Recurrent Neural Networks

Daniël de Kok & Erhard Hinrichs

August 10, 2018

Motivation

Introduction

- So far: classification of disjoint data points representable as feature vectors.

Introduction

- So far: classification of disjoint data points representable as feature vectors.
- In natural language processing we often deal with sequences:
  - Characters
  - Tokens
  - Part-of-Speech tags
- If x^n is the n-th element of a sequence, we also want to take into account:
  - Preceding elements: $\mathbf{x}^1 \ldots \mathbf{x}^{n-2}, \mathbf{x}^{n-1}$
  - Preceding labeling choices: $y^1 \ldots y^{n-2}, y^{n-1}$

Example 1: part-of-speech tagging

(1) Hij hoorde/*vfin* de blikken vallen/*vinf* .
He heard the cans fall .

(Example from Prins, 2005)

- Dutch uses the same form for the infinitive and the third person plural of the verb *vallen* (*to fall*).

Example 1: part-of-speech tagging

(1) Hij hoorde/*vfin* de blikken vallen/*vinf* .
He heard the cans fall .

(Example from Prins, 2005)

- Dutch uses the same form for the infinitive and the third person plural of the verb *vallen* (*to fall*).
- To assign a proper tag for the verb *vallen*, the tagger needs to memorize whether the finite verb for this clause has already occured.

Example 1: part-of-speech tagging

(1) Hij hoorde/*vfin* de blikken vallen/*vinf* .
He heard the cans fall .

(Example from Prins, 2005)

- Dutch uses the same form for the infinitive and the third person plural of the verb *vallen* (*to fall*).
- To assign a proper tag for the verb *vallen*, the tagger needs to memorize whether the finite verb for this clause has already occured.
- A trigram tagger will often mistag such infinitives.

Example 1: part-of-speech tagging

The tagger should **not** use this information in a relative clause:

(2) Hij hoorde/*vfin* de blikken die van de tafel
He heard the cans that off the table
vallen/*vfin* .
fall .

He heard the cans that fall of the table

Example 1: part-of-speech tagging

However, starting a relative clause should not drop the information either:

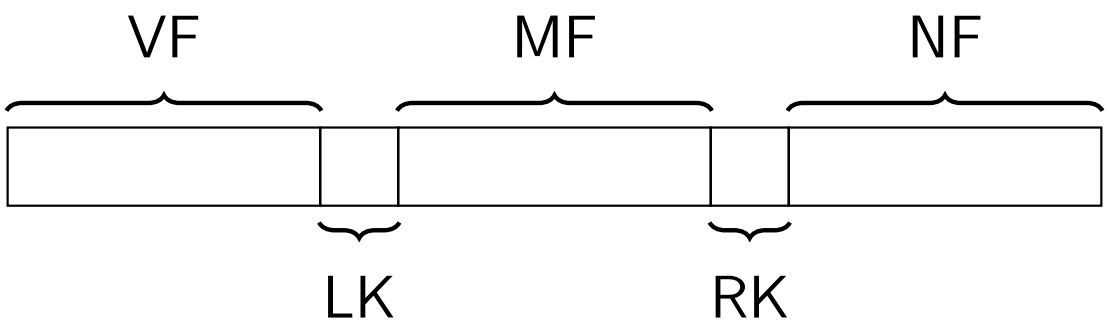
(3)

Hij	hoorde/*vfin*	de	blikken	,	die	Jan	aanstootte/*vfin*	,
He	heard	the	cans	,	that	Jan	nudged	,
vallen/*vinf*	.							
fall	.							

Example 2: topological fields

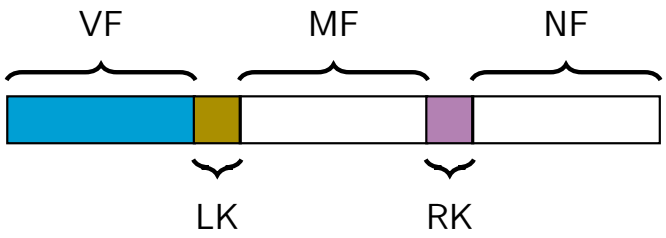
The **topological field model** has been used to account for regularities in word order across different clause types in many germanic languages (ie. German and Dutch).

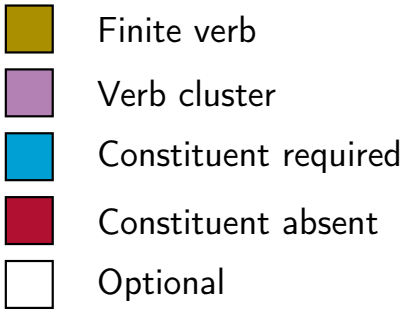
Example 2: topological fields



Example 2: topological fields

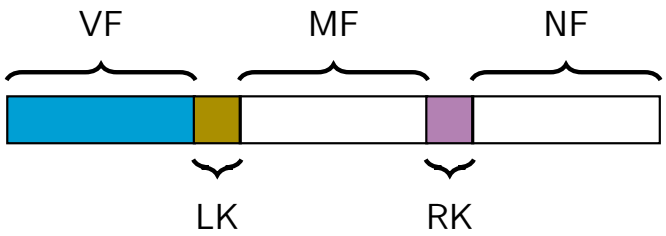
Declarative clause with auxiliary/modal verb:



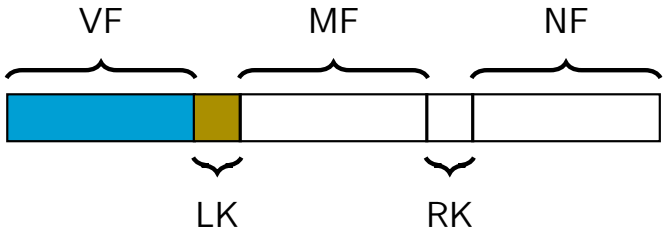


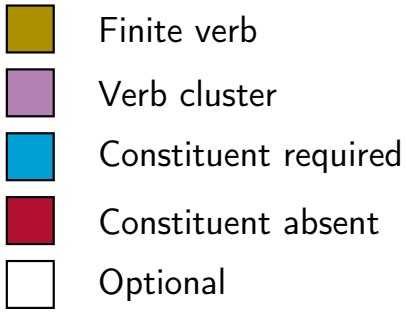
Example 2: topological fields

Declarative clause with auxiliary/modal verb:



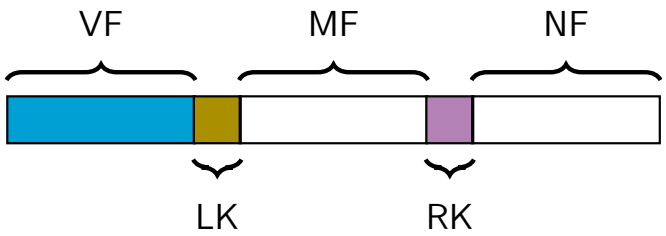
Declarative clause without auxiliary/modal verb:



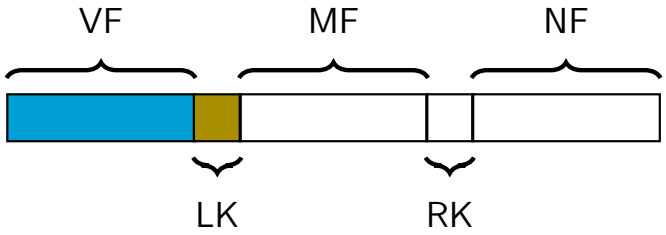


Example 2: topological fields

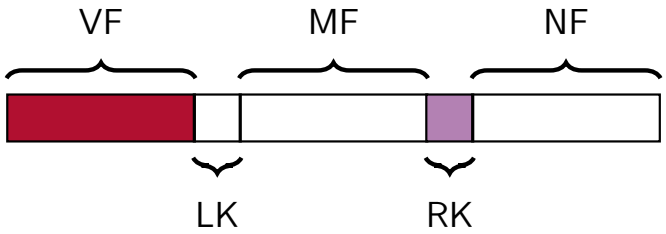
Declarative clause with auxiliary/modal verb:

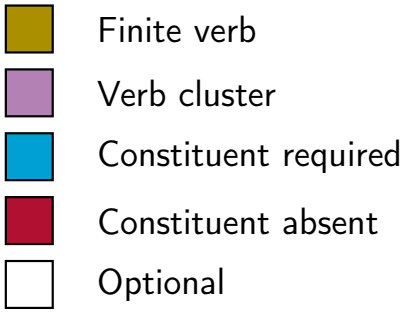


Declarative clause without auxiliary/modal verb:



Verb-final subordinate clause:





Example 2: topological fields

	VF	LK	MF	RK	NF
MC:	Gestern	hat	er häufiger	angerufen	als heute
	Yesterday	has	he more-often	called	than today
MC:	Er	ruft	häufig	an	
	He	calls	frequently	up	
SC:		der	noch häufiger	anruft	als er
		who	more often	calls	than him

Example 2: topological fields

Topological field analysis can be seen as a sequence labeling task:

(4)

Gestern	hat	er	häufiger	angerufen	als	heute
VF	LK	MF	MF	RK	NF	NF

Example 2: topological fields

Topological field analysis can be seen as a sequence labeling task:

(4)

Gestern	hat	er	häufiger	angerufen	als	heute
VF	LK	MF	MF	RK	NF	NF

- Preceding tokens are relevant: similarly to Dutch, V2 word order requires us to keep track of tokens that were seen earlier, such as verbs, separable verb particles, etc.
- Preceding classification choices are also relevant: e.g. the *MF* cannot precede the *VF*.

Why not feed-forward nets?

- Why not use feed-forward neural nets?
- Just add representations of the preceding tokens to the input.

Why not feed-forward nets?

- Why not use feed-forward neural nets?
- Just add representations of the preceding tokens to the input.
- Problems:
  - Sequences are variable-length, while feed-forward networks have a fixed-width input.

Why not feed-forward nets?

- Why not use feed-forward neural nets?
- Just add representations of the preceding tokens to the input.
- Problems:
  - Sequences are variable-length, while feed-forward networks have a fixed-width input.
  - Dependencies between tokens may have large distances.

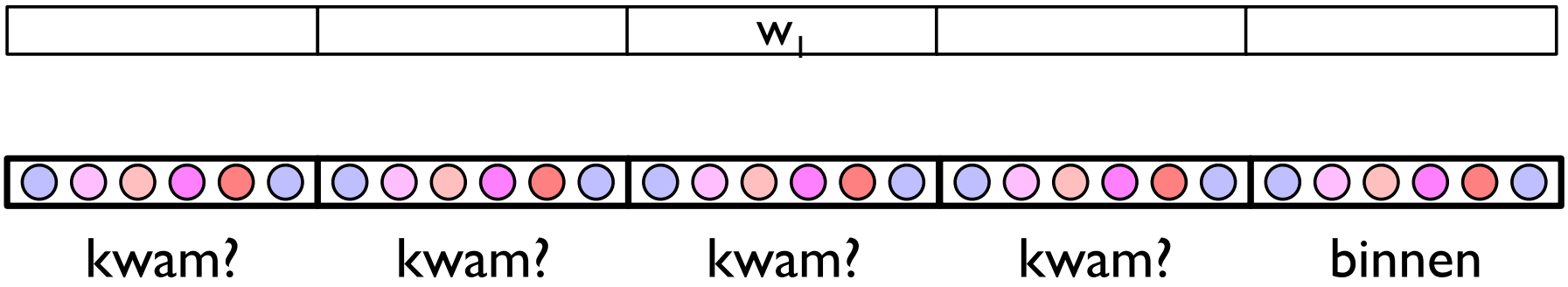
Why not feed-forward nets?

- Why not use feed-forward neural nets?
- Just add representations of the preceding tokens to the input.
- Problems:
  - Sequences are variable-length, while feed-forward networks have a fixed-width input.
  - Dependencies between tokens may have large distances.
  - Does not generalize well, since the network would have to learn separate weights for some interesting phenomenon occuring in every position.

Long-distance dependencies

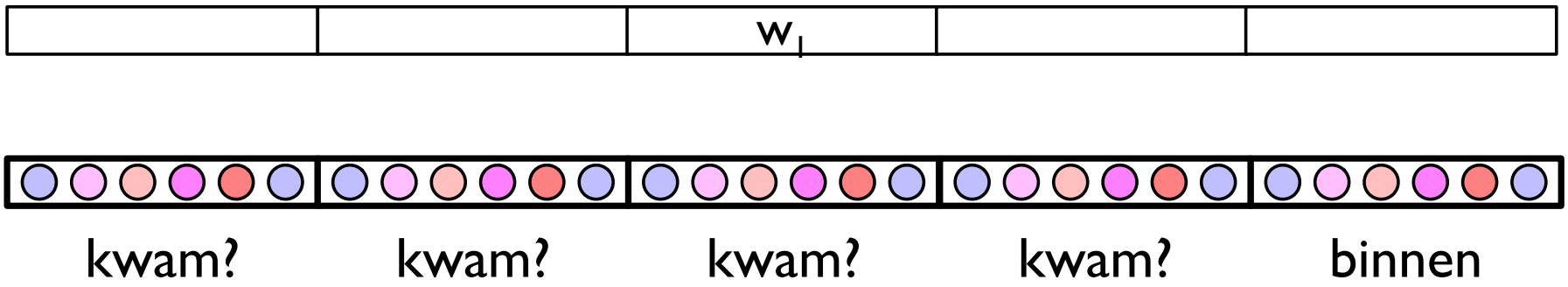
(5) Het kwam/*vfin* allemaal bij stukjes en brokjes bij de
It came all with bits and pieces at the
familie Balemans binnen/*part* .
family Balemans within .
It arrived at family Balemans in bits and pieces.

Lack of generalization



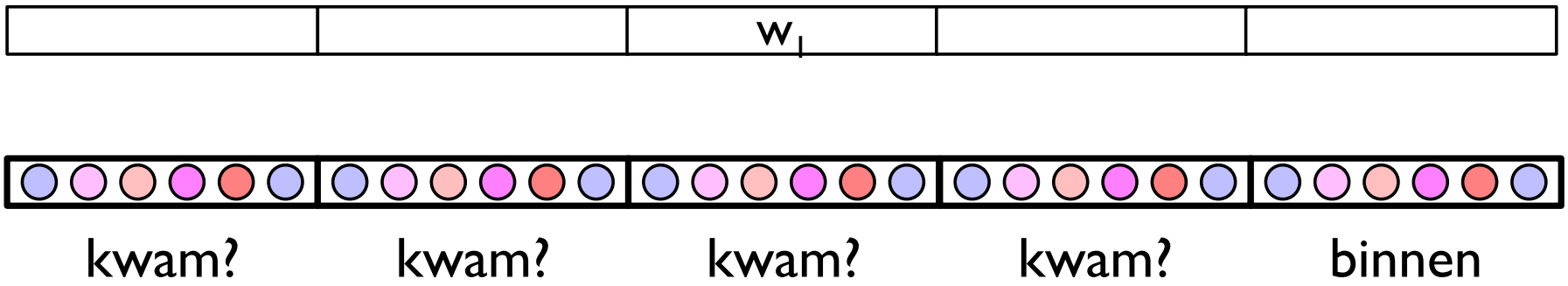
- The first hidden layer computes the dot product of the weight vector(s) and the input.

Lack of generalization



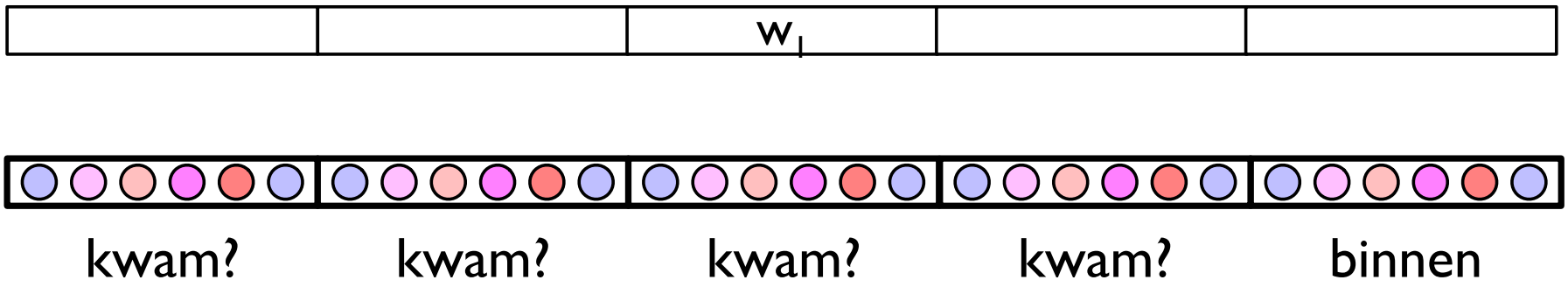
- The first hidden layer computes the dot product of the weight vector(s) and the input.
- Each component of the weight vector corresponds to a component of the input vector.

Lack of generalization



- The first hidden layer computes the dot product of the weight vector(s) and the input.
- Each component of the weight vector corresponds to a component of the input vector.
- Training needs to estimate weights for *kwam* occuring in each position separately.

Lack of generalization



- The first hidden layer computes the dot product of the weight vector(s) and the input.
- Each component of the weight vector corresponds to a component of the input vector.
- Training needs to estimate weights for *kwam* occuring in each position separately.
- Problem: data sparseness.

Basic principles of RNNs

Goal

- Our goal is to process sequences of linguistic units (characters, words, etc.).

Goal

- Our goal is to process sequences of linguistic units (characters, words, etc.).
- In processing an element from the sequence, we want use representations:
  - The current element;
  - previous elements.

Approach

- Consider a sequence $\boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$.

Approach

- Consider a sequence $\boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$.
- Each $\boldsymbol{x}^{<t>} \in \boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$ is called a **time step**.

Approach

- Consider a sequence $\boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$.
- Each $\boldsymbol{x}^{<t>} \in \boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$ is called a **time step**.
- $\boldsymbol{h}^{<i>}$ is a hidden layer at time step i.

Approach

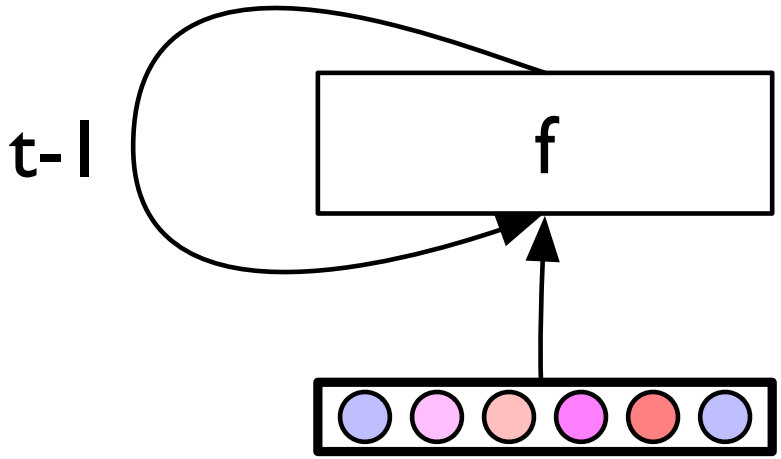
- Consider a sequence $\boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$.
- Each $\boldsymbol{x}^{<t>} \in \boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$ is called a **time step**.
- $\boldsymbol{h}^{<i>}$ is a hidden layer at time step i.
- We will use $\boldsymbol{h}^{<i>}$ as a representation of time step i and all preceding time steps.

Approach

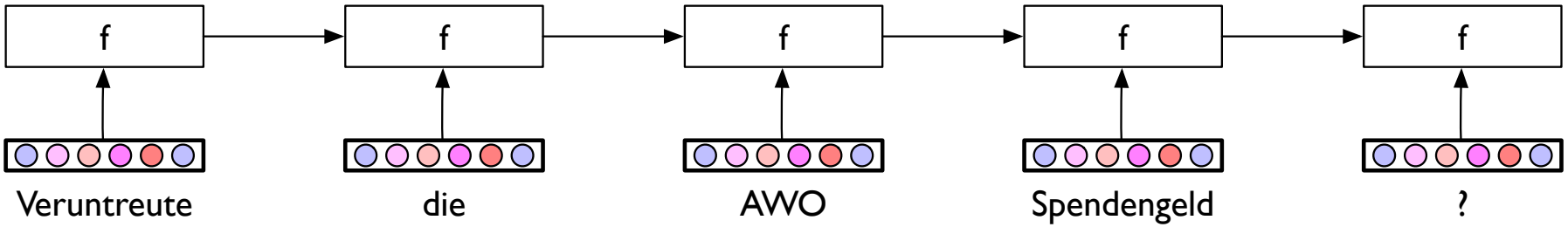
- Consider a sequence $\boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$.
- Each $\boldsymbol{x}^{<t>} \in \boldsymbol{x}^{<1>} \ldots \boldsymbol{x}^{<n>}$ is called a **time step**.
- $\boldsymbol{h}^{<i>}$ is a hidden layer at time step i.
- We will use $\boldsymbol{h}^{<i>}$ as a representation of time step i and all preceding time steps.
- We use a f to compute the hidden state.

$$\boldsymbol{h}^{<t>} = f(\boldsymbol{h}^{<t-1>}, \boldsymbol{x}^{<t>})$$

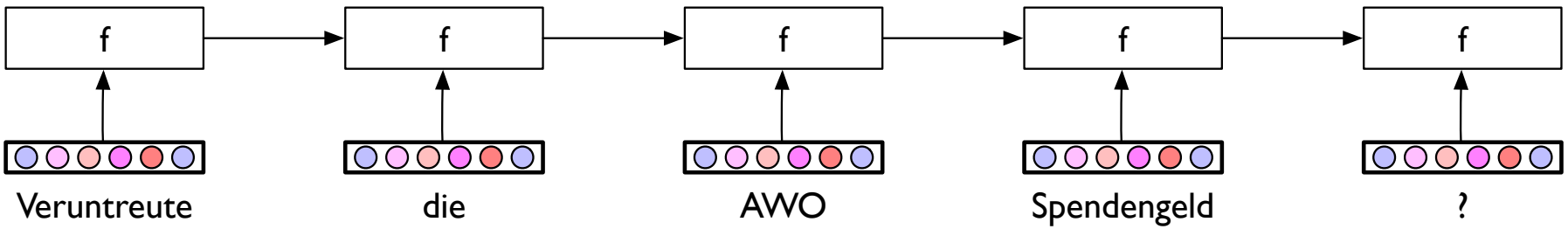
Graphically



Graphically (unrolled)



Graphically (unrolled)



- Every timestep normally uses the same function/parameters.
- During training and prediction, the RNN is typically **unrolled**:
  - A different 'unit' is used for every time step.
  - Parameters are shared between units.

Sequence classification

- $\boldsymbol{h}^{<i>}$ can be used as a regular hidden layer:
  - Use in the logistic function for binary classification.
  - Use in the sofmax function for multinomial classification.

Sequence classification

- $\boldsymbol{h}^{<i>}$ can be used as a regular hidden layer:
  - Use in the logistic function for binary classification.
  - Use in the sofmax function for multinomial classification.
- Applying a classifier to the last time step n produces a classifier for the sequence:

$$p(y|\boldsymbol{x}^{<1>\cdots<n>}) = \frac{e^{\boldsymbol{W}_y \boldsymbol{h}^{<n>}}}{\sum_{i=1}^{k} e^{\boldsymbol{W}_k \boldsymbol{h}^{<n>}}}$$

Sequence classification

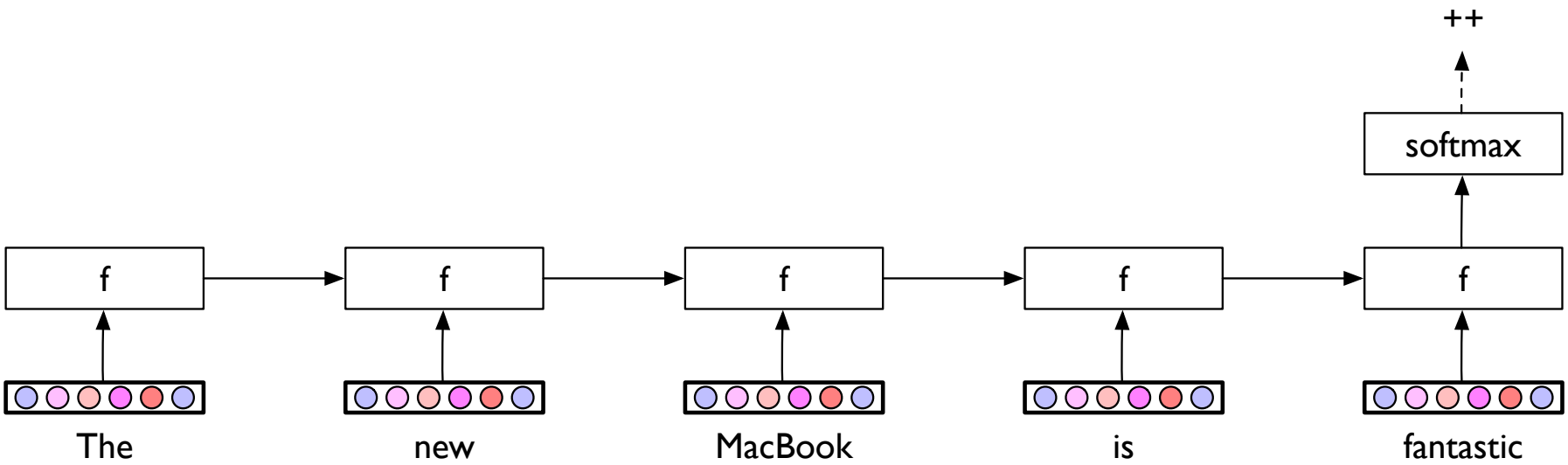
- $\boldsymbol{h}^{<i>}$ can be used as a regular hidden layer:
  - Use in the logistic function for binary classification.
  - Use in the sofmax function for multinomial classification.
- Applying a classifier to the last time step n produces a classifier for the sequence:

$$p(y|\boldsymbol{x}^{<1>\cdots<n>}) = \frac{e^{\boldsymbol{W}_y \boldsymbol{h}^{<n>}}}{\sum_{i=1}^{k} e^{\boldsymbol{W}_k \boldsymbol{h}^{<n>}}}$$

Examples:

- Sentiment analysis
- Question type classification
- Subjectivity detection

Sequence classification



Sequence labeling

- Another scenario is that we want to assign a label to every element in the sequence.
- This task is often called **sequence labeling**.

Sequence labeling

- Another scenario is that we want to assign a label to every element in the sequence.
- This task is often called **sequence labeling**.
- Apply a classifier to the hidden representation of every timestep:

$$p(y|\boldsymbol{x}^{<1>\cdots<t>}) = \frac{e^{\boldsymbol{W}_y \boldsymbol{h}^{<t>}}}{\sum_{i=1}^{k} e^{\boldsymbol{W}_k \boldsymbol{h}^{<t>}}}$$

Sequence labeling

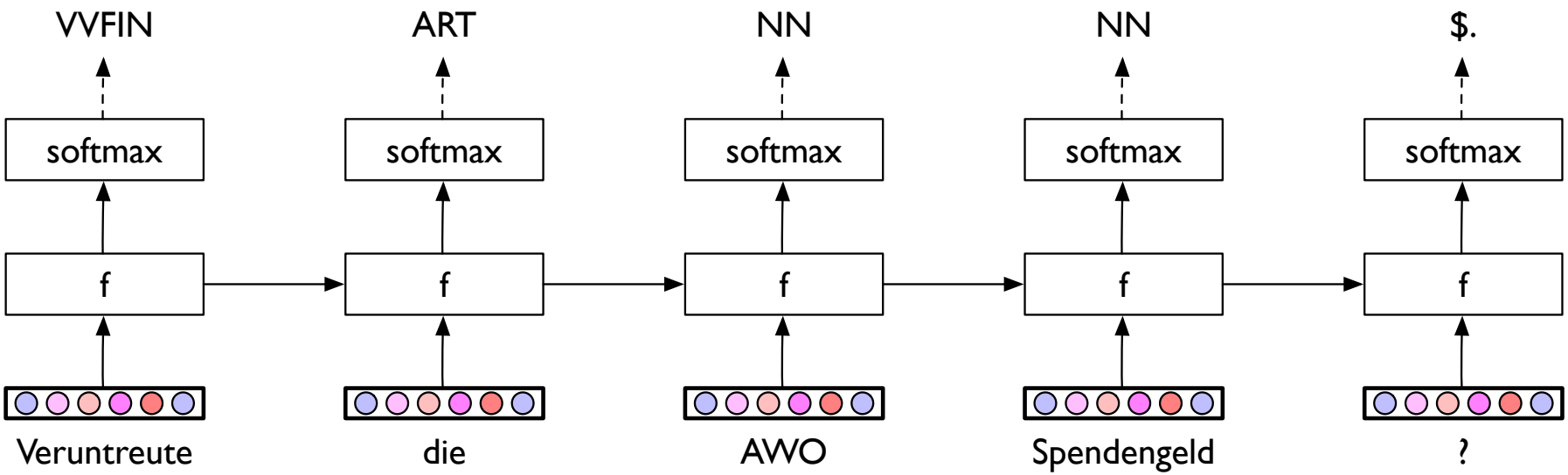
- Another scenario is that we want to assign a label to every element in the sequence.
- This task is often called **sequence labeling**.
- Apply a classifier to the hidden representation of every timestep:

$$p(y|\boldsymbol{x}^{<1>\cdots<t>}) = \frac{e^{\boldsymbol{W}_y \boldsymbol{h}^{<t>}}}{\sum_{i=1}^{k} e^{\boldsymbol{W}_k \boldsymbol{h}^{<t>}}}$$

Examples:

- Tokenization
- Part-of-speech tagging
- **Topological field labeling**

Sequence labeling



Limits of left-context

(6) [die Siegerin]$_{\text{VF}}$ wurde disqualifiziert .
[the winner/*fem*] was disqualified .

(7) [die Siegerin]$_{\text{MF}}$ zu disqualifizieren .
[the winner/*fem*] to disqualify .

(De Kok & Hinrichs, 2016)

Forwards/backwards RNNs

- First we rename our existing hidden layer $\boldsymbol{h}^{<t>}$ to $\overrightarrow{\boldsymbol{h}}^{<t>}$

$$\overrightarrow{\boldsymbol{h}}^{<t>} = f(\overrightarrow{\boldsymbol{h}}^{<t-1>}, \boldsymbol{x}^{<t>})$$

Forwards/backwards RNNs

- First we rename our existing hidden layer $\boldsymbol{h}^{<t>}$ to $\overrightarrow{\boldsymbol{h}}^{<t>}$

$$\overrightarrow{\boldsymbol{h}}^{<t>} = f(\overrightarrow{\boldsymbol{h}}^{<t-1>}, \boldsymbol{x}^{<t>})$$

This is a **forward RNN**.

Forwards/backwards RNNs

- First we rename our existing hidden layer $\boldsymbol{h}^{<t>}$ to $\overrightarrow{\boldsymbol{h}}^{<t>}$

$$\overrightarrow{\boldsymbol{h}}^{<t>} = f(\overrightarrow{\boldsymbol{h}}^{<t-1>}, \boldsymbol{x}^{<t>})$$

  This is a **forward RNN**.
- Then we define a **backward RNN** $\overleftarrow{\boldsymbol{h}}^{t}$ that uses the hidden layer of a future timestep:

$$\overleftarrow{\boldsymbol{h}}^{<t>} = f(\overleftarrow{\boldsymbol{h}}^{<t+1>}, \boldsymbol{x}^{<t>})$$

Bidirectional RNN

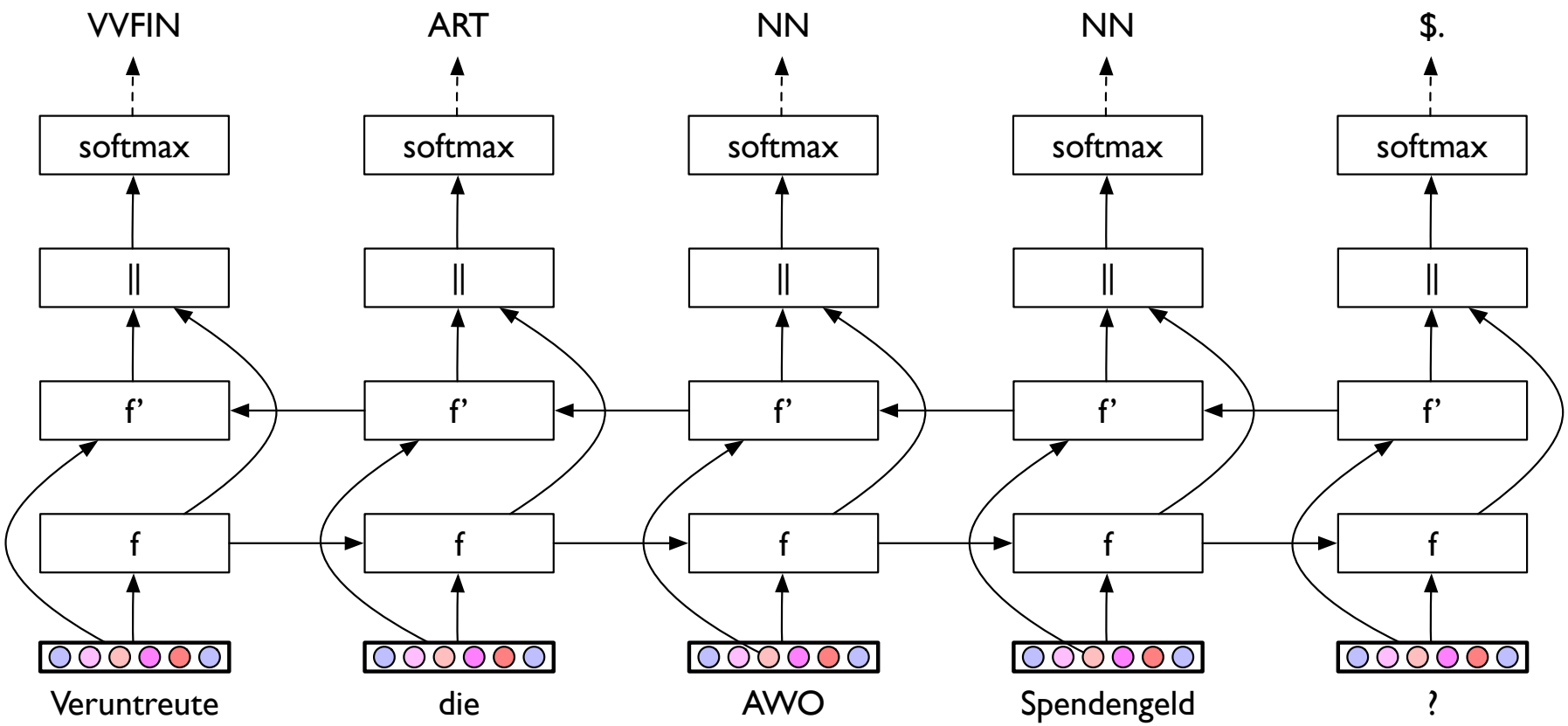
- A **bidirectional RNN** uses a hidden layer for each timestep that is the concatenation of:
  - A forward RNN hidden layer; and
  - a backward RNN hidden layer.

Bidirectional RNN

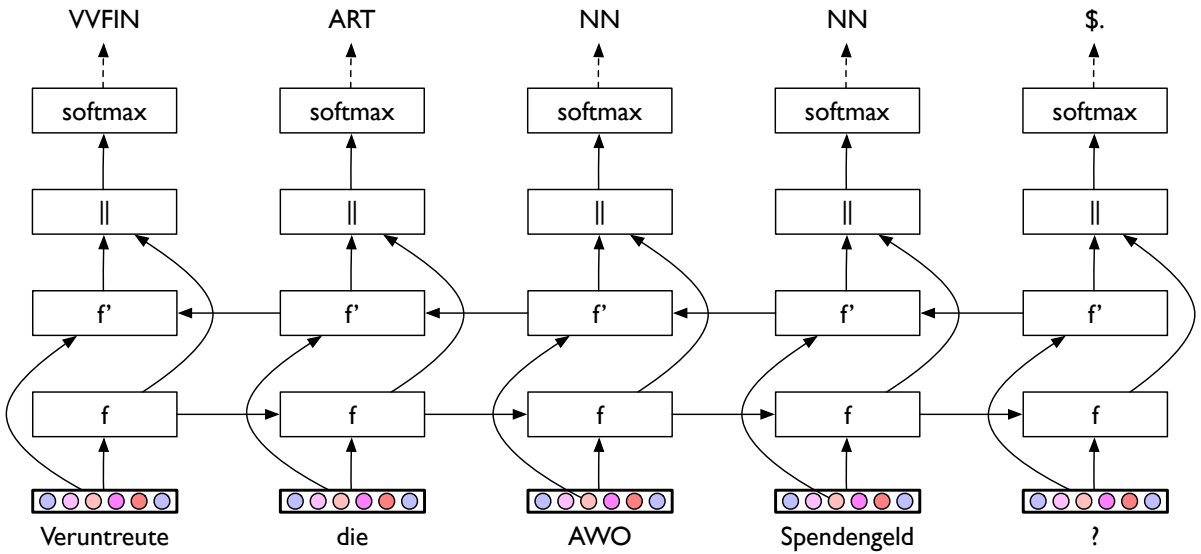
- A **bidirectional RNN** uses a hidden layer for each timestep that is the concatenation of:
  - A forward RNN hidden layer; and
  - a backward RNN hidden layer.
- If $\|$ is the vector concatenation operator, such that $(\boldsymbol{u} \in \mathbb{R}^n \| \boldsymbol{v} \in \mathbb{R}^m) \in \mathbb{R}^{n+m}$, then:

$$\overleftrightarrow{\boldsymbol{h}}^{<t>} = \overrightarrow{\boldsymbol{h}}^{<t>} \| \overleftarrow{\boldsymbol{h}}^{<t>}$$

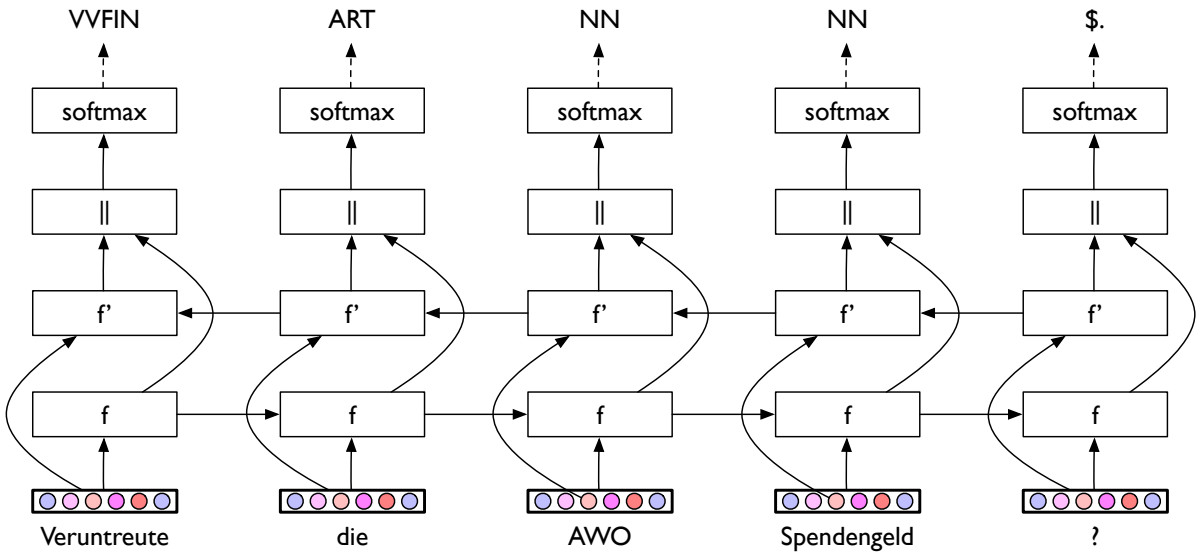
Bidirectional RNN for sequence labeling



Bidirectional RNN for sequence labeling



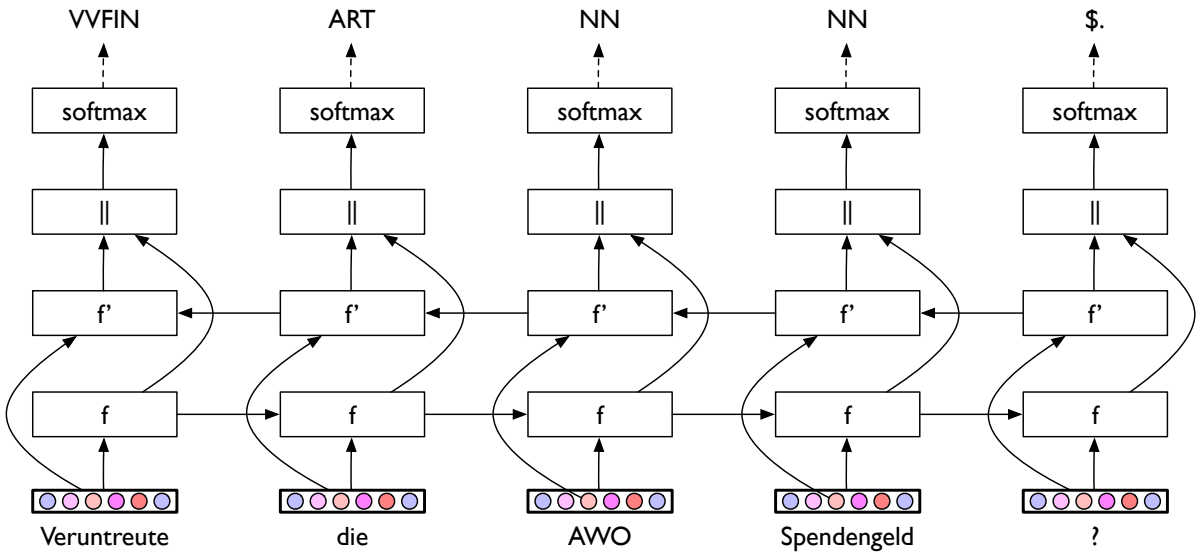
Bidirectional RNN for sequence labeling



Note:

- This network can be made even more powerful by applying another non-linear transformation to $\overleftrightarrow{\boldsymbol{h}}^{<t>}$.

Bidirectional RNN for sequence labeling



Note:

- This network can be made even more powerful by applying another non-linear transformation to $\overleftrightarrow{\boldsymbol{h}}^{<t>}$.
- When this non-linear transformation is also an RNN, this a **stacked RNN**.

RNN functions

Introduction

- We have looked at how RNNs can be used for sequence labeling and classification tasks.
- Open question: what is a good function f for our hidden units?

$$\boldsymbol{h}^{<t>} = f(\boldsymbol{h}^{<t-1>}, \boldsymbol{x}^{<t>})$$

Elman networks (Elman, 1990)

$$f(\boldsymbol{h}^{<t-1>}, \boldsymbol{x}^{<t>}) =$$

Elman networks (Elman, 1990)

$$f(\boldsymbol{h}^{<t-1>}, \boldsymbol{x}^{<t>}) = \quad \boldsymbol{h}^{<t-1>} \quad \boldsymbol{x}^{<t>}$$

Elman networks (Elman, 1990)

$$f(\boldsymbol{h}^{<t-1>}, \boldsymbol{x}^{<t>}) = \quad \boldsymbol{U}\boldsymbol{h}^{<t-1>} \quad \boldsymbol{W}\boldsymbol{x}^{<t>}$$

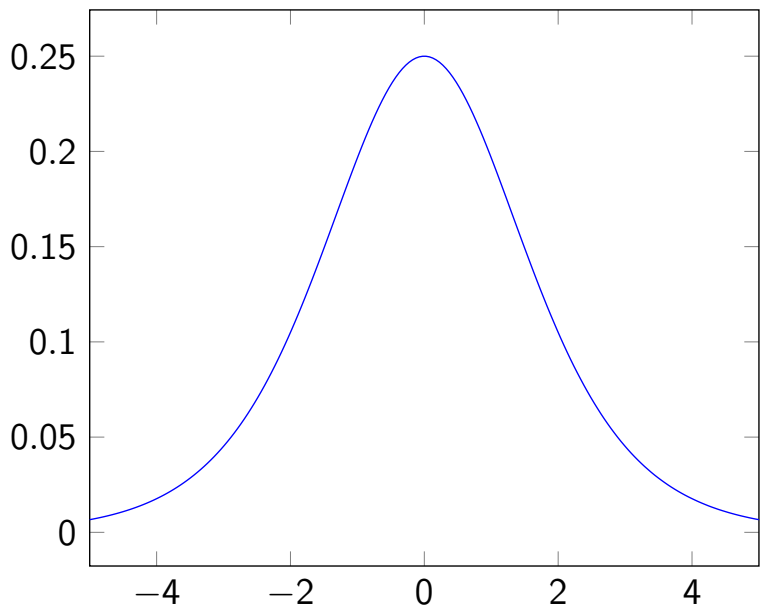
Elman networks (Elman, 1990)

$$f(\boldsymbol{h}^{<t-1>}, \boldsymbol{x}^{<t>}) = \quad \boldsymbol{U}\boldsymbol{h}^{<t-1>} + \boldsymbol{W}\boldsymbol{x}^{<t>}$$

Elman networks (Elman, 1990)

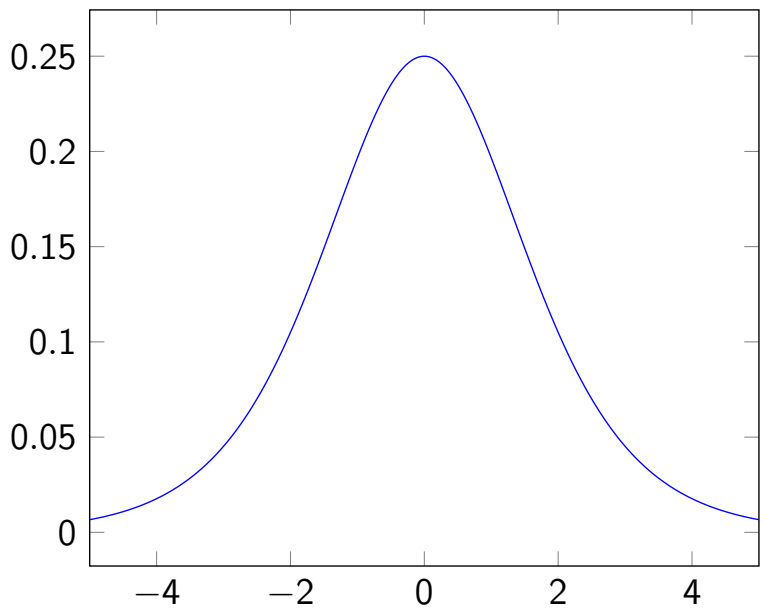
$$f(\boldsymbol{h}^{<t-1>}, \boldsymbol{x}^{<t>}) = \sigma(\boldsymbol{U}\boldsymbol{h}^{<t-1>} + \boldsymbol{W}\boldsymbol{x}^{<t>})$$

Gradient



- In backpropagation, each parameter is updated with a fraction of the partical derivative of the loss with regards to that parameter.
- The derivative of the logistic function is: $\sigma(x)(1 - \sigma(x)$.

Gradient



- In backpropagation, each parameter is updated with a fraction of the partical derivative of the loss with regards to that parameter.
- The derivative of the logistic function is: $\sigma(x)(1 - \sigma(x)$.
- The gradient is in the range $(0, \frac{1}{4}]$.
- We apply the chain rule for computing the derivatives when backpropagating in time.

Gated Recurrent Unit

$$\boldsymbol{h}^{<t>} = \boldsymbol{z}^{<t>} \odot \boldsymbol{h}^{<t-1>} + (1 - \boldsymbol{z}^{<t>}) \odot \tilde{\boldsymbol{h}}^{<t>}$$

Gated Recurrent Unit

$$\boldsymbol{h}^{<t>} = \boldsymbol{z}^{<t>} \odot \boldsymbol{h}^{<t-1>} + (1 - \boldsymbol{z}^{<t>}) \odot \tilde{\boldsymbol{h}}^{<t>}$$

$$\boldsymbol{z}^{<t>} = \sigma(\boldsymbol{W}^{z}\boldsymbol{x}^{<t>} + \boldsymbol{U}^{z}\boldsymbol{h}^{<t-1>} + \boldsymbol{b}^{z})$$

Gated Recurrent Unit

$$\boldsymbol{h}^{<t>} = \boldsymbol{z}^{<t>} \odot \boldsymbol{h}^{<t-1>} + (1 - \boldsymbol{z}^{<t>}) \odot \tilde{\boldsymbol{h}}^{<t>}$$

$$\boldsymbol{z}^{<t>} = \sigma(\boldsymbol{W}^{z}\boldsymbol{x}^{<t>} + \boldsymbol{U}^{z}\boldsymbol{h}^{<t-1>} + \boldsymbol{b}^{z})$$

$$\tilde{\boldsymbol{h}}^{<t>} = \tanh(\boldsymbol{W}^{h}\boldsymbol{x}^{<t>} + \boldsymbol{U}^{h}(\boldsymbol{r}^{<t>} \odot \boldsymbol{h}^{<t-1>}) + \mathbf{b}^{h})$$

Gated Recurrent Unit

$$
\begin{aligned}
\boldsymbol{h}^{<t>} &= \boldsymbol{z}^{<t>} \odot \boldsymbol{h}^{<t-1>} + (1 - \boldsymbol{z}^{<t>}) \odot \tilde{\boldsymbol{h}}^{<t>} \\
\boldsymbol{z}^{<t>} &= \sigma(\boldsymbol{W}^{z}\boldsymbol{x}^{<t>} + \boldsymbol{U}^{z}\boldsymbol{h}^{<t-1>} + \boldsymbol{b}^{z}) \\
\tilde{\boldsymbol{h}}^{<t>} &= \tanh(\boldsymbol{W}^{h}\boldsymbol{x}^{<t>} + \boldsymbol{U}^{h}(\boldsymbol{r}^{<t>} \odot \boldsymbol{h}^{<t-1>}) + \mathbf{b}^{h}) \\
\boldsymbol{r}^{<t>} &= \sigma(\boldsymbol{W}^{r}\boldsymbol{x}^{<t>} + \boldsymbol{U}^{r}\boldsymbol{h}^{<t-1>} + \mathbf{b}^{r})
\end{aligned}
$$

RNNs in parsing

Topological field labeling (De Kok & Hinrichs, 2016)

Labeler	Accuracy (%)
LSTM + LSTM	93.33
Bidirectional LSTM + LSTM	97.24

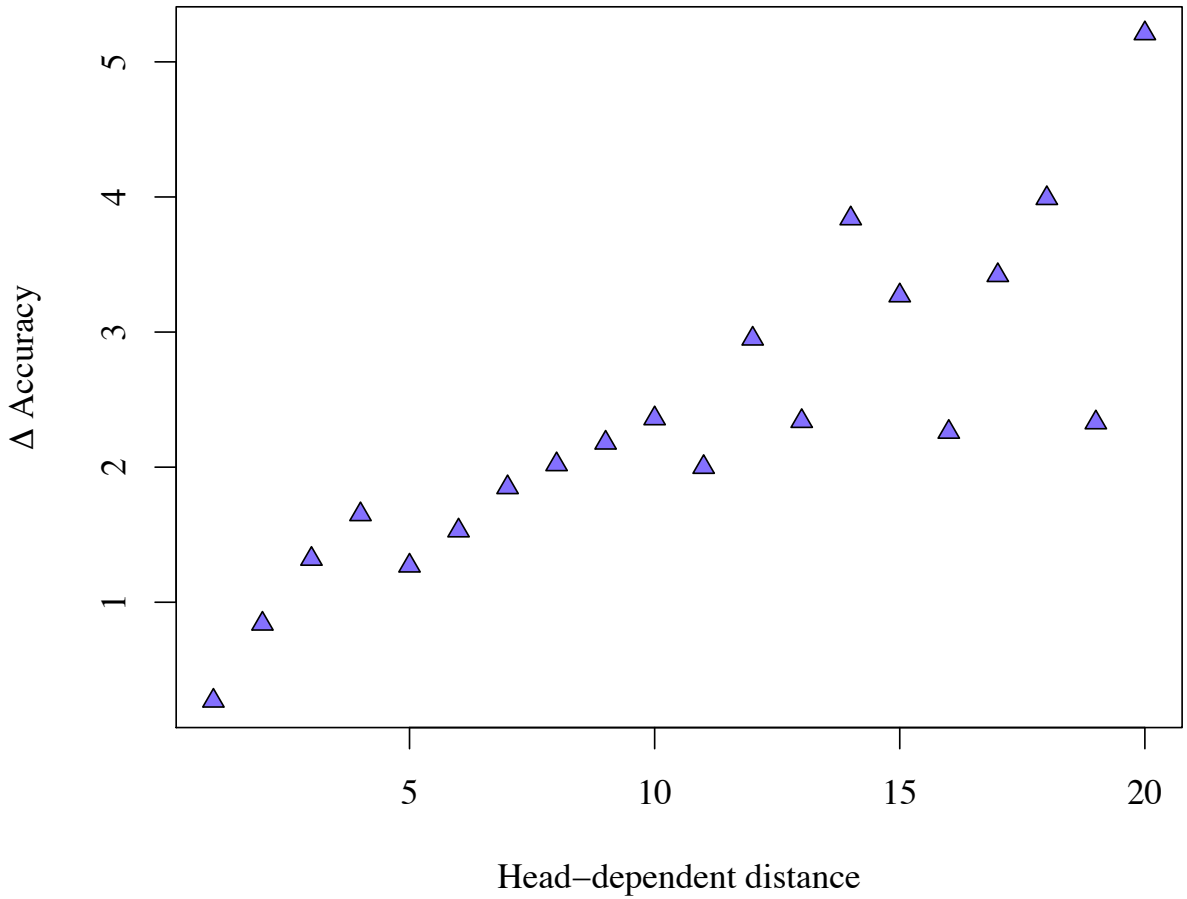
Parsing improvement

Parser	LAS	UAS
De Kok (2015)	89.49	91.88
Neural net + TFs	90.00	92.36
Neural net + gold TFs	90.42	92.76

Analysis of improvements

Dependency label	**LAS Δ**
Coordinating conjunction (clausal)	11.48
Parenthesis	8.31
Dependent clause	3.49
Conjunct	3.38
Sentence root	2.92
Expletive *es*	2.71
Sentence	2.64
Comparative	1.87
Separated verb prefix	1.64
Direct object	1.59

Analysis of improvements (2)



Most frequent parsing errors

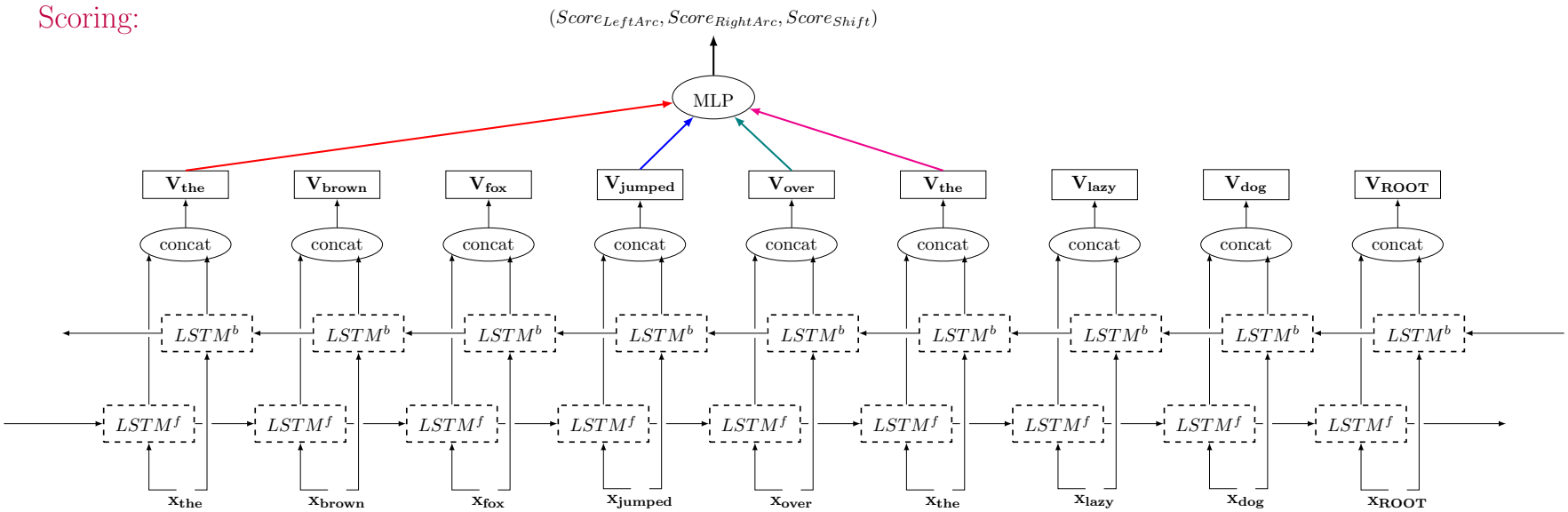
Relation	% of errors
PP	17.48
ADV	11.82
KON	8.70
SUBJ	6.21
OBJA	5.79
ROOT	5.52
OBJP	5.14
PRED	3.66
APP	3.29
REL	3.12

LSTM feature representations (Kiperwasser & Goldberg, 2016)

- Key idea:
  - Take Chen & Manning-like neural dependency parser.
  - Replace word representations by RNN hidden states.
- A token w_i is now represented as: $\boldsymbol{v}^i = \overleftrightarrow{\boldsymbol{h}}^{<i>}$
- v_i is a representation of a token in its sentential context.
- $\mathcal{O}(n)$ time when $\boldsymbol{v}^0 \ldots \boldsymbol{v}^n$ is computed once, not per parser state.

LSTM feature representations (Kiperwasser & Goldberg, 2016)

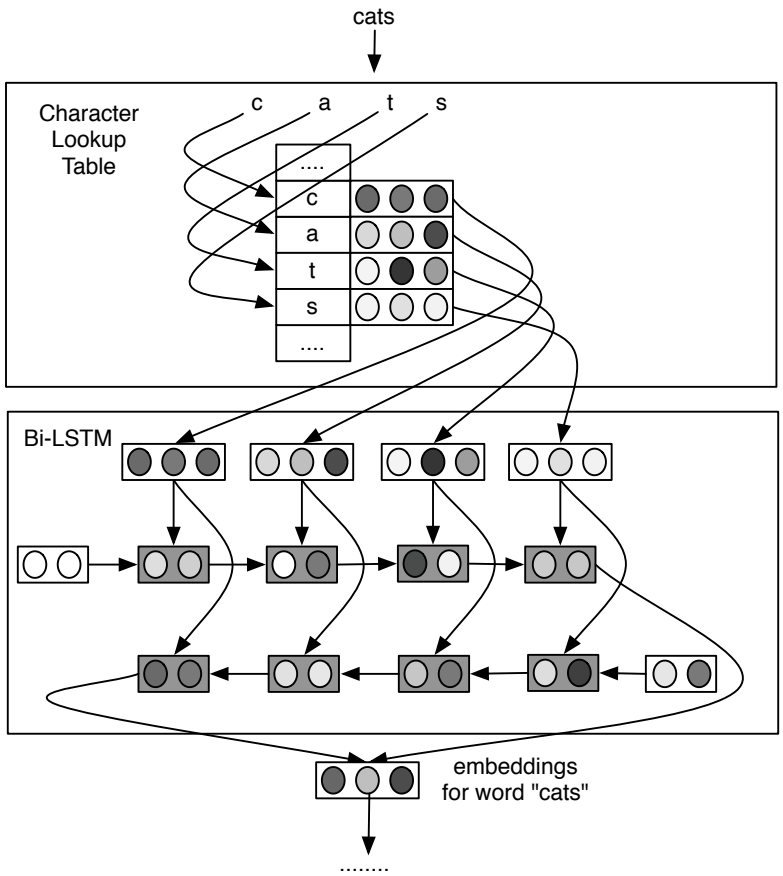
Scoring:



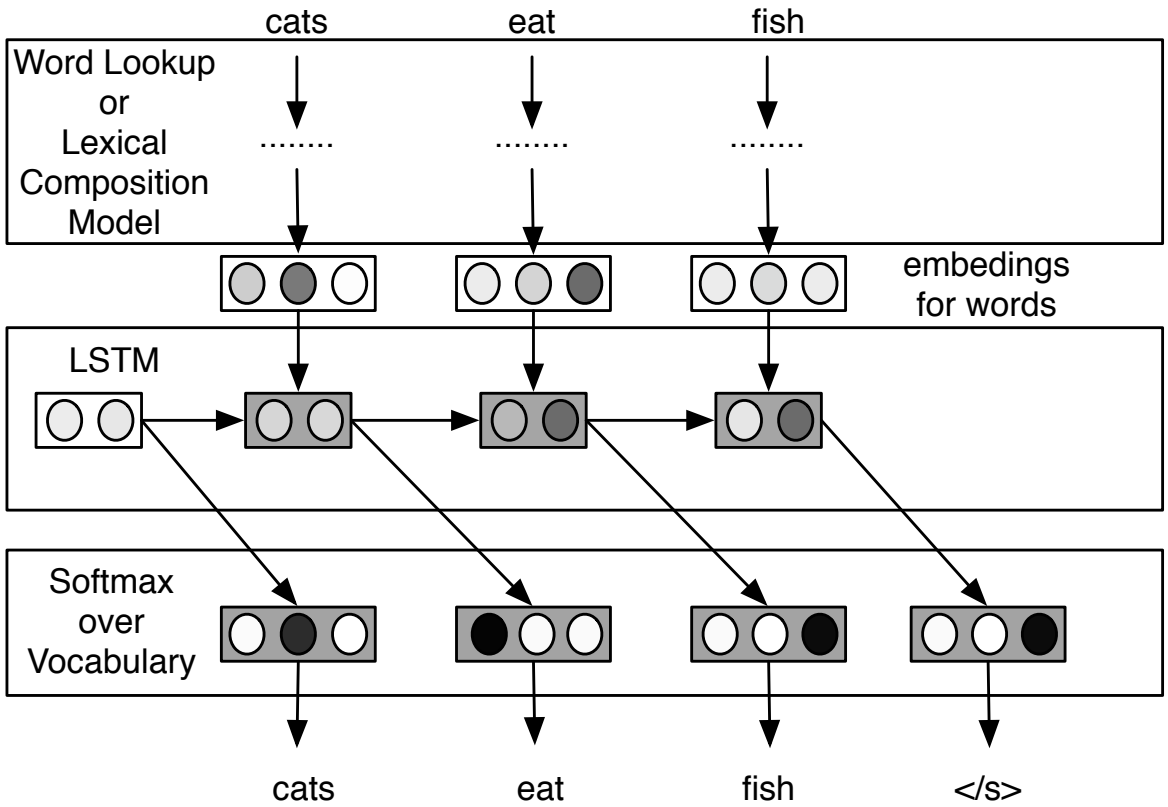
LSTM feature representations (Kiperwasser & Goldberg, 2016)

System	Method	Representation	Emb	PTB-YM	PTB-SD		CTB	
				UAS	UAS	LAS	UAS	LAS
This work	graph, 1st order	2 BiLSTM vectors	–	–	93.1	91.0	**86.6**	**85.1**
This work	transition (greedy, dyn-oracle)	4 BiLSTM vectors	–	–	93.1	91.0	86.2	85.0
This work	transition (greedy, dyn-oracle)	11 BiLSTM vectors	–	–	**93.2**	**91.2**	86.5	84.9
ZhangNivre11	transition (beam)	large feature set (sparse)	–	92.9	–	–	86.0	84.4
Martins13 (TurboParser)	graph, 3rd order+	large feature set (sparse)	–	92.8	93.1	–	–	–
Pei15	graph, 2nd order	large feature set (dense)	–	93.0	–	–	–	–
Dyer15	transition (greedy)	Stack-LSTM + composition	–	–	92.4	90.0	85.7	84.1
Ballesteros16	transition (greedy, dyn-oracle)	Stack-LSTM + composition	–	–	92.7	90.6	86.1	84.5
This work	graph, 1st order	2 BiLSTM vectors	YES	–	93.0	90.9	86.5	84.9
This work	transition (greedy, dyn-oracle)	4 BiLSTM vectors	YES	–	93.6	91.5	87.4	85.9
This work	transition (greedy, dyn-oracle)	11 BiLSTM vectors	YES	–	93.9	91.9	**87.6**	86.1
Weiss15	transition (greedy)	large feature set (dense)	YES	–	93.2	91.2	–	–
Weiss15	transition (beam)	large feature set (dense)	YES	–	**94.0**	**92.0**	–	–
Pei15	graph, 2nd order	large feature set (dense)	YES	93.3	–	–	–	–
Dyer15	transition (greedy)	Stack-LSTM + composition	YES	–	93.1	90.9	87.1	85.5
Ballesteros16	transition (greedy, dyn-oracle)	Stack-LSTM + composition	YES	–	93.6	91.4	**87.6**	**86.2**
LeZuidema14	reranking /blend	inside-outside recursive net	YES	93.1	93.8	91.5	–	–
Zhu15	reranking /blend	recursive conv-net	YES	93.8	–	–	85.7	–

C2W model (Ling, et al. 2015)



C2W model (Ling, et al. 2015)



C2W model (Ling, et al. 2015)

System	Fusional			Agglutinative	
	EN	PT	CA	DE	TR
Word	96.97	95.67	98.09	97.51	83.43
C2W	**97.36**	97.47	**98.92**	**98.08**	**91.59**
Stanford	97.32	**97.54**	98.76	97.92	87.31

A sobering note

- Erik Schill, 2018, out-of-domain tagging:
  - Training on TüBa-D/Z.
  - Evaluation on German NoSta-D corpus of German non-standard varieties.

A sobering note

- Erik Schill, 2018, out-of-domain tagging:
  - Training on TüBa-D/Z.
  - Evaluation on German NoSta-D corpus of German non-standard varieties.
- Network architecture:
  - RNN (GRU) over words
  - CRF layer

A sobering note

- Erik Schill, 2018, out-of-domain tagging:
  - Training on TüBa-D/Z.
  - Evaluation on German NoSta-D corpus of German non-standard varieties.
- Network architecture:
  - RNN (GRU) over words
  - CRF layer
- Various word representations:
  - Word embedding
  - RNN over characters + word embedding
  - Prefix + suffix character, fed through a feed-forward layer (similar to De Kok, 2015) + word embedding
  - FastText embedding
- For word embedding, the context-sensitive skip-gram model was used.

Results

Tagger	Accuracy
Trigram HMM	85.97
RNN + prefix/suffix feed-forward	**89.63**
RNN + char RNN	88.79
RNN + fastText	87.52