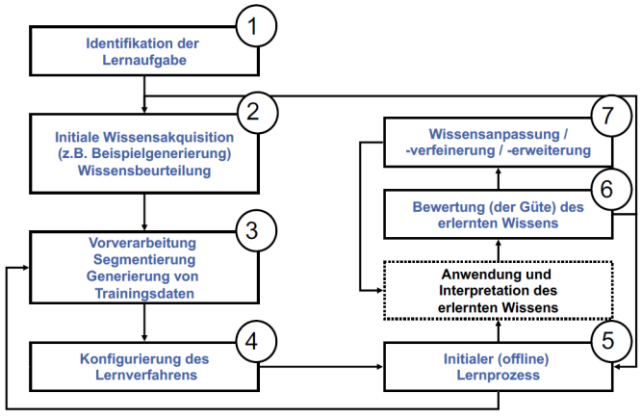
  -
- Deduktiver Schluss
  - Aus einer Menge von Formeln A folgt B
    ⇔ Es gibt eine Folge von Regeln, um B herzuleiten.
  - Beispiele: Modus Ponens, Instantiierung
- Abduktion
  - H folgt aus Hintergrundwissen B und Beobachtungen D abduktiv

$$\Leftrightarrow B \cup H \mapsto D$$

- Induktiver Schluss
  - Geg. Menge D von Grundbeispielen. Hypothese H folgt induktiv aus D und Grundwissen B

$$\Leftrightarrow B \cup H \mapsto D, B \not\mapsto D, B \cup D \not\mapsto \neg H$$

- Wissensrepräsentation
  - Assoziierte Paare (In/Output)
  - Parameter in algebraischen Ausdrücken (Funktionale Form von numerischen Parametern, zB Gewichtsmatrix)
  - Entscheidungsbäume (Klassendiskriminierung)
  - Formale Grammatiken
  - Produktionsregeln
  - Formale logikbasierte Ausdrücke
  - Graphen/Netzwerke
  - Probabilistische graphische Modelle (Verbundwahrsch. von Zufallsvariablen)
  - Frames, Schemata, Schemantische Netze
  - Prozedurale Kodierung
  - Taxonomien (Hierarchische Klassifikation, Lernen durch Beobachtung)
  - Markov-Ketten (zustandsbasiertes Hintergrundwissen)
  - Kombination von Repräsentationsarten
- Lernen als Prozess



(Siehe Folien F01.57)

- Einordunungskriterien von Lernverfahren

Typ der Inferenz	induktiv vs deduktiv
Ebenen des Lernens	symbolisch vs subsymbolisch
Lernvorgang	überwacht vs unüberwacht
Beispielgebung	inkrementell vs nicht inkr.
Umfang der Bsps	umfachreich vs gering
Hintergrundwissen	empirisch vs axiomatisch

1.1 Vorlesung Übersicht

- Einführung
- Induktives Lernen
- Lerntheorie
- Support Vektor Machine
- Neuronale Netze
- Convolutional Neural Networks (CNN)
- Entscheidungsbäume
- Reinforcement Learning
- Hidden Markov Modelle
- Lernen nach Bayes
- Instanzbasiertes Lernen
- Unüberwachtes Lernen
- Deduktives Lernen
- Evolutionäre Algorithen

2 Induktives Lernen

2.1 Induktion und Deduktion

- *Induktion*: Prozess des plausiblen Schließens vom Speziellen zum Allgemeinen. Basis: Viele zutreffende Fälle.
- *Deduktion*: Prozess des korrekten Schließens vom Allgemeinen zum Speziellen. Basis: Hintergrundwissen (Regeln). Bsp: Modus Ponens.
- Induktion vs Deduktion

Induktion	Deduktion
Wahrheitserweiternd	Wahrheiserhaltend
Grundsatz: induktive Lernhyptothese	Logischer Schluss
Plausibilität	Korrektheit

- Induktive Lernverfahren: Konzeptlernen, Finde *Hypothesen* $h \in H$ aus *Hypothesenraum* H die ein *Zielkonzept c(..)* mit Instanzen des *Instanzraums X* erfüllen anhand *Trainingsmenge* $D \subset X$ als Untermenge des Instanzraums.
- Allgemeine Bezeichnungen: Instanzraum = Merkmalsraum, Zielkonzept = Sollausgabe, Erfüllen des Zielkonzepts = Definition eines minimierten Fehlermaß
- *Induktive Lernhypothese*
  - Jede Hypothese, die die Zielfunktion über einer ausreichend großen Menge von Trainingsbeispielen gut genug approximiert, wird die Zielfunktion auch über unbekannten Beispielen gut approximieren.

2.2 Konzeptlernen als Suche im Hyptothesenraum

- *Konzept*: Beschreibt Untermenge von Objekten aus größerer Menge mittels boolscher Funktion.
  - Bsp: vogel: Tier -> {w,f}. vogel(Storch)=w.
- Konzeptlernen: Schließen auf boolsche Funktion aus Trainingsbeispielen von Input/Output. Instanzen werden meist durch Attribute beschrieben.
- *Konsistenz*: Keine False Positives.
- *Vollständig*: Positive Beispiele werden alle positiv klassifiziert.
- Lerne als Suche im Hypothesenraum.
  - Suche vom Allgemeinen zum Speziellen
    - Gehe von allgemeinster Hypothese aus. Spezialisiere auf negativen Beispielen und ignoriere positive Beispiele.
  - Suche vom Speziellen zum Allgemeinen
    - Gehe von speziellster Hypothese aus. Verallgemeinere auf positiven Beispielen und ignoriere negative Beispiele.
  - Paralleles Anwenden beider Methoden: Version Space

2.3 Specific-to-general Suche

2.3.1 Algorithmus

```
Initialisiere h mit spezifischsten Hypothese in H
Für jedes positive Trainingsbeispiel x
  Für jede Attributseinschränkung ai in h
    Wenn ai von x erfüllt wird:
      Tue nichts.
    Sonst:
      Ersetze ai durch die nächstallgemeinere
      Einschränkung, die durch x erfüllt wird.
Gib die Hypothese aus.
```

2.3.2 Beispiel

Siehe Folien F02.20

2.3.3 Beurteilung

- Wichtiges Prinzip im Konzeptlernen.
- Endhypothese ist auch mit negativen Trainingsbeispielen konsistent, solange Trainingsbeispiele konsistent sind und Zielhypothese in H ist.

2.4 Versionsraum/Candidate-Elimination-Algorithmus

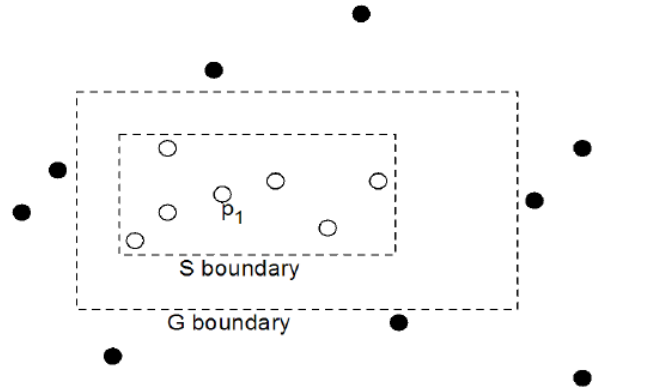
Versionsraum: Der *Versionsraum* $VS_{H,D}$ bzgl. H, D ist die Untermenge der Hypothesen von H, die mit Trainingsbeispielen konsistent und vollständig sind.

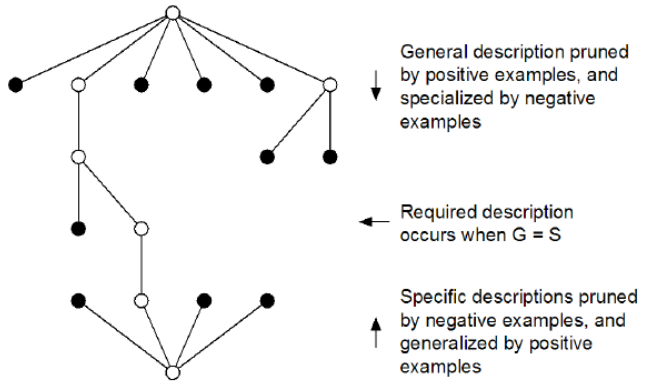
- $VS_{H,D} = \{h \in H : consistent(h, D), compl(h, D)\}$
- Hypothesenraum H, Trainingsmenge D

2.4.1 Algorithmus

- Halte Menge von spezifischsten und Menge von allgemeinsten Hypothesen.
- Ist n ein negatives Beispiel:
  - Lösche aus spezifischsten Hypothesen die Hypothesen, die n abdecken.
  - Spezialisiere die allgemeinsten Hypothesen soweit, dass sie n nicht abdecken und allgemeiner als eine spezielle Hypothese bleiben.
  - Lösche alle allgemeinsten Hypothesen, die spezifischer als eine andere allgemeine Hypothese sind.
- Ist p ein positives Beispiel:
  - Lösche alle allgemeinsten mit p inkonsistenten Hypothesen.
  - Verallgemeinere alle spezifischen Hypothesen soweit, dass sie p abdecken und spezifischer als eine allgemeine Hypothese bleiben.
  - Lösche alle spezifischen Hypothesen, die allgemeiner als eine andere spezifische Hypothese sind.

2.4.2 Veranschaulichung





(●: pruned descriptions)

Beispiel siehe Folien F02.35

2.4.3 Beurteilung

VS konvergiert zur korrekten Hypothese, wenn Beispiele konsistent sind und korrekte Hypothese in H enthalten ist.

Probleme: Rauschen in D, Zielhypothese nicht von Hypothesenrepräsentation abgedeckt.

Wenn mehrere Hypothesen unterschiedlich klassifizieren, zB. probabilistische Ausgabe.

2.5 Notwendigkeit von Vorzugskriterien (Bias)

- Bisher: Wie sehr kann man mit Beispielen verallgemeinern? Der Hypothesenraum ist aber wichtig!
  - Problem: Zielkonzept evtl. nicht im Hypothesenraum.
  - Lösung: Hypothesenraum, der alle möglichen Hypothesen enthält?
- Ein induktives Lernsystem, das keine apriori Annahmen über die Identität des Zielkonzepts macht, hat keine rationale Basis, um unbekannte Instanzen zu klassifizieren.
- Induktives Lernen erfordert Vorannahme „Inductive Bias“
- Bsp: Auswendiglerner haben keine Annahme. Specific-to-General: Alle Instanzen sind negativ-Instanzen bis Gegenteil bekannt ist.
- Je strenger die Vorannahmen, desto mehr unbekannte Instanzen werden erkannt.
- Vorzugskriterien (*Bias*)
  - Vorschrift als Grundlage für entwickelte Hypothesen
  - Beispiele: Verständlichkeit, Klassifikationsgenauigkeit, Messaufwand, Aufwand
  - *Hypothesenraumbias*: Beschränke den Raum von Hypothesen (Polynome, 3-NN…)
  - *Präferenzbias*: Ordne Hypothesenraum. Bevorzuge Hypothesen mit wenig Disjunktionen oder kleinen Entscheidungsbäumen.
  - Problem: Keine Funktion h ist mit allen Trainingsbeispielen kosistent zB wegen Rauschen.
  - Lösungsansätze:
    - Hypothesenraumbias anpassen
      - Komplexe Hypothesen. Aber: Overfitting!
    - Präferenzbias anpassen
      - Bestmögliche Hypothese wählen

3 Lerntheorie

3.1 Definition des Lernens

- Grundsatz von Lernmaschinen
  - Entities should not be multiplied beyxond necessity
  - Löse nie ein Problem komplizierter als nötig, denn die einfachste richtige Erklärung ist die beste.
- Eine *lernende Maschine* wird bestimmt durch Hypothesenraum $H = \{h_\alpha : \alpha \in A\}$ und Lernverfahren (Methode um α_{opt} mit Lernbeispielen zu finden. Dazu Fehlerfunktion und Optimierungsmethode notwendig.)
- Probleme: Welche Maschine? Welcher Hypothesenraum? Welches Lernverfahren? Welche Hypothesen wählen?
- Probleme beim Lernen:
  - *Statistisches Problem* (zu großer Hypothesenraum. Mehrere Hypothesen sind gleich gut)
    - Beispiel: Regression, kann falsch erkannt werden.
  - *Komplexitätsproblem*: Wegen Problemkomplexität kann die optimale Lösung nicht garantiert im Hypothesenraum gefunden werden ⇒ Suboptimale Lsg.
    - Beispiel: Klassen nicht linear trennbar
  - *Repräsentationsproblem*: Hypothesenraum enthält keine ausreichend gute Approximation der Zielfunktion.
    - Beispiel: Keine strikte Klassentrennung möglich.
- Lernproblem
  - $\{Attr_1, \dots, Attr_n\} \times \{true, false\} = Konzept$
  - $R^N \times \{Klasse_1, \dots, Klasse_n\} = Klassifikation$
  - $R^N \times R = Regression/Prognose$

3.2 Fehler

3.2.1 Lernen durch Fehlerminimierung

Fehler: Schätze h so, dass der Fehler $E(h)$ minimal ist mit:

$$E(h_\alpha) = Func((h_\alpha(\vec{x}) \leftrightarrow y), P(\vec{x}, y))$$

- h_α ist die zu lernende Abbildung
- $(\overrightarrow{x_n}, y_n) \in X \times Y$ sind die Lernbeispiele
- $P(\vec{x}, y)$ ist die Wsk der Lernbeispiele
- $\leftrightarrow$ ist der Vergleich zwischen Hypothesenausgabe und Sollausgabe.

Lernen: Wie kann E effektiv minimiert werden?

3.2.2 Empirischer und realer Fehler

- *Realer Fehler* $E(h) = \int l(h(\vec{x}), y) d \cdot P(\vec{x}, y)$ bzgl aller realer Daten ist nicht berechenbar, aber abschätzbar.
- *Empirischer Fehler* $E_D(h) = \frac{1}{|D|} \sum_{(\vec{x},y) \in D} l(h(\vec{x}), y)$
  - Lernfehler für D = Lerndaten
  - Verifikationsfehler für D = Verifikationsdaten
  - Generalisierungsfehler für D = Testdaten
  - Erfordert unabhängige identische Verteilung über Datensätze
- Empirische Fehlerminimierung kann zu Overfitting führen

3.3 Overfitting

Klar. Siehe Folien F03.15

3.4 Hypothesenbewertung, Modellauswahl

3.4.1 Modellauswahl

- Komplexe Maschine => schlechtes Modell: Zu viele Parameter, aber wenig komplexe Maschine => schlechtes Modell: Wenig Parameter
- Metriken beschreiben Modellgüte. Methoden je nach Aufgabenstellung
- Ergebnisklassen der Klassifikation: True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN)
- Konfusionsmatrix: Wahrheitsmatrix für 4 Ergebnisklassen, siehe F03.19
- *Metriken* (Pfeile geben an ob Wert groß oder klein sein sollte)
  - ↓Klassifikationsfehler: $\frac{errors}{total} = \frac{FP+FN}{TP+FN+FP+TN}$
  - ↑Güte (=1-Fehler): $\frac{correct}{total} = \frac{TP+TN}{TP+FN+FP+TN}$
  - ↓FP-Rate/False Alarm Rate FA: $FPR = \frac{FP}{FP+TN}$
  - ↓FN-Rate/Miss Rate MR: $FNR = \frac{FN}{TP+FN} = 1 - TPR$
  - ↑Genauigkeit/Precision: $P = \frac{TP}{TP+FP}$
  - ↑TP-Rate/Positive Rückmeldung/Recall: $TPR = R = \frac{TP}{TP+FN} = 1 - FNR$
  - ↑F1-Maß: $F_1 = \frac{2}{\frac{1}{R}+\frac{1}{P}}$
- Finden guter Modelle
  - Ansätze: TPR/FÜR Graph (ROC), Precision-Recall Graph
  - *Cross-Validation*: Teile wiederholt Daten in Lern- und Validierungsdaten, bestimme jeweils Hypothesen und berechne darauf Metriken (Generalisierung).
    - n-fold-crossvalidation: Teile Daten in n Mengen, trainiere auf n-1 Mengen und teste auf einer.
    - Leave one out: Lerne ohne Beispiel und addiere Fehler für weggelassene Beispiele.

3.4.2 Techniken für Modellfindung

- Bootstrap: Ziehe zufällig mit Zurücklegen aus D jeweils $\#|D|$ Beispiele je m Mal.
  Bestimme jeweils Modellparameter, wiederhole und bestimme jeweils Mittelwert und Varianz der Metriken des Modells.
  Analysiere dann Güte/Stabilität und finde Modell mit höherer Güte.
- Bagging: Bootstrap Aggregation
  Verwende mehrere Lernmaschinen, ziehe $n < |D|$ Beispiele mit Zurücklegen und bestimme jeweilige Modelle.
  Kombiniere dann die Lernmaschinen zB mit gewichteter Summe.
  Das resultierende Modell hat eine höhere Güte und kann Abschätzungen über Stabilität machen (große Abweichungen in einzelnen Modellen = Instabil)
- Bagging für Klassifikation
  - Zerlege D in zB D_1, D_2, D_3.
  - Wähle D_1 und bestimme Modell M_1
  - Wähle $D_2 \subset D$ sodass ~50% durch M_1 korrekt geschätzt werden erstelle damit M_2.
  - Wähle $D_3 \subset D$ sodass M_1 und M_2 auf D_3 gegensätzlich schätzen und erstelle damit M_3.
    - Kombiniere die Modelle zu
      $$M = \begin{cases} M_1 & wenn\ M_1 = M_2 \\ M_3 & sonst \end{cases}$$
- AdaBoost (Adaptive Boosting)
  Gegeben: Lernmenge D mit $|D| = n$ Beispielen.
  Erstelle iterativ in k Stufen einen komplexen Klassifikator aus k zusammengesetzten Klassifikatoren.
  Ziehe Lernbeispiele entsprechend Gewichtung. Die Gewichtung entspricht dem Klassifikationsergebnis des zuletzt generierten „schwachen“ Klassifikators.
  Das Gewicht von korrekt klassifizierten Beispielen wird verringert, das Gewicht von falsch klassifizierten Beispielen wird erhöht.
  Mit der neuen Lernmenge wird der nächste Klassifikator bestimmt. Die k Klassifikatoren werden zu einem Ensemble zusammengefasst und legen durch gewichteten Mehrheitsentscheid die Klasse eines neuen Beispiels fest.
  - Algorithmus und Bsp siehe Folien F03.28ff

3.5 Lernbarkeit

3.5.1 PAC (Probably Approximate Correct)

- Gegeben:
  - Menge X von Instanzen der Länge n.
  - Konzept C
  - Hypothesenraum H
  - Lerndatenmenge D
- Es kann keine korrekte ($h(x) = c(x) \forall x \in D$) Hypothese gefunden werden, aber eine ε-genaue: $E_D(h) \leq \varepsilon, 0 < \varepsilon < \frac{1}{2}$ („*Approximate Correct*“).
- Diese kann nicht sicher, aber mit Wahrscheinlichkeit $1 - \delta, 0 < \delta < \frac{1}{2}$ („*Probably*“) gefunden werden.
- Das Problem „Finden der Hypothese“ hat polynomialen Aufwand $\sim \frac{1}{\delta}, \frac{1}{\varepsilon}, n$ und Speicheraufwand $\sim C$.
- Die Anzahl der benötigten Lerndaten ist $m \geq \frac{1}{\varepsilon}\left(\ln\left(\frac{1}{\delta}\right) + \ln(|H|)\right)$ („Stichprobenkomplexität“).
  - Gewünschte Sicherheit und Hypothesenraum korrelieren positiv mit benötigten Lerndaten, zulässige Fehler korreliert negativ mit benötigten Lerndaten.

3.6 VC (Vapnik-Chervonenkis) Dimension

Für einen Hypothesenraum $H^\alpha = \{h_\alpha : \alpha \in A\}$ ist die VC Dimension $VC(h_\alpha)$ von H^α gleich der maximalen Anzahl Datenpunkten in S, die von H^α beliebig separiert werden können.

Eine Abbildung/Hypothese h separiert die Daten aus S wenn durch h zwei Mengen definiert werden können:

$$\{x|h(x) = 0\}\ \{x|h(x) = 1\}$$

- VC kann als Maß für Datenkomplexität des Lernens genutzt werden.
- Aussagen über Stichprobenkomplexität, Anzahl Lernbeispiele, …
- Vermutung: Je höher VC(h), desto besser kann die Maschine ein Problem einlernen: FALSCH! (Siehe F03.49)

3.7 Abschätzung des Testfehlers

Lernerfolg hängt von Kapazität der Maschine (möglichst gering), Optimierungsmethode und Lernbeispielen ab.

3.8 SRM (Structural Risk Minimization)

Lösungsansatz: Finde Minimum für empirischen Testfehler, also finde $VC(h_\alpha)$ („Maschine“), N („Beispiele“) und α („Minimum des empirischen Fehlers“).

Idealer Metaalgorithmus: Strukturiere Hypothesenraum in Untermengen von Hypothesen, geordnet nach jeweiligem VC-Wert der Dimensionen darin und suche jeweils Optimum mittels empirischen Fehlers. Stoppe wenn Fehler minimal.

Probleme: Berechnung von VC ist schwer und rechenintensiv. Optimierung: TODO

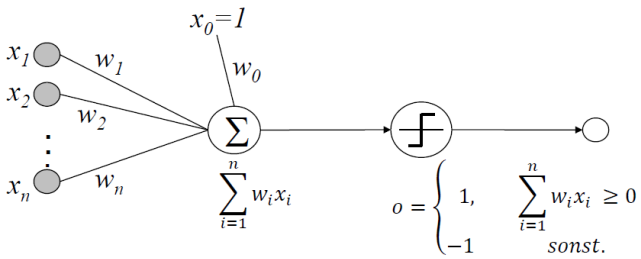
4 Neuronale Netze

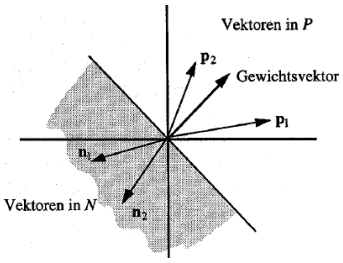
Typische Einsatzfelder: Klassifikation/Mustererkennung, Funktionsapproximation/Regression, Mustervervollständigung.

4.1 Das Perzeptron

Aufbau: Eingabevektor x, Gewichtsvektor w, Summe $\sum$, Aktivierungsfunktion σ (Bsp: $\sigma = \begin{cases} 1, & \sum w_i x_i \geq \varphi \to 0 \\ -1, & \end{cases}$), Ausgabe des Perzeptrons ist

$$o = \sigma(\sum w_i x_i)$$





In R^2 realisiert das Perzeptron eine Geradentrennung (allgemein eine Trennhyperebene).

Achtung: Es gibt ein Gewicht (w_0) mehr als Eingabeparameter! ($|w| = |x| + 1$)

4.1.1 Lernalgorithmus

```
Gegeben Lerndatenmenge P ∪ N
Erzeuge zufälligen Gewichtsvektor w
Während Zähler < Schwelle:
  Wähle zufälliges Trainingsbeispiel x ∈ P ∪ N
  Wenn x ∈ P:
    Wenn w·x > 0 (korrekt klassifiziert):
      Tue nichts.
    Sonst w·x ≤ 0:
      Setze w := w+x
  Sonst x ∈ N:
    Wenn w·x < 0 (korrekt klassifiziert):
      Tue nichts.
    Sonst w·x ≥ 0:
      Setze w := w-x
  Wenn alle x richtig klassifiziert: break
  Zähler++
```

4.1.2 Vom Perzeptron zum Neuron

$|w| \gg |x|$, durch viele Gewichte ist Anpassung langsam. Lösung: Gradientenabstieg.

Gradientenabstieg: Aktivierungsfunktion sollte differenzierbar und nicht zu differenzierbar sein.

Perzeptronen haben eine niedrige Kapazität, durch Kombinationen vieler Perzeptronen („Neuronen“) sind aber viele Funktionen mgl.

Kernel-Perzeptronen: TODO

4.2 Multi Layer Neural Network (MLNN)

- Mehrere versteckte Schichten zwischen Eingabe und Ausgabe. Lernen mittels Backpropagation.
- Nichtlineare Aktivierungsfunktionen

Name	$f(x)$	df/dx
Sigmoid	$\frac{1}{1+e^{-x}}$	$f(x)(1-f(x)$
Tangens Hyperbolicus	$\tanh(x)$	$\big(1+f(x)\big)\big(1-f(x)\big)$
ReLU	$\max(0,x)$	$\begin{cases}1, & x>0\\0, & sonst\end{cases}$
LeakyReLU	$\begin{cases}x, & x>0\\\alpha x, & sonst\end{cases}$	$\begin{cases}1, & x>0\\\alpha, & sonst\end{cases}$

4.2.1 Backpropagation

- Vorgaben: Menge T von Trainingsbeispielen, Lernrate, Netztopologie.
- Ziel: Finde Gewichtsbelegung W, die T wiedergibt.
- Vorgehen: Gradientenabstieg.
- Algorithmus
  - Initialisiere Gewichte zufällig mit kleinen Werten
  - Wiederhole bis Abbruchkriterium
    - Wähle Beispielmuster $d \in T$
    - Bestimme Netzausgabe und Fehler bzgl Sollausgabe.
    - Sukzessives Rückpropagieren des Fehlers δ_j.
    - Anpassung der Gewichtsbelegung

  (Formeln siehe Folien F04.27)
- Gradientenabstieg: Allgemeine Delta-Regel
  - Fehlerfunktion: $E_d(w) = \frac{1}{2}\sum_{k\in O}(t_k - o_k)^2$
    - Für Beispiel d, Gewicht w, Zielausgabe $T = \{t_i\}$, tatsächliche Ausgabe $O = \{o_i\}$.
  - Gewichtsänderung nach Gradienten: $\Delta w_{ij} = -\eta \frac{dE_d}{dw_{ij}}$
    - Für Lernrate η, Fehler E_d, Gewicht w_{ij}.
  - Kettenregel $\frac{dE_d}{dw_{ij}} = \frac{dE_d}{dnet_j} \cdot \frac{dnet_j}{dw_{ij}} = \frac{dE_d}{dnet_j} x_{ij}$
    - Für Propagierungsfunktion net_j.
  - Allgemeine Deltaregel: Ableitung des Fehlers für Ausgabeschicht
    - $\frac{dE_d}{dnet_j} = \frac{dE_d}{do_j} \cdot \frac{do_j}{dnet_j}$ (Siehe Folien F04.30)
    - TODO Tatsächliche Formel, F04.30
- Anpassen der Gewichte
  - Pattern Learning: Nach jedem Beispiel werden Gewichte angepasst, schnelles Lernen.
  - Mini Batch Learning: Kleine Lernmengen werden mit zurücklegen ausgewählt. Gutes Lernen, SotA.
  - Epochen Learning: Mittelung der Gewichtsänderung über alle Beispiele, Anpassung erst danach. „Echter“ Gradientenabstieg, nicht anfällig für Ausreißer.

  Pattern/ Mini Batch Learning sind nur “Stochastische“ Gradientenabstiege, keine echten.

4.3 Probleme und Optimierungen

4.4 Anwendungsbeispiele