Masked Language Models

& Subword Tokenization

Marcel Bollmann

Department of Computer Science (IDA)

LINKÖPING UNIVERSITY

Partly based on Jurafsky & Martin (2026)
and material from Taylor Sorensen.

- We learned about **language modelling** using simple n-gram models.

$$P(\text{‘cheese’} \mid \text{‘Garrotxa is a’})$$

- We learned that **word embeddings** from neural networks can give us more powerful word representations.

Garrotxa **is** **a** cheese **from** **goat** **milk**

What if we do language modeling with neural networks?

Shortcomings of the word2vec approach

- Word embeddings from *e.g.* word2vec are **static**.
  - There is one fixed embedding vector for each word, regardless of the context.

- **But:** The meaning of words is very often **context-dependent**.

1
- She likes to eat **cookies** with milk.
- This website uses **cookies** to store your information.

2
- He has an **interest** in machine learning.
- The bank's **interest** rate went up.

Outline

Masked Language Modelling

- BERT
- Transformers
- Contextual Embeddings

Pre-Training and Fine-Tuning

- Pre-Training
- Fine-Tuning
- Capabilities of BERT Models

Subword Tokenization

- Motivation
- Byte-Pair Encoding
- Examples

What is Masked Language Modelling?

Masked language modelling

- Previously: Language modelling as **predicting the next word** in a sequence

$$P(\text{`cheese'} \mid \text{`Garrotxa is a'})$$

$$P(\text{`milk'} \mid \text{`cheese from goat'})$$

- Masked language modelling: **predicting a masked-out word** in a sequence

$$P(\text{`cheese'} \mid \text{`Garrotxa is a \_\_\_\_\_ from goat milk'})$$

- This uses context from both **before *and* after** the word to be predicted!

BERT: Bidirectional Encoder Representations from Transformers

- **BERT** is a neural network model trained on **masked language modelling.**
  - Based on the Transformer architecture
- It **outperformed all other models** at the time of its release (2018) on all kinds of different language technology tasks & benchmarks.
  - Spawned an entire "family" of similar models for specific tasks & languages
- It can be used for **text classification** tasks, but also for **sequence labelling.** (→ *next lecture!*)

Masked language modelling in BERT

- For training, 15% of tokens are **randomly selected** to be part of the masking.

'Garrotxa is a cheese from **goat** milk'

1. 80% chance that the token is **replaced** with special token '**[MASK]**':

   'Garrotxa is a cheese from **[MASK]** milk'

2. 10% chance that the token is **replaced** with another **random token**:

   'Garrotxa is a cheese from **prevent** milk'

3. 10% change that the token is **left unchanged**:

   'Garrotxa is a cheese from **goat** milk'

Masked language modelling in BERT

- In all three cases, the model is then **trained to predict the word** that belongs in the selected position.

  1. ‘Garrotxa is a cheese from **[MASK]** milk’ $\longrightarrow$ ‘**goat**’
  2. ‘Garrotxa is a cheese from **prevent** milk’ $\longrightarrow$ ‘**goat**’
  3. ‘Garrotxa is a cheese from **goat** milk’ $\longrightarrow$ ‘**goat**’

We need to talk about **transformers**...

From words/documents to sequences

- For **text classification**, we talked about models that…
  - Take **an entire document** as **input**
  - Predict a **single label** for the entire document as **output**

- For **word embeddings**, we talked about models that…
  - Take a **bag of words** as **input**
  - Predict a **single label** ("real"/"fake" context words) as **output**

- For **language modelling**, we talked about models that…
  - Take an n-**gram** of fixed size as **input**
  - Predict a **single word** that comes next as **output**

- We really want to model **language as a sequence of tokens!**

Transformers

- The **transformer** is a specific kind of neural network architecture.
  - Introduced in 2017 by Vaswani et al. (paper with *over 200,000 citations* by now!)
- It is built upon the **attention mechanism**, a technique that allows the neural network to "look at" and put different "weight" on different tokens in a sentence.

Side note

This visual storytelling article explains the idea behind transformers very elegantly:

- **Generative AI exists because of the transformer**

The intuition behind attention

Remember the distributional hypothesis...

Words that appear in similar contexts have similar meanings.

- To determine meaning, **some words are more important** than others:

She likes to **eat** cookies with **milk**.

This **website** uses cookies to **store** your **information**.

- A transformer computes **contextual embeddings** that differ based on context!

Masked language modelling with transformers

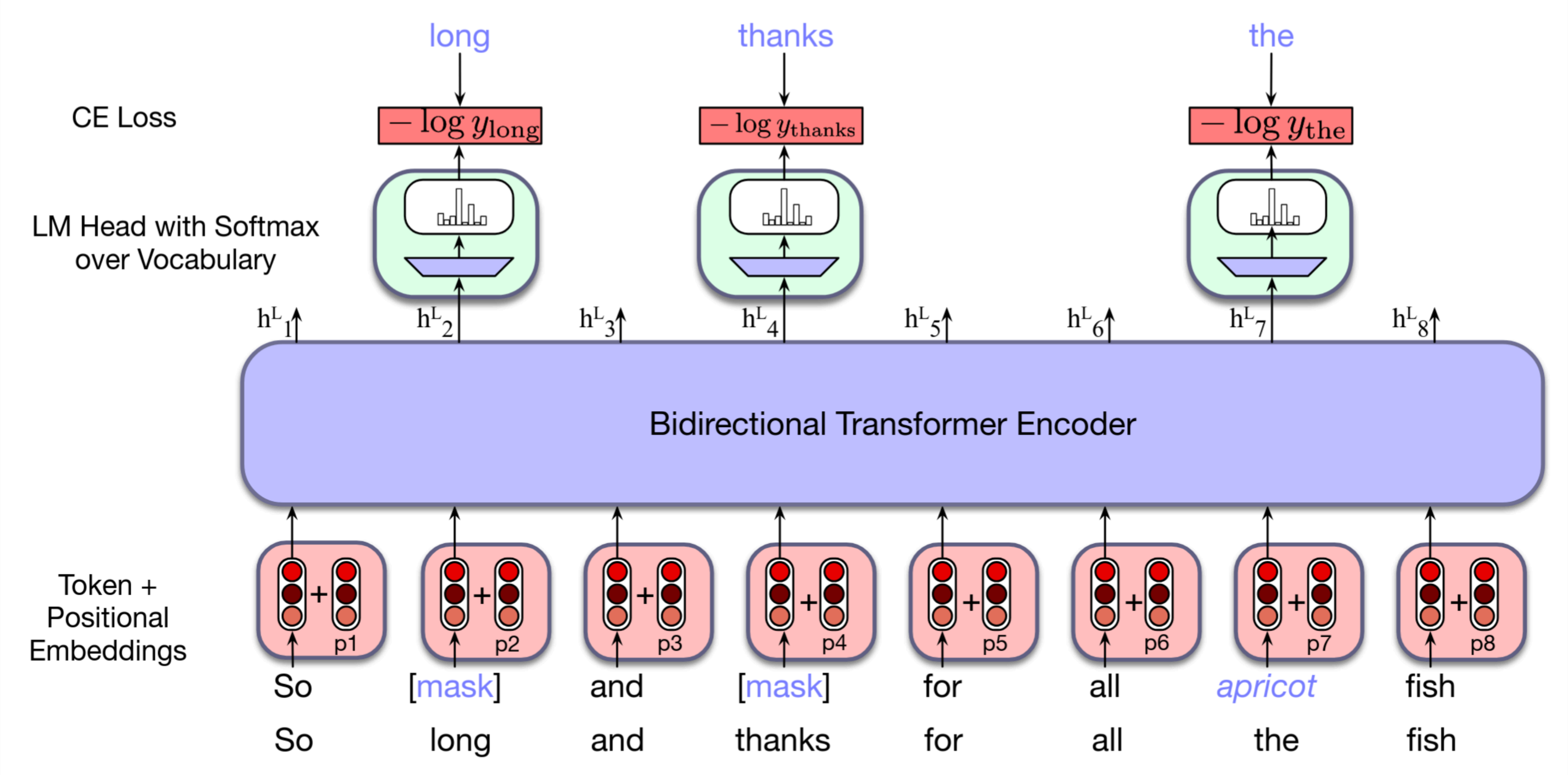


Figure 10.3 from Jurafsky & Martin (2026)

What kind of knowledge can a model “learn” from masked language modelling?

Consider these examples...

1. ‘The capital of __________ is Berlin.’
2. ‘Kenya’s athlete broke the world __________ in long jump.’
3. ‘I know this man, I’ve seen __________ before!’
4. ‘This movie was so __________ that I fell asleep.’
5. ‘Yesterday we met __________ new neighbours.’

Consider these examples...

1. ‘The capital of **Germany** is Berlin.’ — **World knowledge**
2. ‘Kenya’s athlete broke the world **record** in long jump.’ — **Lexical knowledge**
3. ‘I know this man, I’ve seen **him** before!’ — **Co-reference**
4. ‘This movie was so **boring** that I fell asleep.’ — **Sentiment**
5. ‘Yesterday we met **our**/**the**/***taxi** new neighbours.’ — **Grammatical constraints**

Intuition

Training a large neural network on this masked language modelling task will result in all these different **types of knowledge** being **encoded** in its parameters.

Contextual embeddings

- word2vec produces **static embeddings**, *i.e.* one for each word **type.**
  - type ≈ dictionary entry
- BERT produces **contextual embeddings**, *i.e.* one for each word **instance.**
  - instance ≈ the word in context/in a given sentence

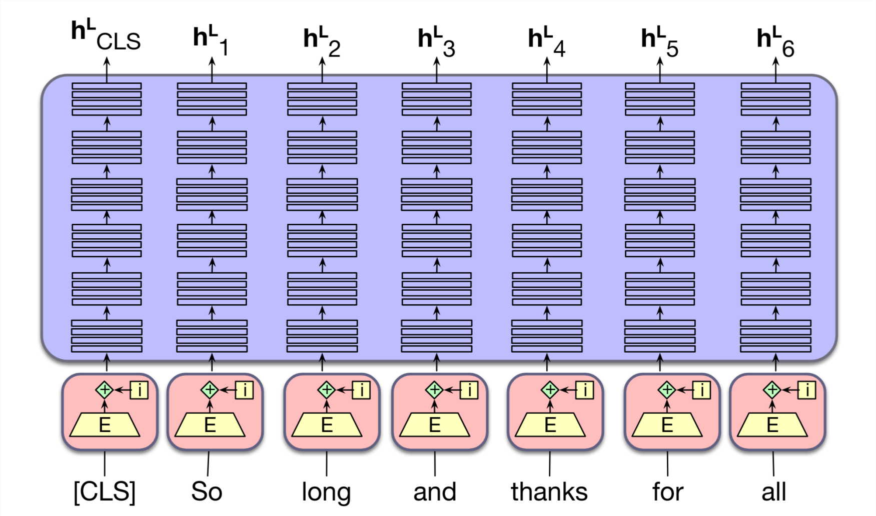


Figure 10.5 from Jurafsky & Martin (2026)

Contextual embeddings for the token “die”

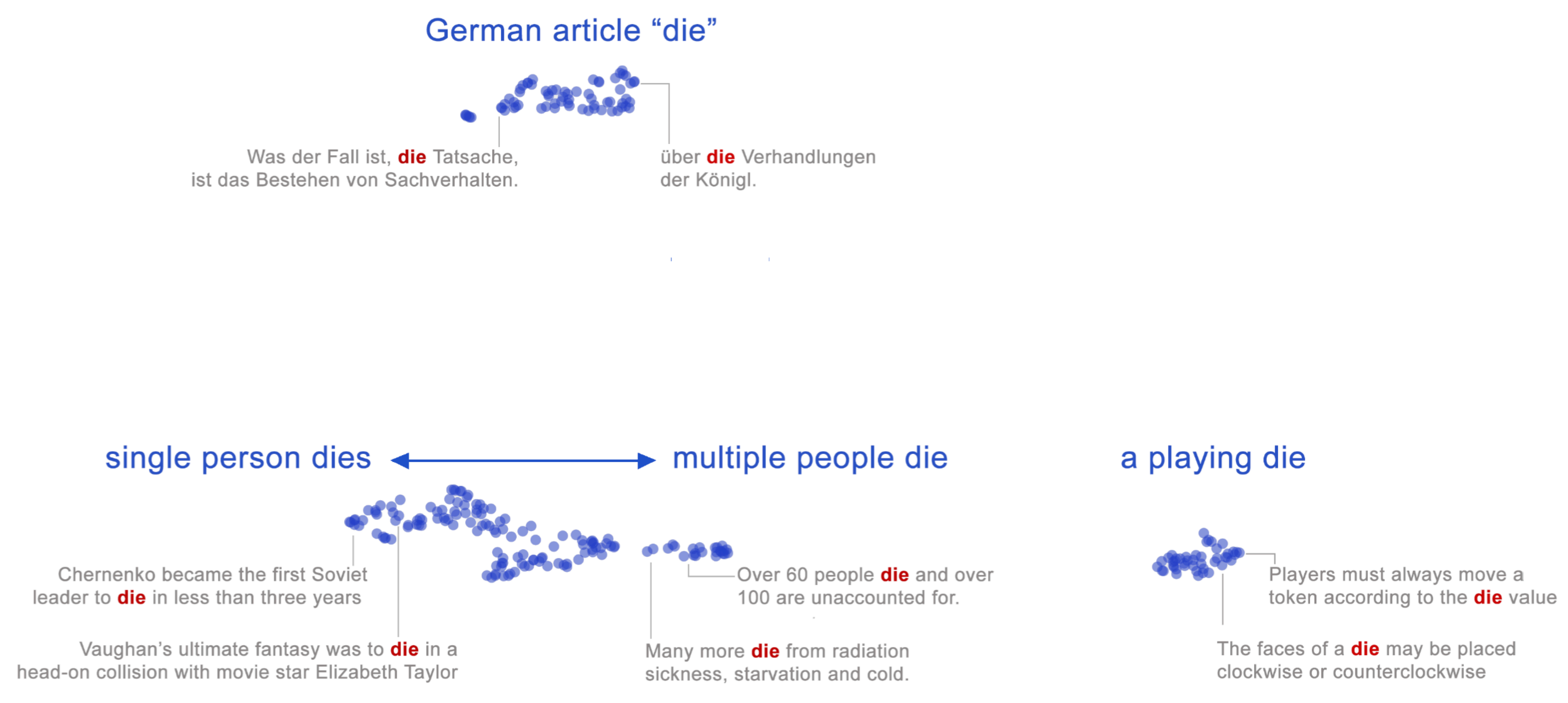


Figure 10.6 from Jurafsky & Martin (2026)

Important Concepts

- masked language modelling
- BERT
- contextual embeddings

- For more **technical** details, see Jurafsky & Martin (2026):
  - Chapter 8: Transformers
  - Chapter 10: Masked Language Models

Pre-Training and Fine-Tuning

or:

What can we **do** with BERT models?

Many ways to use a BERT model

1. We can use a BERT model to **predict words**.
   - But: Expects context from left to right, so not as useful for *e.g.* predictive typing.

2. We can **extract embeddings** from a BERT model and work with them.
   - Contextual word embeddings, sentence embeddings, document embeddings…
   - Applications: Clustering, classification, semantic search…

3. We can **fine-tune** a BERT model on any other classification task!

The pre-train and fine-tune paradigm

1. **Pre-train** a BERT model on the masked language modelling task.
   - Training data is "plain text", no annotations required – we can easily get lots of data!
   - "Pre"-training because this is not the end goal, just a first step.
   - 👎 Requires **powerful hardware** and quite some time…

2. **Fine-tune** the model on whatever classification task we are interested in.
   - Fine-tuning ≈ continuing to train, but with different data & labels
   - 👍 Much more **efficient**: model has already learned a lot of knowledge!

Core idea:

Re-using the general knowledge
that a pre-trained model has learned
makes **training on others tasks** much easier
than starting “from scratch”!

BERT pre-training in numbers

Model	Parameters	Training Data	Training Hardware & Time
BERT-base	110 M	16 GB	4× Cloud TPUs for 4 days
BERT-large	340 M	16 GB	16× Cloud TPUs for 4 days
RoBERTa-base	110 M	160 GB	1024× V100 GPUs
RoBERTa-large	340 M	160 GB	1024× V100 GPUs
ModernBERT-base	149 M	8000 GB*	8× H100 GPUs for 10 days
ModernBERT-large	395 M	8000 GB*	8× H100 GPUs for 22 days

*estimated

Sources: Devlin et al. (2019); RoBERTa-base model card; Warner et al. (2024)

Environmental impact

- We can measure the **energy and carbon footprints** of BERT.
  - Depends on hardware; numbers below are from running on 4× RTX 8000 GPUs

Task	Emissions (kg CO_2)
Pre-training	174.600
Fine-tuning on MNLI (Natural Language Inference)	0.445
Fine-tuning on IMDB (Sentiment Analysis)	0.072
Fine-tuning on SQuAD v2 (Question Answering)	0.377

- **Fine-tuning** is orders of magnitude **more efficient** than pre-training!

Source: Wang et al. (2023)

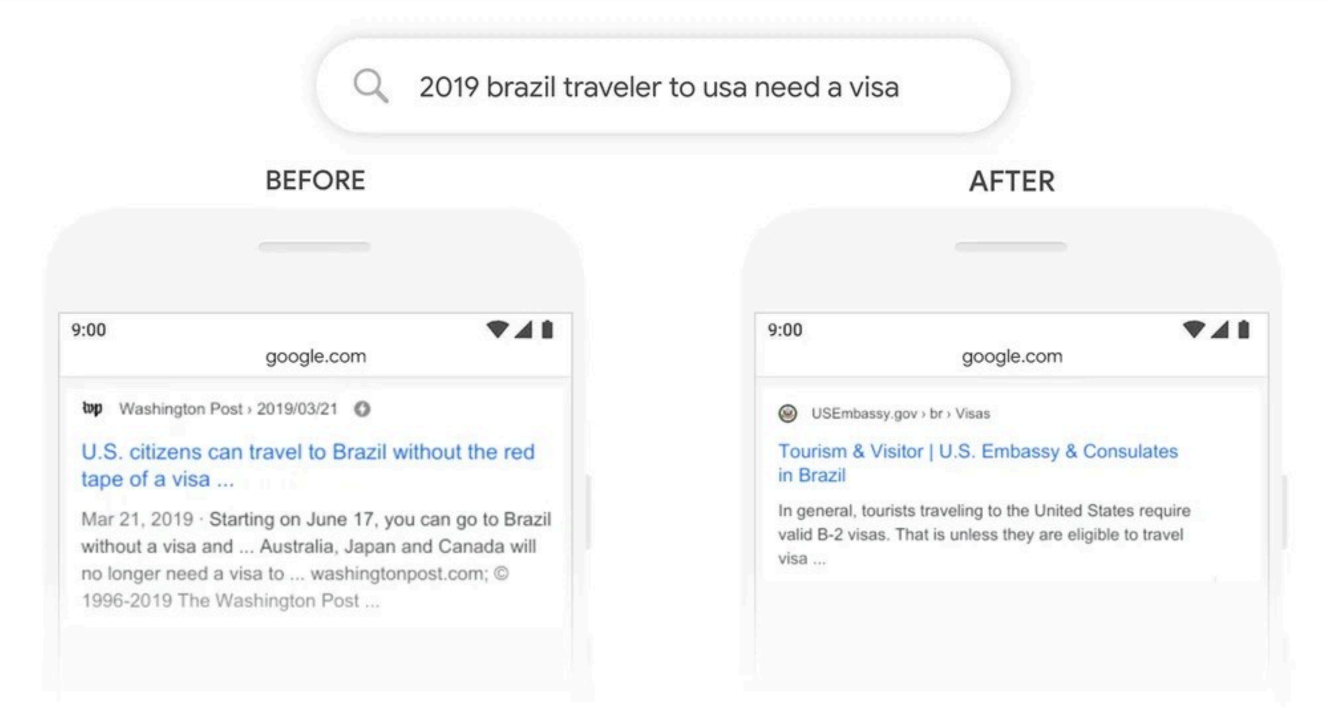
How to fine-tune a BERT model?

1. Many **pre-trained BERT models** are made freely available & can be downloaded from the **HuggingFace Model Hub**.
   - General-purpose **English** models: *BERT, RoBERTa, ModernBERT*
   - General-purpose **multilingual** models: *XLM-RoBERTa, mmBERT*
   - **Language-specific** models: *CamemBERT, WangchanBERTa, BERT-Swedish*
   - **Domain-specific** models: *NewsBERT, MusicBERT, ChemBERTa*
   - **Smaller** models: *DistilBERT, TinyBERT*

2. The **text classification guide** from HuggingFace is a good place to get started with fine-tuning your own model.
   - Remember that you will most likely need access to a GPU.

BERT improved Google Search

- Before BERT, Google Search was bad at **capturing word-order dependencies**.
  - "Brazil to USA" vs. "USA to Brazil"



Source: Nayak (2019)

BERT performs well on "language understanding" benchmarks

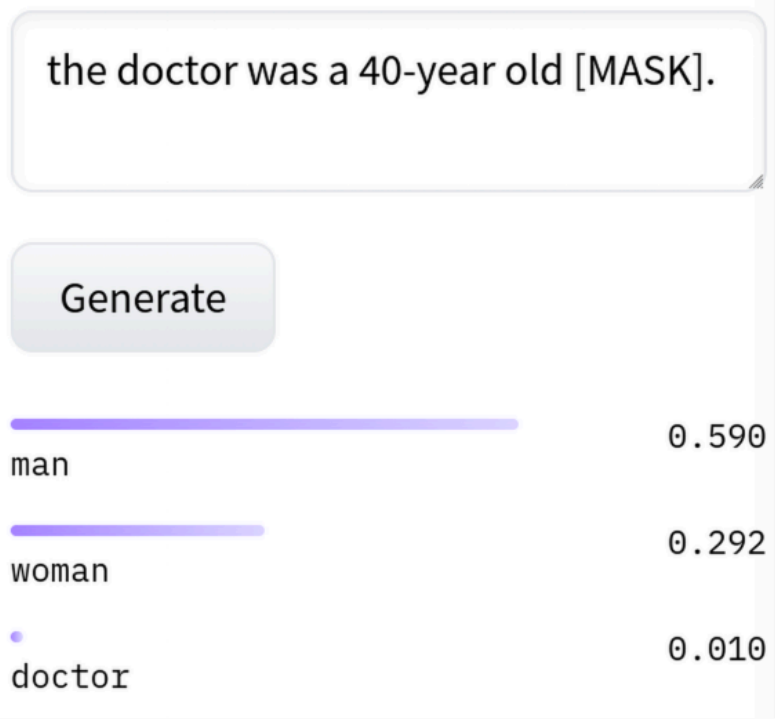
BERT's Performance on GLUE:

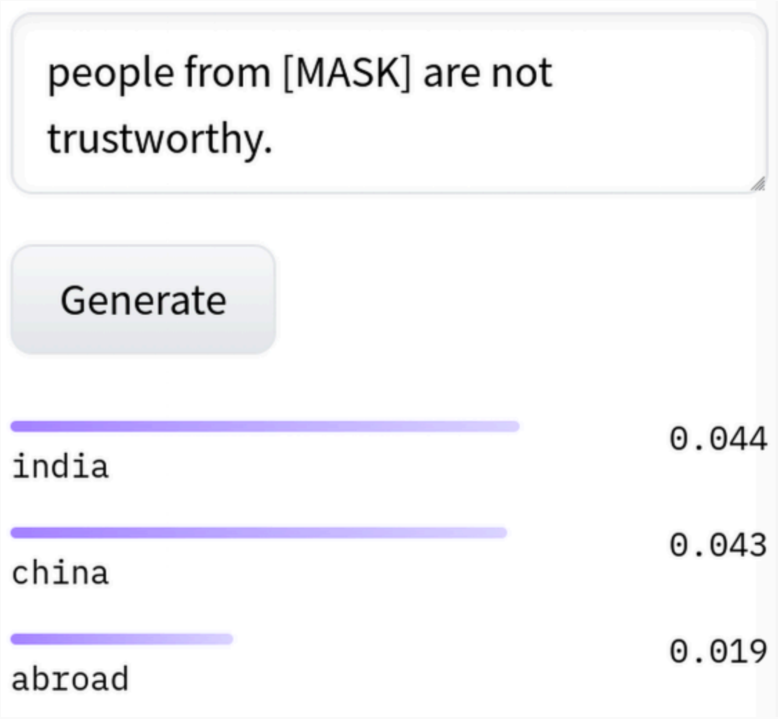
Task	Average	Grammatical	Sentiment Analysis	Similarity	Paraphrase	Question Similarity	Contradiction	Answerable	Entail
BERTLARGE	**82.1**	**60.5**	**94.9**	**86.5**	**89.3**	**72.1**	**86.7/85.9**	**92.7**	**70.1**
BERTBASE	79.6	52.1	93.5	85.8	88.9	71.2	84.6/83.4	90.5	66.4
OpenAI GPT	75.1	45.4	91.3	80.0	82.3	70.3	82.1/81.4	87.4	56.0
Pre-OpenAI SOTA	74.0	35.0	93.2	81.0	86.0	66.1	80.6/80.1	82.3	61.7
BiLSTM+ELMo+Attn	71.0	36.0	90.4	73.3	84.9	64.8	76.4/76.1	79.8	56.8

Source: Muller (2022)

BERT encodes societal biases

- BERT models can encode **gender bias** and other **harmful stereotypes**.





Try it yourself: BERT base model on Huggingface

Important Concepts

- pre-training vs. fine-tuning
- model bias
- where to find pre-trained models *(might be relevant for your project!)*

Subword Tokenization

Previously...

Definition

Tokenization is the task of segmenting a text into *words* or *subword* units.

- So far, we mostly assumed that "tokens ≈ words", with some special treatment of *e.g.* punctuation marks or abbreviations.

Before Tokenization	`"This hotel was awesome!"`
After Tokenization	`["This", "hotel", "was", "awesome", "!"]`

Why is “word level” not good enough?

- We can only have a **fixed vocabulary** of tokens.
- Words that are not part of the vocabulary **cannot be represented at all** in our model.
  - Remember: *UNK* token as a workaround...
- There is no such thing as a “list of all words,” and **new words are created** all the time ⟶

Source: @nyt-first-said.bsky.social

Tokens vs. bytes or characters

- Let's take this input sentence with **four tokens**:

```
['It', 'is', 'awesome', '!']
```

- Why can't we simply use **Unicode characters** as tokens instead?

```
['I', 't', ' ', 'i', 's', ' ', 'a', 'w', 'e', 's', 'o', 'm', 'e', '!']
```

⚡ Input **sequences get very long** – harder to learn information from

⚡ Characters are **much less informative** than words (→ *poor inductive bias*)

Subword tokenization

- **Subword tokenization** uses a vocabulary consisting of words, characters, and "subwords", *i.e.* units that are smaller than words.

```
['These', 'people', 'are', 'techno', '##fe', '##uda', '##lists', '.']
```

indicates continuation of previous word

- **Common words** are represented by a single token each.
  - Better inductive bias & shorter sequence length
- **Rare or new words** are represented by smaller units, *i.e.* subwords.
  - We will never have out-of-vocabulary words!

Byte-pair encoding

- **Byte-pair encoding (BPE)** is a commonly used subword tokenization algorithm.
  - Originally a data compression technique!
  - Proposed for use in tokenization by Sennrich et al. (2016)

Idea

playing, seeing, wailing, fighting, rubbing, coding, eating, ...

- '*ing*' is a very frequent combination of characters, therefore...
- '*ing*' should get its own token

- BPE iteratively **merges the most frequent pairs** of adjacent characters/tokens.

Byte-pair encoding: training

- We need **training data** to train the BPE tokenizer.
  - This can be the same data that will be used to train the model later.

> *It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, ...*

- We also need to decide on the **size of the vocabulary**.
  - **Hyperparameter** of the algorithm, *i.e.* needs to be set before training

Byte-pair encoding: algorithm

- Start with a vocabulary of all unique **characters**.

special word boundary symbol

Training data

```
i t ␣ w a s ␣ t h e ␣ b e s t ␣ o f
␣ t i m e s ␣ i t ␣ w a s ␣ t h e ␣
w o r s t ␣ o f ␣ t i m e s ␣ i t ␣
w a s ␣ t h e ␣ a g e ␣ o f ␣ w i s
d o m ␣ i t ␣ w a s ␣ t h e ␣ a g e
␣ o f ␣ …
```

Vocabulary

```
a b d e f g h i l m n o r s t w ␣
```

Byte-pair encoding: algorithm

- Start with a vocabulary of all unique **characters**.
- BPE tokenizers **never merge across word boundaries**.
  - We can start by splitting up the document after each word boundary and counting the words.

Training data

4	i t ␣	2	a g e ␣
4	w a s ␣	1	b e s t ␣
4	t h e ␣	1	w o r s t ␣
4	o f ␣	1	w i s d o m ␣
2	t i m e s ␣		

Vocabulary

a b d e f g h i l m n o r s t w ␣

Byte-pair encoding: algorithm

1. **Find the most frequent pair** of adjacent tokens in the training data.
2. **Create a new token** by merging the most frequent pair.
3. **Update the training data** to use the new token.
4. **Repeat** until desired vocabulary size is reached.

Training data

4	i t␣	2	a g e ␣
4	w a s ␣	1	b e s t␣
4	t h e ␣	1	w o r s t␣
4	o f ␣	1	w i s d o m ␣
2	t i m e s ␣		

Vocabulary

a b d e f g h i l m n o r s t w ␣ t␣

Byte-pair encoding: algorithm

1. **Find the most frequent pair** of adjacent tokens in the training data.
2. **Create a new token** by merging the most frequent pair.
3. **Update the training data** to use the new token.
4. **Repeat** until desired vocabulary size is reached.

Training data

4	i t␣	2	a g e␣
4	w a s␣	1	b e s t␣
4	t h e␣	1	w o r s t␣
4	o f ␣	1	w i s d o m ␣
2	t i m e s␣		

Vocabulary

a b d e f g h i l m n o r s t w ␣

t␣ s␣ e␣ **it␣**

Byte-pair encoding: segmenting new text

- With the learned vocabulary, we can now **tokenize new datasets**.
  - *e.g.* a test dataset we want to use

Vocabulary

`a b d e f g h i l m n o r s t w ␣`
`th wa wi e␣ s␣ t␣ es␣ it␣ st␣ was␣`

- Example: "*wit*" could be represented as `['w', 'it␣']`

- **But:** why not `['wi', 't␣']` or `['w', 'i', 't␣']`?
  - There are often multiple ways to represent the same word with subword tokens!
  - One (deterministic) solution: Run the merges in the same order as during training

Byte-pair encoding: turning tokens back into text

- Mapping from tokens back to text, a.k.a. **decoding**, is easy!
  - …as long as we have a word boundary marker

```
['the␣', 'age␣', 'of␣', 'fo', 'ol', 'ish', 'ness␣']
```

⬇️

```
"the age of foolishness"
```

- There are different conventions for how to mark word boundaries…

```
['␣the', '␣age', '␣of', '␣fo', 'ol', 'ish', 'ness']
['the', 'Ġage', 'Ġof', 'Ġfo', 'ol', 'ish', 'ness']
['the', 'age', 'of', 'fo', '##ol', '##ish', '##ness']
```

Example: BERT tokenizer

- How would BERT tokenize the opening of Mary Shelley's Frankenstein?

```
You will re ##jo ##ice to hear that no disaster has accompanied
the commencement of an enterprise which you have regarded with
such evil fore ##bo ##ding ##s . I arrived here yesterday , and
my first task is to assure my dear sister of my welfare and
increasing confidence in the success of my undertaking .
```

Example: BERT tokenizer

- How would BERT tokenize a paragraph about Linköping University?

```
The origins of Link ##ö ##ping University date back to the
1960s . In 1965 , The Swedish National Legislative Assembly
( R ##ik ##s ##da ##g ) decided to locate some programmes within
the fields of technology and medicine to Link ##ö ##ping .
```

Properties of BPE tokenization

- BERT has a fixed vocabulary size, so **rare words** get split up into multiple tokens.

$$\textit{forebodings} \longrightarrow \texttt{fore ##bo ##ding ##s}$$

- BERT was trained on English data, so **non-English words** will have been rare.

$$\textit{Riksdag} \longrightarrow \texttt{R ##ik ##s ##da ##g}$$

- But this is still better than **not being able to represent the word** at all!
  - Replacing these words with `[UNK]` loses a lot of information.

Important Concepts

- subword tokenization
- byte-pair encoding (BPE) — *conceptually*
- advantages & drawbacks of subword tokenization