Was bisher geschah

- Maschinelles Lernen
  - überwacht
    - korrigierend
    - bestärkend
  - unüberwacht
- Künstliche Neuronale Netze
  - biologisches Vorbild neuronaler Netze und Lernvorgänge darin
  - künstliche Neuronen
    z.B. McCulloch-Pitts-Neuron, Schwellwertneuron
  - Eingangs-, Aktivierungs-, Ausgangsfunktion
  - Lernen künstlicher Neuronen (Δ-Regel, überwacht)
  - Feed-Forward-Netze
    gerichteter Graph mit Kantengewichten (Matrix)
    (parallele und sequentielle Berechnung)
  - Verwendung künstlicher neuronaler Netze:

WH: Feed-Forward-Netze

Topologie: gerichteter Graph,
Knoten (Neuronen) in einer oder mehrere Schichten $\{0, \ldots, k\}$,
Kanten je nur von Knoten in Schicht i zu Knoten in Schicht $i + 1$ (mitunter auch $j > i$)
Eingabe: Eingänge der Neuronen in Schicht 0
Ausgabe: Ausgänge der Neuronen in Schicht k

Aktivierung: lineare, Stufen-, sigmoide Funktionen

Lernen:
- Ein-Schicht-FFN: Δ-Regel
- Mehr-Schicht-FFN: Backpropagation (Gradientenabstieg der Fehlerfunktion)

Prominente Aktivierungsfunktionen

- Stufenfunktion $A(X) = (x \geq 0)$,
- sigmoide Funktion $A(x) = \frac{1}{1+e^{-x}}$, $A(x) = \tanh x$
  (differenzierbare Approximation der Stufenfunktion)
- lineare Funktion $A(x) = ax + b$
- ReLU $A(X) = \max(0, x)$,
- analytische Funktion $A(X) = \log(1 + e^x)$,
  (differenzierbare ReLU-Approximation, softplus)
- Radiale Basisfunktion $A(x) = r(d(x, w))$ mit
  Eingabe(-vektor) x, Gewicht-(svektor) w (Zentrum), Abstand d,
  radiale Funktion $r : \mathbb{R}_{\geq 0} \to [0, 1]$ mit
  - $r(0) = 1$
  - monoton fallend: $\forall x, y \in \mathbb{R}_{\geq 0} : (x < y) \to (r(x) < r(y))$

  z.B. d Manhattan-Metrik, $r(x) = \max(0, 1 - x)$
- Softmax (hängt von mehreren Eingaben $(x_1, \ldots, x_n)$ ab
  $A(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$
  zur Klassifizierung in mehrere Klassen
  (Wahrscheinlichkeitsverteilung)

Hybride Netze

Idee: Kombination verschiedenartiger Neuronen

Beispiel RBF-Netz (ca. 1989):
schichtweise verschiedene Aktivierungsfunktionen

Topologie: 2-Schicht-FFN
Eingabeschicht 0
RBF-Schicht 1 aus RBF-Neuronen
Ausgabeschicht 2 aus Schwellwertneuronen

Aktivierung abhängig von Schicht

1. RBF-Neuronen: radiale Funktion
2. Ausgabeneuronen: Stufen-, sigmoide Funktion

Lernen:

- RBF-Schicht: Clustering-Verfahren
- Schwellwert-Schicht: Δ-Regel

RBF-Netze: Beispiel

2-2-1-Netz für $\leftrightarrow$:

Idee: $x_1 \leftrightarrow x_2 \equiv (x_1 \wedge x_2) \vee \neg(x_1 \vee x_2)$

- erste Schicht (RBF): $w_{1,h1} = w_{2,h1} = 1$, $w_{1,h2} = w_{2,h2} = 0$,
  Eingabefunktion: Euklidische Metrik
  $d((x_1, x_2), (w_1, w_2)) = \sqrt{(w_1 - x_1)^2 + (w_2 - x_2)^2}$
  radiale Funktion $r_{h1}(x) = r_{h2}(x) = \max(0, x/2))$
  Aktivierung: Stufenfunktion
- zweite Schicht: Gewichte $w_{h1,y} = w_{h2,y} = 1$,
  Eingabefunktion: gewichtete Summe
  Aktivierung: linear
  Schwellwert $\theta_y = 0$

RBF-Netze zur Approximation von Funktionen $f : \mathbb{R} \to \mathbb{R}$ durch Linearkombination (gewichtete Summe) von radialen Funktionen, z.B.

- stückweise konstante Funktionen (Stufen)
- stückweise lineare Funktionen
- Gauß-Funktionen

Rekurrente Netze

Idee: direkte Abhängigkeit von Eingaben in vorangegangenen Schichten

Topologie:	Graphen mit Rückwärtskanten, ggf. Kontextneuronen zum Speichern von Zwischenwerten
Aktivierung	wie bisher, abhängig vom Neuronentyp
Lernen:	z.B. bei FFN mit Rückwärtskanten: Vorwärtskanten in der „Entwirrung“ (mehrere verbundene Kopien, für jeden Zeitschritt eine) des Netzes trainieren (Backpropagation through time) Gewichte der Rückwärtskanten meist fix
Besonderheit:	Zeitschritte definieren Zustände des Netzes Zustand: Zuordnung Neuron $\rightarrow \mathbb{R}$ (Aktivierung)

Rekurrente Netze – Beispiel

- zwei McCulloch-Pitts-Neuronen u, v
- Eingang $x \in \{0, 1\}$
- Ausgang $y \in \{0, 1\}$
- erregende Kanten: $(x, u), (x, v), (u, u), (u, v), (v, y)$
- hemmende Kanten $(v, v), (v, u)$ (Eingabe 1 verhindert Aktivierung)
- Schwellwerte $\theta_u = 1$, $\theta_v = 2$

Satz: Zu jedem NFA existiert ein rekurrentes Netz mit McCulloch-Pitts-Neuronen, welches dieselben Zustandsübergänge simuliert.

Assoziativspeicher

Ziel: Musterassoziation
(Training mit endlich vielen Musterpaaren)

Generalisierung:

- Aus Zuordnung: Muster $x \rightarrow$ Muster y
  folgt für jedes zu x ähnliche Muster x' die
  Zuordnung: Muster $x' \rightarrow$ Muster y.
- Ziel: sinnvolle Zuordnung „verrauschter“ oder unvollständiger Eingabemuster

Netztypen:

heteroassoziativ Eingabemuster $x \in \mathbb{R}^m$,
Ausgabemuster $y \in \mathbb{R}^n$

Mustererkennung Spezialfall heteroassoziativer Netze
assoziiert Muster mit Identifikator, z.B. für Klasse

autoassoziativ Spezialfall heteroassoziativer Netze
Ein- und Ausgabemuster $x \in \mathbb{R}^m$ (prinizipiell) gleich

Heteroassoziativer Speicher

Topologie: vollständig verbundenes Ein-Schicht-FFN,
Eingänge $x \in \mathbb{R}^m$, Ausgänge $y \in \mathbb{R}^n$,
alles Schwellwertneuronen, Gewichtsmatrix $W \in R^{m \times n}$

Aktivierung: Signum = Stufenfunktion

$$A(x) = \operatorname{sgn}(x) \begin{cases} -1 & \text{falls } x < 0 \\ 0 & \text{falls } x = 0 \\ 1 & \text{sonst} \end{cases}$$

Berechnung der Gewichte: Methode der kleinsten Quadrate =
Minimierung von

Berechnung der Eingangsfunktion analog Ein-Schicht-FFN:

$$I(x_1, \ldots, x_m) = (x_1, \ldots, x_m) \begin{pmatrix} w_{11} & \ldots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \ldots & w_{mn} \end{pmatrix}$$

dasselbe kürzer: $I(x) = xW$

Heteroassoziativer Speicher: Beispiel

Berechnung für 3-2-Netz mit Gewichtsmatrix W:

$$W = \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Eingabe $x = (1, -1, 1)$

Berechnung:

$$\text{sgn}(xW) = \text{sgn}\left((1, -1, 1)\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\right) = \text{sgn}(-3, 3) \mapsto (-1, 1)$$

Ausgabe $y = (-1, 1)$

Heteroassoziativer Speicher: Training

Idee: Lernen aus gleichzeitiger Aktivität (Hebb)

biologisches Vorbild: Synapsen zwischen gleichzeitig aktiven Neuronen (x_i und y) werden verstärkt (synaptische Plastizität)

Trainingsmenge $\{(x^{(i)}, y^{(i)}) \mid i \in \{1, \ldots, k\}\}$ (Bipolarvektoren)

Ziel Gewichtsmatrix W, so dass für jedes Trainingspaar $(x^{(i)}, y^{(i)})$ gilt: mit $\text{sgn}(x^{(i)} W) = y^{(i)}$ (komponentenweise)

Startgewichte alle $w_{ij} = 0$

Lernregel von Hebb $\Delta w_{ij} = \eta x_i y_j$, hier mit Lernrate $\eta = 1$

Training : Gewichtsbestimmung $\Delta w_{kl} = x_k y_l$
je Trainigspaar (x, y) einmal
(W ist Korrelationsmatrix von x und y)

Alternative zum Training: direkte Berechnung der Gewichte (z.B. mit Methode der kleinsten Quadrate)

Bidirektionaler Assoziativspeicher (BAM)

(heteroassoziativer Speicher von Musterpaaren)

Netz-Topologie:

- Eingangsschicht, Ausgangsschicht, keine versteckten Neuronen
- Eingänge $x \in \{-1, 1\}^m$
- Ausgänge $y \in \{-1, 1\}^n$
- vollständige symmetrische Verbindungen zwischen jedem Eingangs- und jedem Ausgangsneuron
  (ungerichteter vollständiger bipartiter Graph $K_{m,n}$)
- Gewichte an jeder Kante (für beide Richtungen gleich)
  Gewichte in Gewichtsmatrix $W \in R^{m \times n}$
- Eingangsfunktion: gewichtete Summe
- Aktivierung: Signum
- Ausgangsfunktion: Identität

BAM: Berechnung und Training

(wie im heteroassoziativen Speicher)

überwachtes Lernen

Trainingsmenge $\{(x^{(i)}, y^{(i)}) \mid i \in \{1, \ldots, k\}\}$

Eingabe : Startzustand (initiale Zustände der Eingangsneuronen $x \in \{-1, 1\}^m$)

Lernregel von Hebb: $\Delta w_{ij} = \eta x_i y_j$ mit $\eta = 1$

Berechnung : Folge von Schritten (Zustandsübergängen),

- synchron oder
- abwechselnd

für beide Neuronenschichten:
Aktualisierung: Neuberechnung der Aktivierung der anderen Schicht
wiederholen, bis stabiler Zustand erreicht (Fixpunkt) oder Abbruch ausgelöst wird

Ausgabe : Zustand der Ausgangsneuronen des stabilen Netzes

BAM: Beispiel

ein Trainingspaar (x, y) mit
$x = (1, -1, 1)$ und $y = (-1, 1)$

Gewichtsmatrix W:

$$W = \begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Berechnung:

$$\text{sgn}(xW) = \text{sgn}\left((1, -1, 1)\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\right) = \text{sgn}(-3, 3) \mapsto (-1, 1) = y$$

$$\text{sgn}(Wy^T) = \text{sgn}\left(\begin{pmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix}\right) = \text{sgn}\begin{pmatrix} 2 \\ -2 \\ 2 \end{pmatrix} \mapsto \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = x^T$$

BAM: Training

(wie heteroassoziativer Speicher)

- für ein zu speicherndes Musterpaar $x \in \{-1,1\}^m, y \in \{-1,1\}^n$: $W = x^T y$ (Korrelationsmatrix von x und y)
- für mehrere zu speichernde Musterpaare $(x^{(1)}, y^{(1)}), \ldots, (x^{(k)}, y^{(k)})$:

$$W = \sum_{i=1}^{k} W^{(i)} = \sum_{i=1}^{k} \left(x^{(i)}\right)^T y^{(i)}$$

Für alle k Trainingsmuster gleichzeitig:

- Eingabemuster $X \in \{-1,1\}^{k \times m}$
- Ausgabemuster $Y \in \{-1,1\}^{k \times n}$
- $XX^T \in \{-1,1\}^{k \times k}$ (alle Einträge positiv)
- $\text{sgn}(Y) = \text{sgn}(XX^T Y) \in \{-1,1\}^{k \times n}$
- $\text{sgn}(Y) = \text{sgn}(XW) \in \{-1,1\}^{k \times n}$ mit $W = X^T Y$

Hopfield-Netz

(autoassoziativer Musterspeicher)

Idee: Ein-und Ausgabeknoten in autoassoziativem BAM identifiziert

Topologie: K_n mit symmetrischer Gewichtsmatrix $W \in \mathbb{R}^n$
Jedes Neuron ist zugleich Ein- und Ausgang $x \in \{-1, 1\}^n$
keine Selbstrückkopplung, also $\forall i \in \{1, \ldots, n\} : w_{ii} = 0$

Zustand: Aktivierung aller Neuronen

Eingabe: Startzustand (initiale Zustände aller Neuronen)

Berechnung: Folge von Schritten (Zustandsübergängen):
Aktualisierung: Berechnung der Aktivierung aller Neuronen
wiederholen, bis stabiler Zustand erreicht oder Abbruch

Konvergenz : Erreichen eines stabilen Zustandes (ändert sich bei Aktualisierung beliebiger Neuronen nicht)

Ausgabe : Zustand des stabilen Netzes

Aktualisierung in jedem Schritt:

synchron: gleichzeitige Zustandsänderung für alle Neuronen

asynchron: Zustandsänderung eines zufällige gewählten Neurons
(faire Auswahl)

Hopfield-Netz – Beispiele

$$W = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad x = (1, -1, 1)$$

$$W = \begin{pmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 \end{pmatrix} \quad x = (-1, -1, 1, 1)$$

$$W = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad x = (1, 1, 1)$$

$w_{ii} \neq 0 \rightarrow$ Oszillation

Hopfield-Netz – Training

direkte Berechnung der Gewichte möglich,
kein Training notwendig

- für ein zu speicherndes Muster $x \in \{-1, 1\}^m$:

$$W = x^T x$$

  mit Modifikation: alle Diagonalelemente 0

- für mehrere zu speichernde Muster
  $x^{(1)} \in \{-1, 1\}^m, \ldots, x^{(k)} \in \{-1, 1\}^m$:

$$W = \sum_{i=1}^{n} \left(x^{(i)}\right)^T x^{(i)}$$

  mit Modifikation: alle Diagonalelemente 0

Beispiel (Tafel):

- ein Muster: $x = (1, -1, 1, 1)$
- mehrere Muster: $x^{(1)} = (-1, 1, -1)$ und $x^{(2)} = (1, -1, 1)$

Unüberwachtes Lernen

bei unbekannter Zielfunktion

zur Analyse von Datenmengen

Ziel: Gruppierung ähnlicher Daten (Clustering)
z. B. erste Schicht in RBF-Netzen

Generalisierung durch Einordnung neuer Daten in vorhandene Gruppen (Cluster)

Training durch Menge von Trainingmustern
$T = \{x^i \mid i \in \{1, \ldots, k\} \wedge x^i \in \mathbb{R}^m\}$

Methode: Wettbewerbslernen (competitive learning)
nur „Gewinner“-Neuron feuert

Bündeln von Mustern (Clustering)

Eingabe: Menge von Trainingsmustern $\{x^{(i)}, \ldots, x^{(m)}\}$

Ziel: Gruppierung ähnlicher Muster

Anordnung von Mustern in Bündeln:

- Ähnlichkeit aller Muster eines Clusters
- Trennung von Mustern mit wenig Gemeinsamkeiten

Ähnlichkeit im $\mathbb{R}^n$

Zwei Punkte $x, y \in \mathbb{R}^n$ sind einander ähnlich, falls sie einen geringen Abstand voneinander haben

Beispiele für Abstandsfunktionen: siehe RBF-Netze

statt Euklidischem Abstand

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

einfachere Berechnung
quadrierter Euklidischer Abstand

$$d(x,y) = \sum_{i=1}^{n}(x_i - y_i)^2$$

Zuordnung von Mustern zu Bündeln

Eingaben:

- Menge $\{1, \ldots, k\}$ von Bündeln,
- Muster x

x wird dem Bündel $j \in \{1, \ldots, k\}$ zugeordnet, von welchem es den geringsten Abstand hat

Idee: Jedes Bündel $j \in \{1, \ldots, k\}$ hat ein Zentrum p_j
(durchschnittliche Position aller Punkte im Bündel, Prototyp)

neues Muster x wird dem Bündel j genau dann zugeordnet, wenn der Abstand von x zum Zentrum p_j des Bündels j minimal ist, d.h.

$$\text{gdw. } \forall i \in \{1, \ldots, k\} : d(x, p_j) \leq d(x, p_i)$$

geometrisch: Zerlegung des R^n in Gebiete
z.B. im $\mathbb{R}^2$: Voronoi-Diagramme

Selbstorganisierende Karte (SOM)

(Teuvo Kohonen, ca. 1980)

Netz-Topologie: Ein-Schicht-Feed-Forward-Netz mit

- Eingabe $(x_1, \ldots, x_m) \in \mathbb{R}^m$, interpretiert als Punkt im m-dimensionalen Eingaberaum $\mathbb{R}^m$
- Ausgabeneuronen $(p_1, \ldots, p_n)$(Bündelneuronen) interpretiert als n Punkte $p_i = (w_{1i}, \ldots, w_{mi})$ im $\mathbb{R}^m$, Zentren der Bündel
- vollständig verbunden (alle Vorwärtskanten), Eingabegewichte zum Neuron j: $(w_{1j}, \ldots, w_{mj})$ interpretiert als Koordinaten des Zentrums des Bündels j

Selbstorganisierende Karte – Funktionen

Funktionen des Ausgabeneurons j:

- Eingabefunktion: Distanzfunktion (Metrik)
  z. B. quadrierter Euklidischer Abstand:

$$I_j(x_1, \ldots, x_m) = \sum_{i=1}^{m} (w_{ij} - x_i)^2$$

- Aktivierungsfunktion
  (Auswahl des Neurons mit dem kleinsten Abstand von x)

$$A_j(I_j(x)) = \begin{cases} 1 & \text{falls } \forall l \in \{1, \ldots, n\} : I_j(x) \leq I_l(x) \\ 0 & \text{sonst} \end{cases}$$

  Wettbewerbslernen: genau ein Ausgabeneuron feuert

- Ausgabefunktion: Identität

Selbstorganisierende Karte – Lernen

Beginn: zufällige Eingangsgewichte der Ausgabeneuronen
(Anordnung der Bündelzentren)

Trainieren der Gewichte
für jede Eingabe $x = (x_1, \ldots, x_m)$:

- Eingabe x
- Anpassung der Eingangsgewichte des aktivierten Bündelneurons j
  (Verschiebung des Bündelzentrums p_j in Richtung des aktuellen Eingabevektors x)

$$w'_{ij} = w_{ij} + \eta(x_i - w_{ij})$$

Topologie-erhaltende SOM

Idee: Topologie-erhaltende Abbildung in andere (geringere) Dimension

Anordnung der Bündelneuronen (z.B. linear, eben, räumlich)

Blockparty:
Berücksichtigung der Nachbarschaft (räumlichen Beziehungen) zwischen den Bündelneuronen (in der definierten Anordnung) bei der Aktualisierung der Positionen
z.B. Graph, Gitter, Abstandsfunktionen

Training Topologie-erhaltender SOM

- Start mit zufälligen Gewichten (Bündelzentren)
- Anpassung der Eingangsgewichte des aktivierten Bündelneurons p_j (in Richtung des Eingabevektors)
- Anpassung der Eingangsgewichte aller Bündelneuronen p_k in einer Nachbarschaft des aktivierten Bündelneurons p_j

$$w'_{ik} = w_{ik} + n(p_k, p_j)\eta(x_i - w_{ik})$$

mit Nachbarschaftsfunktion $n(p_k, p_j)$
(Einfluss fällt mit wachsendem Abstand auf 0, z.B. RBF)

SOM – Trainingsverlauf

Idee:
Anpassung (Verringerung) von Lernrate und Radius während der Lernphase

Heuristik (sinnvoller Trainingsverlauf):

- Start mit
  - großem Radius (nahe halbem Kartenradius)
  - großer Lernrate (nahe 1)

  unverändert über ca. 1000 Iterationen
  Ordnungsphase
- Nachbarschafts-Radius (Einflussbereich jedes Bündelneurons) und Lernrate
  werden mit der Zeit verringert (über ca. 10 000 Iterationen)
  Feinabstimmung

SOM – Beispiele

Topologie-erhaltende Abbildung zwischen Räumen (evtl. verschiedener Dimensionen)

Beispiele:

- Projektion Kugel $\rightarrow$ Ebene
  (Robinson-Projektion: Erde $\rightarrow$ Weltkarte)
- Sensorische Karten:
  Abbildung von Reizen auf benachbarten Bereichen der Haut auf benachbarte Bereiche im Gehirn
- Abbildung von akustischen Reizen benachbarter Frequenzen auf benachbarte Bereiche im Gehirn

Cognitron

Beobachtungen im visuellen System:

- sendet vorverarbeitete Signale an Gehirn
- heterogenes Netz: verschiedene Neurone haben verschiedene Wirkungen (Funktionen)
- Neuronen derselben Schicht haben dieseslbe Funktion
- Verbindung benachbarter Neuronen
  horizontale Zellen berechnen Mittelwert (der Helligkeit)
  wirken hemmend auf Signale nahe beim Mittelwert
- ähnlich Faltung in digitaler Bildverarbeitung (Tafel):
  Funktionswert eines Pixels hängt von Werten benachbarter Pixel ab

Bild-Pyramiden

Features:

- Flächen gleicher Farbe
- Kanten
- Formen
- Texturen, ...

Idee aus DBV:
Bilder enthalten Informationen auf verschiedenen Ebenen,
kleinteilige Beobachtung lenkt evtl. von wesentlichen Merkmalen ab
Umsetzung durch Multiskalen-Bilder (Pyramiden)
entstehen durch mehrfache Wiederholung von

- Glättung (durch geeignete Filter)
- Komprimierung durch geringere Abtastrate,
  z.B. Gauß-Pyramide: Löschen jeder zweiten Zeile und Spalte

Umsetzung als KNN (feed-forward)

Neocognitron

Fukushima, 1975: Cognitron: A Self-Organizing Multilayered Neural Network Model

1983: Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition

Motivation: Erkennung handschriftlicher Ziffern

Aufbau Neocognitron:

- Eingabe-Schicht
- vier (oder mehr) versteckte Stufen aus je zwei Schichten:
  1. Transformation in 12 Bilder (Ebenen)
     Feature-Extraktion (Faltungen mit je einem 3×3-Kern)
     Filterkerne durch Eingangsgewichte definiert (weight sharing)
     Gewichte durch Trainingsmuster gelernt
  2. Kombination mehrerer transformierter Bilder
     z.B. punktweise gewichtete Summe, Max
     Gewichte nicht trainiert
- Ausgabe nach letzter Kombinations-Schicht (Klassifikation)
- inkrementelles Lernen stufenweise von Ein- zu Ausgabeschicht

mehrere Varianten mit überwachtem und unüberwachtem Lernen

Convolutional Neural Networks

z.B. Alex Krizhevsky, . . . , 2012:
ImageNet Classification with Deep Convolutional Neural Networks
prinzipieller Aufbau:

- Eingabe-Schicht
- Versteckte Stufen aus je mehreren Schichten
  - Faltungs-Schicht (Feature-Maps)
    alle Gewichte gleich
  - evtl. ReLU-Schicht (nichtlinear)
  - gelegentlich Subsampling-Schicht (Pooling)

  mehrfache Wiederholung (deep), evtl. in verschiedenen Reihenfolgen
- evtl. klassische Schichten mit vollständigen Verbindungen zwischen benachbarten Schichten
- Ausgabe-Schicht

inzwischen auch komplexere Konstruktionen, z.B.

- AlexNet (Dropout-Schichten)
- GoogLeNet (Inception)
- ResNet (skip connections)

CNN-Schichten

aktuelle CNNs bestehen im Wesentlichen aus mehrfacher Wiederholung folgender Schichten:

- Faltung (convolutional)
- Auswahl (pooling, meist Durchschnitt oder Max)
- vollständig verbunden (fully connected, FC)
- Normalisierung (batch normalization, BN)
- softmax (Wahrscheinlichkeitsverteilung)

CNNs unterscheiden sich in

- Reihenfolge der Schichten
- Anzahl der Schichten / Wiederholungen
- spezielle topologische Merkmale,
  z.B. Vorwärtskanten, die Schichten überspringen

KNN zur Klassifikation

z.B. von handgeschriebenen Ziffern

- Linearer Klassifikator (keine versteckte Schicht)
- Ein-Schicht-FFN (eine versteckte Schicht)
- Mehr-Schicht-FFN (mehrere versteckte Schichten)
- CNN LeNet-1 (1989)
- CNN LeNet-5
- ...

CNN – interessante Beispiele

- VGG16
  ImageNet-Datenbank (seit 2010):
  - $< 15 \cdot 10^6$ annotierte Bilder
  - $< 10 \cdot 10^3$ Klassen

  ImageNet Large-Scale Visual Recognition Challenge
  (ILSVRC, seit 2010)
  Gewinner 2014: VGG16
- Nachcolorierung
  `https://richzhang.github.io/colorization` (2016)
  Eingabe: Grauwertbild
  Ausgabe: Farbbild
- SegNet (2015)
  Aufgabe: semantische Segmentierung
  (Jeder Bildposition wird Bedeutung zugeordnet)
  Eingabe und Ausgabe: Bild derselben Größe,
  Farben des Ausgabebildes sind Klassen
  Idee: Verknüpfung VGG16 mit „gespiegelter“ Version

CNN-Lernen

Überwachtes Lernen durch Backpropagation
(angepasste Verfahren für jeden Schicht-/ Neuronentyp)

Nutzung vortrainierter Netze zur
Beschleunigung des Trainingsprozesses

Idee:

- Ausgangspunkt: Netz, welches schon auf allgemeine Daten trainiert wurde
  z.B. Klassifikation, Segmentierung
- Training auf spezielle Aufgabe
  z.B. andere bzw. feinere Klassen