Unit 12
Text Embeddings

EE-UY 4563/EL-GY 9143: INTRODUCTION TO MACHINE LEARNING

PROF. SUNDEEP RANGAN

Learning Objectives

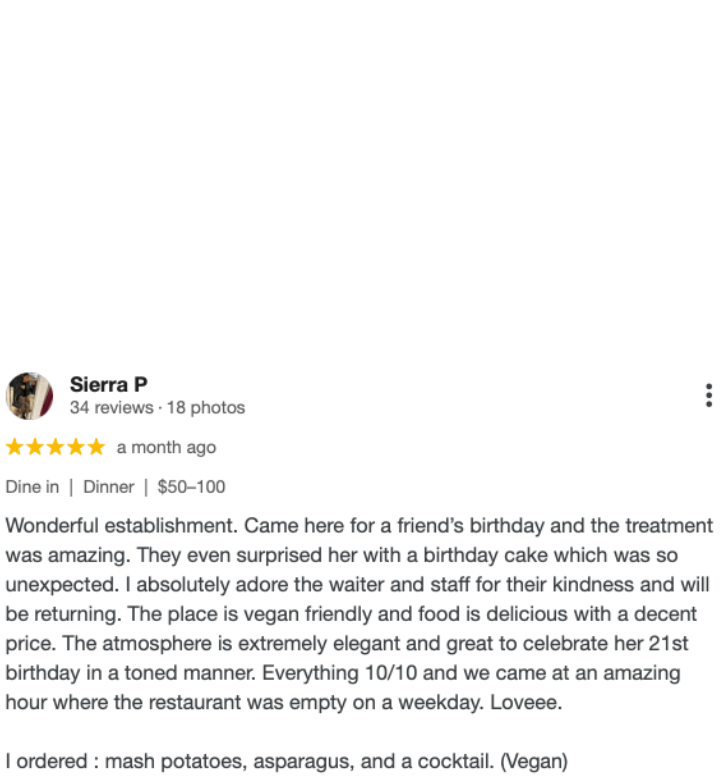
- ❑ Describe the function of a text embedding
- ❑ Describe the role of tokenization in a text embedding
- ❑ Perform Byte-Pair Encoding on simple corpusS
- ❑ Describe attention architecture in modern LLMs
- ❑ Encode text with a pre-trained text embedding
- ❑ Use the text embedding for simple downstream tasks such as classification

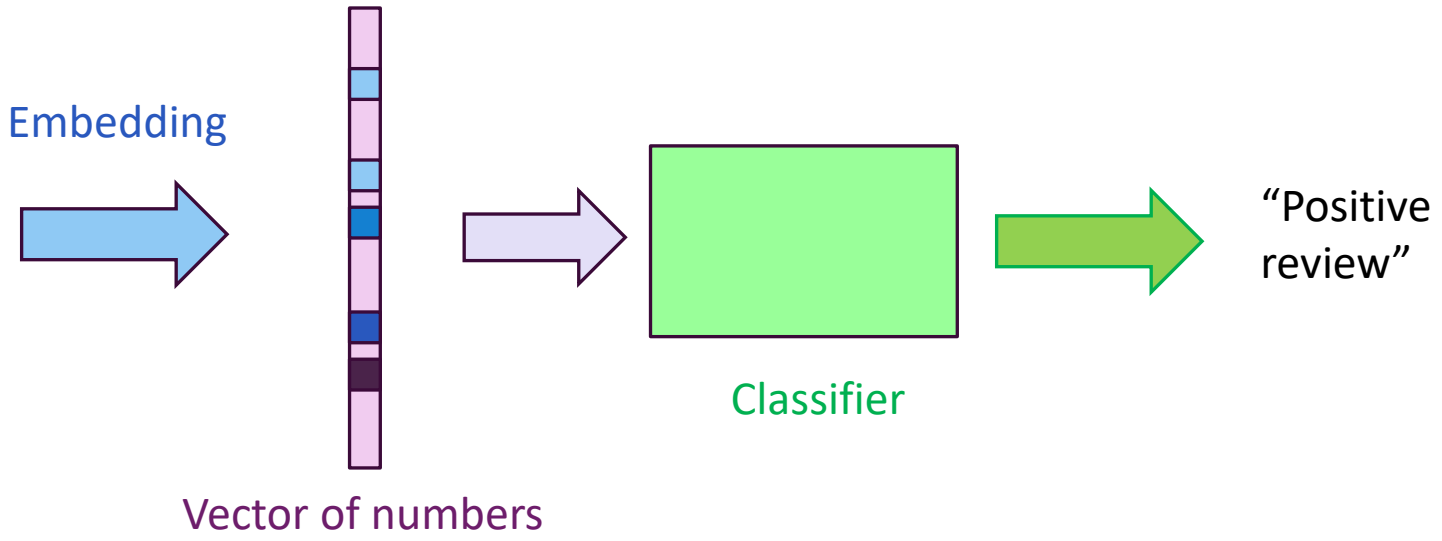
Outline

- Text Embeddings and Tokenization
- Modern LLMs at a Glance
- Using a pre-trained LLM

What is a Text Embedding?

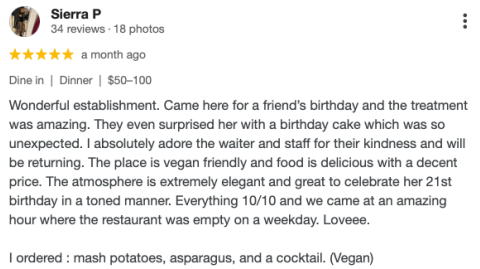
- ML models work on numeric data
- Input is typically a vector
- Embedding: Maps text to a vector
- Used by downstream tasks
  - E.g., classification



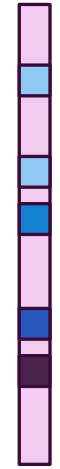


Three Key Goals

- **Compactness**
  - **Low dimensionality** relative to the input space
  - Enables **efficient storage** and **faster downstream computation**
- **Expressiveness**
  - Capture the **salient features** of the input
  - Preserve **semantic, syntactic, and contextual** information
- **Semantic Similarity**
  - Texts with **similar meaning** should have **nearby vectors**
  - Enables clustering, retrieval, and generalization

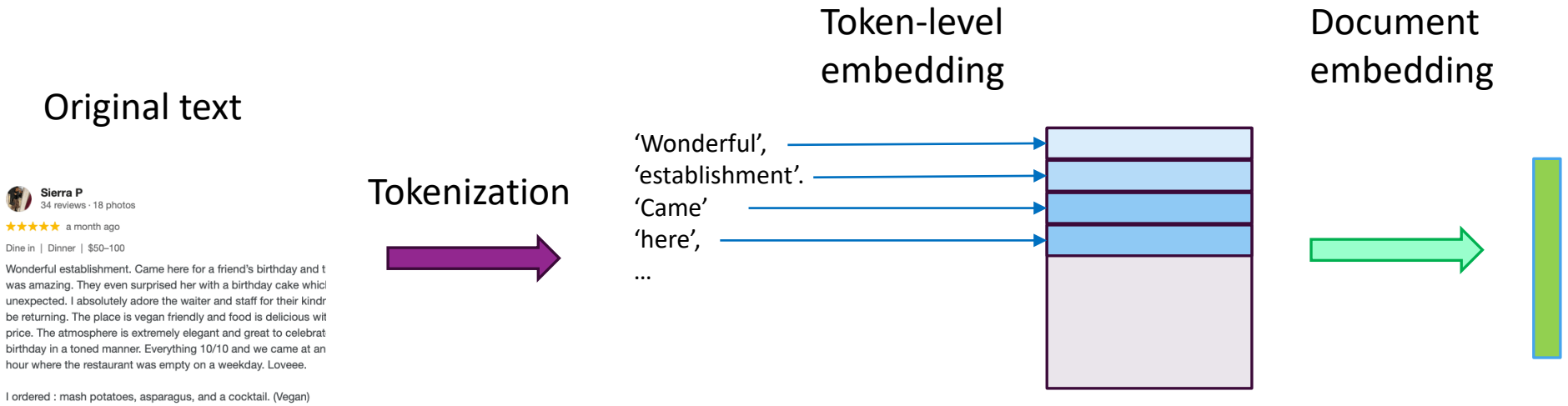


Embedding



Vector

Typical Embedding Flow



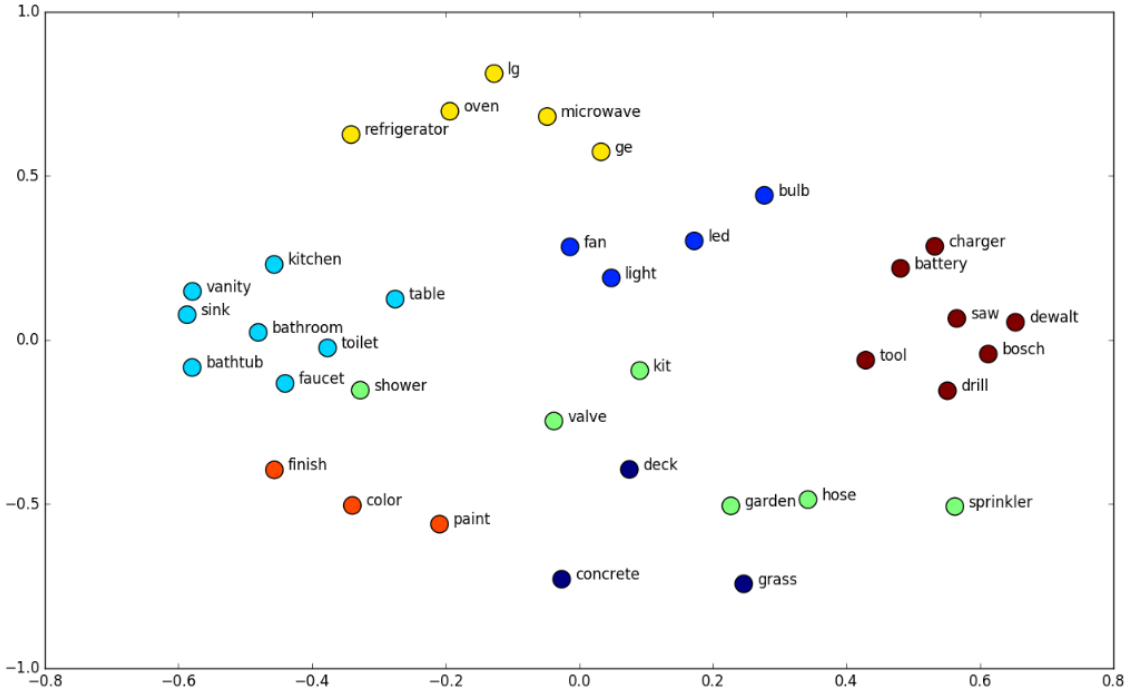
Character and Word Tokenization

- Early systems used one of two simple possibilities for tokens: Character or Word
- Character level: Each character is a token
- Word level: Each work is a token
- Consider example: “The food was delicious”

Encoding type	Token	Robustness	Keeps semantic meaning
Character-level	‘T’, ‘h’, ‘e’, ‘ ‘, ‘f’, …	Simple, works on all text	No semantic meaning
Word-level	‘The’, ‘food’, ‘was’, ‘delicious’	Only if all words are in “vocabulary”	Yes, with proper training

Word Embeddings

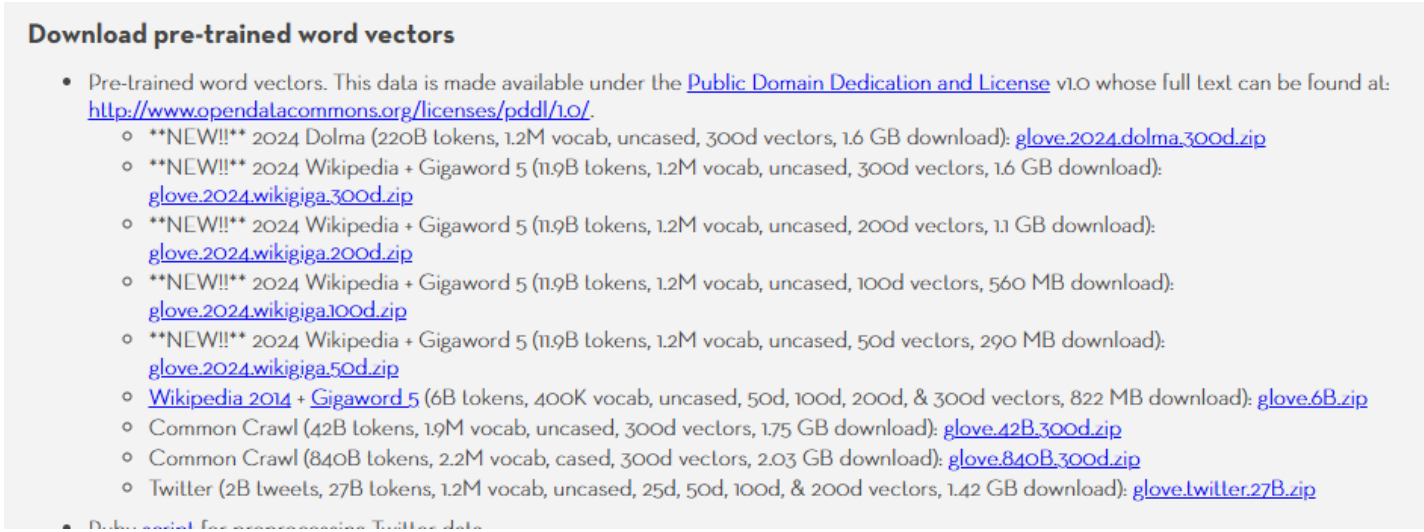
- Used with word-level tokenization
- Assign each word token to a vector
- Training goal:
  - Assign similar words similar embeddings
  - Trained by looking a “context word”
  - Context word = word close to target
- Example: GloVe
  - Global Vectors for Word Representation
  - Stanford 2014



Limitations of Word Embeddings

- Word embeddings were widely-used for many years (up to about 2016)
- But hit many limitations
- Vocabulary size must grow
- Even with large vocabularies may out-of-vocabulary words
- Also, words lose context in sentence

https://nlp.stanford.edu/projects/glove/



Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - **NEW!!** 2024 Dolma (220B tokens, 1.2M vocab, uncased, 300d vectors, 1.6 GB download): glove.2024.dolma.300d.zip
  - **NEW!!** 2024 Wikipedia + Gigaword 5 (11.9B tokens, 1.2M vocab, uncased, 300d vectors, 1.6 GB download): glove.2024.wikigiga.300d.zip
  - **NEW!!** 2024 Wikipedia + Gigaword 5 (11.9B tokens, 1.2M vocab, uncased, 200d vectors, 1.1 GB download): glove.2024.wikigiga.200d.zip
  - **NEW!!** 2024 Wikipedia + Gigaword 5 (11.9B tokens, 1.2M vocab, uncased, 100d vectors, 560 MB download): glove.2024.wikigiga.100d.zip
  - **NEW!!** 2024 Wikipedia + Gigaword 5 (11.9B tokens, 1.2M vocab, uncased, 50d vectors, 290 MB download): glove.2024.wikigiga.50d.zip
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip

Sub-Word Tokenization

- Sub-word tokenization: Overcomes limitations of character and word-level tokenization
- Became widely-used around 2016-2018, especially with BERT
- Consider example: **"The food was unicornishly delicious :)"**

Encoding type	Token	Procs and cons
Character-level	‘T’, ‘h’, ‘e’, ‘ ‘, ‘f’, …	Simple, works on all text, but no semantic meaning
Word-level	‘The’, ‘food’, ‘was’, OOV, ‘delicious’, OOV	Keeps semantic meaning of words, but may be out of vocabulary items
Sub-word	'The', 'food', 'was', 'unicorn', 'ish', 'ly', 'delicious'	Applies to all words and keeps semantic meaning

ASCII Characters

- American Standard Code for Information Interchange
- 1 byte (8 bits) for each character
  - Encodes 128 characters
  - 95 English characters (upper and lower case)
  - 10 digits (0 to 9)
  - Punctuation characters
  - 33 control characters (CR, LF, Tab, …)
- Limitations
  - Non-English characters
  - Accented letters (e.g., é, ñ, ü)
  - Non-Latin scripts (e.g., Cyrillic, Arabic, Chinese)
  - Wastes the 8th bit in modern systems
  - Inflexible for globalization, multilingual content, or emoji

$b_7 b_6 b_5$ → / Bits	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 / Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	—	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

UTF-8 Encoding

- UTF-8: Unicode Transformation Format (8-bit)
- **Encodes all Unicode characters** using 1 to 4 bytes
- **Compatible with ASCII** — first 128 characters are identical
- **Efficient for English**, flexible for global languages
- **Variable-length encoding**:
  - 1 byte for basic Latin (e.g., A–Z)
  - 2–4 bytes for accented letters, symbols, and non-Latin scripts
- **Self-synchronizing** — easy to detect character boundaries
- **Most common encoding** on the web and in modern software

UTF-8 Examples

Character	Description	Unicode Code Point	UTF-8 Byte	UTF-8 Encoding (Hex)
A	Latin capital letter	U+0041	1 byte	41
é	Latin small letter e acute	U+00E9	2 bytes	C3 A9
你	Chinese character “you”	U+4F60	3 bytes	E4 BD A0
😊	Smiling face with smiling eyes	U+1F60A	4 bytes	F0 9F 98 8A

Byte Pair Encoding

- Most common method used today for sub-word encoding
- Encoder is trained on a large (often multi-lingual) corpus
- Start with a token vocabulary of single bytes [0-255] or 256 tokens
- Look for most common pair and add to the vocabulary
- Keep repeating until a desired maximum vocabulary
- Frequent sequences are encoded together
- GPT-2: ~50000 tokens developed in Feb 2019 by Open AI

BPE Encoding 1: Create the Dictionary

- Original text: ['a', 'a', 'a', 'b', 'd', 'a', 'a', 'a', 'b', 'a', 'c'] , Token dictionary: [‘a’, ‘b’, ‘c’, ‘d’]
- Step 1: Most common pair ‘a’+’a’ (4 times)
  - Merged to token →’aa’. New dictionary [‘a’, ‘b’, ‘c’, ‘d’, ‘aa’]
  - Merged [‘aa’, ‘a’, ‘b’, ‘d’ ‘aa’, ‘a’, ‘b’, ‘a’, ‘c’]
- Step 2 Most common pair ‘aa’+’a’ (or ‘a’+’b’)
  - Merge ‘aa’+’a’ to ‘aaa’. New dictionary: [‘a’, ‘b’, ‘c’, ‘d’, ’aa’, ‘aaa’]
  - New text: [‘aaa’, ‘b’, ‘d’, ‘aaa’, ‘b’, ‘a’, ‘c’]
- Step 3: Most common pair: (‘aaa’+’b’)
  - Merge to ‘aaab’. New dictionary: [‘a’, ‘b’, ‘c’, ‘d’, ’aa’, ’aaa’, ‘aaab’]
  - New text: [‘aaab’, ‘d’, ‘aaab’, ‘a’, ‘c’].
- Step 4: Most common pair ‘a’, ‘c’ (there are others too)
  - New dictionary: [‘a’, ‘b’, ‘c’, ‘d’, ’aa’, ‘aaa’, ‘aaab’, ‘ac’]

BPE Encoding 2: Index the tokens

- Token dictionary from previous example:
  - [‘a’, ‘b’, ‘c’, ‘d’, ’aa’, ‘aaa’, ‘aaab’, ‘ac’]
- Assign each token an ID

Token	Token ID
a	0
b	1
c	2
d	3
aa	4
aaa	5
aaab	6
ac	7

BPE Encoding 3: Encode a New Sentence

- New example sentence: ‘aabacaadaaabac’
- Find longest tokens possible
  - [‘aa’, ‘b’ ‘ac’, ‘aa’, ‘d’, ‘aaab’, ‘ac’]
- Set IDs
  - [4, 1, 7, 4,3,6,7]
  -

Token	Token ID
a	0
b	1
c	2
d	3
aa	4
aaa	5
aaab	6
ac	7

Using a Pre-Trained Tokenizer

- Can download many tokenizers from Hugging Face

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://hug
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models o
  warnings.warn(
tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 2.04kB/s]
config.json: 100% 665/665 [00:00<00:00, 58.2kB/s]
vocab.json: 100% 1.04M/1.04M [00:00<00:00, 14.1MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 19.6MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 30.7MB/s]
```

Using the Tokenizer is Easy

```
example_text = "Hello world!"
encoded_text = tokenizer.encode(example_text)

print(encoded_text)
```

```
[15496, 995, 0]
```

```
example_text2 = "Hello 😊世界!"
encoded_text2 = tokenizer.encode(example_text2)

print(encoded_text2)
```

```
[15496, 30325, 232, 10310, 244, 45911, 234, 0]
```

Spaces are Part of the Token

- They are not removed
- They are kept along with all other punction

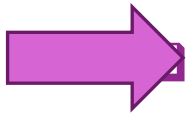
```
text = ['world', ' world', 'World']
for t in text:
  enc = tokenizer.encode(t)
  token_id = enc[0]
  token = tokenizer.convert_ids_to_tokens(token_id)
  print('text: \'%s\' ID: %s token: %s' % (t, enc, token))
```

```
text: 'world' ID: [6894] token: world
text: ' world' ID: [995] token: Ġworld
text: 'World' ID: [10603] token: World
```

Special character to represent ‘Space’ + ‘world’ is one token

Outline

- ❑ Tokenization
- ❑ Modern LLMs at a Glance
- ❑ Using a pre-trained LLM



Transformers

- Landmark paper, 2017
- Introduced the transformer architecture
- Originally for problems in NLP
- Now used in many fields:
  - ViT: An Image is Worth 16x16 Words, Dosovitskiy et al, 2020
  - DETRL End-to-End Object Detection with Transformers Carion et al, 2020

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

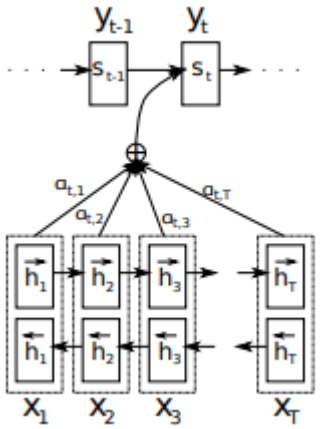
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

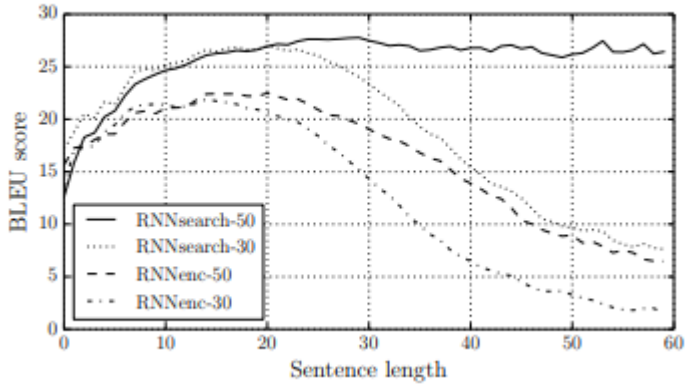
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Prior Work

- Pre-2014 - RNN, LSTM, GRU
  - RNNs struggled with long sequence data
  - Could not handle long sequence lengths (forgetful)
  - Example on next slide
- Neural Machine Translation by Jointly Learning to Align and Translate
  - Bahdanau, 2014
  - Landmark Paper
- Proposed New Architecture:
  - Added attention mechanism to RNN
  - Proposed initial idea of attention
  - Model gave attention to particular hidden states when decoding





Challenges with Long-Term Dependencies

- NLP Challenge: Long-term dependencies
  - Language refers to items far in past
- Conventional architectures: RNN, LSTM, …
  - Iteratively update state on each word / token
  - "Forget" about items in past
  - Struggled with language
- Need to consider a long-term context

Example Problem

When the **scientists** submitted their paper to the journal, they were hopeful. The **reviewers**, however, found several flaws in the methodology. After weeks of revisions and back-and-forth communication, the paper was finally accepted. But when the journal published it, **they** noticed that the **acknowledgments section** was missing, which caused quite a stir among the **authors**.

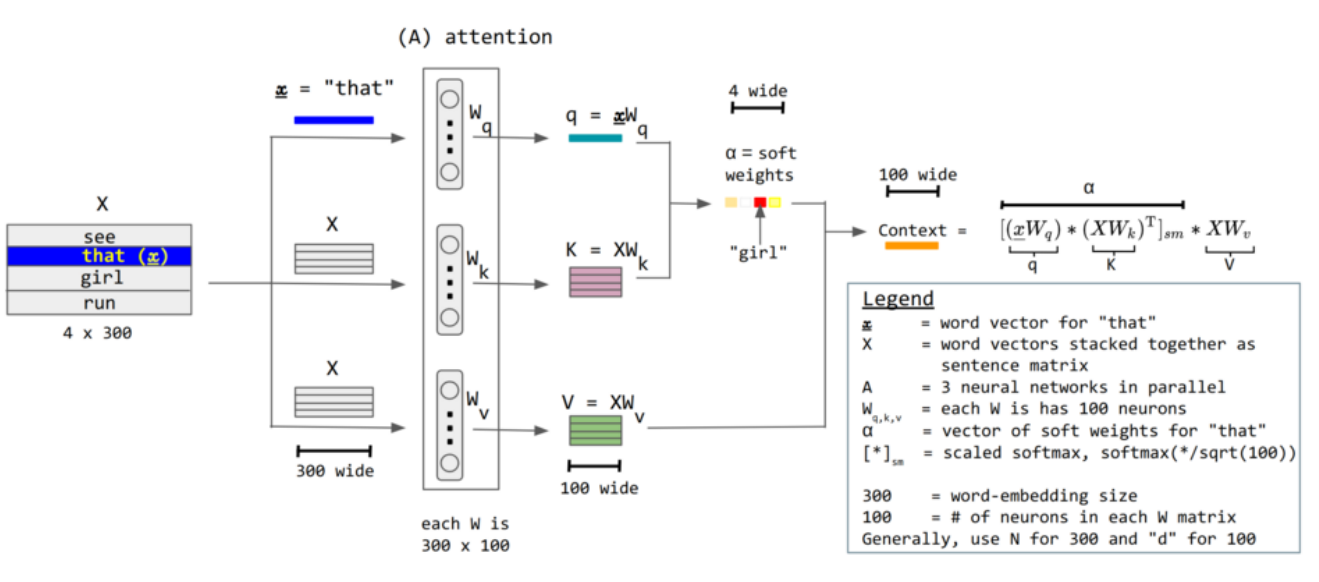
Question:

Who does **"they"** refer to?

Attention

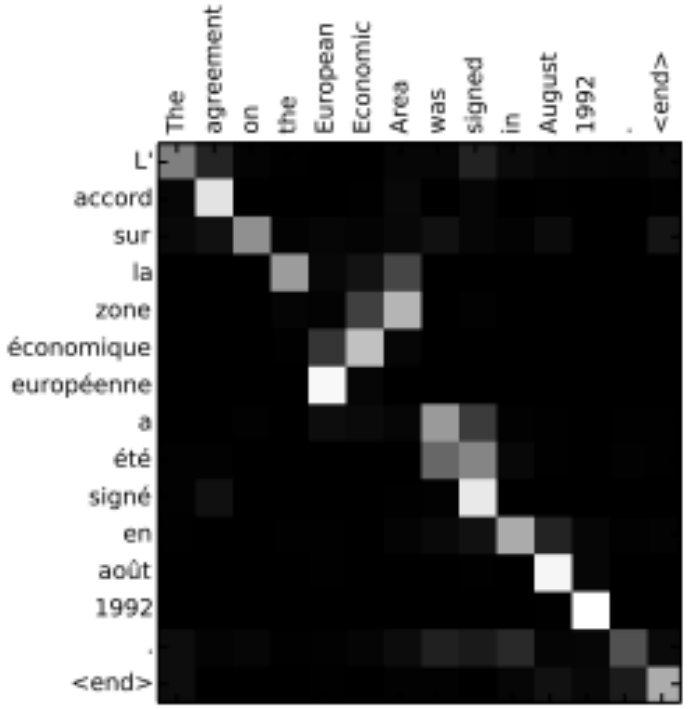
- Takes two contexts of tokens
  - Query and key
  - Each L ≈1024 length
- Finds similarities $L \times L$ matrix
- Similarities look up L values

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



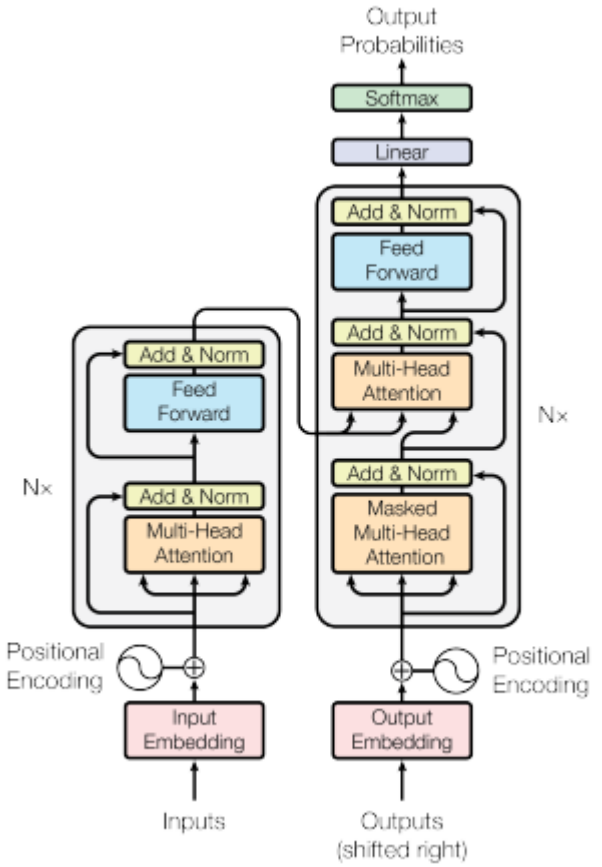
Attention Illustrated

- Find similarity matrix between tokens
- Example to the right:
  - Translation from French to English
- Can find long-term relations easily



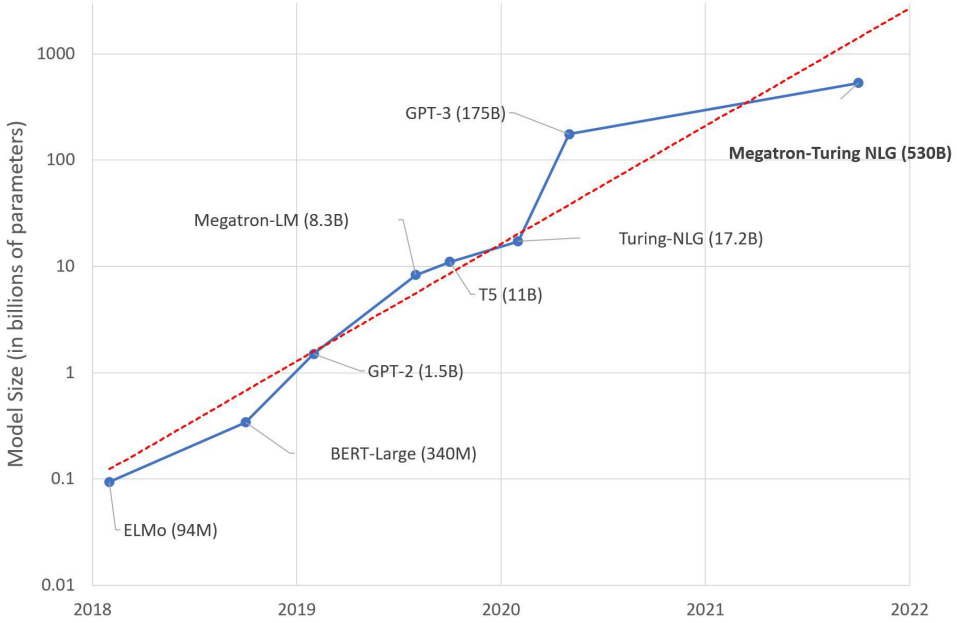
Full Architecture

- Each layer consists of:
  - Multi-headed attention
  - Add and normalization
  - Feedforward neural network
- Repeated multiple attention layers
  - Skip connections between layers



Model Sizes

- Model are massive
- But more efficient architectures:
  - Retrieval-augmented-Generation (RAG)
  - Multi-Agent systems
  - Tool use and external APIs



Model	Estimated Parameters
GPT-4	~1.76 trillion
Claude 3 Opus	~2 trillion
Gemini 1.5	~60–100 billion

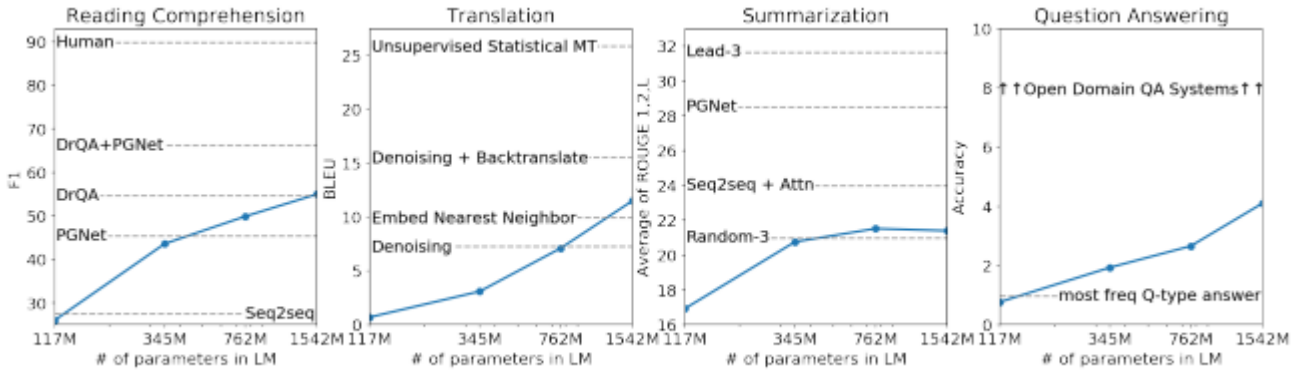
Variant	Layers (n_layer)	Embedding Dim (n_embd)	Context Size (n_ctx)	Parameters
GPT-2 (small)	12	768	1024	117M
GPT-2 Medium	24	1024	1024	345M
GPT-2 Large	36	1280	1024	762M
GPT-2 XL	48	1600	1024	1.5B

Unsupervised Multitask Training

- Landmark OpenAI Paper
- Train on diverse tasks
  - Change final layers for each task
- Training requires:
  - 10000+ GPUs
  - Weeks + $$$

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]



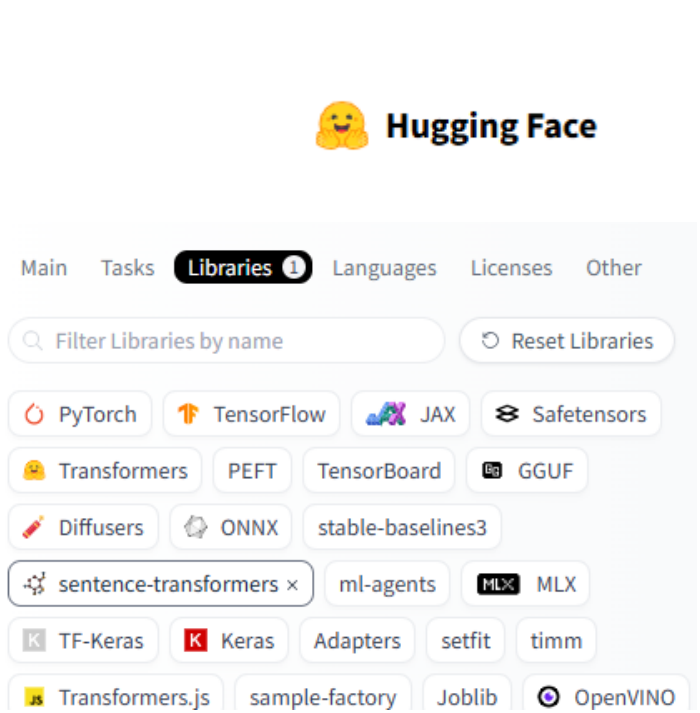
The Llama 3 Herd of Models

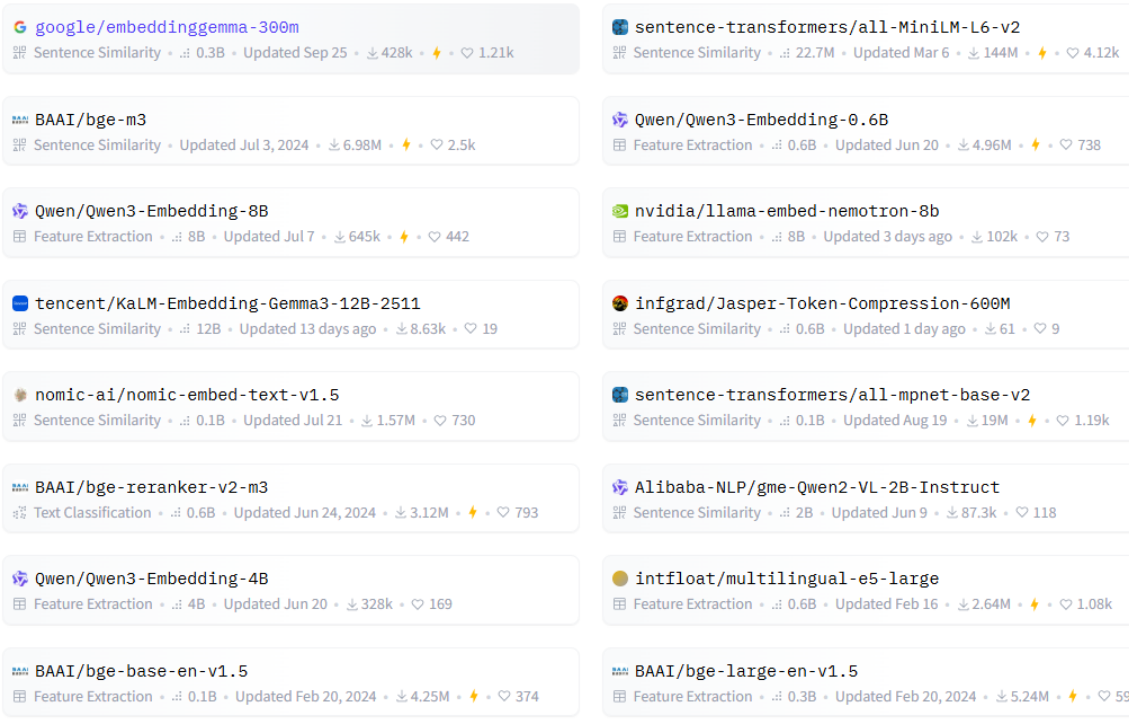
Llama Team, AI @ Meta[1]

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	4	131,072	16	16M	380	38%

Pre-Trained Models in Hugging Face

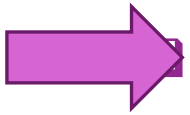
- Hugging Face
  - Has 1000s of pre-trained models
  - Sentence Transformers
- Easy-to-use
  - No training
  - Varying complexity





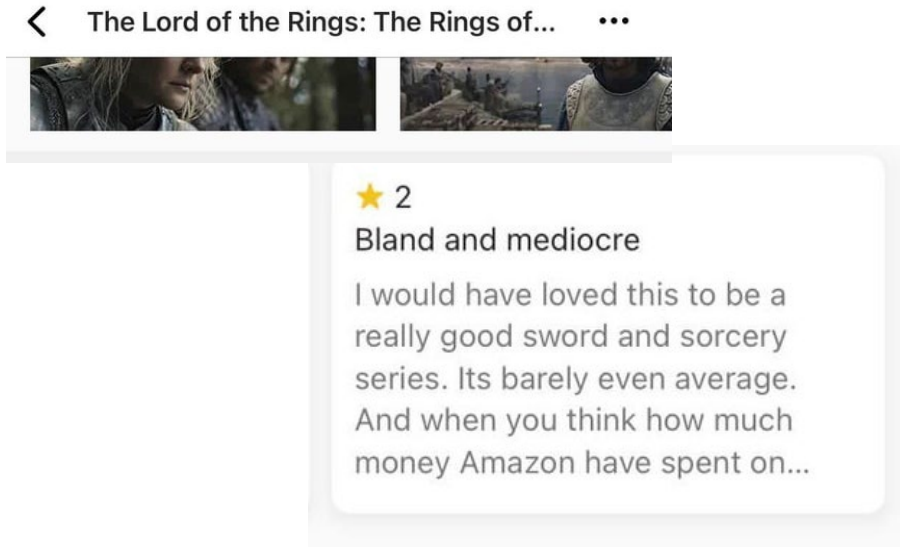
Outline

- ❑ Tokenization
- ❑ Modern LLMs at a Glance
- ❑ Using a pre-trained LLM



Movie Sentiment Analysis

- Problem: Given text for a movie detect sentiment?
  - Not given the numeric score
- Requires deep understanding of text



"Negative review"

IMDB Review Data

- Dataset from 2011 classic paper
- 50,000 reviews taken from IMBD database
- Available from Kaggle

Learning Word Vectors for Sentiment Analysis

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang,
Andrew Y. Ng, and **Christopher Potts**
Stanford University
Stanford, CA 94305
[amaas, rdaly, ptpham, yuze, ang, cgpotts]@stanford.edu

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("lakshmi25npathi/imdb-dataset-of-50k-movie-reviews")
```

Example Reviews

As a another reviewer states Hanna's War is an outstanding film about an outstanding person, Hanna "Anniko" Senesh, who would become the Jewish Joan Of Arc. Unfortunately I diverge in opinion not agreeing that Miss Detmers as the lead is too beautiful to be taken seriously as a resistance fighter. In truth for me her performance is not held back by her beauty but makes it all the more stark in the terror of the sadistic brutality as a resistor she faces. Maruschka Detmers performance is brave, poignant, heartfelt or understood, and totally believable. In other words for me "In the zone." from the opening credits. If you would like to learn about the suffering of someone else for something they believe in and be impressively entertained give Hanna's War with Maruschka Detmers a try. My hat is off also to Ellen Burstyn as Hanna's mother a much well known and famous actress who could have made effort to walk off with the film. In that it is a team effort perhaps of two actress' but not an All About Eve situation.

Example Positive Review

This film is so bad - dialogues, story, actors and actresses - everything! - that it's hard to imagine that we'll see a worse movie this year or in the following years. "Love's Brother" (set in Australia among Italian immigrants) has nothing but shallow clichés about Italian culture to offer, and it is quite telling that even the Italians from and in Italy speak ENGLISH in the film. The message of the film - ugly people have to marry ugly people, beautiful people have to marry beautiful people - is truly discomforting. Giovanni Ribisi is quite good in films like 'Suburbia' or 'Lost in Translation', but here his pseudo-Italian accent is hard to bear. See this film at your own risk. Trash as trash can!

Example Negative Review

Downloading a Pre-Trained LLM

- Use MiniLMV2:
  - From Microsoft 2021
  - Very compact, easy to use
- Download from Hugging Face

MINILMv2: Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers

Wenhui Wang Hangbo Bao Shaohan Huang Li Dong Furu Wei*
Microsoft Research
{wenwan,t-habao,shaohanh,lidong1,fuwei}@microsoft.com

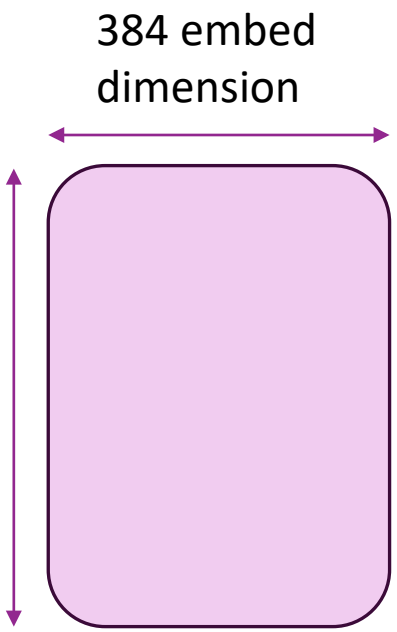
```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
model = model.to(device)
```

Encoding the Reviews

- Encode the reviews with the model
- On Google Colab free T4 GPU
  - About 2 mins for 50,000 reviews
  - With CPU it is too long
- Each embedding is dimension 384
- Output a 50000 × 384 matrix

50000 reviews




```
review_embeddings = model.encode(reviews, convert_to_tensor=True, show_progress_bar=True)
print(f"Shape of review embeddings: {review_embeddings.shape}")
print(f"First review embedding (partial):\n{review_embeddings[0][:5]}")
```

Training a Classifier

- Used a simple neural network
  - Input = 384 (embedding dimension)
  - 50 hidden units
  - Binary output (positive or negative)

```
class SentimentClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SentimentClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.sigmoid(out)
        return out

# Determine input size from the embeddings
input_size = review_embeddings.shape[1]
hidden_size = 200
num_classes = 1 # For binary classification
```

Classification Results

- Not the best, but it works