Neural Networks: Architectures and Applications for NLP

Session 05

Julia Kreutzer

29. November 2016

Institut für Computerlinguistik, Heidelberg

Overview

1. Recap
2. Recurrent Neural Networks
3. LSTM & GRU
4. Ausblick

Recap

Sequenzen in NLP

Buchstabensequenzen – Wortsequenzen – Satzsequenzen – ...

Wie können wir sie geeignet repräsentieren?

- **Reihenfolge**, z.B. für Sentiment Analysis
  Das war nicht schlecht, sondern gut. - Das war nicht gut, sondern schlecht.
- **Kontext**, z.B. für Disambiguierung
  Ich habe eine Maus als Haustier. - Ich habe eine Maus für meinen Computer.
- **Geschichte**, z.B. für Koreferenzauflösung
  Hans hat Hunger. Er hat vergessen zu frühstücken.

Bisher

Sequenzen dargestellen mit:

- **MLP**: Vektoren addieren / mitteln / konkatenieren
  - Konkatenation: resultierender Vektor hat keine feste Länge
  - CBOW: Reihenfolge geht verloren
- **CNN**: Convolution & Pooling
  - Reihenfolge wird nur lokal berücksichtigt

Recurrent Neural Networks

Recurrent Neural Networks (RNN)

Ziel:

- **beliebig lange** Sequenzen repräsentieren
- diese in einem **Vektor fester Länge** repräsentieren
- dabei die **strukturellen Eigenschaften** der Sequenz berücksichtigen (Reihenfolge, Kontext, ...)

RNN Definition

Ein RNN erhält als Eingabe $\mathbf{x}$ als Sequenz aus Vektoren $\mathbf{x}_1, \ldots \mathbf{x}_T$.

Die Ausgabe ist

1. eine Liste von Zustands-Vektoren $\mathbf{s}_1, \ldots \mathbf{s}_T$,
2. eine Liste von Ausgabe-Vektoren $\mathbf{y}_1, \ldots \mathbf{y}_T$.

Die Ausgabe wird dann für Vorhersagen genutzt, z.B.
$p(c = j|\mathbf{x}_t) = \texttt{softmax}(\mathbf{y}_t\mathbf{W} + \mathbf{b})\,[j]$

RNN Bestandteile

$\text{RNN}(s_0, x_{1:t}) = s_{1:t}, y_{1:t}$ mit:

- **Rekursionsfunktion** R zur Berechnung des aktuellen Zustands:

$$s_t = R(s_{t-1}, x_t)$$

- **Ausgabefunktion** O zur Berechnung der aktuellen Ausgabe:

$$y_t = O(s_t)$$

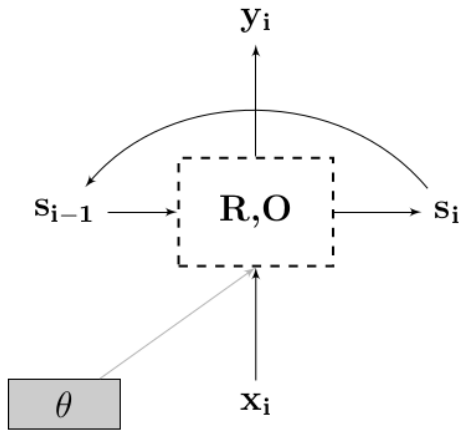


Abbildung 1: Rekursion im RNN, Quelle: [Goldberg, 2015]

Training: Backpropagation through Time

Backpropagation through Time (BPTT) [Werbos, 1990]: Backpropagation mit Auflösen der Rekursion. Ein RNN entspricht dann einem sehr tiefen NN, bei dem dieselben Gewichte mehrmals verwendet werden.

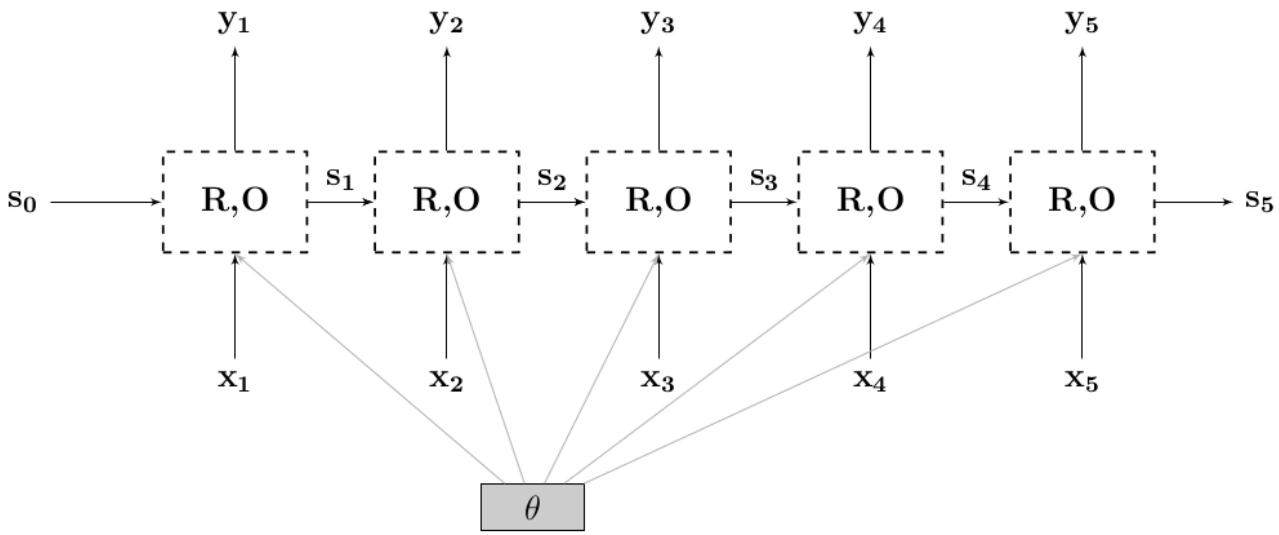
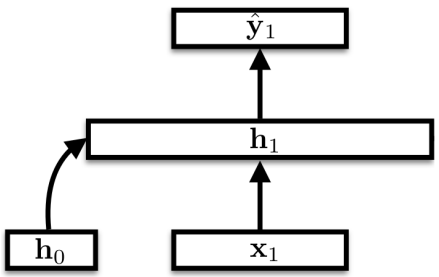
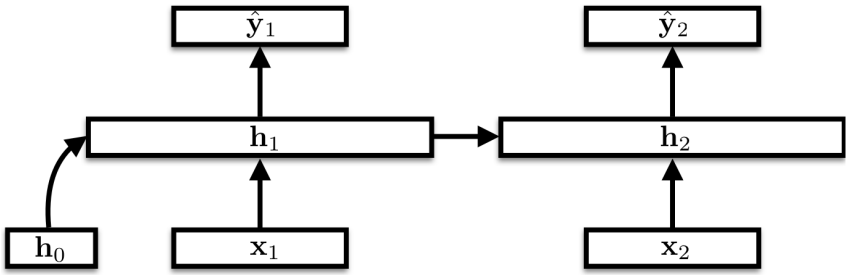


Abbildung 2: Aufgerollte Rekursion, Quelle: [Goldberg, 2015]

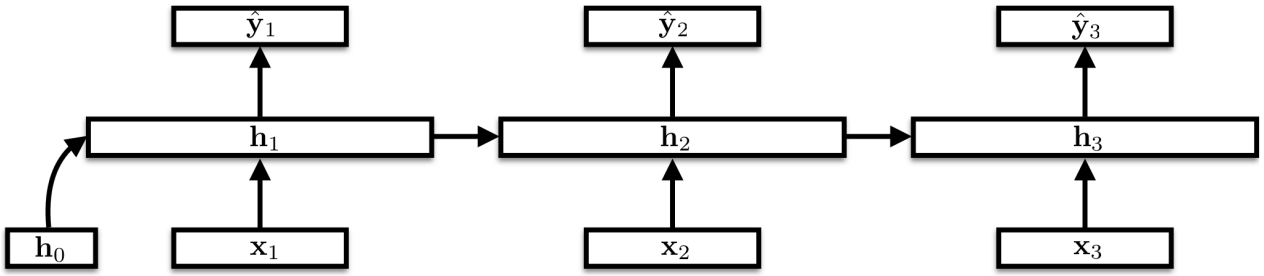
RNN Schritt i



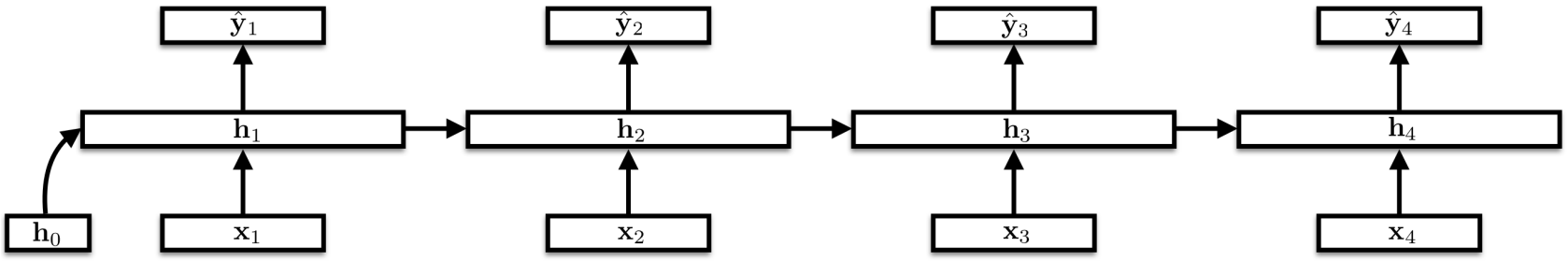
RNN Schritt ii



RNN Schritt iii

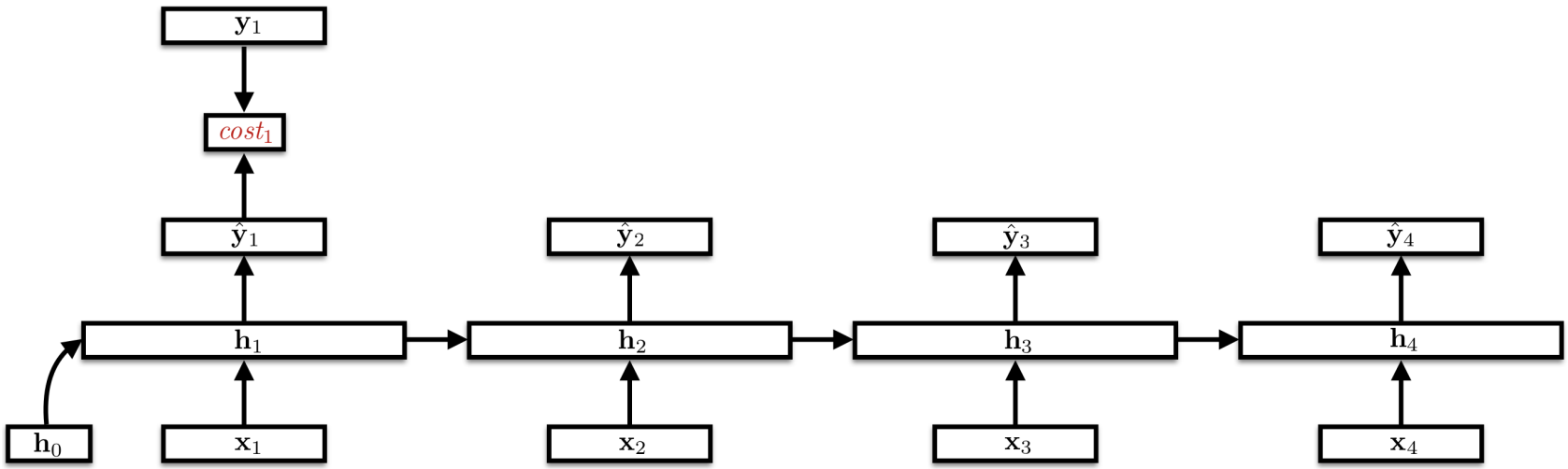


RNN Schritt iv

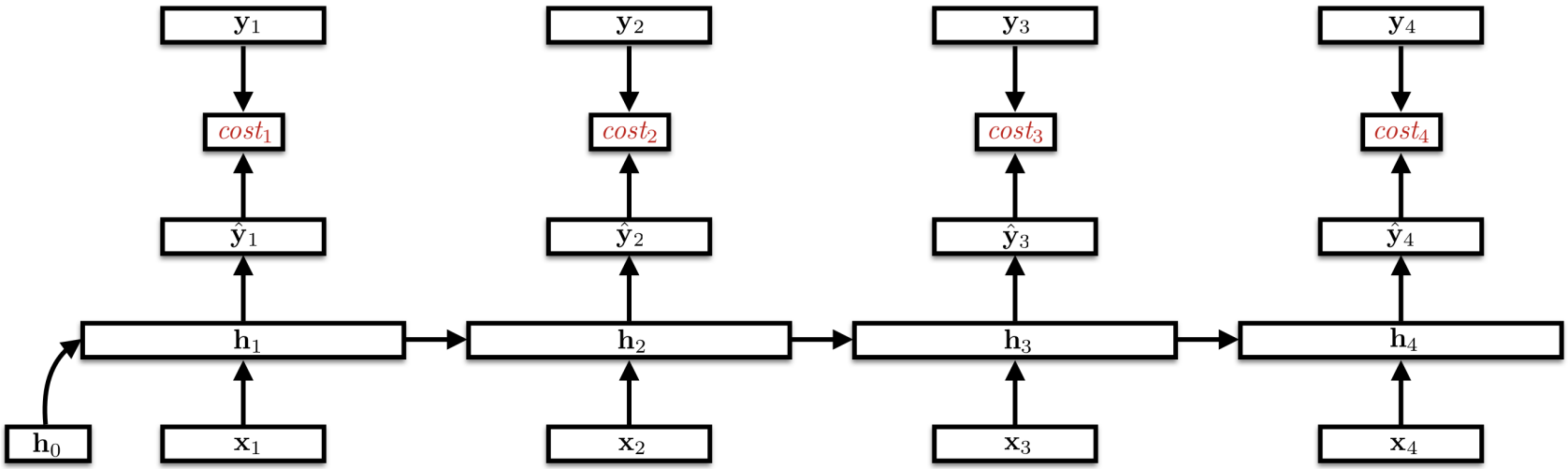


Grafiken von Chris Dyer: “Modeling Sequential Data with Recurrent Networks”, LXMLS 2016

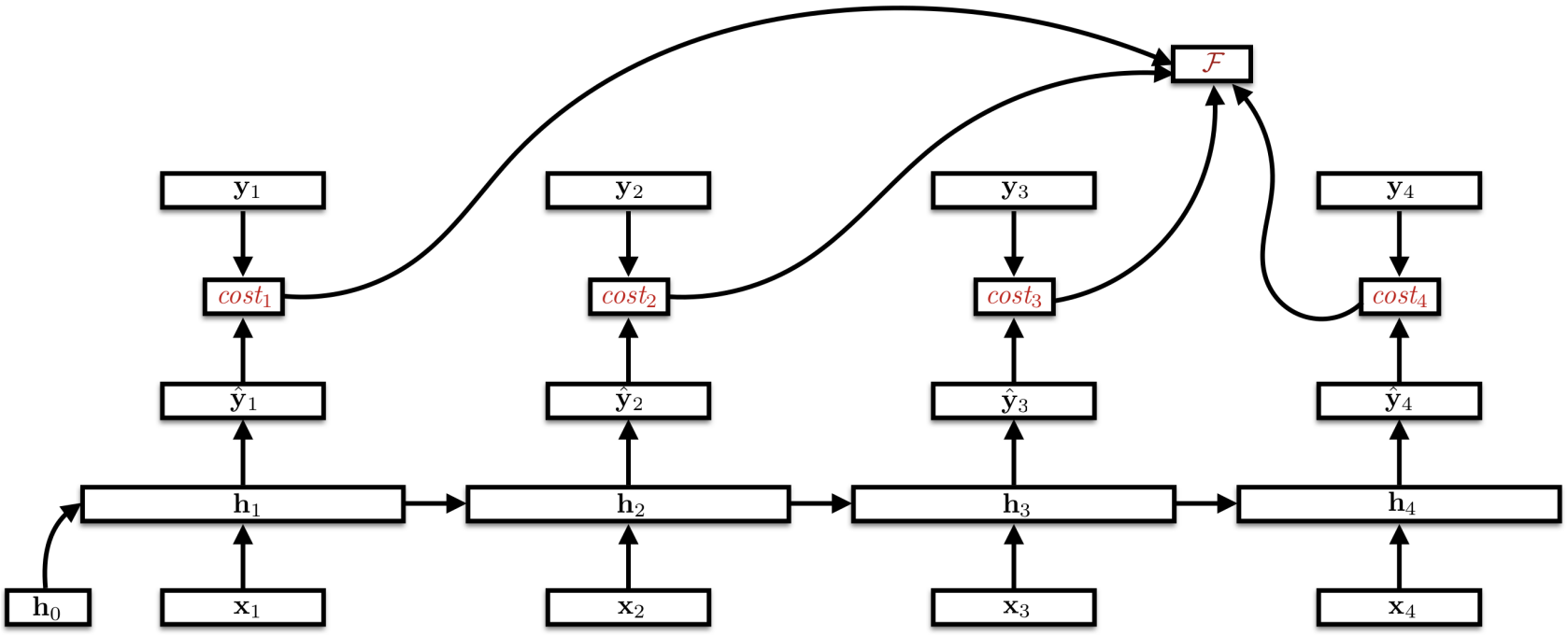
RNN Loss i


$\mathbf{y}_1$
$cost_1$
$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$

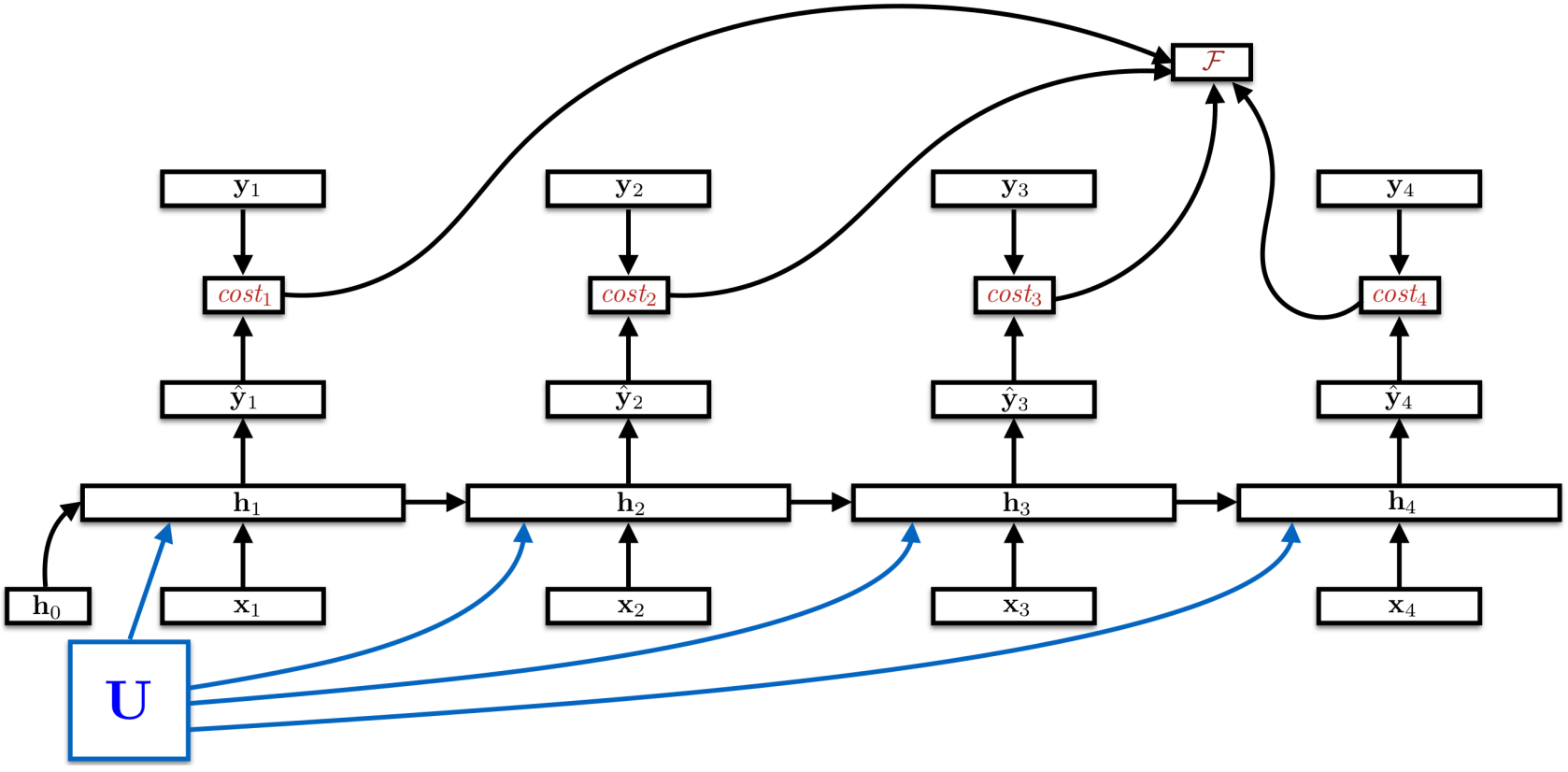
RNN Loss ii


$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\mathbf{y}_4$
$cost_1$
$cost_2$
$cost_3$
$cost_4$
$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$

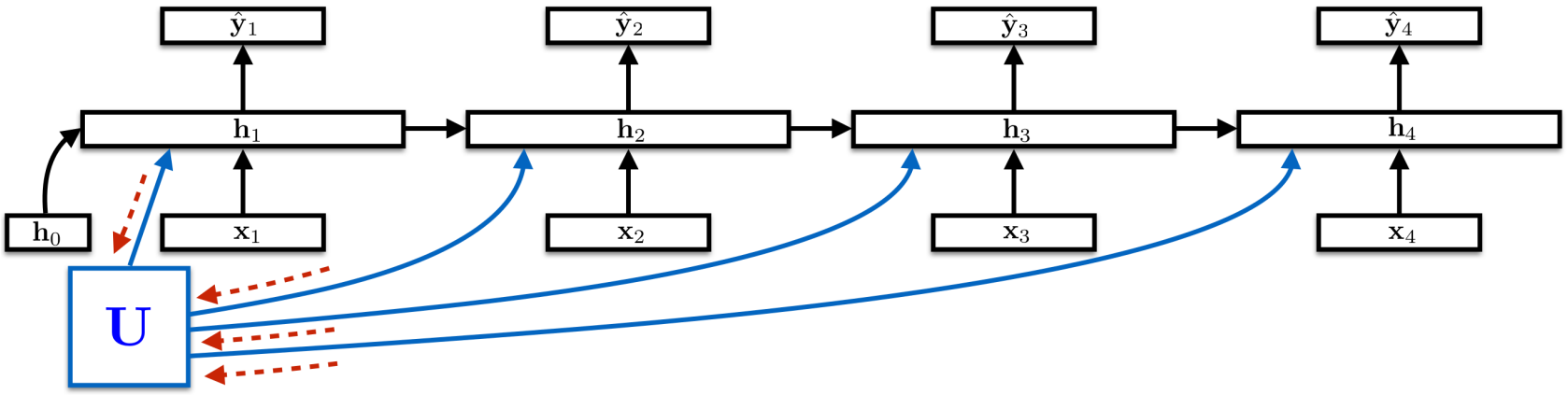
RNN Loss iii


$\mathcal{F}$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\mathbf{y}_4$
$cost_1$
$cost_2$
$cost_3$
$cost_4$
$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$

RNN Loss iv


$\mathcal{F}$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\mathbf{y}_4$
$cost_1$
$cost_2$
$cost_3$
$cost_4$
$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{U}$

RNN Loss v



Backpropagation:

$$\frac{\partial \mathcal{F}}{\partial \mathsf{U}} = \sum_{t=1}^{4} \frac{\partial \mathcal{F}}{\partial \mathsf{h}_t} \frac{\partial \mathsf{h}_t}{\partial \mathsf{U}}$$

Grafiken von Chris Dyer: "Modeling Sequential Data with Recurrent Networks", LXMLS 2016

RNN Supervision

Mehrere Möglichkeiten:

1. **Acceptor**: Supervision für letzte Ausgabe $\mathbf{y}_T$ zur Klassifikation der gesamten Sequenz
   Bsp.: Vorhersage für Sentiment des Satzes

RNN Supervision

Mehrere Möglichkeiten:

1. **Acceptor**: Supervision für letzte Ausgabe $\mathbf{y}_T$ zur Klassifikation der gesamten Sequenz
   Bsp.: Vorhersage für Sentiment des Satzes
2. **Encoder**: auch nur $\mathbf{y}_T$ wird weiterverwendet, hier aber als Enkodierung der Sequenz verstanden, Supervision erst nach Weiterverarbeitung
   Bsp.: Extrahiere Sätze aus Dokument für Zusammenfassung

RNN Supervision

Mehrere Möglichkeiten:

1. **Acceptor**: Supervision für letzte Ausgabe $\mathbf{y}_T$ zur Klassifikation der gesamten Sequenz
   Bsp.: Vorhersage für Sentiment des Satzes
2. **Encoder**: auch nur $\mathbf{y}_T$ wird weiterverwendet, hier aber als Enkodierung der Sequenz verstanden, Supervision erst nach Weiterverarbeitung
   Bsp.: Extrahiere Sätze aus Dokument für Zusammenfassung
3. **Transducer**: Supervision für jeder Ausgabe $\mathbf{y}_t$, Loss wird über die ganze Sequenz summiert oder gemittelt
   Bsp.: Sequence Tagger, z.B. POS, oder auch LM


WE NEED TO GO
DEEPER

Multi-Layer RNNs i

[El Hihi and Bengio, 1995, Pascanu et al., 2013]: RNNs können in Schichten $l = \{1 \dots L\}$ gestapelt werden, sodass sie ein Gitter formen.

Das Input für ein RNN ist (zusätzlich zum vorherigen Zustand $\mathbf{s}_{t-1}^{l}$) das Output des darunterliegenden RNNs $\mathbf{s}_{t}^{l-1}$.

$$\mathbf{s}_t^l = R(\mathbf{s}_{t-1}^l, \mathbf{s}_t^{l-1})$$
$$\mathbf{s}_t^1 = R(\mathbf{s}_{t-1}^1, \mathbf{x}_t)$$
$$\mathbf{y}_t = O(\mathbf{s}_t^L)$$

Multi-Layer RNNs ii

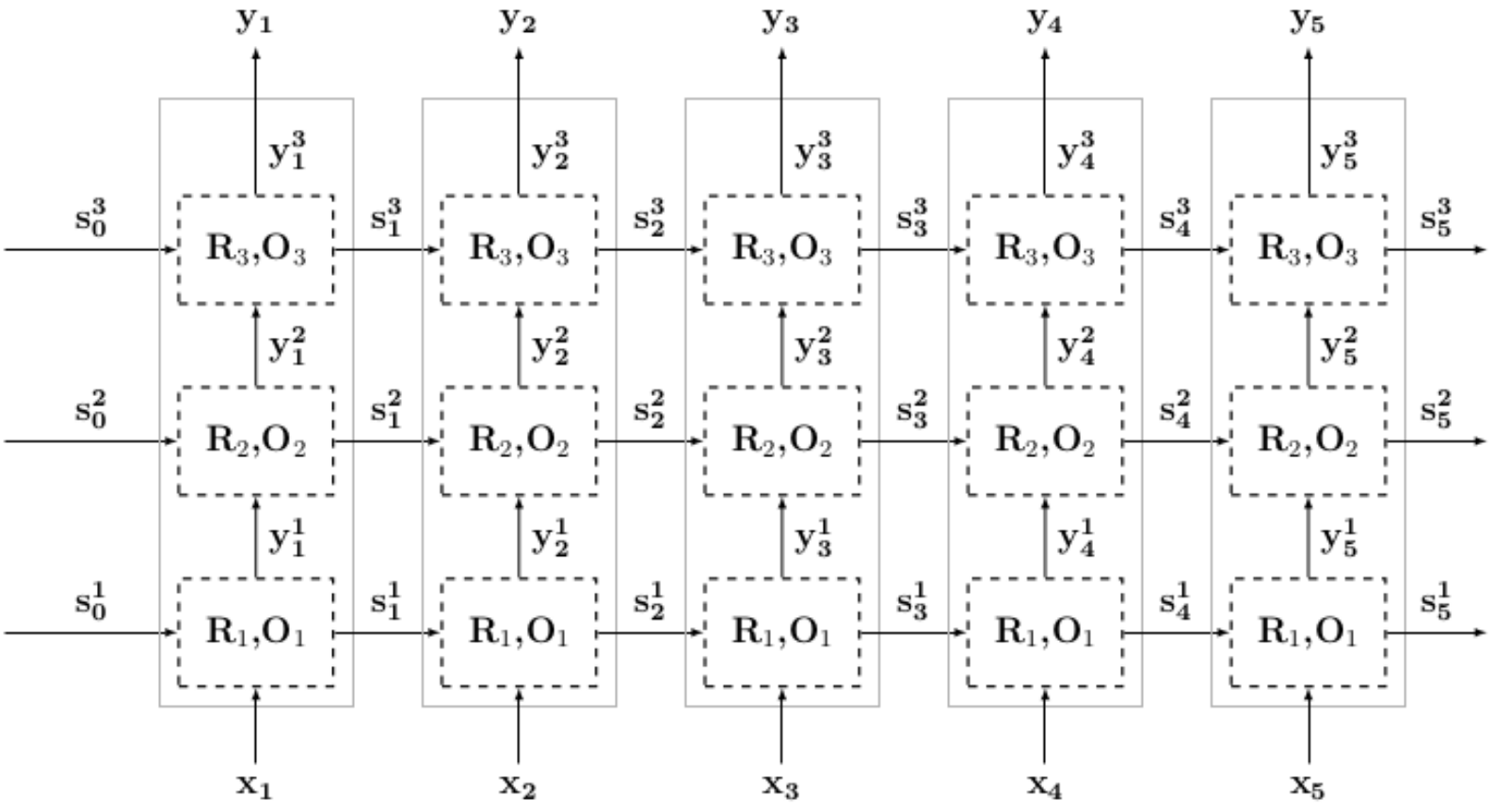


Abbildung 3: Deep RNN, Quelle: [Goldberg, 2015]

Bi-RNN i

[Schuster and Paliwal, 1997]: Bidirektionale RNNs (Bi-RNNs) enkodieren die Sequenz in zwei Richtungen, d.h. konditionieren die Ausgabe auf unbegrenztem Kontext aus **Vergangenheit und Zukunft**.

- **Vorwärts**-Zustand $\mathbf{s}_t^f$ von RNN(R^f, O^f) mit Eingabe $\mathbf{x}_1, \ldots, \mathbf{x}_T$
- **Rückwärts**-Zustand $\mathbf{s}_t^b$ von RNN(R^b, O^b) mit Eingabe $\mathbf{x}_T, \ldots, \mathbf{x}_1$

Bi-RNN: Die Repräsentation für den Zustand an der Position t wird aus beiden RNNs zusammengesetzt: $\mathbf{s}_t = \left[\mathbf{s}_t^f, \mathbf{s}_t^b\right]$.

Bi-RNN ii

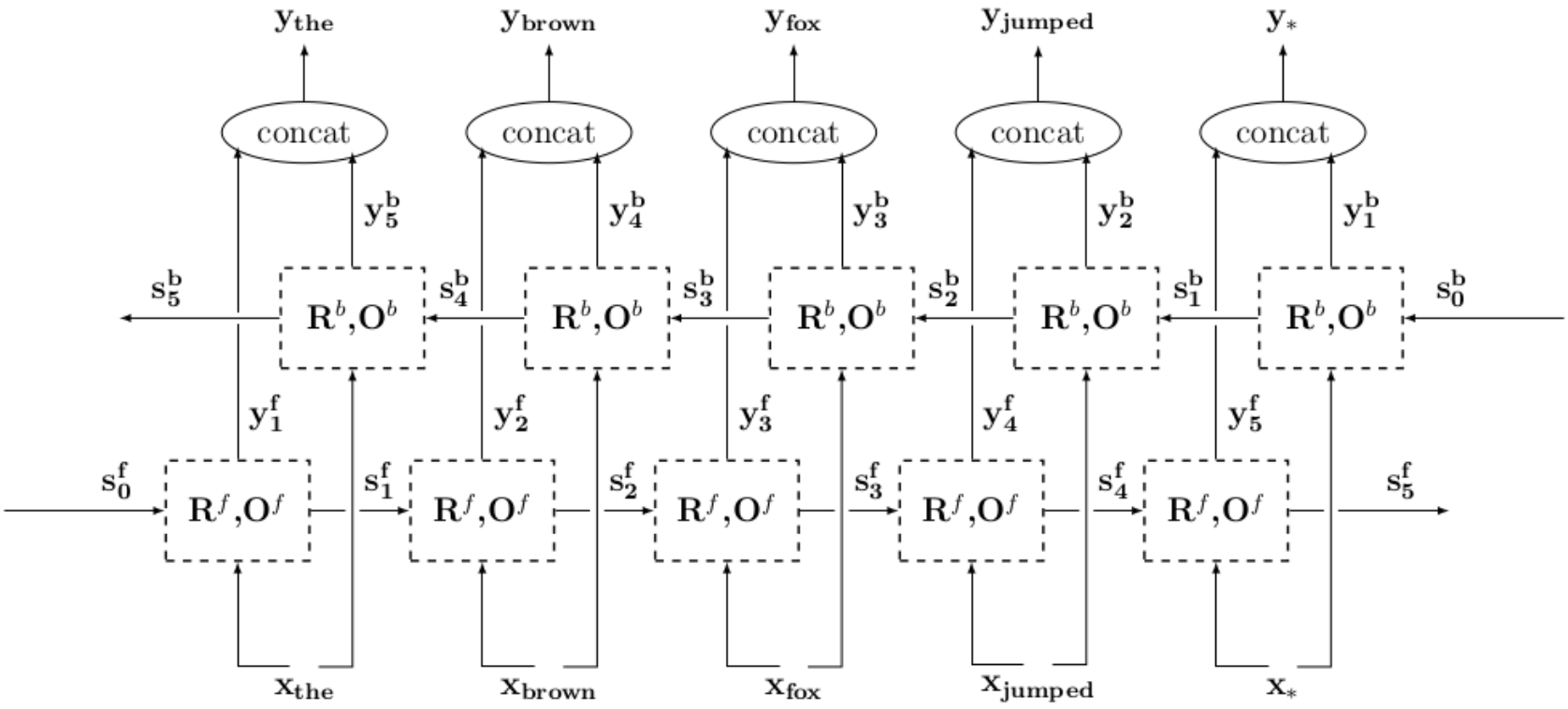


Abbildung 4: Bi-RNN für Language Modeling, Quelle: [Goldberg, 2015]

Instanzen von RNNs

Wie wird R konkret berechnet?

Heute 3 verschiedene Modelle:

1. Elman-RNN
2. Gated RNNs
   - 2.1 LSTM
   - 2.2 GRU

Einfaches RNN

Elman-RNN [Elman, 1990]:

$$\begin{aligned} s_t &= R_{Elman}(s_{t-1}, x_t) = [h_t] \\ &= \sigma(x_t W + h_{t-1} U + b) \\ y_t &= O_{Elman}(s_t) = h_t W_y + b_y \end{aligned}$$

Gewichte des Netzwerks:

- $W \in \mathbb{R}^{d_{in} \times d_h}$ bestimmt Verarbeitung des Inputs
- $U \in \mathbb{R}^{d_h \times d_h}$ bestimmt Verarbeitung des vorherigen Zustands
- $W_y \in \mathbb{R}^{d_h \times d_{out}}$ bestimmt Verarbeitung des Zustands zu Ausgabe

Einfaches RNN

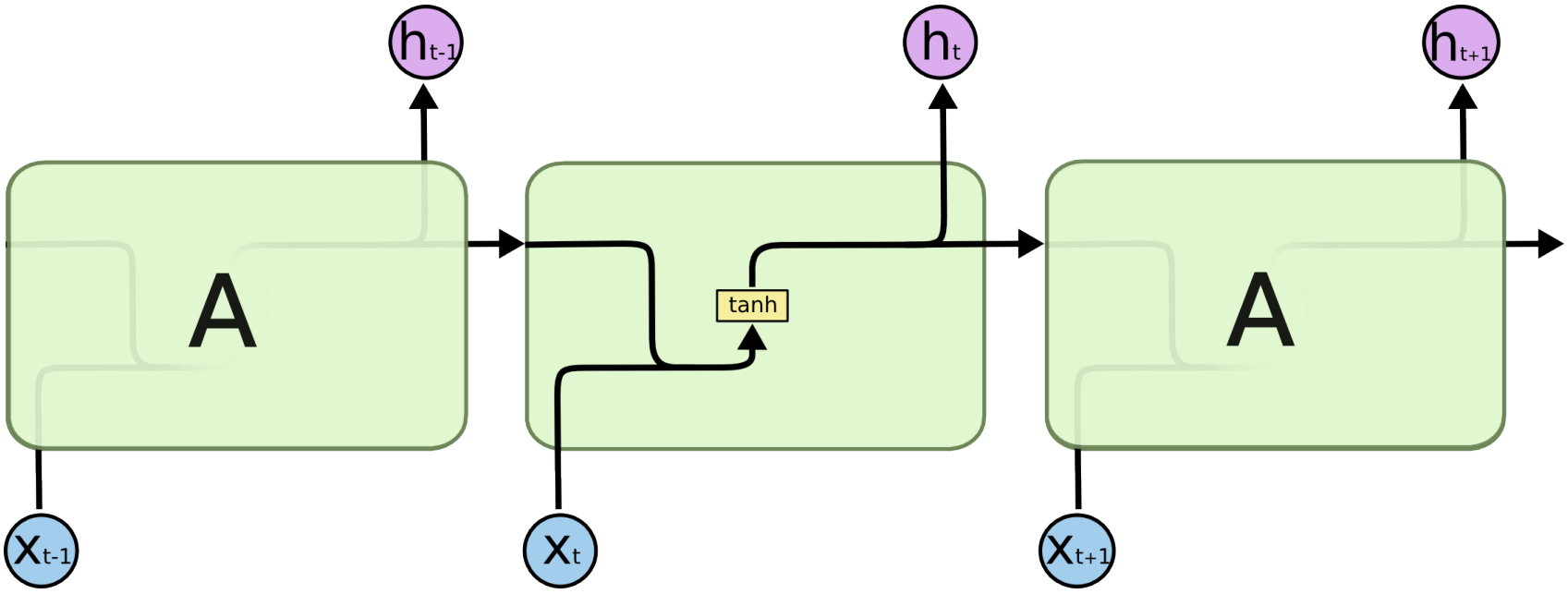


Abbildung 5: RNN-Kette, Quelle: Chris Olah: Understanding LSTM Networks

Beispiel: RNNLM

Konditioniere NN auf allen vorangegangenen Wörtern (keine Markov-Annahme!)

$$p(\mathbf{x}_{t+1} = v_j | \mathbf{x}_t, \dots \mathbf{x}_1) = \texttt{softmax}(\mathbf{y}_i)[j] = \frac{\exp y_{t_j}}{\sum_{i=0} \exp}$$

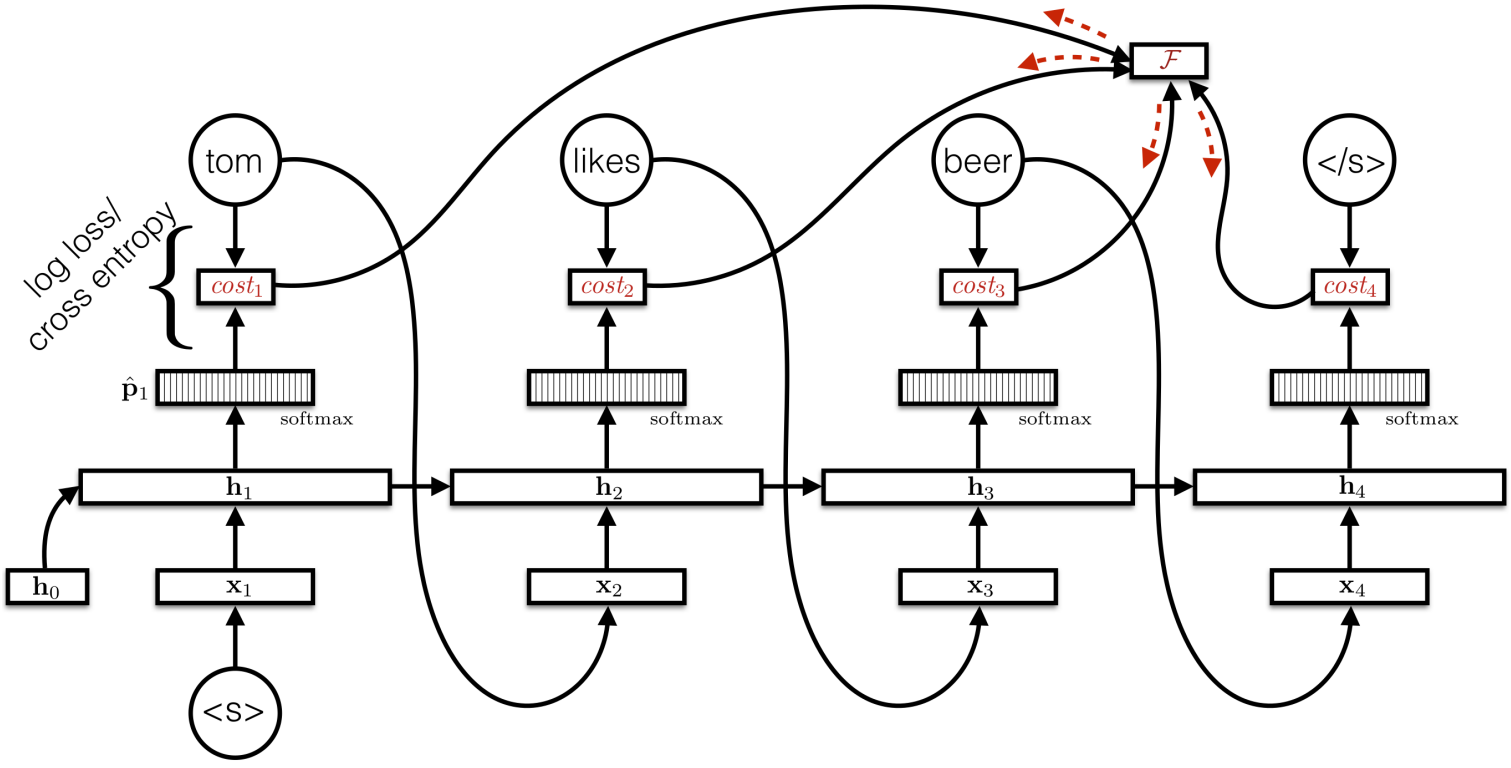


Abbildung 6: RNNLM Training, Quelle: Chris Dyer @LXMLS'16

Ergebnisse

Model	Perplexity			Entropy reduction over baseline	
	individual	+KN5	+KN5+cache	KN5	KN5+cache
3-gram, Good-Turing smoothing (GT3)	165.2	-	-	-	-
5-gram, Good-Turing smoothing (GT5)	162.3	-	-	-	-
3-gram, Kneser-Ney smoothing (KN3)	148.3	-	-	-	-
5-gram, Kneser-Ney smoothing (KN5)	**141.2**	-	-	-	-
5-gram, Kneser-Ney smoothing + cache	**125.7**	-	-	-	-
PAQ8o10t	131.1	-	-	-	-
Maximum entropy 5-gram model	142.1	138.7	124.5	0.4%	0.2%
Random clusterings LM	170.1	126.3	115.6	2.3%	1.7%
Random forest LM	131.9	131.3	117.5	1.5%	1.4%
Structured LM	146.1	125.5	114.4	2.4%	1.9%
Within and across sentence boundary LM	116.6	110.0	108.7	5.0%	3.0%
Log-bilinear LM	144.5	115.2	105.8	4.1%	3.6%
Feedforward neural network LM [50]	140.2	116.7	106.6	3.8%	3.4%
Feedforward neural network LM [40]	141.8	114.8	105.2	4.2%	3.7%
Syntactical neural network LM	131.3	110.0	101.5	5.0%	4.4%
Recurrent neural network LM	124.7	105.7	97.5	5.8%	5.3%
Dynamically evaluated RNNLM	123.2	102.7	98.0	6.4%	5.1%
Combination of static RNNLMs	102.1	95.5	**89.4**	7.9%	7.0%
Combination of dynamic RNNLMs	101.0	**92.9**	90.0	8.5%	6.9%

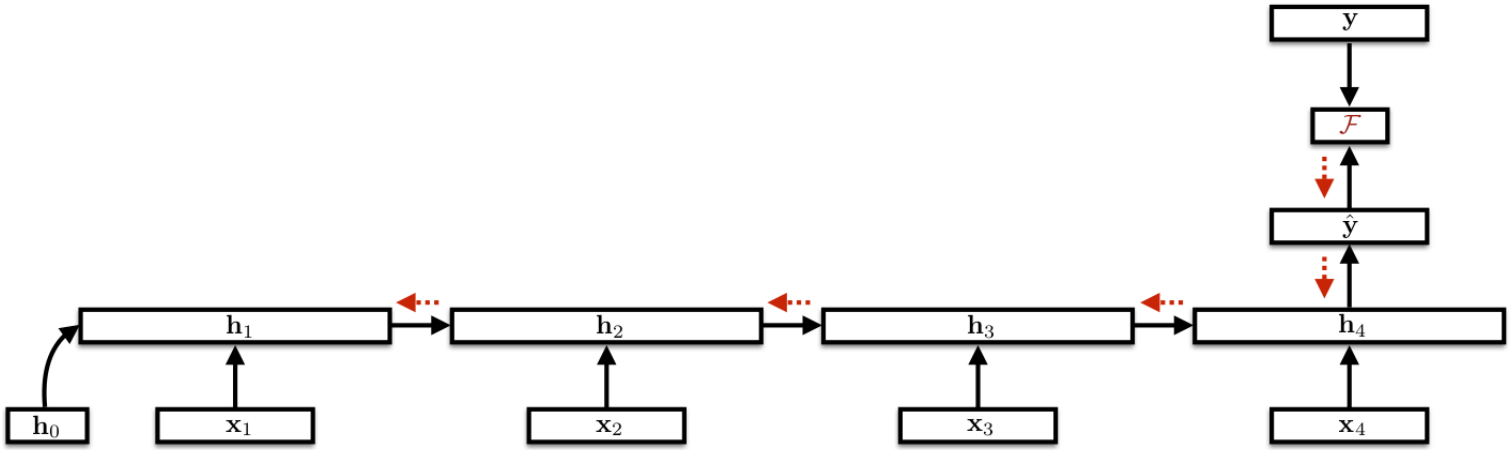
Abbildung 7: Perplexity Ergebnisse auf der Penn-Treebank, [Mikolov, 2012]

BPTT: Beispiel RNN Acceptor i

Nur eine Ausgabe für die ganze Sequenz:

$$\begin{aligned} h_t &= \sigma(x_t W + h_{t-1} U + b) \\ \hat{y} &= h_4 W_y + b_y \end{aligned}$$

Gradient für Gewichtsmatrix U: $\frac{\partial \mathcal{F}}{\partial U} = \sum_{t=1}^{4} \frac{\partial \mathcal{F}}{\partial h_t} \frac{\partial h_t}{\partial U}$



BPTT: Beispiel RNN Acceptor ii

$$\frac{\partial \mathcal{F}}{\partial \mathsf{U}} = \sum_{t=1}^{4} \frac{\partial \mathcal{F}}{\partial \mathsf{h_t}} \frac{\partial \mathsf{h_t}}{\partial \mathsf{U}}$$

für t=1:

$$\begin{aligned}
\frac{\partial \mathcal{F}}{\partial \mathsf{h}_1} &= \frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \frac{\partial \hat{\mathsf{y}}}{\partial \mathsf{h}_1} \\
&= \frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \frac{\partial \hat{\mathsf{y}}}{\partial \mathsf{h}_4} \frac{\partial \mathsf{h}_4}{\partial \mathsf{h}_3} \frac{\partial \mathsf{h}_3}{\partial \mathsf{h}_2} \frac{\partial \mathsf{h}_2}{\partial \mathsf{h}_1} \\
&= \frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \frac{\partial \hat{\mathsf{y}}}{\partial \mathsf{h}_4} \prod_{t=2}^{4} \frac{\partial \mathsf{h}_t}{\partial \mathsf{h}_{t-1}}
\end{aligned}$$

BPTT: Beispiel RNN Acceptor iii

Mit $\sigma(x_t W + h_{t-1} U + b) = \sigma(z_t)$:

$$\begin{aligned}\frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \\ &= \operatorname{diag}(\sigma'(z_t))\, U\end{aligned}$$

Wieder einsetzen:

$$\frac{\partial \mathcal{F}}{\partial h_1} = \frac{\partial \mathcal{F}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \prod_{t=2}^{4} \operatorname{diag}(\sigma'(z_t))\, U$$

Allgemein für $t = 1, \ldots, T$:

$$\frac{\partial \mathcal{F}}{\partial h_t} = \frac{\partial \mathcal{F}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_T} \prod_{t'=t+1}^{T} \operatorname{diag}(\sigma'(z'_t))\, U$$

BPTT: Beispiel RNN Acceptor iv

$$\begin{aligned}\frac{\partial \mathcal{F}}{\partial \mathsf{U}} &= \sum_{t=1}^{T} \left(\frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \frac{\partial \hat{\mathsf{y}}}{\partial \mathsf{h}_T} \prod_{t'=t+1}^{T} \left(\operatorname{diag}(\sigma'(\mathsf{z}_t')) \, \mathsf{U} \right) \frac{\partial \mathsf{h}_t}{\partial \mathsf{U}} \right) \\ &= \sum_{t=1}^{T} \left(\frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \mathsf{W}_\mathsf{y} \, \sigma'(\mathsf{z}_t) \, \mathsf{h}_{t-1} \prod_{t'=t+1}^{T} \operatorname{diag}(\sigma'(\mathsf{z}_t')) \, \mathsf{U} \right)\end{aligned}$$

$\frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}}$ ist die Ableitung der Fehlerfunktion $\mathcal{F}$, hier nicht festgelegt

BPTT: Beispiel RNN Acceptor v

$$
\begin{aligned}
\frac{\partial \mathcal{F}}{\partial \mathsf{U}} &= \sum_{t=1}^{T} \left(\frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \frac{\partial \hat{\mathsf{y}}}{\partial \mathsf{h}_T} \prod_{t'=t+1}^{T} \left(\operatorname{diag}(\sigma'(\mathsf{z}'_t))\, \mathsf{U} \right) \frac{\partial \mathsf{h}_t}{\partial \mathsf{U}} \right) \\
&= \sum_{t=1}^{T} \left(\frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \mathsf{W}_\mathsf{y}\, \sigma'(\mathsf{z}_t)\, \mathsf{h}_{t-1} \prod_{t'=t+1}^{T} \operatorname{diag}(\sigma'(\mathsf{z}'_t))\, \mathsf{U} \right) \\
&= \sum_{t=1}^{T} \left(\frac{\partial \mathcal{F}}{\partial \hat{\mathsf{y}}} \mathsf{W}_\mathsf{y}\, \sigma'(\mathsf{z}_t)\, \mathsf{h}_{t-1}\, \mathsf{U}^{\mathsf{T}-\mathsf{t}+1} \prod_{t'=t+1}^{T} \operatorname{diag}(\sigma'(\mathsf{z}'_t)) \right)
\end{aligned}
$$

Problematisch: $\mathsf{U}^{\mathsf{T}-\mathsf{t}+1}$

Warum?

Vanishing/Exploding Gradients

Je nach der Größe der Werte in der rekurrenten Gewichtsmatrix **U** können die Gradienten **schwinden** oder **explodieren**.

Genauer[1] Sei e_{max} größter Eigenwert von **U**, dann mit $t \to \infty$

falls $e_{max} < 1$: $||\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}|| \to 0$

falls $e_{max} > 1$: $||\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}|| \to \infty$

Was bedeutet das?

[1] Mehr Details in [Cho, 2015], Kapitel 4.3

Vanishing/Exploding Gradients

Je nach der Größe der Werte in der rekurrenten Gewichtsmatrix **U** können die Gradienten **schwinden** oder **explodieren**.

Genauer[1] Sei e_{max} größter Eigenwert von **U**, dann mit $t \to \infty$

falls $e_{max} < 1$: $||\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}|| \to 0$

falls $e_{max} > 1$: $||\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}|| \to \infty$

Was bedeutet das? **Kein Lernen!** von langen Abhängigkeiten

[1] Mehr Details in [Cho, 2015], Kapitel 4.3

Gradient Clipping

Mittel gegen explodierende Gradienten: **“norm clipping”**

$$\tilde{\nabla} = \begin{cases} \tau \frac{\nabla}{||\nabla||} & \text{if } ||\nabla|| > \tau \\ \nabla & \text{otherwise} \end{cases}$$

Gradient Clipping

Mittel gegen explodierende Gradienten: **“norm clipping”**

$$\tilde{\nabla} = \begin{cases} \tau \frac{\nabla}{||\nabla||} & \text{if } ||\nabla|| > \tau \\ \nabla & \text{otherwise} \end{cases}$$

Warum funktioniert das nicht für schwindende Gradienten?

LSTM & GRU

Long Short-Term Memory (LSTM)

[Hochreiter and Schmidhuber, 1997] führen 3 Gates für RNNs ein, die den Zustand eines internen Speichers bestimmen:

- Teil des Zustands ist eine **Speicherzelle**
- Zugang zu diesem Speicher über **"Gates"** kontrolliert
- Gate: $\mathbf{g} \in [0,1]^n$ wird komponentenweise mit anderem Vektor $\mathbf{v} \in \mathbb{R}^n$ multipliziert ($\odot$)
- Gates werden gelernt
- Gates entscheiden, wie viel des neuen Inputs in den Speicher soll, und wie viel vergessen werden soll
- Änderung des Inhalts der Speicherzelle über **Addition**
- verhindert schwindende Gradienten

LSTM: Definition[2]

$$
\begin{aligned}
s_t &= R_{\text{LSTM}}(s_{t-1}, x_t) = [c_t; h_t] && \text{LSTM state} \\
c_t &= c_{t-1} \odot f + g \odot i && \text{cell state} \\
h_t &= \tanh(c_t) \odot o && \text{hidden state} \\
i &= \sigma(x_t W^{xi} + h_{t-1} W^{hi}) && \text{input gate} \\
f &= \sigma(x_t W^{xf} + h_{t-1} W^{hf}) && \text{forget gate} \\
o &= \sigma(x_t W^{xo} + h_{t-1} W^{ho}) && \text{output gate} \\
g &= \tanh(x_t W^{xg} + h_{t-1} W^{hg}) && \text{candidate hidden state} \\
y_t &= O_{\text{LSTM}}(s_t) = h_t W_y && \text{output}
\end{aligned}
$$

[2] Wir nehmen an, Bias-Terme sind in den Gewichtsmatrizen enthalten.

LSTM: Definition[2]

$$
\begin{aligned}
s_t &= R_{\text{LSTM}}(s_{t-1}, x_t) = [c_t; h_t] && \text{LSTM state} \\
c_t &= c_{t-1} \odot f + g \odot i && \text{cell state} \\
h_t &= \tanh(c_t) \odot o && \text{hidden state} \\
i &= \sigma(x_t W^{xi} + h_{t-1} W^{hi}) && \text{input gate} \\
f &= \sigma(x_t W^{xf} + h_{t-1} W^{hf}) && \text{forget gate} \\
o &= \sigma(x_t W^{xo} + h_{t-1} W^{ho}) && \text{output gate} \\
g &= \tanh(x_t W^{xg} + h_{t-1} W^{hg}) && \text{candidate hidden state} \\
y_t &= O_{\text{LSTM}}(s_t) = h_t W_y && \text{output}
\end{aligned}
$$

Welche Dimensionen haben die Matrizen (ohne Bias)?

[2] Wir nehmen an, Bias-Terme sind in den Gewichtsmatrizen enthalten.

LSTM

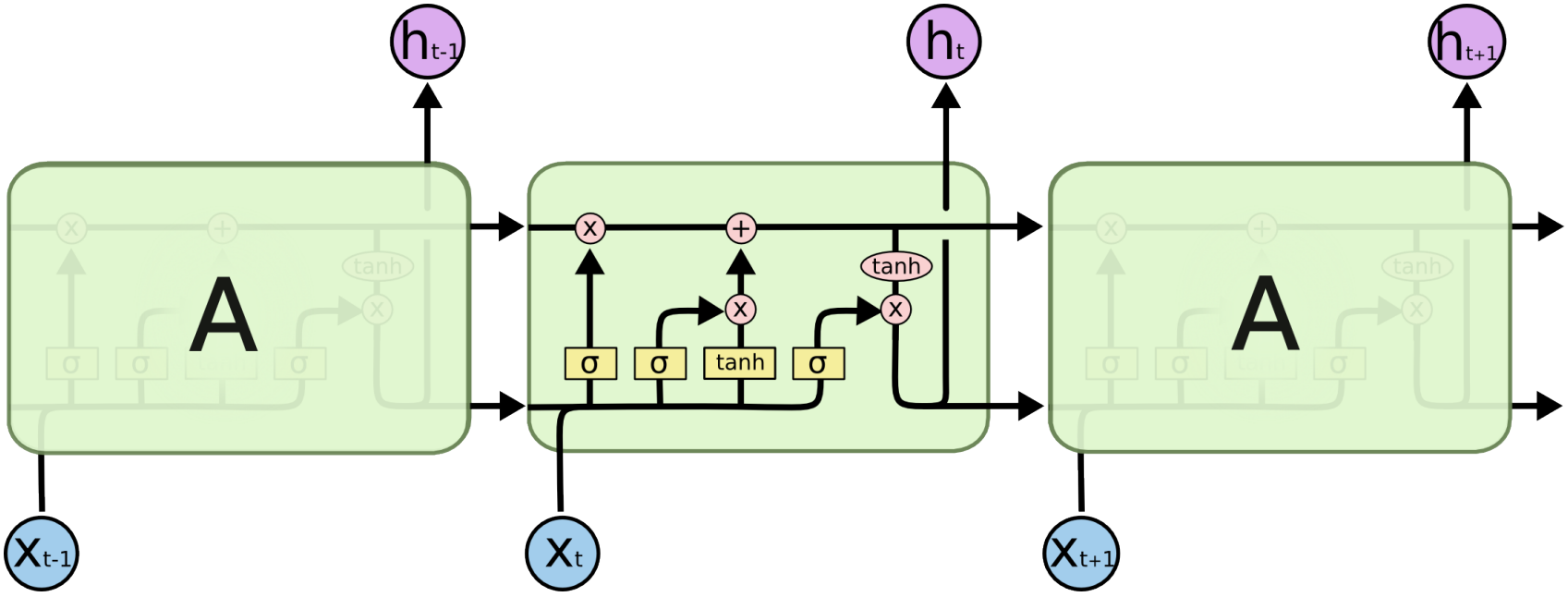


Abbildung 8: LSTM-Kette, Quelle: Chris Olah: Understanding LSTM Networks

Gated Recurrent Units (GRUs)

[Cho et al., 2014] stellen eine ähnliche Idee vor, nur mit weniger Gates:

- gleicher Zweck wie LSTM
- weniger komplex:
  - weniger Gates
  - ohne separate Speicher-Komponente
- einfacher zu implementieren und zu berechnen
- in Experimenten vergleichbare Ergebnisse zu LSTM [Chung et al., 2014], ist aber aufgabenabhängig für NLP [Jozefowicz et al., 2015]

GRU

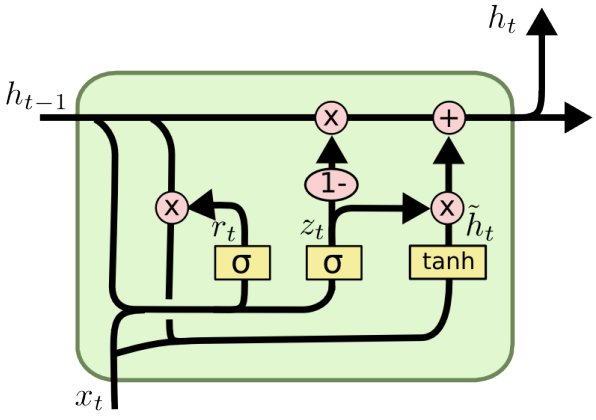


Abbildung 9: GRU, Quelle: Chris Olah: Understanding LSTM Networks

$$
\begin{aligned}
s_t &= R_{\text{GRU}}(s_{t-1}, x_t) = [h_t] && \text{GRU state} \\
&= (1 - z) \odot s_{t-1} + z \odot \tilde{h} && \text{interpolation} \\
z &= \sigma(x_t W^{xz} + h_{t-1} W^{hz}) && \text{update gate} \\
r &= \sigma(x_t W^{xr} + h_{t-1} W^{hr}) && \text{reset gate} \\
\tilde{h} &= \tanh(x_t W^{xh} + (h_{t-1} \odot r) W^{hg}) && \text{proposed state} \\
y_t &= O_{\text{GRU}}(s_t) = s_t W_y && \text{output}
\end{aligned}
$$

Ausblick

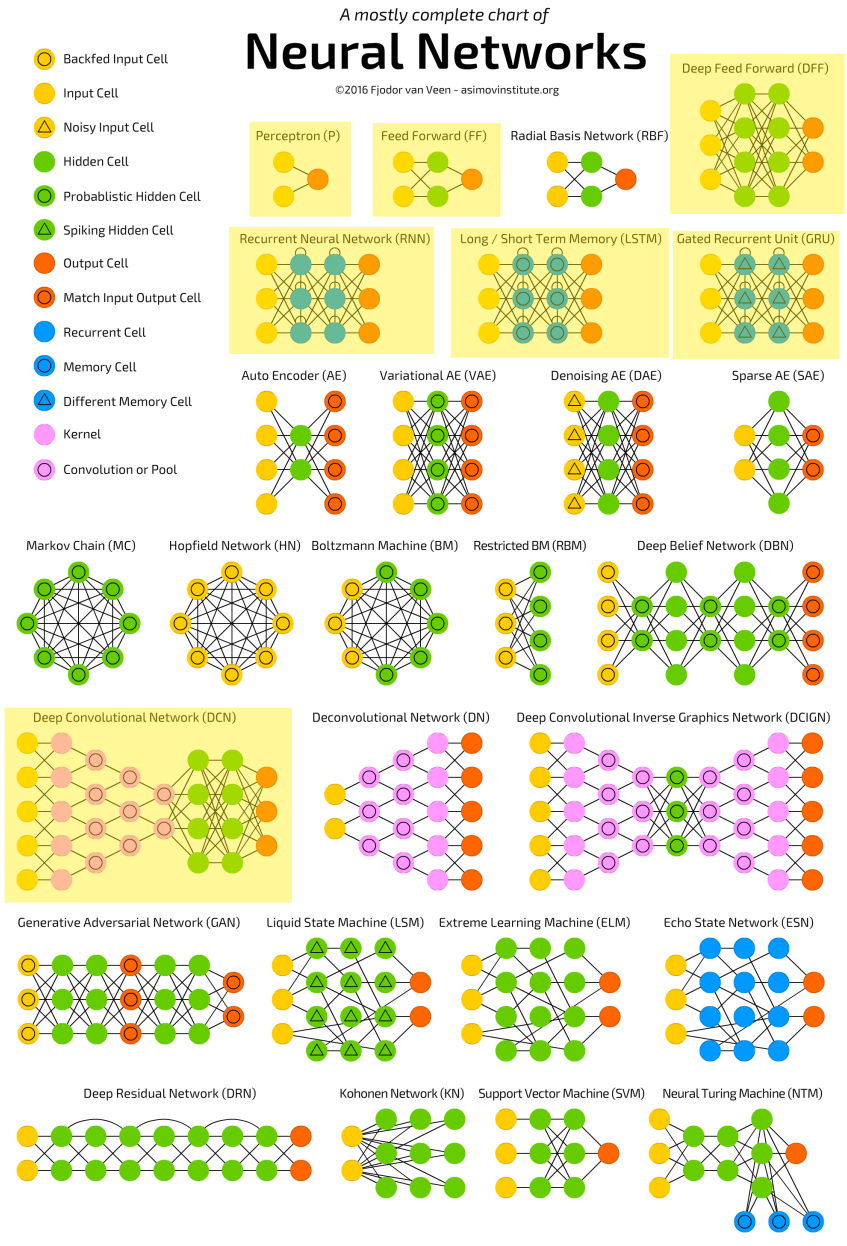


Abbildung 10: The Neural Network Zoo

Recurrent vs. Recursive NN

Sogenannte **Recursive** NN generalisieren über **Recurrent** NN[3]:

- Die Rekursion ist über eine hierarchische Struktur definiert.
- Zustände von Kind-Knoten werden zu Zuständen von Eltern-Knoten kombiniert.

Standard-RNNs erwarten eine sequentiellen Eingabe.

[3] Mehr am 20.12.!

Recurrent vs. Recursive NN

Sogenannte **Recursive** NN generalisieren über **Recurrent** NN[3]:

- Die Rekursion ist über eine hierarchische Struktur definiert.
- Zustände von Kind-Knoten werden zu Zuständen von Eltern-Knoten kombiniert.

Standard-RNNs erwarten eine sequentiellen Eingabe.
Was sind Kind- und Eltern-Knoten?

[3] Mehr am 20.12.!

Recurrent vs. Recursive NN

Sogenannte **Recursive** NN generalisieren über **Recurrent** NN[3]:

- Die Rekursion ist über eine hierarchische Struktur definiert.
- Zustände von Kind-Knoten werden zu Zuständen von Eltern-Knoten kombiniert.

Standard-RNNs erwarten eine sequentiellen Eingabe.
Kind-Knoten sind der vorherige Zustand und das neue Input. Diese werden zu einem Repräsentation für den neuen Zustand kombiniert.

[3] Mehr am 20.12.!

Nächstes Mal

Nächste Woche:

- Vorlesung fällt aus
- Übung findet statt!

In zwei Wochen:

- Encoder-Decoder (“seq2seq”)
- Attention

Spaß mit RNNs

- The Unreasonable Effectiveness of RNNs:
  Shakespeare/LaTeX/Source Code Generator with Character RNNs, u.v.m.

  ```
  PANDARUS:
  Alas, I think he shall be come approached and the day
  When little srain would be attain'd into being never
  fed,
  And who is but a chain and subjects of his death,
  I should not sleep.
  ```

- Awesome RNNs: Code-Bespiele, Anwendungen, Literatur

References i


Cho, K. (2015).
Natural language understanding with distributed representation.
arXiv preprint arXiv:1511.07916.


Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014).
Learning phrase representations using rnn encoder-decoder for statistical machine translation.
arXiv preprint arXiv:1406.1078.


Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014).
Empirical evaluation of gated recurrent neural networks on sequence modeling.
arXiv preprint arXiv:1412.3555.

References ii

El Hihi, S. and Bengio, Y. (1995).
Hierarchical recurrent neural networks for long-term dependencies.
In *NIPS*, volume 400, page 409.

Elman, J. L. (1990).
Finding structure in time.
Cognitive science, 14(2):179–211.

Goldberg, Y. (2015).
A primer on neural network models for natural language processing.
arXiv preprint arXiv:1510.00726.

Hochreiter, S. and Schmidhuber, J. (1997).
Long short-term memory.
Neural computation, 9(8):1735–1780.

References iii


Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015).
An empirical exploration of recurrent network architectures.
In *Proceedings of The 32nd International Conference on Machine Learning*, pages 2342–2350.


Mikolov, T. (2012).
Statistical language models based on neural networks.
Ph.D. Thesis at Brno University of Technology.


Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013).
How to construct deep recurrent neural networks.
arXiv preprint arXiv:1312.6026.


Schuster, M. and Paliwal, K. K. (1997).
Bidirectional recurrent neural networks.
IEEE Transactions on Signal Processing, 45(11):2673–2681.

References iv

Werbos, P. J. (1990).
Backpropagation through time: what it does and how to do it.
Proceedings of the IEEE, 78(10):1550–1560.