

Künstliche Intelligenz

Machine Learning

Dr.-Ing. Stefan Lüdtkke

Universität Leipzig

Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI)

Was ist Lernen

Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task (or tasks drawn from the same population) more efficiently and more effectively the next time.

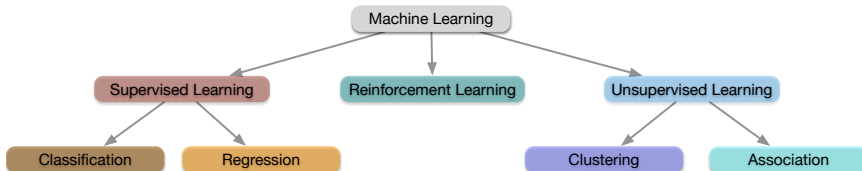
(Herbert Simon)

In vielen Fällen können wir das Verhalten eines Agenten nicht “per Hand” spezifizieren:

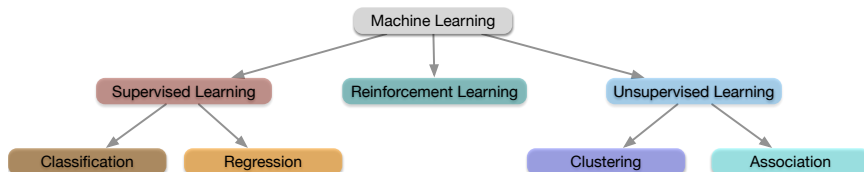
- Problemstruktur zu komplex (natürliche Sprache, Objekterkennung in Videos, ...)
- Formalisierung von Expertenwissen sehr schwierig (*Knowledge Acquisition Bottleneck*)
- Maschine findet überlegene Lösungen

Machine Learning

- **Machine Learning** (maschinelles Lernen) ist der Forschungsbereich, der sich mit dem Design und der Analyse von Algorithmen beschäftigt, die aus Daten lernen und Vorhersagen treffen können
- Machine Learning-Algorithmen lassen sich grob in drei Gruppen unterteilen:
 - supervised learning (überwachtes Lernen)
 - unsupervised learning (unüberwachtes Lernen)
 - reinforcement learning (verstärkendes Lernen)



Supervised Learning

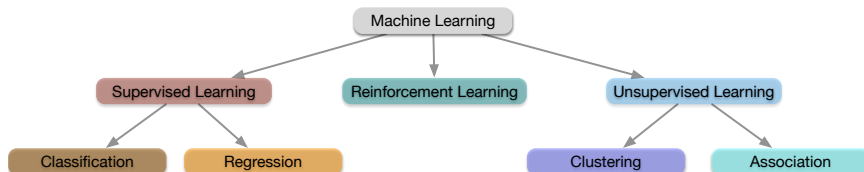


- **Supervised Learning:** Daten bestehen aus Paaren (x, y) von Eingaben $x \in X$ und zugehörigen Ausgaben $y \in Y$, die in einem (unbekannten) Zusammenhang $f(x) = y$ stehen.
- Ziel ist es, eine Funktion \hat{f} mit $\hat{f} \approx f$ zu lernen.
- Kann weiter unterteilt werden in
 - **Klassifikation:** Y ist eine diskrete, endliche Menge (Klassen)
 - **Regression:** Y ist eine unendliche Menge (z.B. reelle Zahlen)

Supervised Learning

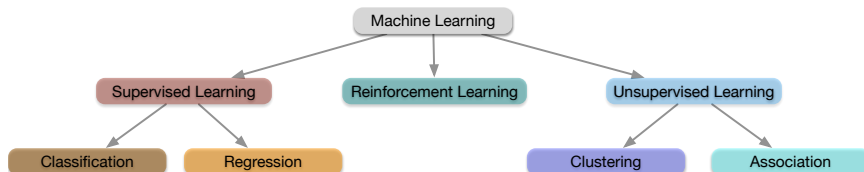
- Aufgabe also: Inferriere eine Funktion \hat{f} aus **gelabelten Trainingsdaten** (Y wird auch als *Label* bezeichnet)
- Die gelernte Funktion \hat{f} kann dann genutzt werden, um für neue Daten x das Label vorherzusagen
- Erfordert, dass der Lernalgorithmus auf geeignete Weise den Zusammenhang von X und Y *generalisiert*
- (Später mehr zur Generalisierbarkeit)

Unsupervised Learning



- **Unsupervised Learning:** Haben nur Eingabe-Daten X
- Ziel ist die zugrundeliegende Struktur der Daten zu lernen, z.B. eine Wahrscheinlichkeitsdichte $p(x)$
- Beispiele:
 - **Dimensionsreduktion:** Lerne eine Abbildung $f(x) = z$, wobei Z geringe Dimensionalität als X hat und *interessante* Eigenschaften in Z erhalten bleiben (z.B. Ähnlichkeit)
 - **Clustering:** Teile die Daten so in k Gruppen (Cluster) auf, sodass die Daten innerhalb einer Gruppe ähnlicher zueinander sind als die Daten verschiedener Cluster

Reinforcement Learning



- **Reinforcement Learning:** Welche Aktionen muss ein Agent ausführen, um seinen Reward zu maximieren?
- Konkret: Agent bekommt Eingaben x (Beobachtungen), muss Aktion wählen, bekommt *Reward*

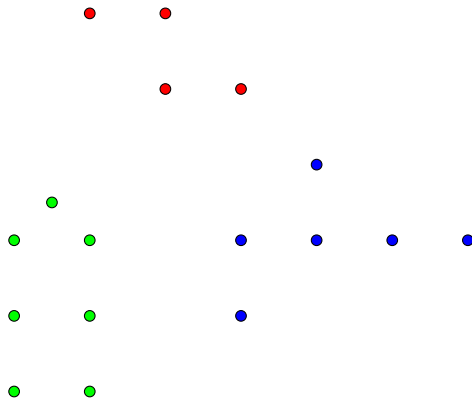
Weiteres Vorgehen

- Beispiel für Clustering-Algorithmus: K-Means
- Beispiel für Klassifikations-Algorithmus: Entscheidungsbaum
- Beispiel für Regressions-Algorithmus: Künstliche Neuronale Netze
- Evaluation von Klassifikatoren, Generalisierbarkeit

Clustering

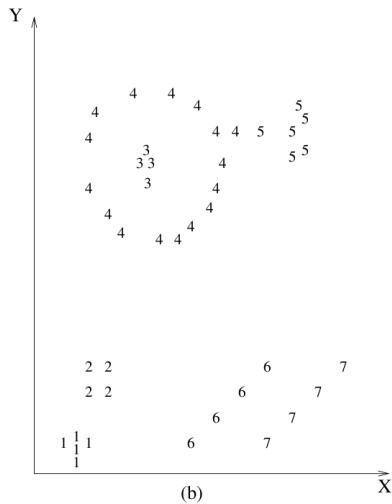
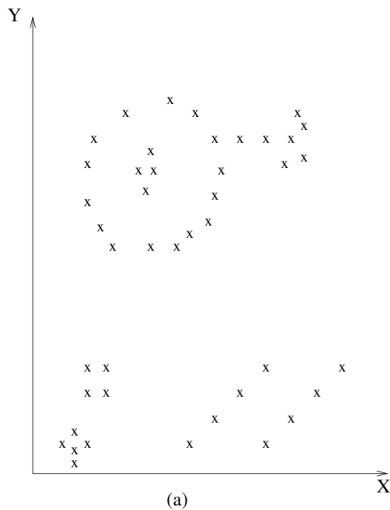
- Ziel: Aufteilung der Daten O in Teilmengen (Cluster) C_1, \dots, C_k , sodass
 - Zwischen Objekten innerhalb eines Clusters möglichst große Ähnlichkeit besteht
 - Zwischen Objekten in verschiedenen Clustern möglichst geringe Ähnlichkeit besteht
- Verschiedene Aufteilungsstrategien: Disjunkte Teilmengen, Hierarchische Cluster, probabilistische Aufteilung (Wahrscheinlichkeit $p(o | C_i)$, dass Sample o zu Cluster C_i gehört)

Clustering: Beispiel (hier im \mathbb{R}^2)



→ disjunkte Aufteilung in 3 Klassen

Beispiel für nicht-konvexe Cluster



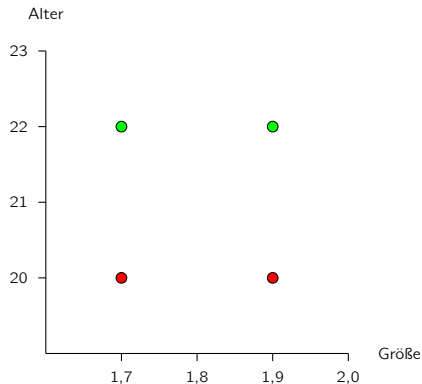
Ähnlichkeitsmaß

- zentrale Bedeutung (neben dem verwendeten Algorithmus)
- z. B. Minkowski Distanz (im \mathbb{R}^n):

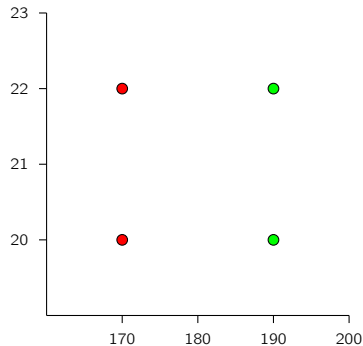
$$d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} = \|\mathbf{x} - \mathbf{y}\|_p$$

- für $p = 1$: Manhattan Distanz
- für $p = 2$: Euklidische Distanz
- Probleme:
 - implizite Annahme, dass in jeder Dimension gleiche Skalierung
 - Behandlung von nominalen Daten (z. B. Wochentage, Spielertyp): Ordnung?, Subtraktion?
 - Berücksichtigung von Domänenwissen

Einfluss der Skalierung auf die Clusterbildung



ungünstige Skalierung
(x-Achse stauchen)



günstige Skalierung
(x-Achse dehnen)

→ Abhilfe: Gewichtung, z. B. durch Normalisierung

k-means Algorithmus

Charakteristika:

- Parameter $k \in \mathbb{N}$ bestimmt die Anzahl der Cluster
- jeder Cluster C_i wird durch Zentroid $\mathbf{c}_i \in \mathbb{R}^n$ repräsentiert
→ Mittelwert bezüglich aller in C_i enthaltenen Punkte, d. h.

$$\mathbf{c}_i = \left(\frac{1}{|C_i|} \sum_{\mathbf{p} \in C_i} p_1, \dots, \frac{1}{|C_i|} \sum_{\mathbf{p} \in C_i} p_n \right)$$

- *Ziel:* wähle Cluster $C_1, \dots, C_k \subseteq O$ so, dass $\{C_1, \dots, C_k\}$ eine Partition von O ist und

$$E(C_1, \dots, C_k) = \sum_{i=1}^k \sum_{\mathbf{p} \in C_i} |\mathbf{p} - \mathbf{c}_i|^2$$

(intra-cluster Varianz) minimiert wird

k-means Algorithmus

- 1 wähle k zufällige Punkte $\mathbf{c}_1, \dots, \mathbf{c}_k \in \mathbb{R}^n$ als Zentroide
- 2 $\forall \mathbf{o} \in O$: ordne \mathbf{o} dem nächsten Zentroid zu, d. h. \mathbf{o} wird \mathbf{c}_i zugeordnet, falls

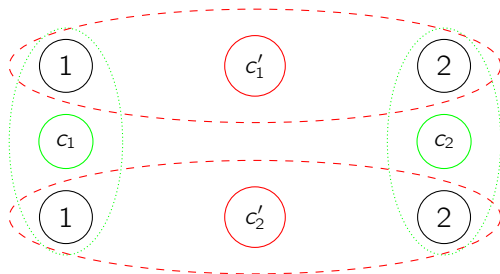
$$d(\mathbf{o}, \mathbf{c}_i) = \min_{1 \leq i \leq k} d(\mathbf{o}, \mathbf{c}_i),$$

wobei $d(,)$ eine Distanzfunktion ist

- 3 Sei C_i die Menge der Objekte, die \mathbf{c}_i zugeordnet wurden. Berechne ausgehend von C_i den Zentroid \mathbf{c}_i neu.
- 4 falls sich im vorherigen Schritt mindestens ein Zentroid geändert hat, gehe zu 2
andernfalls: Stop; C_1, \dots, C_k ist eine Partitionierung von O

Bemerkungen zum k -means Algorithmus

Beobachtung: resultierende Cluster können stark von der Wahl der initialen Zentroide abhängen

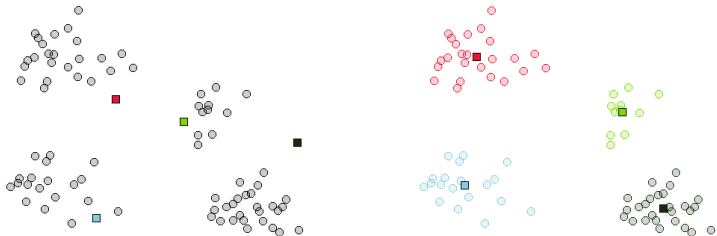


Bemerkungen zum k -means Algorithmus

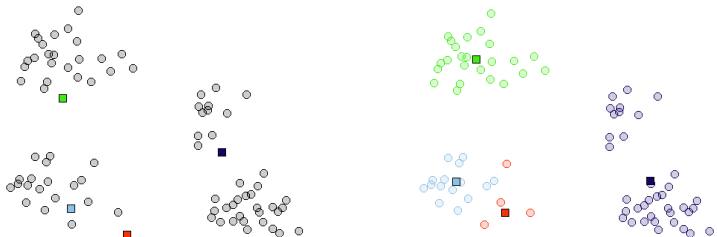
- resultierendes Clustering kann stark von Wahl der initialen Zentroide abhängen \rightarrow keine Garantie dafür, dass globales Minimum bezüglich der Zielfunktion $E(C_1, \dots, C_k)$ gefunden wird
- Abhilfe: mehrere Durchläufe mit unterschiedlichen Startkonfigurationen \rightarrow wähle Resultat mit minimalen Wert für E
- genau genommen: resultierende Anzahl an Clustern kann auch kleiner k sein (Beispiel?)
- initiale Zentroide sollten möglichst weit voneinander entfernt sein

Bemerkungen zum k -means Algorithmus

gutes Resultat:



schlechtes Resultat:



Clustering: Weitere Aspekte

- Wahl von k : Größeres k sorgt immer für geringeres E , aber Clustering ist ja trotzdem nicht immer “besser”
→ Methoden zur Beurteilung, die unabhängig von k sind (werden wir hier nicht weiter behandeln)
- Nachteil von K-Means: Nimmt implizit an, dass Cluster sphärisch sind
→ Methoden für Cluster beliebiger Form, z.B. DBSCAN (hier auch nicht weiter behandelt)

Klassifikation: Entscheidungsbäume

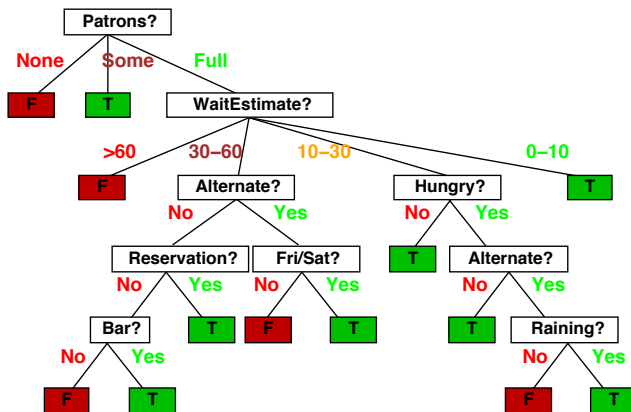
Klassifikation: Entscheidungsbäume

- Wir stehen vor einem Restaurant. Werden wir warten, bis wir eingelassen werden?
- Eingabe X ist Vektor aus mehreren *Attributen*: Alternatives Restaurant (Alt); hat das Restaurant eine Bar (Bar); ist es Freitag oder Samstag (Fri); sind wir hungrig (Hun), wie voll ist es (Pat), wie teuer (Price), regnet es (Rain), haben wir reserviert (Res), Art des Restaurants (Type), wie lange werden wir warten müssen (Est).

Example	Attributes										Target
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0-10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0-10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30-60	T

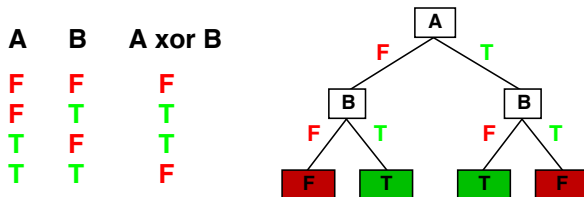
Entscheidungsbäume

- Idee: Repräsentiere Funktion $\hat{f}(x) = y$ als Baum
- Achtung: Der Name *Entscheidungsbaum* ist etwas täuschend: Die Ausgaben sind *Klassen*, nicht unbedingt Entscheidungen



Ausdrucksstärke

Entscheidungsbäume können jede beliebige Funktion der Eingabe-Attribute repräsentieren:



- D.h. es gibt für Trainingsmenge einen Entscheidungsbau, aber dieser wird nicht unbedingt gut für neue Daten funktionieren
- Ziel: Für gegebene Trainingsdaten einen kompakten Entscheidungsbaum konstruieren

Entscheidungsbaum-Lernen

Ziel: Finde einen kompakten Baum, der mit den Trainingsdaten konsistent ist:

Idee: wähle rekursiv das “relevanteste” Attribut als Wurzel von neuem Teilbaum:

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value  $v_i$  of best do
      examplesi ← {elements of examples with best =  $v_i$ }
      subtree ← DTL(examplesi, attributes − best, MODE(examples))
      add a branch to tree with label  $v_i$  and subtree subtree
    return tree
```


2 Aufbau von Bäumen: Prinzip

Überlegung: Betrachte eine aktuelle Teilmenge $\mathcal{A} \subseteq \mathcal{D}$ der Testdaten.

- Wenn alle Instanzen in \mathcal{A} derselben Klasse $\omega_{\mathcal{A}}$ angehören, ist die Teilmenge rein. Dann kann ein Blattknoten angelegt werden, der der Klasse $\omega_{\mathcal{A}}$ zugeordnet wird.
- Wenn die Instanzen in \mathcal{A} unterschiedlichen Klassen angehören, muss eine Entscheidung getroffen werden:
 - Trotzdem einen Blattknoten anlegen und Klassifikation gemäß der Mehrheit.
 - Einen inneren Knoten anlegen, und \mathcal{A} in mehrere Teilmengen \mathcal{A}_j aufspalten. Für die Teilmengen \mathcal{A}_j wird dann das Verfahren rekursiv aufgerufen; die entstehenden Teilbäume werden dann über geeignet bezeichnete Äste mit dem neuen inneren Knoten verbunden.

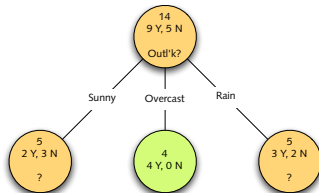
2 Beispiel

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
3	O	H	H	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
7	O	C	N	S	Y
8	S	M	H	W	N
9	S	C	N	W	Y
10	R	M	N	W	Y
11	S	M	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
14	R	M	H	S	N



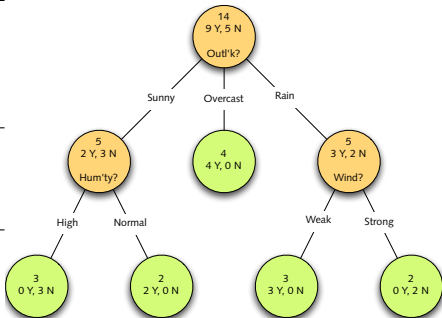
2 Beispiel

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y
3	O	H	H	W	Y
7	O	C	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
10	R	M	N	W	Y
14	R	M	H	S	N



2 Beispiel

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y
3	O	H	H	W	Y
7	O	C	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
10	R	M	N	W	Y
14	R	M	H	S	N



2 Synthese von Bäumen

Beim Aufbau von Bäumen sind folgende Fragen zu klären:

- Sollen an inneren Knoten lediglich binäre Eigenschaften getestet werden oder sind auch komplexere n -äre Spaltungen erlaubt?
- Auf Basis welcher Kriterien sollte eine Eigenschaft für einen Knotentest ausgewählt werden?
- Unter welchen Umständen sollte ein Knoten als Blatt deklariert werden?
- Wenn ein Baum „zu groß“ wird – wie kann man ihn verkleinern und vereinfachen (also „beschneiden“, „zurückstutzen“?)
- Falls ein Blattknoten gemischt ist – wie sollte die Klassenbezeichnung zugewiesen werden?
- Wie sollte man mit fehlenden Merkmalen umgehen?

2 Anzahl der Verzweigungen

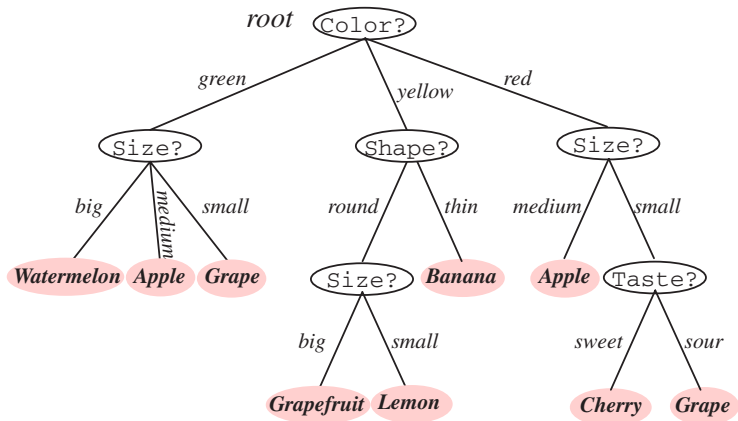
Wählt man eine Eigenschaft mit n Ausprägungen um einen Verzweigungsknoten anzulegen, könnte man eine n -Wege Verzweigung einrichten.

Sollte man das auch?

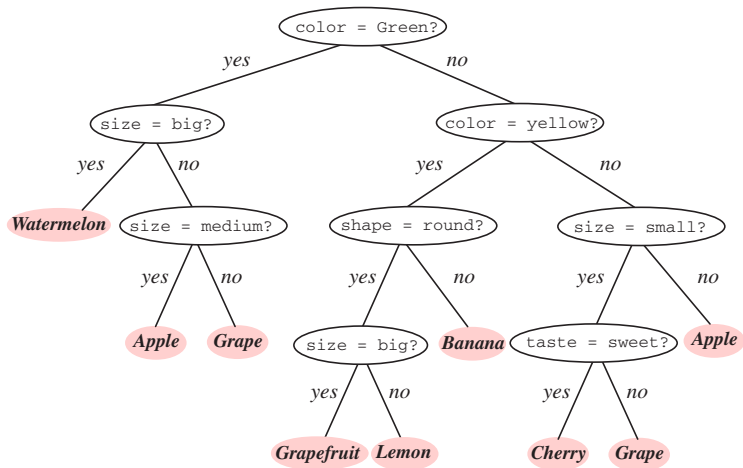
Es gibt Fälle, in denen dies hilfreich ist. Im allgemeinen werden dadurch die Daten aber zu stark fragmentiert, so dass auf der nächsten Baumebene eine sinnvolle Zerlegung behindert wird.

Da zudem jede n -Wege-Zerlegung durch mehrere binäre Zerlegungen repräsentiert werden kann, wird dieser Ansatz oft bevorzugt (vgl. HASTIE et. al.)

2 Früchte eines binären Baums



2 Früchte eines binären Baums



3 Reinheit von Knoten

Grundlegendes Prinzip der Merkmalsauswahl für die Knotenzerlegung ist, möglichst einfache Bäume zu erzeugen, die flach sind und wenige Knoten beinhalten.

(Dies ist eine Variante von OCKHAMS Rasiermesser – ein Prinzip das empfiehlt, stets die einfachste Hypothese zu wählen, die die vorliegenden Daten erklärt. In Abschnitt 9 mehr dazu.)

Um dies zu erreichen, wählen wir an jedem Knoten N diejenige Eigenschaft T , welche die „Reinheit“ der Kinderknoten maximiert.

Es zeigt sich, dass „Unreinheit“ (*impurity*) eines Knotens – intuitiv, die Durchmischtheit der dem Knoten zugeordneten Daten in Bezug auf ihre Klassenzugehörigkeit – einfacher definierbar ist als „Reinheit“.

Die Unreinheit eines Knotens N bezeichnen wir mit $i(N)$.

3 Reinheitsmaße (I)

Für die Definition von $i(N)$ gibt es mehrere plausible Ansätze.

Grundsätzlich soll gelten $i(N) = 0$, falls der Knoten rein ist (d.h., nur Instanzen derselben Klasse enthält). Ausserdem soll gelten, dass $i(N)$ um so größer ist, je stärker ein Knoten durchmischt ist.

Ein Ansatz, „Unreinheit“ zu quantifizieren ist, die Information zu betrachten, die wir brauchen, wenn wir zu einer Instanz $\mathbf{x} \in N$ die zugehörige Klasse bestimmen müssen.

Falls der Knoten rein ist, wissen wir, dass alle Instanzen das gleiche Klassenlabel tragen. Zur Bestimmung des Klassenlabels für \mathbf{x} brauchen wir dann keine weiteren Informationen. Also gilt $i(N) = 0$.

Wie viel Information brauchen wir, falls N ein gemischter Knoten ist, in dem die einzelnen Klassen i mit relativen Häufigkeiten p_i enthalten sind?

3 Reinheitsmaße (I)

Wie viel Information brauchen wir, falls N ein gemischter Knoten ist, in dem die einzelnen Klassen i mit relativen Häufigkeiten p_i enthalten sind, und man uns sagt, dass $\mathbf{x} \in N$ zur Klasse j gehört?

Das sind $\log_2 \frac{1}{p_j} = -\log_2 p_j$ Bits.

Wiederholung: gegeben seien 8 Klassen mit gleicher Wahrscheinlichkeit $1/8$. Wie viele Bits brauchen wir dann, um die Information über die Klasse c zu codieren? Da alle Klassen gleich wahrscheinlich sind, sollten alle Codeworte die gleiche Länge haben. Mit einem Code-Wort von $\log_2 8 = 3$ Bit kann man dann genau 8 unterschiedliche Klassen codieren.

Insgesamt ergibt sich damit der Erwartungswert der Information zu:

$$i(N) = - \sum_{i=1}^c p_i \log_2 p_i$$

3 Reinheitsmaße (I)

Ein mögliches Maß für die Unreinheit eines Knotens ist damit seine Informationsentropie

$$i_{ent}(N) = - \sum_{i=1}^c p_i \log_2 p_i$$

Basis für die Auswahl einer Zerlegung ist dann die Suche nach einer Eigenschaft T , die zwei Kindknoten N_L, N_R erzeugt, die die Unreinheit soweit wie möglich minimiert.

Im Fall der Informationsentropie heißt das: eine Zerlegung, die so wenig wie möglich zusätzliche Information erforderlich macht, um die Klasse einer Instanz zu bestimmen, die im Teilbaum von N liegt. (D. h., ein Test T , der möglichst viel Information liefert.)

3 Reinheitsgewinn

Nehmen wir an, wir finden eine Zerlegung, bei der ein Anteil von P_L Instanzen im linken Kindknoten N_L landet und ein Anteil von $(1 - P_L)$ Instanzen im rechten Kindknoten N_R .

Dann gilt für den Verlust an Unreinheit (Reinheitsgewinn, *Gain*)

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$$

Dieses Maß muss dann durch eine sinnvolle Wahl der Eigenschaft T , mit deren Hilfe die Zerlegung durchgeführt wird, maximiert werden. (Falls wir die Informationsentropie als Maß der Unreinheit betrachten, repräsentiert $\Delta i(N)$ den *Gewinn* an Informationen, in Bits. Eine binäre Entscheidung kann maximal 1 Bit an Information liefern).

Im folgenden Beispiel nehmen wir an, Eigenschaften hätten die einfache Form $m = v$ für ein Merkmal m und ein Merkmalswert v .

3 Beispiel

Nr	Outlook	Temperature	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$\begin{aligned}i(\text{Root}) &= -\left(\frac{9}{14} \log_2 \frac{9}{14} + \frac{5}{14} \log_2 \frac{5}{14}\right) \\ &= 0.940286\end{aligned}$$

3 Beispiel

Spit on *Outlook* = *Overcast*

Outlook = *Overcast*

Nr	O	T	H	W	P
3	O	H	H	W	Y
7	O	C	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y

Outlook \neq *Overcast*

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
10	R	M	N	W	Y
11	S	M	N	S	Y
14	R	M	H	S	N

$$i(N_L) = -(0 \log_2 0 + 1 \log_2 1) = 0$$

$$i(N_R) = -(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}) = 1$$

$$\Delta i(N) = 0.940286 - \frac{4}{14}0 - \frac{10}{14}1 = 0.226000244$$

3 Beispiel

Spit on $Outlook = Rain$

$Outlook = Rain$

Nr	O	T	H	W	P
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
10	R	M	N	W	Y
14	R	M	H	S	N

$Outlook \neq Rain$

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
3	O	H	H	W	Y
7	O	C	N	S	Y
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y
12	O	M	H	S	Y
13	O	H	N	W	Y

$$i(N_L) = -\left(\frac{3}{5} \log_2 \frac{3}{5} + \frac{2}{5} \log_2 \frac{2}{5}\right) = 0.971$$

$$i(N_R) = -\left(\frac{6}{9} \log_2 \frac{6}{9} + \frac{3}{9} \log_2 \frac{3}{9}\right) = 0.918$$

$$\Delta i(N) = 0.94 - \frac{5}{14} 0.971 - \frac{9}{14} 0.918 = 0.003$$

3 Beispiel

Spit on *Outlook = Sunny*

Outlook = Sunny

Nr	O	T	H	W	P
1	S	H	H	W	N
2	S	H	H	S	N
8	S	M	H	W	N
9	S	C	N	W	Y
11	S	M	N	S	Y

Outlook ≠ Sunny

Nr	O	T	H	W	P
3	O	H	H	W	Y
4	R	M	H	W	Y
5	R	C	N	W	Y
6	R	C	N	S	N
7	O	C	N	S	Y
10	R	M	N	W	Y
12	O	M	H	S	Y
13	O	H	N	W	Y
14	R	M	H	S	N

$$i(N_L) = -\left(\frac{2}{5} \log_2 \frac{2}{5} + \frac{3}{5} \log_2 \frac{3}{5}\right) = 0.971$$

$$i(N_R) = -\left(\frac{7}{9} \log_2 \frac{7}{9} + \frac{2}{9} \log_2 \frac{2}{9}\right) = 0.764$$

$$\Delta i(N) = 0.94 - \frac{5}{14} 0.971 - \frac{9}{14} 0.764 = 0.102$$

3 Beispiel

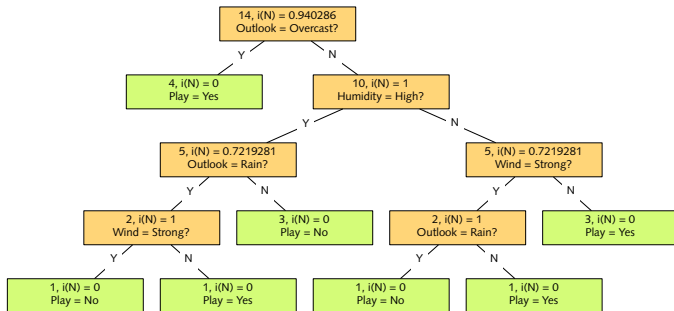
Insgesamt ergibt sich:

Var	Val	$\Delta i(N)$
Outlook	Overcast	0.226000244
Outlook	Rain	0.003184853
Outlook	Sunny	0.102243564
Temperature	Cool	0.014956070
Temperature	Hot	0.025078174
Temperature	Mild	0.001339742
Humidity	High	0.151835501
Humidity	Normal	0.151835501
Wind	Strong	0.048127030
Wind	Weak	0.048127030

Daraus folgt: der beste erste Split ist $Outlook = Overcast$

3 Beispiel

Wenn man auf diese Weise den Baum aufbaut, erhält man schließlich:

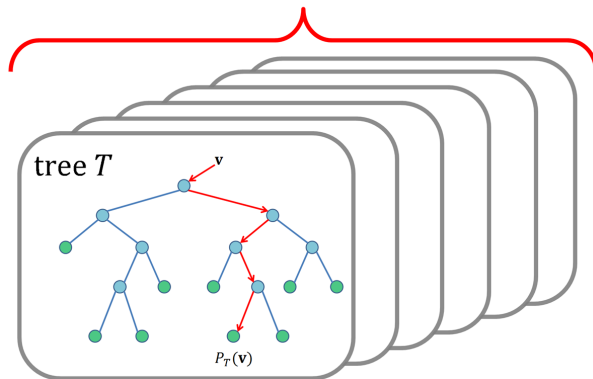


Warum sieht dieser Baum anders aus, als der Ausgangsbaum?

(1) Wir verwenden nur binäre Splits

(2) Die Split-Operation ist eine *lokale* Optimierung

Decision Forest



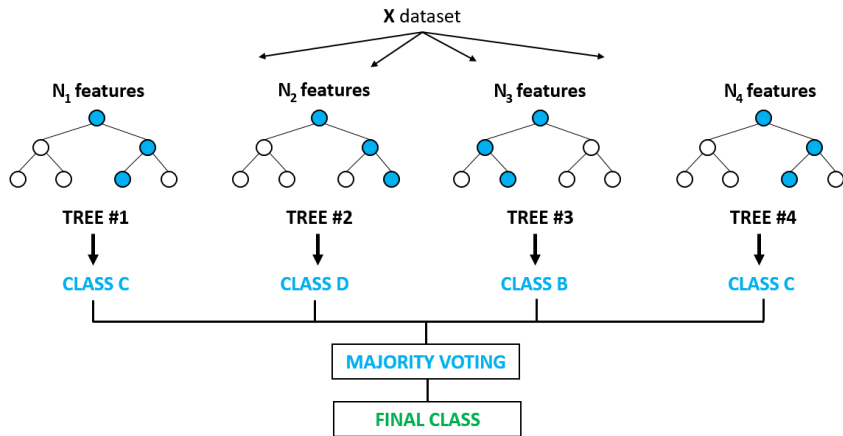
- Idee: Nutze mehrere Entscheidungsbäume, die entweder mit verschiedenen Teilmengen der Attribute (Merkmale) oder Teilmengen der Samples trainiert werden, aggregiere Ergebnisse (Mehrheitsentscheidung)

Merkmale vs. Samples

Sample	weekend	dayTime	frequency	ADJ	ADV	...
T_1	1	morning	12	1	2	
T_2	0	noon	54	6	4	
T_3	0	noon	32	4	0	
...		
T_n	0	evening	102	9	13	...

- Samples: T_1 bis T_n
- Merkmale: weekend, dayTime, frequency, ADJ, ADV, ...

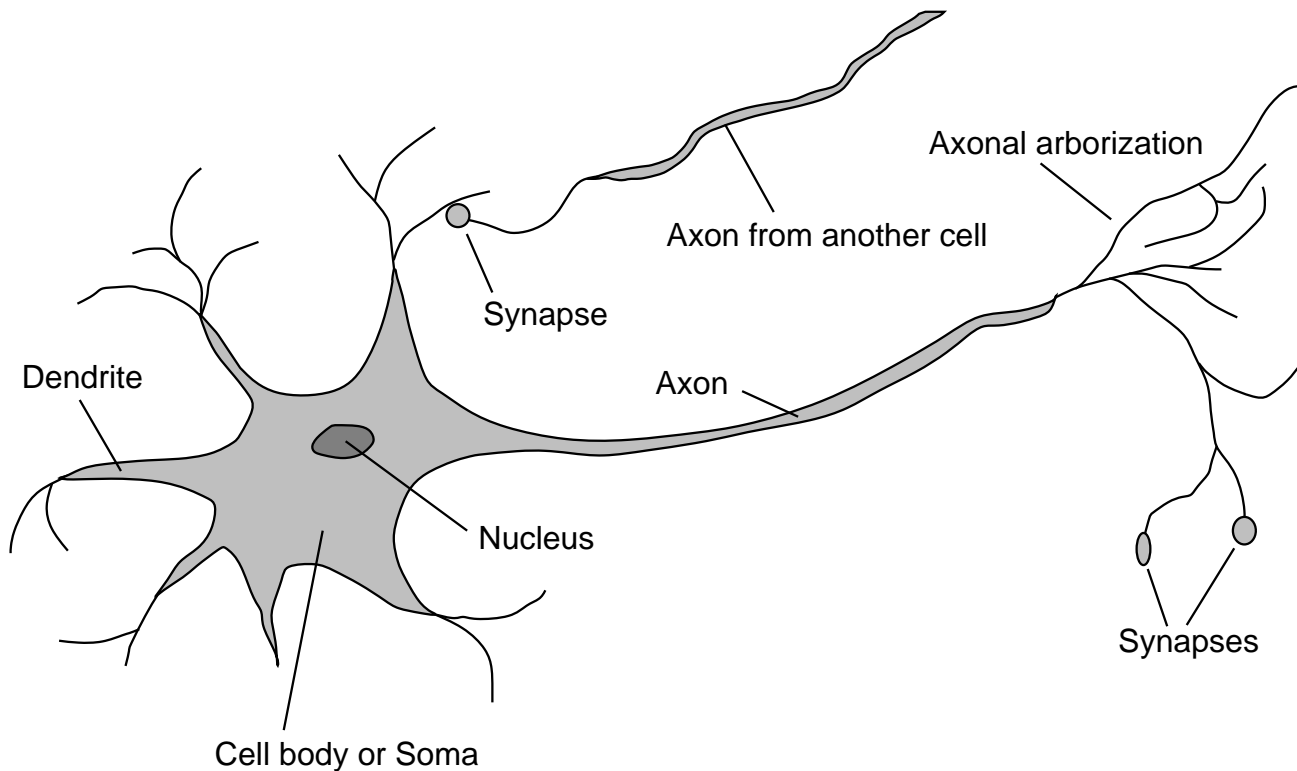
Random Forest



Regression: Künstliche Neuronale Netze

Brains

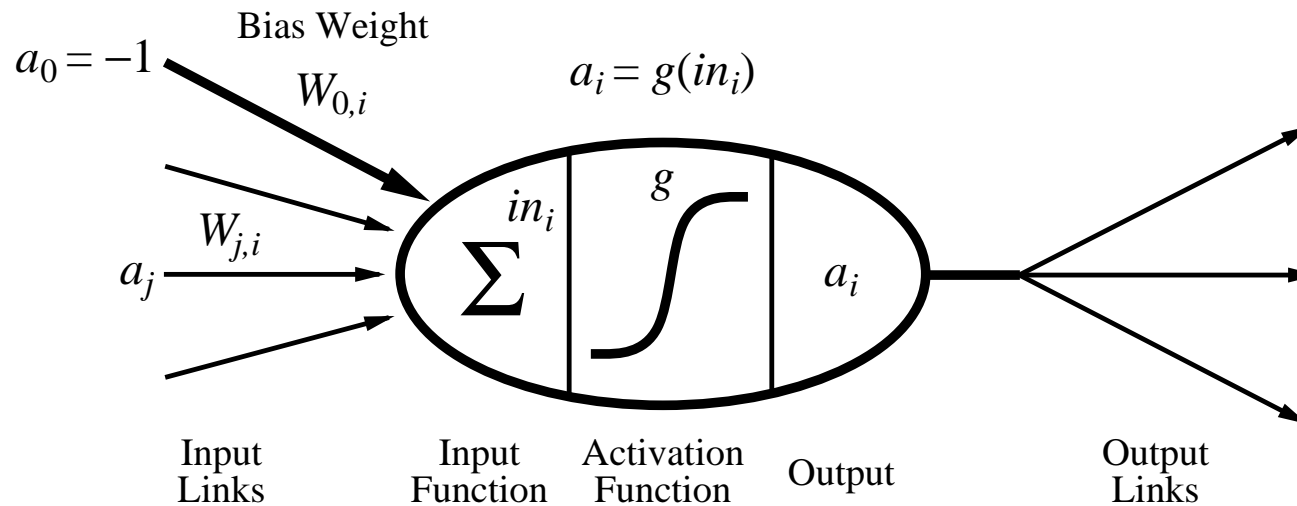
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy “spike trains” of electrical potential



McCulloch–Pitts “unit”

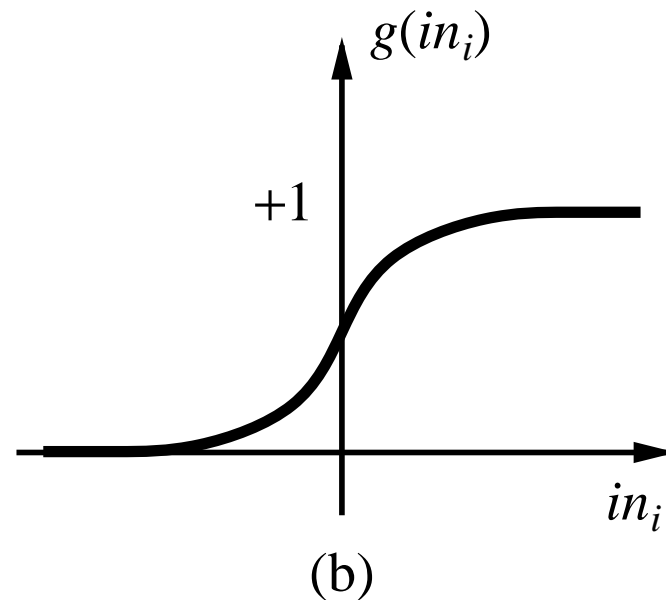
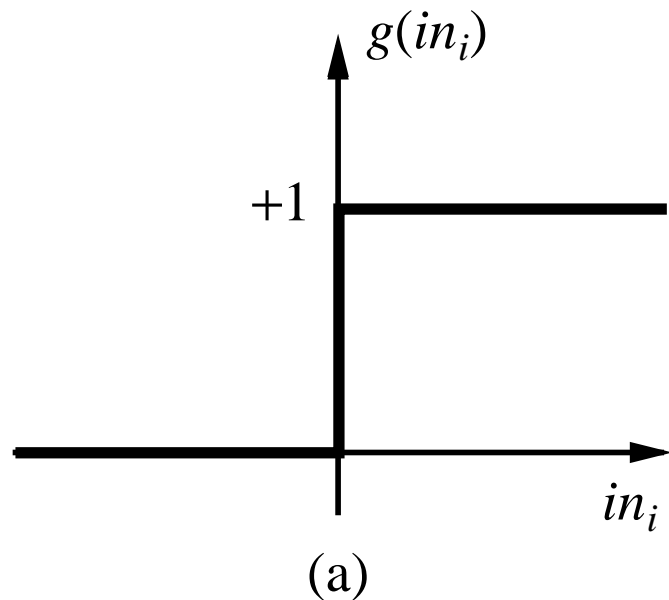
Output is a “squashed” linear function of the inputs:

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions

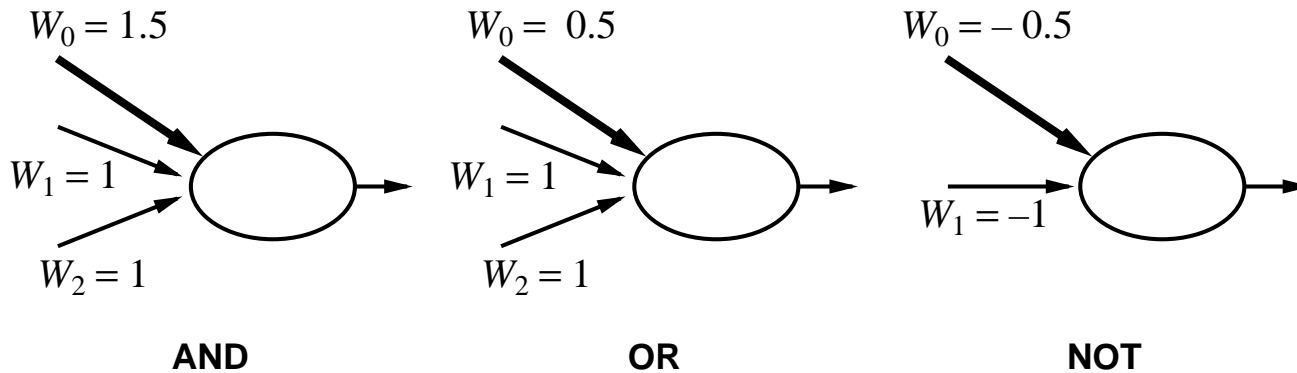


(a) is a **step function** or **threshold function**

(b) is a **sigmoid function** $1/(1 + e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

Network structures

Feed-forward networks:

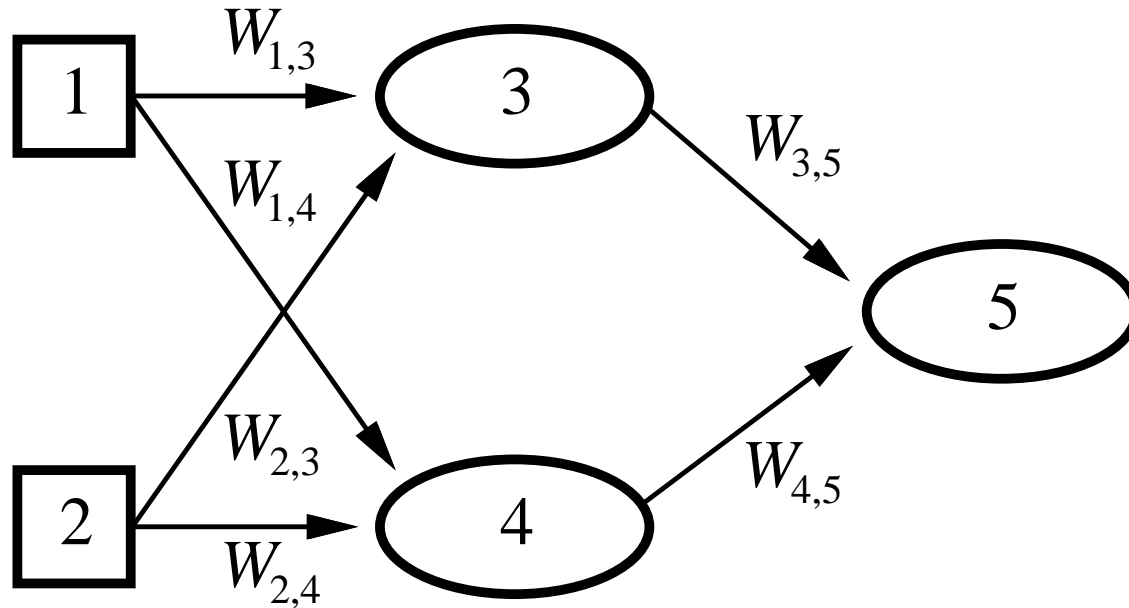
- single-layer perceptrons
- multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Recurrent networks:

- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
 $g(x) = \text{sign}(x)$, $a_i = \pm 1$; **holographic associative memory**
- Boltzmann machines use stochastic activation functions,
 \approx MCMC in Bayes nets
- recurrent neural nets have directed cycles with delays
 \Rightarrow have internal state (like flip-flops), can oscillate etc.

Feed-forward example

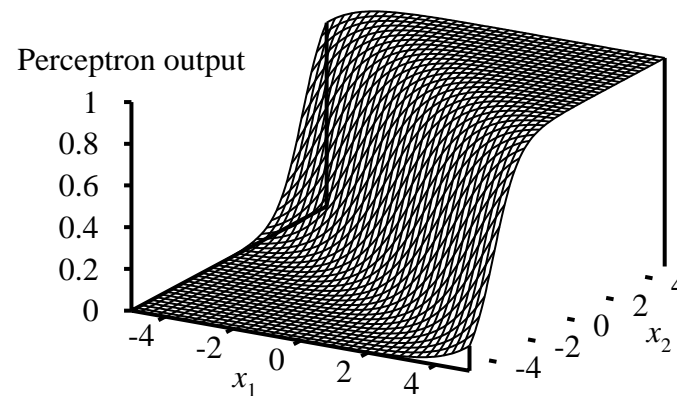
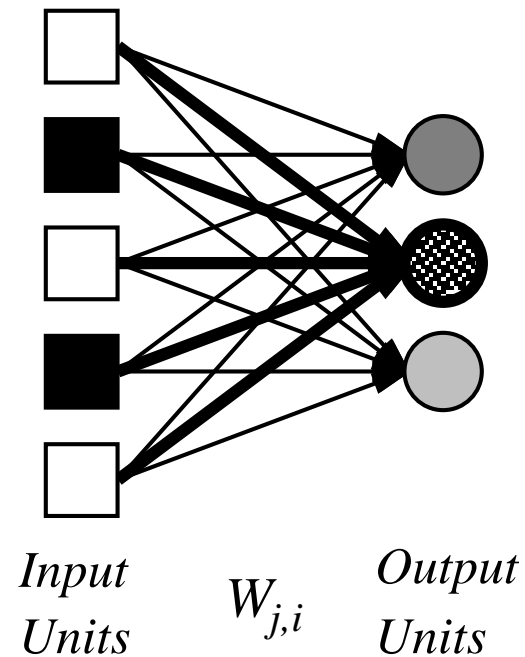


Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

Adjusting weights changes the function: do learning this way!

Single-layer perceptrons



Output units all operate separately—no shared weights

Adjusting weights moves the location, orientation, and steepness of cliff

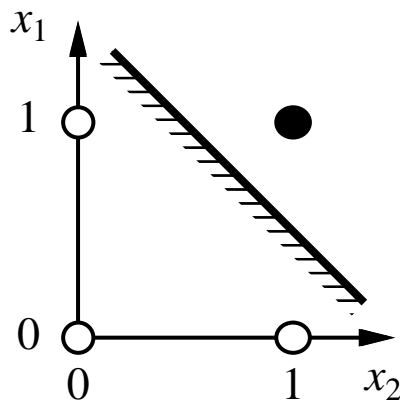
Expressiveness of perceptrons

Consider a perceptron with $g = \text{step function}$ (Rosenblatt, 1957, 1960)

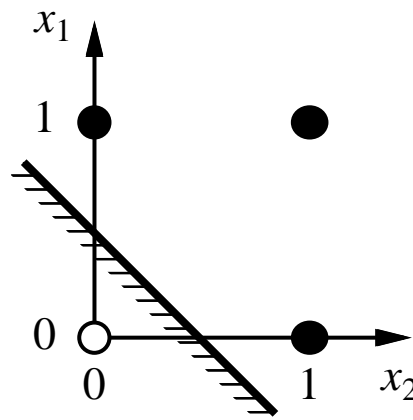
Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a **linear separator** in input space:

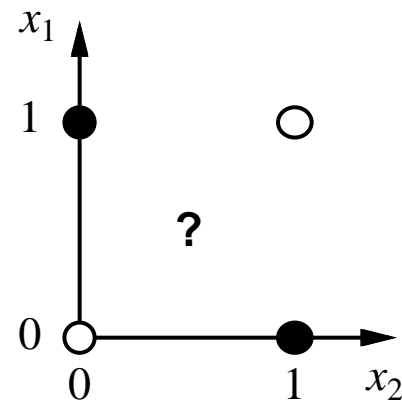
$$\sum_j W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$



(a) x_1 **and** x_2



(b) x_1 **or** x_2

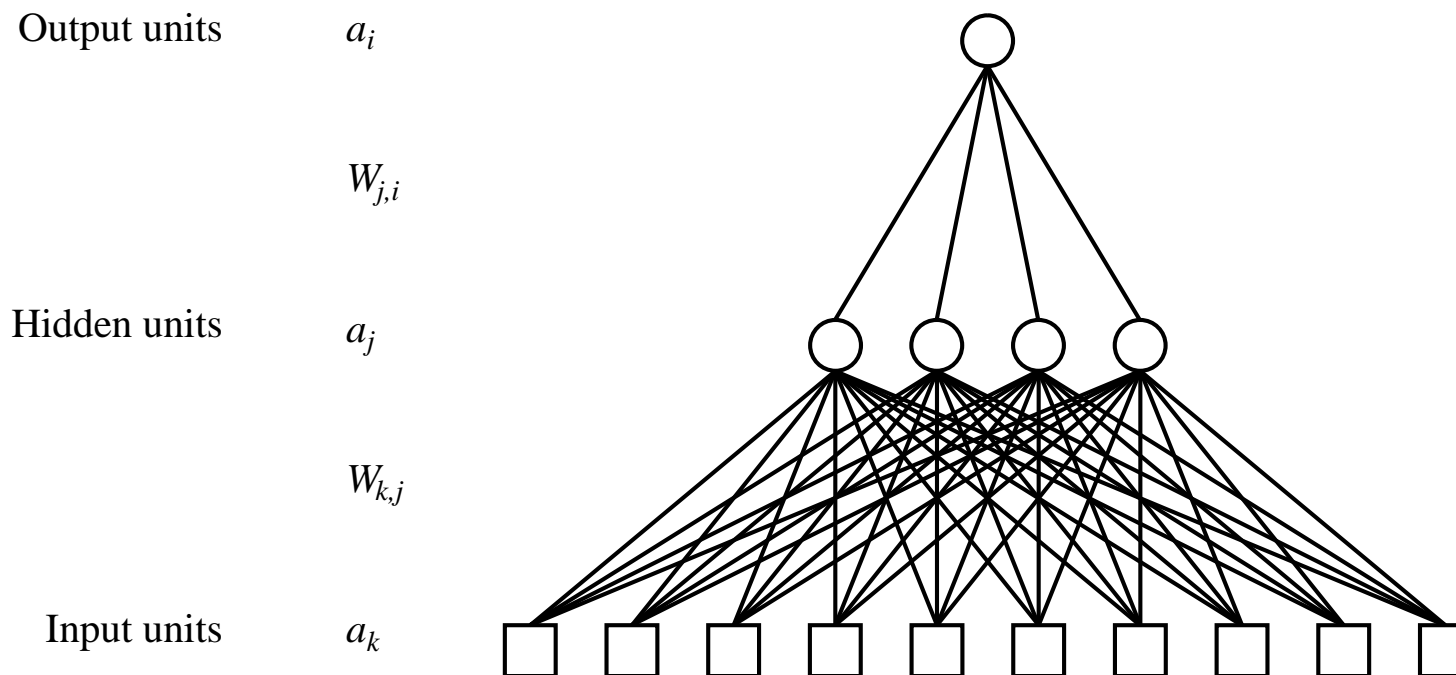


(c) x_1 **xor** x_2

Minsky & Papert (1969) pricked the neural network balloon

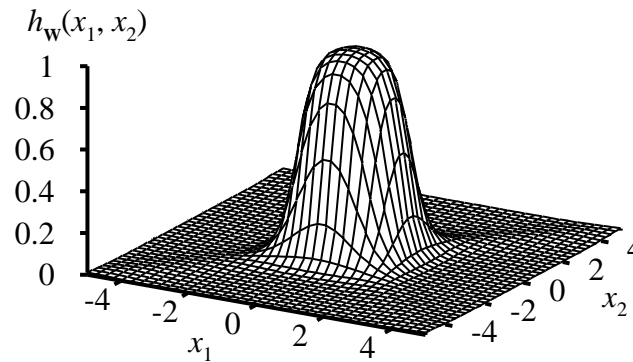
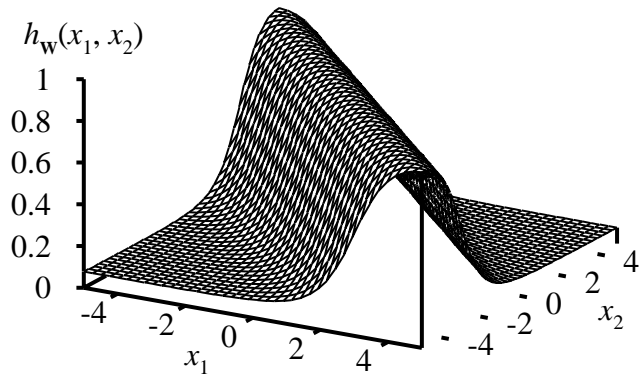
Multilayer perceptrons

Layers are usually fully connected;
numbers of **hidden units** typically chosen by hand



Expressiveness of MLPs

All continuous functions w/ 2 layers, all functions w/ 3 layers



Combine two opposite-facing threshold functions to make a ridge

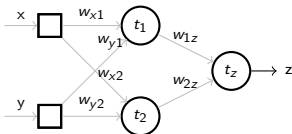
Combine two perpendicular ridges to make a bump

Add bumps of various sizes and locations to fit any surface

Proof requires exponentially many hidden units (cf DTL proof)

A Simple Example

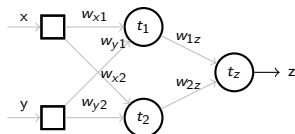
- ▶ Let the following 3-layer feed-forward network be given:



- ▶ Input units:
 - *Input function*: input value set from outside
 - *Output function*: identity
- ▶ Hidden units:
 - *Input function*: weighted sum
 - *Output function*: sigmoidal
- ▶ Output units:
 - *Input function*: weighted sum
 - *Output function*: sigmoidal

A Simple Example (cont.)

- ▶ Let the following 3-layer feed-forward network be given:



- ▶ Input units:

- ▶ Hidden units:
$$z = \text{sig}(w_{1z} * \text{sig}(w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig}(w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$

- ▶ Output units:
$$\text{with } \text{sig}(x) = \frac{1}{1.0 + e^{-x}}$$

Desired vz. Actual Input-Output Behaviour

- ▶ Actual input-output-behaviour as just derived:

$$z = \text{sig}(w_{1z} * \text{sig}(w_{x1} * o_x + w_{y1} * o_y + t_1) + w_{2z} * \text{sig}(w_{x2} * o_x + w_{y2} * o_y + t_2) + t_z)$$
$$= \frac{1}{1.0 + e^{-\left(w_{1z} * \frac{1}{1.0 + e^{-(w_{x1} * x + w_{y1} * y + t_1)}} + w_{2z} * \frac{1}{1.0 + e^{-(w_{x2} * x + w_{y2} * y + t_2)}} + t_z\right)}}$$

- ▶ What is the desired output of the network?
- ▶ Usually it is “specified” in form of training samples D :

x	0.0	0.1	0.0	1.0
y	0.0	0.0	1.0	1.0
z	0.0	1.0	1.0	1.0

$$D := \{((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 1)\}$$

- ▶ Goal of the training: modify the parameters of the network function, i.e.: weights w_{ij} and bias values t_i
- ▶ Please note: the hyper-parameter (e.g., structure and functions) are not adjusted as part of the training

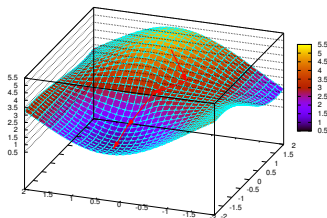
Training a Neural Network

Result: A trained network

Input: A Network N , a set of training data D

- 1 Let π_N be the set of parameters of N :
weights and thresholds
- 2 Initialise all parameters π_N , randomly
- 3 **repeat**
- 4 Compute error E wrt. D and current parameters π_N
- 5 Modify parameters π_N such that the error decreases
- 6 **until** E is acceptable
- 7 **return** *The modified network N*

- ▶ Let a set of samples $\{(x_1, y_1), \dots, (x_n, y_n)\}$ be given.
- ▶ Error of the network defined by a loss function, e.g., quadratic loss $E = \sum_i (\mathcal{N}(x_i) - y_i)^2$.
(with $\mathcal{N}(x_i)$ being the output of the network for the input x_i)
- ▶ Idea: minimise E by gradient descent.



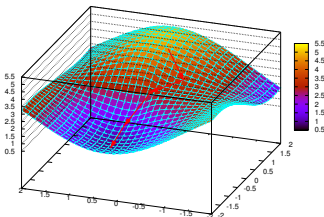
Gradient descent

“Gradient descent is an [...] iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum [...] we take steps proportional to the negative of the gradient [...] of the function at the current point. [...] Gradient descent was originally proposed by Cauchy in 1847.”

 *Gradient descent*

Gradient Descent

- ▶ *Gradient descent* is an optimisation algorithm to find a local minimum of a given function
- ▶ Idea:
 1. Select a starting position
 2. Compute the gradient of the function at the current point
 3. Make a step towards the steepest descent (down hill)
 4. Repeat step 2 and 3 until satisfied



Gradient Descent (GD)

Finding a Local Minimum by Gradient Descent

Input: Differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Input: Starting point $\vec{x} \in \mathbb{R}^n$, Step size $\gamma \in \mathbb{R}^+$

Result: A local minimum

1 **repeat**

2 Compute $\nabla f(\vec{x})$ with $\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$

3 Set $\vec{x} := \vec{x} - \gamma \nabla f(\vec{x})$

4 **until** $\nabla f(\vec{x}) = \vec{0}$ or given number of iterations

5 **return** \vec{x} , which is a minimum iff $\nabla f(\vec{x}) = \vec{0}$

- ▶ for certain function classes (e.g., convex and Lipschitz continuous) and suitable γ , GD converges to a local minimum
- ▶ the step size γ can be different for every iteration
- ▶ if f is convex, every local minimum is a global minimum

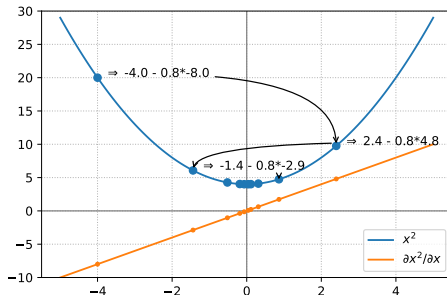
Rules for Partial Derivatives

do you
recall?

Rule	F	$\partial F / \partial x$
Constant	c	0
Factor	$c \cdot f(x)$	$c \frac{\partial f(x)}{\partial x}$
Power rule	x^n	$n \cdot x^{n-1}$
Sum rule	$f(x) + g(x)$	$\frac{\partial f(x)}{\partial x} + \frac{\partial g(x)}{\partial x}$
Product rule	$f(x) \cdot g(x)$	$f \frac{\partial g(x)}{\partial x} + g \frac{\partial f(x)}{\partial x}$
Chain rule	$f(g(x))$	$\frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$
Exponential	e^x	e^x

Gradient Descent by Example

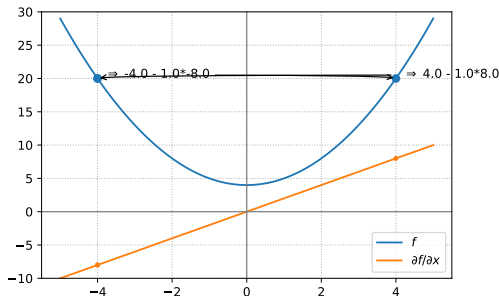
- ▶ How to find the minimum of a function?
- ▶ Let the following function be given:



- ▶ Lets find the minimum by going downhill
 - For this we compute the derivative
 - Starting at some point (e.g., $x_0 = -4.0$), we take the derivative ($\frac{\partial f}{\partial x}(x_0) = -8.0$ and subtract e.g., $\gamma = 0.8$ times the derivative, i.e., $x_{i+1} = x_i - \gamma \frac{\partial f}{\partial x}(x_i)$

Gradient Descent by Example - ping pong

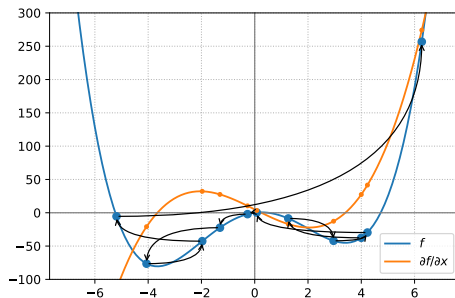
- ▶ But there might be combinations of function and step size for which this fails:



- ▶ There are strategies, e.g., stochastic gradient descent which solve this and similar problems (discussed later)

Gradient Descent by Example - exploding gradients

- ▶ But there are very steep functions with large derivatives:



- ▶ In certain situations, the gradients “explode”
- ▶ There are strategies, e.g., gradient clipping, which solve this and similar problems (discussed later)

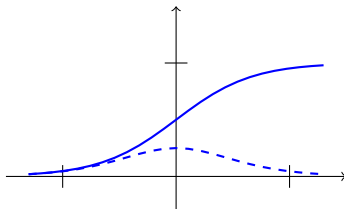
The Sigmoidal function and it's derivative

- First we need to compute the derivative of the sigmoidal

$$\text{sig}(x) = \frac{1}{1.0 + e^{-x}}$$
$$\text{sig}'(x) = \frac{\partial \text{sig}(x)}{\partial x} = \text{sig}(x) \cdot (1 - \text{sig}(x))$$

T Compute the derivative of the sigmoidal function by hand!

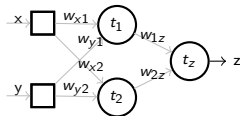
- Shape of the sigmoidal and it's derivative (dashed line):



- A very nice step-by-step explanation can e.g. be found here:

towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

Partial derivative wrt. t_z



- Network function as computed above

$$\mathcal{N}_{xy} = \mathcal{N}(x, y) = \text{sig}(i_z(x, y) + t_z)$$

- An error (quadratic loss) function based on a given sample:

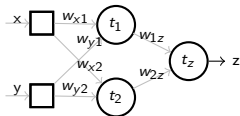
$$E(x, y, z) = (\mathcal{N}(x, y) - z)^2$$

- Compute the derivative of E wrt. t_z

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_z} &= \frac{\partial (\mathcal{N}_{xy} - z)^2}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_z} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_z} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial i_z(x, y) + t_z}{\partial t_z} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy})\end{aligned}$$

Partial derivative wrt. w_{1z}

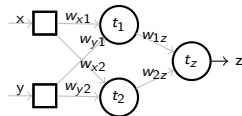


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{1z}} &= \frac{\partial (\mathcal{N}_{xy} - z)^2}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial w_{1z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{1z}} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_1(x, y)\end{aligned}$$

Partial derivative wrt. w_{2z}

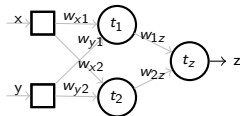


$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial w_{2z}} &= \frac{\partial (\mathcal{N}_{xy} - z)^2}{\partial w_{2z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial w_{2z}} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_2(x, y)\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial w_{2z}} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * o_1(x, y) + w_{2z} * o_2(x, y) + t_z}{\partial w_{2z}} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * o_2(x, y)\end{aligned}$$

Partial derivative wrt. t_1



$$\begin{aligned}\mathcal{N}_{xy} = \mathcal{N}(x, y) &= \text{sig}(i_z(x, y) + t_z) \\ &= \text{sig}(w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z)\end{aligned}$$

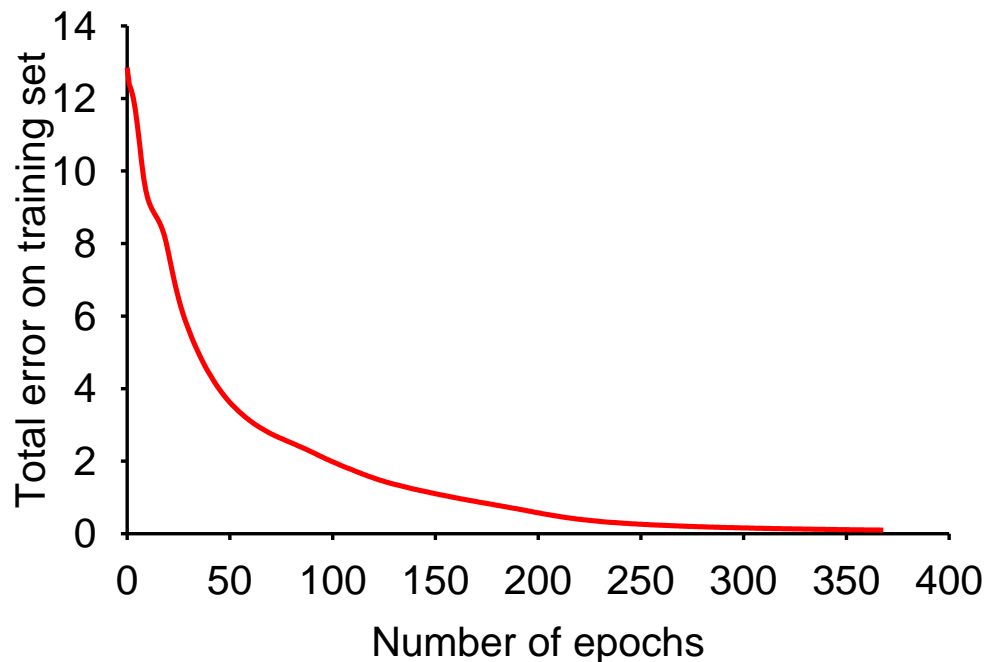
$$\begin{aligned}\frac{\partial E(x, y, z)}{\partial t_1} &= \frac{\partial (\mathcal{N}_{xy} - z)^2}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \frac{\partial \mathcal{N}_{xy}}{\partial t_1} \\ &= 2 * (\mathcal{N}_{xy} - z) * \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y))\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{N}_{xy}}{\partial t_1} &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * \frac{\partial w_{1z} * \text{sig}(i_1(x, y) + t_1) + w_{2z} * o_2(x, y) + t_z}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * \frac{\partial \text{sig}(i_1(x, y) + t_1)}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y)) * \frac{\partial i_1(x, y) + t_1}{\partial t_1} \\ &= \mathcal{N}_{xy} * (1 - \mathcal{N}_{xy}) * w_{1z} * o_1(x, y) * (1 - o_1(x, y)) * 1\end{aligned}$$

Back-propagation learning contd.

At each **epoch**, sum gradient updates for all examples and apply

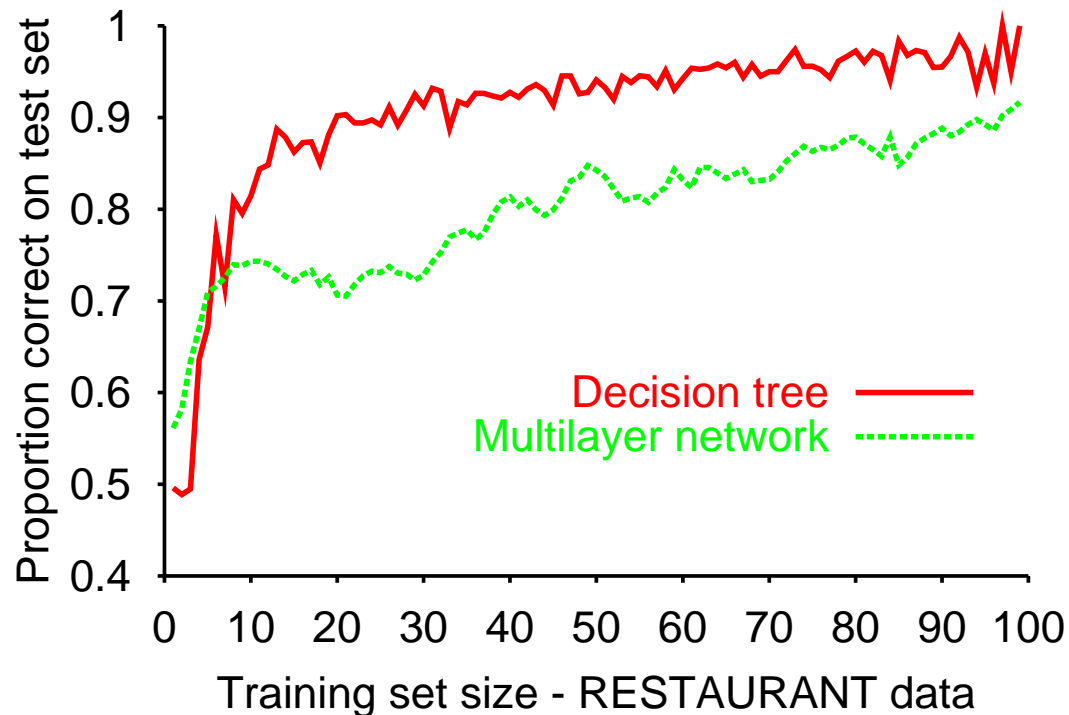
Training curve for 100 restaurant examples: finds exact fit



Typical problems: slow convergence, local minima

Back-propagation learning contd.

Learning curve for MLP with 4 hidden units:



MLPs are quite good for complex pattern recognition tasks,
but resulting hypotheses cannot be understood easily

Künstliche Neuronale Netze

- Adaption für Klassifikation leicht möglich, z.B. One-Hot-Encoding
- Wesentliche Durchbrüche in Machine-Learning-Anwendungen sind auf neuronale Netze zurückzuführen, insbesondere für Computer Vision
- Vielzahl Weiterentwicklungen und Varianten, z.B. Recurrent Neural Networks, Convolutional Neural Networks, Autoencoders, Generative Modelle, ...
- Aber: Im Vergleich zum menschlichen Lernen brauchen künstliche neuronale Netze typischerweise sehr große Datenmengen

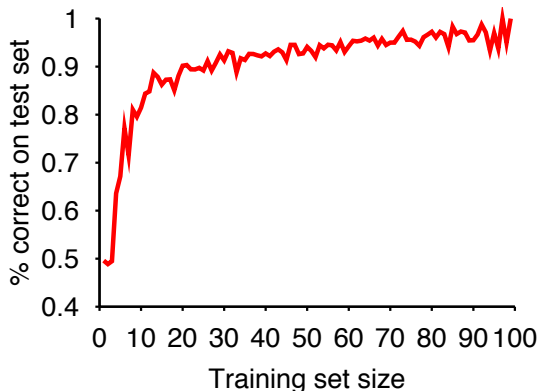
Evaluation

Evaluation

Woher wissen wir, dass $\hat{f} \approx f$?

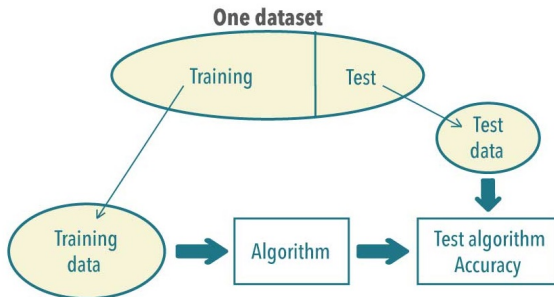
- 1) Verwende Konzepte aus der Statistik, z.B. Likelihood der Daten, R^2 , ...
- 2) Wende \hat{f} auf neuen Daten an (Testdaten)

Lernkurve = % Korrekt klassifizierter Daten in Abhängigkeit von Menge der Trainingsdaten



Trainings- vs. Testdaten

Training data vs. test data



- Lernen: Bestimme Struktur/Parameter des Modells mit Trainingsdaten
- Evaluere Modellgüte auf Testdaten
- Trainingsdaten \neq Testdaten

Performance-Messung

Lernkurve hängt ab von

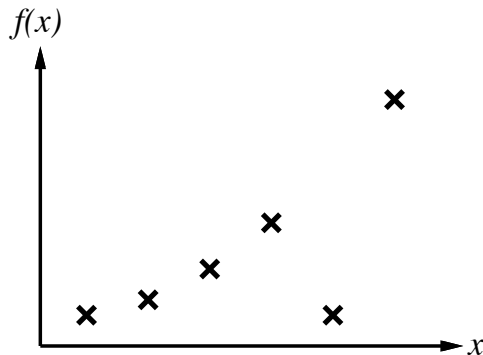
- Realisierbarkeit der tatsächlichen Funktion im Modell (Ausdrucksstärke), z.B. quadratische Abhängigkeit in linearer Regression?
- Redundanz, Rauschen in den Attributwerten – schwierig, relevante Zusammenhänge zu extrahieren

Overfitting

- Redundanz, Rauschen in den Daten + ausdrucksstarke Modelle können zu *Überanpassung* (Overfitting) führen
- D.h. Fehler auf den Trainingsdaten nimmt immer weiter ab, aber nimmt auf Testdaten wieder zu
- Problem: Modell berücksichtigt auch zufälliges Rauschen für Vorhersage, “Daten auswendig gelernt”

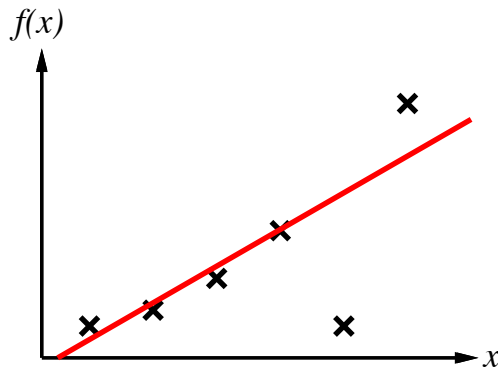
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



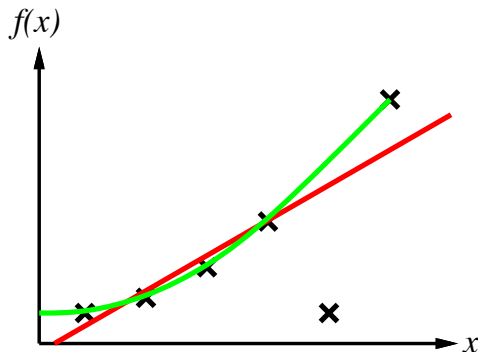
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



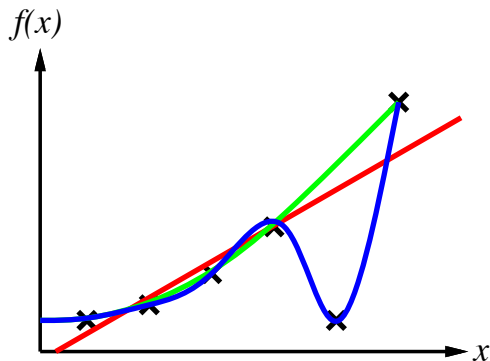
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



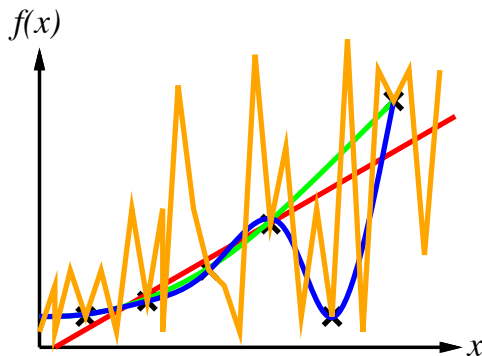
Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?



Beispiel Overfitting

Welche Funktion \hat{f} repräsentiert die Daten auf angemessene Weise?
Welche Funktion wird auch für neue Daten (Testdaten) angemessen sein?







Evaluation von Klassifikations-Algorithmen

- Im Folgenden schauen wir uns die Evaluation von *Klassifikations-Algorithmen* noch etwas genauer an
 - Für Regressions-Algorithmen berechnet man z.B. Wurzel aus mittlerem quadartischen Fehler als Gütekriterium, darauf gehen wir aber hier nicht weiter ein
- Abgesehen von dem Anteil der korrekt klassifizierten Test-Samples (*Accuracy*, *Genauigkeit*) gibt es noch weitere wichtige Performance-Maße:
 - Confusion Matrix (Wahrheitsmatrix)
 - Precision und Recall
 - Sensitivität und Spezifität
 - F1-Score

Confusion Matrix

		Predicted										
		0	1	2	3	4	5	6	7	8	9	
Actual	0	4	0	0	0	0	0	0	0	0	0	
	1	15	28	4	0	5	0	0	0	0	0	
	2	0	0	0	0	48	48	0	0	0	0	
	3	0	0	0	21	0	43	13	0	0	0	
	4	0	0	0	0	704	0	0	0	0	0	
	5	0	0	0	0	0	105	0	0	0	0	
	6	0	0	0	0	0	0	467	0	0	0	
	7	0	0	0	0	0	0	62	300	0	0	
	8	0	0	0	0	0	0	0	0	4	0	
	9	0	0	0	0	2	2	19	59	4	22	




-  True positive $\text{cfm}[i,i]$
-  False positive $\text{cfm}[-i,i]$
-  False negative $\text{cfm}[i,-i]$
-  True negative $\text{cfm}[-i,-i]$

- Trage für alle Testdaten wahre Klasse und vorhergesagte Klasse gegeneinander auf

Accuracy

■ Accuracy: Anteil korrekt klassifizierter Testdaten

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22





-  True positive $cfm[i,i]$
-  False positive $cfm[-i,i]$
-  False negative $cfm[i,-i]$
-  True negative $cfm[-i,-i]$

$$Accuracy = \frac{\sum tp + \sum tn}{\sum tp + \sum tn + \sum fp + \sum fn}$$

Precision

- Anteil der tatsächlich positiven Samples unter allen als positiv erkannten Samples

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22

	True positive	$cfm[i,i]$
	False positive	$cfm[-i,i]$
	False negative	$cfm[i,-i]$
	True negative	$cfm[-i,-i]$

$$Precision = \frac{\sum tp}{\sum tp + \sum fp}$$

Recall (= Sensitivität)

- Anteil der als positiv erkannten Samples aus allen tatsächlich positiven Samples

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22





	True positive	$cfm[i,i]$
	False positive	$cfm[-i,i]$
	False negative	$cfm[i,-i]$
	True negative	$cfm[-i,-i]$

$$Recall = \frac{\sum tp}{\sum tp + \sum fn}$$

Spezifität

Anteil der als negativ erkannten Samples aus allen negativen Samples

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	70	4	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22

-  True positive $cfm[i,i]$
-  False positive $cfm[-i,i]$
-  False negative $cfm[i,-i]$
-  True negative $cfm[-i,-i]$

$$Specificity = \frac{\sum tn}{\sum tn + \sum fp}$$

F1-Score

■ Harmonisches Mittel aus Precision und Recall

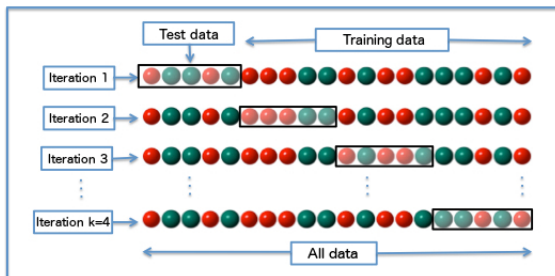
		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	4	0	0	0	0	0	0	0	0	0
	1	15	28	4	0	5	0	0	0	0	0
	2	0	0	0	0	48	48	0	0	0	0
	3	0	0	0	21	0	43	13	0	0	0
	4	0	0	0	0	704	0	0	0	0	0
	5	0	0	0	0	0	105	0	0	0	0
	6	0	0	0	0	0	0	467	0	0	0
	7	0	0	0	0	0	0	62	300	0	0
	8	0	0	0	0	0	0	0	0	4	0
	9	0	0	0	0	2	2	19	59	4	22

- True positive $\text{cfm}[i,i]$
- False positive $\text{cfm}[-i,i]$
- False negative $\text{cfm}[i,-i]$
- True negative $\text{cfm}[-i,-i]$

$$F_1 = \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}$$

Kreuzvalidierung

- Problem bei Aufteilung in Trainings- und Testdaten:
 - Kann nicht alle Daten zum Trainieren verwenden
 - Ergebnis hängt von gewählten Testdaten ab
- Kreuzvalidierung umgeht diese Probleme, indem systematisch verschiedene Train-/Test-Splits evaluiert werden
 - Teile die Daten in N Teilmengen X_1, \dots, X_N auf
 - Für $i \in \{1, \dots, N\}$
 - Trainiere Modell mit $X \setminus X_i$
 - Evaluere Modell mit X_i
 - Berechne Mittelwert der Güte der N Durchläufe



Zusammenfassung

- Lernen ist in wichtig in nicht vollständig bekannten Umgebungen
- Unüberwachtes Lernen versucht, Muster oder Zusammenhänge in Daten zu erkennen
- Überwachtes Lernen versucht, eine Funktion $f(x) = y$ aus Eingabe-/Ausgabe-Paaren (x, y) (Trainingsdaten) zu rekonstruieren
- Ein Entscheidungsbaum ist ein einfacher, interpretierbarer Klassifikationsalgorithmus
- Künstliche Neuronale Netze sind sehr ausdrucksstarke Regressions- und Klassifikationsalgorithmen, die in vielen Fällen eine hohe Güte erreichen
- Die Güte eines gelernten Modells wird auf separaten Testdaten evaluiert