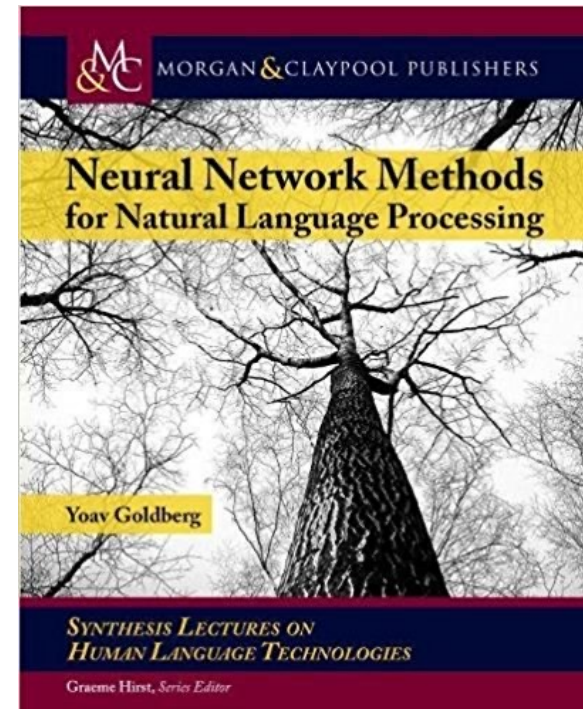


Deep Learning for NLP

Hopes and chances

This lecture is based on:

- The excellent book (Must read!) by Yoav Goldberg
- Various papers that have improved the current state of the art
- Own assessment



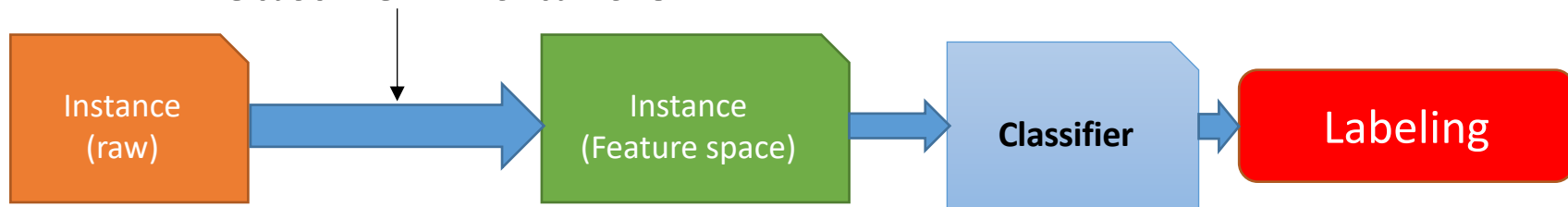
Agenda

1. Why Neural Networks?
 2. The phases of the deep learning move-in
 3. Neural Networks in NLP
- } Mixed!

Why Neural Networks?

- The standard procedure of classification...

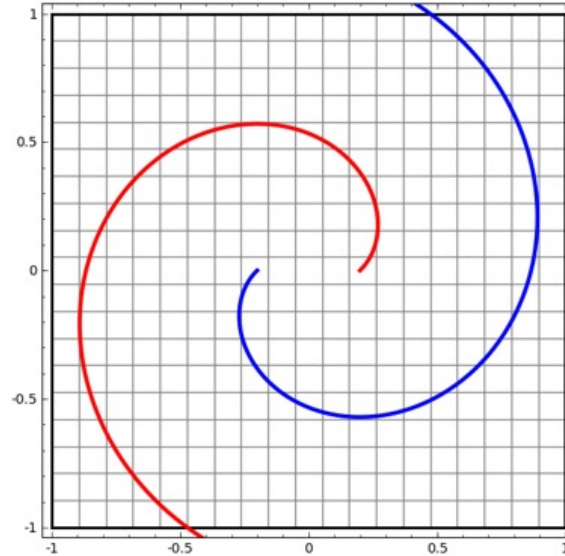
Feature-Extraktion



- Most classifiers are linear functions:
 - Maximum Entropy (and CRF)
 - Naïve Bayes
 - Support Vector Machines

Why Neural Networks?

- What if our data is nonlinear in nature?



Why Neural Networks?

- What if our data is nonlinear in nature?
- Traditional approaches:
 - Put the feature vector into a function F ("e.g., kernel function") and use the new feature vector for classification
 - Example:

$$\vec{x} = \begin{pmatrix} 0.3 \\ 1.5 \end{pmatrix} \text{ use } F(\vec{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 \cdot \log x_2 \end{pmatrix}$$

- Then we obtain:

$$\overrightarrow{x_F} = \begin{pmatrix} 0.3 \\ 1.5 \\ 0.91 \end{pmatrix}$$

Why Neural Networks?

- What if our data is nonlinear in nature?
- Traditional approaches:
 - Put the feature vector into a function F ("e.g., kernel function") and use the new feature vector for classification
 - ➔ How does the needed function F look?
 - ➔ Depending on the problem
 - Must be found manually
 - ➔ Kernel classifiers are extremely memory and runtime intensive!
(see lecture: AI)

Why Neural Networks?

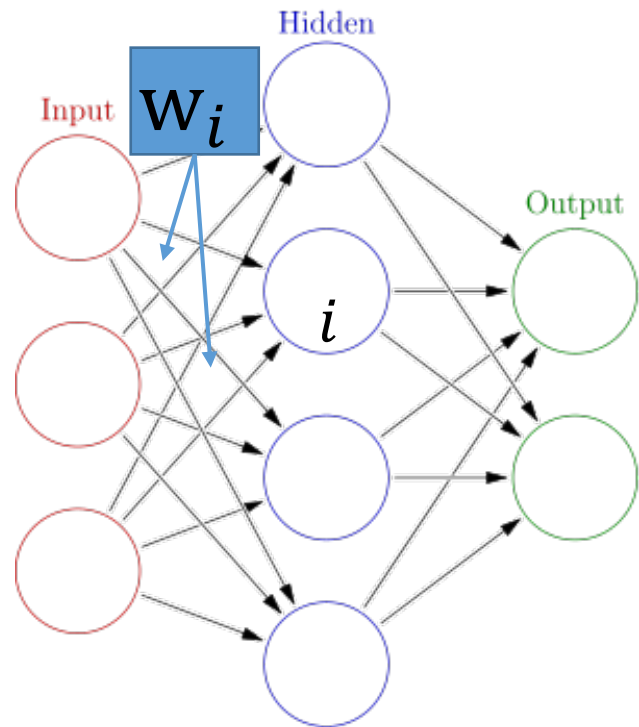
- What function do neural networks describe?

$$F(x) = \dots (\dots \phi(w_i^T x + b_i))$$

Nested many
times!

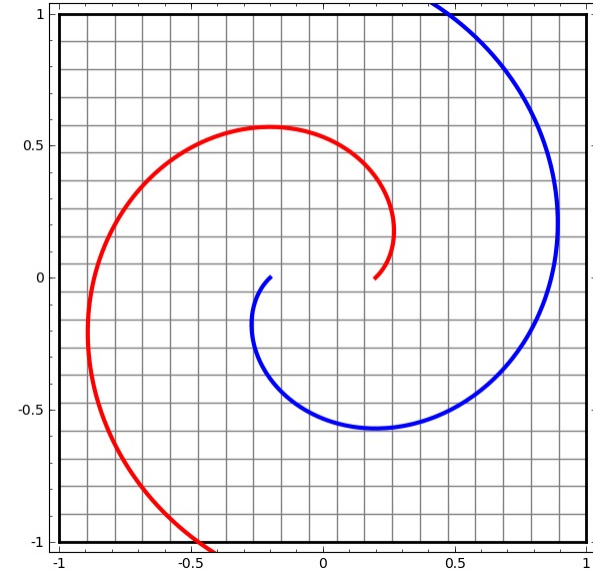
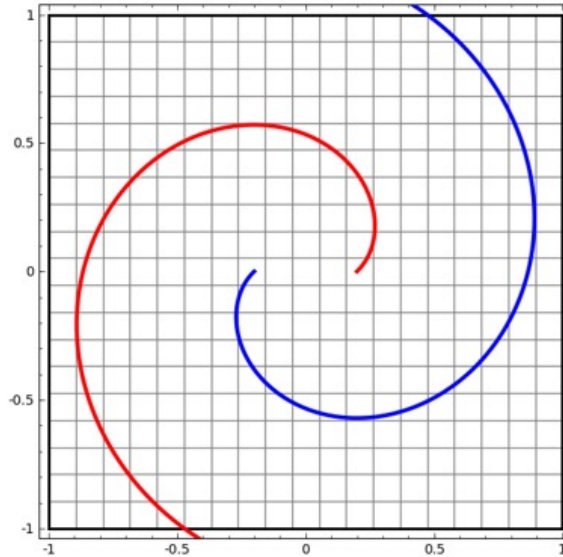
For $\phi(x) = \tanh(x), \text{sig}(x)$

- ➔ The network can approximate ANY function
(**Note:** With arbitrary number of layers!)



Why Neural Networks?

- What if our data is nonlinear in nature?

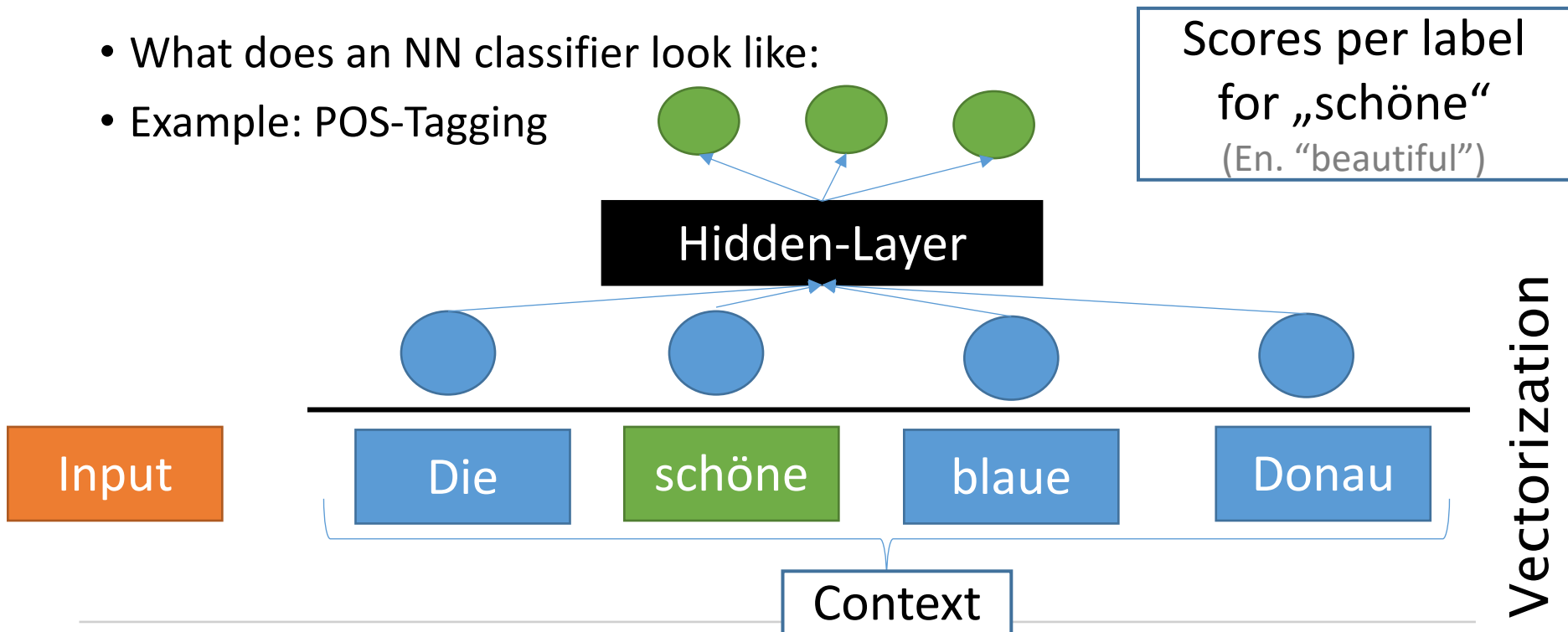


Why Neural Networks?

- First advantage:
 - Architectures with (at least) one hidden layer learn their feature transformation themselves!
- ➔ Better results with the same features?

Why Neural Networks?

- What does an NN classifier look like:
- Example: POS-Tagging



1. Phase in the Arrival of Deep Learning

- Instead of maximum entropy classifiers (CRF) and SVMs:
 - ➔ use feed forward architectures
 - Keep all features ("Profit from the learned kernel")

Method	POS-Tagging: F1	Named-Entity-Recognition: F1
CRF (Baseline)	97.3%	83.02%
Feedforward + CRF	96.37%	81.47%

- ➔ Existing models of text mining often already very good!
- ➔ Requires more than just changing the classifier

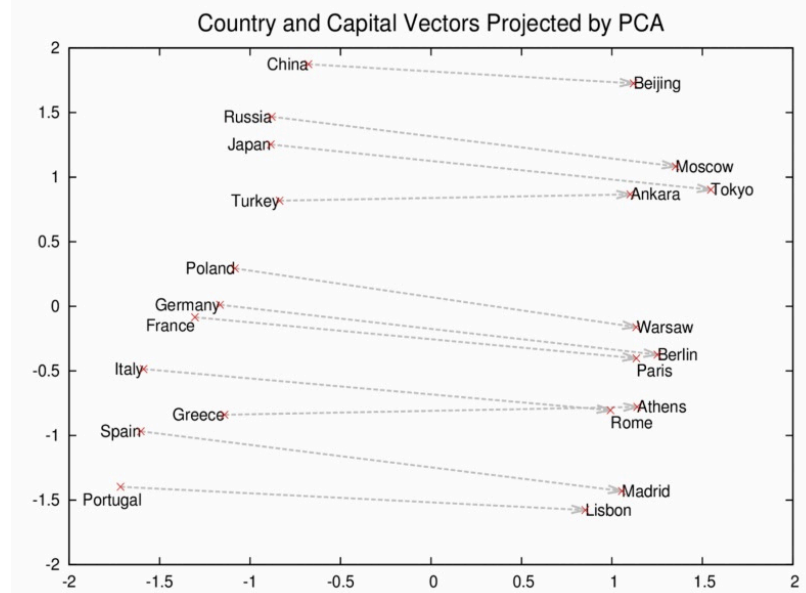
The Breakthrough: Embeddings and RNN

- Replacing the classifier with feed-forward networks hardly improves the state of the art!
 - ...Until "Collobert and Mikolov" put the embeddings in focus
- ➔ Leads to the improvement of the state of the art in almost all areas in a short time!



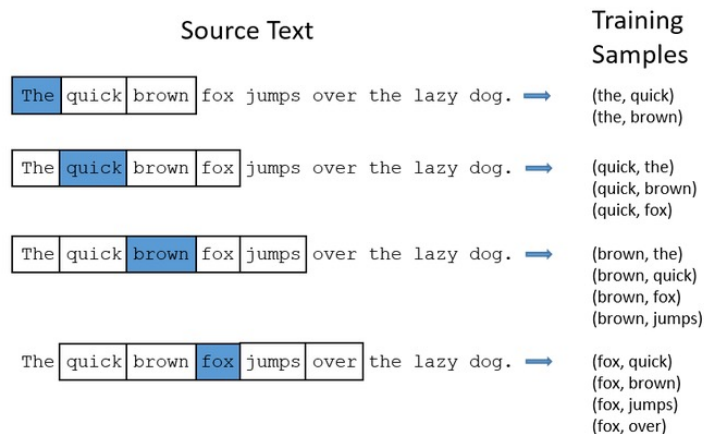
The Breakthrough: Embeddings and RNN

- Neural-Embeddings:
- Map words into a vector space
- Shortly after:
 - POS-Tag
 - Dependency labels
 - Phrases
 - ...



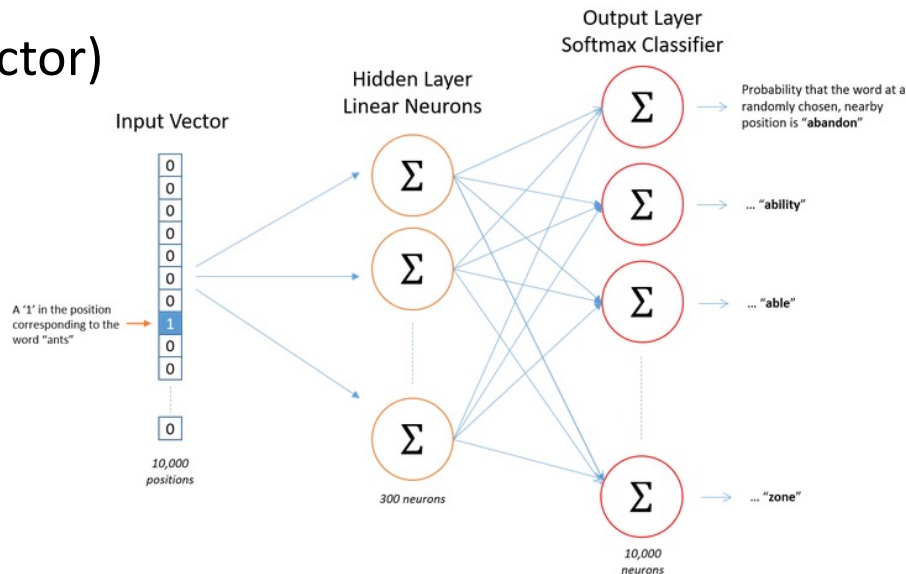
Neural Embeddings:

- E.g. Skip-Gram Word2Vec
- Idea: Input is a word (as a 1-hot vector)
 - ➔ Predict a word from its context



Neural Embeddings:

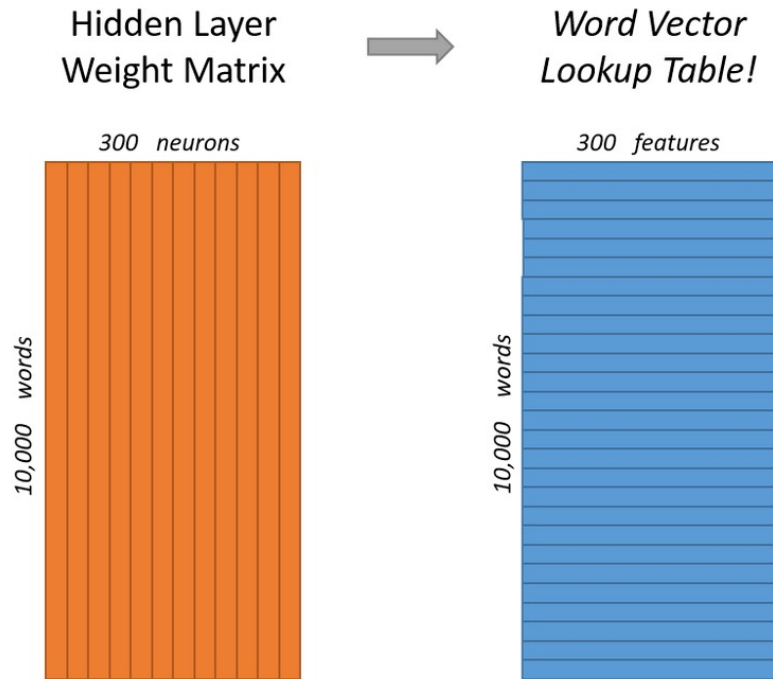
- E.g. Skip-Gram Word2Vec
- Idea: Input is a word (as a 1-hot vector)
 - ➔ Predict a word from its context
- Use a simple network architecture (efficiency!)
- The entries of the hidden layer matrix form our vectors



Neural Embeddings:

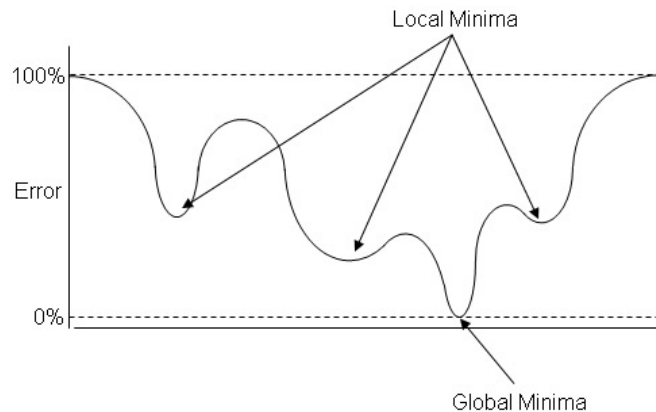
- The weights of the hidden layers (also called "projection layers") serve as a kind of dictionary

$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$



The breakthrough: Embeddings and RNN

- How embeddings help:
 1. Input is smaller (e.g. 100 dimensions per word instead of 50,000).
→ Fewer parameters → less overfitting!
 2. Better starting position in optimization!



The breakthrough: Embeddings and RNN

- Embeddings instead of 1-hot inputs:

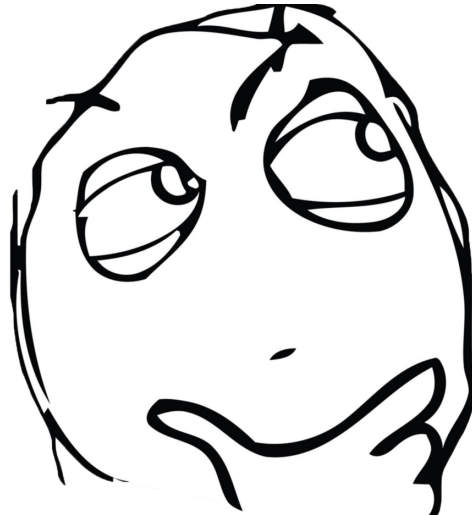
Method	POS-Tagging: F1	Named-Entity-Recognition: F1
CRF (Baseline)	97.3%	83.02%
Feedforward + CRF	96.37%	81.47%
CRF (Baseline) +Embeddings	97.45%	86.13%
Feedforward + CRF +Embeddings	97.29%	88.67%

➔ Better, but classical methods also benefit!

The breakthrough: Embeddings and RNN

- Change the architecture:
Want to process sentences!

→ How does this work?



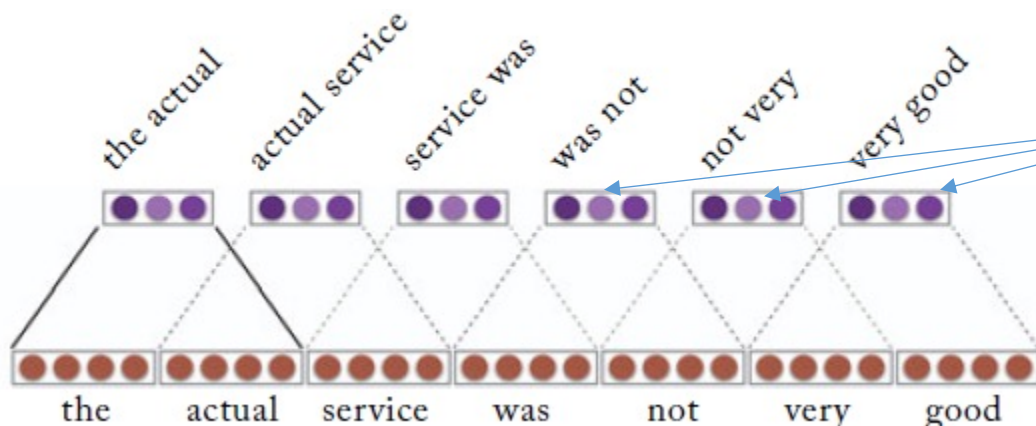
1-D Convolution

- There is a particular challenge with texts....
- For example, if we want to determine POS tags of a sentence
- Each sentence has a different number of words
- Our networks require a fixed input size....
- 2 Approaches:
 1. 1-D Convolution
 2. Recurrent networks
- Both approaches are based on the same principle




1-D Convolution

- There is a particular challenge with texts....
- For example, if we want to determine POS tags of a sentence

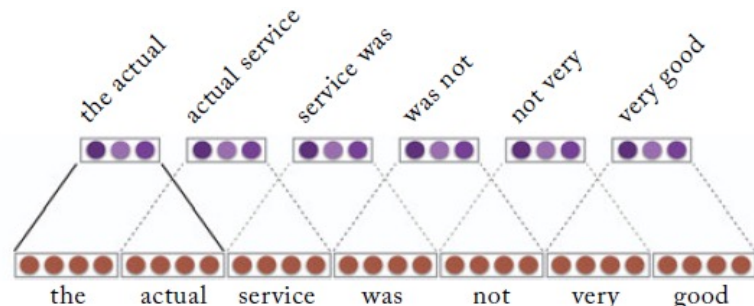


➔ The network becomes more robust against word swapping

1-D Convolution

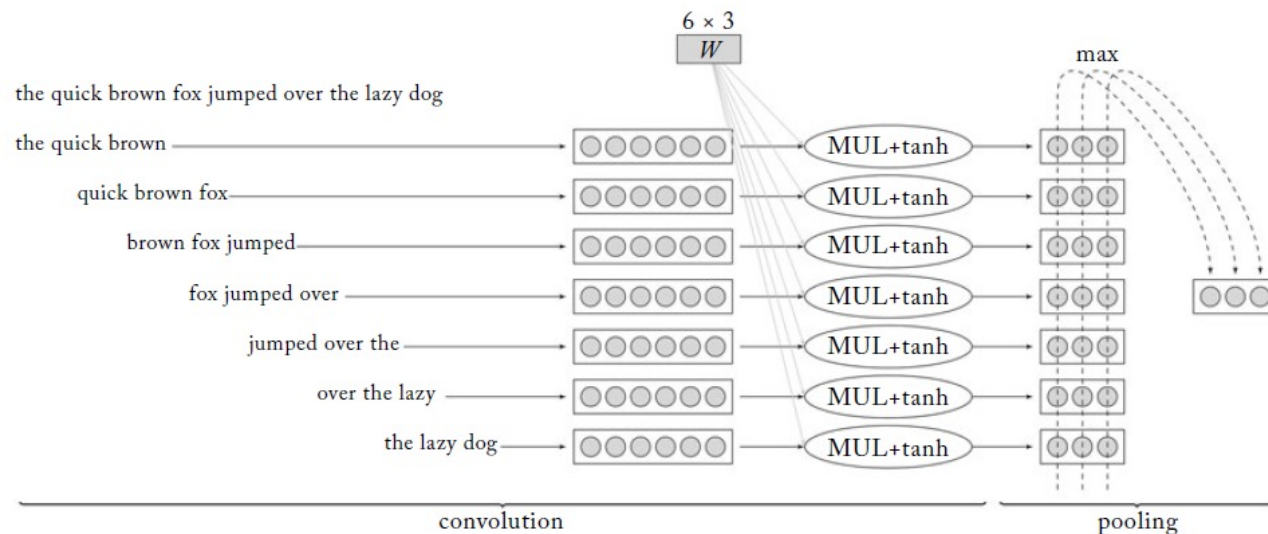
- However, sentences of different lengths generate different numbers of "segments"  here as well

➔ Use "pooling" to map to a fixed length



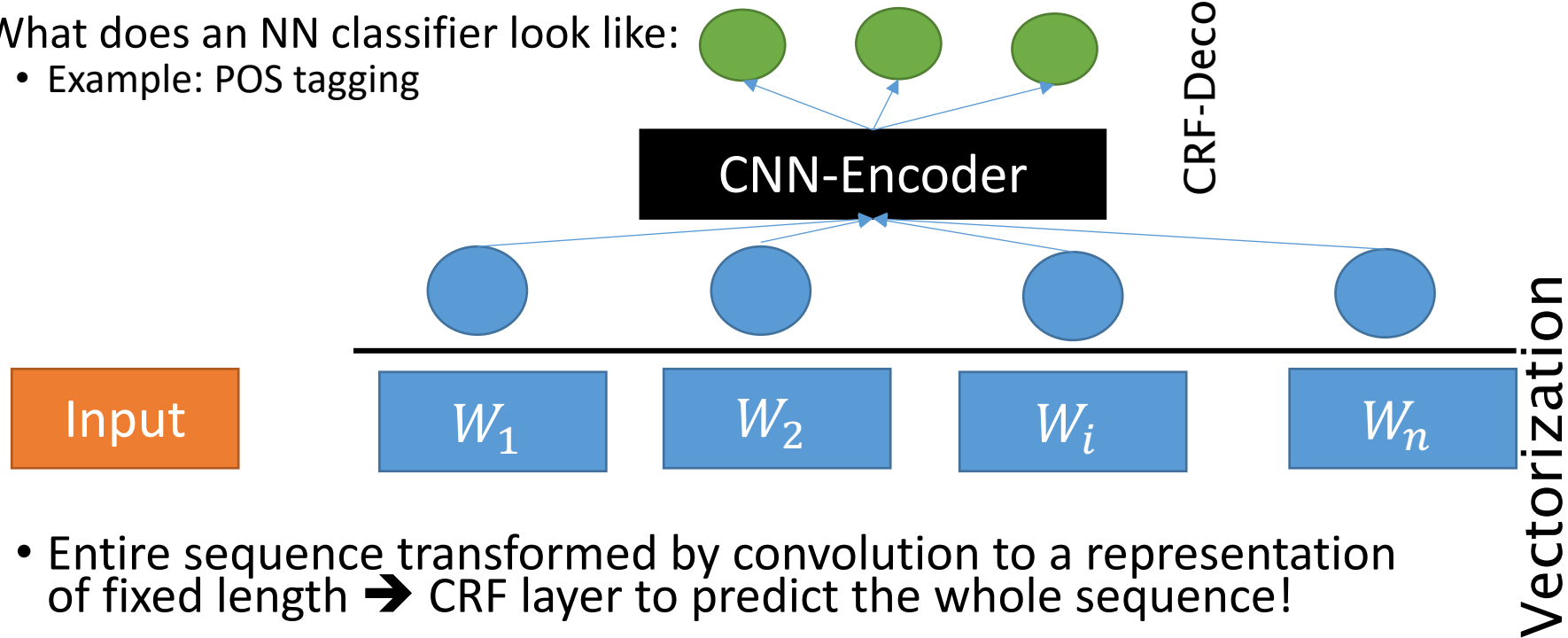
Pooling

- Select only the "interesting" weights from all channels
- Max-Pooling
- Average Pooling



1-D Convolution Classifier

- What does an NN classifier look like:
 - Example: POS tagging



- Entire sequence transformed by convolution to a representation of fixed length → CRF layer to predict the whole sequence!

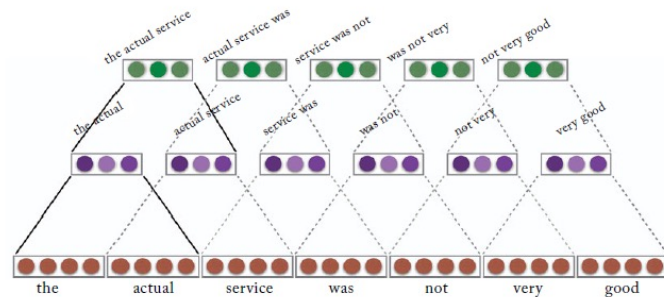
The Breakthrough: Embeddings and RNN

- Embeddings for everything have been created since then...
- ➔ How to create embeddings for phrases/sentences or paragraphs?



The Breakthrough: Embeddings and RNN

- How to create embeddings for phrases/sentences or paragraphs?
- 1-D Convolution uses the local context
- Hierarchical 1-D Convolution enlarges this more and more



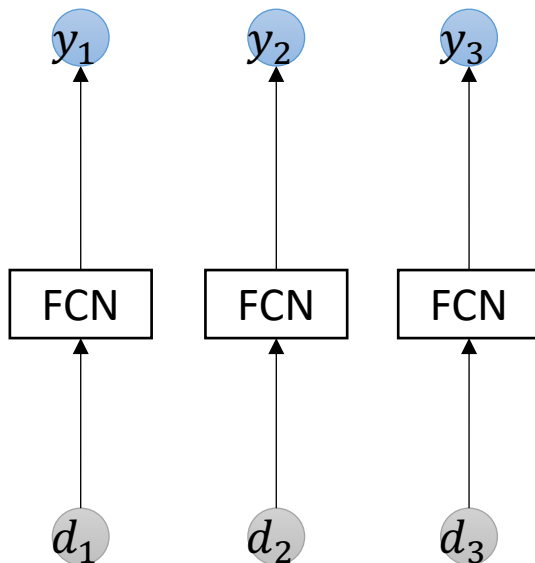
➔ What if we want to use everything as context?

➔ Use of recurrent networks, especially LSTMs

Recurrent Networks

- Another way to dynamically convert long text into a fixed representation
- ➔ thereby the whole already seen context can be used

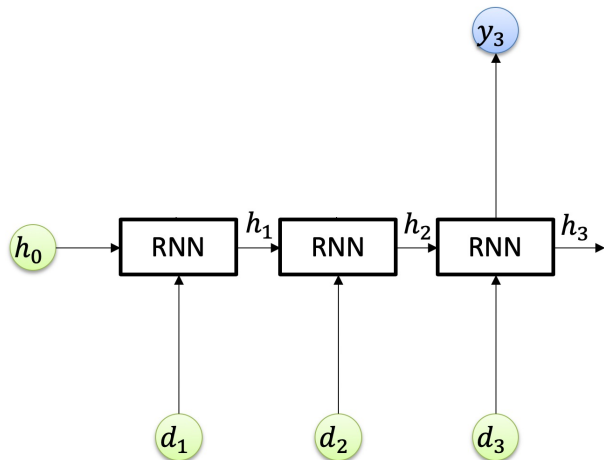
From FCNs to RNNs



$$\begin{aligned} y_1 &= f(Wd_1 + b) \\ y_2 &= f(Wd_2 + b) \\ y_3 &= f(Wd_3 + b) \end{aligned}$$

- Each word is considered individually
 - How do we retain information over multiple words?
- **Add connections between the networks!**

From FCNs to RNNs



⇒ „Unrolled“ RNN

Idea

- „State“ h is transferred between inputs.
- Get output y from current state h
- Same weights for all time steps!

$$h_1 = \sigma_h(W_h d_1 + U_h h_0)$$

$$h_2 = \sigma_h(W_h d_2 + U_h h_1)$$

$$h_3 = \sigma_h(W_h d_3 + U_h h_2)$$

$$y_3 = \sigma_y(W_y h_3 + b_y)$$

Recurrent Networks

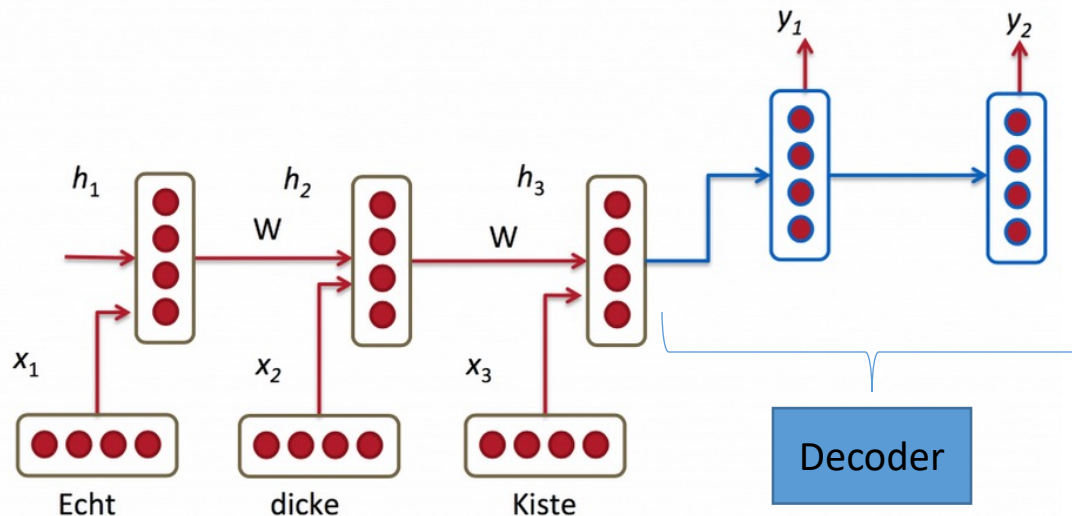
- A recurrent network has at each time step
 - An input d
 - An output y
 - A state h
- ➔ Currently most successful models "LSTM" and "GRU"
- ➔ see lecture "Machine Learning for NLP"

Recurrent Networks

- There are now (at least) 3 ways to use the recurrent nets for POS tagging
 1. Use the output y at each time step to determine the label (\rightarrow like MEMMs)
 2. Encode the entire input into a representation (like CNN) and predict the entire output using a CRF layer (like CRF)
 3. Encode the entire input into one representation and decode the labels from this representation step by step (approximately MEMM with global feature templates)

Recurrent Networks

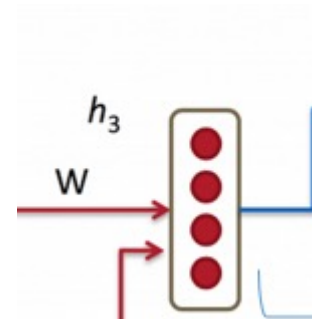
3. Encode the entire input into one representation and decode the labels from this representation step by step (approximately MEMM with global feature templates)



Recurrent Networks

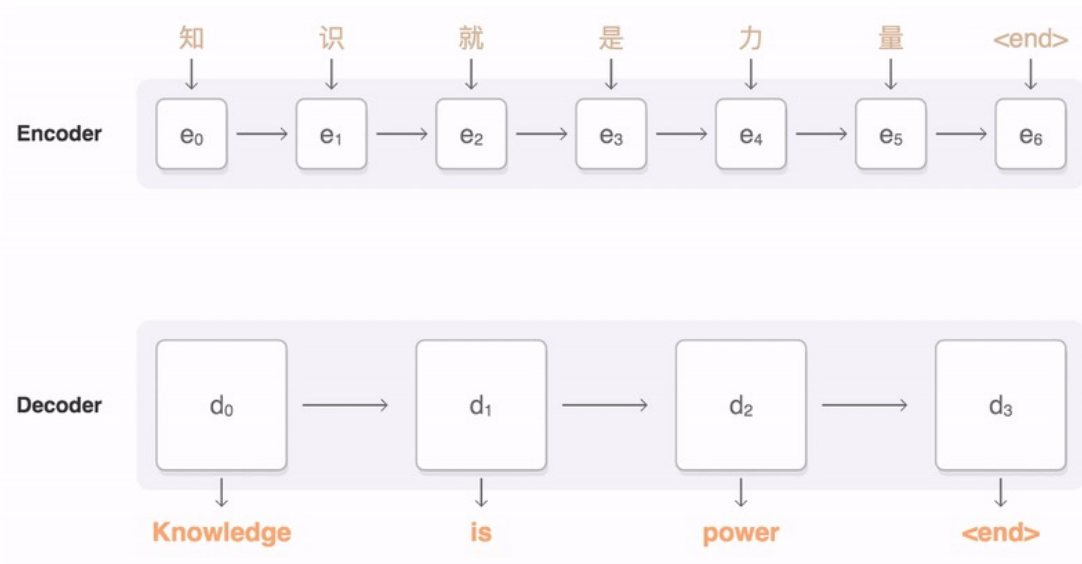
Problem:

- In the encoder-decoder structure the network is forced to retrieve the entire output from a single representation:
- The network is not always successful
- Solution: "Attention mechanism"



Attention Mechanism

- Add additional connections between the input and the output



→ Weighting of
the individual time steps!

Animation from: <https://github.com/google/seq2seq>

The Breakthrough: Embeddings and RNN

- Embeddings and LSTM:

Method	POS-Tagging: F1	Named-Entity-Recognition: F1
CRF (Baseline)	97.3%	83.02%
Feedforward + CRF	96.37%	81.47%
CRF (Baseline) +Embeddings	97.45%	86.13%
Feedforward + CRF +Embeddings	97.29%	88.67%
LSTM-CRF	97.54%	88.36%
BILSTM-CRF	97.55%	88.83%

2. Phase in the Arrival of Deep Learning

- Use advantageous deep learning architectures
 - Embeddings
 - Recurrent networks
- ➔ BiLSTM-CRF: Improve State of the Art in almost all areas!
- What remains to be done?
 - ➔ There are almost always additional features used
 - ➔ Can these also be learned with?
 - ➔ Goal: „End-to-End“-Systems

2. Phase in the Arrival of Deep Learning

- What other features are included:
 - E.g. „Spelling-Features“

→ Our Word embeddings cannot learn features at character level!

→ Integrate character embeddings!

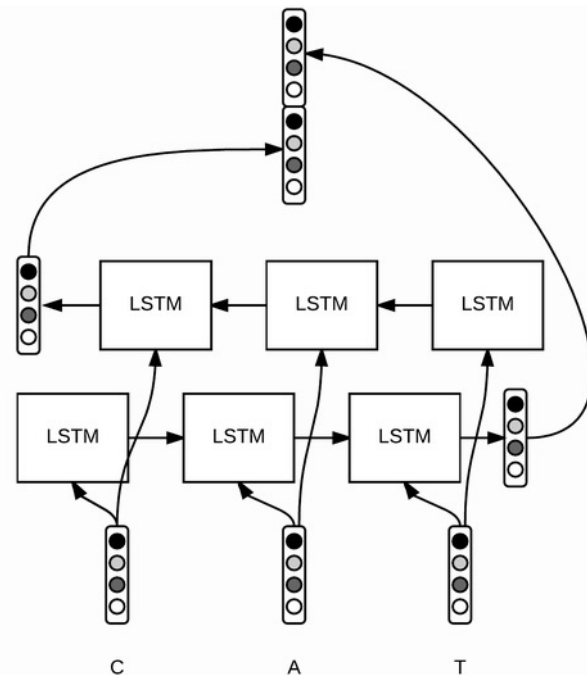
- whether start with a capital letter
- whether has all capital letters
- whether has all lower case letters
- whether has non initial capital letters
- whether mix with letters and digits
- whether has punctuation
- letter prefixes and suffixes (with window size of 2 to 5)
- whether has apostrophe end ('s)
- letters only, for example, I. B. M. to IBM
- non-letters only, for example, A. T. &T. to ..&

Character Neural Networks

- What other features are included:
 - E.g. „Spelling-Features“

➔ Our Word embeddings cannot learn features at character level!

- ➔ Integrate character embeddings!
- Either LSTM or CNN („CNN-CNN`s“ 😊)



3. Phase in the Arrival of Deep Learning

- Eliminate all external feature calculations and train your network „End-To-End“

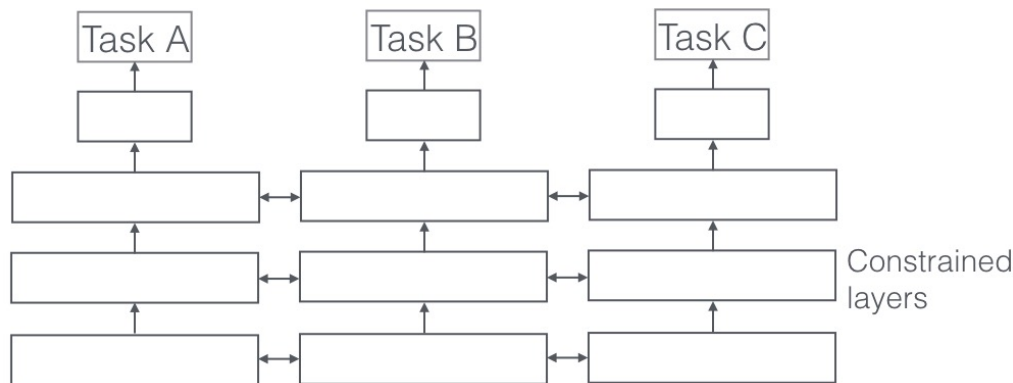
- „End-To-End“ Neural Parsing
- „End-To-End“ Neural NER
- „End-To-End“ Neural Coreference
- „End-To-End“ Neural Translation ...

➔ End-to-end allows to completely skip the pipeline architecture of classical NLP and thus does not have problems with errors of previous engines

4. Phase in the Arrival of Deep Learning

- Learn networks that handle multiple tasks simultaneously ("if suitable data is available", such a network can exploit dependencies between tasks!)

➔ Multi-Task Learning



4. Phase in the Arrival of Deep Learning

- Learn networks that handle multiple tasks simultaneously ("if suitable data is available", such a network can exploit dependencies between tasks!)
- Find a network architecture that can solve all our problems
 - Current: Transformer!
- Learn the network architecture completely, with nothing but a task description and data



Learn!

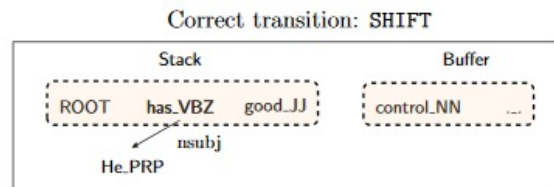
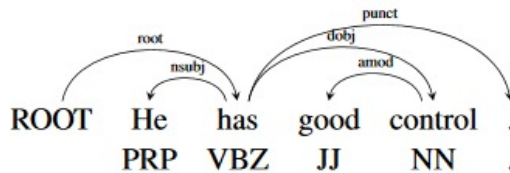
Neural Networks in NLP

Example applications

Neural Networks in NLP

- Dependency parsing: [A Fast and Accurate Dependency Parser using Neural Network]

- Greedy-Transition-based
- Arc-Standard
(SHIFT,LEFT-ARC,RIGHT-ARC)

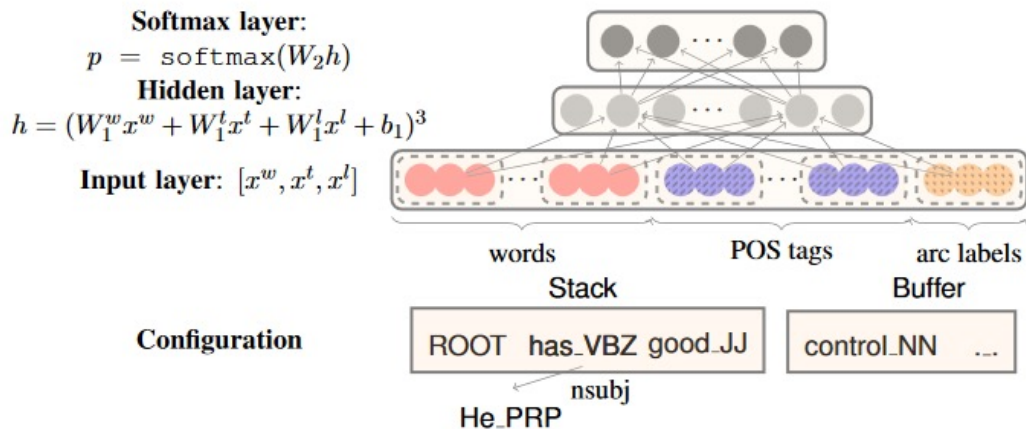


Transition	Stack	Buffer	A
	[ROOT]	[He has good control .]	∅
SHIFT	[ROOT He]	[has good control .]	
SHIFT	[ROOT He has]	[good control .]	
LEFT-ARC (nsubj)	[ROOT has]	[good control .]	AU nsubj(has,He)
SHIFT	[ROOT has good]	[control .]	
SHIFT	[ROOT has good control]	[.]	
LEFT-ARC (amod)	[ROOT has control]	[.]	AU amod(control,good)
RIGHT-ARC (dobj)	[ROOT has]	[.]	AU dobj(has,control)
...
RIGHT-ARC (root)	[ROOT]	[]	AU root(ROOT,has)

Neural Networks in NLP

- Simple feedforward network as classifier
- Word-Embeddings + POS-Embeddings

➔ Why better?



Neural Networks in NLP

→ Why better?

- Classic NLP uses the following features:

- These features are very sparse!
(rarely occur!)

→ Embeddings help to generalize!

Single-word features (9)

$s_1.w; s_1.t; s_1.wt; s_2.w; s_2.t;$
 $s_2.wt; b_1.w; b_1.t; b_1.wt$

Word-pair features (8)

$s_1.wt \circ s_2.wt; s_1.wt \circ s_2.w; s_1.wt s_2.t;$
 $s_1.w \circ s_2.wt; s_1.t \circ s_2.wt; s_1.w \circ s_2.w$
 $s_1.t \circ s_2.t; s_1.t \circ b_1.t$

Three-word features (8)

$s_2.t \circ s_1.t \circ b_1.t; s_2.t \circ s_1.t \circ lc_1(s_1).t;$
 $s_2.t \circ s_1.t \circ rc_1(s_1).t; s_2.t \circ s_1.t \circ lc_1(s_2).t;$
 $s_2.t \circ s_1.t \circ rc_1(s_2).t; s_2.t \circ s_1.w \circ rc_1(s_2).t;$
 $s_2.t \circ s_1.w \circ lc_1(s_1).t; s_2.t \circ s_1.w \circ b_1.t$

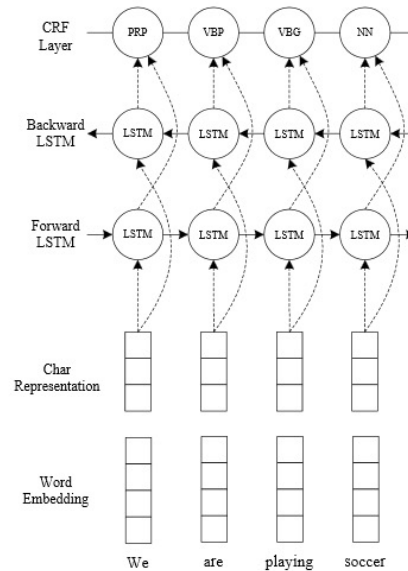
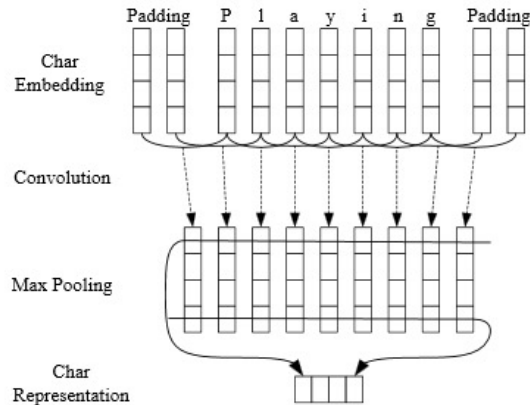
Neural Networks in NLP

- Evaluation on the Penn Treebank dataset

Parser	Dev		Test		Speed (sent/s)
	UAS	LAS	UAS	LAS	
standard	90.2	87.8	89.4	87.3	26
eager	89.8	87.4	89.6	87.4	34
Malt:sp	89.8	87.2	89.3	86.9	469
Malt:eager	89.6	86.9	89.4	86.8	448
MSTParser	91.4	88.1	90.7	87.6	10
Our parser	92.0	89.7	91.8	89.6	654

Neural Networks in NLP

- POS-Tagging/Named Entity Recognition: [End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF]



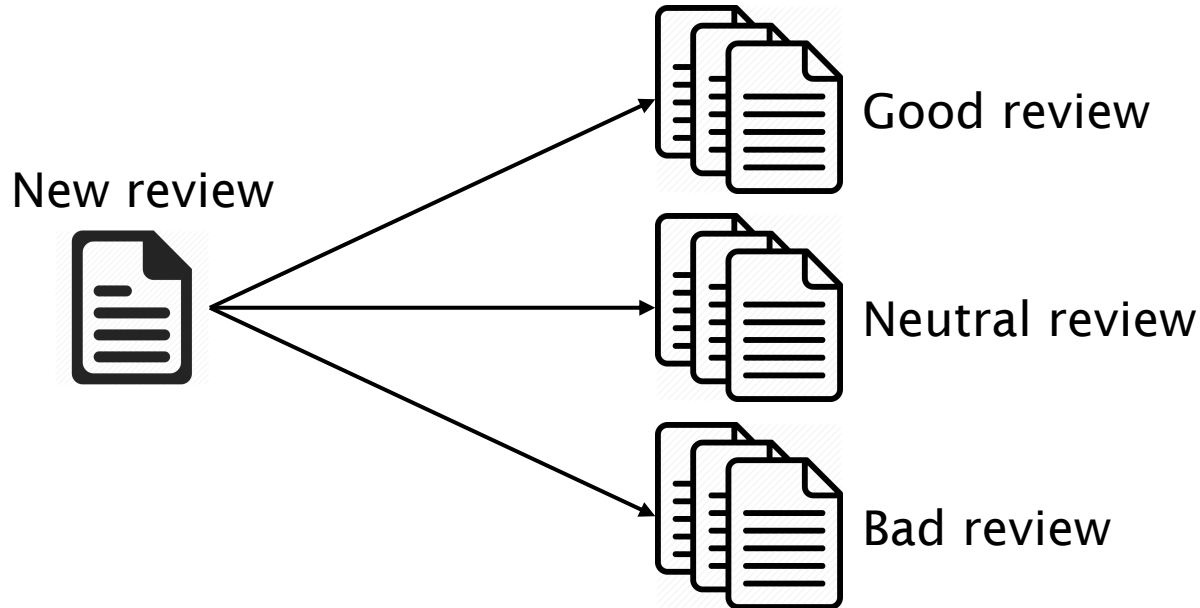
Neural Networks in NLP

- POS-Tagging/Named Entity Recognition: [End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF]
- Characters are "embedded" on a vector of fixed size using convolution and pooling
- This embedding, together with the (pre-trained) word embedding, is brought to a representation of a whole sentence using an LSTM
- CRF layer predicts entire sequence from this representation

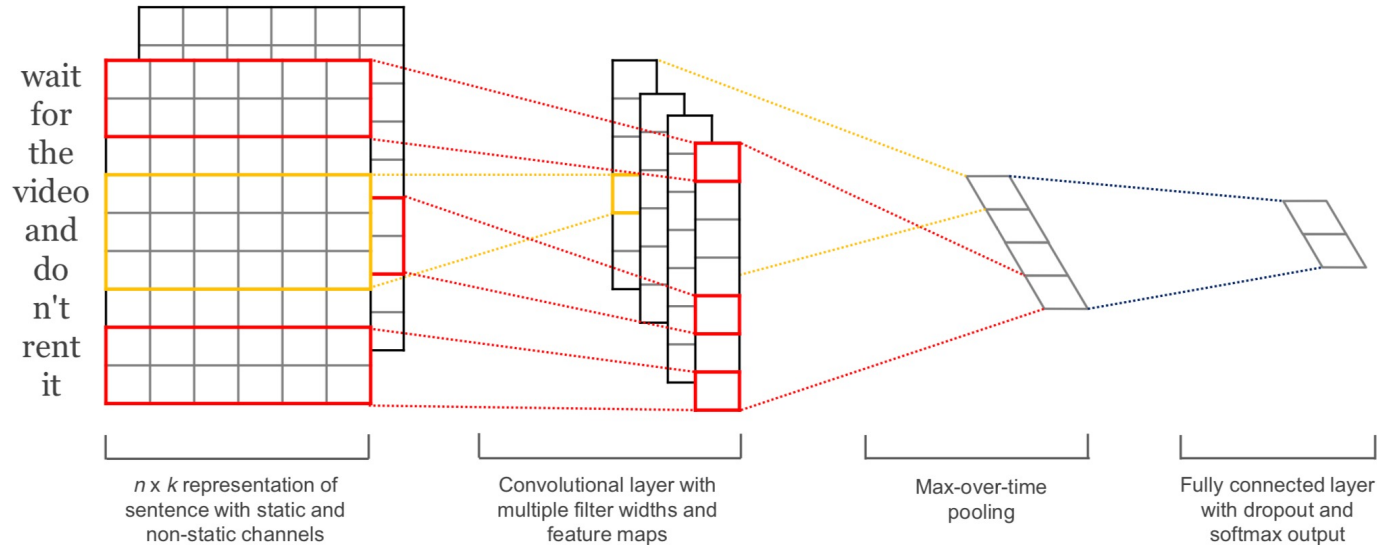
Model	POS		NER					
	Dev	Test	Dev			Test		
	Acc.	Acc.	Prec.	Recall	F1	Prec.	Recall	F1
BRNN	96.56	96.76	92.04	89.13	90.56	87.05	83.88	85.44
BLSTM	96.88	96.93	92.31	90.85	91.57	87.77	86.23	87.00
BLSTM-CNN	97.34	97.33	92.52	93.64	93.07	88.53	90.21	89.36
BRNN-CNN-CRF	97.46	97.55	94.85	94.63	94.74	91.35	91.06	91.21

Neural Networks in NLP

Text Classification - Sentiment Analysis:



Neural Networks in NLP



Convolutional Neural Networks for Sentence Classification
By Yoon Kim, 2014

Neural Networks in NLP

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	89.6
CNN-non-static	81.5	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	88.1	93.2	92.2	85.0	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	—	—	—	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	—	—	—	—
RNTN (Socher et al., 2013)	—	45.7	85.4	—	—	—	—
DCNN (Kalchbrenner et al., 2014)	—	48.5	86.8	—	93.0	—	—
Paragraph-Vec (Le and Mikolov, 2014)	—	48.7	87.8	—	—	—	—
CCAE (Hermann and Blunsom, 2013)	77.8	—	—	—	—	—	87.2
Sent-Parser (Dong et al., 2014)	79.5	—	—	—	—	—	86.3
NBSVM (Wang and Manning, 2012)	79.4	—	—	93.2	—	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	—	—	93.6	—	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	—	—	93.4	—	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	—	—	93.6	—	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	—	—	—	—	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	—	—	—	—	—	82.7	—
SVM _S (Silva et al., 2011)	—	—	—	—	95.0	—	—

Thank you for your attention!

