Character-based Embeddings of Words with Recurrent Nets

Simon Grätzer

Seminar Selected Topics in Human Language Technology and Pattern Recognition – MedBV16 June 10, 2016

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

Outline

1. Introduction
2. Word Embeddings
3. Generating Word Embeddings

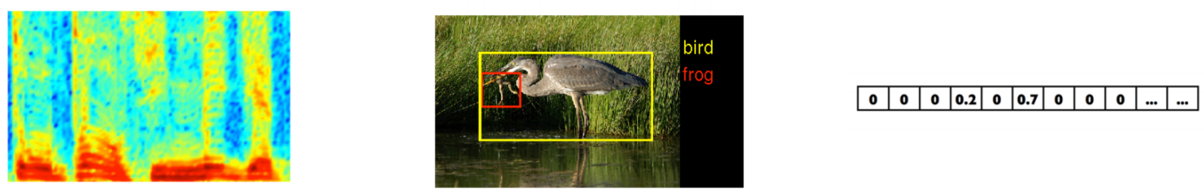
Literature

-
- [?] The tensorflow doumentation by Google Inc.

Introduction

- Word embeddings are real valued vector representations for words.
- In this talk I will present a new idea to generate these representations.
  - Using recurrent neural networks (LSTM)
  - Using individual representations of the characters as inputs.
- The resulting model can be used to improve some taks, such as language modeling or part-of-speech tagging

AUDIO IMAGES TEXT



Audio Spectrogram — DENSE

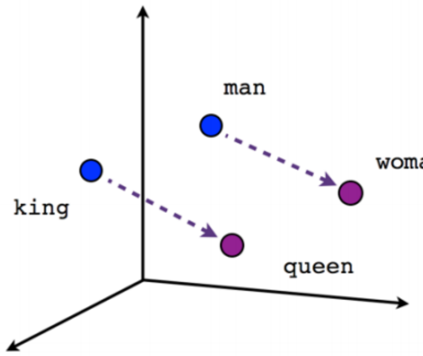
Image pixels — DENSE

Word, context, or document vectors — SPARSE

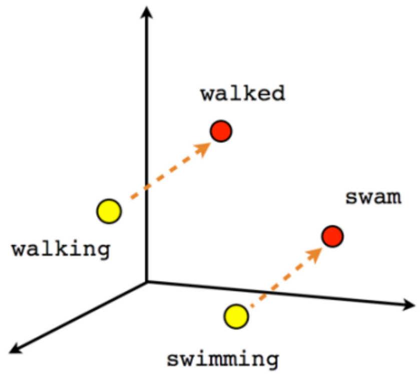
Different input datasets compared to word data [?].

The underlying problem is the sparsity of ordinary word representations.

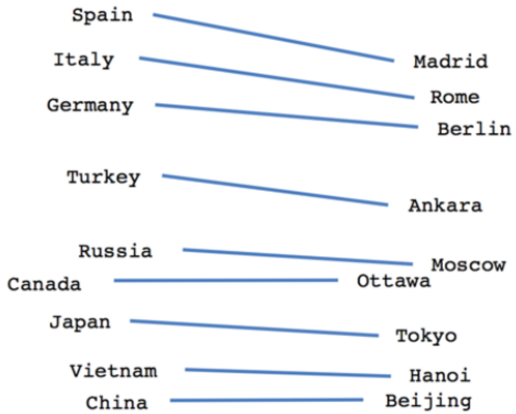
Word Embeddings



Male-Female



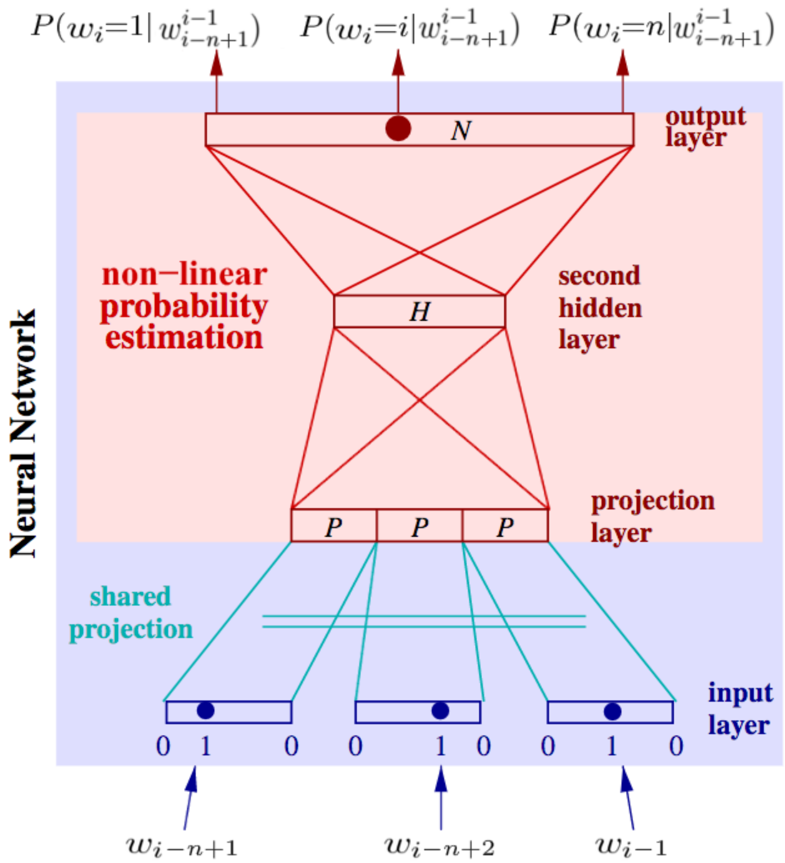
Verb tense



Country-Capital

- **Words which share semantic meaning tend to occur in the same contexts (Distributional Hypothesis [?])**
- **A model can learn relationships between words and represent them.**
- **Words with a similar meaning should be mapped to nearby points in the same vector space.**
- **This captures the intuition that words may be similar along a variety of ways.**

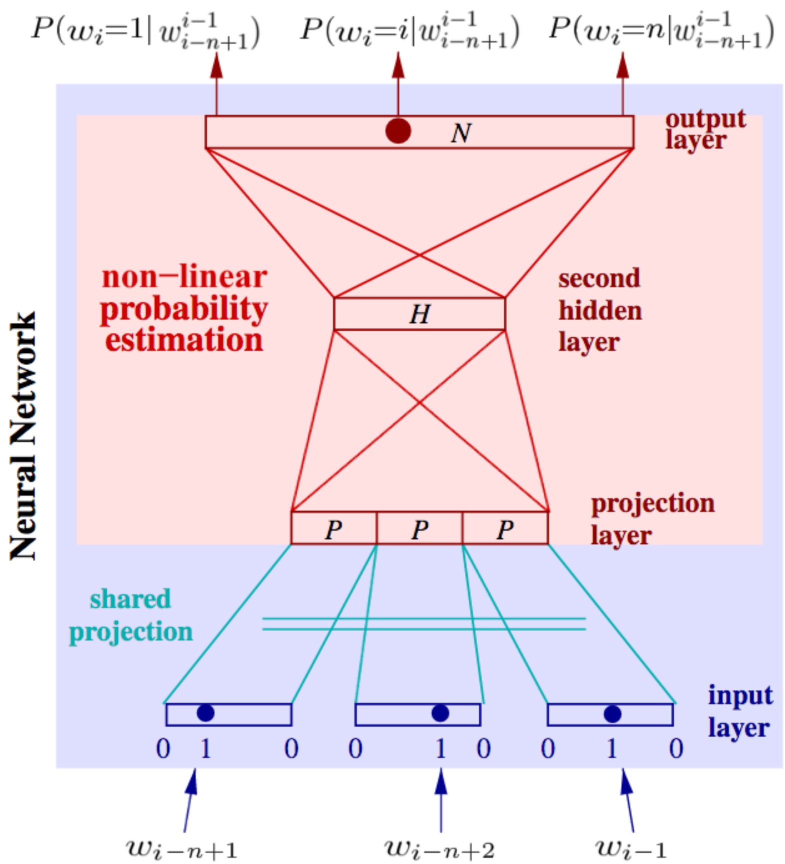
Simple Word Embeddings in a Language Model



Classical natural language model

- **Language modelling estimates $p(w_1,\ldots,w_m) = \prod_{i=1}^{m} p(w_i|w_1,\ldots,w_{i-1})$**
- **The context is approximated with the previous $n-1$ words (n-grams)**
- $p(w_1,\ldots,w_m) = \prod_{i=1}^{m} p(w_i|w_1,\ldots,w_{i-1}) \approx \prod_{i=1}^{m} p(w_i|w_{i-(n-1)},\ldots,w_{i-1})$

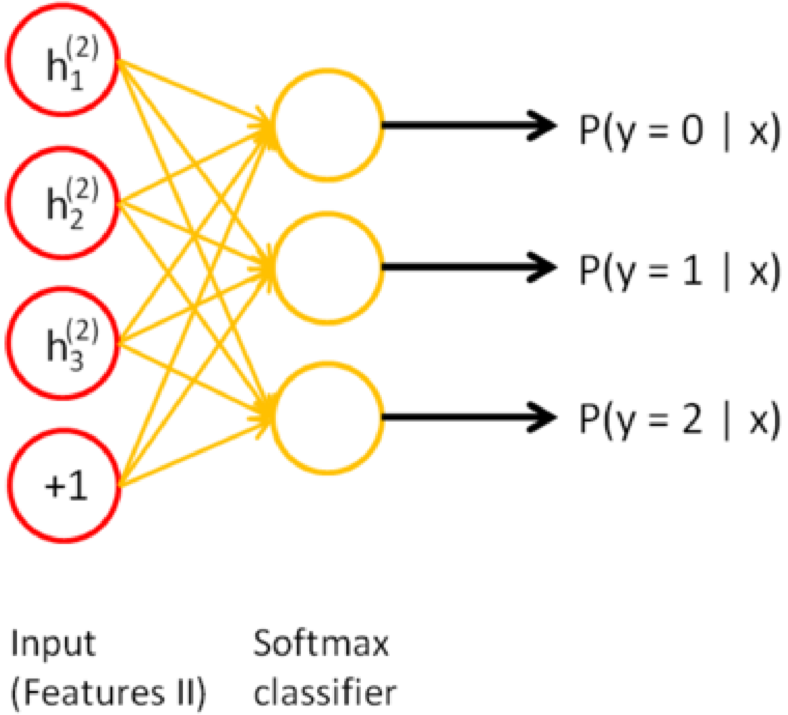
Simple Word Embeddings in a Language Model



Classical natural language model

- **Word Embeddings are trained end to end, as part of the language model**
- **The word embeddings is essentially the output of the first hidden layer for a word.**
- **The embeddings are stored in a lookup table $P \in \mathbb{R}^{|V|\times d}$. An embeddings is calculated as $e_{w_i}^{W} = P * 1_{w_i}$**

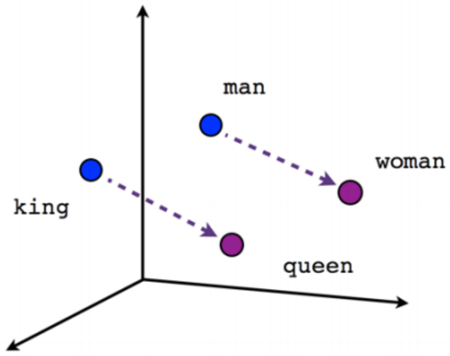
Repetition: Softmax-Layer



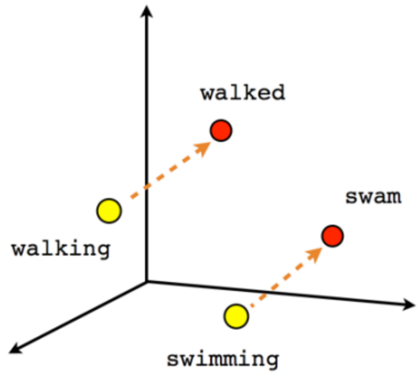
$$p_k = \sigma(\mathrm{z})_k = \frac{e^{z_k}}{\sum_{k=1}^{|V|} e^{z_k}}$$

- The input vector z is computed over the vocabulary: $z_k \forall k \in V$.
- The result of the outputs can be interpreted as posterior probabilities.
- Probability given the context: $p_k = p(w_i = k | w_{i-n+1}^{i-1})$

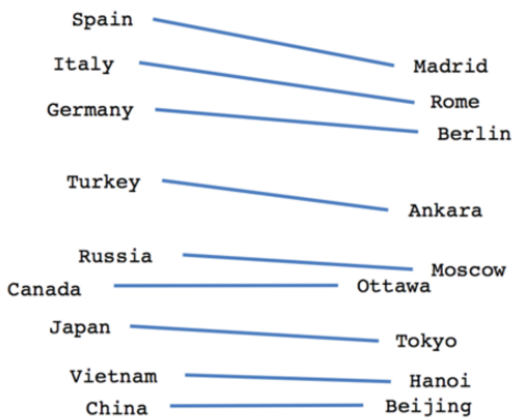
Advanced Word Embeddings: Word2vec



Male-Female



Verb tense



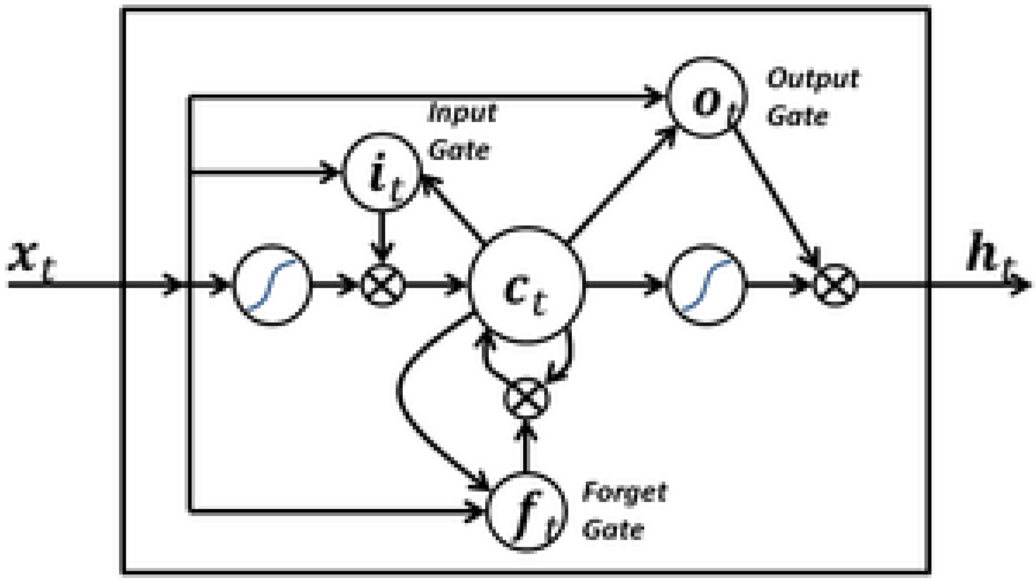
Country-Capital

- **The Skip-Gram based model from Mikolov et.al. was developed at Google.**
- **They use n-gram's but invert the model:** $\sum_{-k\leq j-1,\, j\leq k} \log P(w_{t+j}|w_t)$
- $v(\text{king}) - v(\text{male}) + v(\text{female}) \approx v(\text{queen})$
- **Resulting lookup table of embeddings can be reused for other tasks.**

Drawbacks of these Embeddings

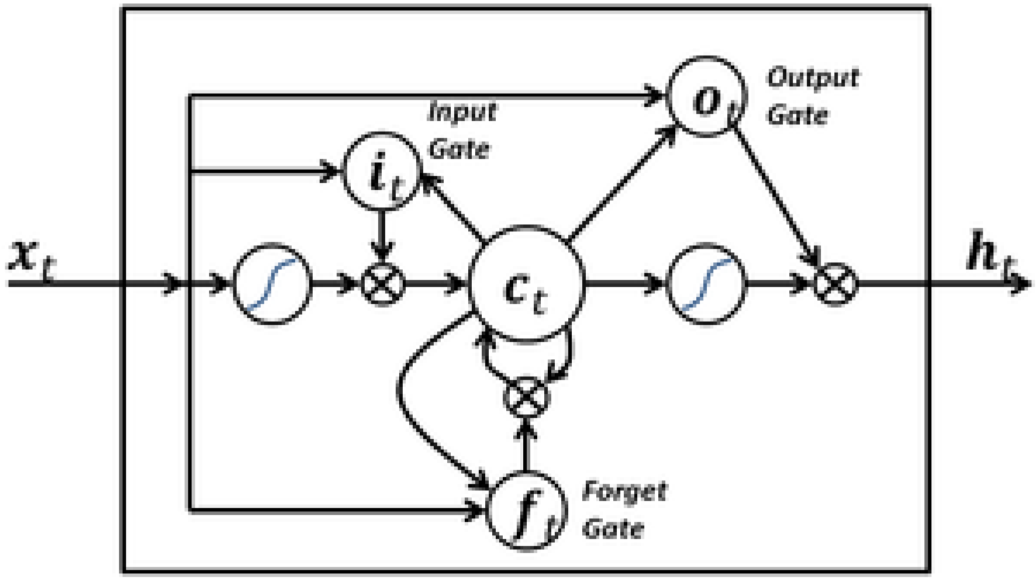
- **Each word embedding vector is completly independent.**
  - **The model captures smililar linear correspondences between words embeddings i.e. *cat* and *apple* compared to *cats* and *apples***
  - **It doesn't capture that the added *s* is responsible for this transformation.**
  - **A word lookup table cannot generate representations for an unknown word.**
  - **Even if it's just the plural form of a known word.**
- **For a large vocabulary it becomes impractical to actually store all word embeddings in a table.**

Repetition: Long-Short Term Memory



- Designed to "remember" inputs over long distances and "forget" them when necessary
- Works well for time series data
- 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.

Repetition: Long-Short Term Memory

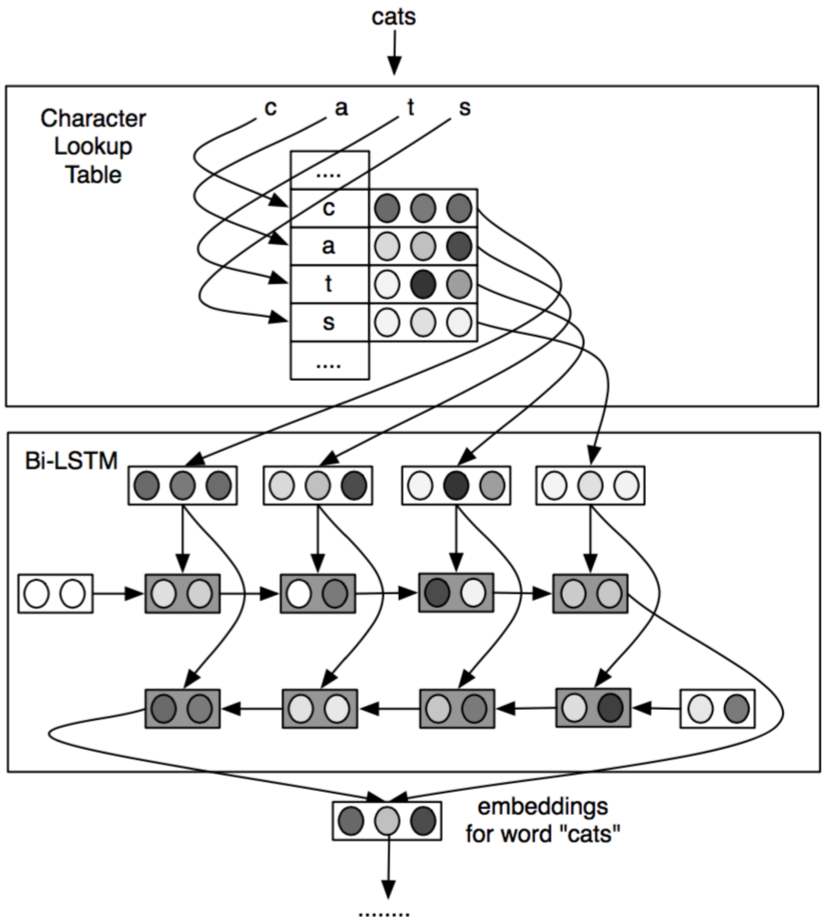


Given the input vectors $x_1, \ldots, x_m$ a LSTM computes the output sequence $h_1, \ldots, h$

$$
\begin{aligned}
i_t &= \sigma(W_{ix} * x_t + W_{ih} * h_{t-1} + W_{ic} * c_{t-1} + b_i) \\
f_t &= \sigma(W_{fx} * x_t + W_{fh} * h_{t-1} + W_{fc} * c_{t-1} + b_f) \\
o_t &= \sigma(W_{ox} * x_t + W_{oh} * h_{t-1} + W_{oc} * c_t + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{cx} * x_t + W_{ch} * h_{t-1} + b_c) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
\tag{1}
$$

LSTM's avoid the vanishing gradient problem, because the activation function in c_t is the identity function.

Character-based Word-Embeddings (C2W)



Character lookup table on top, bidirectional LSTM on the bottom

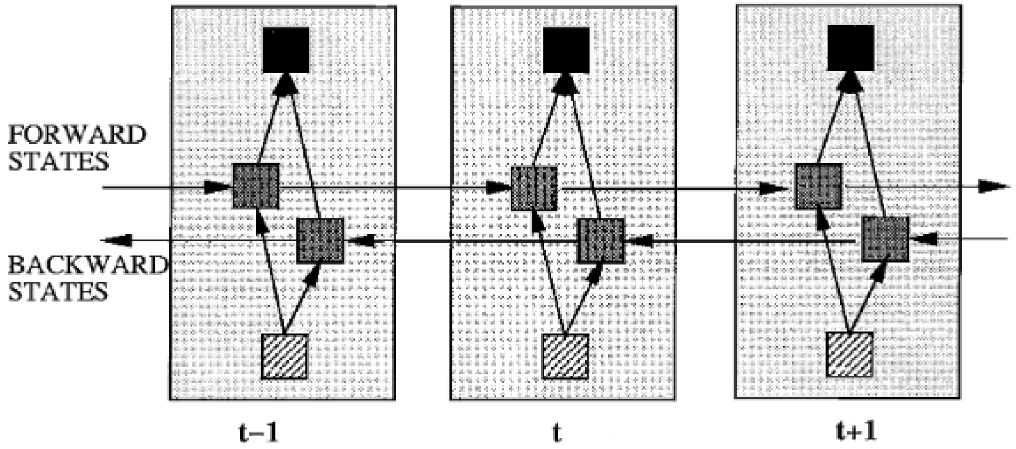
Overview:

- **A word with length m is composed of characters $c_1, \ldots, c_m$.**
- **Decompose each word into a sequence of character embeddings $e^C_{c_1}, \ldots, e^C_{c_m}$ from the alphabet C.**
- **The sequence is fed to two LSTM units (Bidirectional LSTM), forwards and backwards.**
- **In the end the result is combined by an output layer.**

C2W-Model: Character-Lookup Table

- Table of d_C parameters $P_C \in \mathbb{R}^{d_C \times |C|}$ for each character from a predefined character alphabet C.
- Each Input character is transformed into a d_C-dimensional feature vector $e_{c_j}^C$.
- We define the projection of characters as $e_{c_j}^C = P_C * 1_{c_j}$.
- Similar to the previous projection layer for each word.

C2W-Model: Bidirectional LSTM Layer



- Present every input sequence forwards and backwards to two separate recurrent neural networks
- Both RNN's are connected to the same output layer.
- 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.

- **Yields the forward state sequence $s_0^f, \ldots, s_m^f$ and backward state sequence $s_m^b, \ldots, s_0^b$.**

C2W-Model: Output Layer

- **Table of d_C parameters $P_C \in \mathbb{R}^{d_C \times |C|}$ for each character from a predefined character alphabet C.**
- **Each Input character is transformed into a d_C-dimensional feature vector $e_{c_j}^C$.**
- **We define the projection of characters as $e_{c_j}^C = P_C * 1_{c_j}$.**
- **Similar to the previous projection layer for each word.**

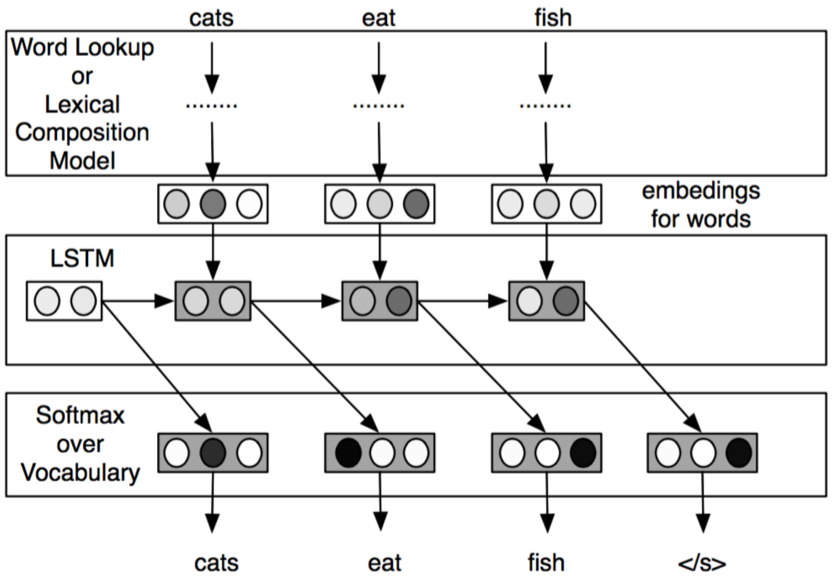
Character-based Word-Embeddings: Advantages

- Simply breaks-up words into simple atomic units.
- Characters are the simplest atomic unit of words.
- Aternative: Use Morphemes as atmoic units
  - A morpheme is the smallest grammatical unit of a language
  - e.g. "Unbreakable" comprises three morphemes: un-, -break-, and -able.
  - Would require a morphological analyser

Output-Layer

- Combines the last states of the forward sequence s_m^f and the backwards sequence s_0^b
- $e_w^C = D^f s_m^f + D^b s_0^b + b_d$
- The variables D^f, D^b, b_d are the weights which determine how the states are combined.
- Automatically learns how much each context is used.

Application: Language Modeling



- Uses the word embeddings from the C2W model combined with a LSTM unit.
- Every time we input a new word w_i from the sequence the model yields the LSTM state s_i.
- In the end a softmax layer is used to compute the likelihood $p(w_i = k|w_{i-n+1}^{i-1})$

Application: Language Modeling - Evaluation

- **TODO include the evalutation tables**

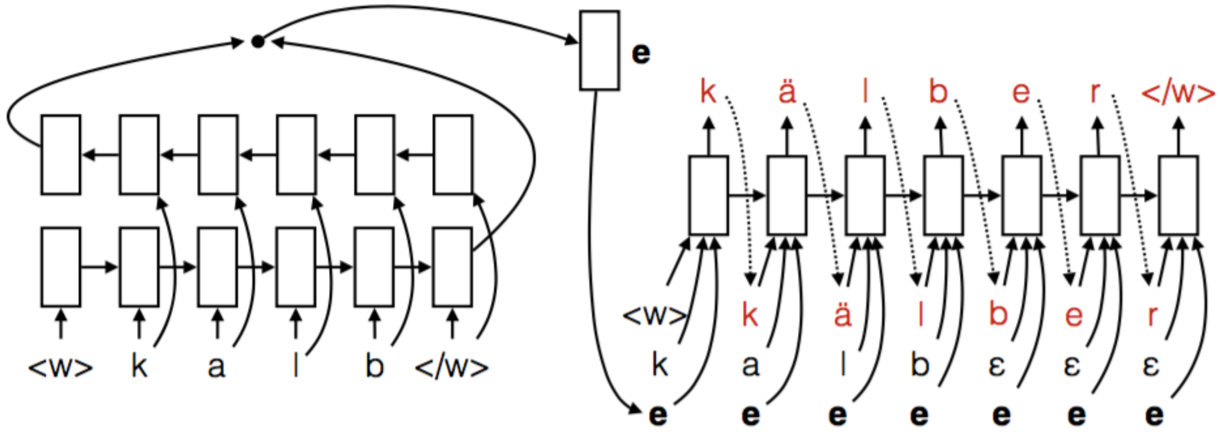
Applications: Morphological Inflection Generation

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

Example of an inflection table for the word "Kalb"

- **Perform morphological transformations, as discussed in Faruqui et al. [?]**
- **The transformations are very common in languages like turkish or german.**
- **Basic idea is to use a neuronal encoder - decoder architecture.**
- **The encoder mirrors the C2W model.**

Application: Morphological Inflection Generation



- **First: Encoder is virtually identically to C2W model and generates a word embedding e_w.**
- **The decoder is just an LSTM unit which receives the following inputs each timestep:**
  1. **The word embedding e_w from the encoder.**
  2. **Current character of the original word c_j**
  3. **Previous output of the model**
- **Once the input word ends, the ϵ character is used instead.**

Summary

- Calculating the word embeddings is cheaper than storing them in huge lookup-tables
- Performance is comparable to other methods
- Lexical features can be learned automatically
- Redundancies in lookup-tables are avoided
- Models scale better with larger vocabularies.

Backup: 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.

Backup: Out of Vocabulary Token

-

Backup: Out of Vocabulary Token

-