Deep Learning mit Neuronalen Netzen

Vorlesung “Computerlinguistische Techniken”

Alexander Koller

29. Januar 2016
Deep Learning Machine Beats Humans in IQ Test

Computers have never been good at answering the type of verbal reasoning questions found in IQ tests. Now a deep learning machine unveiled in China is changing that.
<table>
<thead>
<tr>
<th>TASK</th>
<th>HOURS OF TRAINING DATA</th>
<th>DNN-HMM</th>
<th>GMM-HMM WITH SAME DATA</th>
<th>GMM-HMM WITH MORE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWITCHBOARD (TEST SET 1)</td>
<td>309</td>
<td>18.5</td>
<td>27.4</td>
<td>18.6 (2,000 H)</td>
</tr>
<tr>
<td>SWITCHBOARD (TEST SET 2)</td>
<td>309</td>
<td>16.1</td>
<td>23.6</td>
<td>17.1 (2,000 H)</td>
</tr>
<tr>
<td>ENGLISH BROADCAST NEWS</td>
<td>50</td>
<td>17.5</td>
<td>18.8</td>
<td></td>
</tr>
<tr>
<td>BING VOICE SEARCH (SENTENCE ERROR RATES)</td>
<td>24</td>
<td>30.4</td>
<td>36.2</td>
<td></td>
</tr>
<tr>
<td>GOOGLE VOICE INPUT</td>
<td>5,870</td>
<td>12.3</td>
<td></td>
<td>16.0 (&gt;&gt; 5,870 H)</td>
</tr>
<tr>
<td>YOUTUBE</td>
<td>1,400</td>
<td>47.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DNN = deep neural networks

(Hinton et al. 2012)
Erfolge: Bildererkennung

Microsoft's Deep Learning Project Outperforms Humans In Image Recognition

Technology is blanketed in dishonesty. Computer phones are smart, software automations become intelligence, and coerced financialization becomes sharing. Because of the deceptive language surrounding these instruments it's difficult to talk about how they're used, and at what cost. Instead we're forced into false debates about sharing versus not sharing, intelligence versus inefficiency, progress versus everything.

Deep learning is as big a fraud as any of these endeavors, an expensive and obscure discipline built around the claim that computers can mimic human neuronal function and thus learn as well or better than humans. This week, Microsoft Research announced its newest deep learning project had outperformed humans in a test to identify objects in digital images. Researchers noted their scores shouldn't be taken as proof that computer image identification in general was better than humans, admitting many general case instances where humans were better able to pick out objects in ambiguous or context specific image. However, algorithms excelled at “fine-grained recognition,” which might rely on a category expertise beyond the average person but trivial for massive computer archives of data. “While humans can easily recognize these objects as a bird, a dog, and a flower, it is nontrivial for most humans to tell their species,” the researchers wrote.

(He et al., 2015)
Microsoft's Deep Learning Project Outperforms Humans In Image Recognition

Technology is blanketed in dishonesty. Our financialization becomes sharing. Because they're used, and at what cost. Instead we must consider progress versus everything.

Deep learning is as big a fraud as any other. Microsoft recently announced its newest deep learning program, and their scores shouldn’t be taken as proof of the general case instances where humans were excelled at “fine-grained recognition,” we can say: computer archives of data. “While humans to tell their species,” the researchers concede.

Figure 4. Example validation images successfully classified by our method. For each image, the ground-truth label and the top-5 labels predicted by our method are listed.

Erfolge: Bilderkennung

(He et al., 2015)
Hype?

- Signal im menschlichen Gehirn braucht 10ms, um von einem Neuron zum anderen zu kommen.

- Das bedeutet: Jedes Problem, das ein Mensch in 100ms lösen kann, kann ein zehn-lagiges künstliches neuronales Netzwerk auch lösen.

- Wollen wir das glauben?
Kurze Geschichte

- 1958 - Rosenblatt erfindet Perceptron.
Ein künstliches Neuron

\[ o = f\left(\sum_{i=1}^{n} w_i \cdot a_i + b\right) \]

f ist eine *Aktivierungsfunktion*, typischerweise nicht-linear.
Typische Aktivierungsfunktionen

sigmoid: \[ \sigma(z) = \frac{1}{1 + e^{-z}} \]

tanh: \[ \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2 \cdot \sigma(2z) - 1 \]

HardTanh: \[
\begin{align*}
-1 & \quad \text{if } x < -1 \\
x & \quad \text{if } -1 \leq x \leq 1 \\
1 & \quad \text{if } x > 1
\end{align*}
\]

softsign: \[\text{softsign}(z) = \frac{a}{1 + |a|} \]

rect: \[\text{rect}(z) = \max(z, 0)\]
Perzeptron

einfachstes denkbarees NN: Eine Schicht, $f(x) = x$

Mark I Perceptron, 1957
Nichtlinearitäten

- Example: function approximation, e.g., regression or classification
- Without non-linearities, deep neural networks can't do anything more than a linear transform
- Extra layers could just be compiled down into a single linear transform:

\[ W_1' W_2' x = W x \]

- With more layers, they can approximate more complex functions!

\[ f \text{ linear: kann Klassen nur an Hyperebene trennen} \]
\[ f \text{ nichtlinear: kann an “krummen” Linien trennen} \]

(Bilder von Socher-Folien)
Tiefere Netzwerke

\[ h_j^{(1)} = f_1\left(\sum_{i=1}^{n} w_{ji}^{(1)} \cdot x_i \right) \]

\[ h^{(3)} = f_3(W^{(3)} \cdot h^2) \]

\( W^{(i)} \) sind Gewichtsmatrizen
Tiefere Netzwerke

• Beispiel: Funktionenapproximation, z.B., Regression oder Klassifikation
• Ohne nichtlineare Komponenten können tiefe neuronalen Netze nur eine lineare Transformation durchführen.
• Zusätzliche Schichten können einfach in eine lineare Transformation zusammengeschrieben werden:
  \[ W_{1}' W_{2}' x = W x \]
• Mit mehr Schichten können komplexere Funktionen approximiert werden!

Je mehr Schichten (M), desto einfacher kann ein neuronales Netz auch komplexe Funktionen beschreiben.

Entsprechend steigt die Gefahr von Overfitting!
Log-lineare Modelle und NNs


- Netzwerk von solchen Neuronen repräsentiert beliebig gewichtete Kombinationen von log-linearen Modellen.
Training von NNs

● Wie Parameter von NNs aus Daten schätzen?
  ▶ Parameter = Einträge in den Gewichtsmatrizen
  ▶ überwacht Lernen: Trainingsinstanzen
    \( (x(1), y(1)), \ldots, (x(N), y(N)) \)

● Muss Zielfunktion \( J(o, y) \) festlegen:
  ▶ vergleicht Ausgabe \( o \) des NN mit Goldstandard \( y \)
  ▶ Lerner minimiert Wert von \( J(o, y) \) auf den Trainingsdaten
  ▶ in der Literatur viele Zielfunktionen
Training von NNs

- Standard-Trainingsverfahren für NNs: (Stochastic) Gradient Descent.

- Berechne in jeder Iteration den Gradienten $\nabla J$ von $J$, d.h. den Vektor $\left( \frac{\partial J}{\partial w_{11}^{(1)}}, \ldots, \frac{\partial J}{\partial w_{mn}^{(M)}} \right)$

- Gradient zeigt in Richtung des steilsten Abstiegs. Damit Update:

$$W_{t+1} = W_t + \eta \cdot \nabla J(W_t)$$
Berechnung des Gradienten

- *Backpropagation*-Algorithmus berechnet effizient Gradienten unter Ausnutzung der Kettenregel:

\[
\begin{align*}
\frac{\partial J}{\partial w_1} &= \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial b} \cdot \frac{\partial b}{\partial w_1} \\
&= (y - o) \cdot \frac{\sigma(b)}{1 - \sigma(b)} \cdot x_1
\end{align*}
\]

\[
b = \sum_{i=1}^{n} w_i \cdot x_i
\]

\[
o = \sigma(b)
\]

\[
J(o, y) = \frac{1}{2} (o - y)^2 \quad \text{(Least Mean Squares)}
\]
Berechnung des Gradienten

- Backpropagation funktioniert auch, wenn $x_1$ kein Eingabeknoten ist, sondern innerer Knoten.
  - Dann im nächsten Schritt nochmal Kettenregel anwenden und Gewichte für eingehende Kanten von $x_1$, etc.

- Vanishing-Gradient-Problem: Je weiter Schicht von Ausgabeschicht entfernt liegt, desto näher an Null werden die Gradienten.

- Auswege: Pre-Training; massiv parallele Berechnung (auf Grafikkarten); Tricks.
Anwendungen in der CL

- Mikolov et al. 10, 13: Word Embeddings (word2vec)

- Collobert et al. 11: NLP from scratch (Named Entity Recognition, POS-Tagging, etc.)

- Sutskever et al. 14, Vinyals et al. 15: Maschinelle Übersetzung, Parsing

Zentrale Frage: Wie repräsentiert man Wörter?

→ word embeddings

**Word Embeddings**

| cat | sat | ?? ?? | a | mat |

on
Word Embeddings

- One-hot encoding: Jedes Wort ein 0-1 Vektor.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

- Word Embeddings: \( n \times |V| \)-Matrix \( L \), die one-hot encoding in einen \( n \)-dimensionalen Vektor abbildet.

\[
L = \begin{bmatrix}
|V| \\
\vdots \\
\vdots \\
\vdots \\
|V| \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
the & cat & mat & \ldots
\end{bmatrix}
\]
CBOW = continuous bag of words

where \( C \) is the number of words in the context, \( w_1, \ldots, w_C \) are the words in the context, and \( v_w \) is the input vector of a word \( w \). The loss function is

\[
E = \text{log} \ p(w_O | w_I, 1, \ldots, w_I, C) \quad (19)
\]

\[
= u_j^\ast + \text{log} \ X_{j=1}^V \exp(u_j^0) \quad (20)
\]

\[
= v_0^w OT \cdot h + \text{log} \ X_{j=1}^V \exp(v_0^w h_T \cdot h) \quad (21)
\]

which is the same as (7), the objective of the one-word-context model, except that \( h \) is different, as defined in (18) instead of (1).

The update equation for the hidden output weights stay the same as that for the one-word-context model (11). We copy it here:

\[
v_0^w (\text{new}) = v_0^w (\text{old}) \cdot e_j \cdot h_i \quad \text{for } j = 1, 2, \ldots, V.
\]

Note that we need to apply this to every element of the hidden output weight matrix for each training instance.

6 = continuous bag of words

one-hot encodings von Wörtern an Position -2, -1, +1, +2

ähnlich ist Skip-Gram-Modell: Vorhersage von Kontext aus Wort
Softmax

- Kann NN nicht zwingen, dass in Ausgabeschicht genau ein Neuron Wert 1 hat.

- Stattdessen: konvertiere Ausgabeschicht in eine W.verteilung über mögliche Wörter mit der Softmax-Funktion:

\[ P(y_j \mid x_1, \ldots, x_C) = \frac{e^{y_j}}{\sum_i e^{y_i}} \]

- Zielfunktion für Training: \( \log P(y_j \mid x_1, \ldots, x_C) \).
Table 6: Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.

<table>
<thead>
<tr>
<th>Model</th>
<th>Vector Dimensionality</th>
<th>Training Time [days x CPU cores]</th>
<th>Accuracy [%]</th>
<th>Training Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic</td>
<td>100</td>
<td>6B</td>
<td>34.2</td>
<td>64.5</td>
</tr>
<tr>
<td>Syntactic</td>
<td>1000</td>
<td>6B</td>
<td>57.3</td>
<td>68.9</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>50.8</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Comparison and combination of models on the Microsoft Sentence Completion Challenge.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-gram [32]</td>
<td>39</td>
</tr>
<tr>
<td>Average LSA similarity [32]</td>
<td>49</td>
</tr>
<tr>
<td>Log-bilinear model [24]</td>
<td>54.8</td>
</tr>
<tr>
<td>RNNLMs [19]</td>
<td>55.4</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>48.0</td>
</tr>
<tr>
<td>Skip-gram + RNNLMs</td>
<td><strong>58.9</strong></td>
</tr>
</tbody>
</table>

While the Skip-gram model itself does not perform on this task better than LSA similarity, the scores from this model are complementary to scores obtained with RNNLMs, and a weighted combination leads to a new state of the art result 58.9% accuracy (59.2% on the development part of the set and 58.7% on the test part of the set).

Table 8 shows words that follow various relationships. We follow the approach described above: the relationship is defined by subtracting two word vectors, and the result is added to another word. Thus for example, \( \text{Paris} - \text{France} + \text{Italy} = \text{Rome} \). As it can be seen, accuracy is quite good, although there is clearly a lot of room for further improvements (note that using our accuracy metric that..."
### Ergebnisse

(auf Analogie-Task)

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>France - Paris</td>
<td>Italy: Rome</td>
<td>Japan: Tokyo</td>
<td>Florida: Tallahassee</td>
</tr>
<tr>
<td>big - bigger</td>
<td>small: larger</td>
<td>cold: colder</td>
<td>quick: quicker</td>
</tr>
<tr>
<td>Miami - Florida</td>
<td>Baltimore: Maryland</td>
<td>Dallas: Texas</td>
<td>Kona: Hawaii</td>
</tr>
<tr>
<td>Einstein - scientist</td>
<td>Messi: midfielder</td>
<td>Mozart: violinist</td>
<td>Picasso: painter</td>
</tr>
<tr>
<td>Sarkozy - France</td>
<td>Berlusconi: Italy</td>
<td>Merkel: Germany</td>
<td>Koizumi: Japan</td>
</tr>
<tr>
<td>copper - Cu</td>
<td>zinc: Zn</td>
<td>gold: Au</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>Putin: Medvedev</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: Linux</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>IBM: McNealy</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>France: tapas</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>

(cos-ähnlichster Vektor zu L(Paris) - L(France) + L(Italy) etc.)
NLP “almost from scratch”

Input Window

Text
Feature 1
Feature K

Lookup Table

$LT_{W^1}$
$\vdots$
$LT_{W^K}$

Linear

$M^1 \times$

HardTanh

Linear

$M^2 \times$

one-hot encoding
von Tag (POS, NER, …)

(Collobert et al. 2011)
<table>
<thead>
<tr>
<th>Approach</th>
<th>POS (PWA)</th>
<th>CHUNK (F1)</th>
<th>NER (F1)</th>
<th>SRL (F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark Systems</td>
<td>97.24</td>
<td>94.29</td>
<td>89.31</td>
<td>77.92</td>
</tr>
<tr>
<td>NN+WLL</td>
<td>96.31</td>
<td>89.13</td>
<td>79.53</td>
<td>55.40</td>
</tr>
<tr>
<td>NN+SLL</td>
<td>96.37</td>
<td>90.33</td>
<td>81.47</td>
<td>70.99</td>
</tr>
<tr>
<td>NN+WLL+LM1</td>
<td>97.05</td>
<td>91.91</td>
<td>85.68</td>
<td>58.18</td>
</tr>
<tr>
<td>NN+SLL+LM1</td>
<td>97.10</td>
<td>93.65</td>
<td>87.58</td>
<td>73.84</td>
</tr>
<tr>
<td>NN+WLL+LM2</td>
<td>97.14</td>
<td>92.04</td>
<td>86.96</td>
<td>58.34</td>
</tr>
<tr>
<td>NN+SLL+LM2</td>
<td>97.20</td>
<td>93.63</td>
<td>88.67</td>
<td>74.15</td>
</tr>
</tbody>
</table>

Table 8: Comparison in generalization performance of benchmark NLP systems with our (NN) approach on POS, chunking, NER and SRL tasks. We report results with both the word-level log-likelihood (WLL) and the sentence-level log-likelihood (SLL). We report with (LM) performance of the networks trained from the language model embeddings (Table 7). Generalization performance is reported in per-word accuracy (PWA) for POS and F1 score for other tasks.

Our system does not use such parse trees because we attempt to learn this information from the unlabeled data set. It is therefore legitimate to question whether our ranking criterion (17) has the conceptual capability to capture such a rich hierarchical information. At first glance, the ranking task appears unrelated to the induction of probabilistic grammars that underly standard parsing algorithms. The lack of hierarchical representation seems a fatal flaw (Chomsky, 1956). However, ranking is closely related to an alternative description of the language structure: operator grammars (Harris, 1968). Instead of directly studying the structure of a sentence, Harris defines an algebraic structure on the space of all sentences. Starting from a couple of elementary sentence forms, sentences are described by the successive application of sentence transformation operators. The sentence structure is revealed as a side effect of the successive transformations. Sentence transformations can also have a semantic interpretation.
Rekurrente NNs

- Ein *rekurrentes* NN liest Sequenzen beliebiger Länge als Input (z.B. Sätze).
- In jedem Schritt sieht es auch hidden states des vorherigen Schritts als Input.
Rekurrente NNs

- Training von RNNs durch “Backpropagation over time” = normale Backprop, bei der das RNN über die ganze Eingabesequenz aufgefaltet wird.

- Wichtigster Typ in CL ist das “Long Short-Term Memory Network” (LSTM), in dem verborgene Schicht eine festgelegte Form hat.

http://blog.otoro.net
LSTMs for MT

Sequences pose a challenge for DNNs because they require that the input and output is known and fixed. In this paper, we show that a straightforward application of the Long Short-Term Memory (LSTM) architecture [16] can solve general sequence to sequence problems. The idea is to use one LSTM to read the input sequence, one time step at a time, to obtain large fixed-dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector (fig. 1). The second LSTM is essentially a recurrent neural network language model [28, 23, 30] except that it is conditioned on the input sequence. The LSTM’s ability to successfully learn on data with long range temporal dependencies makes it an automatic choice for this application due to the considerable time lag between the inputs and their corresponding outputs (fig. 1).

There have been a number of related attempts to address the general sequence to sequence learning problem with neural networks. Our approach is closely related to Kalchbrenner and Blunsom [18] who were the first to map the entire input sentence to vector, and is related to Cho et al. [5] although the latter was used only for rescoring hypotheses produced by a phrase-based system. Graves [10] introduced a novel differentiable attention mechanism that allows neural networks to focus on different parts of their input, and an elegant variant of this idea was successfully applied to machine translation by Bahdanau et al. [2]. The Connectionist Sequence Classification is another popular technique for mapping sequences to sequences with neural networks, but it assumes a monotonic alignment between the inputs and the outputs [11].

The main result of this work is the following. On the WMT’14 English to French translation task, we obtained a BLEU score of 34.81 by directly extracting translations from an ensemble of 5 deenp LSTMs (with 384M parameters and 8,000 dimensional state each) using a simple left-to-right beam-search decoder. This is by far the best result achieved by direct translation with large neural networks. For comparison, the BLEU score of an SMT baseline on this dataset is 33.30 [29]. The 34.81 BLEU score was achieved by an LSTM with a vocabulary of 80k words, so the score was penalized whenever the reference translation contained a word not covered by these 80k. This result shows that a relatively unoptimized small-vocabulary neural network architecture which has much room for improvement outperforms a phrase-based SMT system.

Finally, we used the LSTM to rescore the publicly available 1000-best lists of the SMT baseline on the same task [29]. By doing so, we obtained a BLEU score of 36.5, which improves the baseline by 3.2 BLEU points and is close to the previous best published result on this task (which is 37.0 [9]).

Surprisingly, the LSTM did not suffer on very long sentences, despite recent experience of other researchers with related architectures [26]. We were able to do so because we reversed the order of words in the source sentence but not the target sentences in the training and test set. By doing so, we introduced many short term dependencies that made the optimization problem much simpler (see sec. 2 and 3.3). As a result, SGD could learn LSTMs that had no trouble with long sentences. The simple trick of reversing the words in these source sentences is one of the key technical contributions of this work.

LSTM 1 liest Eingabe Wort für Wort und codiert sie in Zustand s
LSTM 2 generiert Ausgabe Wort für Wort aus s

(Sutskever et al. 2014)
<table>
<thead>
<tr>
<th>Method</th>
<th>test BLEU score (ntst14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bahdanau et al. [2]</td>
<td>28.45</td>
</tr>
<tr>
<td>Baseline System [29]</td>
<td>33.30</td>
</tr>
<tr>
<td>Single forward LSTM, beam size 12</td>
<td>26.17</td>
</tr>
<tr>
<td>Single reversed LSTM, beam size 12</td>
<td>30.59</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 1</td>
<td>33.00</td>
</tr>
<tr>
<td>Ensemble of 2 reversed LSTMs, beam size 12</td>
<td>33.27</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 2</td>
<td>34.50</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 12</td>
<td><strong>34.81</strong></td>
</tr>
</tbody>
</table>

Top-System in WMT14: BLEU = 37.0
Baseline + LSTM-Rescoring: BLEU = 36.5

Training von LSTM auf 300M Wörter Bitext dauert 10 Tage

(Sutskever et al. 2014)
Ergebnisse

Zweidimensionale Projektion der Endzustände für verschiedene Sätze.
To our surprise, the sequence-to-sequence model matched the BerkeleyParser that produced the annotation, having achieved an F1 score of 90.5 on the test set (section 23 of the WSJ).

We suspected that the attention model of Bahdanau et al. [2] might be more data efficient and we found that it is indeed the case. We trained a sequence-to-sequence model with attention on the small human-annotated parsing dataset and were able to achieve an F1 score of 88.3 on section 23 of the WSJ without the use of an ensemble and 90.5 with an ensemble, which matches the performance of the BerkeleyParser (90.4) when trained on the same data.

Finally, we constructed a second artificial dataset consisting of only high-confidence parse trees, as measured by the agreement of two parsers. We trained a sequence-to-sequence model with attention on this data and achieved an F1 score of 92.5 on section 23 of the WSJ – a new state-of-the-art. This result did not require an ensemble, and as a result, the parser is also very fast. An ensemble further improves the score to 92.8.

2 LSTM+A Parsing Model

Let us first recall the sequence-to-sequence LSTM model. The Long Short-Term Memory model of [5] is defined as follows. Let $x_t$, $h_t$, and $m_t$ be the input, control state, and memory state at timestep $t$. Given a sequence of inputs $(x_1,\ldots,x_T)$, the LSTM computes the $h$-sequence $(h_1,\ldots,h_T)$ and the $m$-sequence $(m_1,\ldots,m_T)$ as follows.

$$i_t = \text{sigm}(W_1 x_t + W_2 h_t)$$

$$i_0_t = \text{tanh}(W_3 x_t + W_4 h_t)$$

$$f_t = \text{sigm}(W_5 x_t + W_6 h_t)$$

$$o_t = \text{sigm}(W_7 x_t + W_8 h_t)$$

$$m_t = m_1_t f_t + i_t i_0_t$$

$$h_t = o_t \cdot h_t$$

The operator $\cdot$ denotes element-wise multiplication, the matrices $W_1,\ldots,W_8$ and the vector $h_0$ are the parameters of the model, and all the nonlinearities are computed element-wise.

In a deep LSTM, each subsequent layer uses the $h$-sequence of the previous layer for its input sequence $x$. The deep LSTM defines a distribution over output sequences given an input sequence:

$$P(B|A) = \prod_{t=1}^T \text{softmax}(W_o \cdot h^T_{A} + b)$$

where $B_t$, $A_t$, and $B$ denote the output, input, and input sequences, respectively.
### Ergebnisse

<table>
<thead>
<tr>
<th>Parser</th>
<th>Training Set</th>
<th>WSJ 22</th>
<th>WSJ 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline LSTM+D</td>
<td>WSJ only</td>
<td>&lt; 70</td>
<td>&lt; 70</td>
</tr>
<tr>
<td>LSTM+A+D</td>
<td>WSJ only</td>
<td>88.7</td>
<td>88.3</td>
</tr>
<tr>
<td>LSTM+A+D ensemble</td>
<td>WSJ only</td>
<td>90.7</td>
<td>90.5</td>
</tr>
<tr>
<td>baseline LSTM</td>
<td>BerkeleyParser corpus</td>
<td>91.0</td>
<td>90.5</td>
</tr>
<tr>
<td>LSTM+A</td>
<td>high-confidence corpus</td>
<td>93.3</td>
<td>92.5</td>
</tr>
<tr>
<td>LSTM+A ensemble</td>
<td>high-confidence corpus</td>
<td><strong>93.5</strong></td>
<td><strong>92.8</strong></td>
</tr>
<tr>
<td>Petrov et al. (2006) [12]</td>
<td>WSJ only</td>
<td>91.1</td>
<td>90.4</td>
</tr>
<tr>
<td>Zhu et al. (2013) [13]</td>
<td>WSJ only</td>
<td>N/A</td>
<td>90.4</td>
</tr>
<tr>
<td>Petrov et al. (2010) ensemble [14]</td>
<td>WSJ only</td>
<td>92.5</td>
<td>91.8</td>
</tr>
<tr>
<td>Zhu et al. (2013) [13]</td>
<td>semi-supervised</td>
<td>N/A</td>
<td>91.3</td>
</tr>
<tr>
<td>McClosky et al. (2006) [16]</td>
<td>semi-supervised</td>
<td>92.4</td>
<td>92.1</td>
</tr>
</tbody>
</table>

Table 1: F1 scores of various parsers on the development and test set. See text for discussion.

First, let us remark that our training setup differs from those reported in previous works. To the best of our knowledge, no standard parsers have ever been trained on datasets numbering in the hundreds of millions of tokens, and it would be hard to do due to efficiency problems. We therefore cite the semi-supervised results, which are analogous in spirit but use less data.

Table 1 shows performance of our models on the top and results from other papers at the bottom. We compare to variants of the BerkeleyParser that use self-training on unlabeled data [15], or built an ensemble of multiple parsers [14], or combine both techniques. We also include the best linear-time parser in the literature, the transition-based parser of [13].

It can be seen that, when training on WSJ only, a baseline LSTM does not achieve any reasonable score, even with dropout and early stopping. But a single attention model gets to 88.3 and an ensemble of 5 LSTM+A+D models achieves 90.5 matching a single-model BerkeleyParser on WSJ 23.

When trained on the large high-confidence corpus, a single LSTM+A model achieves 92.5 and so outperforms not only the best single model, but also the best ensemble result reported previously. An ensemble of 5 LSTM+A models further improves this score to 92.8.

Generating well-formed trees.

The LSTM+A model trained on WSJ dataset only produced malformed trees for 25 of the 1700 sentences in our development set (1.5% of all cases), and the model trained on full high-confidence dataset did this for 14 sentences (0.8%). In these few cases where LSTM+A outputs a malformed tree, we simply add brackets to either the beginning or the end of the tree in order to make it balanced. It is worth noting that all 14 cases where LSTM+A produced unbalanced trees were sentences or sentence fragments that did not end with proper punctuation.

There were very few such sentences in the training data, so it is not a surprise that our model cannot deal with them very well.

**Score by sentence length.**

An important concern with the sequence-to-sequence LSTM was that it may not be able to handle long sentences well. We determine the extent of this problem by partitioning the development set by length, and evaluating BerkeleyParser, a baseline LSTM model without attention, and LSTM+A on sentences of each length. The results, presented in Figure 3, are surprising. The difference between the F1 score on sentences of length upto 30 and that upto 70 is 1.3 for the BerkeleyParser, 1.7 for the baseline LSTM, and 0.7 for LSTM+A. So already the baseline LSTM has similar performance to the BerkeleyParser, it degrades with length only slightly.

(Vinyals et al. 2015)
Zusammenfassung

● Neuronale Netze:
  ▶ alte Technologie
  ▶ in Sprach- und Bilderkennung dramatische Erfolge
  ▶ “next big thing” in der Computerlinguistik

● In der Praxis:
  ▶ stabile, schnelle Frameworks wie z.B. Theano sind
    erstaunlich einfach zu verwenden
  ▶ Probieren Sie es aus!