

Neuronale Netze

Zusammenfassung SS17

Manuel Lang

1. August 2017

INHALTSVERZEICHNIS

1 Grundlagen	5
1.1 Wieso neuronale Netze?	5
1.2 Mustererkennung	5
1.3 Supervised Learning	6
1.4 Unsupervised Learning	6
1.5 Reinforcement Learning	6
1.6 Features	6
1.7 Normalisierung	6
2 Aktivierungsfunktionen	7
2.1 Step function (Schwellwertfunktion)	7
2.2 Sigmoid	7
2.3 Tangens Hyperbolicus	7
2.4 Softmax	7
2.5 Lineare Funktion	7
2.6 Rectified Linear Unit (ReLU)	8
3 Fehlerfunktionen	8
3.1 Sum-Squared-Error (SSE)	8
3.2 Cross-Entropy (CE)	8
4 Perzeptron	8
5 Gradientenabstieg	9
5.1 Varianten des Gradientenabstiegs	9
5.1.1 Batch	9
5.1.2 Online (SGD)	9
5.1.3 Mini-batch	10
5.1.4 Optimierung	10
5.2 Adaptive Lernraten	11
5.2.1 Adagrad	12
5.2.2 Adadelta	12
5.2.3 RMSprop	12
5.2.4 Adam - ADaptive Moment estimation	12
5.3 Generalisierungsfähigkeit	13
5.4 Regularisierung	13
6 Unüberwachtes Lernen / Autoencoders	14
6.1 Autoencoder	14
6.1.1 Erweiterungen	14
6.1.2 Anwendungen	15
6.2 Bottleneck Features	15

7 RBM	16
7.1 Hopfield Netze	16
7.2 Boltzmann Maschinen	17
7.3 Restricted Boltzmann Maschinen	17
8 Spracherkennung - TDNN/CNN	18
8.1 Komponenten eines Spracherkenners	18
8.2 Grundlagen der Spracherkennung	18
8.3 Hidden Markov Models	18
8.4 Time-Delay Neural Network (TDNN)	19
8.5 2-dimensionale TDNNs	19
8.6 Convolutional Neural Networks (CNN)	19
8.7 Multi-State Time Delay Neural Network (MS-TDNN)	20
8.8 NN-HMM Hybrids	20
9 Vektorquantisierer lernen	21
9.1 Vector Quantization (VQ)	21
9.2 Learning Vector Quantization	21
9.3 LVQ1	21
9.4 LVQ2	22
9.5 LVQ2.1	22
9.6 LVQ3	22
9.7 Initialisierung der Codebook-Vektoren	22
9.8 Anwendung in der Praxis	22
10 Self-Organizing Maps	23
10.1 Selbst-organisiertes Lernen vs statistischen Lernen	23
10.2 Prinzipien der Selbst-Organisation	23
10.3 Grundlagen	23
10.4 Algorithmus	23
10.5 Beispiel	24
11 RNN	25
11.1 Rekurrente neuronale Netze	25
11.2 Backpropagation Through Time (BPTT)	25
11.3 Probleme von RNNs	26
11.4 Long Short Time Memory RNNs	26
12 Reinforcement Learning	26
12.1 Grundlagen	27
12.2 Policy-based RL	27
12.3 Value-based RL	27
12.4 Policy Gradients mit Neuronalen Netzen	27
12.5 Q-Learning	27

12.6 Temporal Difference Learning	27
12.6.1 SARSA	27
12.7 Schwierigkeiten	27
12.8 AlphaGo	27
13 Sprachverarbeitung	27
13.1 Sprecherunabhängigkeit	27
13.1.1 Frequency Invariance	28
13.1.2 Multi-Speaker Reference Model	28
13.1.3 Cross-Language DNNs	29
13.2 Lippenlesen	30
13.2.1 McGurk Effekt	30
13.2.2 Maschinelles Lippenlesen	30
14 Machinelle Übersetzung	32
15 Verarbeitung natürlicher Sprache	32
16 Control - Robotik	32
16.1 Kontrolltheorie/Grundlagen	32
16.2 Distal Learning Approach	33
16.3 Distally Trained Inverse Model	33
16.4 Distal Teacher with State	33
17 Computer Vision	34
17.1 Einführung Computer Vision	34
17.1.1 Aufgaben	34
17.2 Anwendungen	35
17.3 Features	35
17.4 Convolutional Neural Networks (CNN)	35
17.5 Training von CNNs	36
17.6 Image Captioning	37
17.6.1 Image Captioning mit RNNs	37
17.6.2 Image Captioning mit CNNs	37
17.7 Attention and Image Captioning	38
18 Erkennung von Handschrift	38

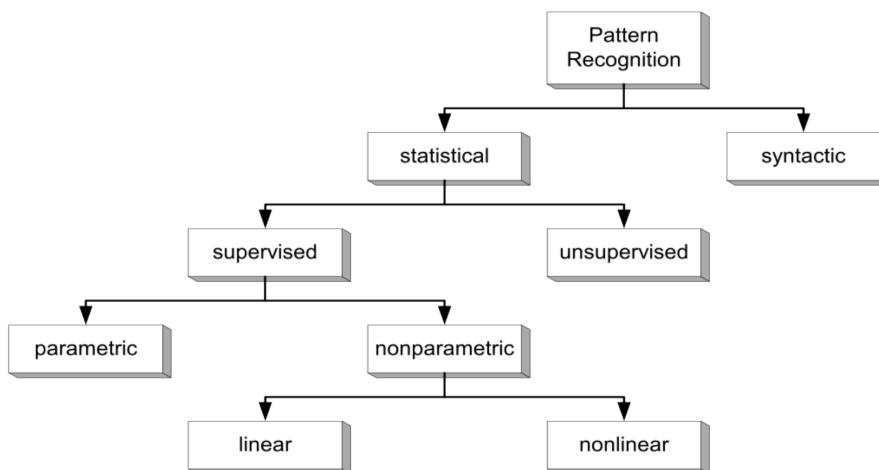
1 GRUNDLAGEN

- Aufbau des Gehirns: 1000 Supercomputer, 1/10 Taschenrechner, sehr gut für bestimmte Probleme (Sehen, Sprechen, Sprache)
- Von Neumann-Architektur
- Vorteile von neuronalen Netzen
- Generationen von NNs
- Geschichte der NNs
- Anwendungen von NNs
- Design Kriterien
- Modelle

1.1 WIESO NEURONALE NETZE?

- stark parallelisierbar
- einfache Recheneinheiten
- viele Units mit vielen Verbindungen
- nicht-lineare Klassifikatoren (gute Performanz)
- Lernen/Adaption
- dem Gehirn nachempfunden

1.2 MUSTERERKENNUNG



1.3 SUPERVISED LEARNING

- Labels sind bekannt -> ist aufwendig
- Klassifikation: Prediktion einer Klasse für ein Sample
- Regression: Approximation einer Funktion
- Parametrisch: verwendet zugrundeliegende Wahrscheinlichkeitsverteilung (z.B. Bayes)
- Nicht-Parametrisch: keine zugrundeliegende Wahrscheinlichkeitsverteilung (z.B. NN)

1.4 UNSUPERVISED LEARNING

- Label nicht für jedes Trainingssample bekannt
- Strukturanalyse
- mehrere Daten nötig
- Anwendungen: Clustering, Detektion von Anomalien, Dimensionsreduktion, Strukturvorhersage, Feature Lernen

1.5 REINFORCEMENT LEARNING

- Label/Klasse nicht für jeden Lernschritt bekannt
- Umgebung muss teilweise bekannt sein
- Belohnung/Bestrafung muss beobachtbar sein
- Anwendung: Sequential Decision Making

1.6 FEATURES

	\times	\div	$+$	$-$	$<$	$>$	$=$	\neq	
Nominal							✓		Qualitative
Ordinal					✓		✓		Qualitative
Interval			✓		✓		✓		Quantitative
Ratio	✓		✓		✓		✓		Quantitative

1.7 NORMALISIERUNG

- Max-Min (Rescaling): $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$
- Standardisierung: $x' = \frac{x - \bar{x}}{\sigma}$
- Skalierung auf Einheitslänge: $x' = \frac{x}{\|x\|}$
- lückenhafte Daten: Null filling - Smoothing

2 AKTIVIERUNGSFUNKTIONEN

2.1 STEP FUNCTION (SCHWELLWERTFUNKTION)

- Aktivierungsfunktion wurde in originalem Perzeptron verwendet
- Ableitung immer 0
- $\phi(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$

2.2 SIGMOID

- $\phi(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$
- leicht ableitbar: $\frac{\partial \phi(x)}{\phi(x)} = \phi(x)(1 - \phi(x))$
- Saturation gut für hidden Knoten, schlecht für Ausgabeknoten

2.3 TANGENS HYPERBOLICUS

- $\phi(t) = \tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$
- Ableitung: $\frac{\partial \phi(t)}{\phi(t)} = 1 - \tanh^2(t) = 1 - \frac{(e^t - e^{-t})^2}{(e^t + e^{-t})^2}$
- Ausgabe $-1 < \phi(t) < 1$
- Falls Ausgabe Mittel 0 hat, bleibt dies erhalten.

2.4 SOFTMAX

- generalisierte Sigmoid-Funktion
- $\phi(a_j) = \frac{e^{a_j}}{\sum_k e^{a_k}}$
- leicht ableitbar: $\frac{\partial \phi(a_j)}{\phi(a_j)} = \phi(a_j) - \phi(a_j)^2 = \phi(a_j)(1 - \phi(a_j))$
- Ausgabe ist a-posteriori Wahrscheinlichkeit $p(c|x)$
- gut für Klassifikation

2.5 LINEARE FUNKTION

- $\phi(t) = t$
- Ableitung immer 1
- Lineare Funktionen (auch in Kombination) können nur lineare Probleme lösen
- Kann in einem Knoten zur Funktionsapproximation verwendet werden

2.6 RECTIFIED LINEAR UNIT (RELU)

- $\phi(t) = \max(0, t)$
- biologisch plausibler
- Ableitung: $\phi'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$

3 FEHLERFUNKTIONEN

3.1 SUM-SQUARED-ERROR (SSE)

- $E_{MSE}(w) = \frac{1}{2} \sum_{x \in X} \sum_k (t_k^x - o_k^x)^2$
- größere Unterschiede werden stärker bestraft als kleinere
- gut für Regression (Funktionsapproximation)
- Mean-Squared-Error = $\frac{1}{N} * SSE$

3.2 CROSS-ENTROPY (CE)

- $E_{CE}(w) = - \sum_{x \in X} \sum_k [t_k^x * \log(o_k^x) + (1 - t_k^x) * \log(1 - o_k^x)]$
- Differenz der Wahrscheinlichkeitsverteilungen
- gut für Klassifikation

4 PERZEPTRON

TODO

5 GRADIENTENABSTIEG

$$w = w - \eta \nabla_w E(x, w) \text{ mit } \nabla_w E = \frac{\partial E}{\partial o} \frac{\partial o}{\partial \sigma} \frac{\partial \sigma}{\partial w}$$

- Häufiges Problem: Finde Parameter x so, dass Funktion $f(x)$ minimiert wird
- oftmals funktioniert der analytische Weg nicht -> Gradientenabstieg
- Ablauf
 1. Starte mit bestimmten Parameterwerten
 2. Update die Parameter iterativ entgegen der Richtung des Gradienten $w = w - \eta \Delta_w L(x, w)$ mit Fehlerfunktion L und Schrittweite η
 3. solange bis Gradient nahe genug an 0 ist

Gradientenabstieg mit logistischer Regression:

- Fehler: $L(x, w) = - \sum_k [t_k^x \log(o_k^x) + (1 - t_k^x) \log(1 - o_k^x)]$
- Aktivierung: $o_k^x = \sigma(wx_k) = \frac{1}{1+e^{-wx_k}}$
- $\Delta_w L = \frac{\partial L}{\partial o} \frac{\partial o}{\partial \sigma} \frac{\partial \sigma}{\partial w} = \frac{o-t}{o(1-o)} o(1-o)x = (o-t)x$
- Update-Regel: $w = w - \eta(o-t)x$ oder $w = w + \eta(t-o)x$

5.1 VARIENTEN DES GRADIENTENABSTIEGS

5.1.1 BATCH

- Update Parameter nachdem alle Trainingsbeispiele betrachtet wurden (1 Epoche)
- Konvergiert zum bestmöglichen Minimum
- Nachteile: Gradienten müssen für den kompletten Datensatz berechnet werden um Gewichte zu updaten, langsam und speicherintensiv, kein Online-Training

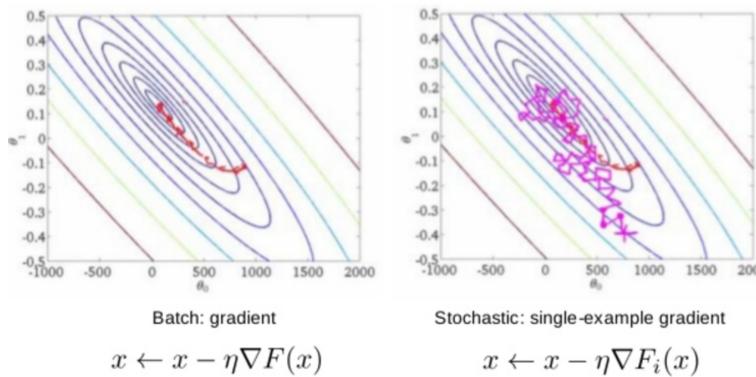
5.1.2 ONLINE (SGD)

- Update Parameter nach jedem Trainingsbeispiel
- Konvergiert deutlich schneller, kein großer Specheraufwand
- Online-Training
- Nachteile: Annäherung an Gradienten, hohe Varianz beim Updaten
- Praxis: Konvergiert zu einem guten lokalen Minimum nahe dem des Batch-Lernens

- Iteriere:
 1. $\text{grad} = 0$
 2. Vertausche Trainingsdaten
 3. Iteriere über jedes Trainingsbeispiel
 - a) Berechne Output $o_x = \sigma(wx)$
 - b) Berechne Gradient = $(t_x - o_x)x$
 - c) Update $w \leftarrow w + \eta \text{grad}$
 4. bis Konvergenzkriterium erreicht

5.1.3 MINI-BATCH

- Kompromiss zwischen batch und stochastischem Gradientenabstieg
- Update nach k Trainingsbeispielen (1 Iteration)
- state of the art
- Schneller als Batch Gradientenabstieg (Gewichter werden öfters geupdated, Speicherbelastung vorgebeugt, anwendbar für Online-Training)
- Schneller und stabiler als stochastischer Gradientenabstieg (weniger & kleinere Gewichtsupdates, leicht parallelisierbar (GPU-geeignet))



5.1.4 OPTIMIERUNG

- Fehler nach Gewichten ableiten ergibt Anpassung der Gewichte
- Gehe entlang dieser Richtung mit Schrittweite $\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}(t)}$
- Problem: richtige Schrittweite (zu groß: überspringen, zu klein: nicht erreichen des Minimum)

- Lösung: Analyse des Fehlerverhaltens mit Schrittweite -> adaptive Schrittweite (Anfang groß kleiner werdend wie Golf)
- Momentum-Term: Beschleunigung/Moment entlang der Achse des Fortschritts (um globales Minimum zu finden), bezieht letzte Schritte mit ein, ist auch einzustellen
- Aufteilung von Trainingsdaten
 - richtige Daten nicht zwangsweise wiederholen (richtig mit Schwellwert)
 - über alle Muster (genaue Ergebnisse, langsam)
 - für jedes Muster einzeln (sprunghafte Ergebnisse, schnell)
 - in der Praxis: UpdateWeight pro Epoche
 - Mini Batch Training: Training auf Unterdatenbasen
 - Fehlerkurve muss immer betrachtet werden
- Anpassung von Schrittweite durch Richtungsänderung von Gradient (Cosinus), Formel nicht wichtig (Folie 12)
- Hack: Konstante auf Ableitung addieren führt zu schnellerer Konvergenz (z.B. XOR), manchmal schlechtere Generalisierung (mathematisch nicht korrekt) -> besser: Normierung der Fehlerfunktion v.a. bei vielen Schichten
- Quickprop (auch mathematisch falsch): Annahme Fehlergewichtskurve hat Form einer Parabel, Sprung von Punkt in Minimum (direkt), Neuberechnung da nicht parabelförmig -> erneute Berechnung, deutlich schneller v.a. für hochgradig nichtlinear
- zufällige Initialisierung der Gewichte: -0.3 bis +0.3 lineare Region der Sigmoid-Funktion
- Output-Codierung: für jede Klasse ein Ausgabe-Neuron (One Hot) Klasse A: 1000 B: 0100 C: 0010 D: 0001 (schlecht bei vielen Klassen -> Netz tendiert alles auf 0 zu schalten) -> richtige und besten falschen Klasse (am stärksten ausschlagende) und deren Differenz maximieren mit anderer Fehlerfunktion (oder cross entropy zwischen richtig und falsch oder Softmax optimiert Wahrscheinlichkeit der richtigen über Summe der falschen Klassen)

5.2 ADAPTIVE LERNRATEN

- Idee: kleinerwerdende Lernrate
- schnell am Anfang, konvergiert am Ende
- einfaches aber effektives Anpassen der Lernrate: $\eta_{t+1} = \beta\eta_t$
- Es existieren andere Lernratenscheduling-Methoden
- nicht zwangsweise einfach

5.2.1 ADAGRAD

- Idee: adaptive Lernrate, größere Updates für seltene Parameter und kleinere Updates für häufige Parameter
- Gradientenabstieg $w_t = w_{t-1} - \eta \Delta_w L(x, w_{t-1})$
- Adagrad $w_t = w_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} L(x, w_{t-1})$ mit der Diagonalmatrix G_t , die die Beträge des Gradienten enthält und ϵ : Smoothingterm, um Division durch 0 zu verhindern
- Vorteile: gut bei wenig Daten, kein manuelles Anpassen der Lernrate nötig
- Nachteile: G_t muss gespeichert werden, G_t wird über die Zeit aufsummiert -> adaptive Lernrate wird zeitlich kleiner

5.2.2 ADADELTA

- verhindert Nachteile von Adagrad
- Idee: ähnlich zu Adagrad, aber speichere nicht alle Gradienten, sondern nur wenige (normalerweise die 2 letzten)
- Adagrad: $w_t = w_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$
- Adadelta: $w_t = w_{t-1} - \frac{RMS[\Delta w]_{t-1}}{RMS[g]_t} g_t$, wobei $RMS[\Delta w]_t$ der „root mean squared error“ $\sqrt{E[\Delta w^2]_t + \epsilon}$ ist.
- Vorteile: verhindert Nachteile von Adagrad, Lernrate muss nicht initial gesetzt werden
- Nachteile: Gradient zweiter Ordnung $E[\Delta w^2]_t$ muss berechnet werden

5.2.3 RMSPROP

- sehr ähnlich zu Adadelta und Adagrad aber einfacher als Adadelta
- Adagrad: $w_t = w_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$
- Adadelta: $w_t = w_{t-1} - \frac{RMS[\Delta w]_{t-1}}{RMS[g]_t} g_t$
- RMSprop: $w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$

5.2.4 ADAM - ADAPTIVE MOMENT ESTIMATION

- Adaptive Lernrate kombiniert durch Gradienten erster und zweiter Ordnung
- RMSprop: $w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
- Adam: $w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$ mit dem Durchschnitt der Gradienten \hat{m}_t und dem Durchschnitt der quadrierten Gradienten \hat{v}_t

- konvergiert schneller, aber eher Overfitting
- 2 Matrizen von vorherigen Gradienten müssen gespeichert werden

5.3 GENERALISERUNGSFÄHIGKEIT

- Fähigkeit, das aus den Trainingsdaten Erlernte auf neue Daten (Testdaten) anzuwenden.
- Gründe für schlechte Generalisierung: Overfitting/Overtraining (zu lange trainiert), zu viele Parameter (Gewichte) bzw. zu wenig Trainingsdaten, falsche Netzwerkstruktur

Methoden zur Verbesserung der Generalisierung

- Reduzierung der Komplexität durch Regularisierung
 - Weight Decay: Anpassung der Fehlerfunktion, Gewichte werden aufaddiert
 - Weight Elimination: Anpassung der Fehlerfunktion, Netzwerke wird für große $|w|$ bestraft
 - Optimal Brain Damage: entferne unwichtigste Verbindungen (also die, die den geringsten Einfluss auf den Fehler haben)
 - Optimal Brain Surgeon
- Schrittweise Vergrößern eines zu kleinen Netzes
 - Cascade Correlation ist ein konstruktiver Algorithmus zum erzeugen von Feed-Forward Neuronalen Netzen. Diese haben eine andere Architektur als typische multilayer Perceptrons. Bei Netzen, welche durch Cascade Correlation aufgebaut werden, ist jede Hidden Unit mit den Input-Neuronen verbunden, mit den Output-Neuronen und mit allen Hidden Units in der Schicht zuvor.
 - Meiosis Netzwerke bauen ein neuronales Netz auf. Sie beginnen mit einer einzelnen hidden Unit. Diese hidden Unit wird aufgespalten, wenn die „Unsicherheit“ zu groß ist.
 - Automativ Structure Optimat (ASO) passt Anzahl der Hidden Units, Größe des Input-Fensters und Anzahl der Zustände automatisch im Training an.

5.4 REGULARISIERUNG

- L1 Norm: $\|w\|_{L1} = \sum_j |w_j|$
- L2 Norm: $\|w\|_{L2} = \sqrt{\sum_j w_j^2}$

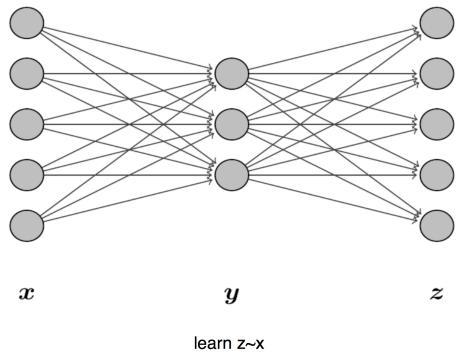
6 UNÜBERWACHTES LERNEN / AUTOENCODERS

Anwendungszwecke von unüberwachtem Lernen:

- Clustering (Ballungsanalyse)
- Erkennung von Anomalien
- Dimensionalitätsreduzierung
- Strukturvorhersage
- Feature Learning: schwierig bspw. bei Sprache, wie können Features gelernt werden?
-> Autoencoder

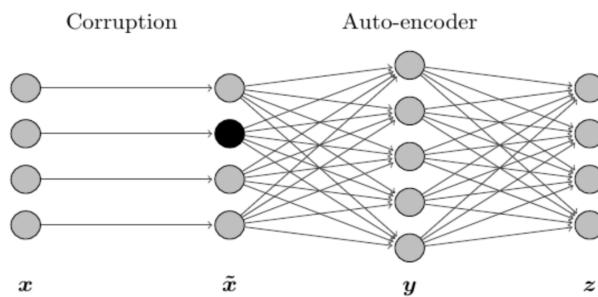
6.1 AUTOENCODER

- NN wird trainiert Eingabe zu rekonstruieren
- 3-schichtiges MLP (Architektur auf Folien)
- lineare Aktivierungsfunktion würde kein Encoding lernen -> nicht-lineare AF
- hidden units lernen Features aus den Daten
- Anzahl der hidden units abhängig von Anzahl und Struktur der Daten

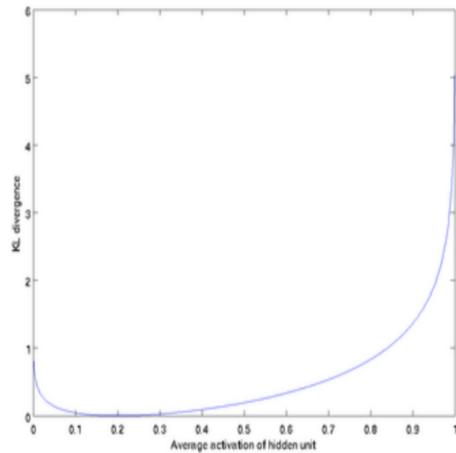


6.1.1 ERWEITERUNGEN

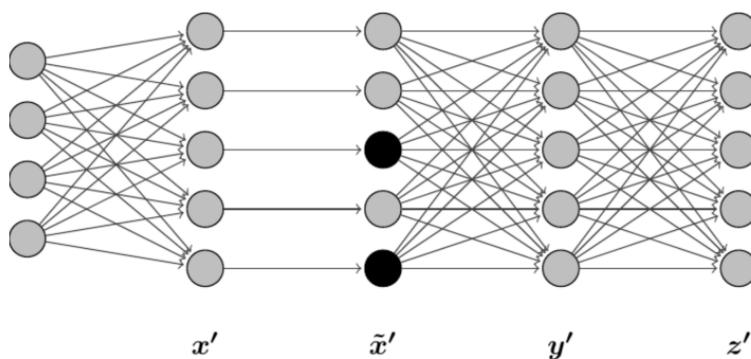
- Denoising Autoencoders: füge Rauschen zu Eingabe hinzu und rekonstruiere die nicht verrauschte Eingabe mit Autoencoder



- Sparse Autoencoders: reduziere Aktivierung der Neuronen, viele inaktive, wenig aktive (sparse active) behalten die Features mit dem höchsten Informationsgehalt



- Stacked Autoencoders: zusätzliche Hidden-Schicht um Features aus der gefunden Repräsentation (1. Hidden-Schicht) zu extrahieren

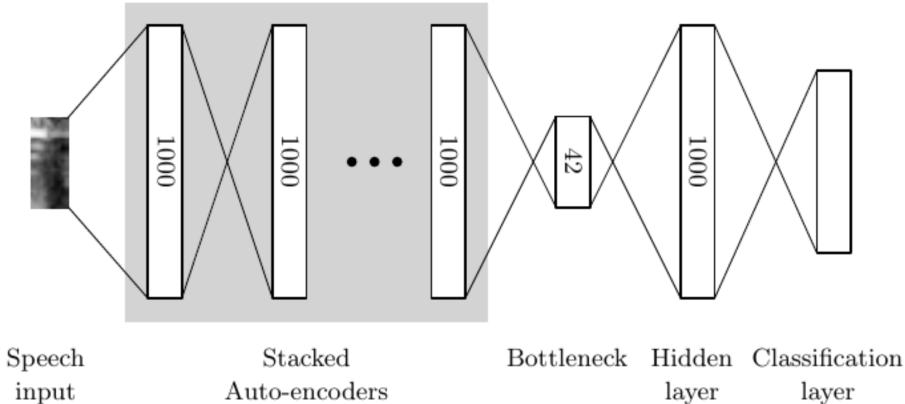


6.1.2 ANWENDUNGEN

- Feature Learning: Automatischen Lernen von Features und Kombinationen dieser
- besseres Training: bessere Initialisierung der Gewichter
- Dimensionalitätsreduktion: reduziere Modellgröße, schönere Visualisierung
- Rauschreduktion

6.2 BOTTLENECK FEATURES

- kleine Hidden-Schicht (weniger Neuronen als Schichten davor und danach) mit gelabelten Daten
- kann für eine niedrigdimensionale Repräsentation der Input-Features verwendet werden



7 RBM

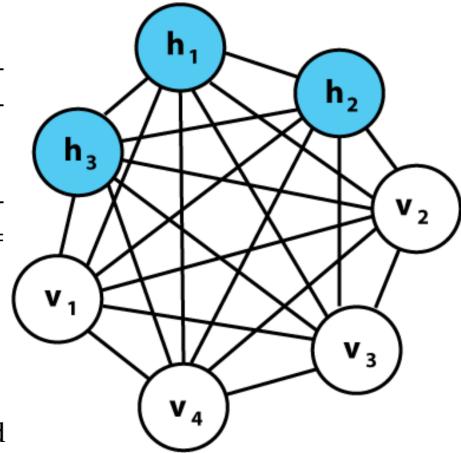
7.1 HOPFIELD NETZE

- nicht sehr effizient
- entsprechen statistischer Mechanik
- Anwendungen: Assoziativspeicher, Optimierung
- einschichtig
- binäre units $u_i = -1 \text{ or } 1$ (Notation +/-)
- Vektor der Zustände der units $U = (u_1, u_2, \dots, u_n) = (+, +, -, \dots, +)$
- vollverbunden
- rekurrent
- Netzwerk kann sich in stabilen Zustand einpendeln
- Gewicht von unit j zu unit i ist T_{ij}
- Symmetrische Gewichte $T_{ji} = T_{ij}$ führen zu Konvergenz (einpendeln in stabilem Zustand)
- Trainingsmethode anwendungsspezifisch
- Updateregel: $u_i = g(x_j) = \begin{cases} 1 & \text{für } \sum_{i,i \neq j} u_i T_{ji} \geq 0 \\ -1 & \text{sonst} \end{cases}$ (vorheriger Zustand wird nicht beachtet)
- Update asynchron (neuronweise) oder synchron (alle units zeitgleich) solange bis Netz stabilen Zustand erreicht

- Energiefunktion: $E = -\frac{1}{2} \sum_j \sum_{i,i \neq j} u_i u_j T_{ji}$ wird im Netz optimiert

7.2 BOLTZMANN MASCHINEN

- stochastisches rekurrentes neuronales Netz
- lernen interne Repräsentationen
- Problem: beliebige Kanten (ohne Bedingungen)
- Zustände visible (beobachtete Daten, Eingabe/Ausgabe) und hidden (zu lernende Variablen), zusätzlich noch Bias, Werte 0 oder 1
- Entscheidung ob Zustand aktiv ist ist stochastisch $z_i = b_i + \sum_j s_j w_{ij} \rightarrow p(s_i = 1) = \frac{1}{1+e^{-z_i}}$
- Energie $E = - \sum_{i < j} w_{ij} s_i s_j - \sum_i b_i s_i$



- Pro: Boltzmann Maschinen mit genügend Hidden Neuronen können jede Funktion berechnen
 - Kontra: Training ist langsam und rechenintensiv
- Simulated Annealing

- starte mit hoher Temperatur
- abkühlen durch verringern von T
- Ziel: Ausbrechen aus lokalen Minima

7.3 RESTRICTED BOLTZMANN MASCHINEN

- Boltzmann Maschinen mit Nebenbedingungen: Graph muss bipartit sein (-> effizienteres Training)
- Energie: $E(V, H) = - \sum_{i=1}^m \sum_{j=1}^F W_{ij} h_j v_i - \sum_{i=1}^m v_i a_i - \sum_{j=1}^F h_j b_j$
- Wahrscheinlichkeit von hidden unit: $p(h_j = 1 | V) = \sigma(b_j + \sum_{i=1}^m W_{ij} v_i)$
- Wahrscheinlichkeit von Eingabevektor: $p(v_i | H) = \sigma(a_i + \sum_{j=1}^F W_{ij} h_j)$ mit Aktivierungsfunktion $\sigma = \frac{1}{1+e^{-x}}$

- Softmax Units: Generalisierung von binären Units auf K verschiedene Zustände:

$$p(s_i = j) = \frac{e^{z_{ij}}}{\sum_{i=1}^K e^{z_{ij}}}$$

- Training: Gradientenabstieg zur Erhöhung des Log-Likelihood

- Update

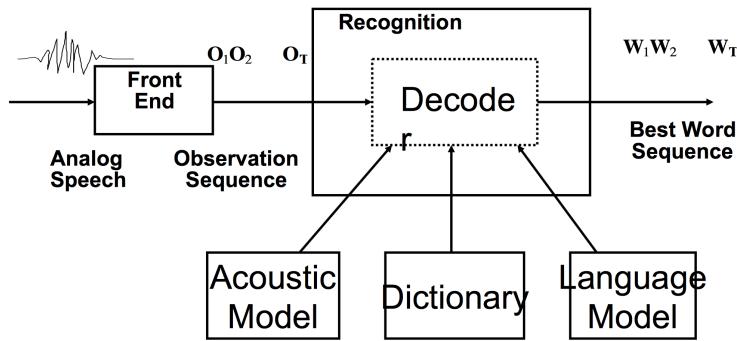
$$-\Delta W_{ij}^k = \epsilon \frac{\partial \log(p(V))}{\partial W_{ij}^k} = \epsilon (< v_i^k h_j >_{data} - < v_i^k h_j >_{model})$$

– kann nur in exponentieller Zeit berechnet werden -> andere Funktion (Constrastive Divergence)

– $\Delta W_{ij}^k = \epsilon (< v_i^k h_j >_{data} - < v_i^k h_j >_T)$ wobei $< v_i^k h_j >_T$ die Verteilung eines Gibbs samplers nach T Zeitschritten ist ????

8 SPRACHERKENNUNG - TDNN/CNN

8.1 KOMPONENTEN EINES SPRACHERKENNERS



8.2 GRUNDLAGEN DER SPRACHERKENNUNG

- Ziel
 - Gegeben akustische Daten $A = a_1, a_2, \dots, a_k$
 - Finde Wortsequenz $W = w_1, w_2, \dots, w_n$ so, dass $P(W|A)$ maximiert wird
- Bayes Regel: $P(W|A) = \frac{P(A|W) \cdot P(W)}{P(A)}$ wobei $P(A|W)$ das akustische Modell, $P(W)$ das Sprachmodell und $P(A)$ eine Konstante für eine komplette Sequenz ist.

8.3 HIDDEN MARKOV MODELS

- Zustände $S = \{s_0, s_1, \dots, s_N\}$
- Übergangswahrscheinlichkeiten $P(q_t = s_i | q_{t-1} = s_j) = a_{ji}$
- Ausgabewahrscheinlichkeiten $P(y_t = O_k | q_t = S_j) = b_j(k)$ (Zustand j für Symbol k)

- Emissionswahrscheinlichkeiten können geschätzt werden durch...
 - Gauss-Mixturen
 - Neuronale Netze
 - Hierarchies of Neurons

8.4 TIME-DELAY NEURAL NETWORK (TDNN)

- MLP mit nichtlinear Entscheidungsoberfläche
- appropriate architecture: Integration des Sprachwissens. Minimiere Lerndauer und notwendige Traingsdaten
- Time-Delay: Netwerke können zeitliche Strukturen der Sprache repräsentieren
- Verschiebungsinvarianz: hidden units lernen Features zeitunabhängig
- Units mit freien (lernbaren) Eingabeverbindungen
- tiefe Features - pretrained
- Finetuning - nochmal durchgehen und Adaption
- heute viel größere Netze
- Verschiebungs-Invarianz über Zeit hinweg
- unüberwachtes Pretraining mit Autoencodern - encodieren des Sprachsignals

8.5 2-DIMENSIONALE TDNNs

- Kacheln, die verschiebungsinvariant über Zeit und Frequenz sind
- menschliche Performanz bei Spracherkennung
- Hall durch Reflexionen an Wand produziert schwammiges Spektrum (Mikrofone nicht direkt vor dem Mund) -> nahe Mikrofon für Lecture Translator schwächt Verhallungseffekt ab, Verhallung in jedem Raum unterschiedlich -> direkte Enthallung, Verhallung = gefaltetes Signal, Kanalrauschen (Faltung auf Signal)

8.6 CONVOLUTIONAL NEURAL NETWORKS (CNN)

- lokale Merkmale (z.B. Kantendetektoren) -> verschiebungsinvariant
- Kacheln im xy-Bereich verbunden mit Neuronen
- Faltung von Neuronenpositionen mit Gewichten -> Verschiebung über xy-Bereich -> Aktivierung

- mehr als 1-2 hidden layer -> größerer Kontext -> Reduzieren der Auflösung, damit Input nicht komplett weiter geführt werden muss
- gute Ergebnisse in Bildverarbeitung (z.B. Handschrifterkennung, verrauschte Straßen-schilder / Nummernschilder)
- liefert Feature-Detektoren für Klassifikation
- Benchmark Imagenet
- Go-Programm verwendet CNN für Training von Reinforcement Learning
- Spracherkennung: Reihenfolge wichtig, z.B. Cat - ist C, a und t vorgekommen? act? Sprechgeschwindigkeit modellieren? -> Wortmodellierungstechniken notwendig
- unabhängig der Zeit durch shift-Invarianz
- Erkennung von Sequenzen (z.b. abc - 1, cba - 0 unabhängig von Länge der Buchstaben)
- Dynamic Timewarping (Viterbi-Decoder): Alignierung zwischen zwei Achsen, unterschiedliche Sprechgeschwindigkeiten nicht linear -> nicht-lineares time alignment (Folie 61) -> Alinierungspfad der 2 Achsen aufeinander anpasst, Vektoren die gut oder schlecht aufeinander passen, Pfad für gute Zuordnung, iteratives/induktives Verfahren
- Statistische Spracherkennung: akustisches Modell (Bayes/HMM)
- HMM: Zustände (Phoneme von Wörtern), Übergänge mit Wahrscheinlichkeiten, Emissionswahrscheinlichkeiten in Zuständen
- HMM in der Spracherkennung: pro Phonem 3 HMM-Zustände (begin,middle,end)
- Decoder sucht Pfad, der am besten auf Sprachsignal passt (z.B. mit Viterbi), Aufpassen bei Wortübergängen (Viterbi nur Alinieren) -> moderne Decoder können sowas -> große Alinierungsmatrizen (Backtrace liefert Wahrscheinlichkeit der Worte)

8.7 MULTI-STATE TIME DELAY NEURAL NETWORK (MS-TDNN)

TODO

8.8 NN-HMM HYBRIDS

TODO

9 VEKTORQUANTISIERER LERNEN

9.1 VECTOR QUANTIZATION (VQ)

- approximieren den Datenraum mit einer kleineren Anzahl an Vektoren
- kategorisieren ein Set von Eingabevektoren x mit M Klassen
- Jede Klasse hat einen assoziierten Prototyp-Vektor (alle Prototypen zusammen ergeben Codebook)
- Jeder Vektor wird durch seine Klasse repräsentiert.
- ähnlich zu „competitive learning“ -> finde Klasse durch Gewinner $\|u_c - x\| \leq \|u_i - x\|$ für alle i
- Anwendungen: Multimedia-Kompression, Dimensionsreduktion, Klassifikation

9.2 LEARNING VECTOR QUANTIZATION

- überwachte Form von Vektorquantisierern
- verschiedene Algorithmen: LVQ1, LVQ2, LVQ3 und QLVQ
- Optimale Entscheidung durch Bayes Theorie
- Ansatz
 - Jeder Klasse S_k wird sein Teil des Codebooks zugewiesen
 - Suche nach nächstem Codebook-Vektor m_i zur Eingabe x
 - x wird zur gleichen Klasse wie m_i klassifiziert

9.3 LVQ1

- Annahme: mehrere Codebook-Vektoren jeder Klasse zugewiesen und x dem nächsten m_i zugewiesen
- Index vom Gewinner-Codebook c : $c = \operatorname{argmin}_i \{\|x - m_i\|\}$
- Eingabebeispiel $x(t)$
- $m_i(t)$ sind diskrete Werte von m_i
- Starte mit wohldefinierten Initialwerten
- Wende Lernregel für jedes x an: $m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$ falls x und m_c der gleichen Klasse angehören, sonst $m_c(t+1) = m_c(t) - \alpha(t)[x(t) - m_c(t)]$
- Samples werden zyklisch angewandt

9.4 LVQ2

- Verschiebe die Entscheidungsgrenzen in Richtung der Bayesian Limits
- Selbe Klassifikationsentscheidung wie LVQ1
- Unterschiede:
 - 2 Codebook-Vektoren m_i und m_j werden simultan geupdated
 - x muss in einem Fenster in der Nähe von m_i und m_j sein

9.5 LVQ2.1

- Verbesserung von LVQ2
- Erlaubt entweder m_i oder m_j der nächste Codebook-Vektor zu sein

9.6 LVQ3

- Verbesserung von LVQ2
- Korrektur damit m_i weiterhin die Klassenverteilung ($f(x)$) approximiert

9.7 INITIALISIERUNG DER CODEBOOK-VEKTOREN

- iterative Zuweisung: da Klassenverteilungen unbekannt sind können optimale Zahlen nicht frühzeitig bestimmt sind
- praktischer Schritt: Starte mit gleicher Anzahl an Codebook-Vektoren in jeder Klasse
- Bestimme minimale Anzahl an Codebook-Vektoren pro Klasse
- Initialwerte der Codebookvektoren: k_Nearest-Neighbor Test

9.8 ANWENDUNG IN DER PRAXIS

- Optimales Lernen: beginne mit OLVQ1 für schnelle Konvergenz und mache dann mit anderem Algorithmus mit kleiner Lernrate weiter
- Stoppregel: wie NN -> teile in Trainings-, Validierungs- und Testdaten

10 SELF-ORGANIZING MAPS

10.1 SELBST-ORGANISIERTES LERNEN VS STATISTISCHEN LERNEN

Selbst-organisiertes Lernen mit neuronalen Netzen

- Motiviert durch neurobiologische Systeme
- Eingabe-Ausgabe Berechnung abhängig von lokaler Nachbarschaft
- Lernen bedingt durch Struktur der Trainingsdaten

Statistisches Lernen

- klassischer Ansatz im maschinellen Lernen
- Augenmerk liegt auf etablierter Mathematik

10.2 PRINZIPIEN DER SELBST-ORGANISATION

1. Self-amplification (Selbstverstärkung): Eigenschaften des Hebb'schen Lernens
2. Competition: winner-takes-all Ansatz
3. Cooperation
4. Structural Information

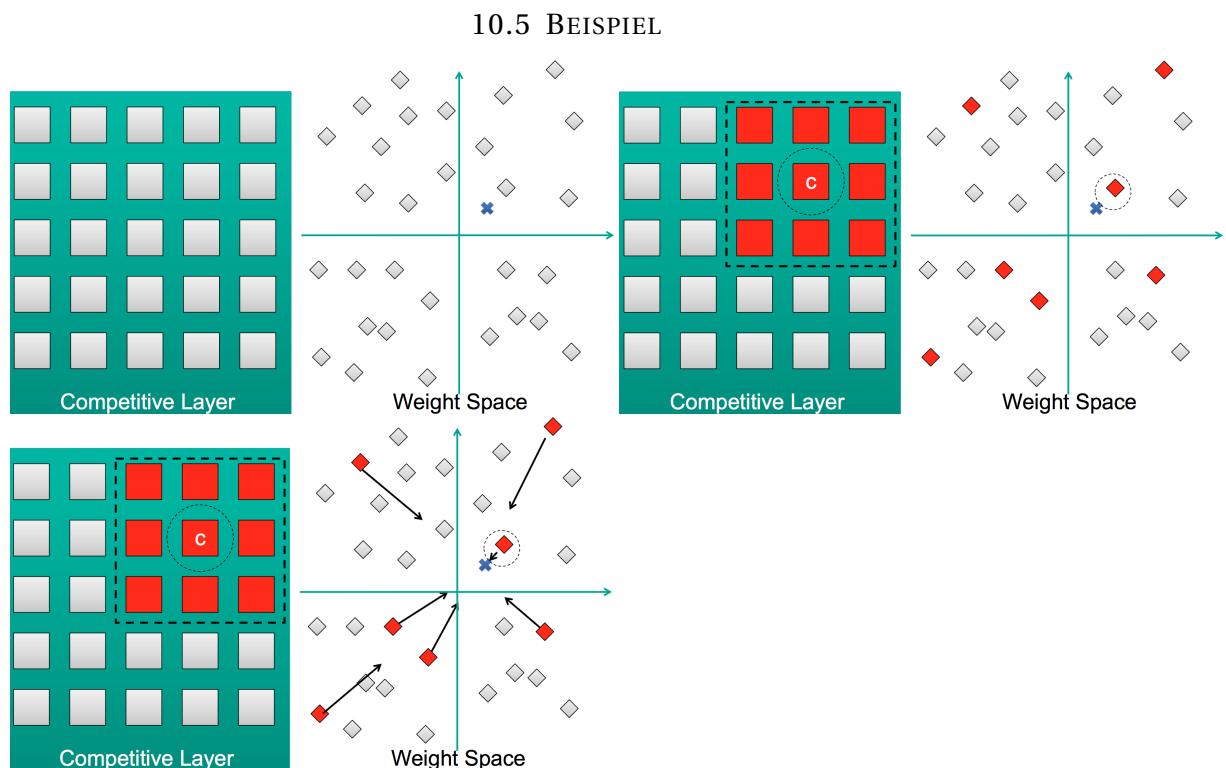
10.3 GRUNDLAGEN

- Unterklasse der neuronalen Netzen
- basiert auf self-organized competitive learning
- Ausgabeunit berechnet als winning neuron
- Neuronen platziert auf Knoten eines ein- oder zweidimensionalen Gitters
- Neuronen werden selektiv optimiert an verschiedene Eingabemuster oder Klassen von Eingabemustern
- Nachbarschaftsbeziehungen im Gitter beeinflussen Gewichtsupdates

10.4 ALGORITHMUS

1. Initialisiere synaptische Gewichte mit kleinen Zufallswerten.
2. Competition: Berechne Werte von allen Neuronen abhängig vom Eingabemuster. Das Neuron mit dem höchsten Wert gewinnt.
 - Daten $x = [x_1, x_2, \dots, x_m]^T \in X$

- Gewichtsvektoren zwischen Eingabe und jedem Neuron U_j gegeben durch $w_j = [w_{j1}, w_{j2}, \dots, w_{jm}]^T$ mit $j = 1, 2, \dots, l$
 - Berechne für alle U_j : $w_j^T x$
 - Neuron i mit höchstem Wert gewinnt
 - Gewinnerneuron für Vektor x hat Index $i(x) = \arg \min_j \|x - w_j\|$
 - Kontinuierlicher Eingaberaum wird auf diskreten Eingaberaum gemappt
3. Cooperation: Das Sieger-Neuron definiert den Mittelpunkt einer relevanten Nachbarschaft
- Nachbarschaft kann einfach durch einen Abstand der Neuronen $d_{k,i} < D$ definiert werden
 - Neurobiologische Motivation: Einfluss von näheren Neuronen ist höher als von weiter entfernten (Decay)
 - Definiere Nachbarschaftsfunktion $h_{k,i}$ um das Gewinnerneuron i mit relevanten Neuronen k
4. Synaptic adaption: Die Neuronen der relevanten Nachbarschaft passen ihre Gewichte an, um eine bessere Ausgabe zu erzeugen
5. Wiederhole 2-4 solange nötig



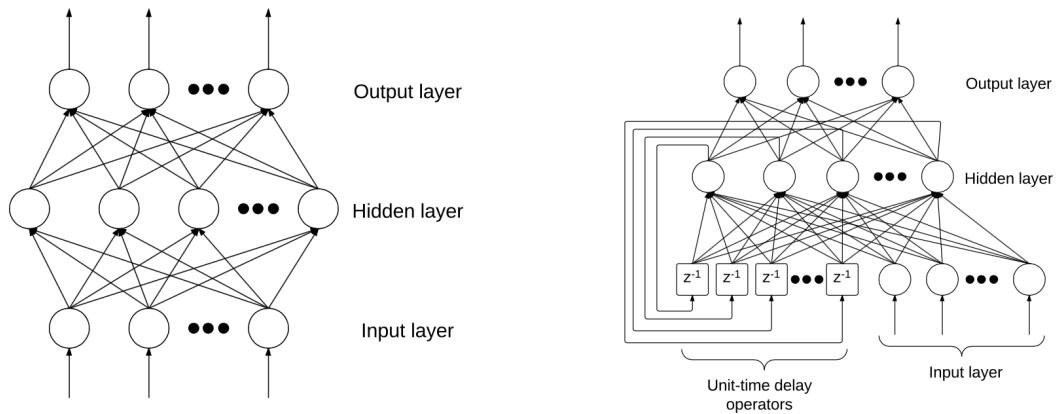
11 RNN

Sequence Learning

- Anwendungen: Börsenpreise, Zeichenfolgen, Wortfolgen, Folgen von Messdaten
- Ziele: Vorhersage des nächsten Wertes, Klassifikation von Sequenzen, Konvertieren von Sequenzen zu anderen Sequenzen, Generierung von Sequenzen
- Sequenzen haben verschiedene Längen (I am ..., Last week we were in ..., I like to go ...), shift invariant (I like to go, I really like to go, My best friend like to go) -> Auswerten der Sequenzen auf Wortebene

11.1 REKURRENTE NEURONALE NETZE

Feed Forward vs Recurrent Neural Networks

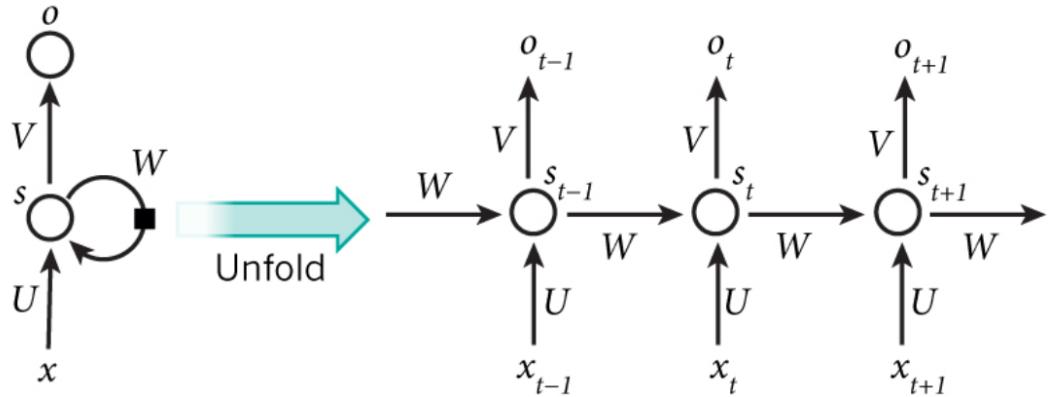


Architekturen

- Elman Networks: Ausgabe der hidden Schicht wird als Eingabe im nächsten Schritt verwendet
- Jordan Networks: Ausgabe des Netzwerks wird als Eingabe im nächsten Schritt verwendet

11.2 BACKPROPAGATION THROUGH TIME (BPTT)

- Wie trainiert man RNN?
- Durch BPTT kann ein RNN in ein Feed-Forward-Netz umgewandelt werden, anschließend kann BackProp angewendet werden



11.3 PROBLEME VON RNNs

- Future Context
 - RNNs nutzen nur vergangenen Kontext
 - Zukünftiger Kontext könnte Vorteile bieten, z.B. in Speech Tagging
 - Lösungen: Kombiniere Ausgaben von Forward und Backward RNNs, bidirektionale RNNs, verzögerter Output, stacked input context
- Vanishing Gradient
 - Gradient geht gegen 0, falls $\|max_{W_{hh}}\| \|max_{\phi'}\| < 1$
 - Gradient geht gegen ∞ , falls $\|max_{W_{hh}}\| \|max_{\phi'}\| > 1$
 - Lösung: setze $\|max_{W_{hh}}\| \|max_{\phi'}\| = 1 \rightarrow$ LSTM

11.4 LONG SHORT TIME MEMORY RNNs

- beuge vanishing gradient vor
- Idee: konstanter Fehlerfluss (?), $W_{hh}^T diag[\phi'(net_{i-1})] = 1$
- Ansatz: $W_{hh} = 1$ und $\phi(x) = x$
- Constant error carousel (CEC)

12 REINFORCEMENT LEARNING

TODO Marius

12.1 GRUNDLAGEN

12.2 POLICY-BASED RL

- Policy π ist eine Wahrscheinlichkeitsverteilung von Aktionen gegeben einem Zustand
- $a = \pi(s)$
- Anfänger würden die Dame wohl immer schlagen, fortgeschrittenere Spieler situationsabhängig
- suche direkt die optimale Policy π^*
-

12.3 VALUE-BASED RL

12.4 POLICY GRADIENTS MIT NEURONALEN NETZEN

12.5 Q-LEARNING

12.6 TEMPORAL DIFFERENCE LEARNING

12.6.1 SARSA

12.7 SCHWIERIGKEITEN

12.8 ALPHAGO

13 SPRACHVERARBEITUNG

13.1 SPRECHERUNABHÄNGIGKEIT

Normal TDNNs are time invariance. However men, women and children differ in *frequency*. Men normally have a darker voice than women and children. Therefore we have to compensate this invariance.

Observations on TDNNs

- TDNNs develop linguistically plausible features in the hidden units
- TDNNs developed alternate internal representations that can link quite different acoustic realizations to the same higher level concept (because of multilayer arrangement)
- hidden units fire synchrony because they operate independent of precise time alignment or segmentation (time invariant)
- small network output may not be useful in complex task (but the internal abstractions may be valuable)
- complex concepts -> use stages with different knowledge
- new learning strategies should be built in existing knowledge

Model invariance

- frequency shift, tilt, compression

Variability

- adaption
 - slow adaption -> modify weights
 - fast adaption -> pretrained specific submodels
- normalization
 - environment: to the room
 - speaker: mapping new speaker to standard speaker (with standard sentences)

Combine **two standard TDNNs** to one (overall better classification): one with MSE and the other with CFM. Those combination yields the correct classification.

Even better than two TDNNs: **three TDNNs!** The third TDNN uses CE.

13.1.1 FREQUENCY INVARIANCE

Convolutional Acoustic Models:

- parameter sharing across spectrum and time -> exploit 2D structure of features
- upper layer fully connected
- pooling gives more robustness and less overfitting

13.1.2 MULTI-SPEAKER REFERENCE MODEL

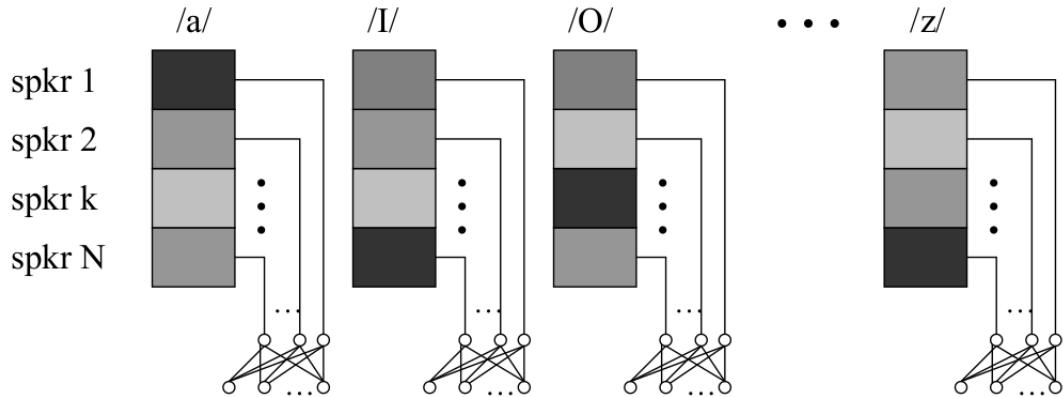
Idea: A speaker-specific reference model is composed from several well trained reference models:

Meta-Pi-Net:

A separate neural net *Meta-Pi-Net* controls the activation / deactivation of the single nets. The whole net is a combination of one net per speaker. In general we look which speaker net is closest to the input (which net classifies the best). The Meta-Pi-Net produces weights for each net which control how much each net is taking into account for the classification.

It is not important that the correct speaker is chosen. Sometimes a combination of two or more speakers might fit better to the input. The actual correct speaker net might not be even taking into account (results in a mixture of the other nets).

Speaker Normalization: we have speaker dependent nets. Now when we use those nets to classifier what another speaker says (no dependent net for this speaker!) the error rate is 41.9%. With 40 text-dependent training sentences the error rate is reduced to 6.8%.



phoneme specific reference model selection networks

i-vectors: identity vectors (i-vectors) describe the speaker-characteristic offset to an universal background model. Those i-vectors are used to train a recognition system.

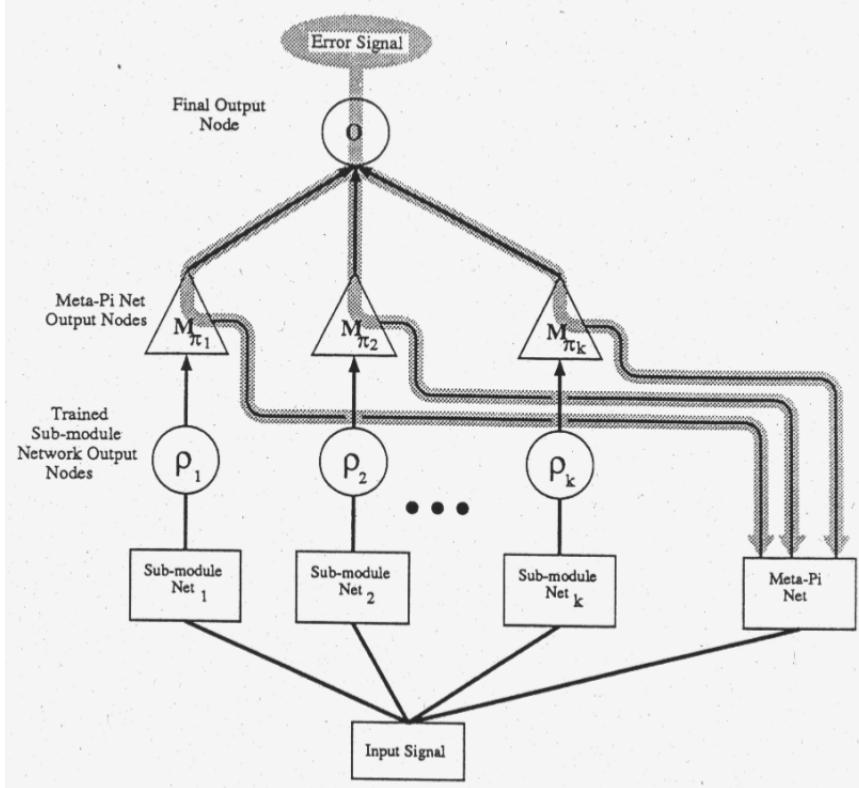
Speaker Adaptive Training of DNN:

1. train initial DNN with (and keep it fixed)
 - SI features: fbank
 - SA features: Feature space Maximum Likelihood Linear Regression (fMLLR)
2. Train an i-vector NN
 - inputs: i-vectors
 - outputs: linear shift to the original feature vectors
 - added features become speaker-normalized
3. update the DNN in the new feature space, i-vector NN fixed -> yields SAT-DNN

13.1.3 CROSS-LANGUAGE DNNs

Multilingual Bottleneck Features:

- Humans can only produce a finite amount of different sounds
- Subset of sounds is used in individual languages
- Some sounds are used in different languages



- Share data of the same sounds from different languages
- Extract more robust features using data with more variability

Cross-Language DNNs with Language-Universal Feature Extractors:

DNNs can be trained on multiple languages. The hidden layers (*language-universal feature extractor*) are used in other DNNs.

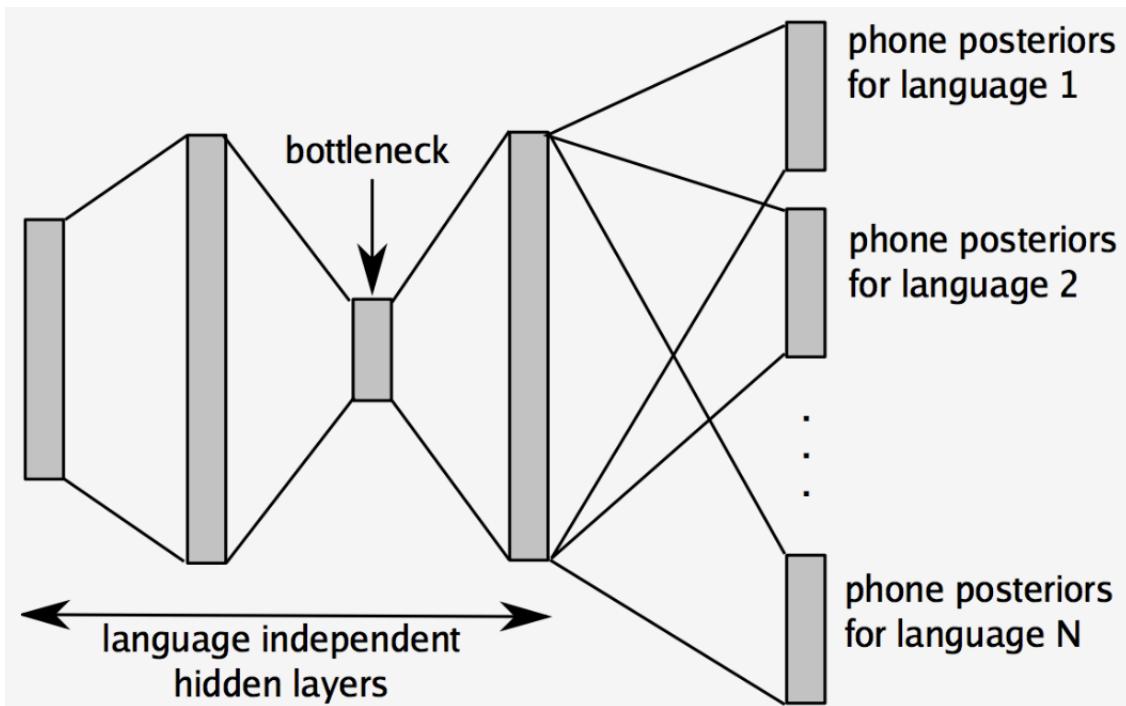
13.2 LIPPENLESEN

13.2.1 MCGURK EFFEKT

- gleicher Ton aber unterschiedliche Lippenbewegung
- Mensch erkennt verschiedene Lippenbewegung -> Wahrnehmung wird beeinflusst (unbewusstes Lippenlesen)
- nicht nur Audio für Spracherkennung relevant für Menschen -> auch Maschinen profitieren von Lippenlesen (z.B. bei starken Hintergrundgeräuschen)

13.2.2 MASCHINELLES LIPPENLESEN

- Kamera -> Face Tracking (facial points) -> Lip Finding (feature based) -> Kombination mit akustischen Signalen



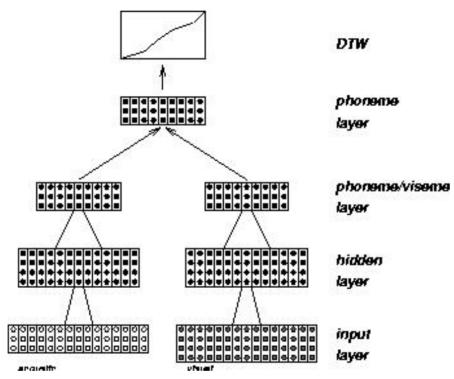
- Preprocessing wichtig, sonst anfällig für Beleuchtungsunterschiede
- Architekturen: MS-TDNN kombiniert Lippenlesen und Spracherkennner

Audio-Visual Recognizer



$$hyp_c = \lambda_a hyp_a + \lambda_v hyp_v$$

$$1 = \lambda_a + \lambda_v$$



Features

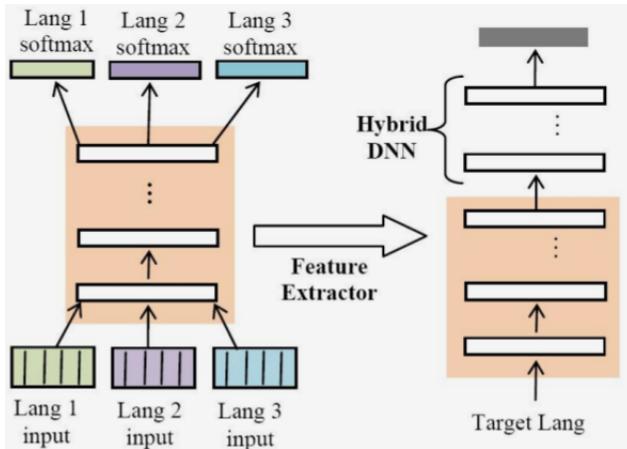
- What Features to Use?

Fusion Level

- Feature Vector
- Phone Streams
- Word Level

Fusion Methods

- Trained Weights
- Entropy Weights
- SNR Weights



14 MACHINELLE ÜBERSETZUNG

TODO Marius

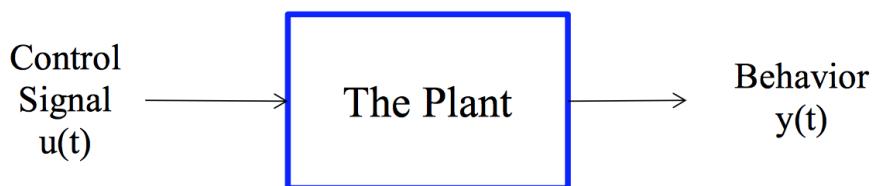
15 VERARBEITUNG NATÜRLICHER SPRACHE

TODO Marius

16 CONTROL - ROBOTIK

16.1 KONTROLLTHEORIE/GRUNDLAGEN

- Kontrollprobleme: Roboterarm soll Bierflasche greifen, Auto parallel parken, Flugzeug landen, ...
- Kontrolltheorie: Regler entwerfen, Stabilität überprüfen, adaptive Regelung, ...
- Regler oft manuell designt
- Neuronale Netze (typischer RNN) können Regelung lernen
- „Plant“(Blackbox) wird geregelt



- Forward Model: Eingabe $u(t)$ liefert nach Plant Ausgabe $y(t)$ und in Forward Model $\hat{y}(t)$. Nehme Fehler und Backprop.

- Inverses Model: $y(t) \rightarrow u(t)$ durch Plant und $y(t) \rightarrow \hat{u}(t)$ durch Inverses Model

16.2 DISTAL LEARNING APPROACH

- Trainiere Forward Model mit Backprop -> Freeze
- Verwende Backprop Fehler ($\frac{\partial E}{\partial y_i^T}$) um Regler oder inverses Model zu trainieren
- Distal learning ist zielgerichtet: konzentriere auf relevante Regionen des Zustandsraums
- Nicht gesamter Raum muss bekannt sein
- Fehler muss nur in relevanter Region gering sein
- Ähnlich zu Reinforcement Learning

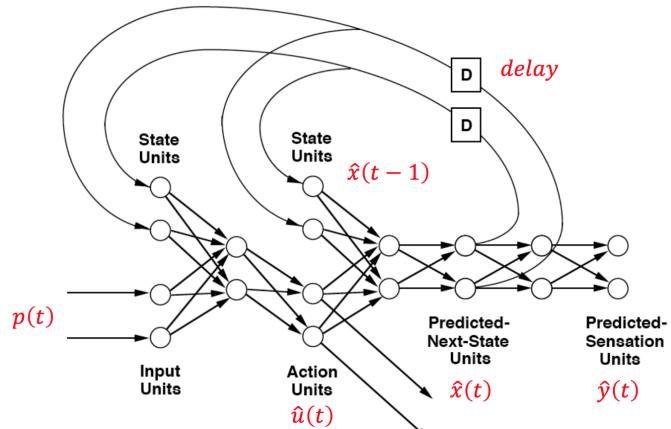
16.3 DISTALLY TRAINED INVERSE MODEL

- Trainiere Forward Model zusammen mit direktem inversen Model
- Freeze Forward Model
- Retrain inverses Model mit Forward Model als Distal Teacher
- liefert geringeren Fehler als direkt trainiertes inverses Model

16.4 DISTAL TEACHER WITH STATE

- Zustand ist in statischer Umgebung nicht wichtig, bei sich ändernder (dynamischer) Umgebung dagegen schon (Bsp: Roboterarm dessen Winkel nicht direkt gesetzt werden, sondern über Motoren gesteuert werden)
- Forwardmodel sagt sowohl den nächsten Zustand als auch die „Sensations“ voraus.

Recurrent Network with Forward Model

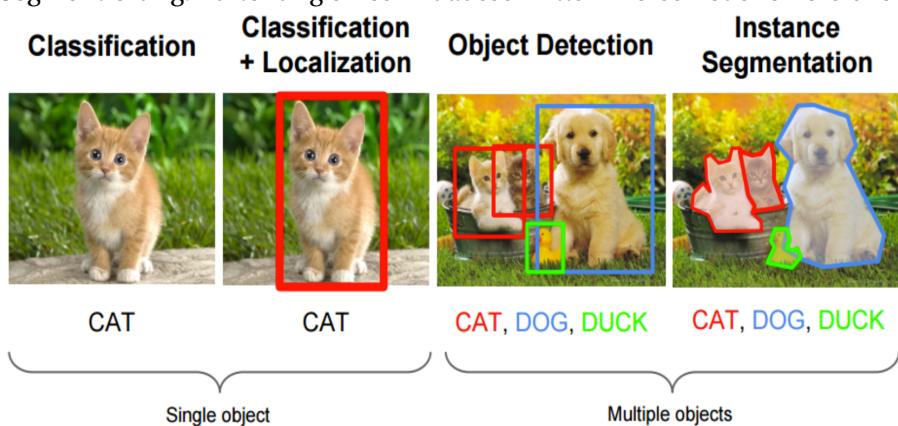


17 COMPUTER VISION

17.1 EINFÜHRUNG COMPUTER VISION

17.1.1 AUFGABEN

- Klassifikation: Unterscheidung zwischen verschiedenen Klassen
- Lokalisation: Wo befindet sich ein Objekt? -> Sliding Window
- Detektion: Was befindet sich wo? -> Sliding Window + Klassifikation oder besser Regioning, da Sliding Window sehr rechenaufwendig ist
- Segmentierung: Aufteilung eines Bildausschnitts in verschiedene Bereiche



- Training: Backprop Through Time

17.2 ANWENDUNGEN

- Robotik: Greifen unbekannter Objekte
- Autonomes Fahren: Ampelerkennung
- Überwachung: Erkennung von Gefahren

17.3 FEATURES

- wichtig für herkömmliches maschinelles Lernen
- Features wie SIFT, HoG, etc. funktionieren gut, aber...
- aktuelle Lernverfahren lernen Featurerepräsentation selbst
- Diese sind bspw. in CNNs absteigend in ihrer Feinheit (low-level (Kanten) -> mid-level (kleinere Bereiche) -> high-level (Bildausschnitte), rezeptives Feld wird durch Pooling vergrößert)
- Features unüberwacht lernen: Modellierung der Inputdaten z.B. über k-means, PCA, Independent Component Analysis, Autoencoder, RBMs

17.4 CONVOLUTIONAL NEURAL NETWORKS (CNN)

- Parameter die ein MLP zur Klassifikation von Bildern lernen müsste bewegen sich im Bereich von vielen Millionen
- CNNs wirken dem entgegen
- biologisch motiviert
- verschiebungsinvariant
- Aufbau
 - Convolutional Layer: Faltung (k Faltungskernel -> k Featuremaps)
 - Subsampling/Downsampling/Pooling Layer: nimmt Max/Durchschnitt eines Bildausschnitts (oder probabilistic pooling)
 - Fully connected Layer(s): Abbildung aller Features auf alle Klassen
 - Anwendungen: Bilderkennung, Sequenzlabeling, Handschrifterkennung

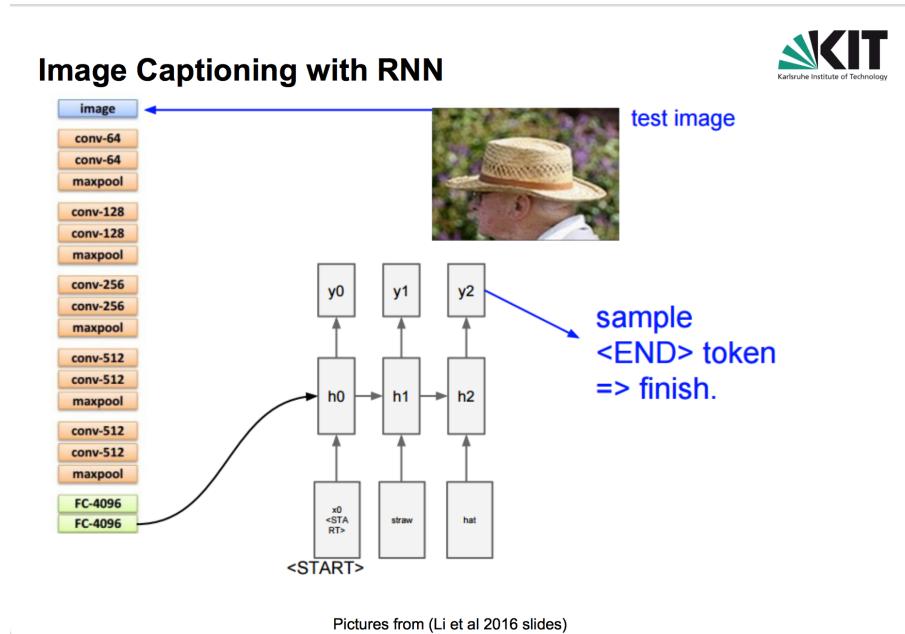
17.5 TRAINING VON CNNS

- BackProp mit Anpassungen
 - Teile des Inputs ist lokal mit Filter verbunden
 - Shared Weights in convolutional Layern
 - Aktivierungsfunktion verwendet in conv. Layern
 - In Pooling wird nichts gerlernt
- Fehlerfunktion: Cross Entropy
- Forward Pass
 - Conv Layer: Summiere gewichtete Einflüsse des vorherigen Layers und wende nichtlineare Aktivierung an

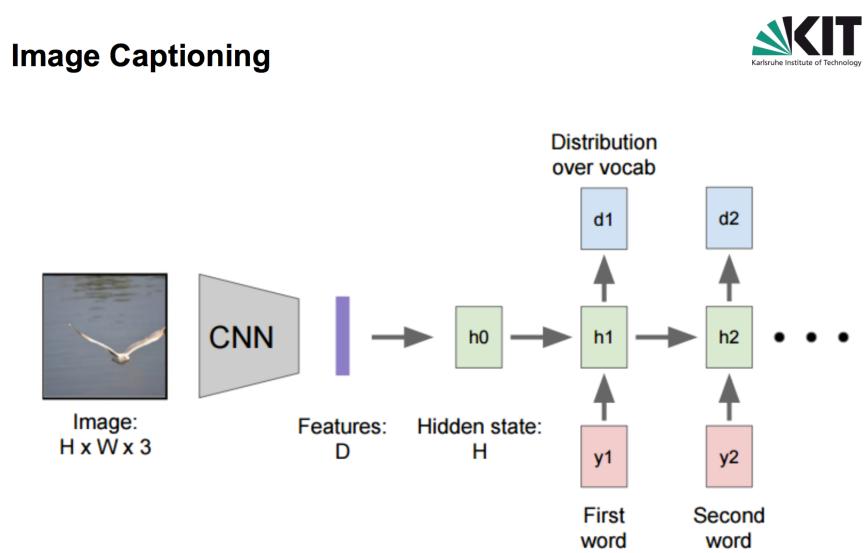
$$h_j^l = \sigma\left(\sum_{k=1}^K h_k^{l-1} w_{kj}^l\right)$$
 - Pooling Layer (Bsp. max pooling): $h_j^l = \max_{\bar{a} \in K(a), \bar{b} \in K(b)} h_{\bar{a}\bar{b}}^{l-1}$
- Backward Pass
 - Conv Layers
 - * Shared weights w_{kj} : $o_j^l = \sum_{k=1}^K h_k^{l-1} w_{kj}^l \Rightarrow \frac{\partial o_j^l}{\partial w_{kj}} = h_j^{l-1}$
 - * Ableitung des Fehlers $\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial h_j^l} \frac{\partial h_j^l}{\partial o_j^l} \frac{\partial o_j^l}{\partial w_j} = \delta_j^l h_j^{l-1}$
 - * Fehlerwert: $\delta_j^l = \sum_{k=1}^K \delta_j^{l-1} w_{kj}$
 - Pooling Layers
 - * $\delta_j^l = \begin{cases} 1, & \text{if } h_j^{l-1} = \max(h^{l-1}) \\ 0, & \text{sonst} \end{cases}$

17.6 IMAGE CAPTIONING

17.6.1 IMAGE CAPTIONING MIT RNNs

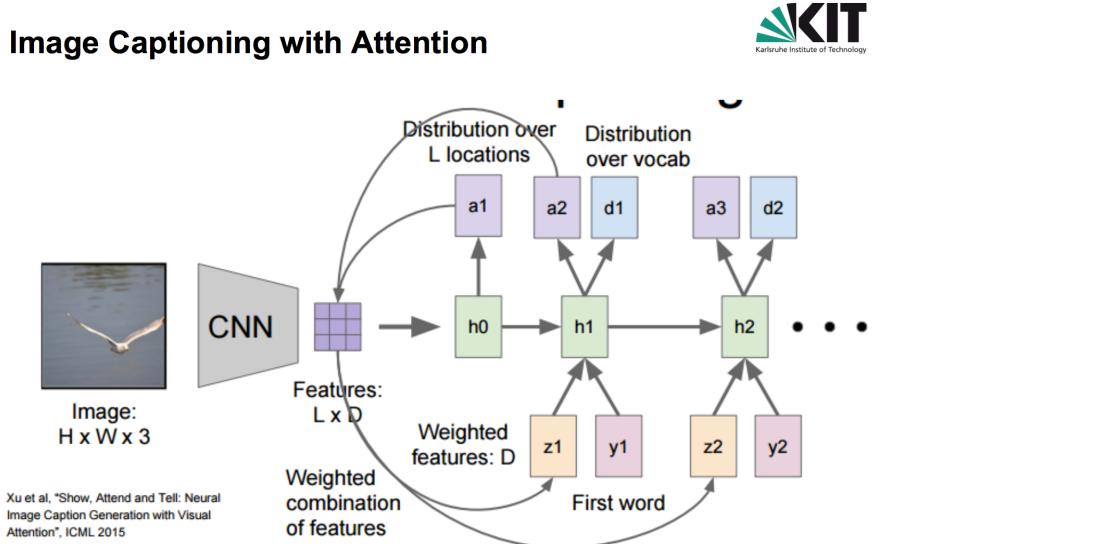


17.6.2 IMAGE CAPTIONING MIT CNNs



17.7 ATTENTION AND IMAGE CAPTIONING

Attention: Bildbereich soll verändert werden nachdem Caption generiert wurde (gleich wie MT)



18 ERKENNUNG VON HANDSCHRIFT

Nicht in Summary-Folien erwähnt - daher nur sehr oberflächlich abgedeckt

- Problem: viele verschiedene Zeichen in vielen verschiedenen Sprachen
- Obwohl Handschrift langsamer ist als digital, wird sie oft verwendet (Metings, Präsentationen, Militär, Briefe, ...)
- Unterschrift/Schreibstil als biometrisches Merkmal
- Erkennung: Handschrift -> Ascii
- Anwendung: Sortieren von Briefen (Post), Schreiben auf Tablet
- On-Line (digitale Eingabe mit Tablet und Stift) vs Off-Line (keine zeitabhängige Funktion aufstellbar)
- Wholistic (Erkennung auf Wortbasis) vs analytische (kleinere Einheiten z.B. Buchstaben) Ansätze

- Umsetzung mit 2D-TDNN (laut Folien) -> Realität

Method	Training Set	Error Rate (%)
THU [6]	CASIA-HWDB 1.0-1.1, 2.0-2.2	7.44
HIT [7]	CASIA-HWDB 1.0-1.1	7.38
Liu et al. [5]	CASIA-HWDB 1.0-1.1	7.28
Our ICDAR'13 [12]	CASIA-HWDB 1.1	5.23 ¹
MCDNN [22]	CASIA-HWDB 1.1	5.53
MCDNNs Voting [22]	CASIA-HWDB 1.1	4.35 ²
R-CNN	CASIA-HWDB 1.1	5.32±0.09
R-CNNs Voting	CASIA-HWDB 1.1	4.45 ³
ATR-CNN	CASIA-HWDB 1.1	4.96±0.08
ATR-CNNs Voting	CASIA-HWDB 1.1	3.94⁴
Human [12]	-	3.87