Character-based Embeddings of Words with Recurrent Nets

Simon Grätzer

Seminar Selected Topics in Human Language Technology and Pattern Recognition June 20, 2016

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6
Computer Science Department
RWTH Aachen University, Germany

Outline

Literature

Wang Ling et al 2015 : Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. CoRR Volume abs/1508.02096, 2015

- Introduces a model for constructing vector representations of words by composing characters using bidirectional LSTMs.

Faruqui et al 2015 : Morphological Inflection Generation Using Character Sequence to Sequence Learning. CoRR Volume abs/1512.06110, 2015

- Approach to generate inflected versions of words by modelling the process as a character sequence to sequence learning problem.

Mikolov et al 2013 : Distributed Representations of Words and Phrases and their Compositionality. CoRR Volume abs/1310.4546, 2013

- Improvements on the Skip-gram model used to generate word embeddings.

Literature

Schwenk 2007 : Continuous Space Language Models. Journal Computer Speech and Language. Volume 21 Issue 3, July, 2007 Pages 492-518

- Describes the use of a neural network language model for large vocabulary continuous speech recognition.

Peirsman 2015 : Visualizing Word Embeddings with t-SNE. Online 2015

- Creating useful low-dimensional visualizations for high-dimensional datasets

TensorFlow 2016 : Vector Representations of Words, From the TensorFlow documentation

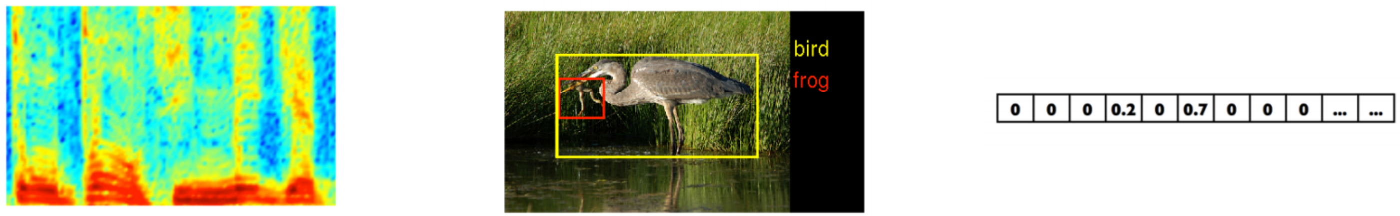
- Implementation of word2vec model of [Mikolov et al 2013] with the TensorFlow framework.

Introduction

- **Word embeddings are real valued vector representations for words.**
- **This talk is about generating word embeddings and their applications.**
- **Specifically:**
  - **Generating word embeddings by composing their individual character representations**
  - **Using Long short-term memory to capture complex relationships between words.**
- **The resulting model can be used for many tasks, such as language modeling or part-of-speech tagging**
- **Reduces the need for manual feature engineering**
- **Can improve performance for many tasks**

Introduction

AUDIO **IMAGES** **TEXT**



Audio Spectrogram — DENSE

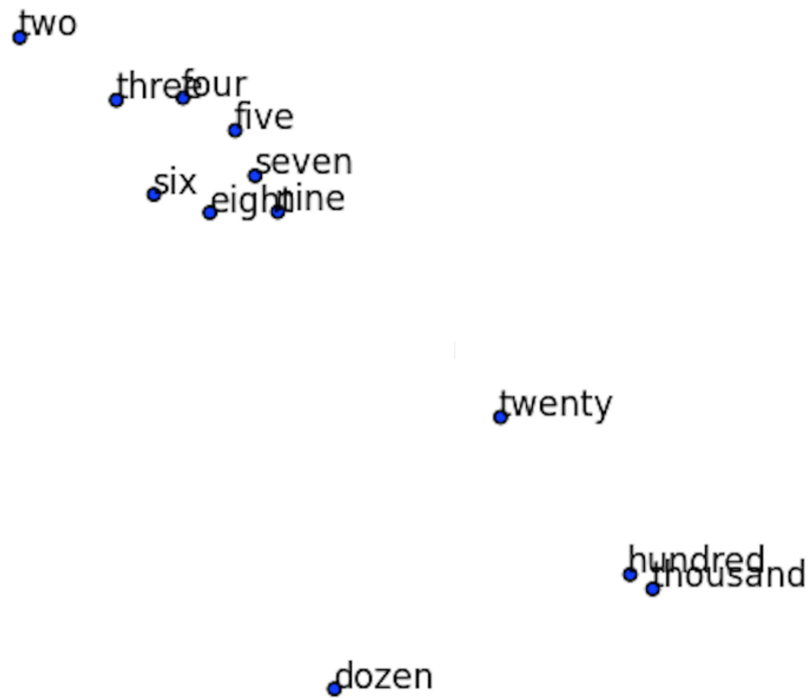
Image pixels — DENSE

Word, context, or document vectors — SPARSE

[TensorFlow 2016]

- **Natural language processing systems can treat words as atomic symbols, encoded as indexes.**
- **E.g. 'apple' might become index *Id123* and 'orange' becomes index *Id124***
- **This way of encoding is very sparse, and arbitrary.**
- **No representation of the relationship between these word types (such as both are fruit, . . .).**

Word Embeddings



Embeddings visualized [Peirsman 2015]

- A statistical model should be able to learn relationships between word types.
- Transform a word type w and turn it into an embedding $e_w = v(w) \in \mathbb{R}^d$
- Words with a similar meaning should be mapped to (geometrically) nearby points in the vector space.
- The dimension d should be low compared to the vocabulary size.
- Capture the intuition that words may be similar along different dimensions.

Word Embeddings

To this end there are some assumptions we make:

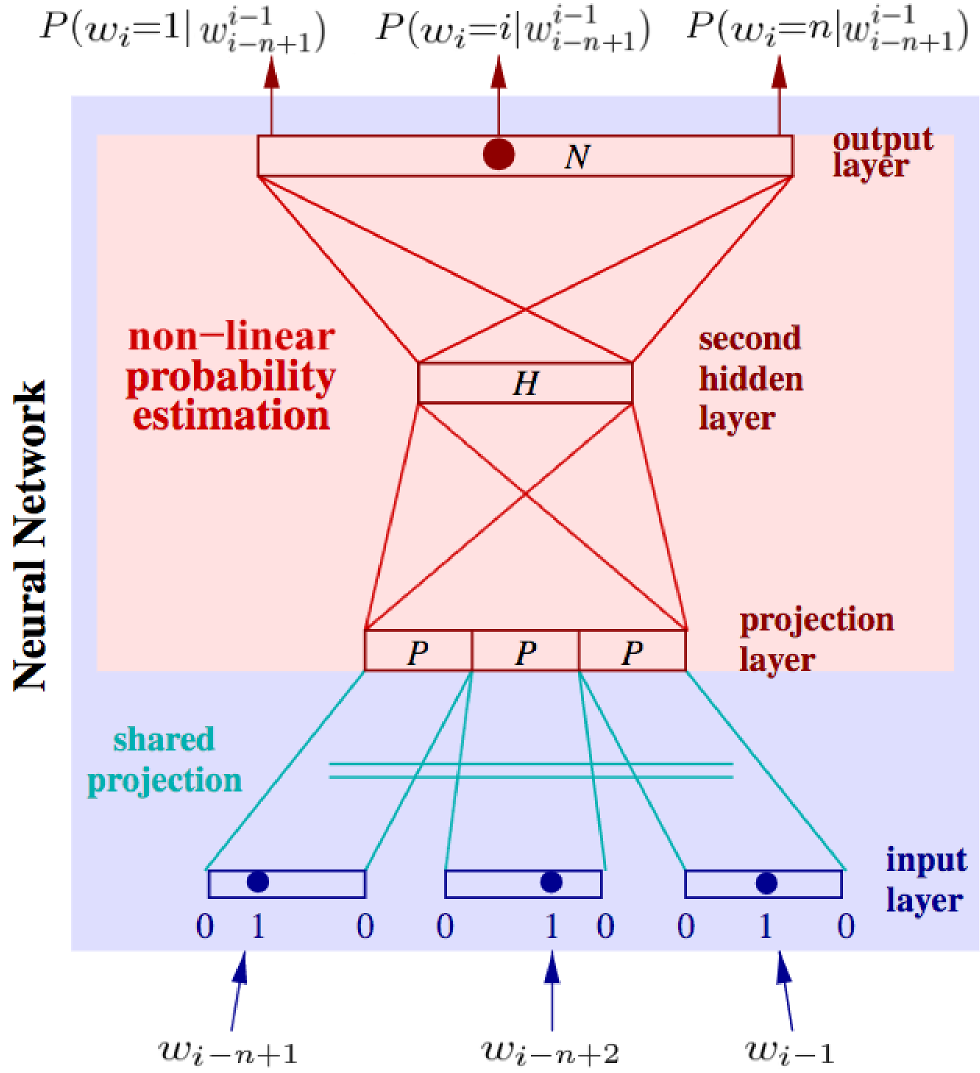
- **Words which share semantic meaning tend to occur in the same contexts (Distributional Hypothesis)**
- **The composition of words themselves can sometimes hint at similarities (e.g. 'apple' *vs* 'apples')**

General Approach for Word Embeddings

Generating word embeddings within a NNLM to better estimate $p(w_i|w_1, \ldots, w_{i-1})$

1. **Associate each word w in the vocabulary V with a word embedding e_w**
2. **Express the joint probability function for the word sequence $w_1, \ldots, w_{i-1}$ in terms of these embeddings.**
3. **Simultaniously learn the word embeddings and the parameters of the probability function.**

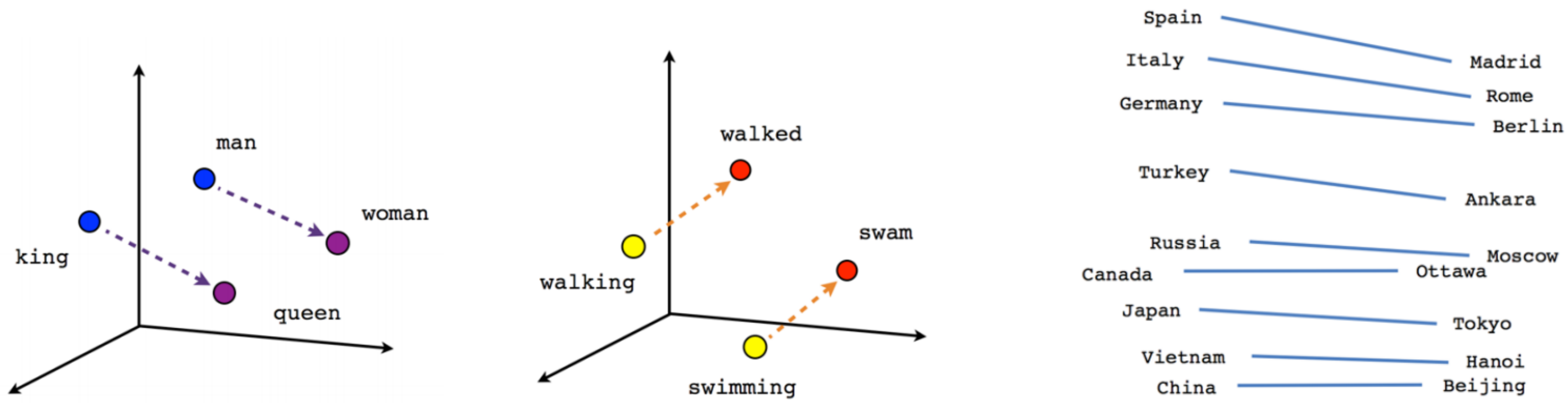
Continuous Space Language Model



[Schwenk 2007]

- The context for w_i is approximated with n-grams.
- Input word types are encoded as one-hot vectors.
- First Hidden Layer: Lookup table $P \in \mathbb{R}^{d \times |V|}$, projects the input into a continouus vector space.
- The word embedding e_w is the output of layer P.
- Second Hidden Layer: Estimates the joint probability of the word sequence
- The Softmax-Layer projects this up to vocabulary size $|V|$

Resulting Embeddings



Male-Female Verb tense Country-Capital

[TensorFlow 2016]

- **This setup can be used to generate a large table of word embeddings.**
- **Lookup table can be reused for other tasks (if there is not enough training data).**
- **E.g. [Mikolov et al 2013] have created a model which can encode complex patterns:**
- $v(\text{king}) - v(\text{male}) + v(\text{female}) \approx v(\text{queen})$.

Shortcomings

- A model with a lookup table treats each word embedding as independent from each other.
  - The model captures smililar linear correspondences between words embeddings.
  - E.g. *cat* and *apple* compared to *cats* and *apples*.
  - It doesn't capture that the added *s* is responsible for this transformation.
  - The model doesn't examine lexical similarities between words.
  - It doesn't capture morphological word transformations: e.g. *test* vs. *testing*
- A word lookup table cannot easily deal with unknown words.
- The lookup table contains at least $|V| \times d$ parameters. This can require large amount of memory for tasks with large vocabularies.

Possible Solutions

Some requirements should be satisfied by a better model:

- **The model should capture[*] orthographic similarities between words e.g. *test* vs. *testing* (Compositional effects).**
- **The model must still capture functionally similar words, with no orthographic similarities e.g. *rich* vs. *affluent*.**
- **However not all similary spelled words have similar meanings e.g. *butter* vs. *batter*.**
- **The resulting model should be able to replace the previous projection layer.**
- **Idea: Break down words into smaller atomic units and try to *compose* them into word embeddings.**

***) In terms of geometric locality**

Possible Solution 1: Morpheme-based Embeddings

- **Morphemes are the smallest defined grammatical unit of a language.**
- **E.g. "Unbreakable" comprises three morphemes:**
  1. **un-**
  2. **-break-**
  3. **-able**
- **Use morphemes as input and compose them into the word embeddings.**
- **Requires a morphological analyser: Extra processing step.**
- **Each target language requires an extra morpological analyser.**
- **The quality of the model depends on the analyser.**

Possible Solution 2: Character-based Embeddings

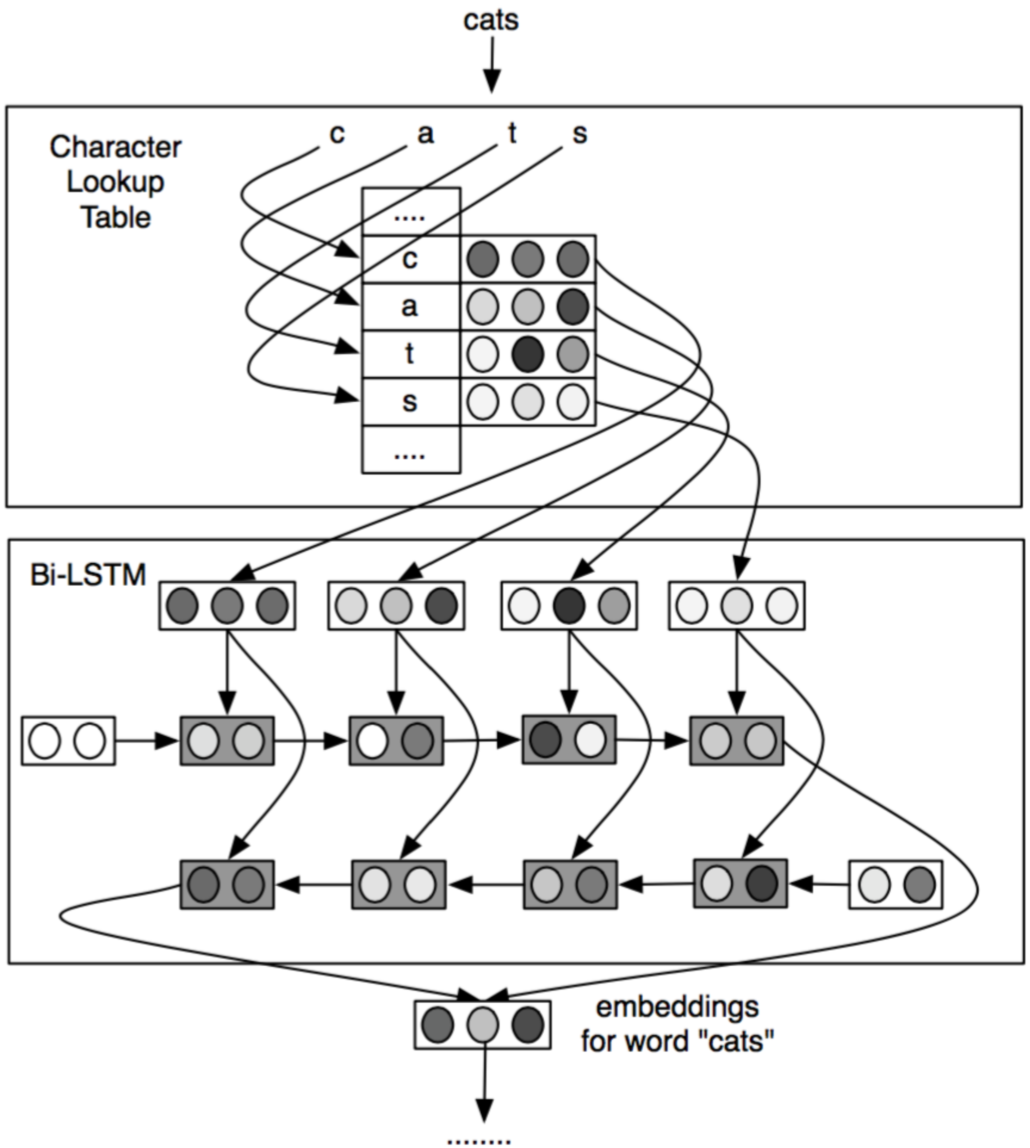
- **Break down words into characters**
- **Represent characters as real valued feature vectors (character embeddings).**
- **Feed them to an recurrent neural net, which "remembers" each character.**
- **Learn character embeddings simultaniously with other model parameters.**

Character-based Word-Embeddings

The "Compositional Character to Word" (C2W) model:

1. **A word w with length m is decomposed into characters $c_1, \dots, c_m$ from the alphabet C.**
2. **Transform characters into a sequence of character embeddings $e_{c_1}, \dots, e_{c_m}$.**
3. **The sequence is "read" one-by-one forwards as well as backwards by two LSTMs.**
4. **During "reading" the sequence is composed into the forward state s_m^f and backward state s_1^b.**
5. **The two states are recomposed to form the word-embedding e_w.**

Compositional Character to Word Model

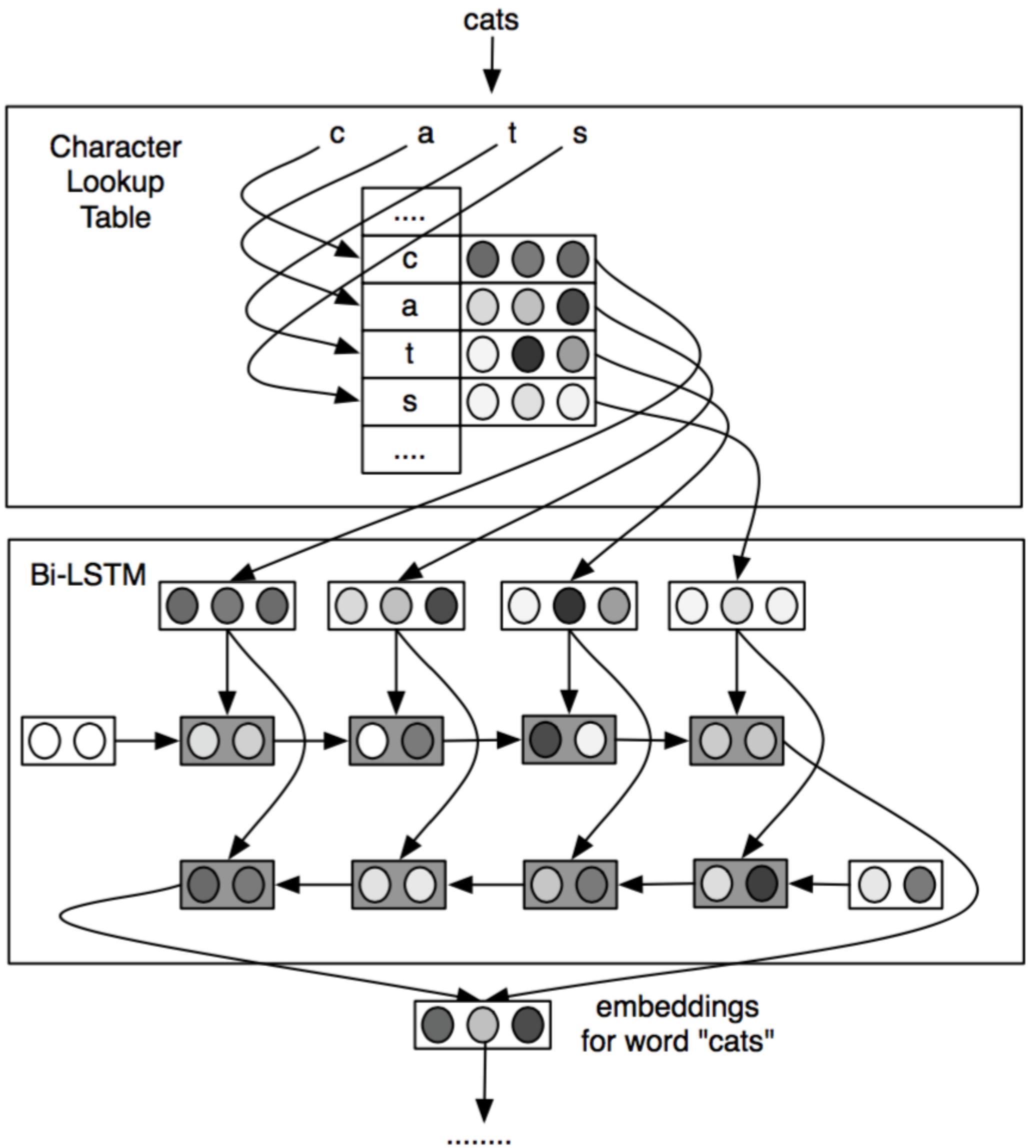


[Wang Ling et al 2015]

The C2W model is visualized for input "cats":

- **Squared boxes represent vectors of neuron activations.**
- **Shaded boxes indicate a nonlinear output.**
- **The two actual LSTM units are displayed unfolded.**

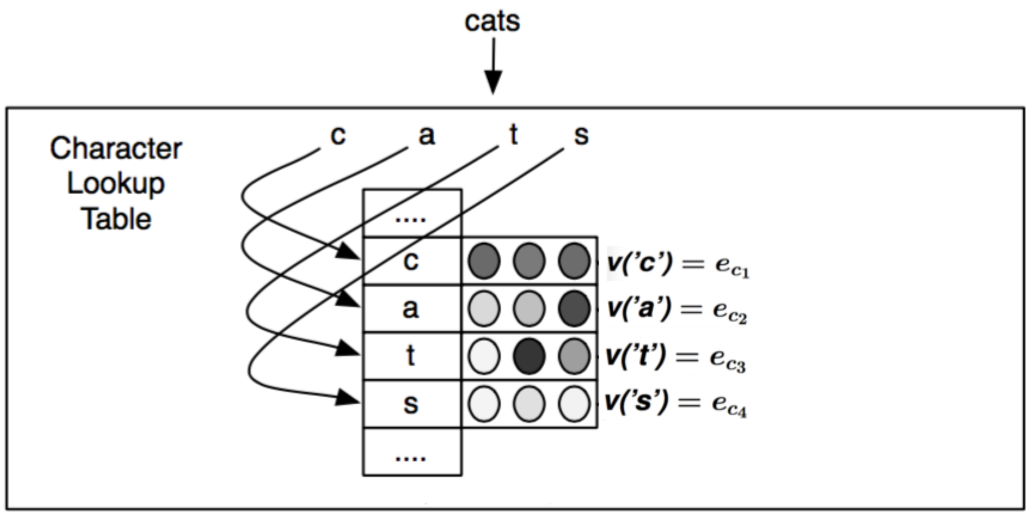
C2W-Model: Layers



[Wang Ling et al 2015]

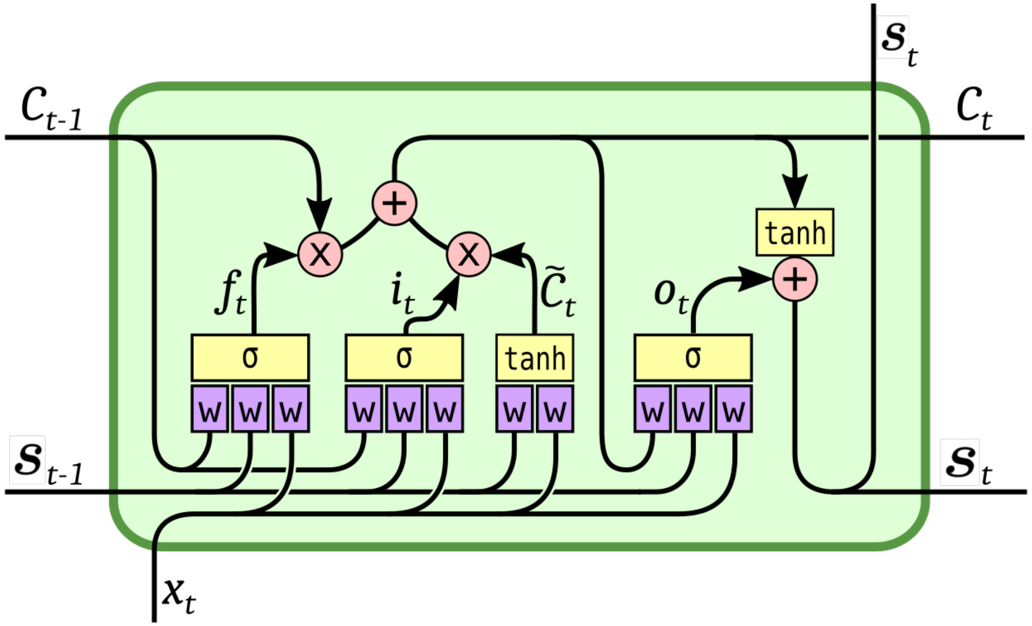
1. Table of character embeddings.
2. Bidirectional-LSTM layer, processing forward- and backward-sequences of character embeddings.
3. The combining layer, to merge the two outputs from the Bi-LSTM.

C2W-Model: Character-Lookup Table



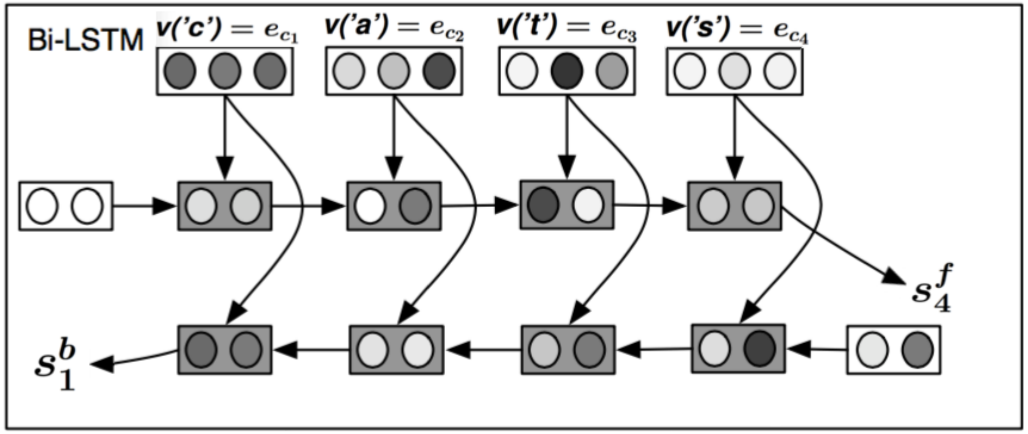
- **Table of d_C parameters $P_C \in \mathbb{R}^{d_C \times |C|}$.**
- **Each Input character c is transformed into a d_C-dimensional feature vector e_c.**
- **The dimension d_C becomes a hyperparameter of the model.**
- **Basically similar to the previous projection layer for words.**

Reminder: Long Short-Term Memory



- **Designed to "remember" inputs over arbitary distances and "forget" them when necessary**
- **Gate i_t to determine when to learn an input value**
- **Gate f_t to determine if it should continue to remember or forget the currently stored value**
- **Gate o_t to determine wether it should output the value.**
- **Additionally and bias values not explicitly displayed here.**
- **The dimension d_{CS} of the LSTM state becomes another hyperparameter.**

C2W-Model: Bidirectional LSTM Layer

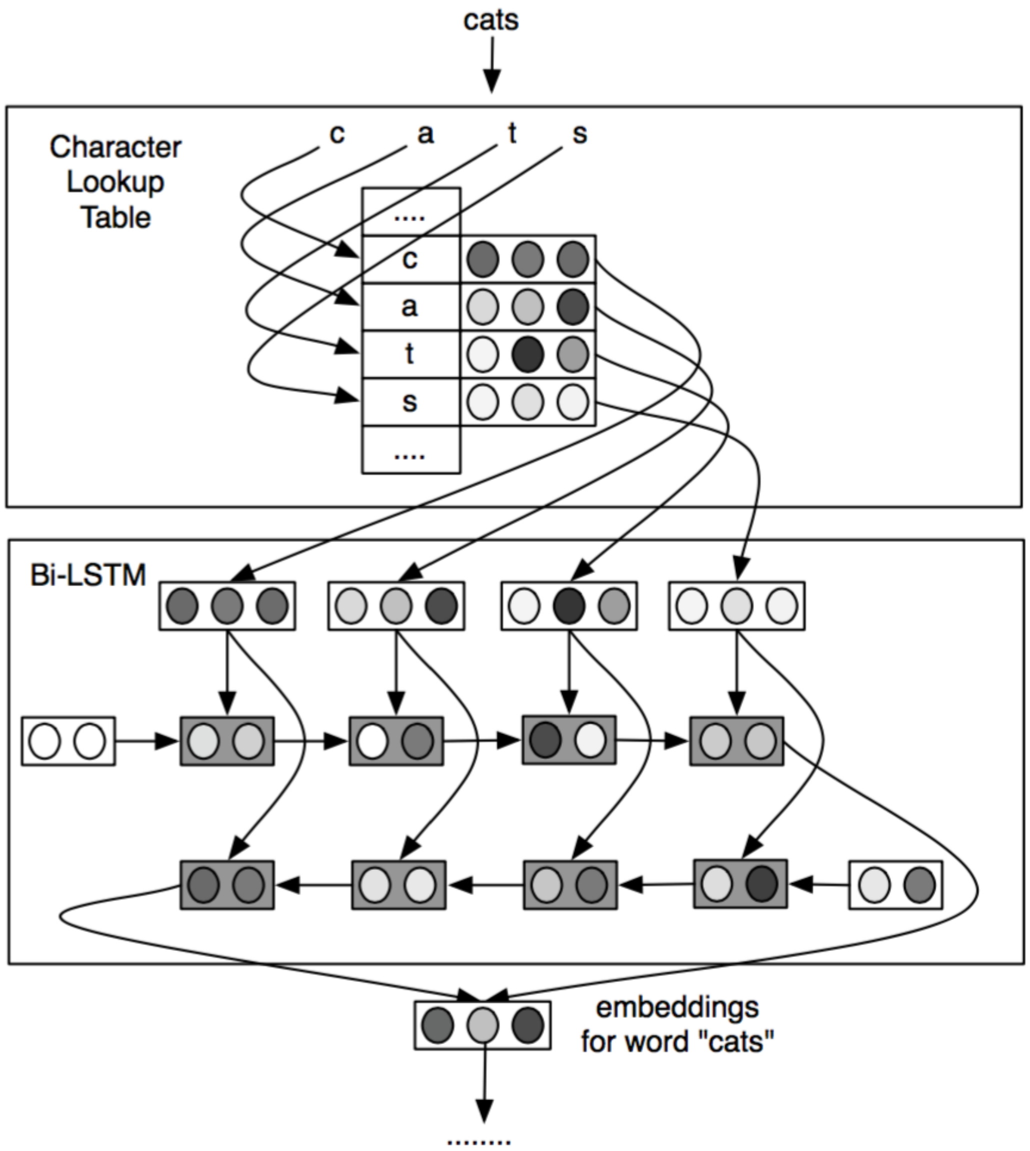


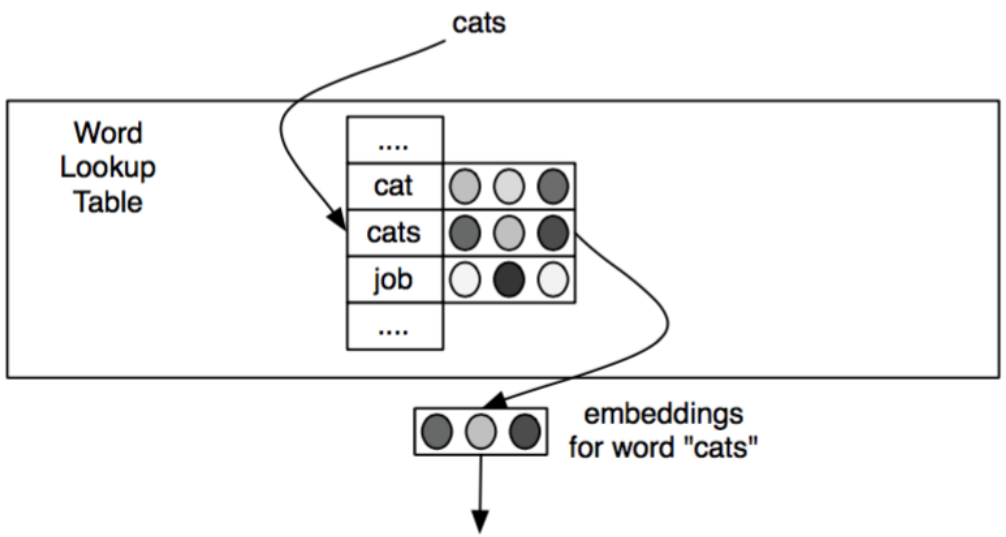
- Present the character sequence forwards and backwards to two separate LSTMs.
- Yields the forward state sequence $s_1^f, \ldots, s_m^f$ and backward state sequence $s_m^b, \ldots, s_1^b$.
- The network has simultanious access to all inputs before and after the current one.
- No need for fixed window sizes for the input, the net decides how much context to use.

C2W-Model: Combining Layer

- **Combines the last forward state s_m^f and the last backward state s_1^b**
- $e_w = D^f s_m^f + D^b s_0^b + b_d$
- **The variables D^f, D^b, b_d are the weights which determine how the states are combined.**
- **Automatically learns to determine how much each context is used.**

C2W-Model vs Lookup Tables





C2W-Model vs Word Lookup Tables

- Lookup Tables are conceptually much simpler, but requires a lot of parameters ($|V| \times d$).
- The C2W model uses less parameters (more in the evaluation section).
- The C2W model can easily be used for open vocabulary tasks.
- Looking up a word embedding is in $O(1)$, whereas the C2W model has to compute the embedding
  - Can be aliviated by caching e_w for frequently occuring words.
  - However cached values still need to be recomputed when parameters change during training time.

Experimentation

We are going to introduce three use cases, where a C2W based model either:

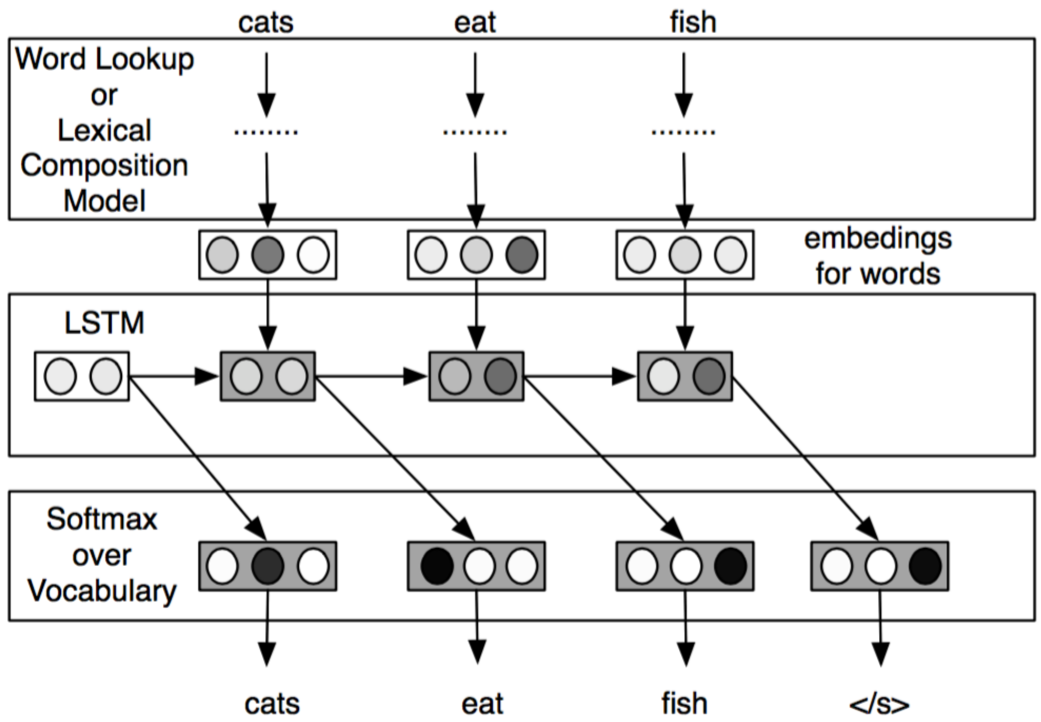
- Outperforms a model which uses much more parameters.
- Yields comparable results without relying on manually engineered features.

Wang Ling et al 2015 : Language Modelling

Wang Ling et al 2015 : Part-Of-Speech Tagging

Faruqui et al 2015 : Morphological Inflection Generation

Application 1: Language Modeling



- Computes the joint probability for the word sequence $w_1, \ldots, w_{i-1}$ with a hidden LSTM layer.
- Test two versions of this NLM: One with the C2W model and one with a word lookup table as projection layer.
- Compare accuracy of these two versions.

Application 1: Training & Testing Data

- **Perform testing on English, Portuguese, Catalan, German and Turkish.**
- **Choosen because these are morphologically rich languages.**
- **Training data was obtained by randomly extracting wikipedia articles until 1 million words were obtained.**
- **Additionaly 20000 words were obtained for testing.**

Application 1: Parameter Count

- The word embedding dimension is set to $d = 50$.
  - A word lookup table contains at least $d \times |V|$ parameters,
  - A language with 80000 words will have at least 4 million parameters.
- The C2W model has two additional hyperparameters which are set to $d_C = 50$ and $d_{CS} = 150$
  - The LSTMs use 8 matrices of size $d_{CS} \times d_C + 2d_{CS}$ (one for each decision gate).
  - The $d \times 2d_{CS}$ parameters in the combining output layer.
  - The $d_C \times |C|$ parameters in the character table.
  - For english this works out to roughly 180000 parameters.

Reminder: Perplexity

- **Measure of how well a probability distribution predicts sample data.**
- **Can be interpreted as the number of choices per word position.**
- **Defined as** $2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$
- **To minimize the perplexity value means to have a better fitting language model.**

Application 1: Evaluation

Perplexity	English	Portugese	Catalan	German	Turkish
Word Lookup	59.38	46.17	35.34	43.02	44.01
C2W Model	**57.39**	**40.92**	**34.92**	**41.94**	**32.88**
#Parameters					
Word Lookup	4.3M	4.2M	4.3M	6.3M	5.7M
C2W Model	**180K**	**178K**	**182K**	**183K**	**174K**

Perplexities and test configuration [Wang Ling et al 2015].

- **Training is performed with mini-batch gradient descent with 100 sentences each.**
- **Speed of both model versions is aproximatly 300 words per second, main bottleneck is the softmax layer.**
- **In general C2W outperforms word lookup tables and requires less parameters.**

Application 1: Nonce Words

Noahshire	phding
Nottinghamshire	mixing
Bucharest	modelling
Saxony	styling
Johannesburg	blaming
Gloucestershire	christening

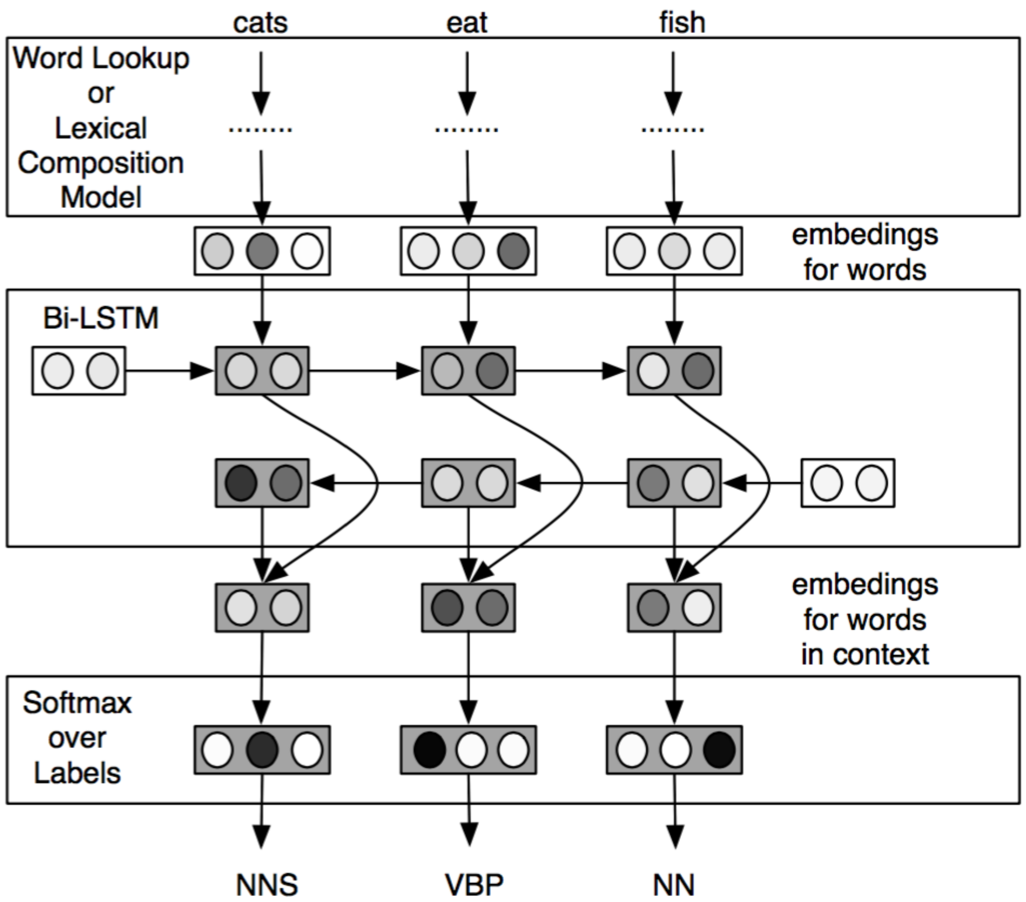
Nonce words and their most similar words from the vocabulary [Wang Ling et al 2015].

- **Nonce words are words created for use in a single occasion.**
- **The C2W model is able to generate embeddings for these words.**
- **No need for an OOV token for out of vocabulary words.**

Application 2: Part-Of-Speech Tagging

- **Process of labeling words as corresponding to a particular part of speech**
- **For example a simple tagging would be to identify words as nouns, verbs, adjectives, adverbs, etc.**
- **One use of this is to disambiguate homonyms:
  For Example "I fish a fish" should become "Je pêche un poisson" in french.**
- **By tagging the first occurance of "fish" as verb and the second as noun, they are now distinct.**

Application 2: Part-Of-Speech Tagging



- Conceptually similar to the previous NLM model.
- We don't just use a single LSTM block, but a bidirectional LSTM.
- The softmax layer computes over all possible labels instead of the vocabulary

Application 2: Testing Setup

- For english annotated sentences of the Wall Street Journal from the "Penn Treebank" dataset are used.
- For other languages data provided by the "Conference on Natural Language Learning" was used.
- The dimension of the states in the additional Bi-LSTM layer are set to 50.
- Out of vocabulary words are replaced with an OOV token.

Application 2: Evaluation

	acc	parameters	words/sec
Word Lookup	96.97	2000k	6K
Convolutional (S&Z)	96.80	42.5k	4K
Forward RNN	95.66	17.5k	4K
Backward RNN	95.52	17.5k	4K
Bi-RNN	95.93	40k	3K
Forward LSTM	97.12	80k	3K
Backward LSTM	97.08	80k	3K
Bi-LSTM $d_{CS} = 50$	97.22	70k	3K
Bi-LSTM	**97.36**	150k	2K

- As previously the C2W based POS-model is compared with versions using different word representation models.
- Table contains accuracies for the english WSJ dataset only.
  - There are different configurations using regular RNNs and LSTMs.
  - The LSTMs always outperforms regular RNNs by about 2%.
  - Row "Convolutional (S&Z)" contains results of a convolutional model from [Santos and Zadrozny, 2014].

[Santos and Zadrozny, 2014]: Learning Character-level Representations for Part-of-Speech Tagging

Application 2: Evaluation

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

- This table contains testing results for a number of languages.
- As previously the row "word" contains results for a model version using word lookup tables.
- Additionally it is compared with Stanford's POS tagger, with the default set of features.

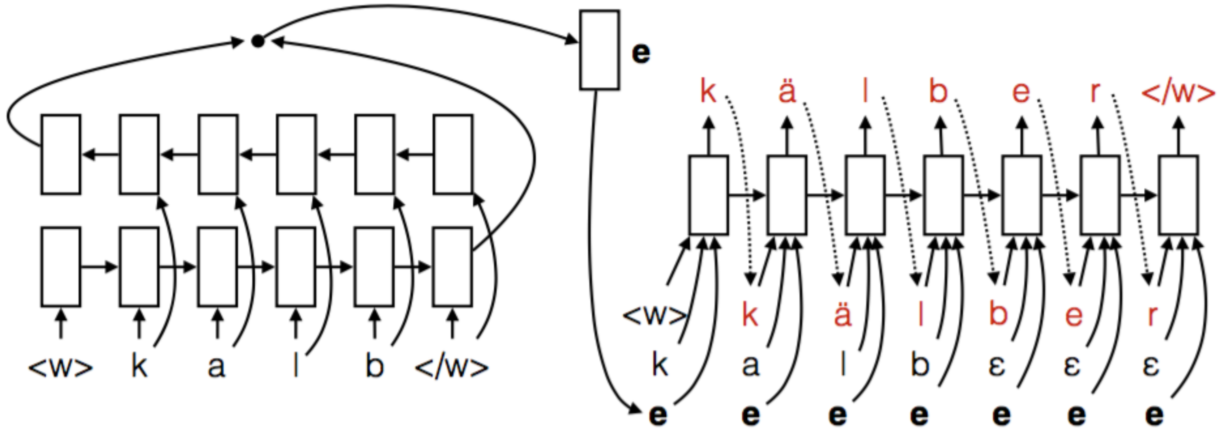
Application 3: Morphological Inflection Generation

	singular	plural
nominative	Kalb	Kälber
accusative	Kalb	Kälber
dative	Kalb	Kälbern
genitive	Kalbes	Kälber

Example of an inflection table for the word "Kalb" [Faruqui et al 2015]

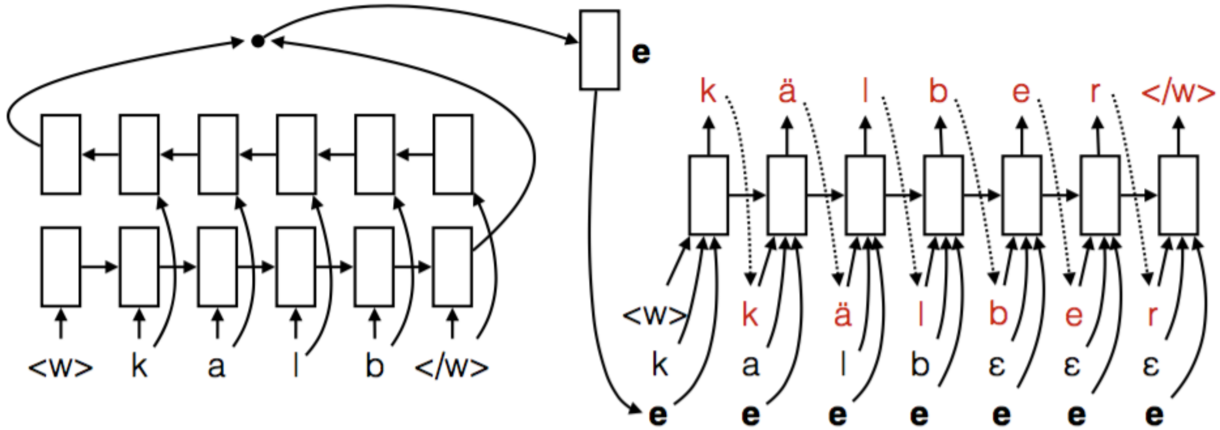
- **We want to perform morphological transformations of words.**
- **These kind transformations are very common in languages like turkish or german.**
- **Could be used as post- or preprocessing step for machine trainlation.**

Application 3: Overview



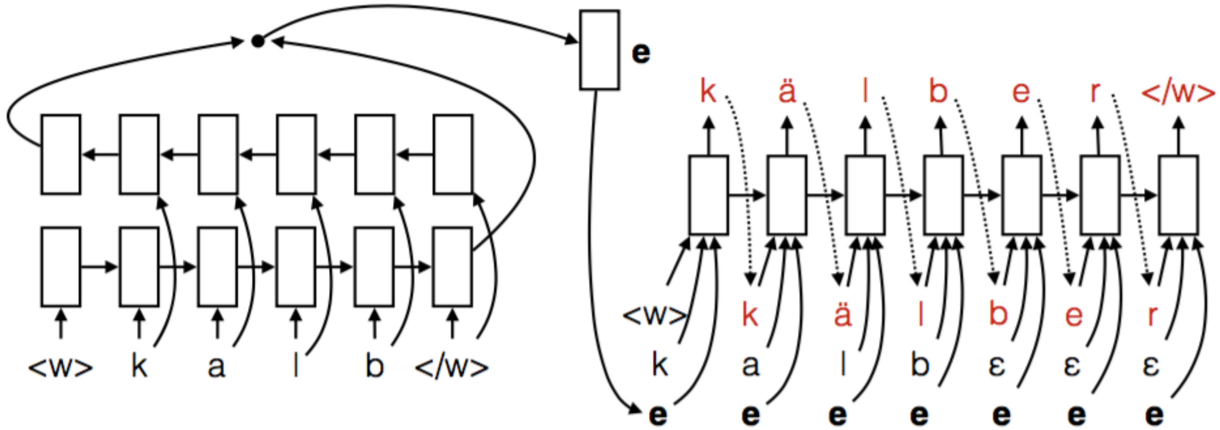
- **We use a neuronal encoder - decoder architecture.**
- **The word embedding e is used as an intermediate representation.**
- **The decoder constructs the inflected version of the word - character by character.**

Application 3: Overview



- The encoder part is identically to the C2W model and generates a word embedding e.
- The decoder is just an LSTM unit which receives the following inputs each timestep:
  1. The word embedding e from the encoder.
  2. Current character of the original word c_j
  3. Previous output of the model

Application 3: Decoder



- **The decoder output is driven by the characters of the original word**
- **There is a chance that output length is greater than the input word length.**
- **Once the input word ends, the ϵ character is used instead.**
- **The decoder stops by outputting the word end token "$<$/w$>$"**

Application 3: Training & Testing Data

Dataset	root forms	Infl.
German Nouns (DE-N)	2764	8
German Verbs (DE-V)	2027	27
Spanish Verbs (ES-V)	4055	57
Finnish NN & Adj. (FI-NA)	6400	28
Finnish Verbs (FI-V)	7249	53
Dutch Verbs (NL-V)	11200	9
French Verbs (FR-V)	6957	48

The languages above were tested with the data published by:

- **[Durrett and DeNero 2013] containing inflections for German, Finnish and Spanish.**
- **[Nicolai et al., 2015] adding dutch and french to this dataset.**
- **The development and test sets contain about 200 inflection tables each.**

[Durrett and DeNero 2013]: Supervised learning of complete morphological paradigms. In Proc. of NAACL.

[Nicolai et al., 2015]: Inflection generation as discriminative string transduction. In Proc. of NAACL

Application 3: Evaluation

	DDN13	NCK15	Ours
DE-V	94.76	**97.50**	96.72
DE-N	88.31	**88.60**	88.12
ES-V	99.61	99.80	**99.81**
FI-V	97.23	**98.10**	97.81
FI-NA	92.14	93.00	**95.44**
NL-V	90.50	96.10	**96.71**
FR-V	98.80	**99.20**	98.82
Avg.	94.47	96.04	**96.20**

- **The results are comparable or better than other approaches.**
- **On average the results are better.**
- **No feature engineering necessary.**

Summary

- Generating word embeddings by composing character representations, works usually just as well as approaches using word lookup tables.
- Lexical features can be learned automatically, manual feature engineering can be avoided
- In combination with caching of frequently used words, the performance is comparable to models based on word lookup tables.
- Models scale better with larger vocabularies and are able to deal with open vocabularies.