Tokenization Contd. / Masked LMs

CSE 5525: Foundations of Speech and Natural Language Processing

https://shocheen.github.io/courses/cse-5525-spring-2026

THE OHIO STATE UNIVERSITY

Logistics

- Hw2 questions / comments?

- Final project questions?
  - Everyone has tinker credits? – please use your buckeyemail to create the account.

Last Class Recap: *Subword* Tokenization

- "Word"-level: issues with unknown words and information sharing, and gets complex fast
  - Also, fits poorly to some languages

- Character-level: long sequences, the model needs to do a lot of heavy lifting in learning good representations

- Subword tokenization – a middle ground
  - Byte Pair Encoding or BPE

Subword tokenizer – Byte-Pair Encoding

- A bottom-up tokenizer
  - Start with bytes / characters and keep merging until you reach a limit
  - A variant: WordPiece which uses longest prefix instead of merge rules to tokenize.

- Next up, Unigram-LM tokenizer
  - Takes a top-down approach

Unigram LM Tokenizer

1. Start with a **large base vocabulary**, **remove tokens** until a desired size is reached.

1. **How to construct a base vocabulary**: all substrings of pre-tokenized words

1. **How to remove tokens**:
   a. Compute **the unigram LM loss** over the corpus (more details later)
   b. Removing tokens increases this loss.
   c. Select and remove tokens that increase it the least.
   d. Repeat

Base Vocabulary

- The corpus:

  ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

- Initial Vocabulary (all strict substrings)

  ("h", 15) ("u", 36) ("g", 20) ("hu", 15) ("ug", 20) ("p", 17) ("pu", 17) ("n", 16)

  ("un", 16) ("b", 4) ("bu", 4) ("s", 5) ("hug", 15) ("gs", 5) ("ugs", 5)

Unigram LM loss

- Unigram LM loss = negative log **probability of the corpus**.

- **Probability of a corpus** = product of marginal **probability of individual words**

$$p(\text{pug hugs bugs}) = p(\text{pug}) \times p(\text{hugs}) \times p(\text{bugs})$$

Probability of a word

- product of marginal probability of its subwords (based on frequency)

  p(pug) = p(“p”) x p(“u”) x p(“g”)

  Or, p(pug) = p(“pu”) x p(“g”)

  Or, p(pug) = p(“p”) x p(“ug”)

  Choose highest of all possible splits

- How to this efficiently: Dynamic programming (Viterbi algorithm)
  - For each position, compute the subwords with the best scores ending in that position (more details in the reading)

Unigram Tokenization Algorithm

1. Start with a base vocabulary

2. Compute the unigram loss, L, over the corpus

3. For every token, w, in the vocabulary

VERY SLOW!

   a. Remove w from the vocabulary and recompute the loss, L'(w)
   b. Define score(w) = L'(w) - L

4. Compute w* = min_w score(w).
   a. Remove w* from the vocabulary.
   b. Got to step 2. Repeat until a desired vocabulary size is reached.

Unigram Tokenization Algorithm (Slightly Faster)

1. Start with a base vocabulary

2. Compute the unigram loss, L, over the corpus

3. For every token, w, in the vocabulary
    a. Remove w from the vocabulary and recompute the loss, L'(w)
    b. Define score(w) = L'(w) - L

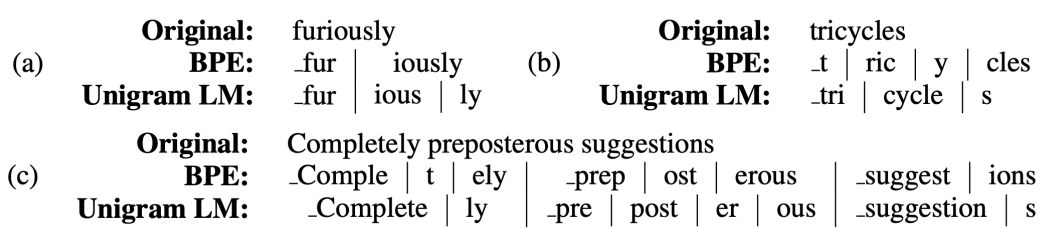
4. Compute **W** = x% of tokens with the lowest score.
    a. Remove w* from the vocabulary.
    b. Got to step 2. Repeat until a desired vocabulary size is reached.

How to tokenize once the vocabulary is decided

- Tokenization which maximizes the unigram probability of the word
  - Consider all possible segmentations, pick the one with higher probability of the word.
  - Speed up with dynamic programming (Viterbi algorithm).

- Gives you a ranked list of tokenizations:
  - Can make use of top *k* tokenizations. Randomly choose one of them for each instance of the word in the corpus.
  - Can make the model robust to spelling mistakes at inference.

Unigram vs BPE



1. Why unigram over BPE and WordPiece?
   a. Unigram finds **optimal coding length** of a sequence (according to Shannon's entropy).
   b. Unigram allows sampling multiple tokenizations for every word – subword regularization – robustness

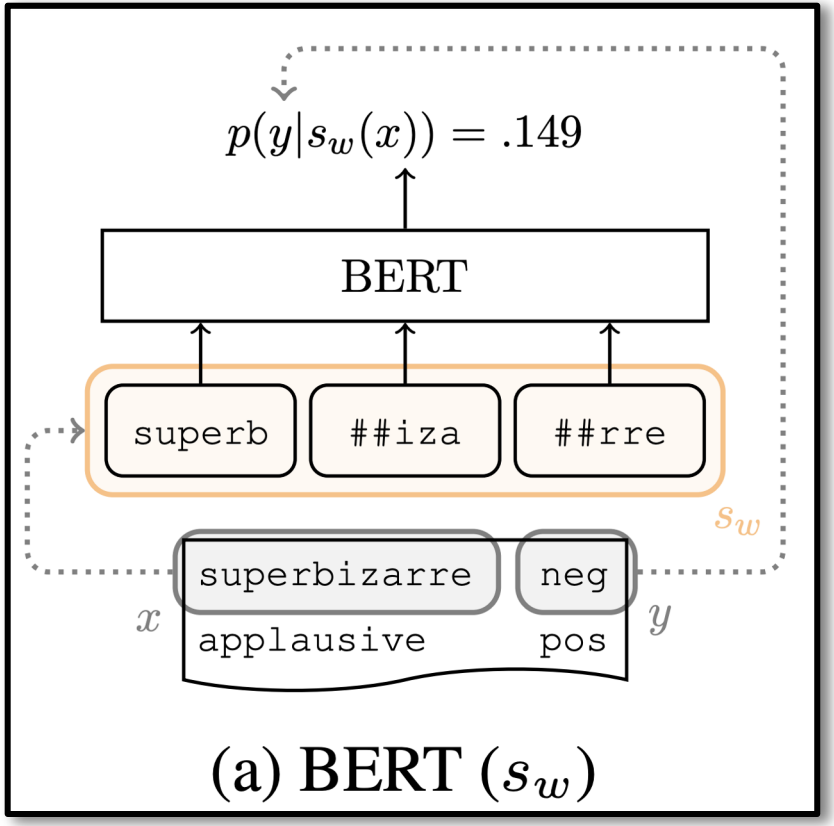
1. Unigram tokenization is not as popular as BPE
   a. Slower than BPE to train
   b. In many papers simply referred to as SentencePiece, which is actually not a tokenizer but a library wrapping several tokenizers
   c. Does the actual subword tokenizer matter as we scale up models?

Subword Models -- Summary

- Split text in tokens learned statistically from the training corpus
- Makes the language models “open” vocabulary
- Subword methods prove to be an effective method of “compressing text”

Issues with subword models

BERT (a text encoder; we will discuss this next) thinks the sentiment of "superbizarre" is positive because its tokenization contains the token "superb"



[Hofmann et al., 2021]

Subword Tokenization and “noise”

- He fell and broke his **coccix** (vs coccyx)
- Neighbor = 1 token, neighbour = two tokens
- John Smith = 2 tokens, Srinivas Ramanujam = ??

Non-concatenative Languages

كتب	k-t-b	"write" (root form)
كَتَبَ	**k**a**t**a**b**a	"he wrote"
كَتَّبَ	**k**a**tt**a**b**a	"he made (someone) write"
اِكْتَتَبَ	i**k**ta**t**a**b**a	"he signed up"

Table 1: Non-concatenative morphology in Arabic.[4] The root contains only consonants; when conjugating, vowels, and sometimes consonants, are interleaved with the root. The root is not separable from its inflection via any contiguous split.

Subword Tokenization and “numbers”

- Why are LMs bad at basic arithmetic?
  - 3.14 is tokenized as 3.14
  - 3.15 is tokenized as 3 . 15

Recent tweaks to subword tokenizers

- Tokenize each digit separately (i.e. no merge on digits)

- Add special tokens to deal with everything else:
  - E.g. special tokens for keywords from programming languages

- Train the tokenizer on a more balanced dataset.
  - Apply tricks like “alpha-sampling” – up sample lower resource languages and scripts to up their frequency.
  - Does not solve all issues but helps.

Masked Language Models

Reminder: Different senses, but same embedding 👎

- *The Amazon **Basin** is home to the largest rainforest on Earth.*
- *She filled the **basin** with water to wash the dishes.*
- *The neurosurgeon examined the cranial **basin** for signs of trauma.*

word2vec, GloVe, or other **static embeddings** assign exactly the same embedding to each of these occurrences of the word "basin" despite their different senses

Our next goal:
Assign different, **contextualized embeddings** based on the surrounding context

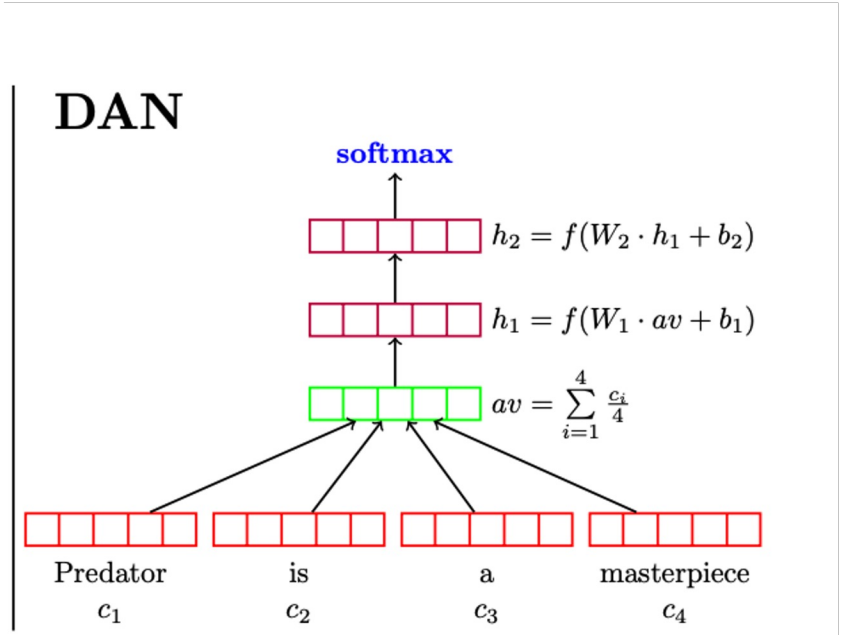
The final transformer encoder representation of each token is highly contextualized

Reminder: Token order & long-range dependencies with DAN 👎

Transformer considers token order with positional embeddings

Due to multi-headed self-attention & cross-attention, we are able to model long-range dependencies

Deep Averaging Network (DAN)



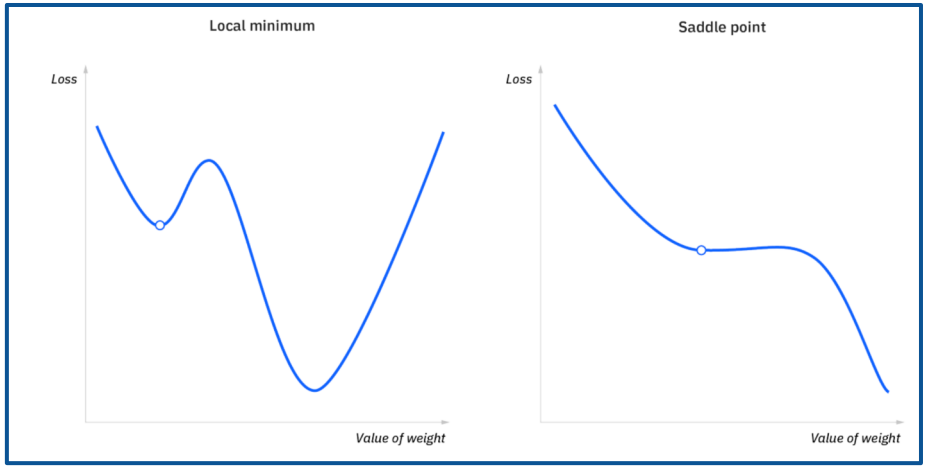
[Iyyer et al., 2015]

Reminder: Non-convex optimization, so initialization matters

Instead of starting from randomly initialized weights, we are going to make use of **large corpora** to **pretrain a model** and find weights that are a much better **starting point for all NLP tasks** than random weights

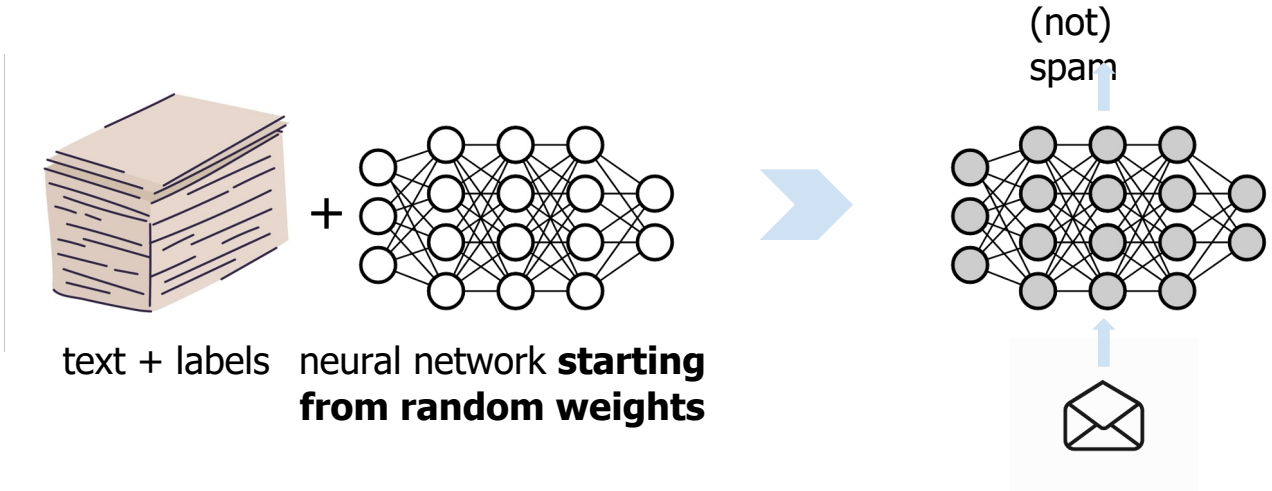
We are going to talk about this next!

Randomly initialized transformer still has this issue



Source: https://www.ibm.com/topics/gradient-descent

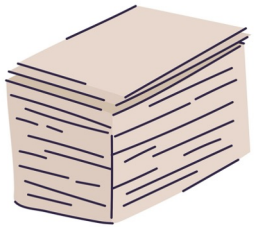
Standard Supervised Deep Learning



Pretrain-then-Finetune (also called post-train)

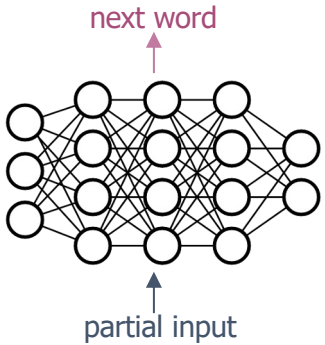
A form of transfer learning

Stage 1:
Pretrain a model



text

+



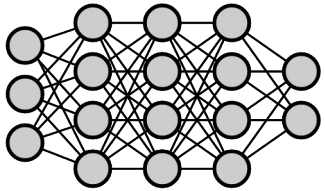
neural network **starting from <u>random</u> weights**

Objective: self-supervision
(does not require that people label the next word)

Stage 2:
Finetune the model

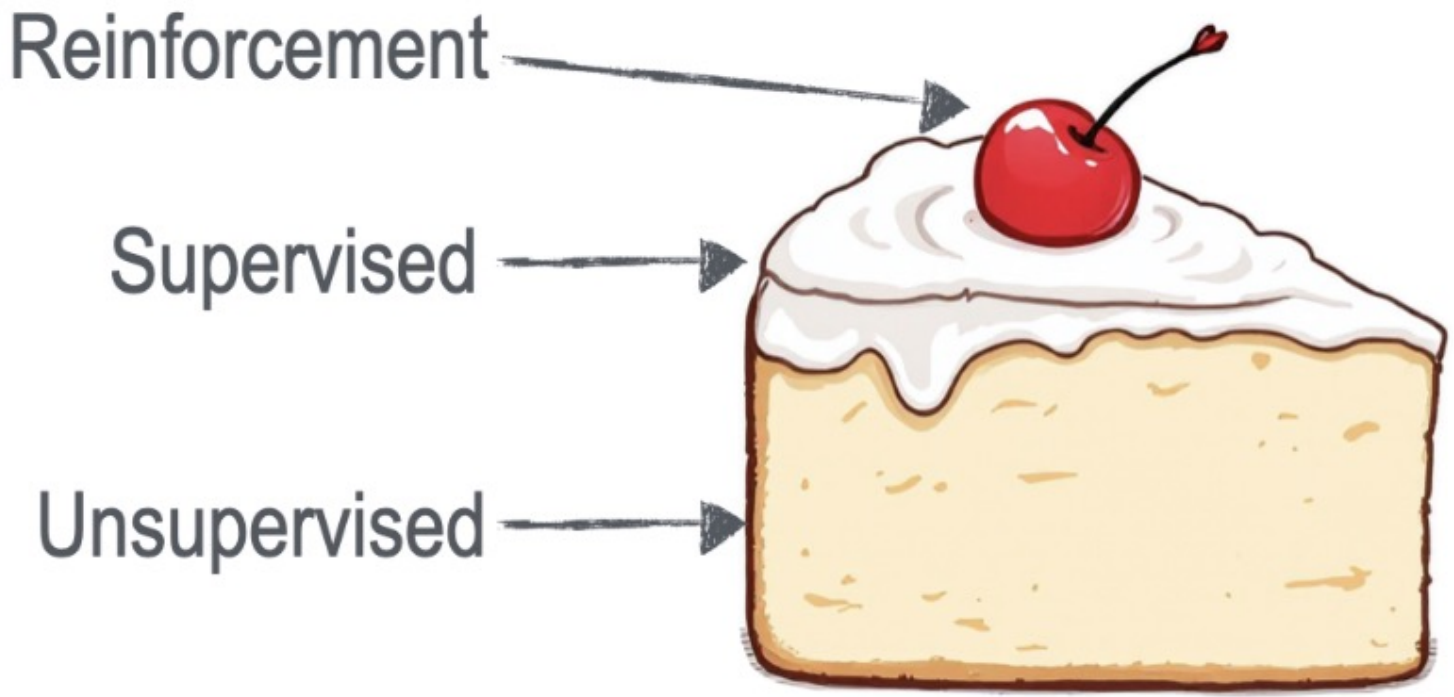
text + **labels**

+



neural network **starting from <u>pretrained</u> weights**

Objective: standard supervised training



arxiv.org/pdf/2502.03038

LM pretraining

- LMs so far: predict the next token given the previous tokens also sometimes called “Causal Language Modeling”.

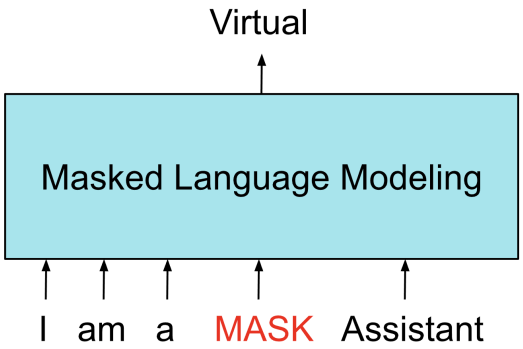
- This enables a self-supervised task – train on only raw text (similar to word2vec).

- And get really interesting and useful representations.
  - With many usecases

Masked LMs

- LMs so far: predict the next token given the previous tokens
- Since we have a complete sentence?
  - So: can we incorporate future context to learn better representations
- How can we formulate a self-supervised prediction task?

Masked LMs



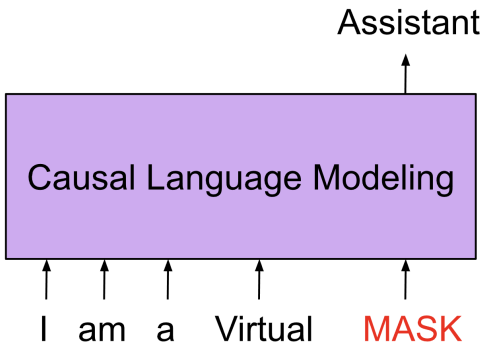


Image from https://www.holisticai.com/blog/from-transformer-architecture-to-prompt-engineering

Masked Language Modeling (MLM)

(text) Sylvester Stallone has made some **terrible** films in his lifetime, but this has got to be one of the **worst**. A totally **dull** story...

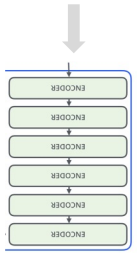
(masked text) Sylvester Stallone has made some **<mask>** films in his lifetime, but this has got to be one of the **<mask>**. A totally **<mask>** story...

(output final layer) vector(Sylvester) .. vector(**<mask>**) vector(films) vector(in) vector(his) ... vector(the) vector(**<mask>**) ... vector(**<mask>**) vector(story)...

Masked Language Modeling (MLM)

(text) Sylvester Stallone has made some **terrible** films in his lifetime, but this has got to be one of the **worst**. A totally **dull** story...

(masked text) Sylvester Stallone has made some **<mask>** films in his lifetime, but this has got to be one of the **<mask>**. A totally **<mask>** story...



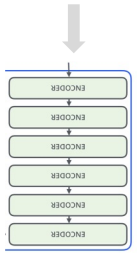
(output final layer) vector(Sylvester) .. vector(**<mask>**) vector(films) vector(in) vector(his) ... vector(the) vector(**<mask>**) ... vector(**<mask>**) vector(story)...

loss(predicted_word,“terrible”)

Masked Language Modeling (MLM)

(text) Sylvester Stallone has made some **terrible** films in his lifetime, but this has got to be one of the **worst**. A totally **dull** story...

(masked text) Sylvester Stallone has made some **<mask>** films in his lifetime, but this has got to be one of the **<mask>**. A totally **<mask>** story...



(output final layer) vector(Sylvester) ... vector(<mask>) vector(films) vector(in) vector(his) ... vector(the)

vector(<mask>) ... vector(<mask>) vector(story)...

loss(predicted_word,"worst")

loss(predicted_word,"dull")

loss(predicted_word,"terrible")

average these losses and do a gradient descent step

Masked LM: Only using Encoder transformer

- Encoders assume we have the complete sequence
- Self-attention computes weighted sum over entire context (i.e., entire sequence)
- There is no generation problem, we just want representations
  - We will learn how to use them later on

BERT

Bidirectional Encoder Representations from Transformers

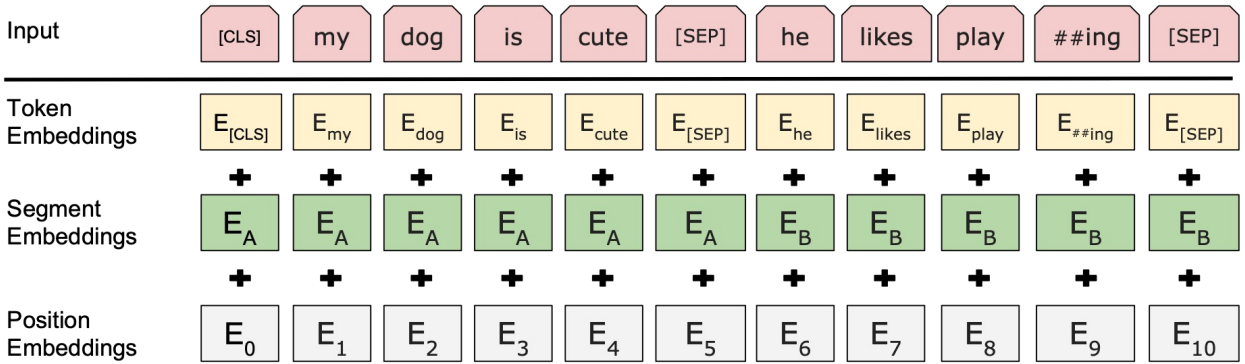
- Encoder transformer
- BERT Base: 12 transformer blocks, 768-dim word-piece tokens, 12 self-attention heads → 110M parameters
- BERT Large: 24 transformer blocks, 1024-dim word-piece tokens, 16 self-attention heads → 340M parameters
- 100's of variants since BERT came out.

[Devlin et al. 2018]

BERT

Inputs

- One or two sentences
  - Word-piece token embeddings
  - Position and segment embeddings



[figure from Devlin et al. 2018]

BERT

Training

- Data: raw text
- Two objectives:
  - Masked LM
  - Next-sentence prediction
- Later development in “RoBERTa”:
  - More data, no next-sentence prediction, dynamic masking

BERT

Masking Recipe for Training

- Randomly mask and predict 15% of the tokens
  - For 80% (of these 15%) replace the input token with a special token [MASK]
  - For 10% replace with a random token from the vocabulary
  - For 10% keep the same (input and output are the same, trivial task)

BERT

Next-sentence Prediction

- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- Training data: 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk
- Predict whether the next chunk is the true next chunk
- Prediction is done on the [CLS] output representation

BERT

Related Techniques

- Central Word Prediction Objective (context2vec) [Melamud et al. 2016]
- Machine Translation Objective (CoVe) [McMann et al. 2017]
- Bi-directional Language Modeling Objective (ELMo) [Peters et al. 2018]
- Then BERT came …
- … and many more followed (the whole sesame street arrived)

BERT

What Do We Get?

- We can feed complete sentences to BERT
- For each token, we get a contextualized representation
  - Meaning: computed taking the other tokens in the sentence into account
- In contrast to word2vec representations that fixed and do not depend on context
- While word2vec vectors are forced to mix multiple senses, BERT can provide more instance-specific vectors

BERT

How Do We Use It?

- Widely supported by existing frameworks
  - E.g., Transformers library by Hugging Face
- We will soon see how to use it when working with annotated data
- Large BERT models quickly outperformed human performance on several NLP tasks
  - But what it meant beyond benchmarking was less clear
- Started an arms race towards bigger and bigger models, which quickly led to the LLMs of today

BERT

What It Is Not Great For?

- primarily used for discriminative tasks, cannot generate text
  - Can manipulate in weird ways to generate text but not the original purpose of this model
  - Generating from masked LMs requires specialized training -- keyword: masked diffusion LMs (MDLMs)

Finetuning a MLM-pretrained model

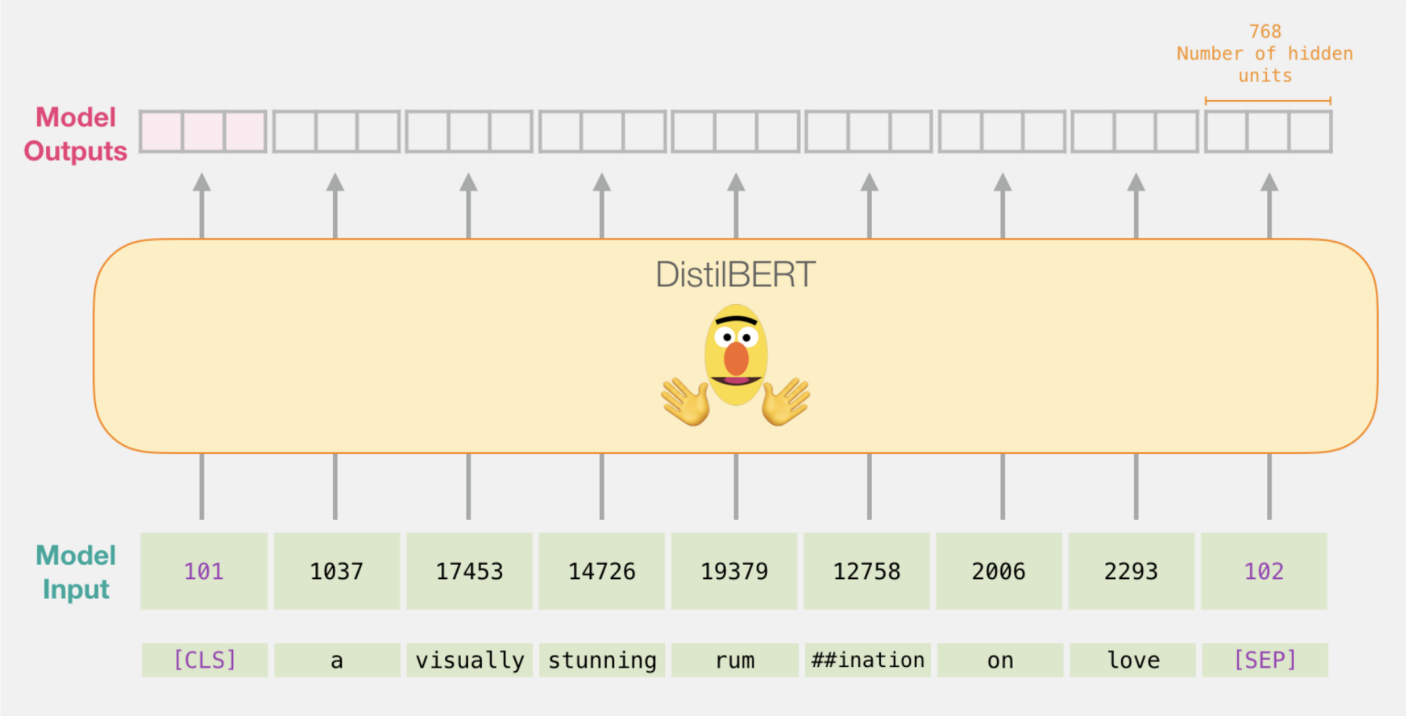


Figure: Jay Alammar

Finetuning a MLM-pretrained model

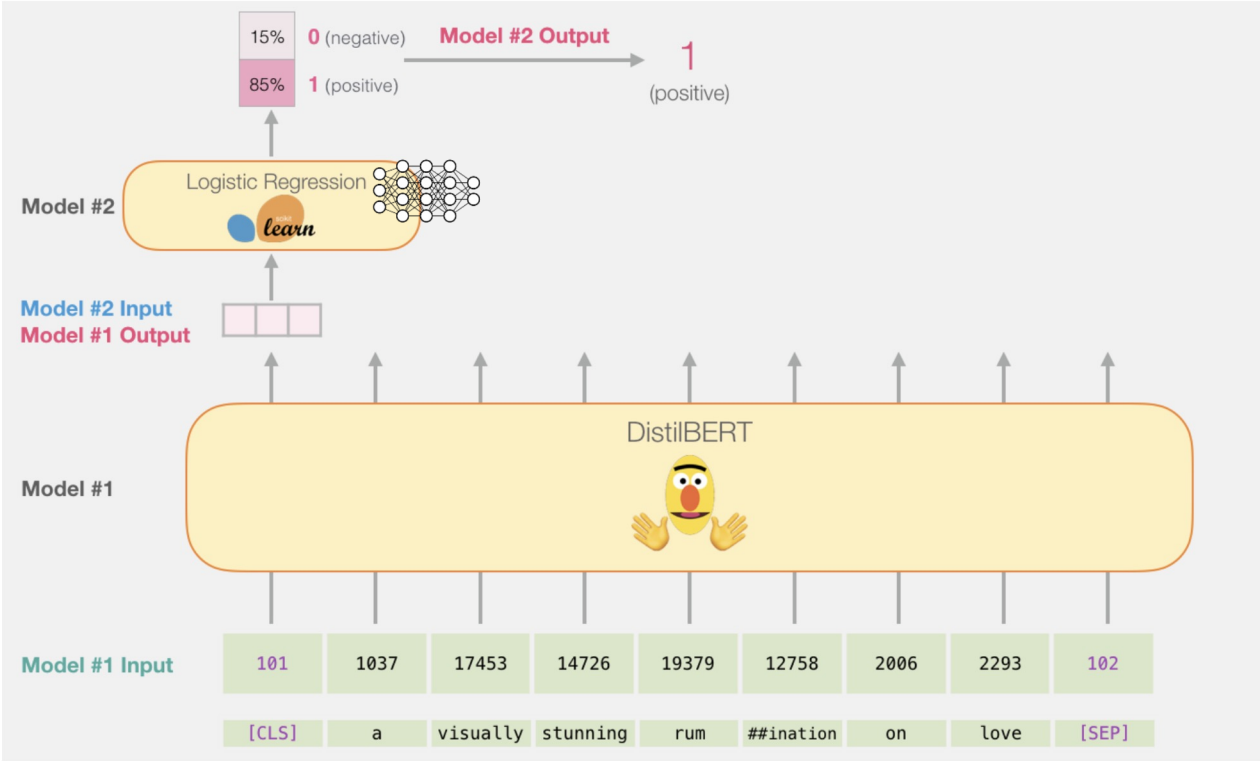


Figure: [Jay Alammar]

Output layer when finetuning a MLM-pretrained LM

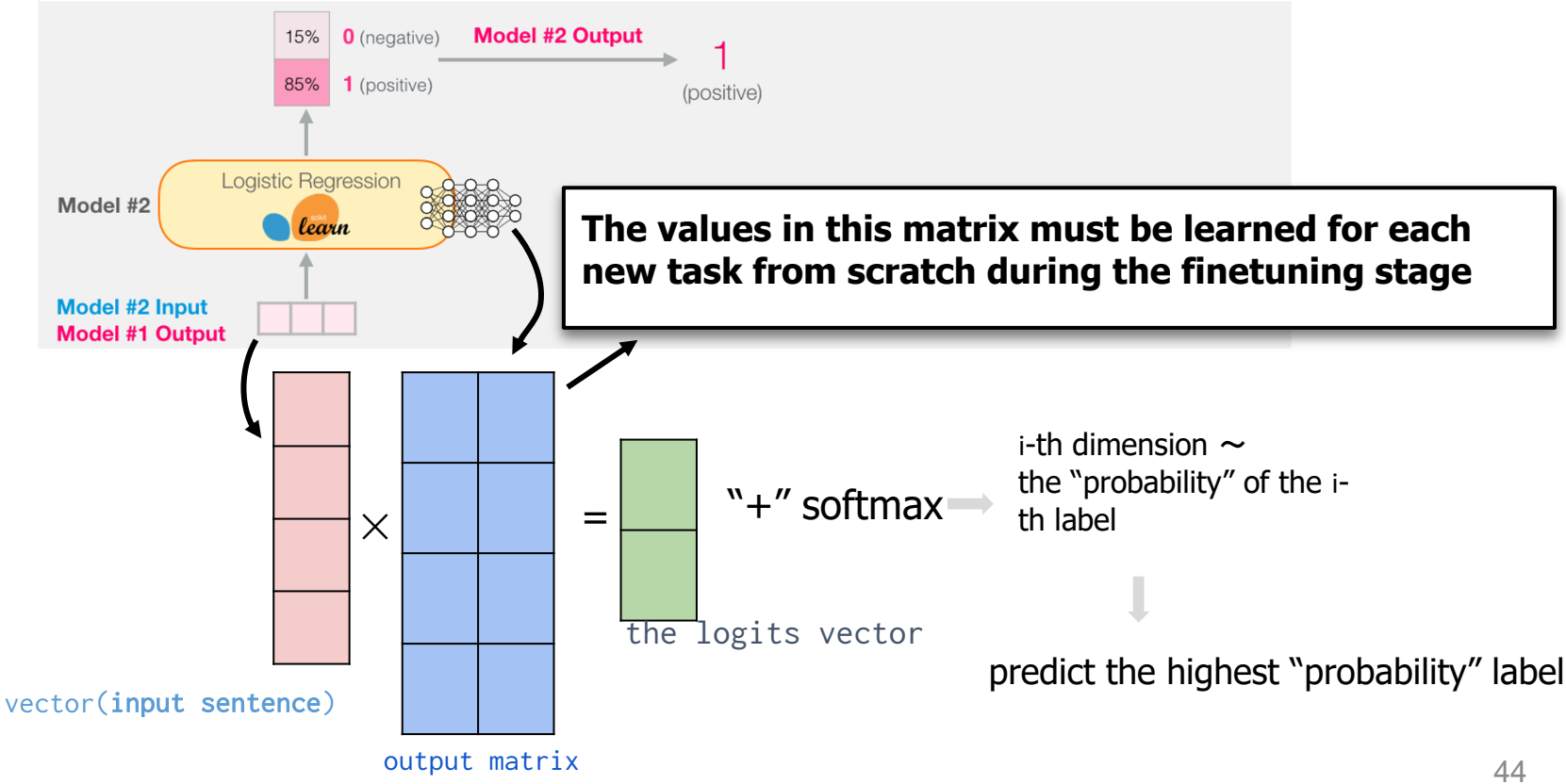


Figure: A Visual Guide to Using BERT for the First Time by Jay Alammar

Finetuning a MLM-pretrained model

- Add a special `[CLS]` token to the beginning of the sequence & `[SEP]` at the end of it
- Use the `d`-dimensional vector representation of the `[CLS]` token from the final layer to represent the entire sequence
  - Reminder: This makes sense because of the self-attention
- **Replace the pretrained output layer with a new output layer that has a new randomly initialized output weight matrix of the size `d x n_labels`**
- Multiply it with the `d x n_labels` output matrix ⇒ softmax ⇒ argmax = predicted label
- Cross entropy / NLL loss using the human-annotated labels

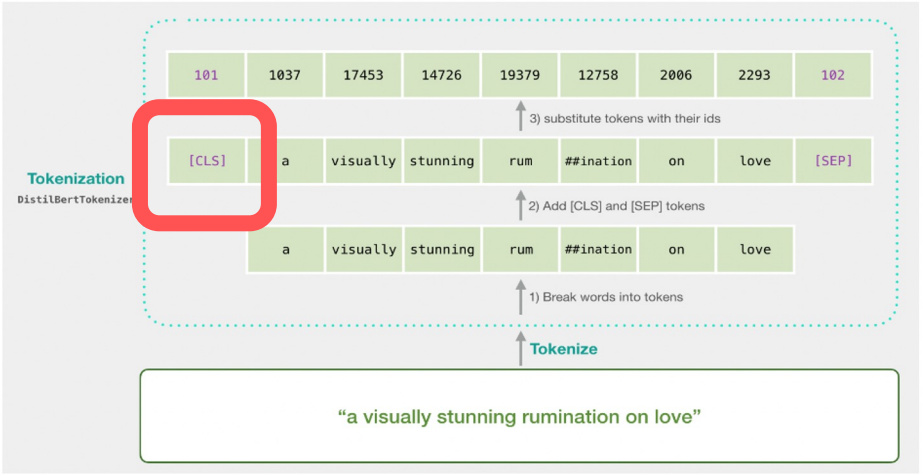
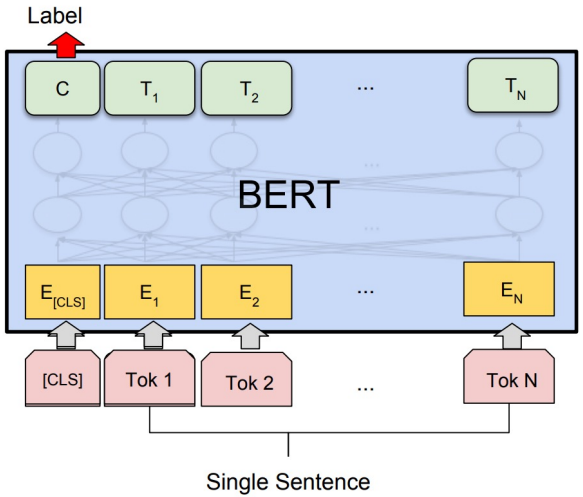
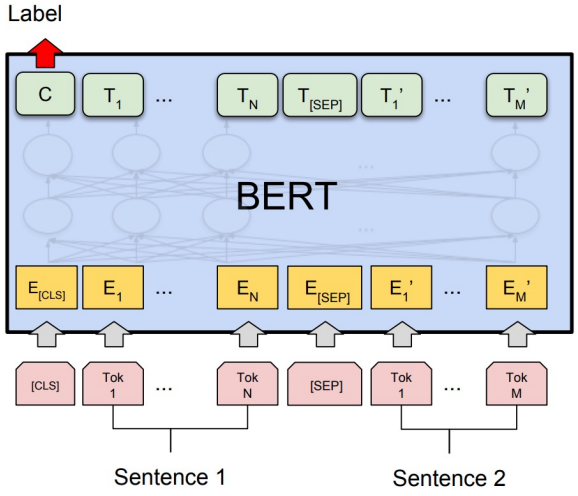


Figure: Jay Alammar

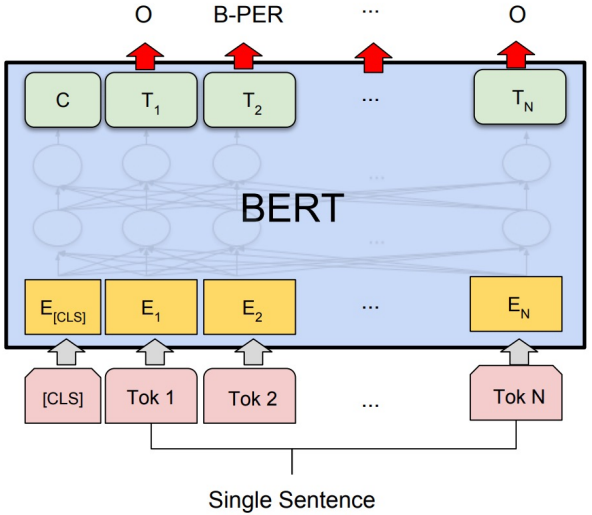
What can BERT do?



(b) Single Sentence Classification Tasks: SST-2, CoLA



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

- Artificial [CLS] token is used as the vector to do classification from
- Sentence pair tasks (entailment): feed both sentences into BERT
- BERT can also do tagging by predicting tags at each word piece

Devlin et al. (2019)

Reading Comprehension with BERT

Q: What was Marie Curie the first female recipient of?

Passage: One of the most famous people born in Warsaw was Marie Skłodowska-Curie, who achieved international recognition for her research on radioactivity and was the first female recipient of the Nobel Prize. Famous musicians include Władysław Szpilman and Frédéric Chopin. Though Chopin was born in the village of Żelazowa Wola, about 60 km (37 mi) from Warsaw, he moved to the city with his family when he was seven months old. Casimir Pulaski, a Polish general and hero of the American Revolutionary War, was born here in 1745.

Answer = Nobel Prize

- Assume we know a passage that contains the answer. Answer prediction is a classification task over the tokens.

SQuAD

Q: What was Marie Curie the first female recipient of?

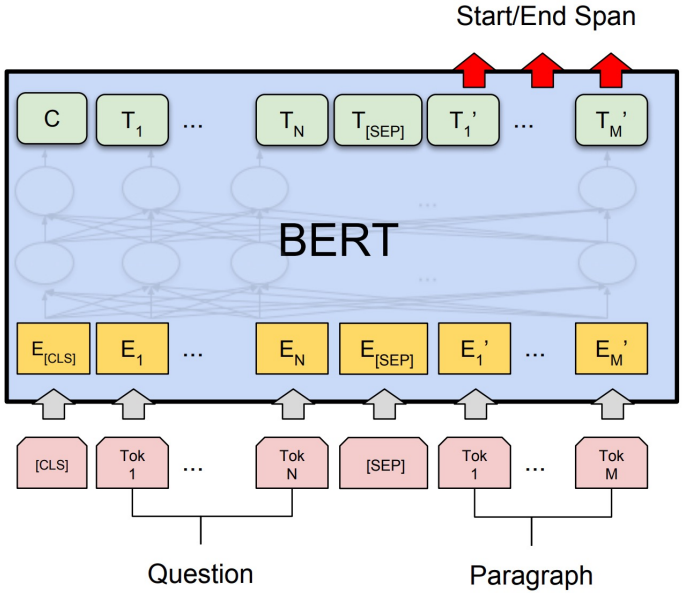
Passage: One of the most famous people born in Warsaw was Marie Skłodowska-Curie, who achieved international recognition for her research on radioactivity and was the first female recipient of the Nobel Prize. …

- Predict answer as a pair of (start, end) indices given question q and passage p; compute a score for each word and softmax those

P(start | q, p) =

P(end | q, p) = same computation but different params

QA with BERT



What was Marie Curie the first female recipient of ? [SEP] One of the most famous people born in Warsaw was Marie …

Devlin et al. (2019)

BERT results, BERT variants

Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

- Huge improvements over prior work
- Effective at "sentence pair" tasks: textual entailment (does sentence A imply sentence B), paraphrase detection

Devlin et al. (2018)

RoBERTa

- "Robustly optimized BERT"
- 160GB of data instead of 16 GB
- Dynamic masking: standard BERT uses the same MASK scheme for every epoch, RoBERTa recomputes them
- New training + more data = better performance

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

Liu et al. (2019)

BERT / RoBERTa / DeBERTa

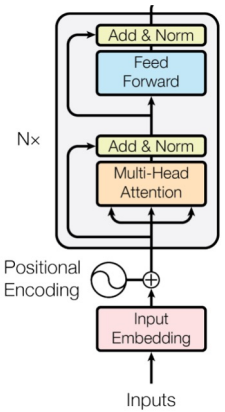
[Devlin et al., 2018 / Liu et al., 2019 / He et al., 2023]

- Encoder-only transformer
- Masked language modeling (MLM), ~~next sentence prediction~~
- 110M, 340M parameters

⚓ These models are a good option if you want to solve a text classification problem for which you have thousands of labeled datapoints & you know how to train a model (which you all will know after this course)

⚓ Don't work for generation as good as decoder-only or encoder-decoder models

`[*]` `[*]` **`[sat_]`** `[*]` **`[the_]`** `[*]`



`[The_]` `[cat_]` **`[MASK]`** `[on_]` **`[MASK]`** `[mat_]`

Figure by: Lucas Beyer

Latest Variants

NeoBERT: A Next-Generation BERT

Lola Le Breton[1,2,3] **Quentin Fournier**[2] **John X. Morris**[4] **Mariam El Mezouar**[5]
Sarath Chandar[1,2,3,6]

[1]Chandar Research Lab [2]Mila – Quebec AI Institute [3]Polytechnique Montréal
[4]Cornell University [5]Royal Military College of Canada [6]Canada CIFAR AI Chair

Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Benjamin Warner[1†] **Antoine Chaffin**[2†] **Benjamin Clavié**[1†]
Orion Weller[3] **Oskar Hallström**[2] **Said Taghadouini**[2]
Alexis Gallagher[1] **Raja Biswas**[1] **Faisal Ladhak**[4*] **Tom Aarsen**[5]
Nathan Cooper[1] **Griffin Adams**[1] **Jeremy Howard**[1] **Iacopo Poli**[2]

[1]Answer.AI [2]LightOn [3]Johns Hopkins University [4]NVIDIA [5]HuggingFace

*†: core authors, *: work done while at Answer.AI*

Correspondence: {bw,bc}@answer.ai, antoine.chaffin@lighton.ai

Using BERT

- HuggingFace Transformers: big open-source library with most pre-trained architectures implemented, weights available
- Lots of standard models… and "community models"

Model architectures

🤗 Transformers currently provides the following NLU/NLG architectures:

1. **BERT** (from Google) released with the paper BERT: Pre-training of Deep Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Krist
2. **GPT** (from OpenAI) released with the paper Improving Language Under Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever.
3. **GPT-2** (from OpenAI) released with the paper Language Models are Un Jeffrey Wu*, Rewon Child, David Luan, Dario Amodei** and Ilya Sutskev
4. **Transformer-XL** (from Google/CMU) released with the paper Transform Fixed-Length Context by Zihang Dai*, Zhilin Yang*, Yiming Yang, Jaime
5. **XLNet** (from Google/CMU) released with the paper XLNet: Generalized Understanding by Zhilin Yang*, Zihang Dai*, Yiming Yang, Jaime Carbon
6. **XLM** (from Facebook) released together with the paper Cross-lingual L and Alexis Conneau.
7. **RoBERTa** (from Facebook), released together with the paper a Robustly

…

`mrm8488/spanbert-large-finetuned-tacred`

`mrm8488/xlm-multi-finetuned-xquadv1`

`nlpaueb/bert-base-greek-uncased-v1`

`nlptown/bert-base-multilingual-uncased-sentiment`

`patrickvonplaten/reformer-crime-and-punish`

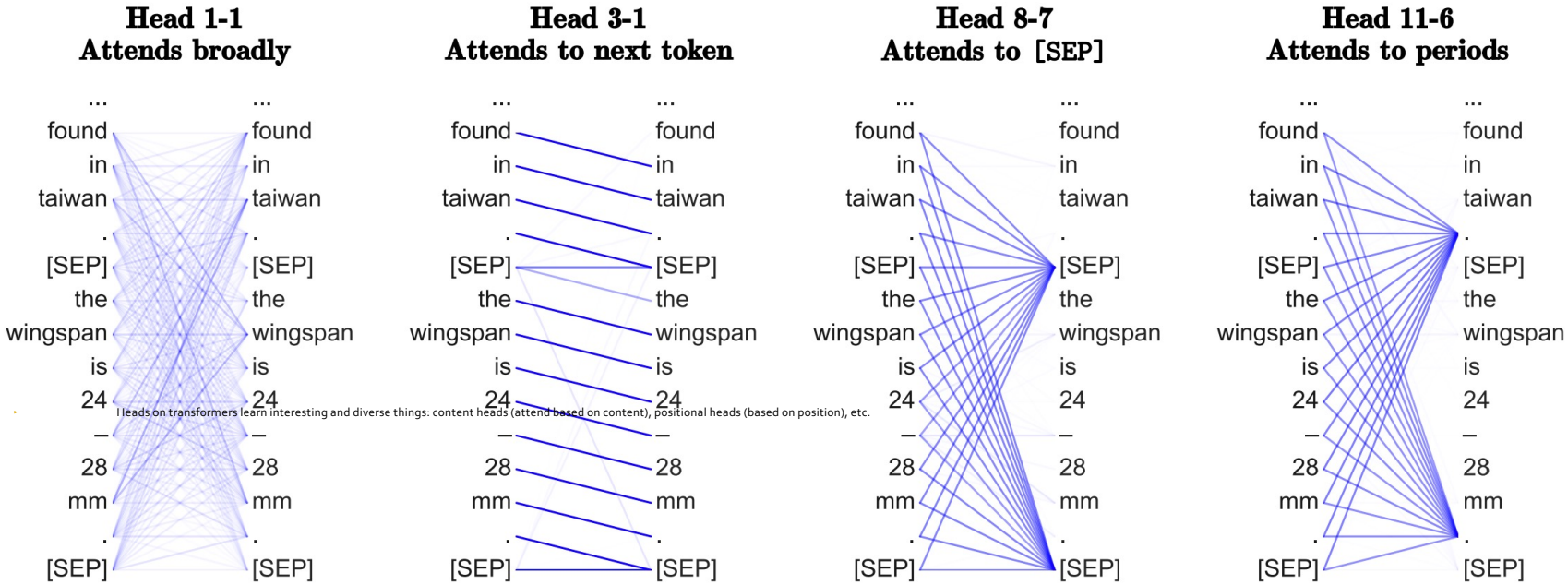
`redewiedergabe/bert-base-historical-german-rw-cased`

`roberta-base`

`severinsimmler/literary-german-bert`

`seyonec/ChemBERTa-zinc…base-v1`

What does BERT learn?



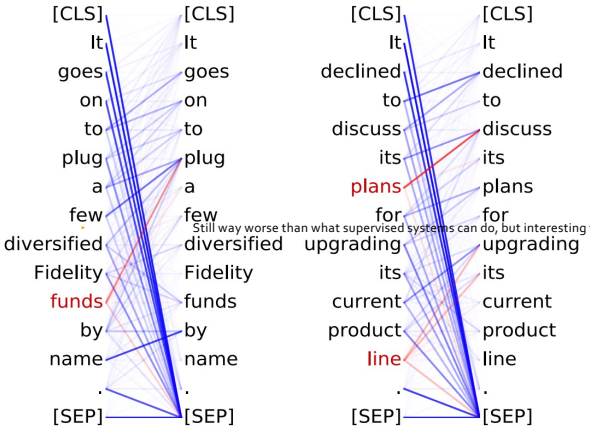
- Heads on transformers learn interesting and diverse things: content heads (attend based on content), positional heads (based on position), etc.

Clark et al. (2019)

What does BERT learn?

Head 8-10

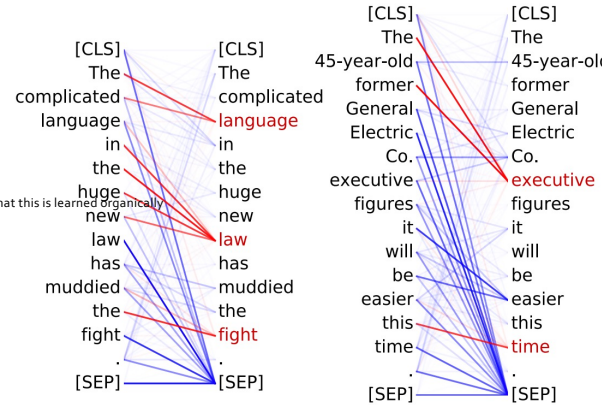
- **Direct objects** attend to their verbs
- 86.8% accuracy at the `dobj` relation



Still way worse than what supervised systems can do, but interesting that this is learned organically

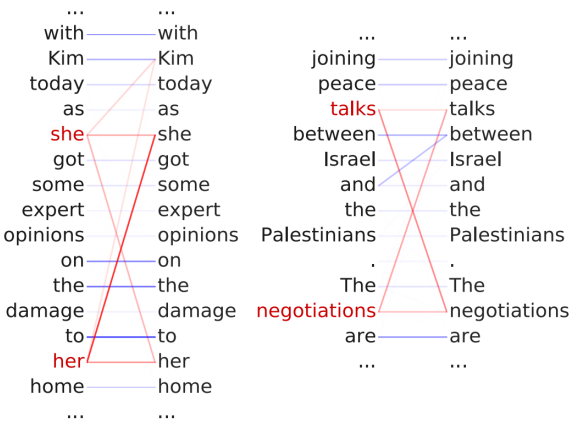
Head 8-11

- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the `det` relation

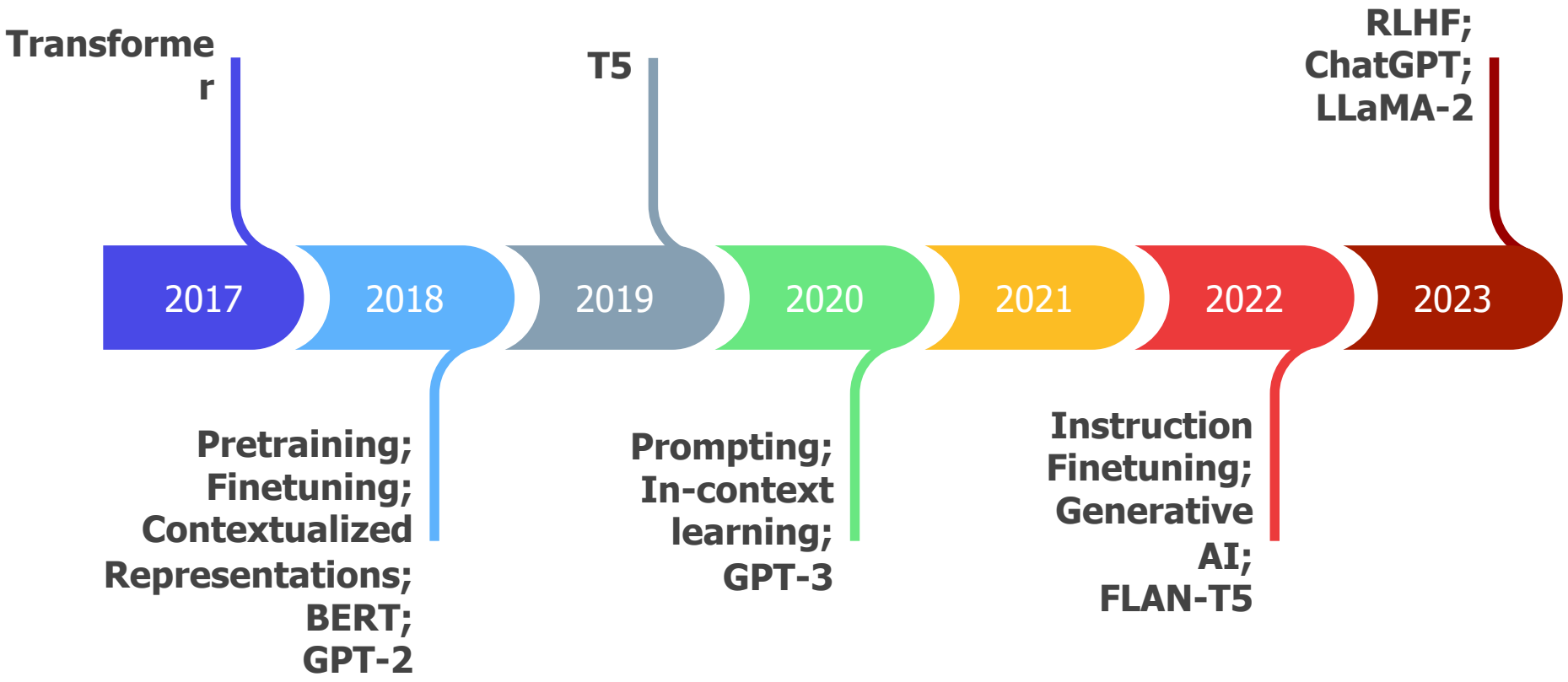


Head 5-4

- **Coreferent** mentions attend to their antecedents
- 65.1% accuracy at linking the head of a coreferent mention to the head of an antecedent



Clark et al. (2019)


Transforme
r
T5
RLHF;
ChatGPT;
LLaMA-2
2017
2018
2019
2020
2021
2022
2023
Pretraining;
Finetuning;
Contextualized
Representations;
BERT;
GPT-2
Prompting;
In-context
learning;
GPT-3
Instruction
Finetuning;
Generative
AI;
FLAN-T5