University of Zurich[UZH]

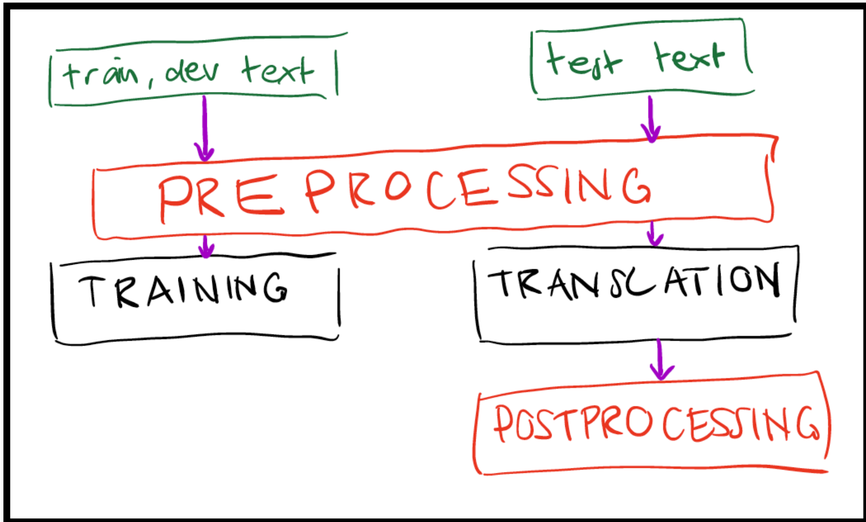
Institute of Computational Linguistics

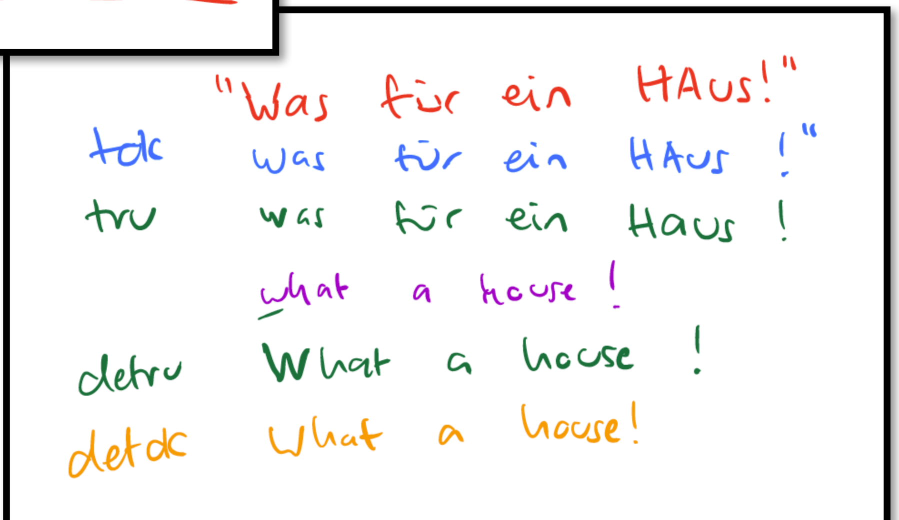
Machine Translation

4 Phrase-based Statistical Machine Translation (PBSMT)

Mathias Müller

Last time


train, dev text
test text
PREPROCESSING
TRAINING
TRANSLATION
POSTPROCESSING


"Was für ein HAus!"
tok was für ein HAus ! "
tru was für ein Haus !
what a house !
detru What a house !
detok What a house!

Topics of today

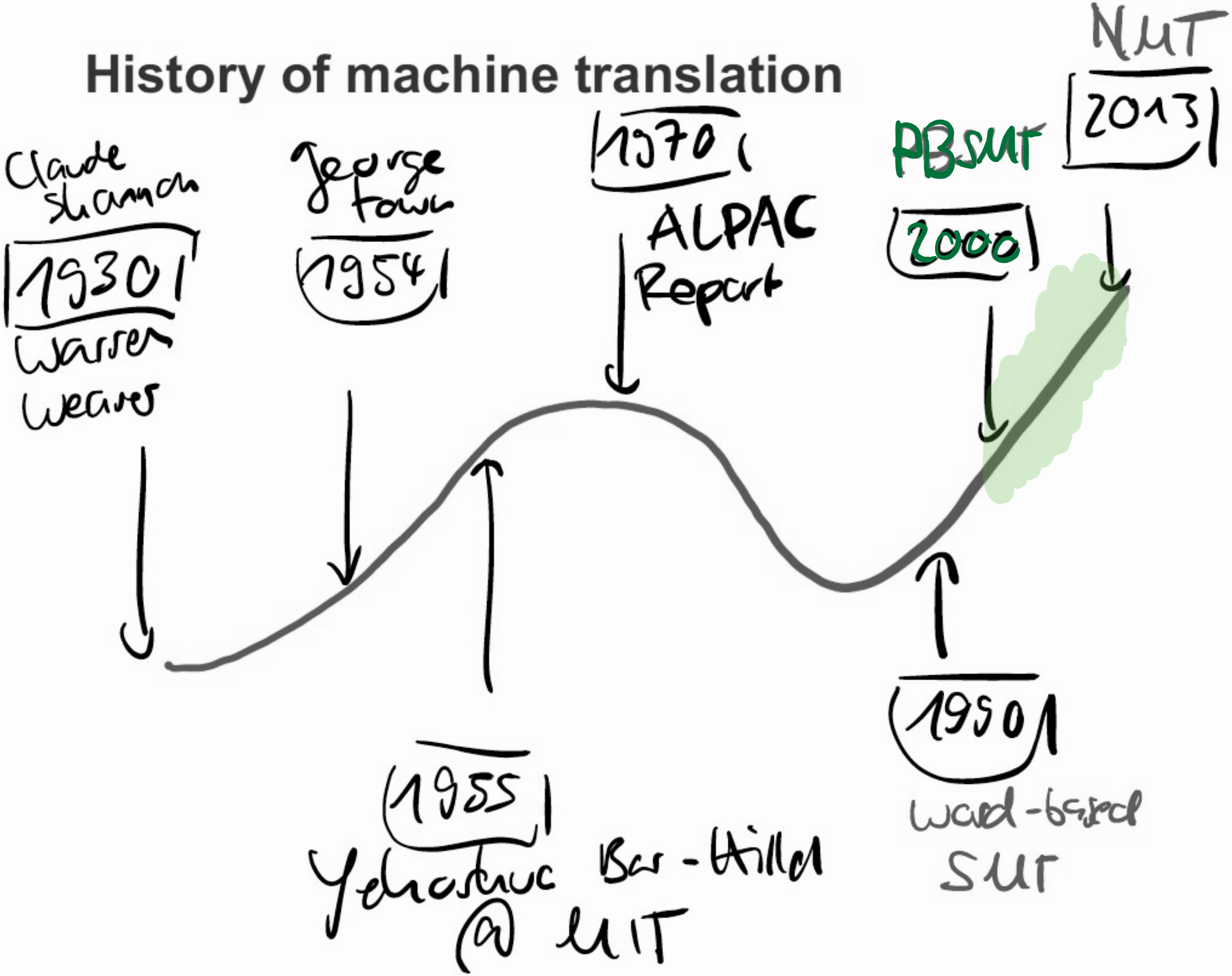
- learn how phrase-based, statistical machine translation works

  PBSMT

- main components:
  - translation model   TM
  - language model   LM

- how to combine both for translation

History of machine translation



What we know until now

```python
class TranslationSystem:

    def train(self, source_sentences, target_sentences):
        # estimate probabilities from training data

    def translate(self, source_sentence):
        # pick most probable translation
        return target_sentence

preprocessed_source_sentences = []
preprocessed_target_sentences = []

for source_sentence in open("train.de"):
    preprocessed_source_sentences.append(preprocess_sentence(source_sentence))
for target_sentence in open("train.en"):
    preprocessed_target_sentences.append(preprocess_sentence(target_sentence))

ts = TranslationSystem()
ts.train(preprocessed_source_sentences, preprocessed_target_sentences)

translation = ts.translate(preprocess_sentence("This is a test sentence"))

translation = postprocess_sentence(translation)
```

PBSMT

A more concrete train function for PBSMT

- training has two main parts:

translation model TM

language model LM

```
class TranslationSystem:

    def train(self, source_sentences, target_sentences):
        # estimate a translation model from parallel data
        self.train_translation_model(source_sentences, target_sentences)

        # estimate a language model from monolingual data
        self.train_language_model(target_sentences)
```

What a translation model looks like

phrase table

phrases = ngrams

- contains phrases in two languages, together with their translation probability

source phrases	target phrases	probabilities
natürlich	of course	0.7
natürlich	natural	0.3
natürlich ,	of course ,	0.000031
Fallout 76	crap game	0.679

What a ngram language model looks like

n = 2

- contains ngrams of a certain order, together with their probability

target ngrams	probability
natürlich ,	0.000071
er ist	0.000065
ist sehr	0.000083

How both models are used for translation

Input : "Fallout 76 is a crappy game."

- TM suggests a list of hypotheses, with scores

  Fallout 76 ist ein tolles Spiel! 0.0071
  Fallout 76 ist ein doofes Spiel. 0.0036

- LM scores each hypothesis

  Fallout 76 ist ein tolles Spiel! 0.00001
  Fallout 76 ist ein doofes Spiel. 0.0076

How both models are used for translation (2)

	TM score	LM score
Fallout 76 ist ein tolles Spiel!	0.0071	0.00001
Fallout 76 ist ein doofes Spiel.	0.0036	0.0076

- then TM and LM scores are combined with weights

Fallout 76 ist ein tolles Spiel!	$7 * 10^{-8}$
Fallout 76 ist ein doofes Spiel.	$2 * 10^{-5}$

- and that's the final ranked list of hypotheses (**nbest list**)

Translation Model in PBSMT

natürlich	of course	0.7
natürlich	natural	0.3
natürlich ,	of course ,	0.000031
Fallout 76	crap game	0.679

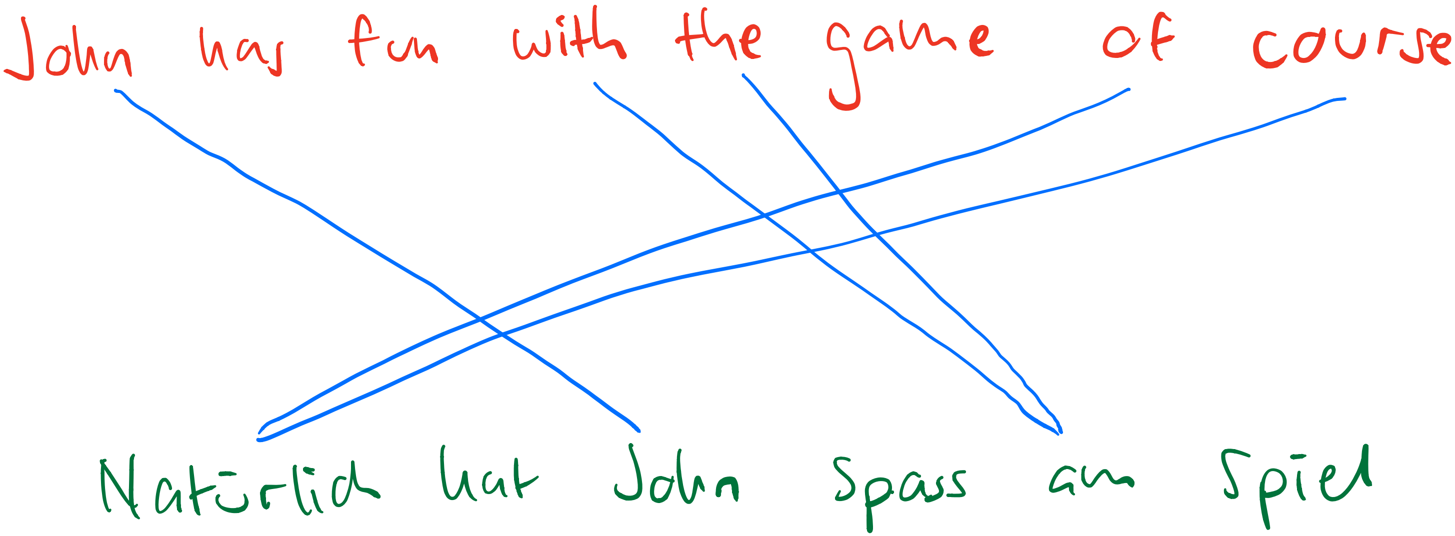
- remember: it's a parallel list of phrases, each with a probability
- Steps to create a TM:
  1) - word alignment for sentence pairs
  2) - phrase extraction from word-aligned data

TM steps in pseudo code

```
class TranslationSystem:

    def train(self, source_sentences, target_sentences):
        # estimate a translation model from parallel data
        self.train_translation_model(source_sentences, target_sentences)

        # estimate a language model from monolingual data
        self.train_language_model(target_sentences)

    def train_translation_model(self, source_sentences, target_sentences):
        self.train_word_alignment_model(source_sentences, target_sentences)

        word_alignments = self.apply_word_alignment_model(source_sentences,
                                                          target_sentences)

        translation_model = self.extract_phrases(source_sentences,
                                                 target_sentences,
                                                 word_alignments)
```

Word alignments

the	am	0.2
the	der	0.7
the	dies	0.1



Word alignment model

- basically: a word-based translation model

John	hat	0.001
John	John	0.99
John	am	0.003
John	natürlich	0.0008

- a veritable chicken and egg problem

Training a word alignment model

- learned from parallel sentences
- using an iterative algorithm like Expectation Maximization (EM), roughly:
  - initialize model, all translations equally probable
  - count occurrences of words
  - update model with counts

EM for a word alignment model

Corpus :

John is	John ist
John has	John hat
John has	Johannes hat

- initialize model, everything equiprobable

John	ist	1/3
John	hat	1/3
John	John	1/3

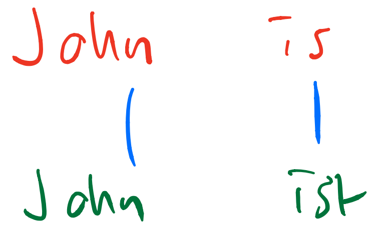
- count occurrences of words

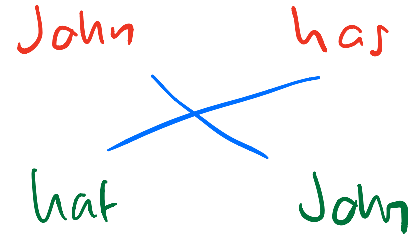
John + ist : 1
John + hat : 1
John + John : 2

- update model with counts

John	ist	1/4	0.0001
John	hat	1/4	0.0001
John	John	1/2	0.9999

Result of word alignment





Pharaoh-Format

1-1 2-2

1-2 2-1

- Tools: **fast_align** (recommended), Giza++ (not recommended)

Next up: phrase extraction

- Steps to create a TM:
  - word alignment for sentence pairs ✓
  - phrase extraction from word-aligned data

1)

John is

John ist

1-1 2-2

2)

John is

John ist

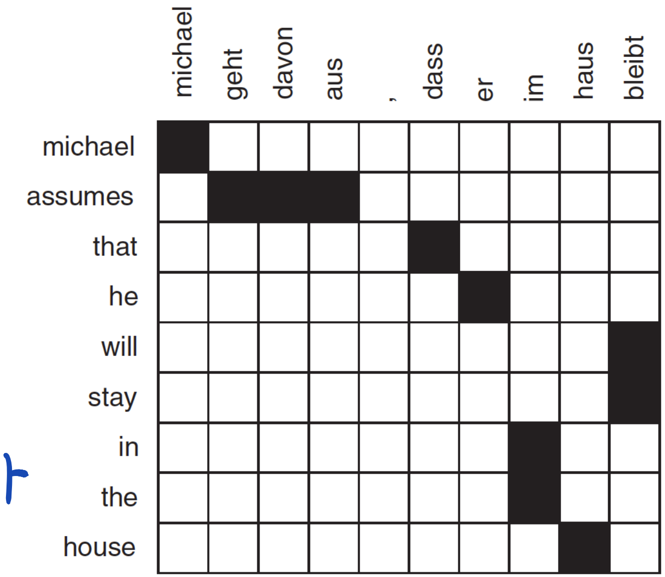
0.007

Phrase extraction from word-aligned sentences

michael   michael

assumes   geht davon aus

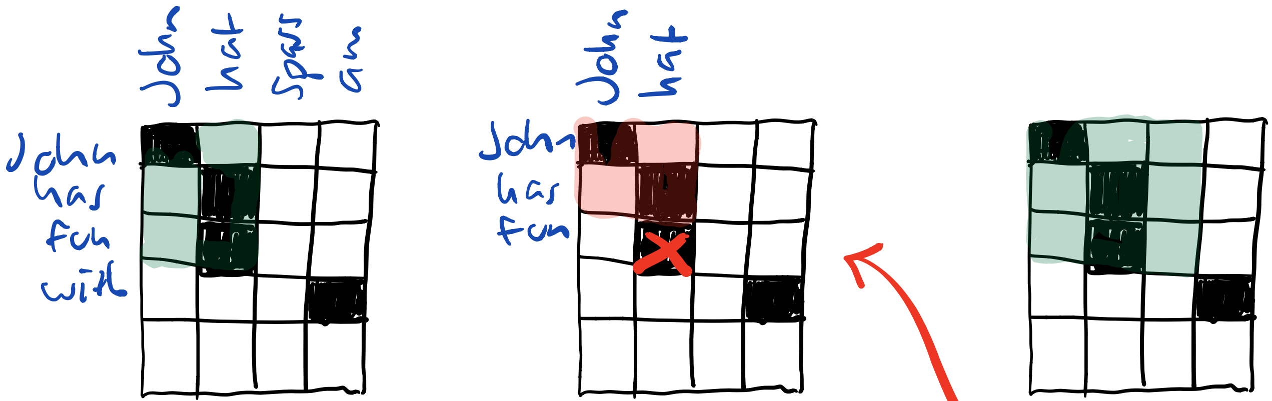
that   dass



1 phrases ≤ 7 token

2 consistent with word alignment

Rules for extracting consistent phrases



(John has fun, John hat) ✓

(John has, John hat) ✗

$(\bar{e},\bar{f})$ consistent with $A \Leftrightarrow$

$$\forall e_i \in \bar{e} : (e_i, f_j) \in A \Rightarrow f_j \in \bar{f}$$

$$\text{AND } \forall f_j \in \bar{f} : (e_i, f_j) \in A \Rightarrow e_i \in \bar{e}$$

$$\text{AND } \exists e_i \in \bar{e}, f_j \in \bar{f} : (e_i, f_j) \in A$$

Probabilities for extracted phrases

	counts	p(EN | DE)
(im Haus, in the house)	17	17/40
(im Haus, inside the house)	12	12/40
(im Haus, indoors)	11	11/40
	40	

John ist, John is

Haus house

Haus building

Haus shell

John ist | im Haus

Summary translation model

- TM is a collection of phrases that are translations of each other, together with a probability

- to create a TM,
  - train and apply a word alignment model
  - from word-aligned sentences, extract phrases
  - estimate phrase translation probabilities

Language Models

- What language models do:
  - take a text as input and output its probability

"Fallout 76 is a great game" → 0.0001

  - take a prefix of text and generate what should follow

"Fallout 76 is a" → "terrible"

Ngram Language Models

- What ngram language models look like:

natürlich ,	0.000071
er ist	0.000065
ist sehr	0.000083

Ngram Language models

- language models are trained on **monolingual data**
- in PBSMT, only for the **target language**
- ngram language models have a specific **ngram order**

Estimating an ngram LM

- for each ngram, count all occurrences in corpus, divided by total ngrams

```
def train_language_model(target_sentences, ngram_order):

    # your task right now

    return probabilities


def get_score_from_model(text, probabilities, ngram_order):

    score = 1.0

    for ngram in generate_ngrams(text, ngram_order):
        score *= probabilities[ngram]

    return score
```

Estimating an ngram LM: fun activity

Your task right now:

- implement an ngram language model
- take input data and code from OLAT:
  - `Materials/Code/4_Statistical.zip`

- everything except 1 function is already implemented

Estimating an ngram LM

```
def train_language_model(target_sentences, ngram_order):

    # your task right now:
    # implement training an ngram language model

    # probabilities must be a dictionary, with ngrams as keys,
    # and probability of those ngrams as values
    # Example:
    # >>> probabilities[("its", "culture)]
    # 5.3697041293024756e-05

    return probabilities
```

Estimating an ngram LM

Sample call:

```
$ echo "that concept of Montevideo" | python3 lm.py --text un.1k.en
TEXT:                that concept of Montevideo
PROBABILITY:         3.0965711685816526e-13
```

windows:

echo that concept...

no quotes

Estimating an ngram LM

Simple solution:

```python
def train_language_model(target_sentences, ngram_order):

    counts = defaultdict(int)

    for sentence in target_sentences:
        for ngram in generate_ngrams(sentence, ngram_order):
            counts[ngram] += 1

    total_ngrams = float(sum(counts.values()))

    probabilities = {}

    for ngram, count in counts.items():
        probabilities[ngram] = count / total_ngrams

    return probabilities
```

Translation: rank hypotheses by score

Input : "Fallout 76 is a crappy game."

- for new sentences:

"Fallout 76 ist ein tolles Spiel!"

- we now know the TM score and LM score

TM score : 0.0071 LM score : 0.00001

- and can combine them:

$$\text{score} = \text{TM score}^{\lambda_{TM}} * \text{LM score}^{\lambda_{LM}}$$

$\lambda_{TM} = 0.7$ $\lambda_{LM} = 0.3$

Weighted, linear combination of scores

$$\text{score} = \text{TM score}^{\lambda_{TM}} * \text{LM score}^{\lambda_{LM}}$$

- different notation:

$$\text{scores} \quad s = \begin{bmatrix} \text{TM score} \\ \text{LM score} \end{bmatrix} \qquad \prod_{i=1}^{2} s_i^{\lambda_i}$$

$$\text{weights} \quad \lambda = \begin{bmatrix} \text{TM weight} \\ \text{LM weight} \end{bmatrix}$$

- we can actually be more general and combine N scores, with N weights:

$$\text{score} = \prod_{i=1}^{n} s_i^{\lambda_i}$$

$x =$ "Fallout 76 ist ein tolles Spiel!"

Log-linear models

$$\prod_{i=1}^{n} s_i^{\lambda_i}$$

- in practice, we compute:

$$\text{score} = \exp\left(\sum_{i=1}^{n} \lambda_i * \log(h_i(x))\right)$$

- h are called “features”. More examples for common features are
  - reordering model
  - length penalty
  - more language models

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \begin{bmatrix} h_1 = \text{TM score} \\ h_2 = \text{LM score} \\ h_3 = \text{reordering model} \end{bmatrix}$$

Log-linear models

- h are called "features". Each h has its own weight

$$h = \begin{bmatrix} h_{TM} \\ h_{LM} \\ h_{Reordering} \end{bmatrix} \qquad weights = \begin{bmatrix} \lambda_{TM} \\ \lambda_{LM} \\ \lambda_{Reordering} \end{bmatrix}$$

- finding the optimal weights is an optimization problem called "tuning". Tools that do tuning in SMT: MERT, MIRA

Summary

- main components of a phrase-based, statistical MT system are:
  - **translation model:** table of phrases that are translations of each other, with probabilities
  - **language model:** table with probabilities for ngrams of a given order
- translation:
  - build a set of candidate translations from the translation model
  - rank the list with a score that is a log-linear combination of arbitrary features and their weight

Tools / Further Reading

- *Statistical Machine Translation* - book by Phillip Koehn
- the only relevant SMT framework: Moses, http://www.statmt.org/moses/
- modern word alignment tool: fast_align, https://github.com/clab/fast_align
- a Python tool written by Samuel Läubli and me, for convenience: mtrain, https://github.com/ZurichNLP/mtrain

Statistical poetry!

Moses

Wie Moses sich ganz leis und schnell,
von reinem Text ernährt,
am besten viel und parallel,
wird hier im Gedicht erklärt.

Nimm den Text und gib ihn schlicht,
in einen Satz-Aligner,
der sagt was sich entspricht,
und schon ist die Struktur viel feiner.

fast_align

Jetzt ist klar, was Sätze sind,
doch Wörter sind noch ganz verloren,
aber nur bis ~~Giza~~ ganz geschwind,
hat Alignment-Punkte auserkoren.

IBM Model 1, 2, 3
draus Phrasen extrahiert,
ist keine Hexerei,
mit grow-diag-final navigiert.

So kriegt man auf die schnelle,
eine schöne Phrasentabelle!

Ein Sprachmodell dazu, trainiert,
auf Zielsprachtext, ne ganze Menge,
das bewertet Sätze ungeniert,
treibt die Übersetzung in die Enge.

A language model, trained,
To target language, a whole lot,
Which evaluates sentences
uninhibited,
Drives the translation into the
narrowness.

Neue Sätze schliesslich gibt man,
dem Decoder, der aus Kandidaten,
den besten finden kann,
mit log-linearem Raten.

Automatisch evaluieren immer,
mit BLEU und METEOR und TER,
nicht schwieriger oder schlimmer,
als Kochen mit Jamie Oliver.

Das ist dir zu banal?
Dann werd neuronal.

Next Time

- beginning our journey to NMT!

Termin	Thema
19.02.	Einführung; regelbasierte vs. datengetriebene Modelle
26.02.	Evaluation
05.03.	Trainingsdaten, Vor- und Nachverarbeitung
12.03.	N-Gramm-Sprachmodelle, statistische Maschinelle Übersetzung
19.03.	Grundlagen Lineare Algebra und Analysis, Numpy
26.03.	Lineare Modelle: lineare Regression, logistische Regression
02.04.	Neuronale Netzwerke: MLPs, Backpropagation, Gradient Descent
09.04.	Word Embeddings, Recurrent neural networks
16.04.	Tensorflow und Google Cloud Platform
30.04.	Encoder-Decoder-Modell
07.05.	Decoding-Strategien
14.05.	Attention-Mechanismus, bidirektionales Encoding, Byte Pair Encoding
21.05.	Maschinelle Übersetzung in der Praxis (Anwendungen)
28.05.	Zusammenfassung, Q&A Prüfung

EVALUATION
TRAINING DATA
SMT

NMT

this is kinda important