Elements of Data Science and Artificial Intelligence

-- Language Models --

WS 2019/2020

Vera Demberg

What's a language model?

A statistical language model is a probability distribution over a sequence of words.
Given a sequence of words $w_1 \ldots w_n$, it assigns a probability $P(w_1 \ldots w_n)$ to the sequence.

Language models can be evaluated by comparing how well they manage to "guess" a missing word in a sequence (or the next word in a sentence) given the beginning of a sentence.

Why do you need a language model?

Language models allow to estimate the likelihood of a sentence. This is useful for NLP applications where we want to generate text, as it allows us to quantify how “good” a text is.

- Speech recognition
- Machine translation
- Optical character recognition
- Handwriting recognition
- Summarization
- Language generation in chatbots or dialog systems

Why do you need a language model?

Language models allow to estimate the likelihood of a sentence. This is useful for NLP applications where we want to generate text, as it allows us to quantify how “good” a text is.

- **Speech recognition**
- Machine translation
- Optical character recognition
- Handwriting recognition
- Summarization
- Language generation in chatbots or dialog systems

Sprachverarbeitung

Schallsignal

↓

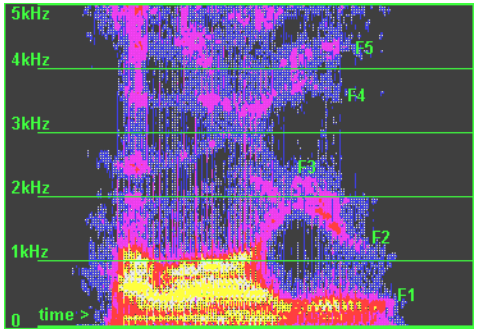
Wortkette

↓

Information

Spracherkennung

Sprachverstehen

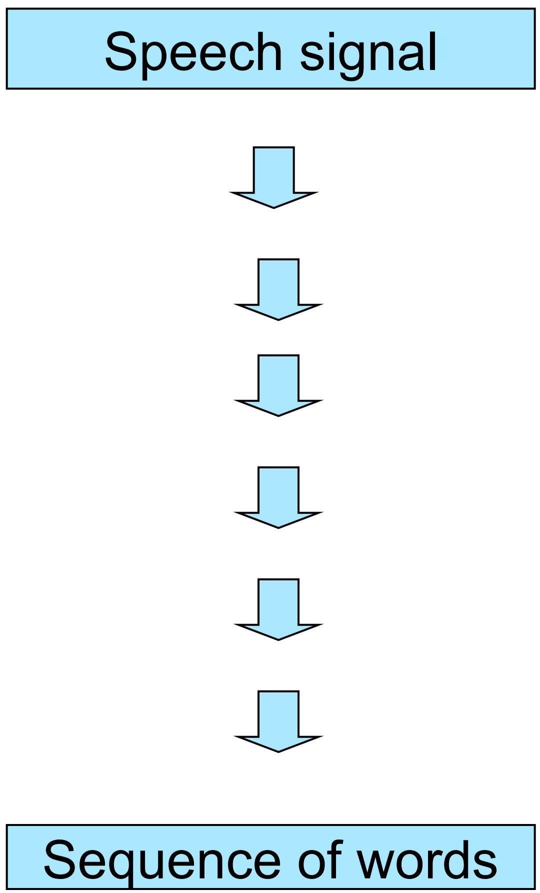


↓

Laura schläft

↓

Speech recognition



Basic challenge in speech recognition:

- Given a continuous speech signal, we need to determine what sequence of words was uttered.

Speech recognition

Speech signal

↓ Digital recording

Oszillogramm

↓

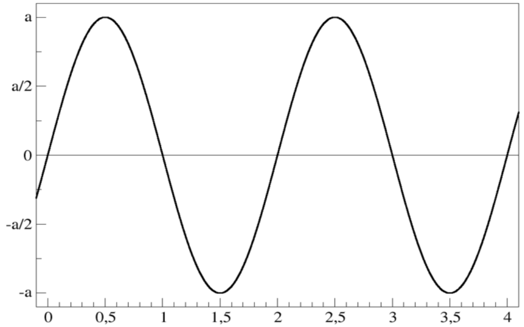
↓

↓

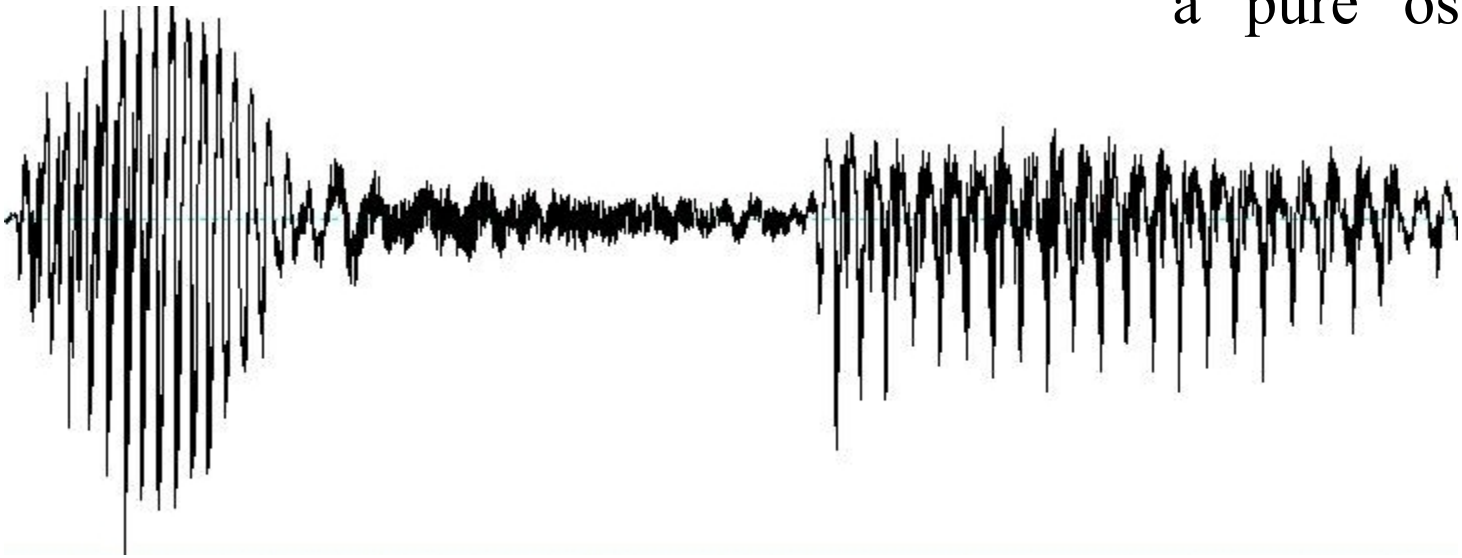
↓

Sequence of words

Ein Oszillogramm



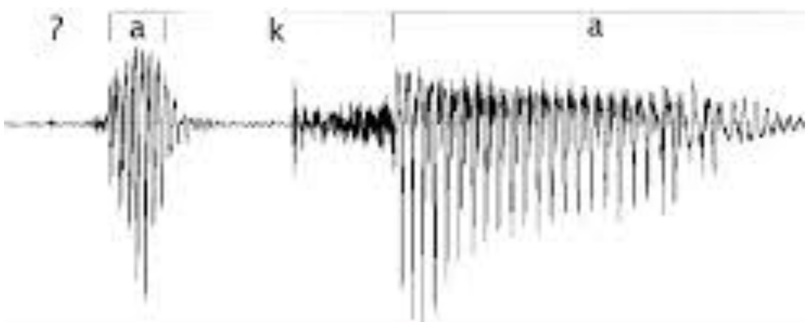
a “pure” oscillation



Visualization of oscillations for „afa“

Oscillations for other sounds

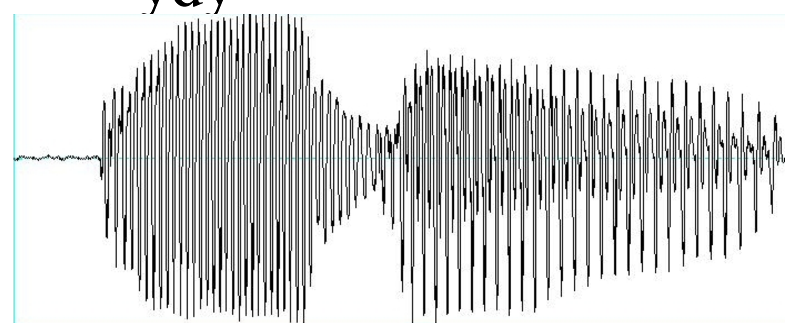
aka



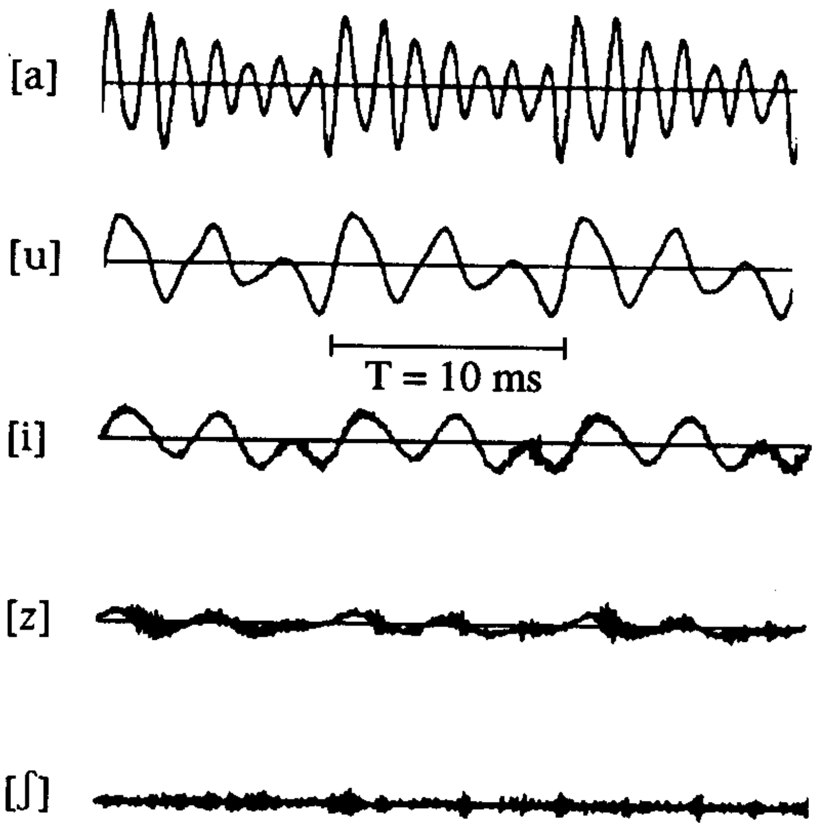
ama

acha

ydy

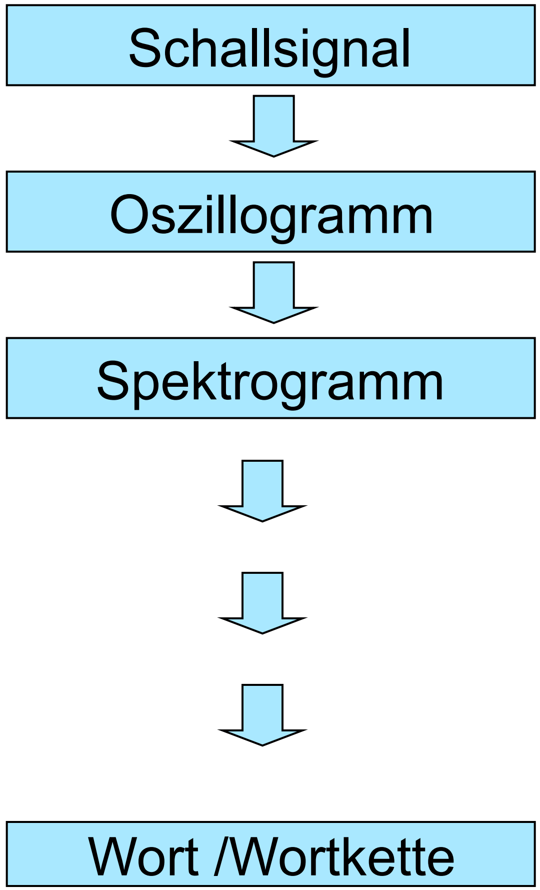


Einzelne Laute als Oszillogramme



- Sounds are characterized by combinations of oscillations in different frequencies.
- Frequencies are hard to see as they overlay each other.
- Therefore, fourier transform is used to analyse what components a complex oscillation consists of. The result is a spectrogram.

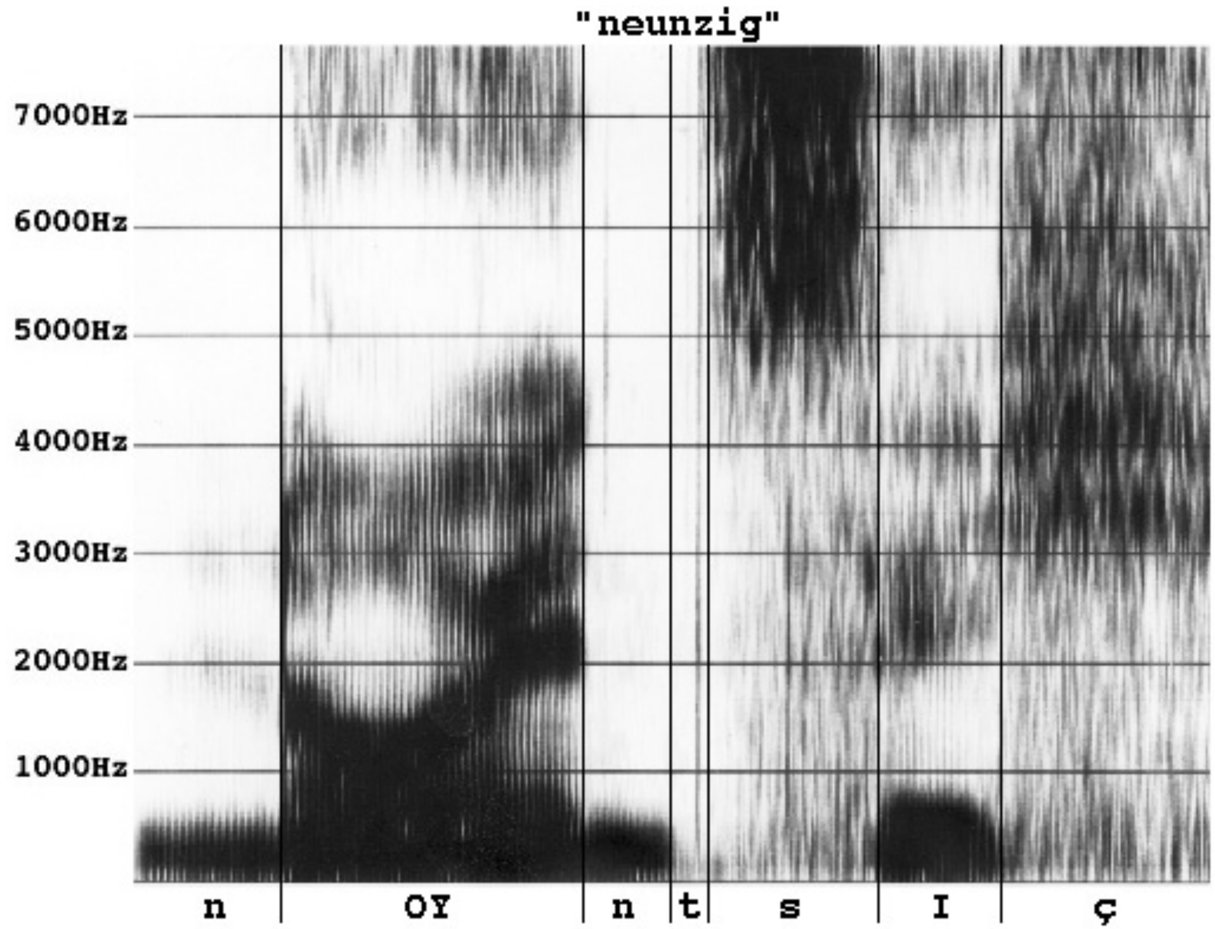
Spracherkennung: (Vereinfachtes) Schema



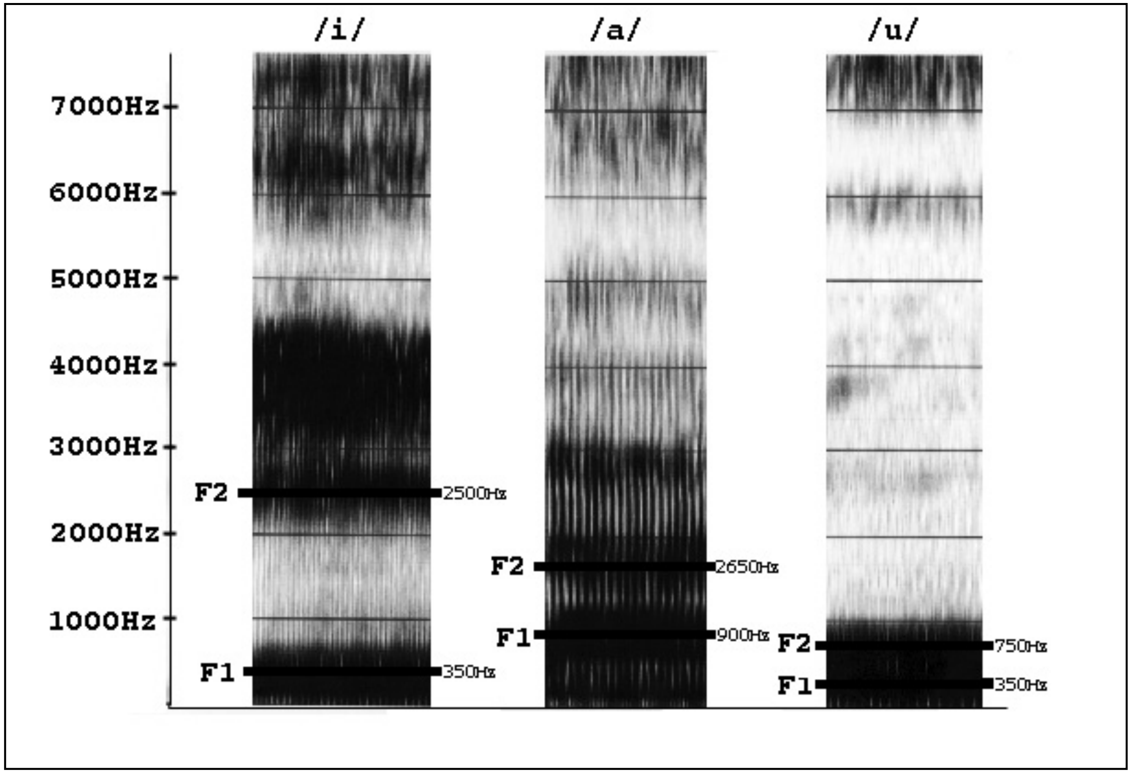
Digital recording

Analysis of frequencies contained in oscillations

Spektrogramm für eine Aufnahme von „neunzig“



Spektrogramm für die Vokale i,a,u



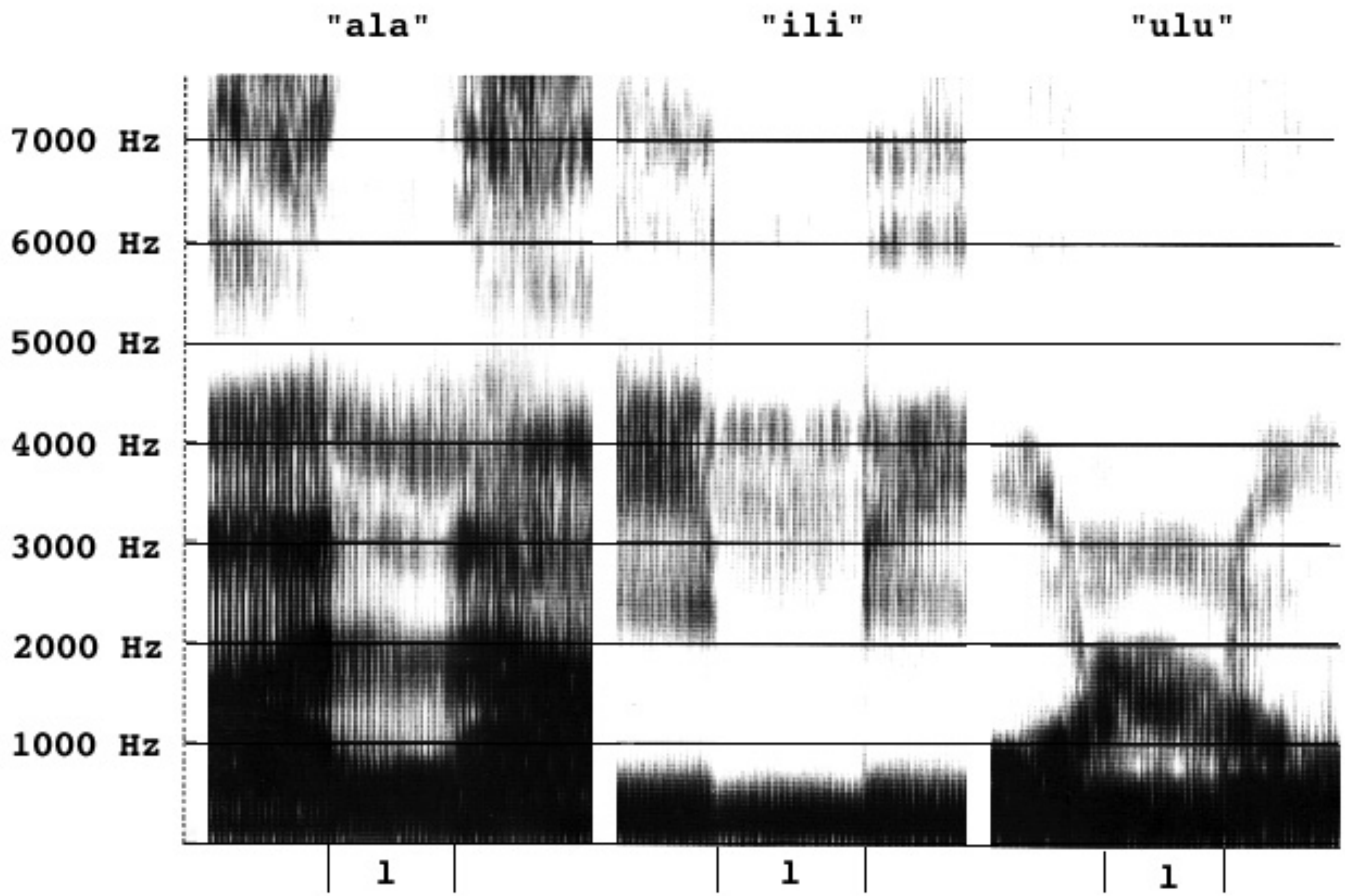
- Different vowels differ in terms of the frequencies at which there are high levels of energy.

How to get from the spectrogram to words

Just reading off the sounds from the spectrogram is hard, because of

- variance in the signal (different voices, dialects)
- continuity of the signal (no pauses between words)
- coarticulation

Koartikulation / Kontextabhängigkeit



How to get from the spectrogram to words

Just reading off the sounds from the spectrogram is hard, because of

- variance in the signal (different voices, dialects)
- continuity of the signal (no pauses between words)
- Coarticulation

Example for speech recognition output based only on acoustics:

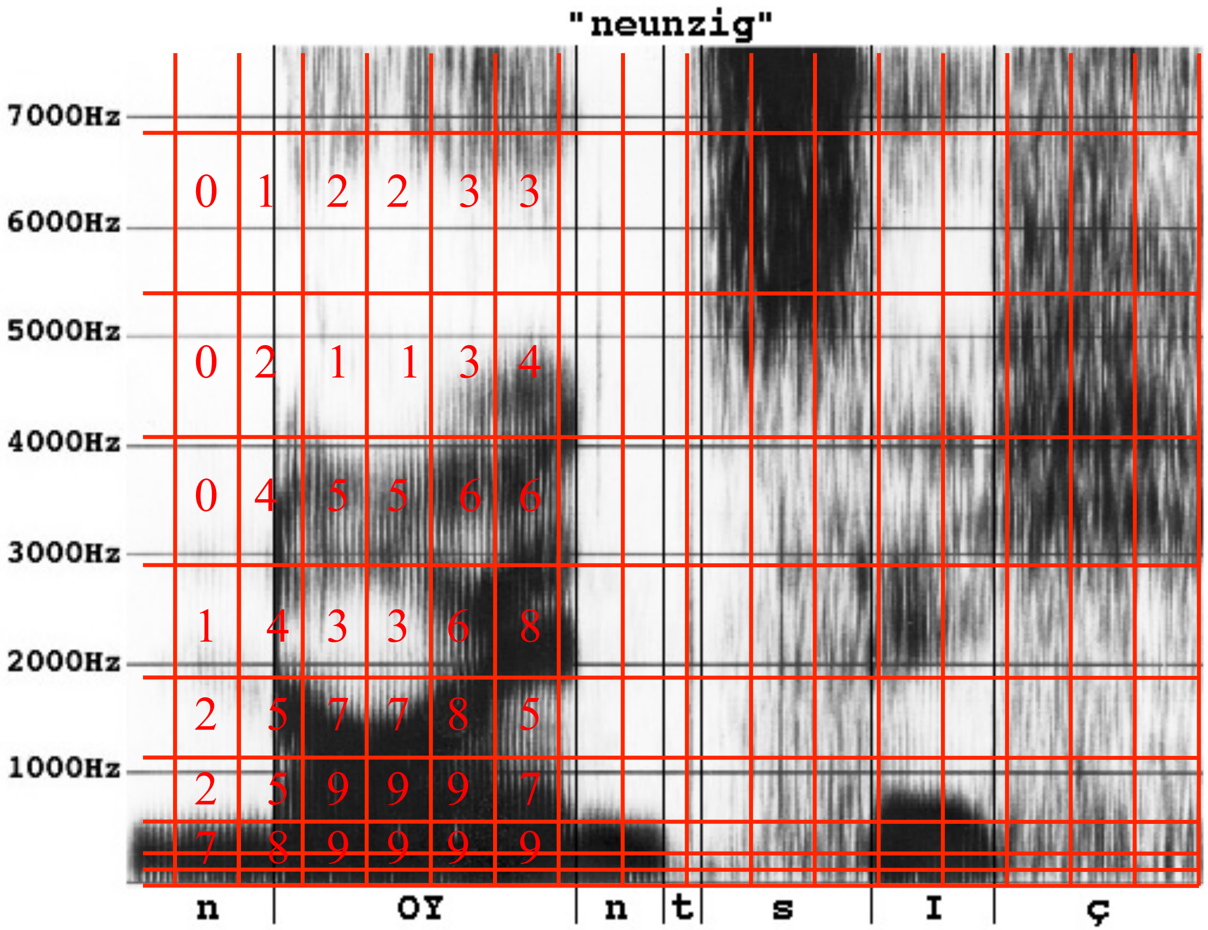
Input: *What is your review of linux mint?*

ASR output: WHEW AW WR CZ HEH ZZ YE AW WR OF YE WR ARE 'VE LENOX MAY AND

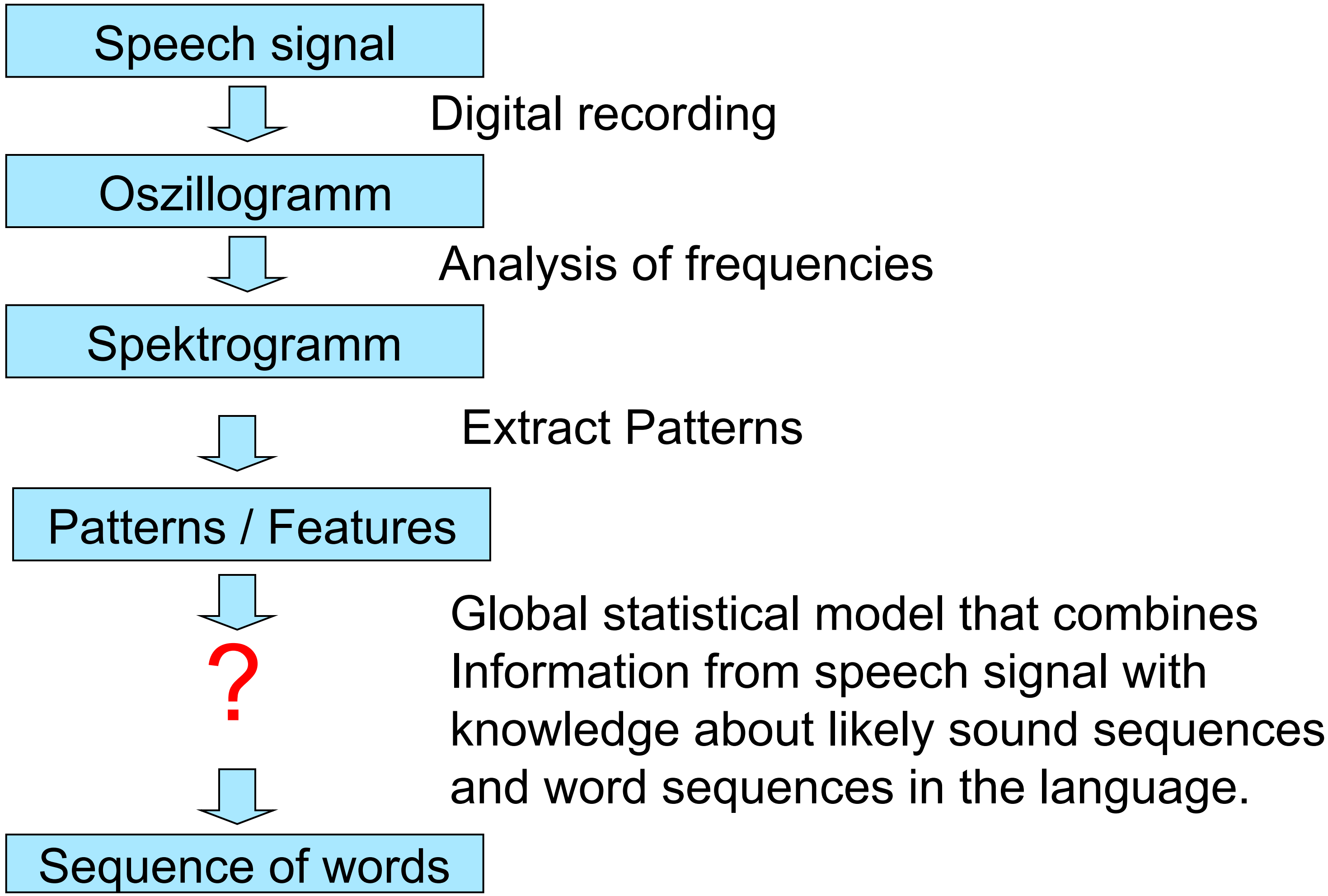
Learning from data

- It is in practice impossible to specify all combinations of sound intensities etc. for a mapping of what the sound might be.
- Therefore, data-driven approaches are used:
  - Annotate a recording with what was said on a sound by sound level
  - Convert the recording into features that can be used for ML
  - Train a (statistical or neural) model
  - Evaluate

Idea: split up time and frequency into little windows and note intensities, to make a feature vector which can then be mapped to sounds.



Speech recognition: (Simplified) Schema



Statistische Modellierung

- Task: estimate what word sequence $w_1 \ldots w_n$ is most likely given sound pattern sequence $O = o_1\, o_2 \ldots o_m$:

$$max_W\, P(W|O) = P(w_1 w_2 \ldots w_n |\, o_1\, o_2 \ldots o_m)$$

- This is very hard to estimate, because we may never have observed the exact sequence $o_1\, o_2 \ldots O_m$ before. => „**sparse data**“

- Using Bayes‘ Rule, we can instead estimate P(W|O) as follows:

$$P(W \mid O) = \frac{P(O \mid W) \cdot P(W)}{P(O)}$$

Wie bestimmen wir *P(W|O)*?

- **Symptom:** Folge von akustischen Beobachtungen $O = o_1\ o_2 \ldots o_m$
- **Ursache:**
  vom Sprecher geäußerte, intendierte Wortkette $W = w_1 w_2 \ldots w_n$
- Mit Bayes-Regel : $$P(W \mid O) = \frac{P(O \mid W) \cdot P(W)}{P(O)}$$

How do we estimate *P(W|O)*?

- Bayes rule : $P(W \mid O) = \frac{P(O \mid W) \cdot P(W)}{P(O)}$
- Most probable word sequence: $\max_W P(W \mid O) = \max_W \frac{P(O \mid W) \cdot P(W)}{P(O)}$
$$= \max_W P(O \mid W) \cdot P(W)$$
- *P(O)* is the probability of the speech pattern; we don‘t need it when caring only about the maximally probable word sequence.
- *P(O|W)* is the acoustic model (i.e., likelihood that a word is pronounced as a specific sound pattern sequence).
=> acoustic model
- *P(W)* is the probability of the word sequence $w_1...w_n$.
=> language model

How to get from the spectrogram to words

Just reading off the sounds from the spectrogram is hard, because of

- variance in the signal (different voices, dialects)
- continuity of the signal (no pauses between words)
- coarticulation

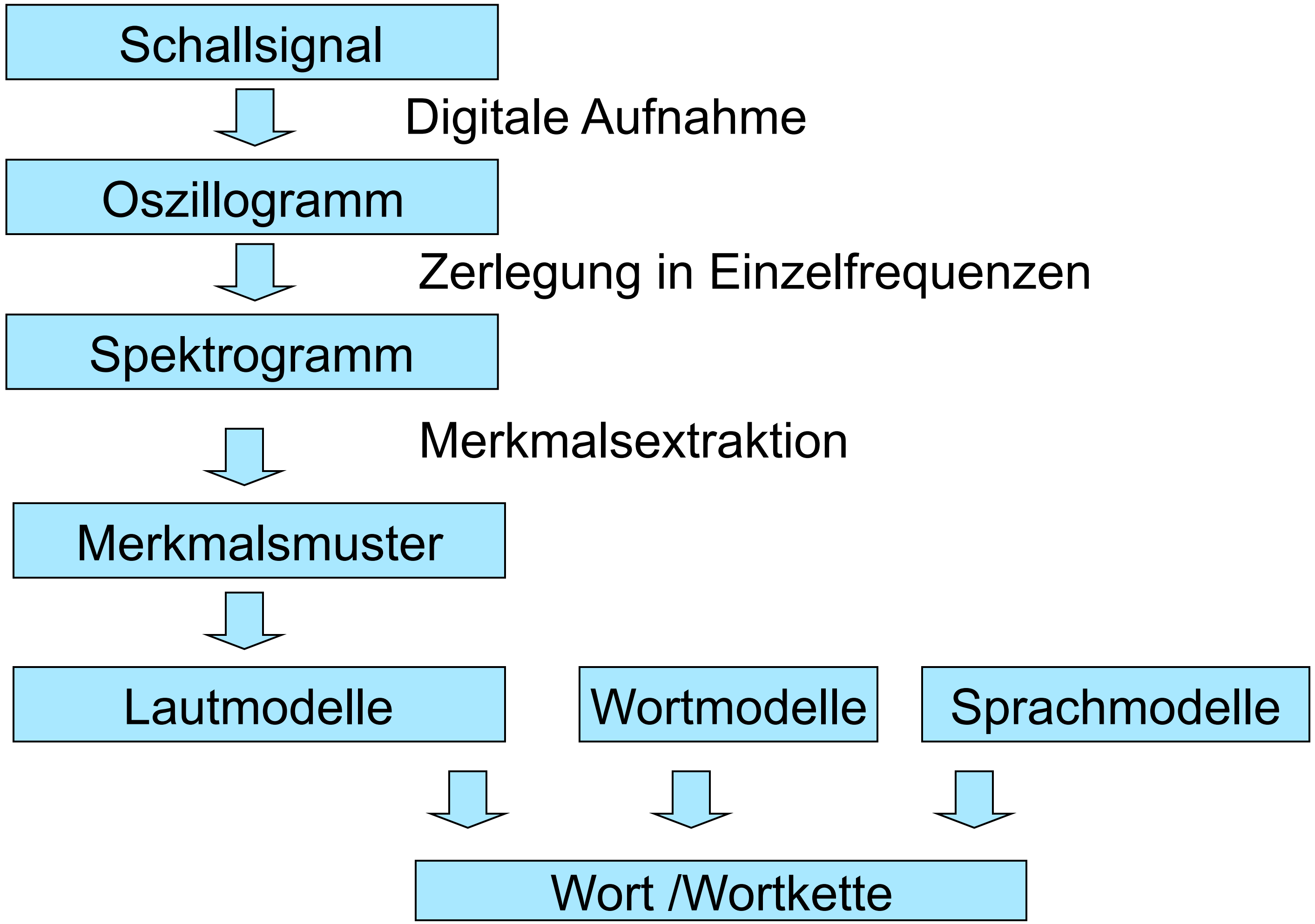
Example for speech recognition output based only on acoustics:

Input: *What is your review of linux mint?*

ASR output: WHEW AW WR CZ HEH ZZ YE AW WR OF YE WR ARE 'VE LENOX MAY AND

ASR output with language model: WHAT IS YOUR REVIEW OF LINUX MINT?

Speech recognition



Language models

- How can we estimate the probability of word sequence $P(W) = P(w_1 w_2 \dots w_n)$?
- We can estimate this from the frequency of word sequences in texts.
- But we still have a data sparsity problem: complete sentences have rarely been seen before; in fact, one can easily say a sentence that has never been said before.
- Chain rule allows us to reduce the joint probability $P(w_1 w_2 \dots w_n)$ to conditional probabilities:

  $P(w_1 w_2 \dots w_n)$

  $$= P(w_1) * P(w_2|w_1) * P(w_3|w_1 w_2) * \dots * P(w_n|w_1 w_2 \dots w_{n-1})$$
- *But this didn't solve the data sparsity problem:* $P(w_n|w_1 w_2 \dots w_{n-1})$

n-grams

- n-gram method:
  – We approximate the probability of observing a word *w* in the context of the previous words by the probability of this word occurring given a limited-length context of previous words. ("Markov-assumption")
  – E.g.: A bigram is the probability of a word given the previous word $P(w_n|w_{n-1})$.
  – Usually, we use trigrams, 4-grams or 5-grams.
  – What do you think are the (dis)advantages of bigrams vs. 5-grams?

- Example for bigram approximation:
  – $P(w_n|w_1 w_2 ... w_{n-1}) \approx P(w_n|w_{n-1})$

    $P(w_1 w_2 ... w_n) \approx P(w_1)*P(w_2|w_1)*P(w_3|w_2)* ... P(w_n|w_{n-1})$

How to calculate n-grams from texts

Example for bigram approximation:

- $P(w_n|w_1 w_2 \ldots w_{n-1}) \approx P(w_n|w_{n-1})$
  $P(w_1 w_2 \ldots w_n) \approx P(w_1) * P(w_2|w_1) * P(w_3|w_2) * \ldots P(w_n|w_{n-1})$

We simply calculate the probability $P(w_3|w_2)$ *as* $P(w_2 w_3)/P(w_2)$
And estimate probabilities from observed numbers of occurrences in texts.

$P(w_2 w_3) = freq(w_2 w_3)/\#bigrams\ in\ text$
$P(w_2) = freq(w_2)/\#words\ in\ text$

Hence $P(w_3 \mid w_2) = freq(w_2 w_3)/freq(w_2)$

Try it for yourself

$$P(w_3 \mid w_2) = freq(w_2 w_3)/freq(w_2)$$

Example text:

A tall girl lived in a small house next to a tall tree. One day, the tall girl wanted to climb onto the tall tree.

Please calculate the bigram probability $P(girl|tall)$

The Era of Deep Learning in CL

Since 2015, Deep Learning (aka neural networks) has become the dominant paradigm in CL.

LM model	Model class	PTB test perplexity
old-school	5-grams with Kneser-Ney	125.7
Mikolov et al. 2011	neural (RNN)	101.0
Gong et al. 2018	neural (complex)	46.5

The Era of Deep Learning in CL

We will now take a look at how RNNs (and an improved version, called LSTMs) work.

LM model	Model class	PTB test perplexity
old-school	5-grams with Kneser-Ney	125.7
Mikolov et al. 2011	neural (RNN)	101.0
Gong et al. 2018	neural (complex)	46.5

Disadvantages of ngram models

Key observation regarding problems with n-gram models:

- You have to decide on a fixed length context (bigram, trigram, 5-gram)
- If a short context is chosen, many long distance dependencies are missed.
- If a long context is chosen, we have data sparsity issues (cannot estimate probabilities accurately because we haven't observed these exact contexts frequently enough).
- Dependencies in language can be arbitrarily long:
  - Syntactic dependencies
  - Topic-related dependencies

RNNs

If we use a neural network, we also need to make sure that the context of previous words is represented in the model. It therefore makes sense to design a neural network architecture that reflects this challenge.

Solution that (in principle) allows to model arbitrarily long context:

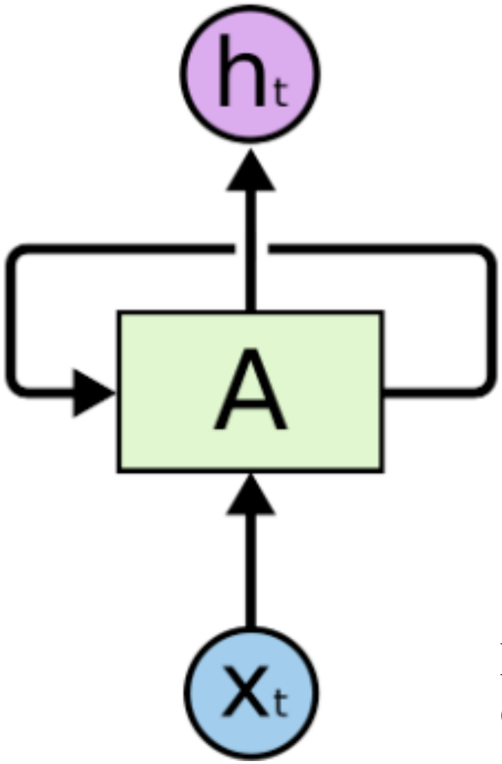
Recurrent Neural Network

x_t is the input word

h_t is the predicted next word

A is an internal hidden state

The network is "recurrent" because it contains a loop.



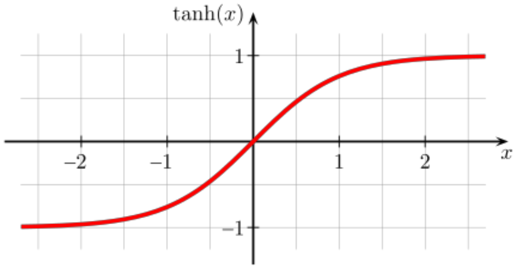
Picture credit: Christopher Olah

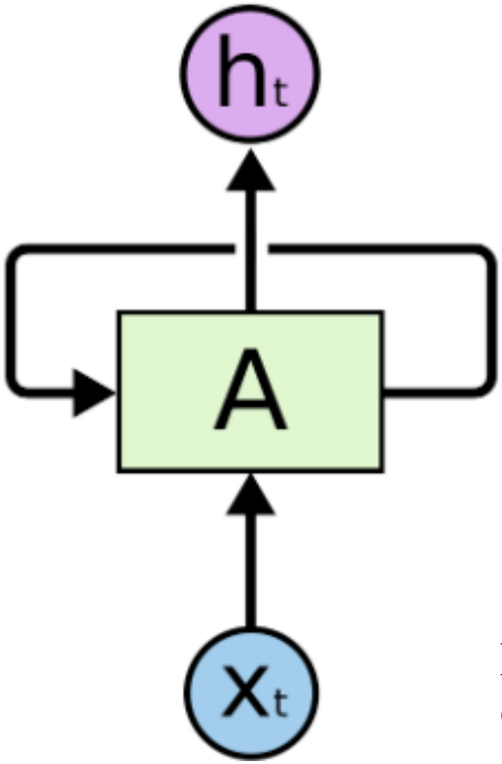
RNNs

If we use a neural network, we also need to make sure that the context of previous words is represented in the model. It therefore makes sense to design a neural network architecture that reflects this challenge.

$$A_t = \tanh(W_{AA}A_{t-1} + W_{xA}x_t)$$

$$h_t = W_{Ay}A_t$$





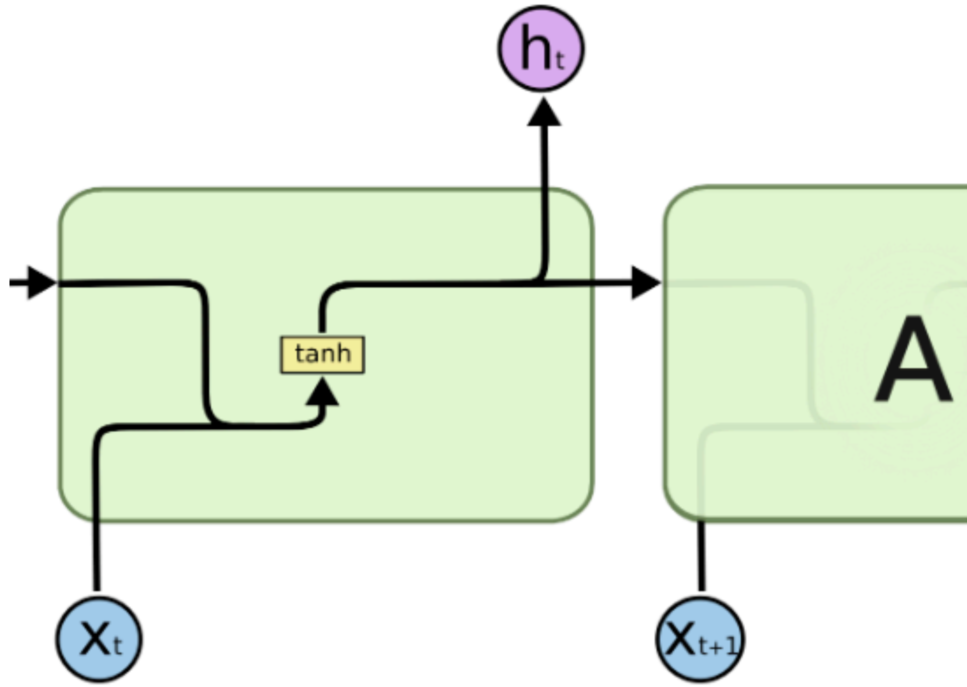
Picture credit: Christopher Olah

RNNs

If we use a neural network, we also need to make sure that the context of previous words is represented in the model. It therefore makes sense to design a neural network architecture that reflects this challenge.

$$A_t = \tanh(W_{AA}A_{t-1} + W_{xA}x_t)$$

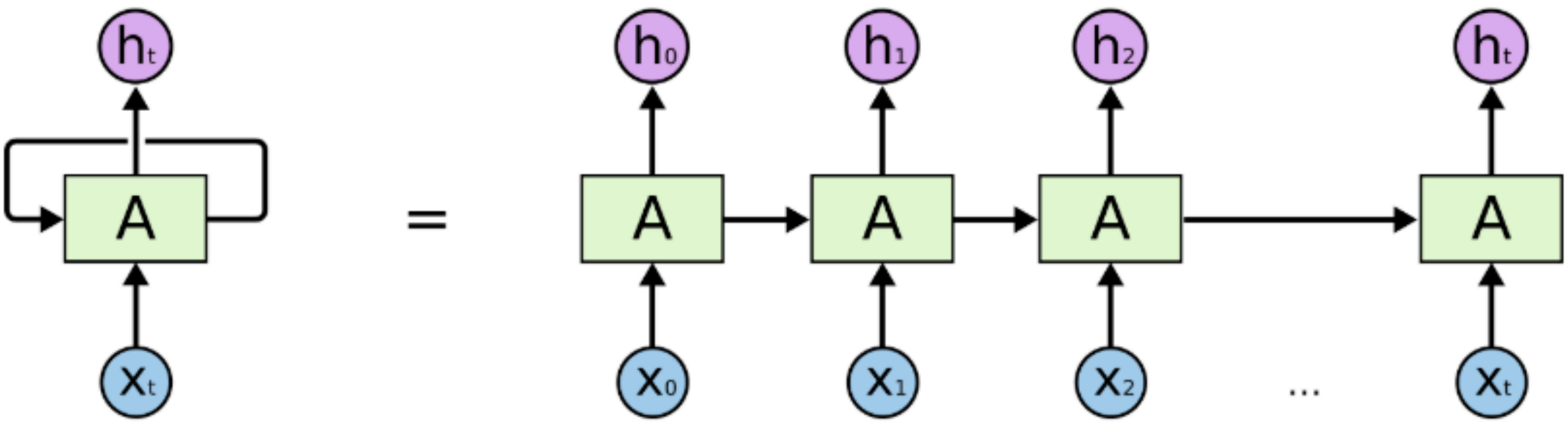
$$h_t = W_{Ay}A_t$$



Christopher Olah

RNNs

At word x_n, the network contains information about the new word and a representation of the previous words.



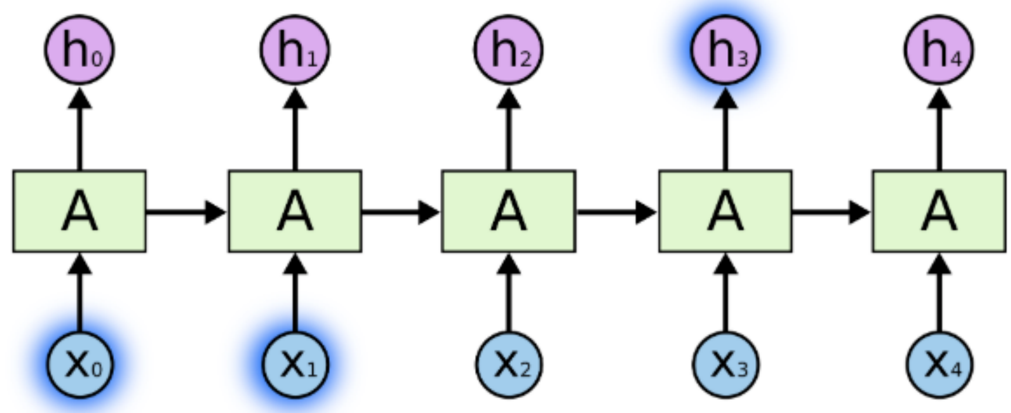
An unrolled recurrent neural network.

Picture credit: Christopher Olah

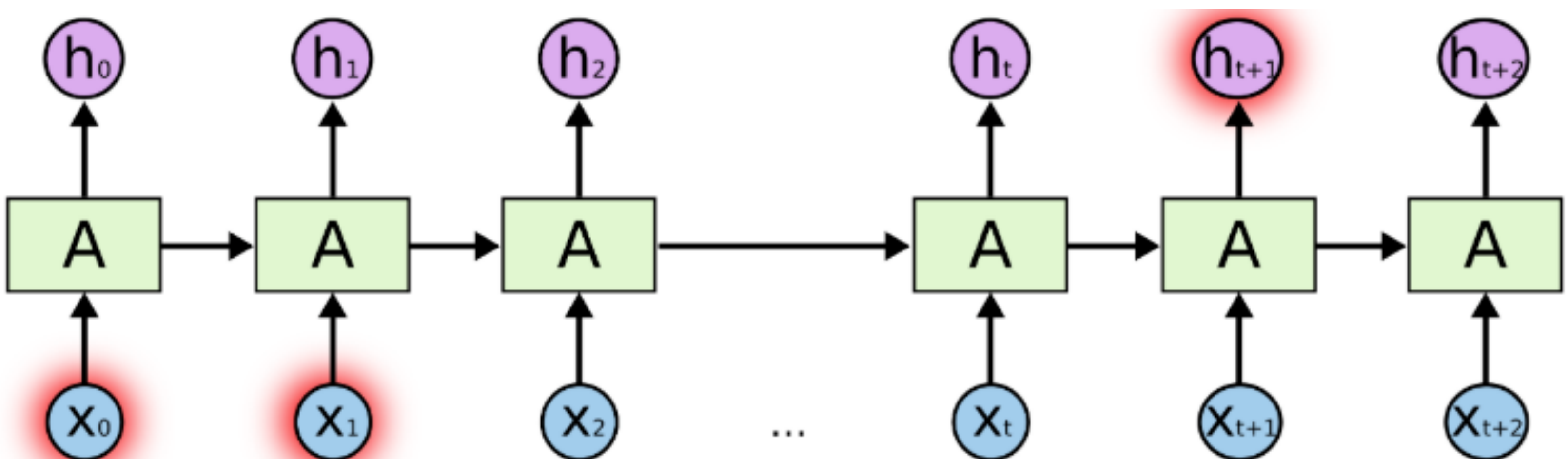
RNNs – how much context do they really capture?

Picture credit:
Christopher Olah

Short contexts are captured well.



For long gaps, we still have a problem.

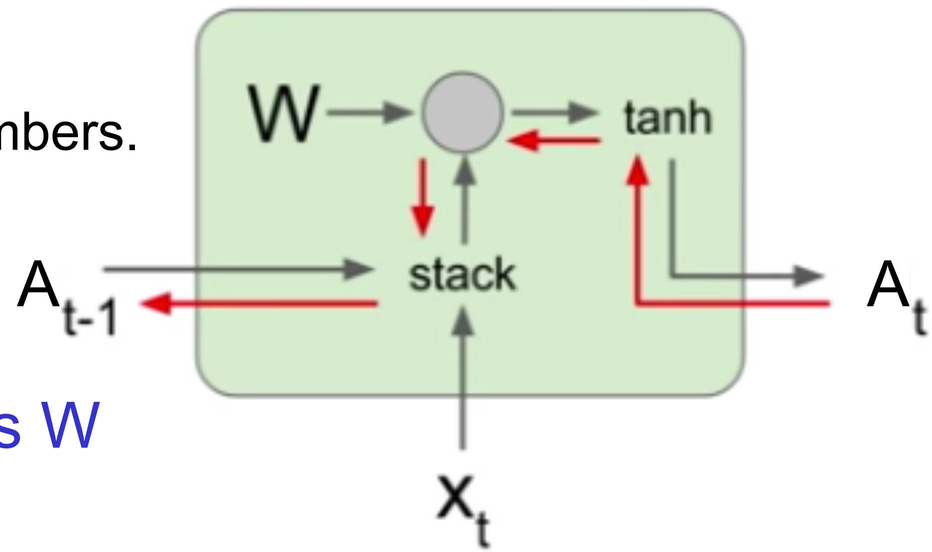


RNNs – how much context do they really capture?

Long contexts can get forgotten, because weights become too small during backpropagation (multiplying many small numbers).

Or we get „exploding gradients“ from multiplying many large numbers.

Picture credit: Justin Johnson



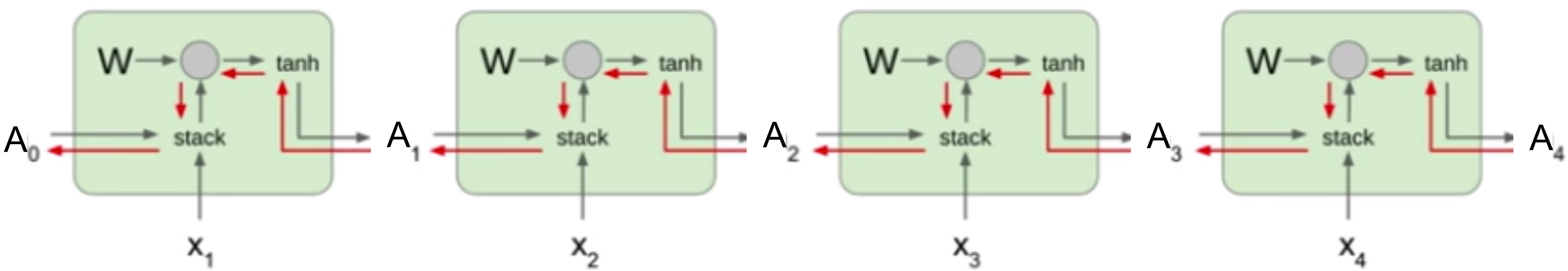
Backpropagation from A_t
To A_{t-1} multiplies by weights W (actually W^T)

RNNs – how much context do they really capture?

Long contexts can get forgotten, because weights become too small during backpropagation (multiplying many small numbers) => „vanishing gradients".

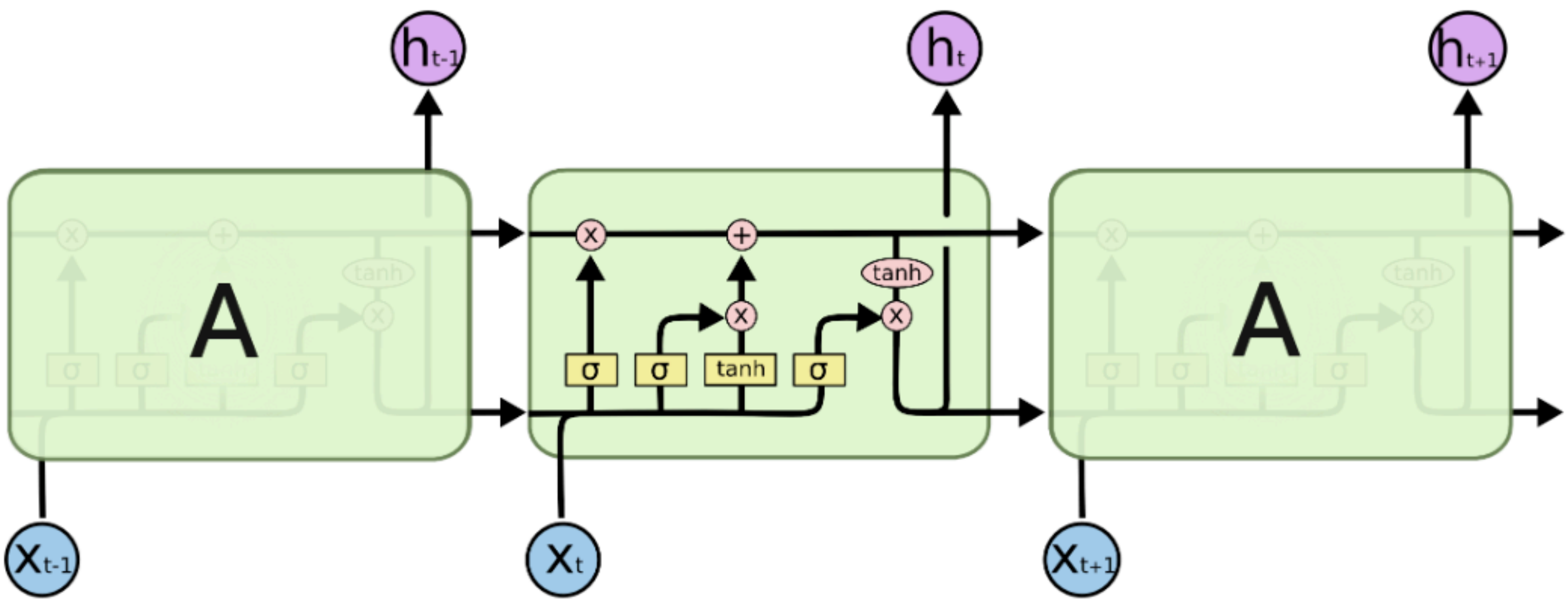
Or we get „exploding gradients" from multiplying many large numbers.

Picture credit:
Justin Johnson



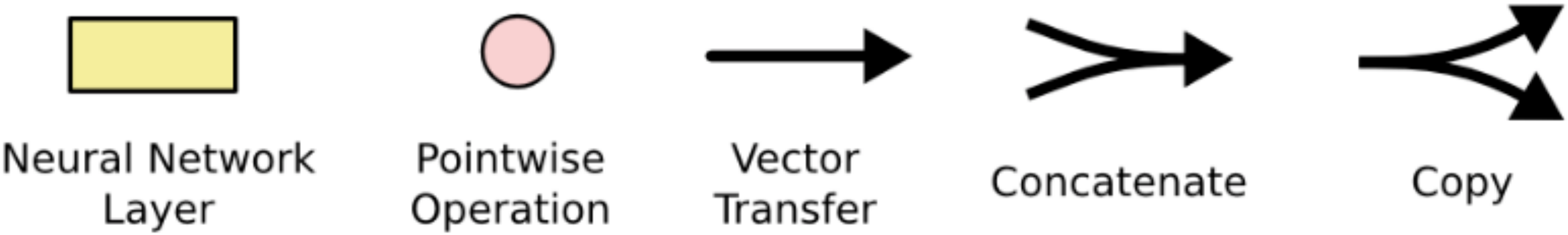
Long Short Term Memory networks (LSTM)

- Proposed by Hochreiter & Schmidhuber (1997)
- An LSTM is a more complicated form of recurrent neural network
- Widely used for language modelling
- Explicitly designed to handle long-term dependencies



The repeating module in an LSTM contains four interacting layers.

Long Short Term Memory networks (LSTM)



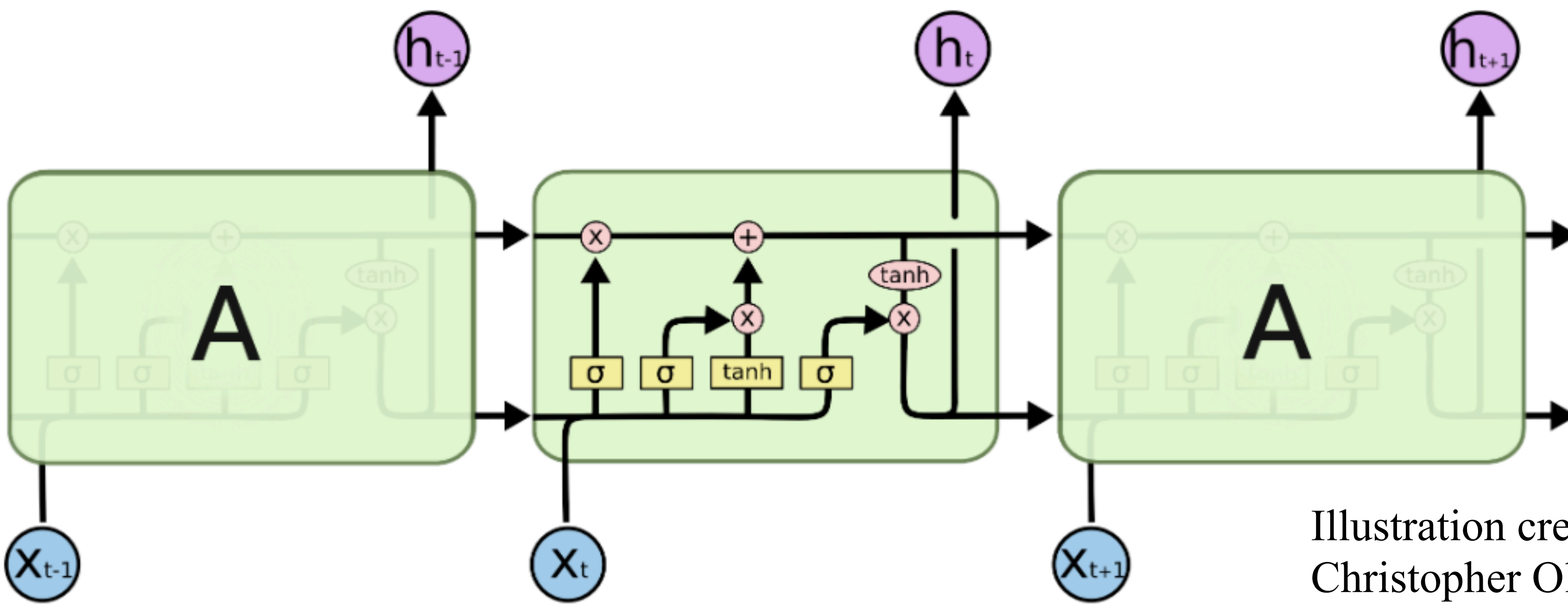


Illustration credit:
Christopher Olah

The repeating module in an LSTM contains four interacting layers.

Long Short Term Memory networks (LSTM)

Core idea:

Cell state C_t avoids the many multiplication by same weight matrix.

The LSTM can remove information from the cell state or add new information; this is regulated by the „gates".

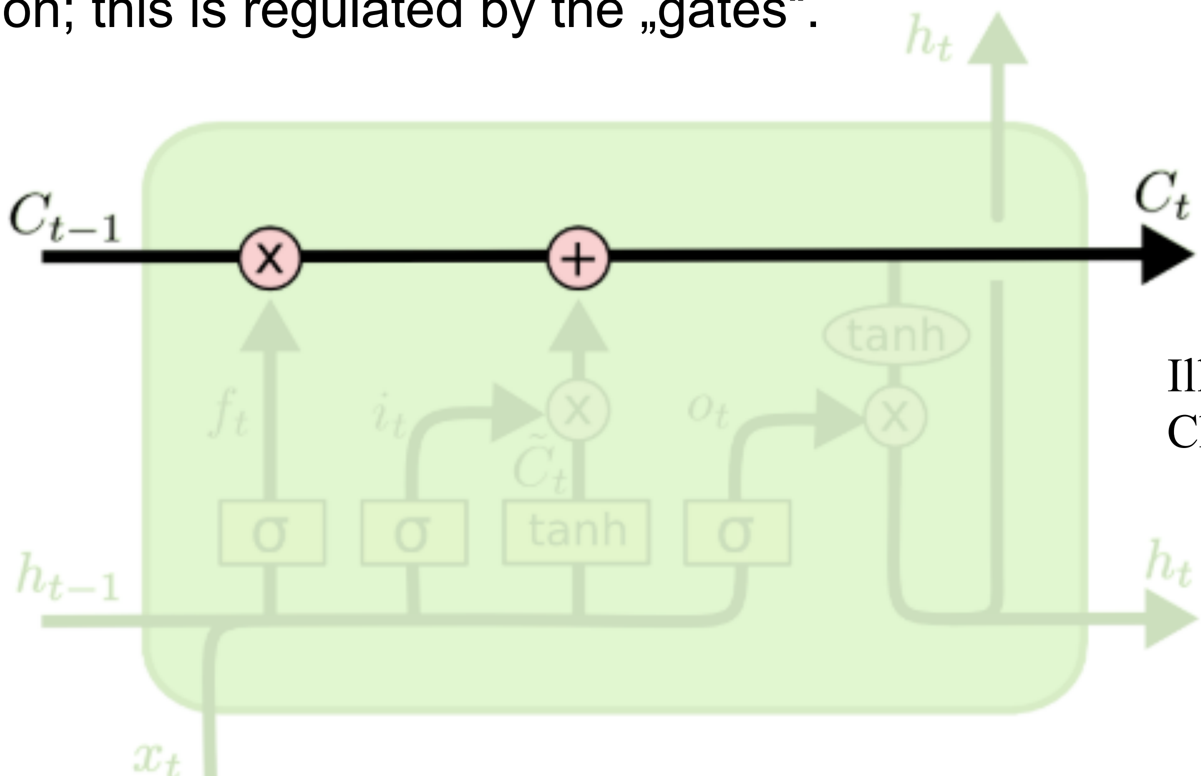
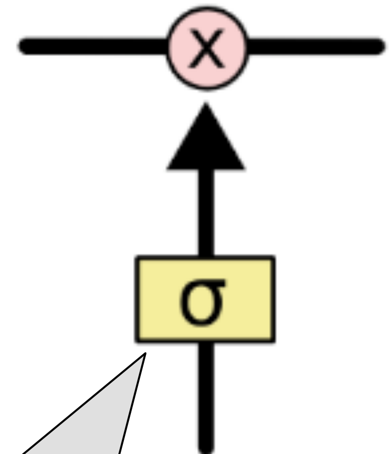
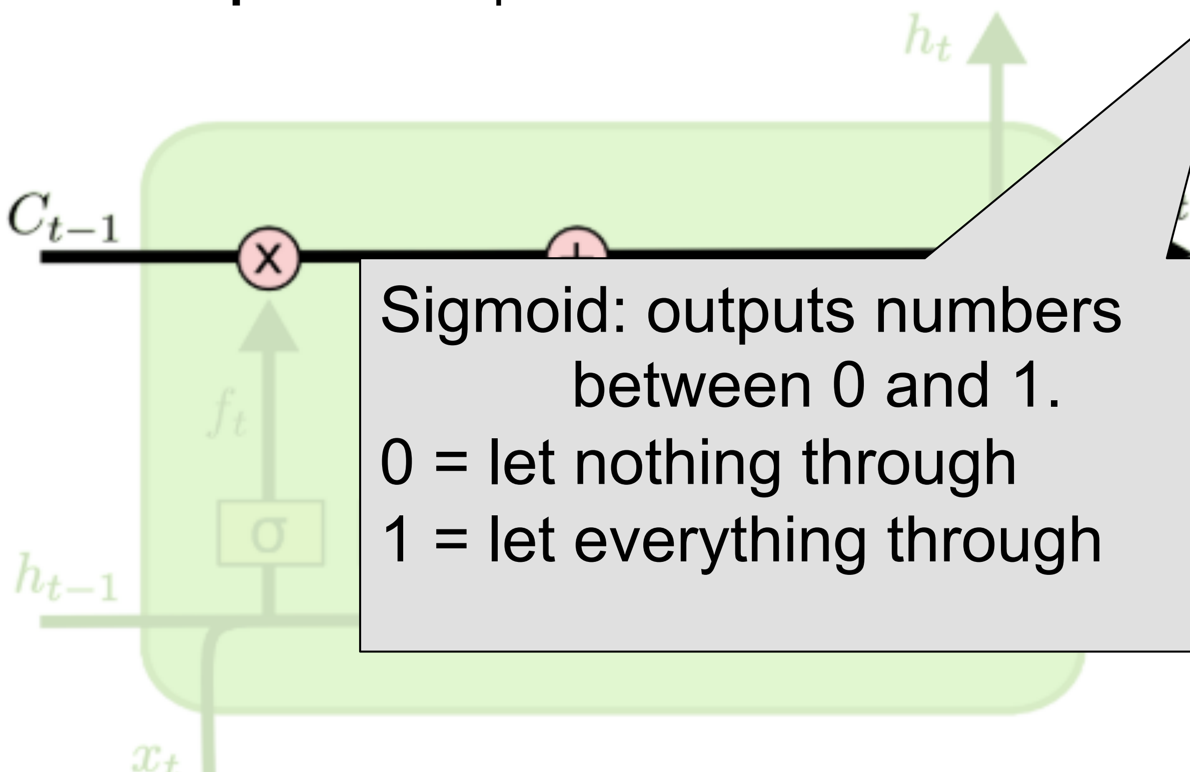


Illustration credit: Christopher Olah

Long Short Term Memory networks (LSTM)

Gates are a way to control
what information gets through.
They consist of a **sigmoid neural net layer** and
a **pointwise multiplication** operation.





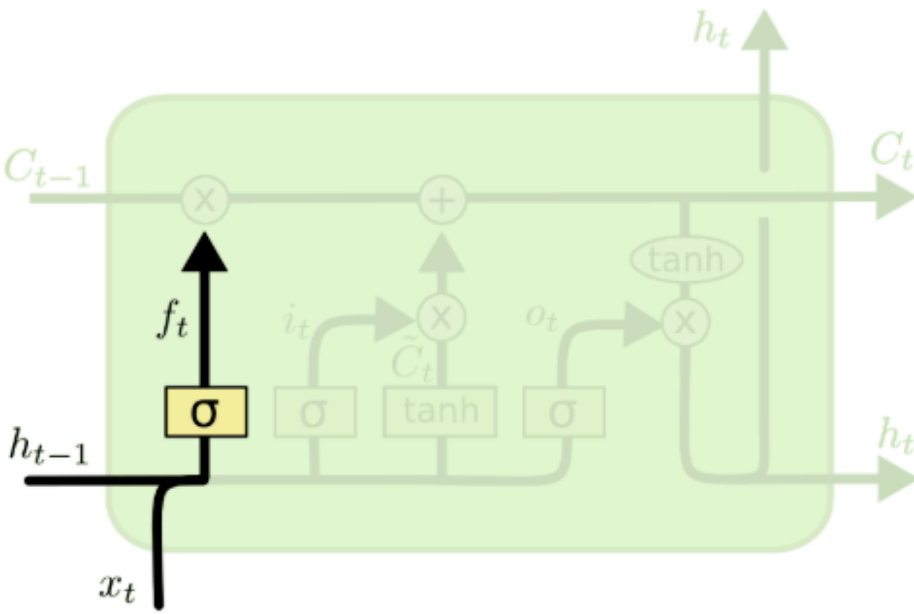
Sigmoid: outputs numbers between 0 and 1.
0 = let nothing through
1 = let everything through

Illustration credit:
Christopher Olah

LSTM “forget gate”

What information from the state h_{t-1} should we forget vs. remember?

- e.g., forget gender of previous noun if we are encountering a new noun at x_t.



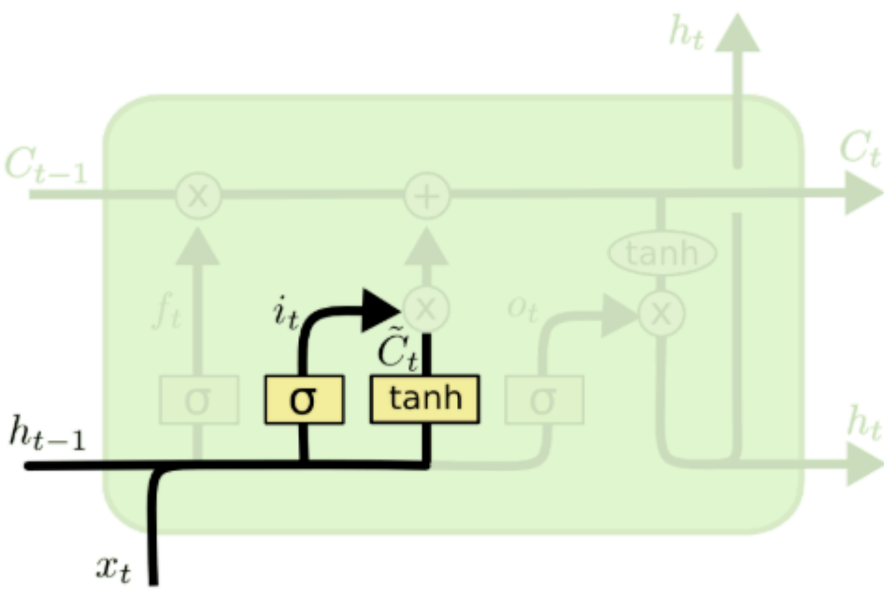
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Illustration credit: Christopher Olah

LSTM “input gate”

What information from x_t should we add to C_{t-1} to obtain cell state C_t?

- e.g., add gender of new noun if we are encountering a new noun at x_t.



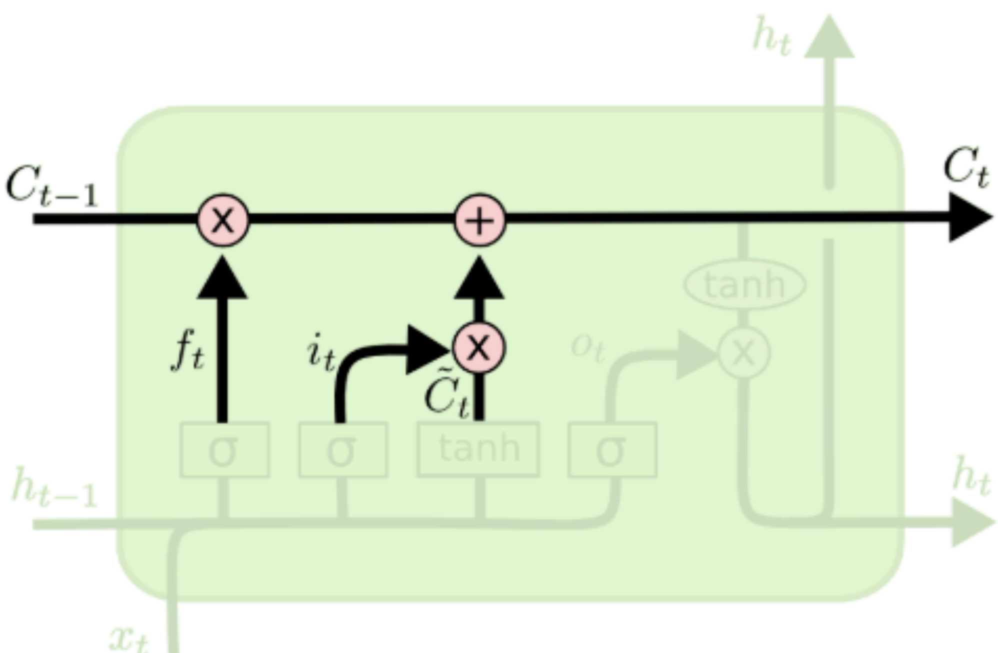
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Illustration credit: Christopher Olah

LSTM update to cell state $C_{t-1} \rightarrow C_t$

1) Multiply old state by f_t (in order to remove what we want to forget)
2) Add the new contribution from x_t to the cell state.



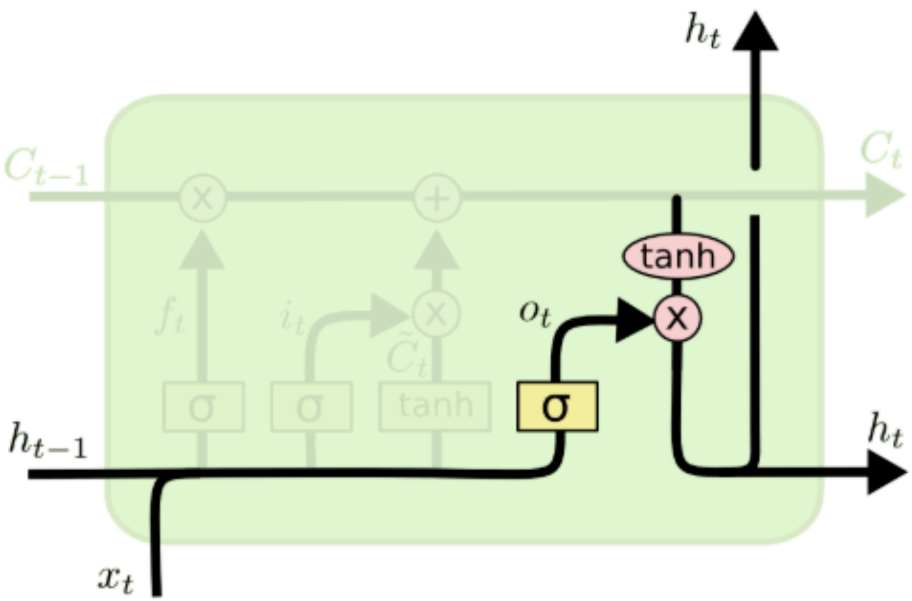
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Illustration credit:
Christopher Olah

LSTM “output gate”

What information from the new cell state should we hand on to predict the target output (and for flowing into the next cell state)?

- e.g., if we just encountered a new noun in subject role, might want to output information that‘s relevant for predicting the verb.

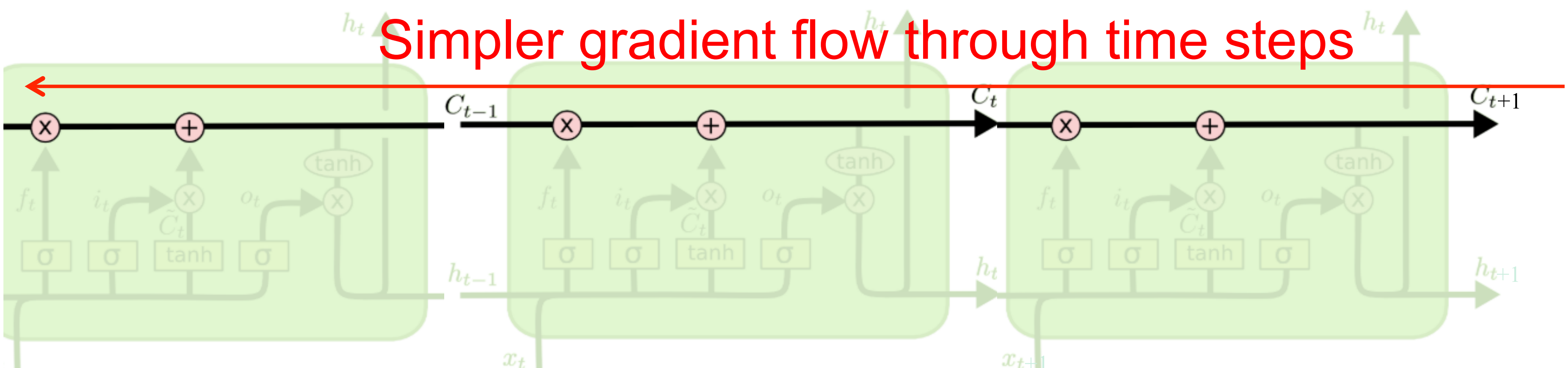


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Illustration credit: Christopher Olah

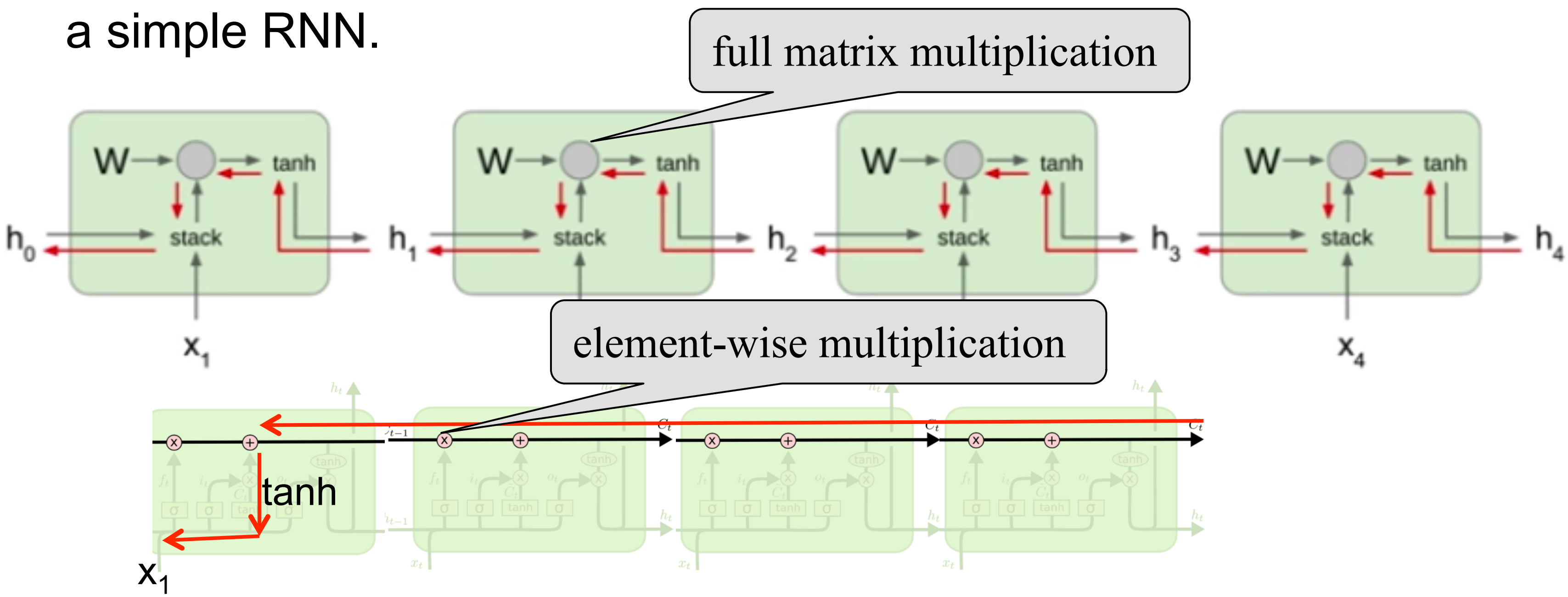
Long Short Term Memory networks (LSTM)

- During back-propagation, gradients flow through cell states with little modification: addition operation and multiply *element-wise* by forget gate
- Forget gate can vary by time stamp, therefore, less likely to have exploding or vanishing gradients.
- Doesn't have to go through *tanh* at each time step during back propagation (just once).
- Updates to weight matrices for gates are local.



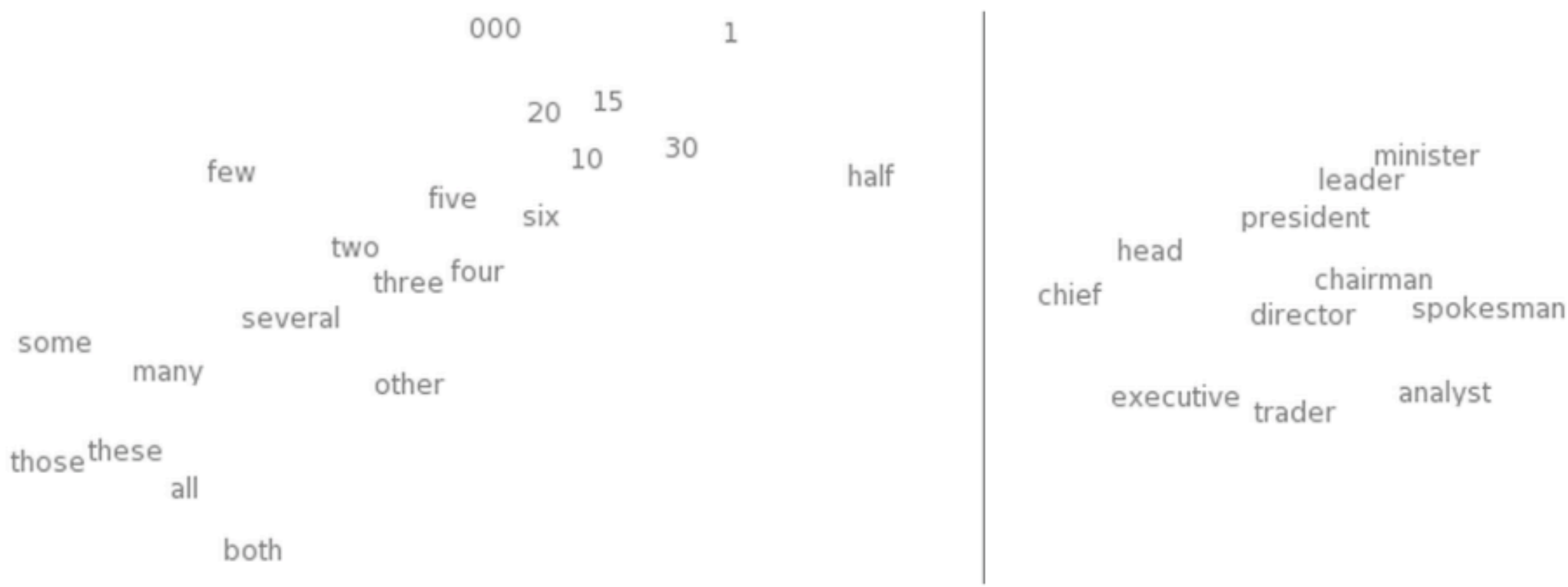
Summary simple RNN vs. LSTM

- RNNs generally allow to represent arbitrarily long contexts
- But a simple RNN has problems with vanishing and exploding gradients because it keeps multiplying with same weight matrix during back prop for each time step.
- LSTM avoids this problem by using the cell state and updating weight matrices more locally.
- LSTM has **a lot more parameters** that it needs to learn compared to a simple RNN.



Useful by-products: embeddings

- Training a simple RNN or an LSTM consists of learning the **weights** (in LSTMs, weight matrices for each of the gates)
- The learned weights can be extracted for each input word, yielding a vector of real numbers for each word, these are called **embeddings.**
- Similar words have been shown to have similar embeddings.



t-SNE visualizations of word embeddings. Left: Number Region; Right: Jobs Region. From Turian *et al.* (2010), see complete image.

Useful by-products: embeddings

- Training a simple RNN or an LSTM consists of learning the **weights** (in LSTMs, weight matrices for each of the gates)
- The learned weights can be extracted for each input word, yielding a vector of real numbers for each word, these are called **embeddings.**
- Similar words have been shown to have similar embeddings.

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

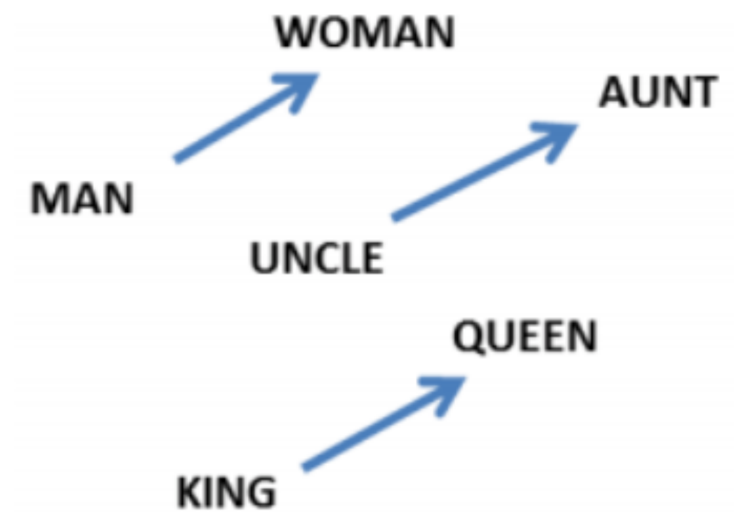
What words have embeddings closest to a given word? From Collobert *et al.* (2011)

Useful by-products: embeddings

- Training a simple RNN or an LSTM consists of learning the **weights** (in LSTMs, weight matrices for each of the gates)
- The learned weights can be extracted for each input word, yielding a vector of real numbers for each word, these are called **embeddings.**
- Similar words have been shown to have similar embeddings.

- This property can be exploited for analogy tasks:

  W(“woman”) – W(“man”) ≈
  W(“queen”) – W(“king”) ≈
  W(“aunt”) – W(“uncle”)



From Mikolov *et al.* (2013a)

Useful by-products: embeddings

Embeddings have been found to capture highly sophisticated relationships between words.

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

Relationship pairs in a word embedding. From Mikolov *et al.* (2013b).

Useful by-products: embeddings

- Embeddings have been found to capture highly sophisticated relationships between words.
- They are therefore very useful for most NLP tasks, as they capture syntactic as well as semantic information about words.
- There exist context-independent embeddings for words (each word has one embedding independent of its context)
- and context-dependent word embeddings (these work better).
- Word embeddings are often used to initialize representations for words when learning a network for a new task.
- This saves a lot of compute time, and improves performance substantially if limited training data is available for the target task.