

Programmieren mit Neuronalen Netzen

Michael Gabler

13. Juli 2019

Inhaltsverzeichnis

1 Grundlagen	2
1.1 Training	2
1.2 Netzarchitektur	4
2 Layer	5
2.1 Fully Connected / Dense	5
2.2 Aktivierungsfunktion	5
2.3 Softmax	5
2.4 Dropout	6
2.5 Convolutional	6
2.6 Pooling	7
2.7 Vanilla RNN	8
2.8 Gru	9
2.9 LSTM	10
2.10 Highway Layer	11
3 Loss-Funktionen	11
3.1 Cross Entropy	11
3.2 Mean Squared Error	11
4 Bildverarbeitung	12
4.1 Klassifizierung	12
4.2 Segmentierung	12
4.3 Objekterkennung	13
5 Sequence2Sequence	13

1 Grundlagen

Neuronales Netzwerk ist Funktion, die auf Eingabedaten angewendet wird.

Optimierung durch Minimierung der Loss-Funktion

Loss-Funktion Maß, wie gut das Netzwerk Vorhersagen trifft. Berechnet sich aus Vorhersage und tatsächlichen Werten (Ground Truth).

- Euklidischer Loss, Mean-Squared-Error: $l_2 = \frac{1}{2N} \sum_i (f_\theta(x_i) - t_i)^2$
- Negative-Log-Likelihood, Cross-Entropy: $NLL = -\frac{1}{|D|} \sum_i \log[f_\theta(x_i)|_{t_i}]$

Konfusionsmatrix welche Klassen werden wie oft mit welcher Klasse verwechselt?

Linearisiertes Speichern mehrdimensionaler Objekte

Row-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & \cancel{a_{22}} & a_{23} \\ a_{31} & a_{32} & \cancel{a_{33}} \end{bmatrix}$$

Column-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & \cancel{a_{22}} & a_{23} \\ \cancel{a_{31}} & a_{32} & a_{33} \end{bmatrix}$$

1.1 Training

Daten werden aufgeteilt in Train/Validierung/Test (z.B. 60/20/20)

Datenaugmentierung Generieren zusätzlicher Daten (z.B. bei Bildern) durch Spiegelung, Rotation, Skalierung, Anpassung Helligkeit und Farbe, etc.

Epoche Verarbeitung aller Trainingsdaten

Iteration Verarbeitung eines Batches

Batch Mehrere Trainingsbeispiele werden gerechnet bevor Gewichte einmal geupdated werden (z.B. 10 Beispiele pro Batch)

Learning Rate Faktor η , wie stark das Netzwerk durch die Deltas verändert werden soll (d.h. wie schnell es lernt bzw. seine Meinung ändert). Wird beim Update der Gewichte verwendet. **Evaluation auf Validierungsdaten** zur Anpassung der Hyperparameter (Learning-Rate, Netzstruktur, ...)

Evaluation auf Testdaten einmalig, um Genauigkeit des trainierten Netzes zu ermitteln

Forward-Pass Berechnen des Outputs des Netzwerkes für bestimmte Eingabedaten (z.B. ein Batch)

Backward-Pass Bilden der partiellen Ableitung für jeden Input in jedem Layer und Speichern der Werte als Deltas

Backpropagation am Beispiel

$$L = \frac{1}{2} \sum_{i=1}^2 (y_i - t_i)^2$$

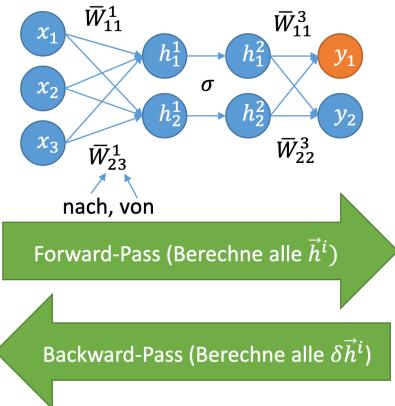
- Gesucht: Fehler an Knoten y_1 :

$$\delta y_1 \stackrel{\text{def}}{=} \frac{\partial L}{\partial y_1} = y_1 - t_1$$

- analog, bzw. allgemein

$$\delta y_i \stackrel{\text{def}}{=} \frac{\partial L}{\partial y_i} = y_i - t_i$$

Entspricht dem „Fehler“ des Netzwerks



02.05.2019

Programmieren mit Neuronalen Netzen

29

Berechnung der Gewicht-Deltas Bilden der partiellen Ableitung für jedes Gewicht jedes Layers und Speichern der Werte als Deltas. Zur Berechnung sind die Deltas der Outputs (siehe Backward-Pass) erforderlich. Für Batches werden die Deltas der Gewichte aufsummiert und nach dem Batch geupdated.

Backpropagation am Beispiel

$$\delta y_i \stackrel{\text{def}}{=} \frac{\partial L}{\partial y_i}, \quad \vec{y} = \vec{h}^2 \cdot \vec{W}^3 + \vec{b}^3$$

- Gesucht: „Fehler“ des Gewichts \bar{W}_{11}^3 :

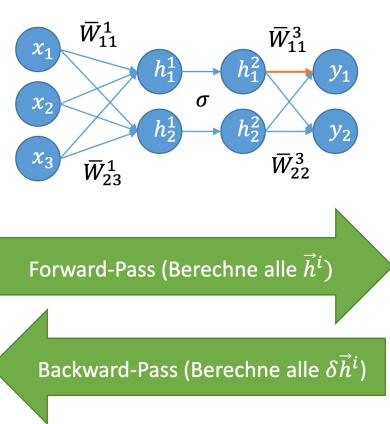
$$\delta \bar{W}_{11}^3 \stackrel{\text{def}}{=} \frac{\partial L}{\partial y_1} \frac{\partial y_1}{\partial \bar{W}_{11}^3}$$

- analog, bzw. allgemein

$$\delta \bar{W}_{ij}^3 \stackrel{\text{def}}{=} \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial \bar{W}_{ij}^3}$$

- Einsetzen und Ableiten:

$$\delta \bar{W}_{ij}^3 = \delta y_i \cdot h_j^2$$



02.05.2019

Programmieren mit Neuronalen Netzen

30

Update der Gewichte Die Gewichte werden nun geupdated, in dem die Deltas der Gewichte mit den ursprünglichen Gewichten verrechnet werden. Dafür gibt es verschiedene Optimierungsverfahren:

- Gradient Descent: für Gewicht $w'_{ij} = w_{ij} - \eta \cdot \delta w_{ij}$

- Adam-Optimizer: robuster gegenüber schlecht gewählter Learning-Rate
- Adagrad
- RMSProp

Regularisierung Methoden um Overfitting vorzubeugen

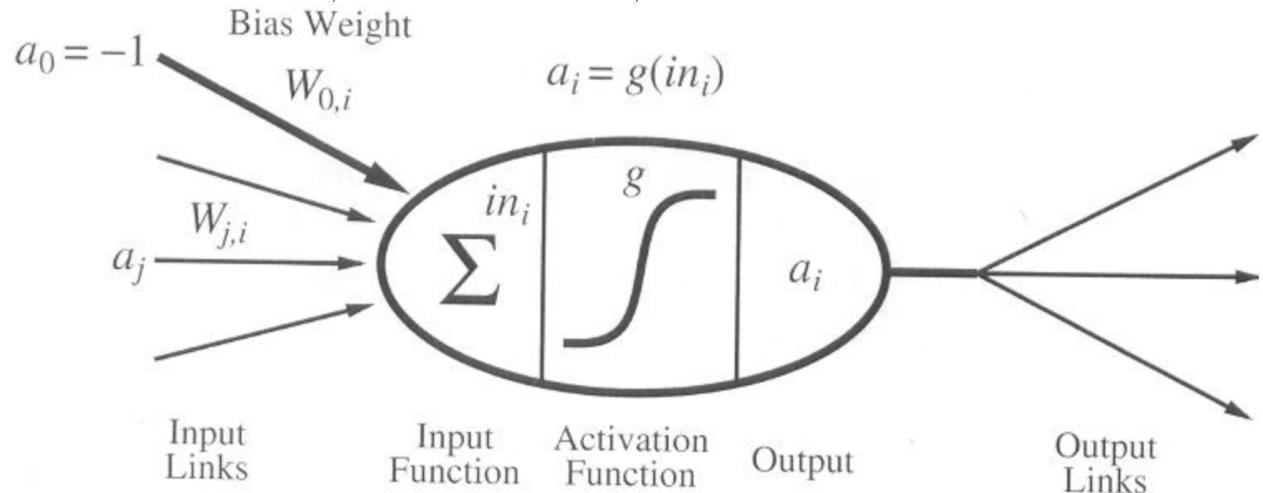
- Rauschen auf Eingabedaten, Gewichten, Ausgabe (Zufallswerte hinzufügen)
- Datenaugmentierung
- Early-Stopping: Laufende Validierung auf Validierungsset, Trainingsabbruch wenn Accuracy abnimmt
- Dropout (siehe Layer)

Pre-Training Verwende Startgewichte, die bereits auf ähnlichen Daten trainiert wurden → besser als zufällige Initialisierung.

Transferlearning Verwende vortrainiertes Netz und trainiere nur einzelne Schichten (z.B. letzte Schicht für Klassifikation) neu.

1.2 Netzarchitektur

Perceptron (= künstliches Neuron) stellt lineare Trennung (binäre Klassifikation) dar. Kann Funktionen AND, OR und NOT lernen, nicht aber XOR.



mit Inputs a_j gewichtet mit W_{ji} (ergeben zusammen die Gewichtsmatrix W), addiert mit Bias b , Outputs a_i und der Aktivierungsfunktion g mit der der Output berechnet wird.

$$Ausgabe_{Schicht(x)} = g(Eingabe_{Schicht(x)} \cdot W + b) = Eingabe_{Schicht(x+1)}$$

Aktivierungsfunktion muss bei Multi-Layer-Perceptronen (MLP) nicht-linear sein, da sonst nicht mehr Information gespeichert werden kann.

$$(x \cdot W + b) \cdot V + a = x \cdot W \cdot V + b \cdot V + a = x \cdot W' + b'$$

2 Layer

Mögliche Schichten aus denen ein neuronales Netz bestehen kann.

2.1 Fully Connected / Dense

Voll verbundene Schicht, d.h. jeder Input landet in jedem Neuron. Wird z.B. als letzter Layer zur Klassifikation verwendet, um Vektor mit Logits für Klassen auszugeben.

Parameter $X \in \mathbb{R}^{1 \times a}$: Eingabe, $W \in \mathbb{R}^{a \times n}$: Gewichtsmatrix, $b \in \mathbb{R}^{1 \times n}$: Bias, n : Anzahl der Neuronen, $Y \in \mathbb{R}^{1 \times n}$: Ausgabe

Forward-Pass

$$Y = X \cdot W + b$$

Backward-Pass

$$\delta X = \delta Y \cdot W^T$$

Calculate Delta Weights

$$\begin{aligned}\frac{\delta L}{\delta W} &= X^T \cdot \delta Y \\ \frac{\delta L}{\delta b} &= \delta Y\end{aligned}$$

2.2 Aktivierungsfunktion

Elementweise Anwendung

Funktion	Forward	Backward
tanh	$Y = \tanh(X)$	$\delta X = (1 - \tanh^2(X)) \odot \delta Y$
Sigmoid	$Y = \sigma(X) = \frac{1}{1+e^{-X}}$	$\delta X = (\sigma(X) \cdot (1 - \sigma(X))) \odot \delta Y$
ReLU	$Y = \text{ReLU}(X)$	$\delta X = \text{ReLU}'(X) \odot Y$

ReLU: $\text{ReLU}(x) = x > 0 ? x : 0$, $\text{ReLU}'(x) = x > 0 ? 1 : 0$

2.3 Softmax

Normalisiert eine Menge von Werten, sodass deren Summe 1 ergibt. Wird zur Berechnung der Wahrscheinlichkeitsverteilung bei Klassifikation verwendet.

Forward-Pass

$$Y = \text{softmax}(X) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

Backward-Pass

$$\delta X = \delta Y \cdot \begin{bmatrix} \frac{\delta x_1}{\delta y_1} & \dots & \frac{\delta x_n}{\delta y_1} \\ \dots & \ddots & \dots \\ \frac{\delta x_1}{\delta y_n} & \dots & \frac{\delta x_n}{\delta y_n} \end{bmatrix}$$

2.4 Dropout

Für Regularisierung verwendet. Ein Teil der Gewichte wird zufällig je Durchlauf auf 0 gesetzt (deaktiviert). Wird nicht trainiert, sollten alle Gewichte weitergegeben, aber durch die Dropout-Rate geteilt werden, da sonst eine zu hohe Aktivierung der nächsten Schicht statt findet.

Parameter d : Dropout-Rate gibt den prozentualen Anteil der zu deaktivierenden Gewichte an.

2.5 Convolutional

Extrahiert Features aus einer Matrix. Wird oft in der Bildklassifizierung verwendet oder NLP zur Satzklassifikation (Kernel Größe der Embeddings \times Anzahl betrachteter Wörter).

Parameter

$X \in \mathbb{R}^{h \times w \times d}$: Eingabe

(fh, fw) : Filtergröße

fn : Anzahl Filter

$F \in \mathbb{R}^{fh \times fw \times fd \times fn}$: Filtertensor

$b \in \mathbb{R}^{fn}$: Bias (ein Wert pro Ausgabechannel, wird auf jeden Wert des Channels addiert)

$Y \in \mathbb{R}^{(h-fh+1) \times (w-fw+1) \times fn}$: Ausgabe

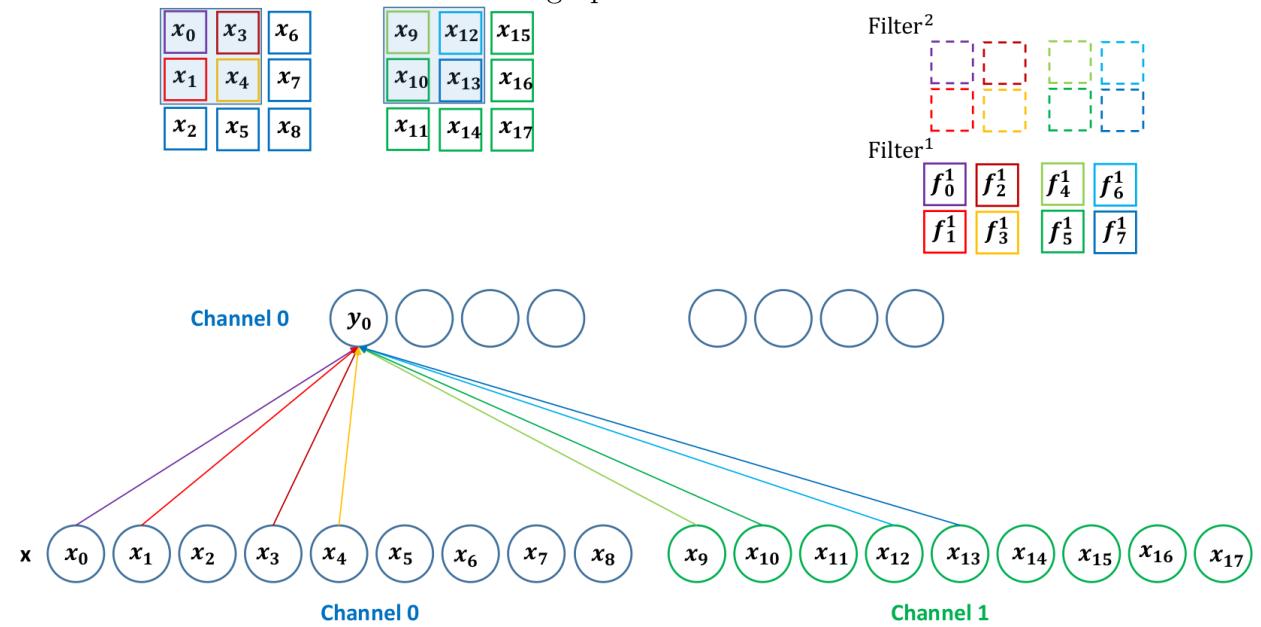
Optionale Parameter Stride: wie weit der Filter verschoben wird (default = 1), Dilation: zusätzliche 0en im Filter (Filter über nicht benachbarte Elemente)

Padding Covolution verkleinert die Daten. Um dies zu verhindern, kann die Eingabe gepadded werden (Hinzufügen von 0en am Rand der Eingabe).

Half-Padding: Ausgabegröße = Eingabegröße, $ph = \lfloor \frac{fh}{2} \rfloor$, $pw = \lfloor \frac{fw}{2} \rfloor$

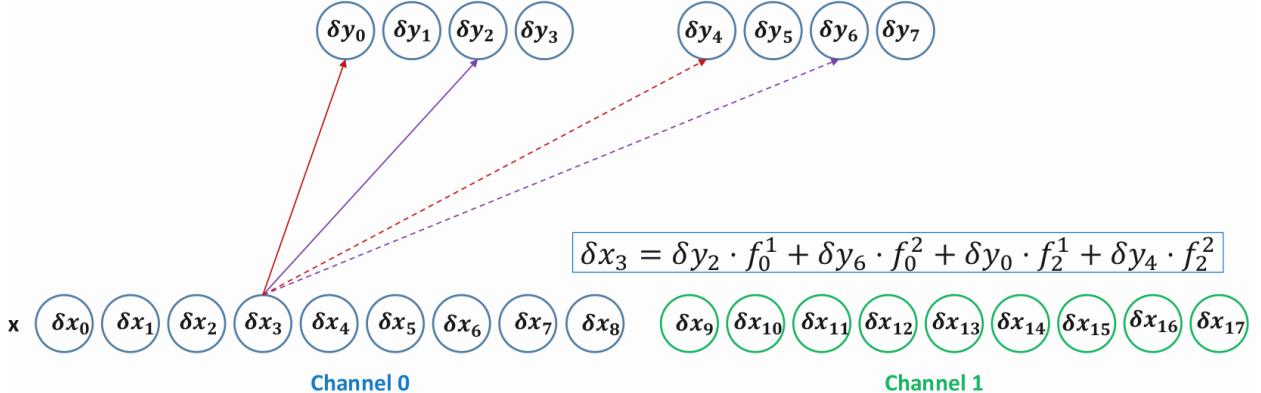
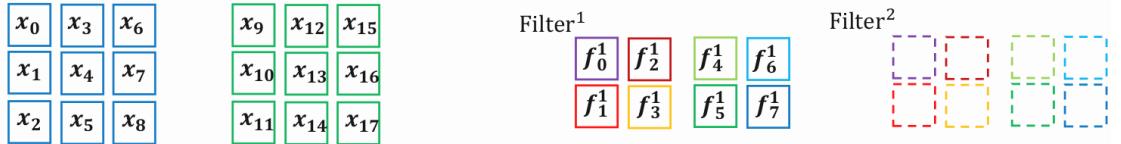
Full-Padding (Full-Convolution): Ausgabegröße > Eingabegröße, $ph = fh - 1$, $pw = fw - 1$

Forward-Pass $*$ bezeichnet die Faltungsoperation



$$Y = X * F + b$$

Backward-Pass $*_F$ bezeichnet die Full-Convolution



$$\delta X = \delta Y *_F rot_{h,w}^{180}(trans(F))$$

Calculate Delta Weights $*_{ch}$ bezeichnet die Channel-Wise-Convolution, f ist ein Element des Filtertensors

$$\frac{\delta L}{\delta f} = X *_{ch} \delta Y$$

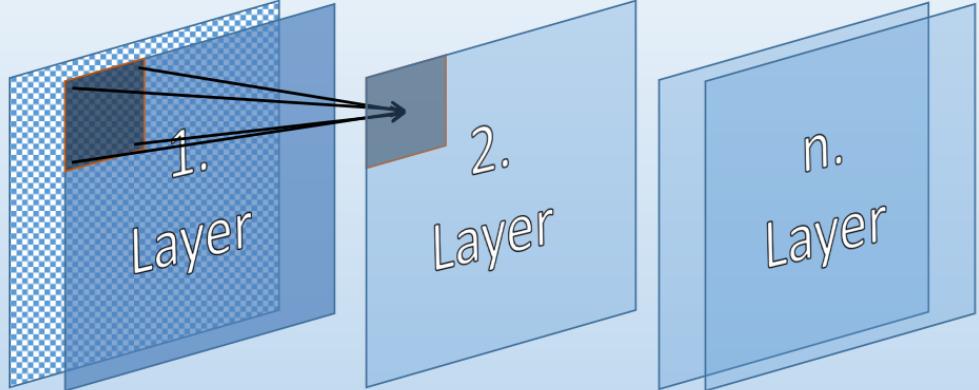
$$\frac{\delta L}{\delta b_f} = \sum_i \delta y_{i,f}$$

2.6 Pooling

Reduziert die Werte innerhalb eines Filters auf einen Wert (z.B. Maximum oder Durchschnitt). Filter wird ohne Überlappung (mit Stride) verschoben.

Forward-Pass

Max-Pooling



$$\max \circ \begin{bmatrix} 4 & 1 & 1 & 3 & 3 & 4 \\ 8 & 6 & 0 & 5 & 2 & 2 \\ 0 & 6 & 9 & 5 & 1 & 8 \\ 4 & 8 & 5 & 7 & 4 & 2 \\ 9 & 8 & 5 & 9 & 3 & 0 \\ 3 & 6 & 7 & 4 & 7 & 5 \end{bmatrix} = \begin{bmatrix} 8 & 5 & 4 \\ 8 & 9 & 8 \\ 9 & 9 & 7 \end{bmatrix}$$

Backward-Pass Nur die Elemente, die im Forward-Pass an der Berechnung des Outputs beteiligt waren, bekommen anteilig oder ganz das Delta des jeweiligen Outputs.

$$\delta x_i = (x_i == \max)?\delta y : 0$$

2.7 Vanilla RNN

Recurrent Neuronal Networks (RNNs) werden verwendet, um Daten unterschiedlicher Länge zu verarbeiten. Dabei wird ein Hidden State h in jedem Schritt um Informationen der Eingabe ergänzt.

Parameter

$[x_0, \dots, x_n] = X \in \mathbb{R}^{n \times m}$: Eingabevektoren mit jeweils Länge m

g : Zwischenvariable nach der Addition vor \tanh

h : Hidden State (h_{-1} kann mit Nullen initialisiert werden)

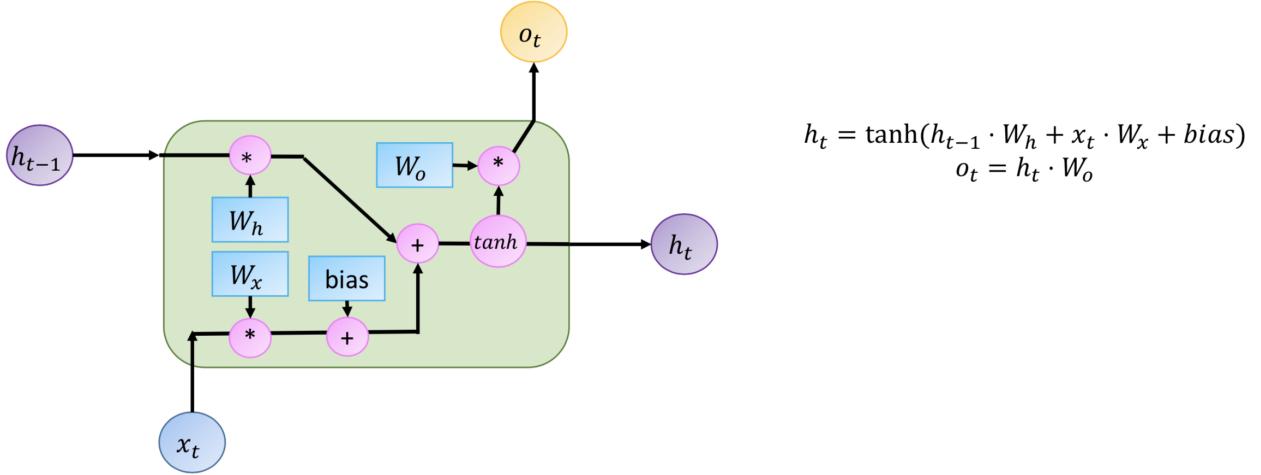
W_x, W_h, W_o : Gewichtsmatrizen

b : Bias

$[o_0, \dots, o_n] = O \in \mathbb{R}^{n \times l}$: Ausgabevektoren mit jeweils Länge l

Funktionsweise

The Vanilla Recurrent Cell (one possibility)



Forward-Pass

$$h_t = \tanh(h_{t-1} \cdot W_h + x_t \cdot W_x + b)$$

$$o_t = h_t \cdot W_o$$

Backward-Pass

$$\delta h_t = \delta o_t \cdot W_o^T + \delta g_{t+1} \cdot W_h^T$$

$$\delta g_t = \delta h_t \cdot (1 - \tanh^2(g_t)) = \delta h_t \cdot (1 - h_t^2)$$

$$\delta x_t = \delta g_t \cdot W_x^T$$

Calculate Delta Weights

$$\delta W_h = \sum_t h_{t-1}^T \cdot \delta g_t$$

$$\delta W_x = \sum_t x_t^T \cdot \delta g_t$$

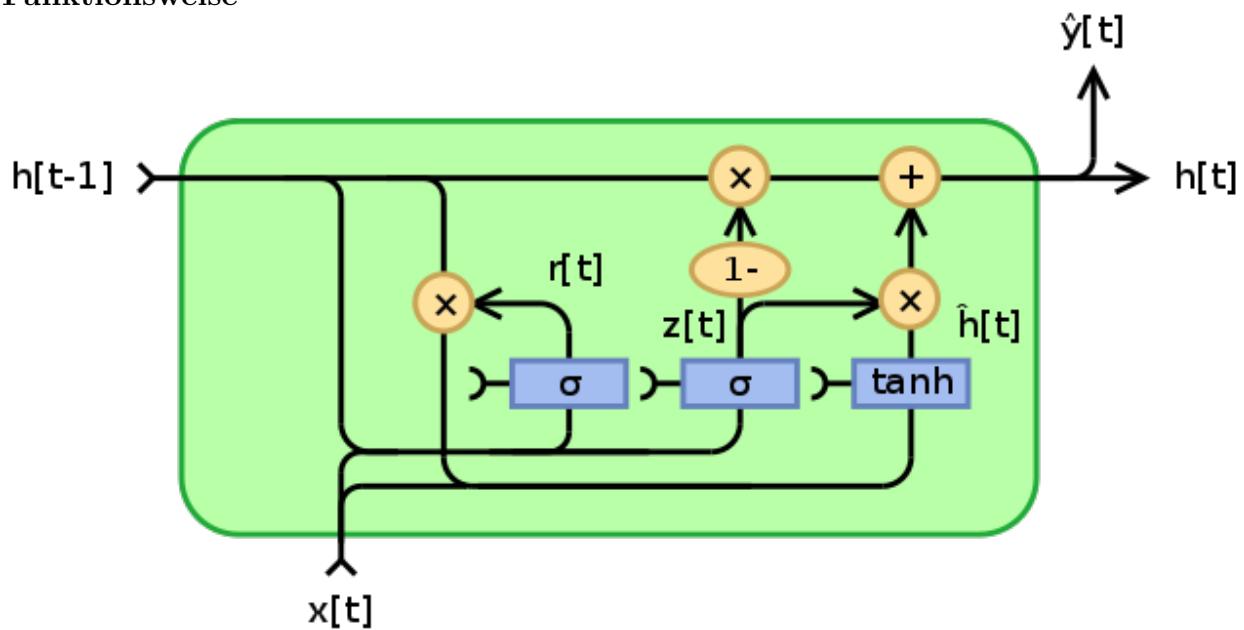
$$\delta W_o = \sum_t h_t^T \cdot \delta o_t$$

$$\delta b = \sum_t \delta g_t$$

2.8 Gru

Verwendet zwei Gates (wie g im Vanilla RNN), um zu bestimmen, was aus dem alten Internal State h übernommen wird und was von den aktuell verarbeiteten Daten hinzugefügt wird.

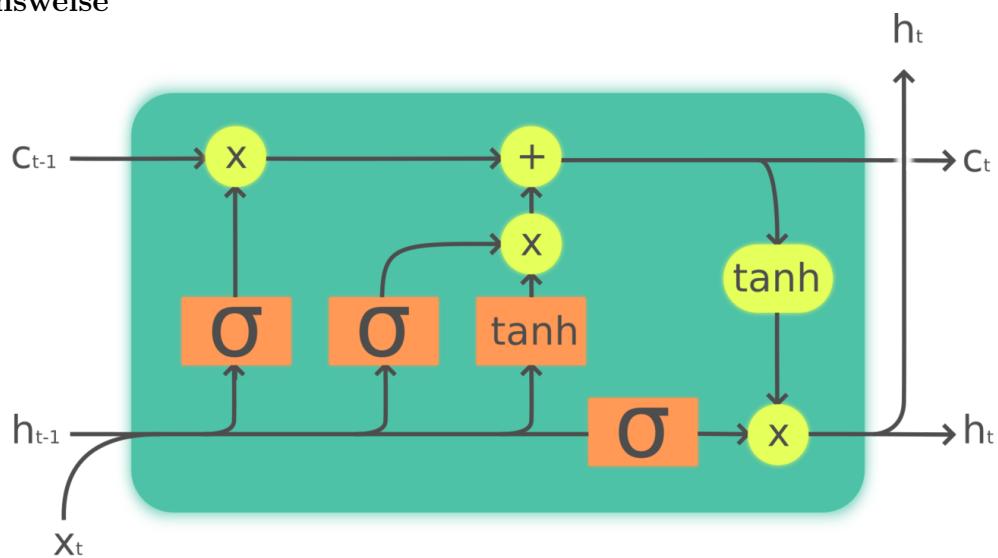
Funktionsweise



2.9 LSTM

Verwendet drei Gates (Forget, Input, Output) und zusätzlichen State c

Funktionsweise



Legend:

Layer	Pointwise op	Copy

2.10 Highway Layer

Verwendet ein Gate, das bestimmt, ob der Layer wie ein Fully Connected Layer funktioniert oder lediglich den Input weitergibt. Wird verwendet bei sehr tiefen Netzwerken (> 100 Layer), um Daten über lange Zeit/viele Layer hinweg nutzbar zu machen.

Forward-Pass

$$Y = (X \cdot W + b) \cdot T(x) + x \cdot (1 - T(x))$$

$$T(x) = \sigma(x \cdot W_T + b_T)$$

3 Loss-Funktionen

X : Eingabe der Loss-Funktion bzw. Ausgabe des Netzes (Prediction)

T : Erwartetes Ergebnis (Ground Truth)

3.1 Cross Entropy

Negative-Log-Likelihood, Cross-Entropy

Forward-Pass

$$L = - \sum_i t_i \cdot \log(x_i)$$

Backward-Pass

$$\frac{\delta L}{\delta x_i} = -\frac{t_i}{x_i}$$

3.2 Mean Squared Error

Euklidischer Loss, Mean-Squared-Error, l_2 -Loss

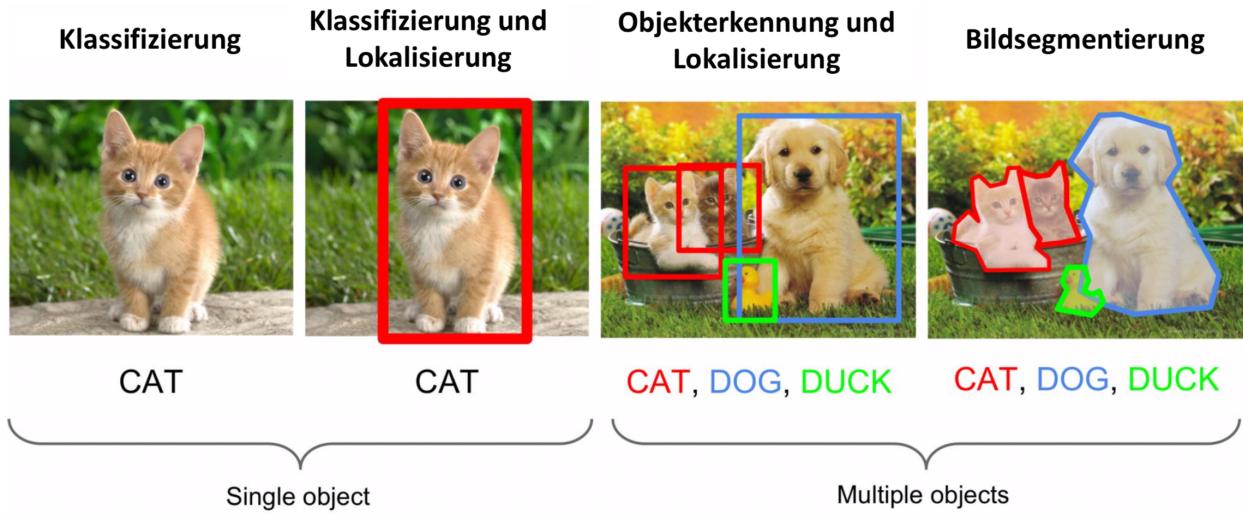
Forward-Pass

$$L = \sum_i \frac{1}{2} (x_i - t_i)^2$$

Backward-Pass

$$\frac{\delta L}{\delta x_i} = t_i - x_i$$

4 Bildverarbeitung



4.1 Klassifizierung

In der Regel neuronales Netz mit Convolution und Fully Connected Layern, das das gesamte Bild als Eingabe bekommt.

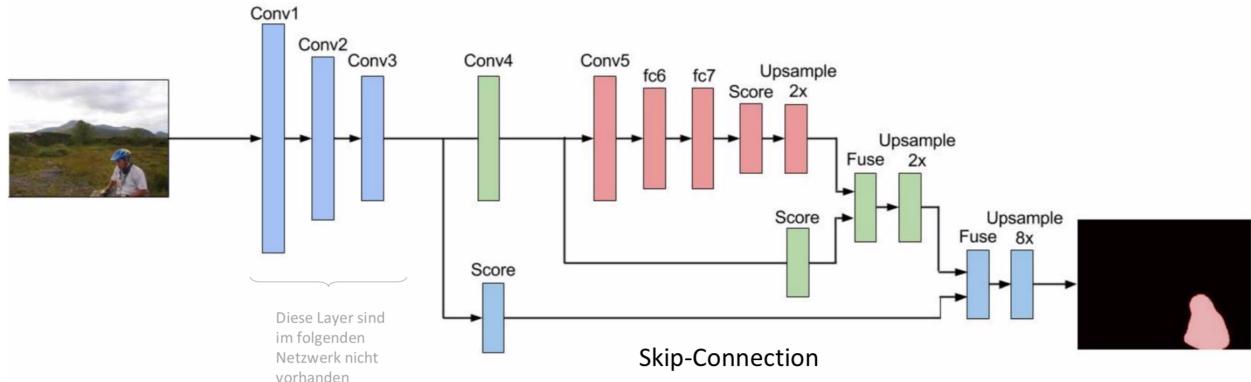
4.2 Segmentierung

Zuweisung einer Klasse für jeden Pixel eines Bildes.

Sliding Window Klassifiziere kleinere Bildausschnitte. Probleme: Keine Flächen (benachbarte Vorhersagen haben keinen direkten Einfluss), Auflösung wird niedriger durch CNN, kleiner Stride führt zu Mehrfachberechnung

Fully Convolution Network CNN als Encoder, das Größe verkleinert. Anschließend hochskalieren (Decoder) → schnell aber ungenau

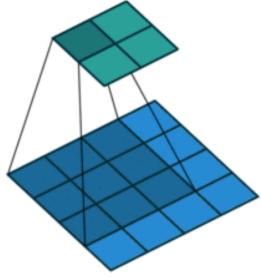
Skip Connections Addieren/Konkatinieren mit Werten aus vorherigen Layern



Transposed Convolution Anwenden der transponierten Convolution zur Bildvergrößerung (wie im Backward-Pass der regulären Convolution)

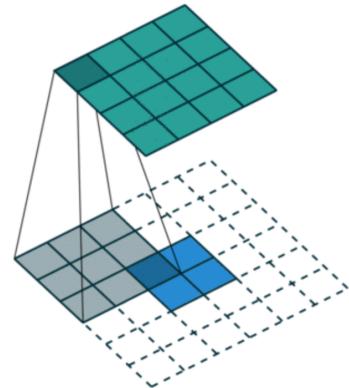
Normale Convolution

- 3x3 Filter, Stride 1
- 4x4 Input
- 2x2 Output



Transponierte Convolution

- 3x3 Filter, Stride 1
- 2x2 Input
- 4x4 Output



Unpooling Umkehren des Poolings analog zum Backward-Pass (z.B. bei Max-Pooling bekommt das ursprüngliche Maximum alles). Kann nur mit einer Pooling-Schicht verwendet werden, die umgekehrt werden soll → Alternative zum Upsampling

4.3 Objekterkennung

Single Shot Detector Definiere Standard Bounding Boxes (BB) im Bild. Netzwerk soll Klasse und Abweichung von einer Standard BB angeben.

$$L = \frac{1}{N}(L_{conf} + \alpha L_{loc})$$

mit L_{conf} als Cross-Entropy über die Klassen und $L_{loc} = -\sum_n^N smooth_{l_1}(l_n - g_n)$

Multi Object Detection mehrere BBs (ca. 4) pro Pixel. BB gilt als gematched, wenn Flächeninhalt der Vorhersage mit BB zu z.B. 50 Prozent übereinstimmt.

Non-Maximum Suppression Da oft mehrere BBs matchen, suche BB mit höchster Konfidenz und lösche alle BBs, die z.B. mind. 50 Prozent Overlapping haben. Wiederhole, bis keine BBs mehr gelöscht wurden.

5 Sequence2Sequence

Abbildung von Sequenzen auf Sequenzen, z.B. bei OCR (Eingabe: Pixelspalten, Ausgabe: Buchstabenklassifikation) oder Spracherkennung. Kann z.B. mit LSTMs umgesetzt werden.

Greedy-Decoder Ermittelt Klasse durch Maximum auf Ausgabe → Entfernen doppelter Zeichen → Blanks entfernen

Labellings Beschreibt die Abbilung der Netzausgabe auf sinnvolle Sequenz.

$$B : L'^T \rightarrow L^{\leq T}$$

$$B(a - ab-) = B(-aa --aabb) = aab$$

$$B^{-1}(aab) = a - ab-, ..., aa - ab, ...$$

Summiere die Wahrscheinlichkeiten aller möglichen Pfade, die zum gewünschten Label führen.

CTC-Loss (Connectionist Temporal Classification)

Wird angewendet, wenn die Eingabe eine Zeitsequenz der Länge T ist und als Ausgabe eine unalignierte Labelsequenz der Länge $\leq T$ vorliegt

α	0	1	2	3	4	5	6	7	8	β	0	1	2	3	4	5	6	7	8
-	0.9	0.36	0.036	0.029	0.009					-									
a	0.09	0.495	0.684	0.108	0.014	0.002				a									
-		0.036	0.053	0.590	0.209	0.045				-									
f			0.008	0	0	0.419	0.257	0.030		f						0.000			
-				0	0	0	0.168	0.383		-						0.535			
f				0	0	0	0	0.017	0.240	-						0.089	0.594		
-					0	0	0	0	0.005	f						0.267	0.297		
e						0	0	0	0	e						0.000	0.000	0.990	
-							0	0	0	-						0.003	0.003	0.010	
	0	1	2	3	4	5	6	7	8										
P(-)	0.9	0.4	0.1	0.8	0.3	0.2	0.9	0.3	0.01										
P(a)	0.09	0.5	0.8	0.15	0.1	0.1	0	0	0										
P(b)	0	0	0	0	0	0.01	0	0.1	0										
P(c)	0	0	0	0	0	0.09	0	0	0										
P(d)	0	0.09	0.1	0.05	0	0	0	0	0										
P(e)	0.01	0	0	0	0	0.2	0	0	0.99										
P(f)	0	0.01	0	0	0.6	0.4	0.1	0.6	0										

$$P(l|x) = \sum_u \frac{\alpha(u,t)\beta(u,t)}{y_{l(u)}^t}, \quad \forall t$$

$$P(l|x) = \sum_{\pi \in B^{-1}(l)} P(\pi|x)$$

mit l als gewünschte Sequenz und x als Netzausgabe

Loss: Maximiere $P(l|x)$ bzw. minimiere $-\log(P(l|x))$