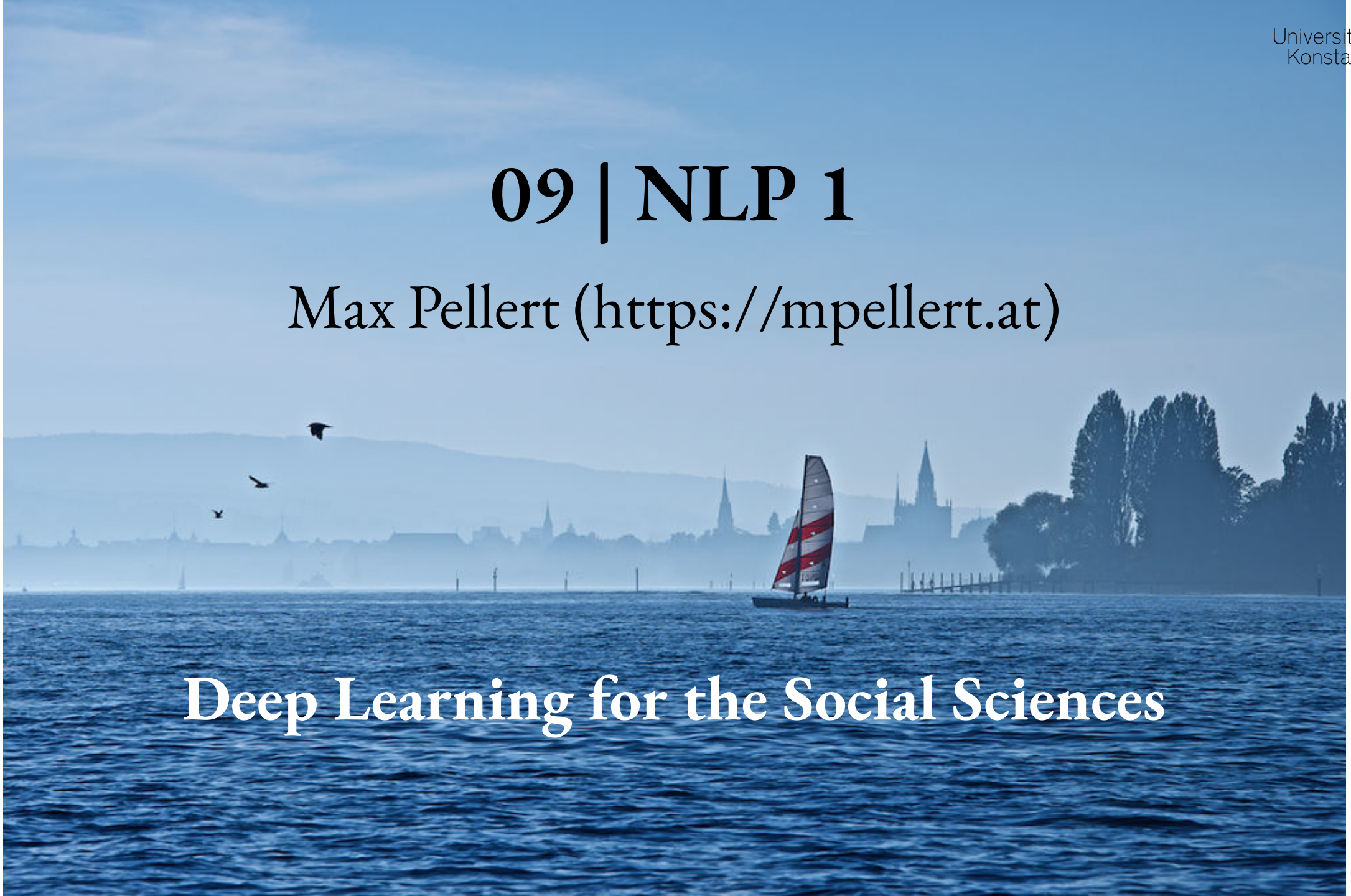


Universität Konstanz

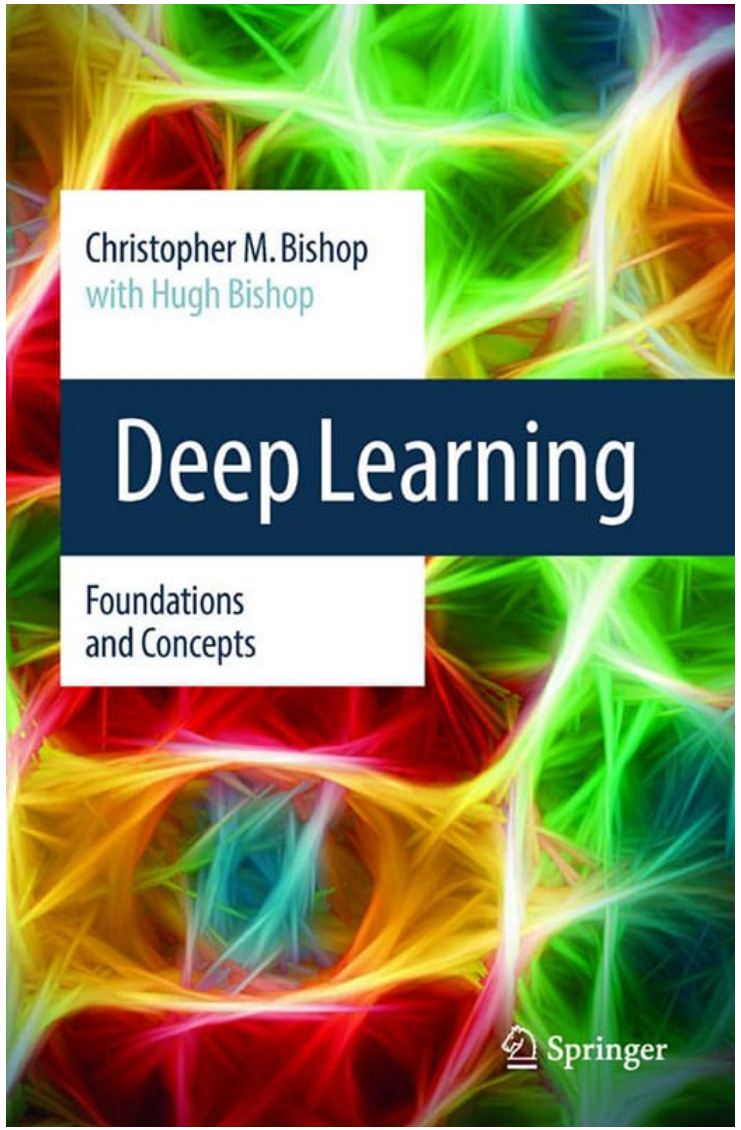
09 | NLP 1

Max Pellert (https://mpellert.at)

Deep Learning for the Social Sciences

Date	Topic	Who?
28.5.	No class	
4.6.	Generative Deep Learning 1	Giordano
11.6.	NLP 1	Max
18.6.	NLP 2	Max
25.6.	Reinforcement Learning	Giordano
2.7.	Project presentation session	Max
9.7.	Large Language Models	Max
16.7.	Generative Deep Learning 2	Giordano

Additional course book



Complementary in many aspects to “Understanding Deep Learning”

For the next lectures: check out Chapter 12 on transformer architectures

Online copy here:
https://www.bishopbook.com/

PDF Ebook provided by the university library

What is NLP?

“Natural language processing (NLP) concerns itself with the interaction between natural human languages and computing devices. NLP is a major aspect of computational linguistics, and also falls within the realms of computer science and artificial intelligence.”
https://www.kdnuggets.com/2017/02/natural-language-processing-key-terms-explained.html

“NLP is a field of linguistics and machine learning focused on understanding everything related to human language. The aim of NLP tasks is not only to understand single words individually, but to be able to understand the context of those words.”
https://huggingface.co/course/

Many many more definitions of NLP are available, the specific wording often shows from which sub-areas they originate

The following is a list of some of the common NLP tasks, with some examples of each:

- Classifying whole sentences: Getting the sentiment of a review, detecting if an email is spam, determining if a sentence is grammatically correct or whether two sentences are logically related or not
- Classifying each word in a sentence: Identifying the grammatical components of a sentence (noun, verb, adjective), or the named entities (person, location, organization)
- Generating text content: Completing a prompt with auto-generated text, filling in the blanks in a text with masked words

- Extracting an answer from a text: Given a question and a context, extracting the answer to the question based on the information provided in the context
- Generating a new sentence from an input text: Translating a text into another language, summarizing a text

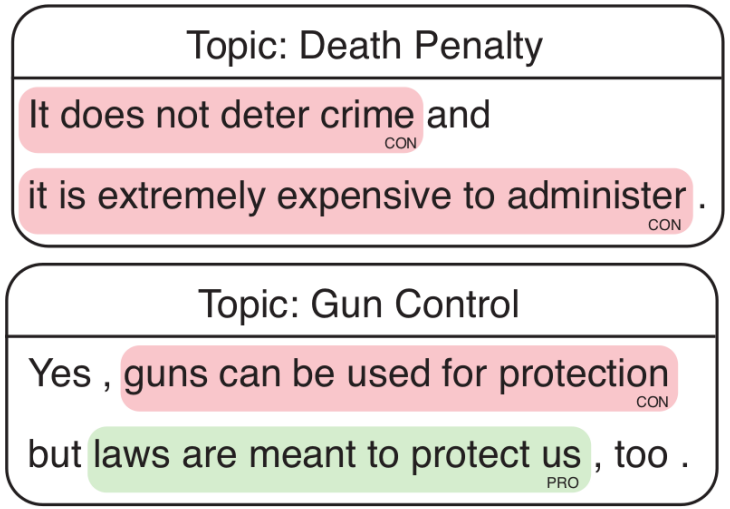
NLP isn't strictly limited to written text though. It also tackles complex challenges in speech recognition and computer vision, such as generating a transcript of an audio sample or a description of an image.

→ For many tasks that involve more than simple word counting and matching you need complex NLP models that are usually based on deep learning

Fine-Grained Argument Unit Recognition and Classification

Universität Konstanz

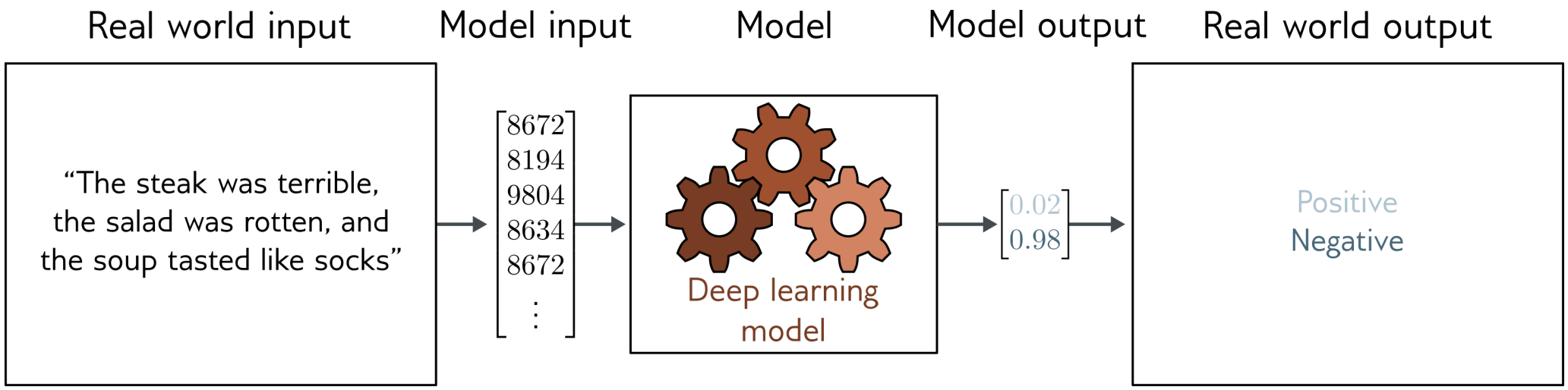
Dietrich Trautmann,[†] Johannes Daxenberger,[‡] Christian Stab,[‡] Hinrich Schütze,[†] Iryna Gurevych[‡]
[†]Center for Information and Language Processing (CIS), LMU Munich, Germany
[‡]Ubiquitous Knowledge Processing Lab (UKP-TUDA), TU Darmstadt, Germany
dietrich@trautmann.me; inquiries@cislmu.org
http://www.ukp.tu-darmstadt.de



Trautmann, D., Daxenberger, J., Stab, C., Schütze, H., & Gurevych, I. (2020). Fine-Grained Argument Unit Recognition and Classification. Proceedings of the AAAI Conference on Artificial Intelligence, 34(05), 9048–9056. https://doi.org/10.1609/aaai.v34i05.6438

Typical NLP pipeline

If we consider the following NLP workflow



We need some concepts to put all of this together

First, we break up the text that we want to feed into the model in its single elements, i.e. we **tokenize**

Tokens

Texts are sequences of characters and other symbols

To work with those sequences, for many tasks we have to separate them into their individual units

Those units are called **tokens**

The process to seperate the tokens in a sequence is called **tokenization**

Many different strategies, the simplest is to tokenize on whitespaces

Often, a token is understood as referring to one word, but in fact it is a more general concept: an emoji can also be a token for example

A note on pre-processing

Apart from tokenization, we usually don't do any further pre-processing steps with complex NLP models based on deep learning

In fact, with complex models filtering out certain tokens such as for example stopwords (as is often done with more simple, dictionary based approaches) can even be counterproductive because it can destroy **context**

We are going beyond simple approaches and, eventually, we are going to abandon the **bag of words** assumption (= word order matters)

A note on pre-processing

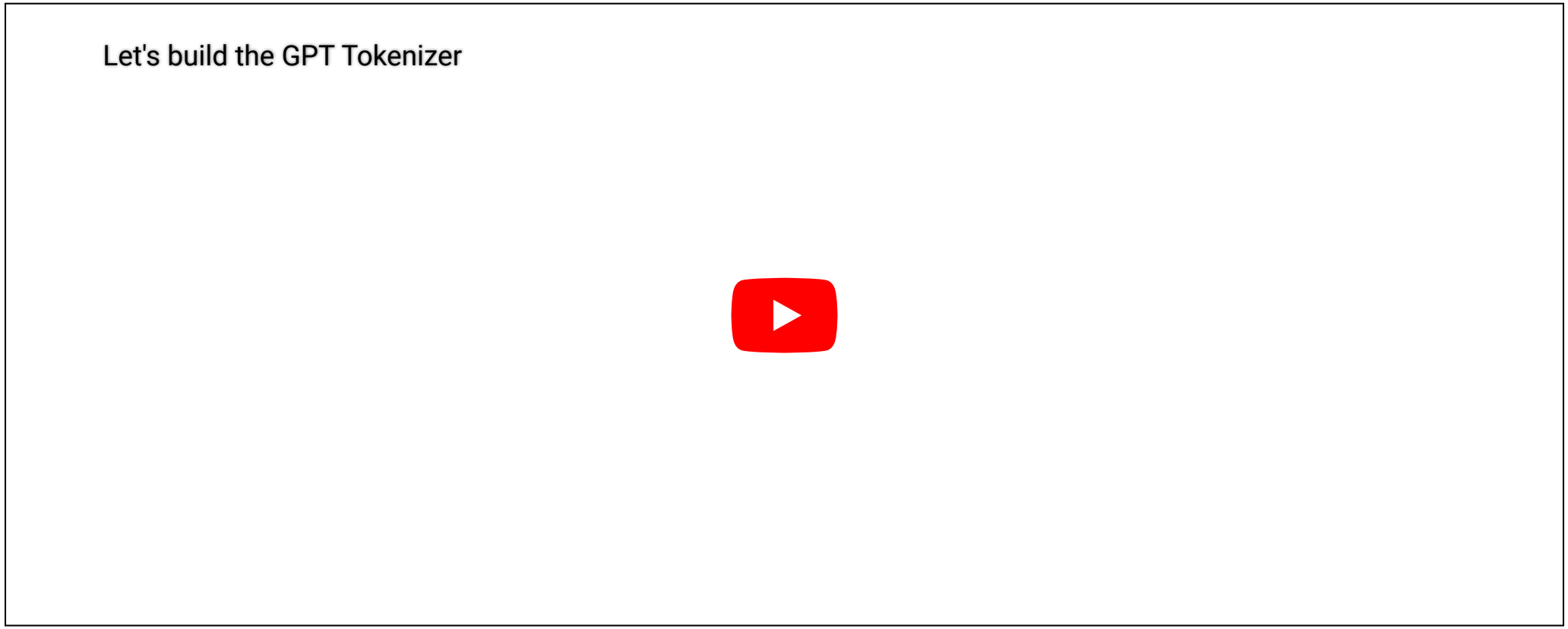
Here, we are seeing texts not as bag of words, but as collection of tokens with informative order

A token is always contextual, depending on its specific location in the collection of tokens

Our goal is to create vector representation of ordered sequences of tokens that can be used as model inputs with removing as little as possible from the original sequence

A note on tokenizing

Just tokenizing at first glance seems to be a rather neutral operation, but check out this informative video to get a feeling for the unexpected effects tokenization can have:



Model	#Tokens	Tied Emb.	#Confirmed	Examples
GPT-2 Small (0.1B)	50,257	Yes	64/999	InstoreAndOnline, _TheNitrome, quickShip
GPT-2 Medium (0.4B)	50,257	Yes	49/999	InstoreAndOnline, reportprint, _externalToEVA
GPT-2 Medium (0.4B)	50,257	Yes	49/999	InstoreAndOnline, reportprint, _externalToEVA
GPT-2 XL (1.5B)	50,257	Yes	64/999	InstoreAndOnline, _RandomRedditor, embedreportprint
GPT-J 6B	50,400	Yes	200/999	_attRot, _externalToEVA, _SolidGoldMagikarp
Phi-2 (2.7B)	50,295	Yes	103/999	DragonMagazine, _TheNitrome, _SolidGoldMagikarp
Pythia 6.7B	50,277	No	14/993	FFIRMED, _taxp, _affidav
GPT-NeoX 20B	50,277	No	10/993	FFIRMED, _taxp, _affidav
OLMo v1 7B	50,280	No*	270/993	_§\\[, ^-^, marinedrugs, FFIRMED
OLMo v1.7 7B	50,280	No*	178/993	_§\\[, _[****, medscimonit, FFIRMED
Llama2 7B	32,000	No	20/639	_Mediabestanden, _Portály, oreferrer
Llama2 70B	32,000	No	32/639	_Mediabestanden, _Portály, ederbörd
miqu 70B	32,000	No	30/639	_Mediabestanden, _Portály, _regnig
Mistral 7B v0.1	32,000	No	49/637	\uefc0, _/**\r, ↬, _febbra, iNdEx
Zephyr 7B beta	32,000	No	76/637	\uefc0, _/**\r, ↬, _febbra, iNdEx
Mixtral 8x7B	32,000	No	44/637	\uefc0, _/**\r, ↬,];\r
Solar 10.7B	32,000	No	58/638	\uefc0, _/**\r, ↬, _febbra, _gepublice
Rakuten 7B	48,000	No	66/957	\uefc0, _/**\r, ↬, _febbra, 稲田大学
Qwen1.5 32B	151,646	No	2450/2966	_ForCanBeConvertedToF, (stypy, $PostalCodesNL
Qwen1.5 72B Chat	151,646	No	2047/2968	_ForCanBeConverted, useRalative, _typingsJapgolly
StableLM2 12B	100,288	No	138/1997	_ForCanBeConverted, \tTokenNameIdentifier, _StreamLazy
Llama3 8B	128,256	No	556/2540	_ForCanBeConverted, ŷыцNŷыцN, _CLIIIK, krvldkf, 글상위
Command R (35B)	255,029	Yes	302/5012	AddLanguageSpecificText, _ARStdSong, 目前尚未由人工引
Command R+ (104B)	255,029	Yes	79/5012	AddLanguageSpecificText, tocguid, ephritidae
Gemma 2B	256,000	Yes	3308/5117	हिंदीखरीदारी, ^(◐)$_, _coachTry, _AcceptedLoading, ICTOGRAM
Gemma 7B	256,000	Yes	802/5117	हिंदीखरीदारी, EnglishChoose, _quefto, _stockfotografie, 図
Starcoder2 15B	49,152	No	128/968	ittrLoremipumdolorsitametconsecteturadipiscingelitIntegervelvel
Yi 9B	64,000	No	245/1278	\\+::\\+, mcited, mabaochang, nConsequently
Jamba v0.1 (52B)	65,536	No	6/1280	derrelsc,]{}]{}, ronicsystems

Table 1: **Detection of under-trained tokens.** #Confirmed are the confirmed/tested numbers for the tokens tested in verification that are predicted with a maximal probability of $< 1\%$ across verification prompts. Examples were manually chosen for readability, similarity across models or for being particularly striking. Note that the leading '_' in tokens such as `_SolidGoldMagikarp` indicates a leading space.
*We use an unembedding-based indicator for these models (cf. section 3.3.2).

Land, S., & Bartolo, M. (2024). Fishing for Magikarp: Automatically Detecting Under-trained Tokens in Large Language Models (arXiv:2405.05417). arXiv. http://arxiv.org/abs/2405.05417

From tokens to vector representations

Universität Konstanz

Assume that we have tokenized a sequence of text that we want to feed as input to the model, by splitting up on whitespaces and adding a few rules how to handle symbols such as ‘!’, ‘.’ or ‘,’

To get from an ordered collection of text tokens to a first vector representation, we can just replace the token by it’s index (=row number) in the list of all tokens that the model know, the **vocabulary**

We will take a look a little bit later on ways to construct the vocabulary

For now, assume we have already constructed such a vocabulary

Index	Vocabulary Entry
1	a
2	aardvark
...	...
7	an
8	ant
9	antelope
10	anvil
11	are
12	aspic
13	ate
...	...

an aardvark ate an ant→ [7,2,13,7,8]

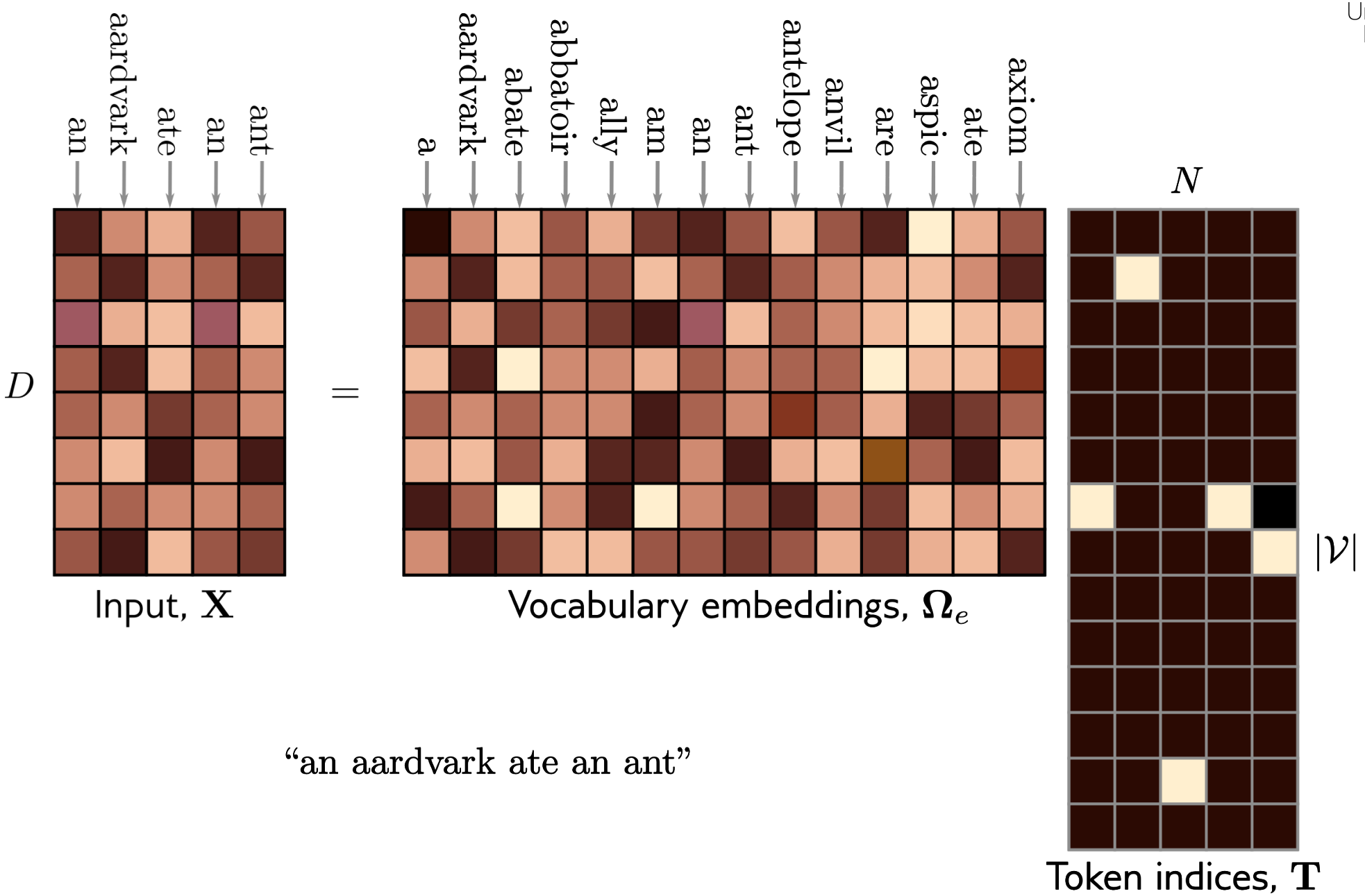
Special Tokens

Depending on the model architecture and the tokenizer used, the vocabular will also contain special tokens

Examples are:

- [EOS] ... denotes the end of a sequence
- [SEP] ... used for training with sequence pairs, denotes the end of the first sequence
- [CLS] ... used for classification task, contains information of the whole sequence
- [MASK] ... used to “hide” one token by replacing it

Let's convert to matrices that the model can work with



"an aardvark ate an ant"

How to create the vocabulary of a model?

Vocabulary

Using all unique tokens in a text that were seperated by using heuristics such as splitting on whitespace and similar rules would result in a very large vocabulary

This will affect performance because of hard- and software constraints

Often, some tokens are very rare and uninformative, for example usernames (see SolidGoldMagikarp)

This approach would lead to an extremely large vocabulary that can be used to represent token sequences with just one integer per token

Vocabulary

The other extreme would be to use only unique characters as the vocabulary

Character tokenization leads to a small vocabulary but produces long sequences to represent even individual words

For example the word “clown” alone would need to be represented by 5 integers, which also leads to performance bottlenecks

With that approach it proved very hard for a model to learn a useful representation of the letter “c” instead of the word “clown” for example

Vocabulary

For these reasons, a middle way is often used: subword tokenization to build the vocabulary.

There are different algorithms (Byte-Pair Encoding (BPE), WordPiece, Unigram, ...) that work in specific ways to achieve one goal: Frequent words should be included in the vocabulary as-is (“clown”), but not so frequent words should be combinations of words in the vocabulary (“clown” + “ish”)

Parts of the vocabulary that are added to other words are prefixed, for example “##ish”.

Let’s have a look at one of the algorithms to achieve that...

Tokenization alogorithms

Goal: Tokenizer chooses it's input "units"

To recap, some things to consider:

- Inevitably, some words (e.g., names) will not be in the vocabulary
- It's not fully clear for example how to handle punctuation
- The vocabulary should have one token for one base word and allow to add different suffixes (e.g., walk, walks, walked, walking)
- Thereby, we want to make it possible for the model to later learn that these variations are related

Solution: Sub-word tokenization

a)

a_sailor_went_to_see_see_see_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_see_see_see_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

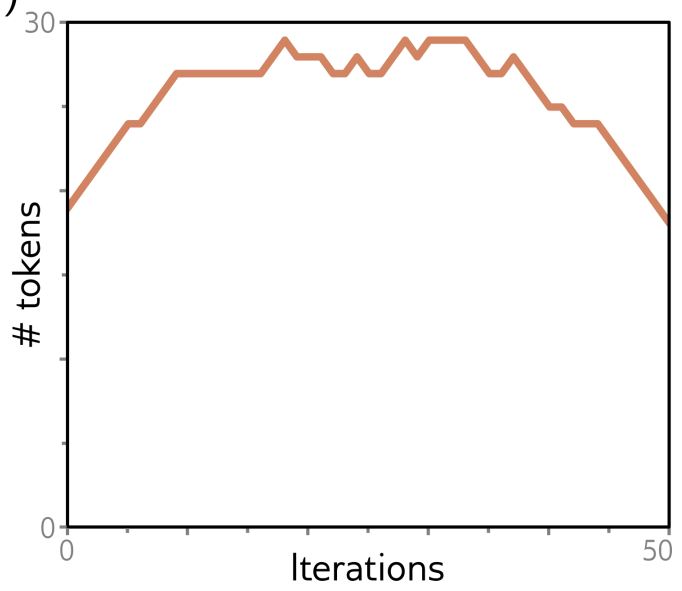
_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_see_see_see_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

f)



Peter Piper picked a peck of pickled peppers

Peter Piper picked a peck of pickled peppers

Peter Piper picked a peck of pickled peppers

Peter Piper picked a peck of pickled peppers

Peter Piper picked a peck of pickled peppers

Peter Piper picked a peck of pickled peppers

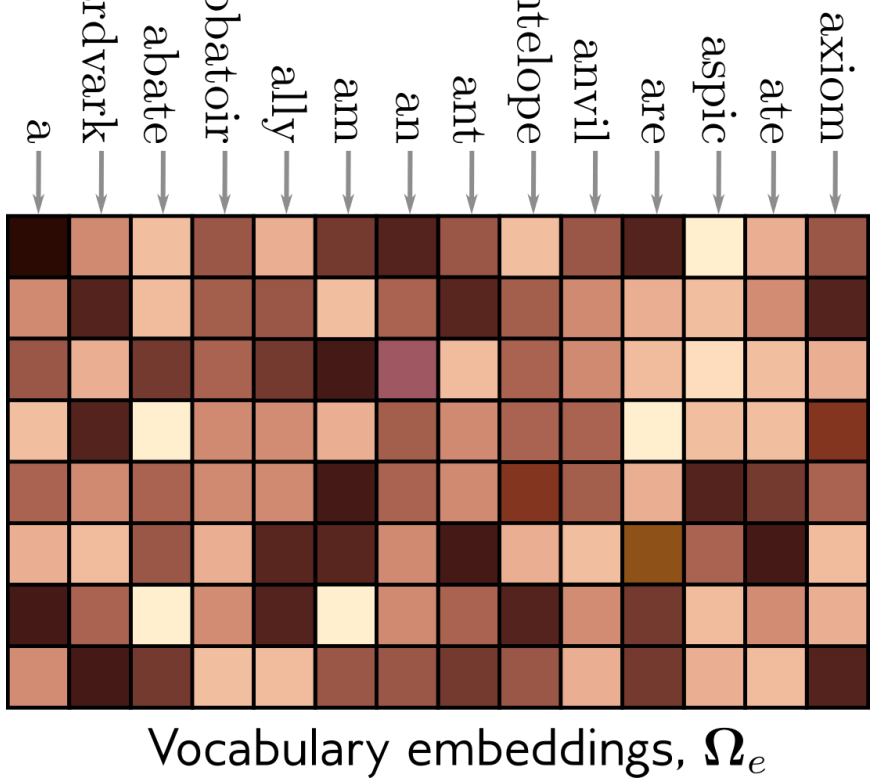
Neural Machine Translation of Rare Words with Subword Units

Rico Sennrich and **Barry Haddow** and **Alexandra Birch**
School of Informatics, University of Edinburgh
{rico.sennrich,a.birch}@ed.ac.uk, bhaddow@inf.ed.ac.uk

We set a stopping criterion at the desired size of the vocabulary (for example 50 000 or 150 000 tokens)

Provides a tradeoff between vocabulary size and size of integers necessary to represent a tokenized sequence

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural Machine Translation of Rare Words with Subword Units. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 1715–1725. https://doi.org/10.18653/v1/P16-1162



The vocabulary embeddings can actually just be randomly initialized in the beginning and be trained like other parameters

Or pretrained embeddings, e.g. from Word2Vec or GloVe can be used

Vocabulary Embeddings

Dimensionality is specified according to the model architecture, for example 1024 dimensions or more

This is one of the hyperparameters of NLP models

Another one concerns the **batch size**: how many individual sequences are passed in one iteration to the model, usually more is better here (and the limitation is often GPU memory)

Individual sequences longer than x tokens (for example 512 for BERT or also more for later models) often have to be **truncated**, shorter sequences are usually increased with 0 to maximum length (**padding**)

Typical parameters

Let's take a look at hyperparameters of a widely used model (that has this kind of information still publicly available)

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners (arXiv:2005.14165). arXiv. http://arxiv.org/abs/2005.14165

Traditional language modeling

Now we have our inputs ready, let's consider the model part

Let's take a step back and consider we have a collection of tokens in a **corpus**

The simplest approach: we could count the frequency of each token and record that in table

Let's consider this still as a bag of words (no order), and create a sequence by consulting the table:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n)$$

Traditional language modeling

Such approaches can be used, for example with Naive Bayes classifiers, where we assume tokens are independent within each class C_k but with a different distribution for each class

We can estimate the prior and the class-conditional densities from an annotated training data set:

$$p(\mathcal{C}_k|\mathbf{x}_1,\ldots,\mathbf{x}_N) \propto p(\mathcal{C}_k)\prod_{n=1}^{N} p(\mathbf{x}_n|\mathcal{C}_k)$$

While such approaches can be useful, for example to create comparison baselines, we want to use the information contained in **word order**

Autoregressive models

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$$

We could represent each term on the right-hand side by a table whose entries are as before estimated using simple frequency counts from the training corpus

However, the size of these tables grows exponentially with the length of the sequence, and so this approach would become prohibitively expensive

We need to find a cure against this computational intractability

Restricted autoregressive models

One way would be to simplify the model dramatically by assuming that each of the conditional distributions on the right-hand side is independent of all previous observations except the L most recent words

For example, if $L = 2$ then the joint distribution for a sequence of N observations under this model is given by

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1) \prod_{n=3}^{N} p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{x}_{n-2})$$

Restricted autoregressive models

With $L = 1$ we would call this a **bi-gram** model because it depends on pairs of adjacent words

$L = 2$ involves triplets of adjacent words, is called a **tri-gram** model, and in general these are called **n-gram** models

All the models discussed so far in this part can be run **generatively** to synthesize novel text

If we for example provide the first and second words in a sequence, then we can sample from the tri-gram statistics $p(x_n | x_{n-1}, x_{n-2})$ to generate the third word, and then we can use the second and third words to sample the fourth word, and so on

Issues of text generation

The resulting text, however, will be incoherent because each word is predicted only on the basis of the two previous words

High-quality text models must take account of the long-range dependencies in language

In addition, n-gram approaches don't generalize well, for example to tokens that are not included in their training corpus (here usually some smoothing by adding small probabilities to such unknown tokens is used)

Issues of text generation

At the same time, we want to avoid the exponential growth in the number of parameters of an n-gram model because of computational constraints

Building huge tables is also generally not efficient, because even semantically very close sequences are recorded separately on their own, for example sentences that are exactly the same except for a synonym

A proposed solution should also account for that redundancy in natural language by making use of shared parameters

→ Neural models

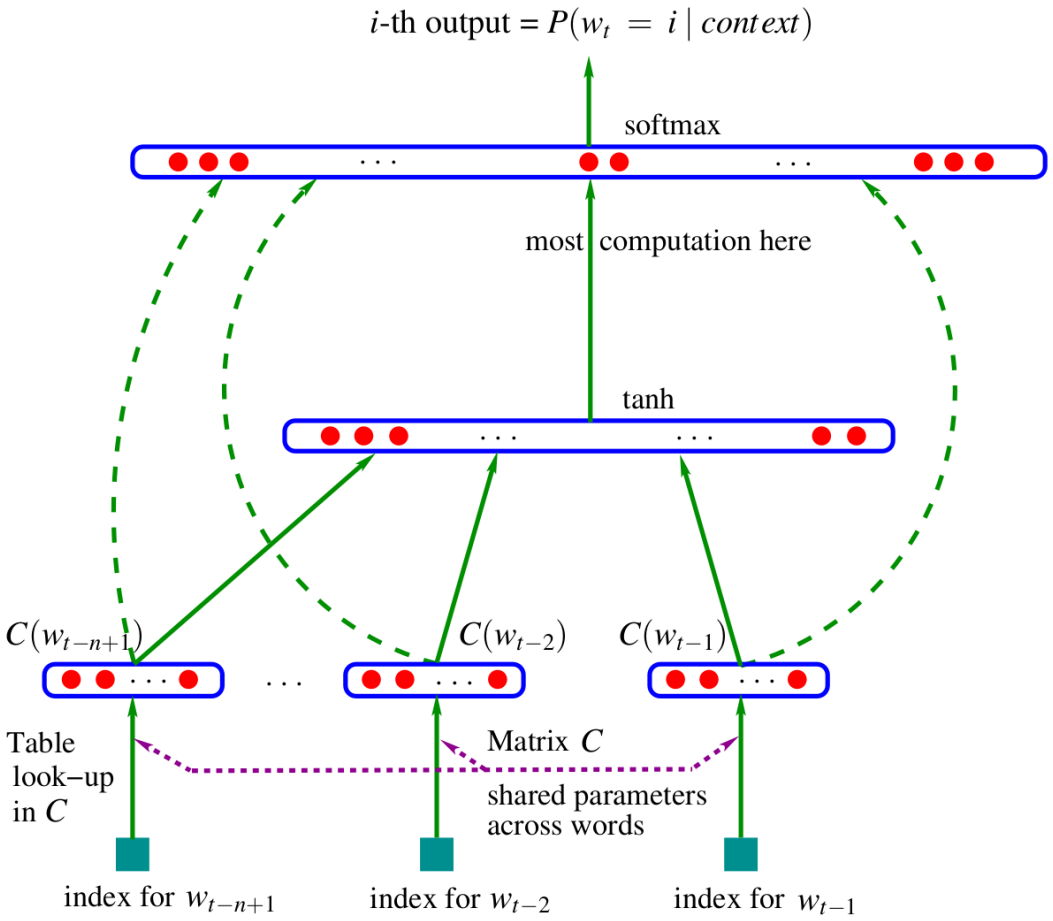


Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. J. Mach. Learn. Res., 3, 1137–1155.

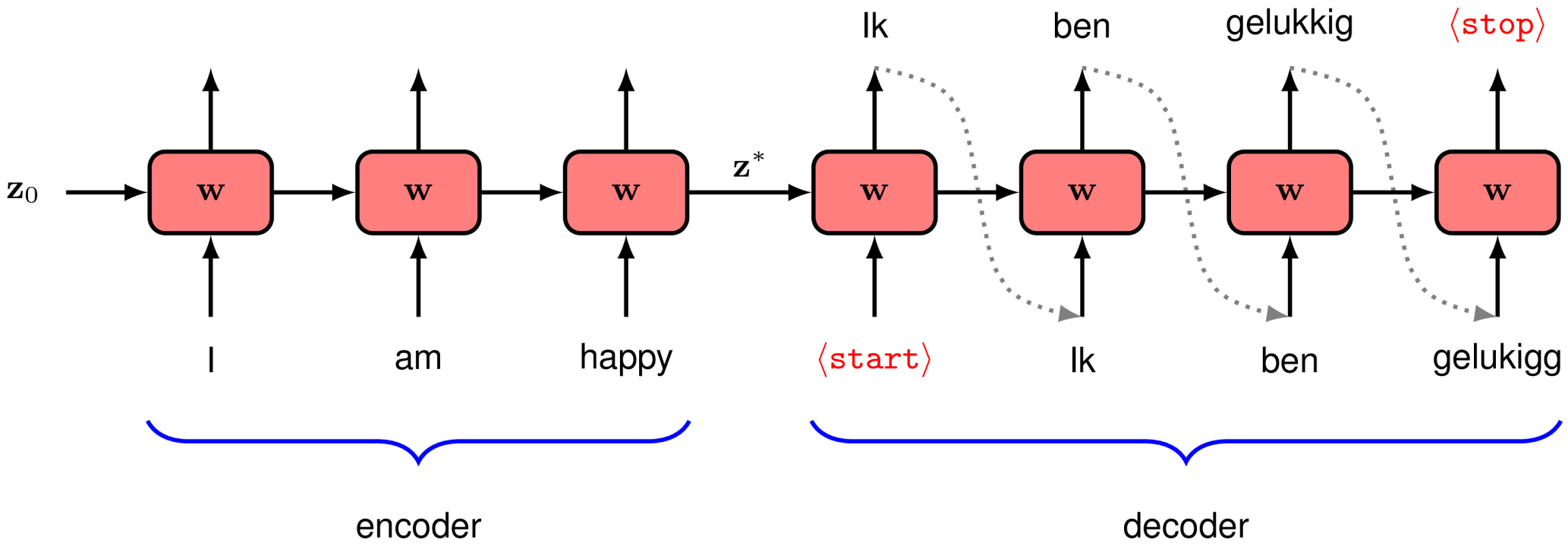
Bengio et al. (2003)

These early neural language models (NLMs) have intuitive limitations:

1. They ignore the global context provided by prefix tokens further than k tokens away (and typically $k = 5$ only)
2. They use a different set of parameters for each position in the prefix window
3. They have a relatively small number of parameters, which limits their expressiveness.

Sun, S., & Iyyer, M. (2021). Revisiting Simple Neural Probabilistic Language Models. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 5181–5188. https://doi.org/10.18653/v1/2021.naacl-main.407

Extensions to RNNs



Cho, K., Van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1724–1734. https://doi.org/10.3115/v1/D14-1179

Motivation to develop something new

One of the problems with standard RNNs is that they still deal poorly with long-range dependencies

This is especially problematic for the domain of natural language where such dependencies are widespread

In a long passage of text, a concept might be introduced that plays an important role in predicting words occurring much later in the text

At each new step of the RNN, information from longer before “washes out”

Motivation to develop something new

Also, with such an RNN approach, the entire concept of the English sentence must be captured in the single hidden vector $\mathbf{z}^*$ of fixed length

The network can start to generate the output translation only once the full input sequence has been processed

This becomes increasingly problematic with longer sequences

This is known as the **bottleneck problem** because it means that a sequence of arbitrary length has to be summarized in a single hidden vector of activations