NLP basics

Learning Text Representations

MVA - Speech and Language Processing #6 (NLP 2)

Paul Michel & Benoît Sagot

Outline

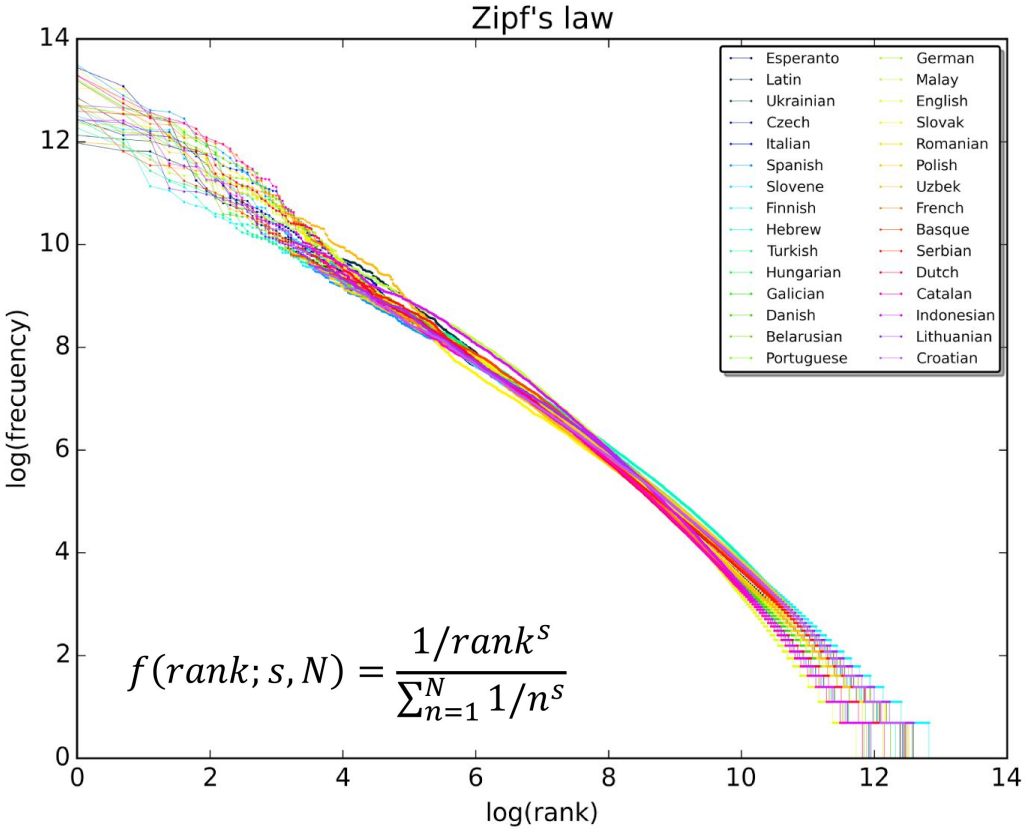
- Non-contextual word embeddings
- Contextual word embeddings from pre-trained language models
- Transfer learning with pre-trained language models
- Sentence embeddings
- Beyond representation learning

Credit and disclaimer: some of the following slides are taken from, illustrated or inspired by presentations and article figures by Jurafsky, Goldberg, Melamud, Mazaré and others.

Lexical sparsity

A plot of the rank versus frequency for the first 10 million words in 30 Wikipedias

(source: Wikipedia; data: dumps from Oct 2015)



Representing words: lexicons and thesauruses

- **Advantages**
  - Possibility to encode rich linguistic information and to cover rare cases not seen in corpora
  - It is another source of linguistic information, next to annotated corpora
- **Drawbacks**
  - Costly to develop, do not exist for all languages
  - Static, fixed meanings and words
  - Limited coverage
  - Structural organisation not always relevant (it is difficult to create a hierarchy of meanings for adjectives and verbs)

Vector representations: what for?

- Question answering:
  - Question: *How **tall** is Mt. Everest?*
    Candidate answer: *The official **height** of Mount Everest is 8848m*
- Plagiarism detection:
  - *Mainframes **are primarily** referred to as **large** computers with **rapid**, advanced processing capabilities that **can execute and** perform tasks **equivalent to many** Personal Computers (PCs)*
  - *Mainframes **usually are** referred to as large computers with **fast**, advanced processing capabilities that **could** execute and perform **by itself** tasks **that may require a lot of** Personal Computers (PCs)*
- We need to be able to capture **word similarity** in our representations
  - Thanks to its context of occurrences in a corpus
  - … but ideally, with different representations for different occurrences, to fully accommodate the fact that a same word (token) can have multiple meanings depending on the context

Distributional models of meaning

- Example (modified by Lin (1998) from (Nida, 1975, page 167)): Suppose I asked you what is tesgüino?

  *A bottle of **tesgüino** is on the table*
  *Everybody likes **tesgüino***
  ***Tesgüino** makes you drunk*
  *We make **tesgüino** out of corn.*

- From context words humans can guess tesgüino means *an alcoholic beverage akin to beer*
- Intuition: **Two words are similar if they have similar word contexts.**
  - Zellig Harris (1954): “oculist and eye-doctor … occur in almost the same environments”
    “If A and B have almost identical environments we say that they are synonyms.”
  - Firth (1957): “You shall know a word by the company it keeps!”
  - **Distributional hypothesis**

Different kinds of vector models

- **Sparse word vector representations**
  - Mutual-information weighted word co-occurrence matrices
- **Dense word vector representations, aka non-contextual word embeddings**
  - Brown clusters
  - Singular value decomposition (and Latent Semantic Analysis)
  - Neural-network-based representations (skip-grams, CBOW)
- **Dense word occurrence vector representations, aka contextual word embeddings**
  - Neural language model-based representations
- **Shared intuition**
  - The meaning of a word is modelled by “embedding” the word in a vector space
  - A meaning is represented as a vector
    - The meaning can be lexical (averaged over all occurrences, as in a dictionary) or contextual

Co-occurrence vectors

Cooccurrence matrices

- **Cooccurrence** = appearing in the same environment
  - document
  - immediate context
- **Cooccurrence matrix** = frequency counts of *(word, environment)* pairs
- Two main categories of cooccurrence matrices
  - **Term-document matrix**: how often word *i* occurs in document *j*
  - **Term-term (word-word, word-context) matrix**: how often word *i* occurs in the immediate vicinity of word *j*

Term-document matrix

- **Each cell: count of word $w \in V$ in a document $d \in D$**

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Term-document matrix

- **Each cell: count of word $w \in V$ in a document $d \in D$**
  - Each document is a count vector in $\mathbb{N}^{|V|}$ (a column)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Term-document matrix

- **Each cell: count of word $w \in V$ in a document $d \in D$**
  - Each document is a count vector in $\mathbb{N}^{|V|}$ (a column)
  - Each word is a count vector in $\mathbb{N}^{|D|}$ (a row)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Similarity in term-document matrices

- Two documents are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Similarity in term-document matrices

- Two documents are similar if their vectors are similar
- Two words are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Similarity in word-word matrices

- Instead of entire documents, use **smaller contexts** (Paragraph, window of ± 4 words)
- A word is now defined by a vector over **counts of context words**
- Instead of each vector being of length $|D|$
  - each vector is now of length $|V|$, the word-word matrix is $|V|^2$
- Example with a 7-word context

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…	…
apricot	0	0	0	1	0	1		
pineapple	0	0	0	1	0	1		
digital	0	2	1	0	1	0		
information	0	1	6	0	4	0		

Word-word matrix

- We showed only 4x6, but the real matrix is 500,000 x 500,000
  - **Very sparse** (most values are 0)
  - That's acceptable, since there are lots of efficient algorithms for sparse matrices.
- Similarity is measured using the **cosine** between two vectors (i.e. the **dot product** between normalised vectors)
- The **size of context windows** depends on your goals
  - The **shorter** the windows, the more **syntactic** the representation
    1-3 ~ syntactic similarity
  - The **longer** the windows, the more **semantic** the representation
    4-10 ~ semantic/topical similarity

Positive Pointwise Mutual Information

Problems with raw counts

- Raw word frequency is not a great measure of association between words
  - It is very skewed
    - For ex.: “the” and “of” are very frequent, but maybe not the most discriminative
- We would be more interested in a measure that would give more importance to context words that are **more informative** about the target word
  - Pointwise Mutual Information (PMI): do words *x* and *y* cooccur more than if they were independent?

$$\mathrm{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

(ranges from -∞ to +∞)

  - But this metrics has issues

Positive Pointwise Mutual information (PPMI)

- **Negative PMI values are problematic**
  - They correspond to words co-occurring **less** than we expect by chance
  - Unreliable without a really huge corpus
    - Imagine two rare words w_1 and w_2 (say, frequency = 10^{-6}): it is hard to compare $P(w_1,w_2)$ with 10^{-12} and know whether the difference is statistically significative…
  - It is unclear whether we have intuitions about unrelatedness
    - We are interested in similarity (relatedness)
- **We replace negative PMI values by 0 => PPMI**

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

PPMI on an example

Count(w,context)

	computer	data	pinch	result	sugar
apricot	0	0	1	0	1
pineapple	0	0	1	0	1
digital	2	1	0	1	0
information	1	6	0	4	0

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

Weighting PPMI

- **PPMI is biased toward infrequent events**
  - Very rare words have very high PMI values
- Two solutions
  - Transform counts using an exponential (exponent <1) to give rare context words slightly higher counts

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

  - Use “**Laplace smoothing**”, i.e. **add 1 to all counts** (it has a similar effect)
    - Or **add 2**, etc.

Beyond cosine similarity

- Another way to improve over cosine similarity on count matrices is to use **more sophisticated similarity measures**

$$\text{sim}_{\text{cosine}}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| \cdot |\vec{w}|}$$

$$\text{sim}_{\text{Jaccard}}(\vec{v}, \vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} \max(v_i, w_i)}$$

$$\text{sim}_{\text{Dice}}(\vec{v}, \vec{w}) = \frac{2 \sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} (v_i + w_i)}$$

- When working on word-document matrices, the most frequently used transformation is not PPMI but **tf-idf**

$$(\text{tf-idf}_D)_{ij} = (\text{tf}_D)_{ij} \cdot (\text{idf}_D)_{ij}$$

$(\text{tf}_D)_{ij}$ = "term frequency" = #occurrences of the word w_i in document d_j (can be normalised by the document length, binarised, log…)
$(\text{idf}_D)_{ij}$ = "inverse document frequency" = log ($|D|/ \text{df}_i$), where df_i is the number of documents containing w_i

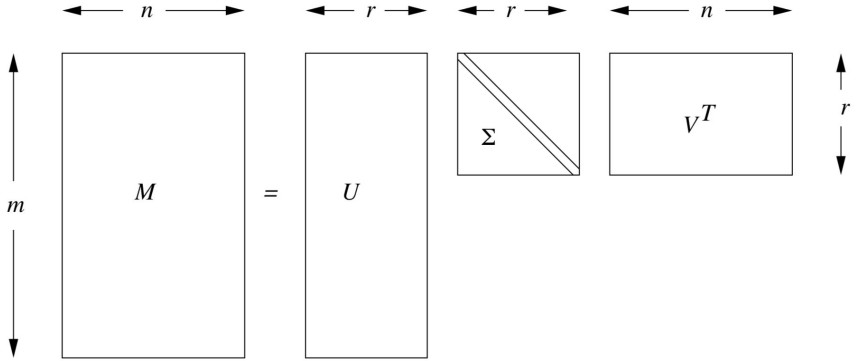
Building dense vectors using SVD

Sparse vs. dense vectors

- **PPMI vectors are sparse**
  - Up to 500,000 dimensions, if not more (depends on the language)
  - Most values are 0
  - Cosine distance not optimal on such vectors, no information is shared between similar words
  - Machine learning systems using such vectors have a large number of weights to train
- **How can we get dense, low-dimensionality vectors?**
  - Many techniques have been proposed. 2 examples:
    - Classical: singular value decomposition (SVD) of PPMI matrices
    - More recently: side-effect of predictive neural models

Singular Value Decomposition

- **Theorem**
  for any rectangular *m*×*n* real matrix M of rank *r*, there exist
  - an *m*×*r* orthogonal matrix U,
  - an *r*×*r* diagonal matrix Σ with diagonal values ≥ 0 (“singular values”),
  - and an *n*×*r* orthogonal matrix V
  - such that $M = U \Sigma V^T$
- If singular values are sorted in decreasing order (e.g. amount of variance captured by the corresponding dimension), then Σ is unique



Singular Value Decomposition: an example

	Matrix	Alien	Star Wars	Casablanca	Titanic
John	1	1	1	0	0
Jack	3	3	3	0	0
Jill	4	4	4	0	0
Jenny	5	5	5	0	0
Jane	0	0	0	4	4
Joe	0	0	0	5	5
Jim	0	0	0	2	2

Singular Value Decomposition: an example

	Matrix	Alien	Star Wars	Casablanca	Titanic
John	1	1	1	0	0
Jack	3	3	3	0	0
Jill	4	4	4	0	0
Jenny	5	5	5	0	0
Jane	0	0	0	4	4
Joe	0	0	0	5	5
Jim	0	0	0	2	2

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix} \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}$$

$$M \qquad U \qquad \Sigma \qquad V^{\mathrm{T}}$$

SVD applied to term-document matrices: Latent Semantic Analysis

- **We discard all latent dimensions apart from the first *k* ones** (e.g. *k*=300)
  - U is replaced by an *m*×*k* matrix U_k,
    Σ is replaced by a *k*×*k* diagonal matrix Σ_k with only the *k* first singular values,
    U is replaced by an *n*×*k* matrix V_k — and $\mathbf{M} \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^{\mathsf{T}}$
  - We get an optimal approximation of M (least-square)
- We get a *k*-dimensional vector for each word (an "**embedding**") by reading U_k's rows

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix} \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}$$

$$M \qquad U \qquad \Sigma \qquad V^{\mathrm{T}}$$

SVD applied to word-word matrices

- **Same technique**
  - Only difference: $m = n$
- Again, we only keep the top k dimensions
  - In fact, it might help to discard the first dimension(s) and keep the k following ones
- Again, we get a k-dimensional vector for each word (an "**embedding**") by reading U_k's rows

Does it work better than sparse vectors?

- **In short, it does**
  - Lower dimensions represent information that is not important, not necessarily significant: **denoising**
  - Removing lower dimensions results in **generalisations**, including capturing **higher-order cooccurrence**
  - Fewer dimensions = models easier to learn (**fewer weights**)

Building dense vectors using a neural network

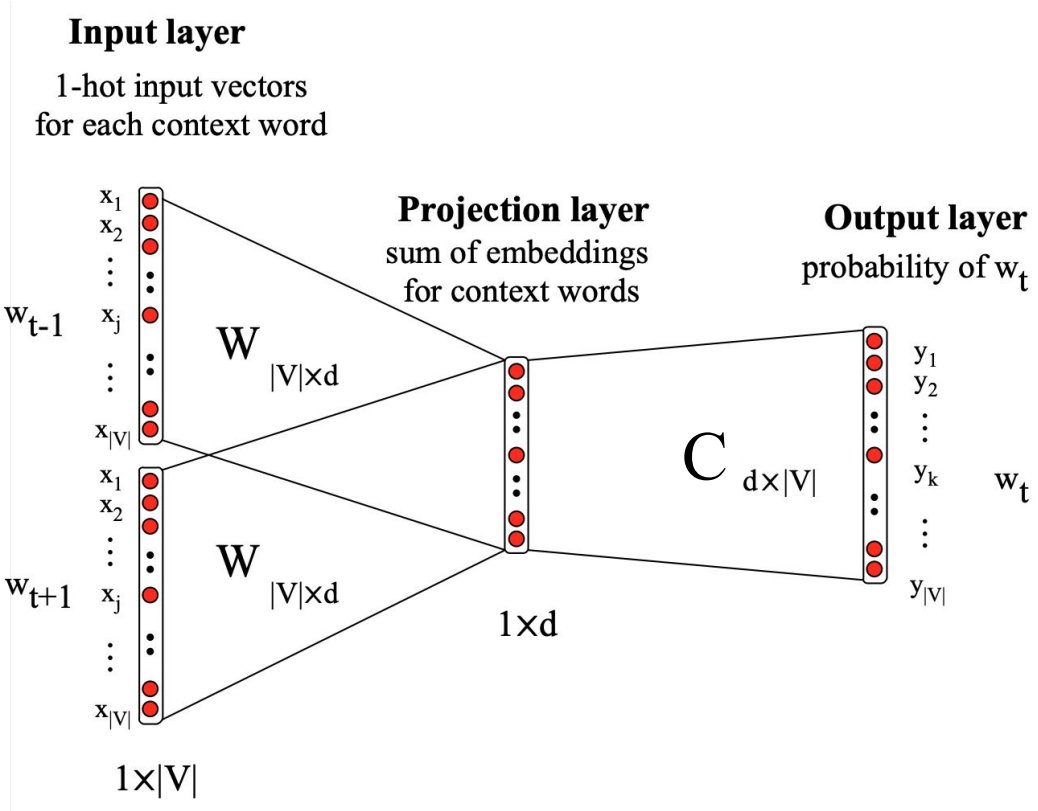
Prediction-based embeddings

- Different approach for implementing the same intuition
  - Underlying principle is still the distributional hypothesis
  - Instead of starting with **word counts to quantify the notion of distribution**, we will **learn to predict words based on distributional properties of their contexts**
  - We will do so **using a neural approach**
- Underlying idea: we will train a neural network to perform a given word-based task and extract **intermediate representations** as word representations (word embeddings)
- Approach popularised by the **word2vec** package (Mikolov et al. 2013a,b)
  - Easy and fast to train
  - Freely available, pre-trained embeddings

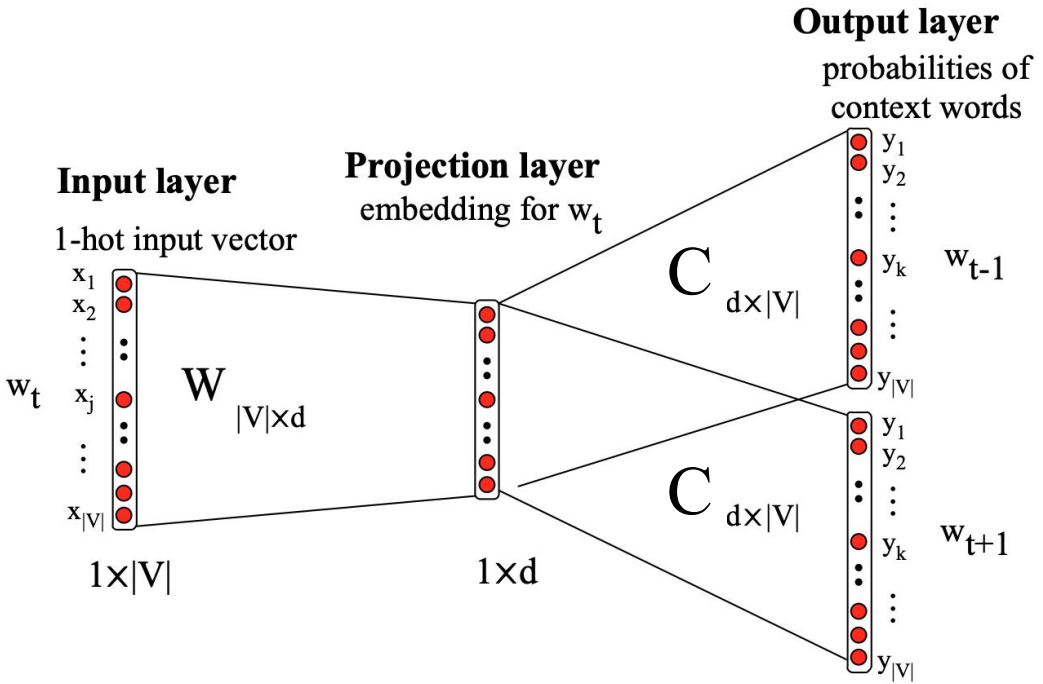
word2vec

- Neural network with one hidden layer > This hidden layer will provide the embeddings
- Input and output layers use **1-hot vector representations**
  - Word $w_i \in V$ is represented by a $|V|$-dimensional vector whose values are all 0 except for the i-th one which is 1
  - Contexts involving several words have 0 values except for those dimensions corresponding to these words
  - In the output layer, each value is interpreted as a probability via the application of the SoftMax operation
- Two types of predictions in word2vec:
  - Given a context surrounding a position, predict the word that should fill this position: **CBoW** (continuous bag of words)
  - Given a word, predict its neighbours: **skip-gram**

CBoW


Input layer
1-hot input vectors
for each context word
Projection layer
sum of embeddings
for context words
Output layer
probability of w_t
w_{t-1}
w_{t+1}
x_1 x_2 x_j $x_{|V|}$
$W_{|V|\times d}$
$C_{d\times|V|}$
y_1 y_2 y_k $y_{|V|}$
w_t
$1\times d$
$1\times|V|$

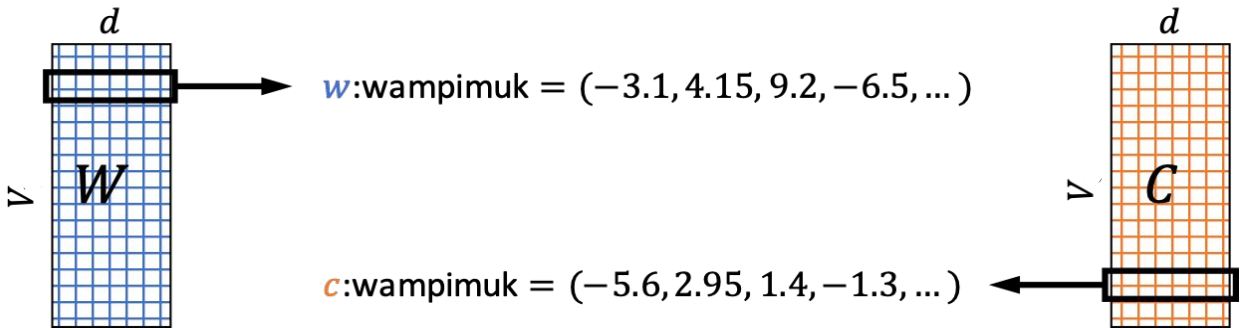
Skip-grams


Output layer
probabilities of context words
Input layer
1-hot input vector
Projection layer
embedding for w_t
w_t
x_1 x_2 x_j $x_{|V|}$
$W_{|V|\times d}$
$C_{d\times|V|}$
y_1 y_2 y_k $y_{|V|}$
w_{t-1}
$C_{d\times|V|}$
y_1 y_2 y_k $y_{|V|}$
w_{t+1}
$1\times|V|$
$1\times d$

Focus on skip-grams

- The skip-gram approach creates a vector for each word $w \in V$
  - Each such vector has d latent dimensions (e.g. d=100)
- **It learns a matrix W** whose rows represent each word $w \in V$
- It also learns **a similar auxiliary matrix C** of context vectors
  - In fact, each word has two embeddings

Example (Goldberg apud Baroni): *Marco saw a furry little wampimuk hiding in the tree.*



Negative sampling

Marco saw a furry little wampimuk hiding in the tree.

- Positive examples: (wampimuk, furry), (wampimuk, little), (wampimuk, hiding), etc.
- No negative examples can be directly extracted from the data
  - **Negative sampling** is used to artificially create negative examples
  - How? By replacing the context word in such pairs with randomly selected words
    - "Randomly selected" in the sense of the unigram distribution
    - For each positive example, *k* negative examples are built

- **Maximize:** $\sigma(\vec{w} \cdot \vec{c})$
  - c was **observed** with w

words	contexts
wampimuk	furry
wampimuk	little
wampimuk	hiding
wampimuk	in

- **Minimize:** $\sigma(\vec{w} \cdot \vec{c}')$
  - c' was **hallucinated** with w

words	contexts
wampimuk	Australia
wampimuk	cyber
wampimuk	the
wampimuk	1985

Goldberg *et al.* 2014

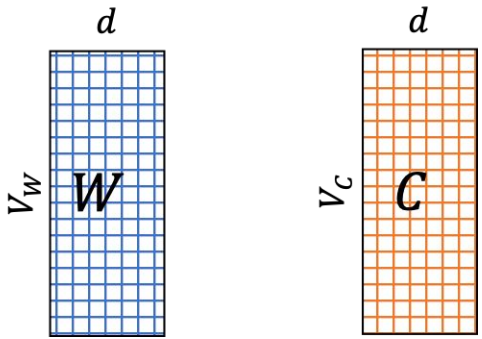
Skip-Gram with Negative Sampling (SGNS)

- Word embeddings created by the skip-gram model using negative sampling is the classical state-of-the-art for non-contextual embeddings

- word2vec's SGNS outperforms count-based (PPMI) vectors when used in virtually all NLP tasks
  - or does it?
  - Goldberg and colleagues have investigated what makes word2vec's SGNS so much better…

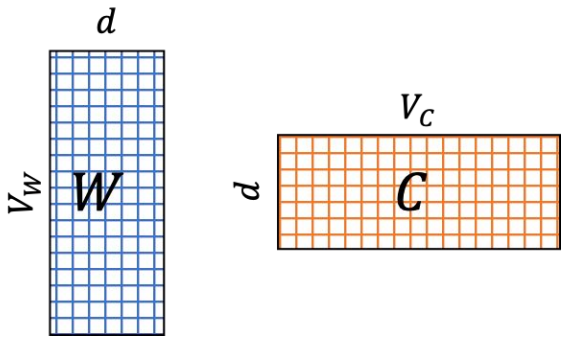
What is SGNS learning?

- Take both SGNS embedding matrices,



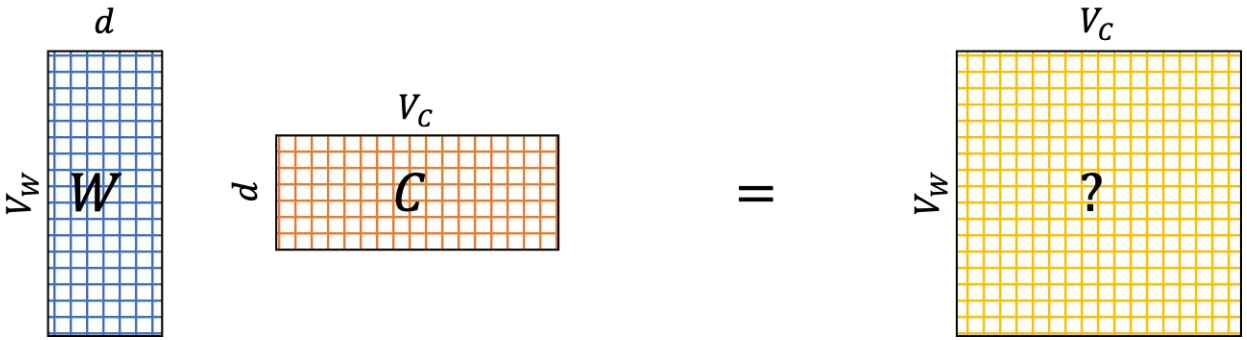
What is SGNS learning?

- Take both SGNS embedding matrices, multiply them



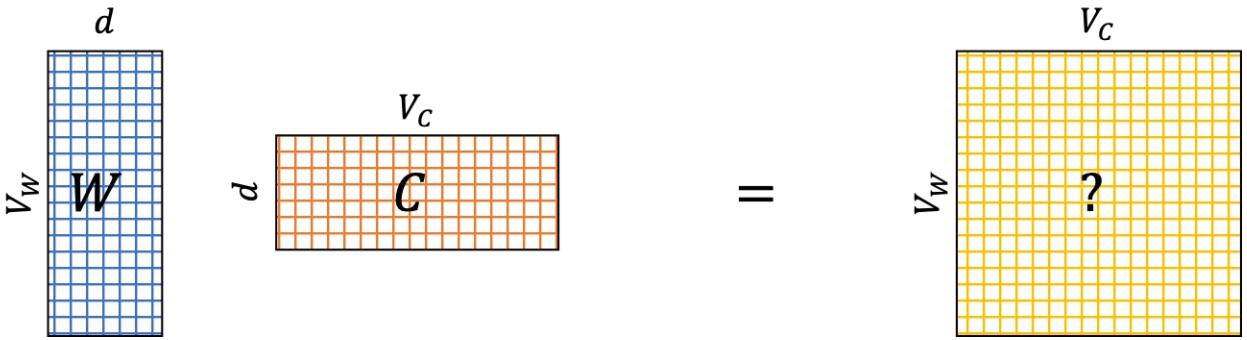
What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word



What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word
- It turns out that when *d* is large and after enough training iterations, **you get the PMI matrix**…



What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word
- It turns out that when *d* is large and after enough training iterations, **you get the PMI matrix**… except for a constant:

$$W \cdot C^T \rightarrow M_{PMI} - \log k$$ (Levy & Goldberg 2014)

What is SGNS learning?

- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word
- It turns out that when *d* is large and after enough training iterations, **you get the PMI matrix**… except for a constant:

$$W \cdot C^T \rightarrow M_{PMI} - \log k \qquad \text{(Levy \& Goldberg 2014)}$$

- So **SGNS is factorising the word-word PMI matrix**
  - **SVD does this too!**
  - Mathematically, nothing really new then…

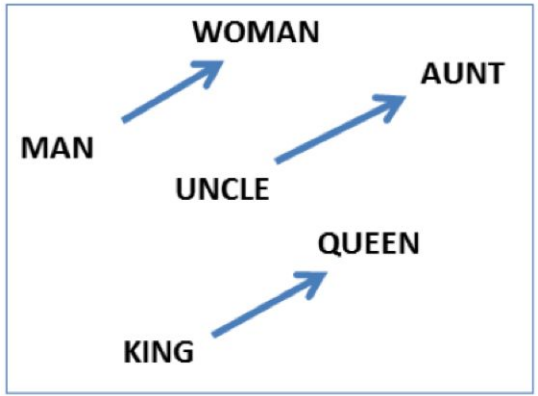
What is SGNS learning?

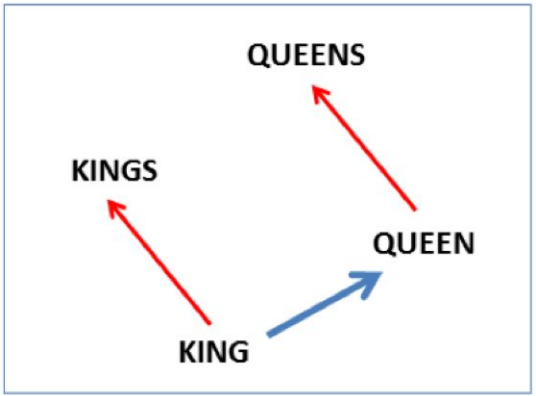
- Take both SGNS embedding matrices, multiply them,
- You get a $|V|^2$ matrix in which each cell describes the relation between a target word and a context word
- It turns out that when *d* is large and after enough training iterations, **you get the PMI matrix**… except for a constant: $W \cdot C^T \rightarrow M_{PMI} - \log k$
  (Levy & Goldberg 2014)
- So **SGNS is factorising the word-word PMI matrix**
  - **SVD does this too!**
  - Mathematically, nothing really new then…
  - …but a number of clever refinements make the difference — they can be adapted to the SVD approach, with very good results! (Levy et al. 2015)

Analogy: the structure of word similarity

vector(*'king'*) – vector(*'man'*) + vector(*'woman'*) ≈ vector('*queen*') — if vector('king') is left aside

vector(*'Paris'*) – vector(*'France'*) + vector(*'Italy'*) ≈ vector('*Rome*')





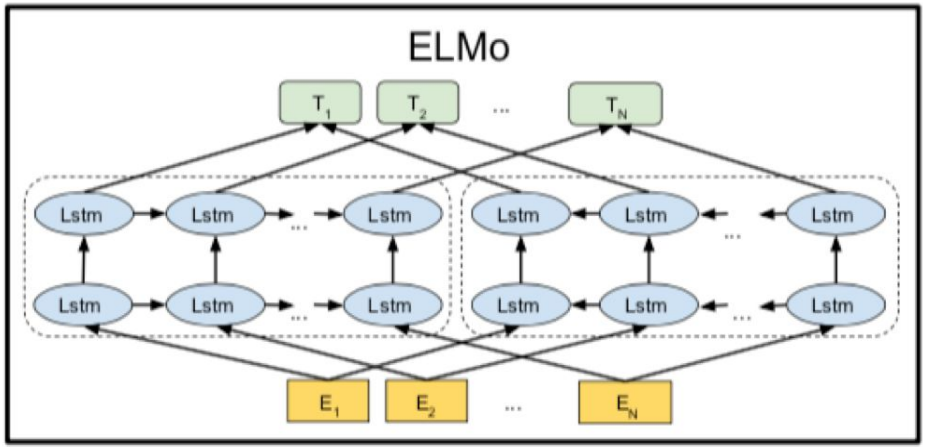
Contextual word embeddings

Word representation meet language models

- What we want to do
  - Build a representation that takes the **whole context** into account
  - We need to **selectively accumulate information** from the (left and right) context
- This can be achieved using a **sequence modelling NN**
  - Language modelling do create context-dependent representations of (sub)word occurrences
  - This is the **point of convergence** of two very different lines of research:
    1. Language modelling (cf. last class)
    2. Word vector representation

ELMo (Peters et al. 2018)

- Train a BiLSTM on a large dataset for bidirectional language modelling, in this case to predict the next word
- Encode the sentence by running it through both forward and backward LSTMs
- Combine forward and backward representations into final contextual embeddings



ELMo (Peters et al. 2018)

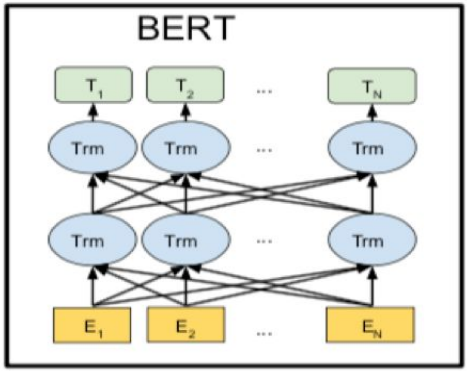
- So how contextual are ELMo embeddings?

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

- Additional benefit : standard ELMo models use as input vectors the result of a CNN to capture the character-level content of words
    - So they can even handle unknown words!
- A task-specific NN can then be trained to perform a task when ELMo contextual embeddings of words are provided as input

BERT (Devlin et al. 2019)

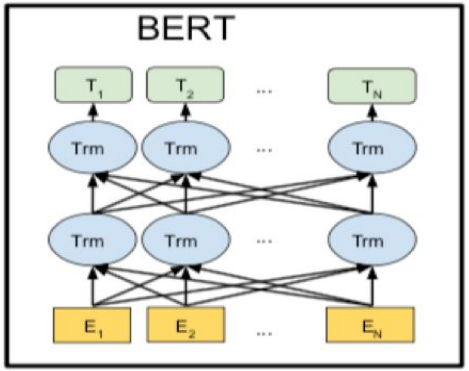
- Same underlying idea : contextual embeddings via word prediction
- Use the encoder part of a Transformer architecture instead of a BiLSTM



Each level sees the whole level below, contrarily to RNNs such as LSTMs

BERT (Devlin et al. 2019)

- Same underlying idea : contextual embeddings via word prediction
- Use the encoder part of a Transformer architecture instead of a BiLSTM

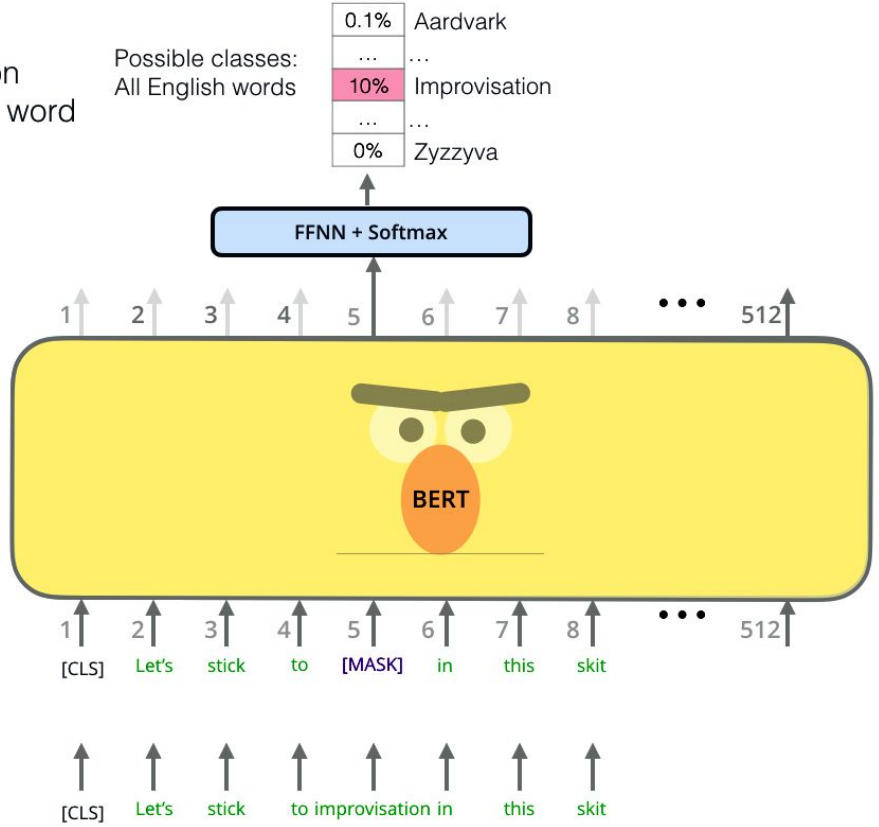


Each level sees the whole level below, contrarily to RNNs such as LSTMs

- Different Language Modelling objective: the Masked Language Model
  - Given a sentence with some words masked at random, can we predict them? (cf. Taylor 1954, “Cloze task”)
  - Randomly select 15% of tokens for which an embedding will be computed and evaluated; 80% of them are replaced with “<MASK>”, 10% are left unchanged, 10% are replaced by a random subword

BERT (Devlin et al. 2019)

Use the output of the masked word's position to predict the masked word

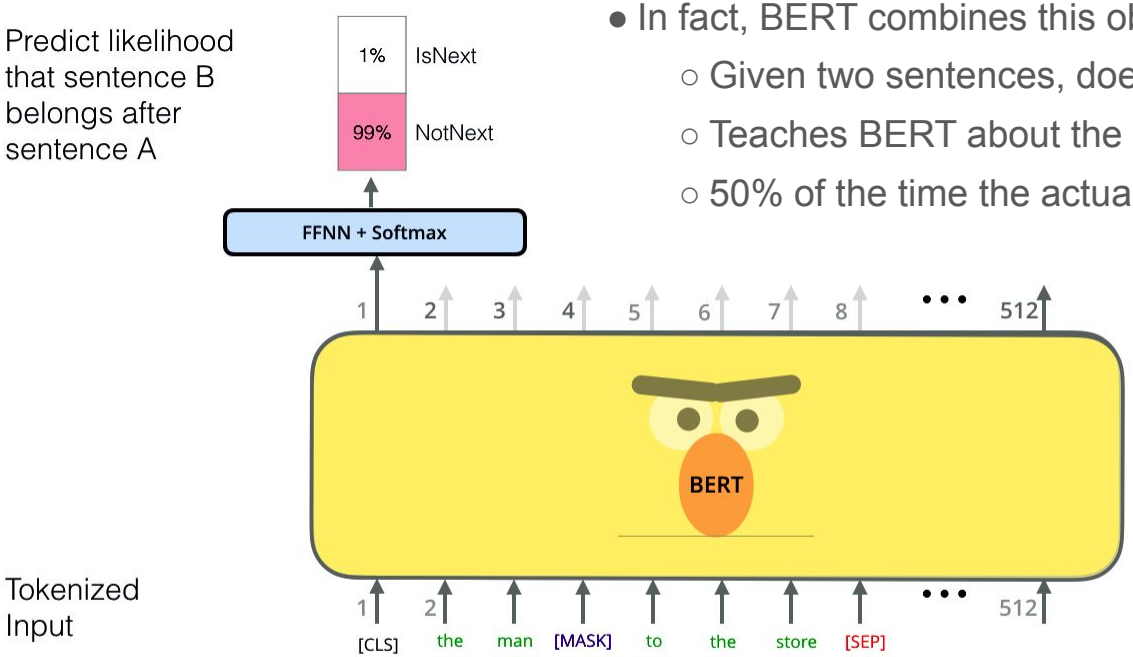


Randomly mask 15% of tokens

Input

Figure (adapted) from http://jalammar.github.io/illustrated-bert/

BERT (Devlin et al. 2019)



- In fact, BERT combines this objective with another one:
  - Given two sentences, does the first follow the second?
  - Teaches BERT about the relationship between two sentences
  - 50% of the time the actual next sentence, 50% random

Figure (adapted) from http://jalammar.github.io/illustrated-bert/

BERT variants and extensions

- **Multilingual BERT** (same paper as BERT): BERT trained on texts covering ~100 languages (mostly Wikipedia editions)
- **ROBERTa** (Liu et al. 2019): more training, better hyperparameters, only the masked word model objective, and other technical changes
- **ALBERT** (Lan et al. 2019): technique for having fewer parameters than in vanilla BERT (see also **DistilBERT**)
- **CamemBERT** (Martin et al. 2019): a ROBERTa-like model trained on French data, with a few (technical) differences. Works significantly better than the multilingual BERT

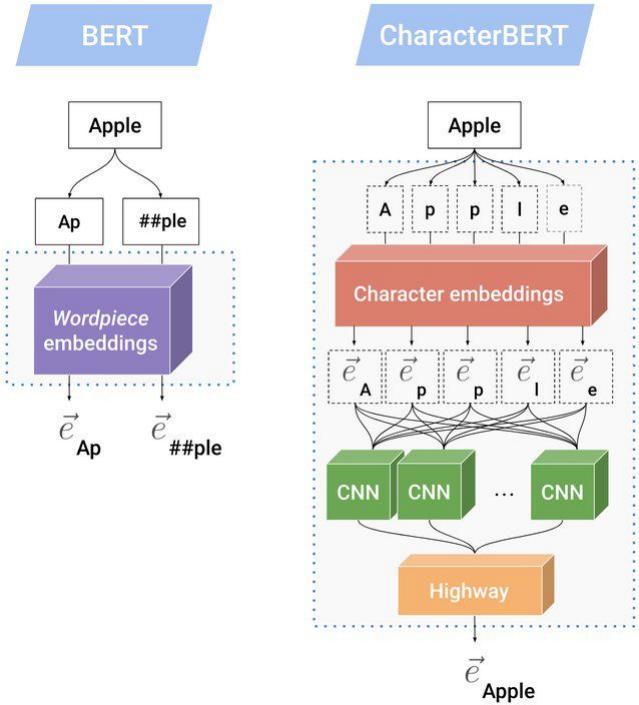
- **XLM** (Lample & Conneau 2019): The MLM objective is similar to the one of BERT, but with continuous streams of text as opposed to sentence pairs. A Translation Language Modelling (TLM) objective is added, whereby to predict a masked English word, the model can attend to both the English sentence and its French translation, and is encouraged to align English and French representations.
- **BART** (Lewis et al. 2019): uses a *full* Transformer architecture to train a denoising autoencoder to be used as pretraining for sequence-to-sequence models. It is trained by corrupting text with an arbitrary noising function (e.g. masking), and learning a model to reconstruct the original text

Training BERT/ROBERTa

- **How much data** is required to train a BERT (or ROBERTa) ?
- Short answer: we don’t really know yet
- The original BERT model was trained on 16GB of uncompressed text (~4B tokens, Wikipedia + Book Corpus)
- The authors of the ROBERTa paper have shown that more data helps
    - They use 160GB of uncompressed text (~40B tokens, more diverse)
- For CamemBERT, we experimented with the French Wikipedia (1B tokens) and our web-based, Common-Crawl-derived corpus, OSCAR (32B tokens)
- Using 1B tokens randomly selected from OSCAR works as well as the whole corpus,
- …but better than the French Wikipedia: heterogeneity helps!
- At least 38 languages have an OSCAR corpus with 1B tokens or more

Character-level language models

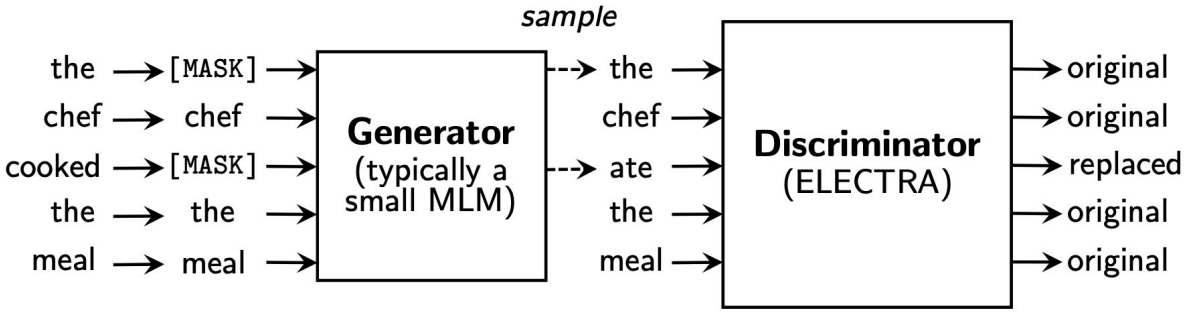
- **Tokenisation (subword segmentation) is somewhat arbitrary, not very robust to small changes** (spelling errors…) **and not very good at dealing with neologisms**
- Hence the development of architectures that **process the input at the character level**
  - **CharacterBERT** (El Boukkouri et al. 2020): ELMo-like (CNN-based) pretoken-level embedding followed by a Transformer encoder à la BERT
  - **CharBERT** (Ma et al. 2020): character-level representation (bi-GRU) + subword-level representation are merged, followed by a modified (dual channel) Transformer encoder architecture with a “noisy language modelling” loss
  - **CANINE** (Clark et al. 2021): tokenisation-less architecture that combines downsampling, which reduces the input sequence length, with a deep transformer stack, which encodes context



CharacterBERT (El Boukkouri et al. 2020)

ELECTRA (Clark et al. 2020)

- Two-tier Transformer-based architecture

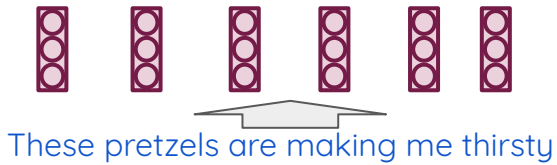


- The Generator replaces a random proportion of input tokens using a MLM loss
- The Discriminator is trained to distinguish replaced tokens from original ones (including when the generator replaced the input token by itself)
- Looks like an adversarial architecture, but it is not: no random input given to the generator, generator loss not designed to fool the discriminator
- Reaches performance levels better than BERT or ROBERTa with a given amount of pre-training data (especially useful when little training data is available)

Transfer Learning with Pre-trained Models

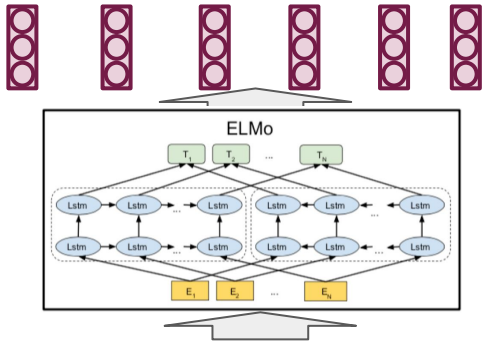
Transfer Learning: Frozen Embeddings

1. Pre-train word-/contextualized word-/sentence embeddings



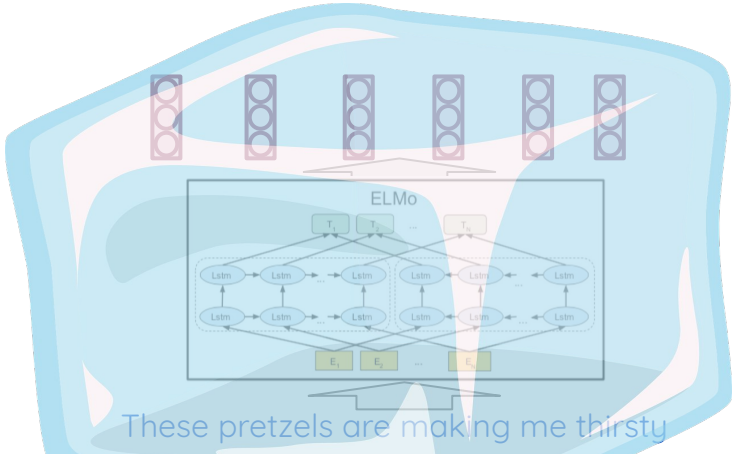
Transfer Learning: Frozen Embeddings

1. Pre-train word-/contextualized word-/sentence embeddings



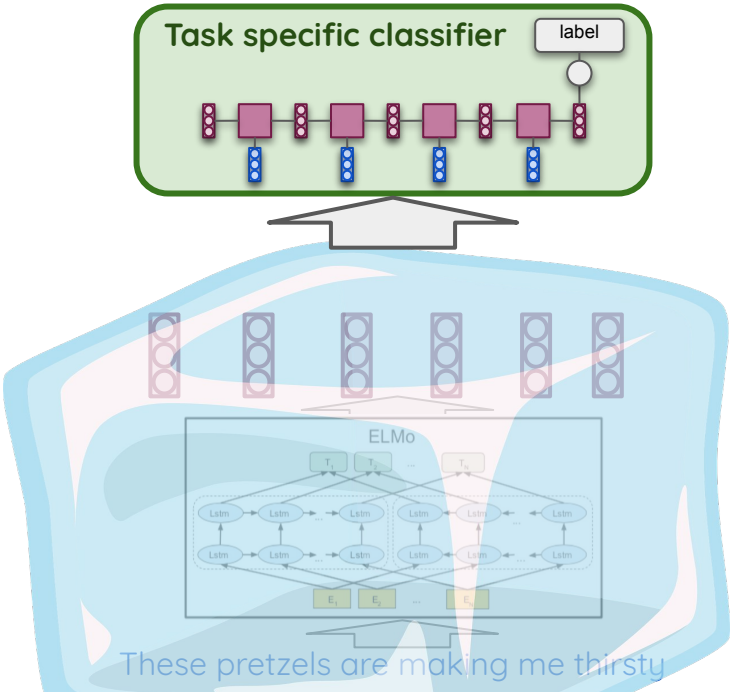
Transfer Learning: Frozen Embeddings

1. Pre-train word-/contextualized word-/sentence embeddings
2. Freeze weights



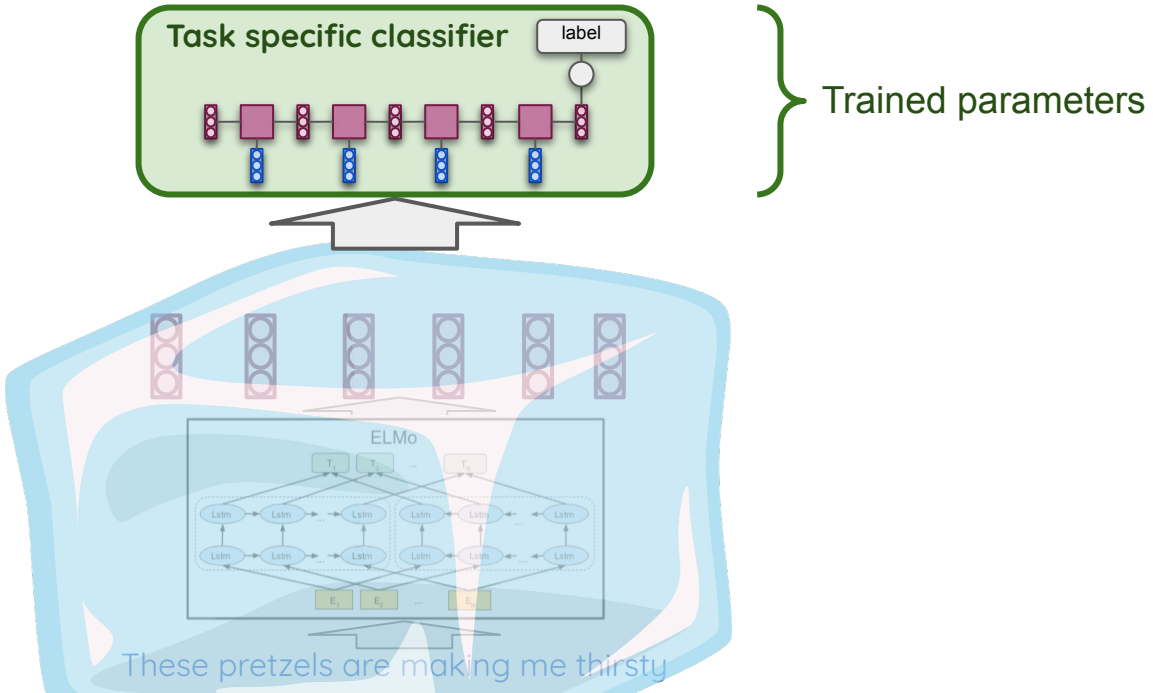
Transfer Learning: Frozen Embeddings

1. Pre-train word-/contextualized word-/sentence embeddings
2. Freeze weights
3. Train classifier on top



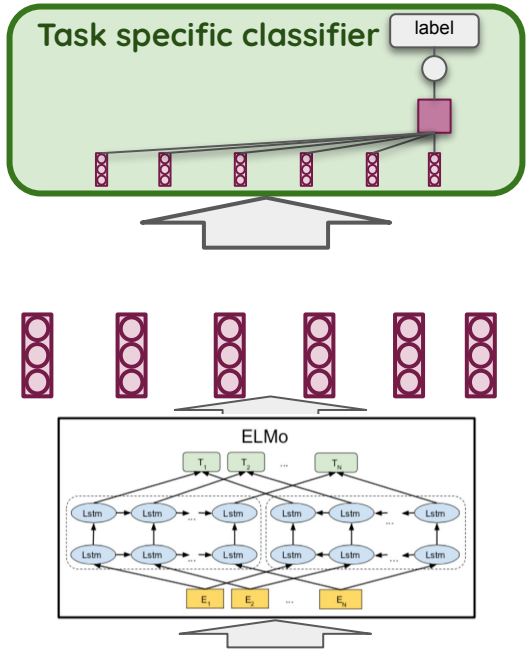
Transfer Learning: Frozen Embeddings

1. Pre-train word-/contextualized word-/sentence embeddings
2. Freeze weights
3. Train classifier on top



Transfer Learning: Fine-tuning

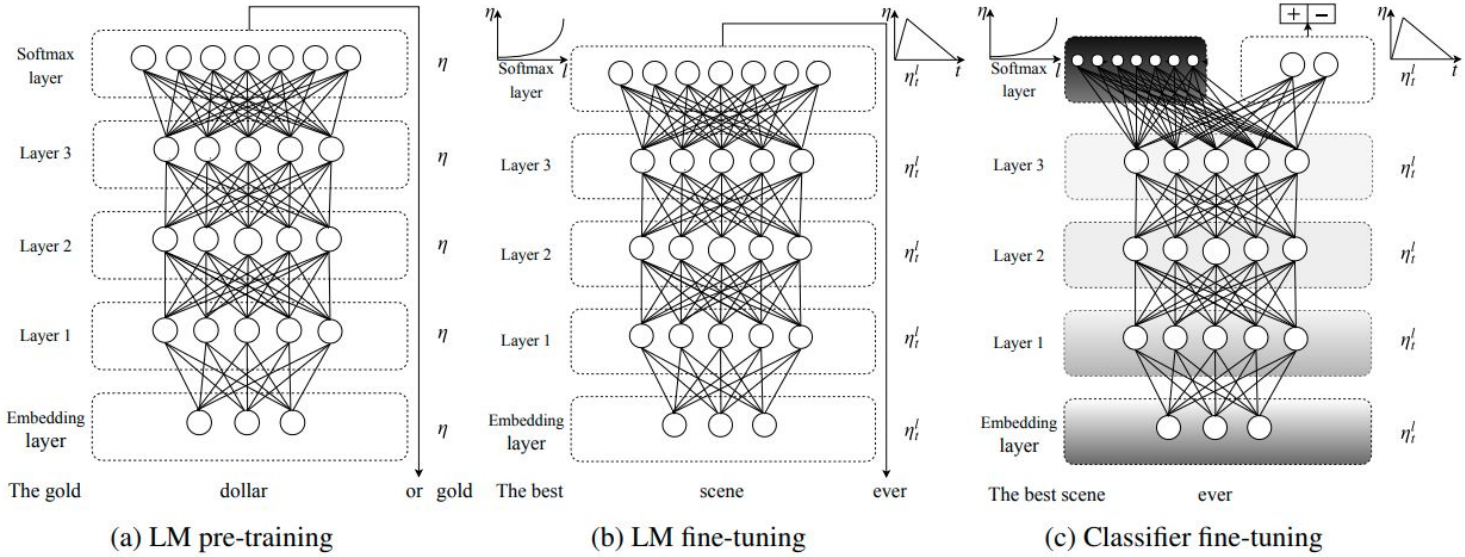
1. Pre-train word-/contextualized word-/sentence embeddings
2. Train (**simple**) classifier on top
3. **Also update base model**



Trained parameters

ULMFit: Fine-tuning Contextual Embeddings

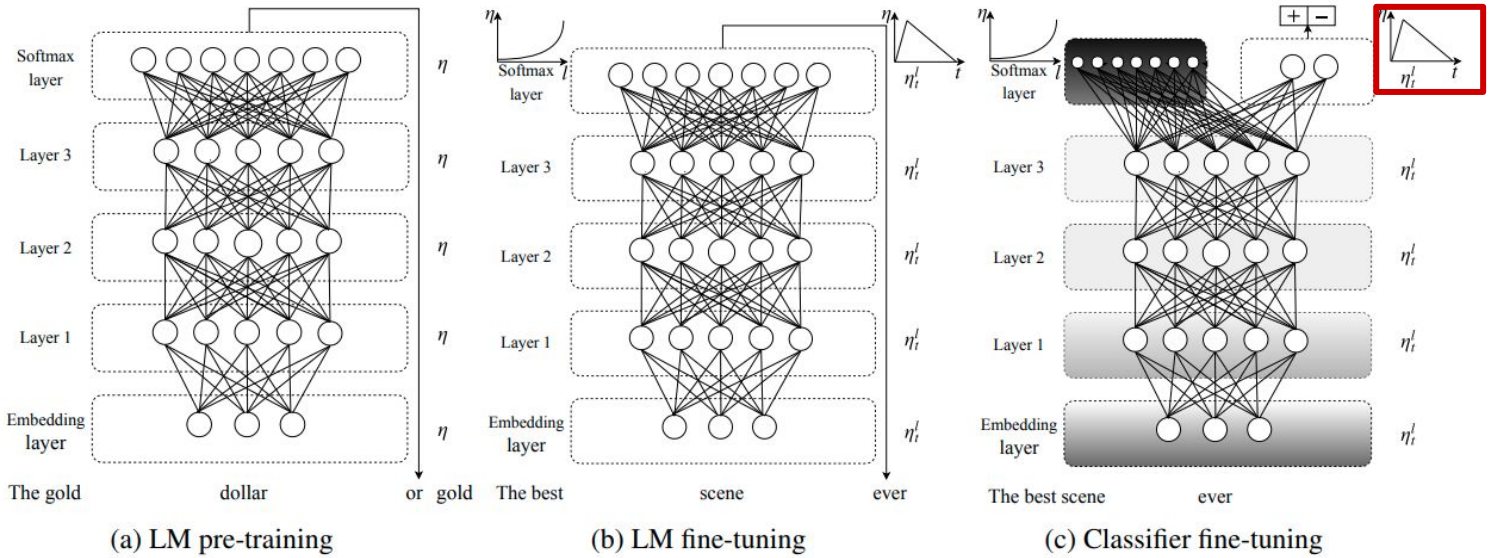
- Pre-trained big (Bi)LSTM language model(s)
- Fine-tuned on task + auxiliary LM objective



Howard & Ruder (2018)

ULMFit: Fine-tuning Contextual Embeddings

- Pre-trained big (Bi)LSTM language model(s)
- Fine-tuned on task + auxiliary LM objective
  - With carefully tuned learning rate schedule

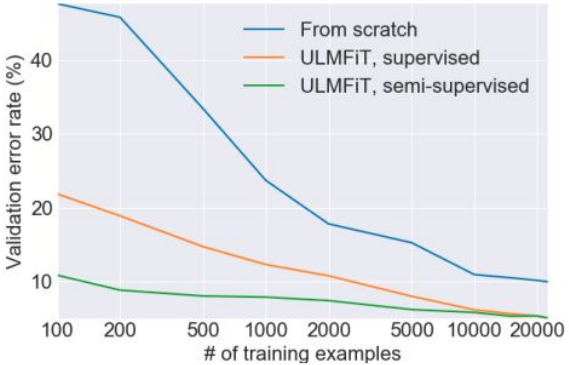


Howard & Ruder (2018)

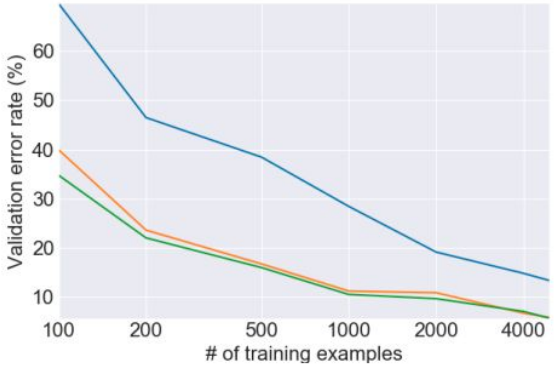
ULMFit: Fine-tuning Contextual Embeddings

- Pre-trained big (Bi)LSTM language model(s)
- Fine-tuned on task + auxiliary LM objective
  - With carefully tuned learning rate schedule
- **Better than training same model from scratch**

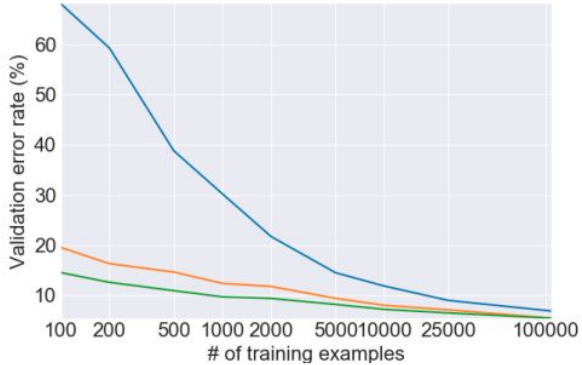
Sentiment Classification



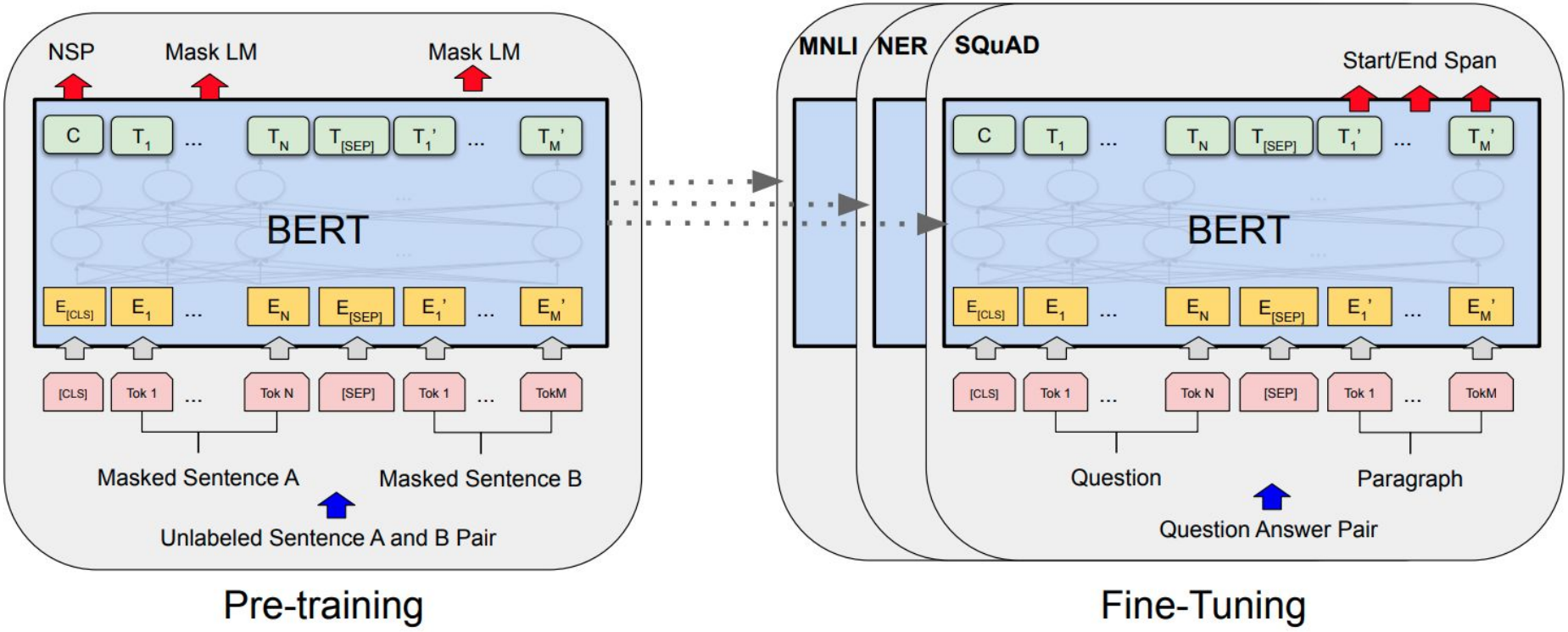
Question Classification



Topic Classification

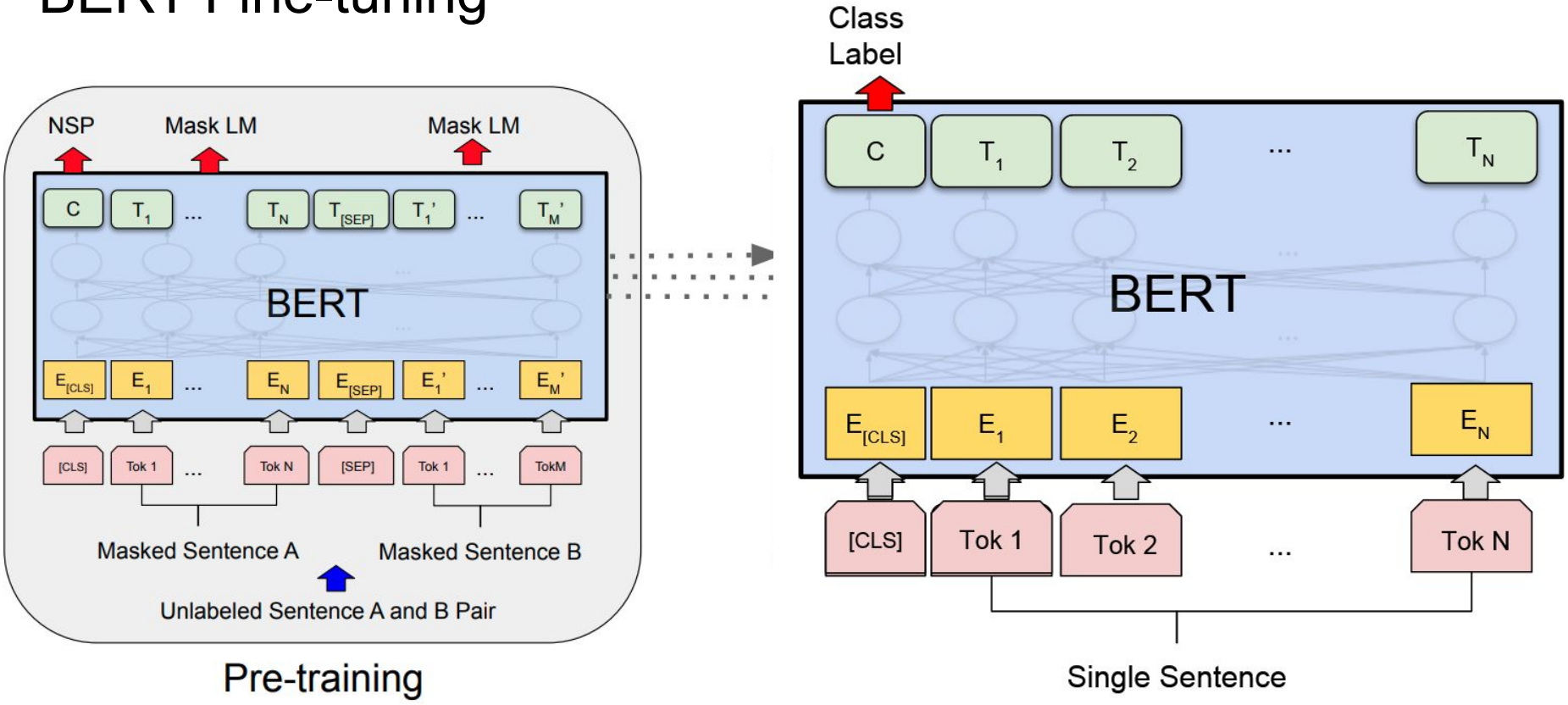


BERT Fine-tuning



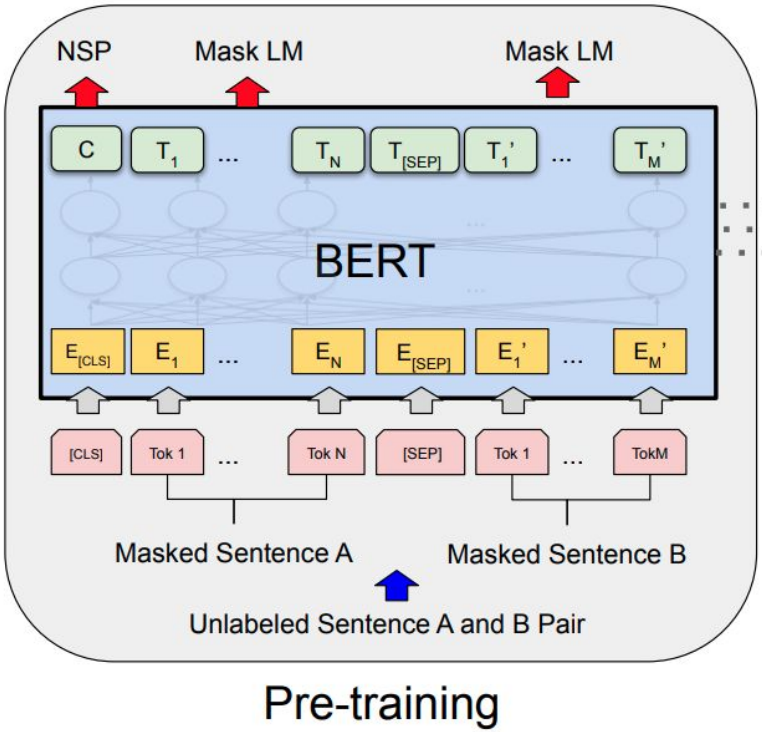
Devlin et al. (2018)

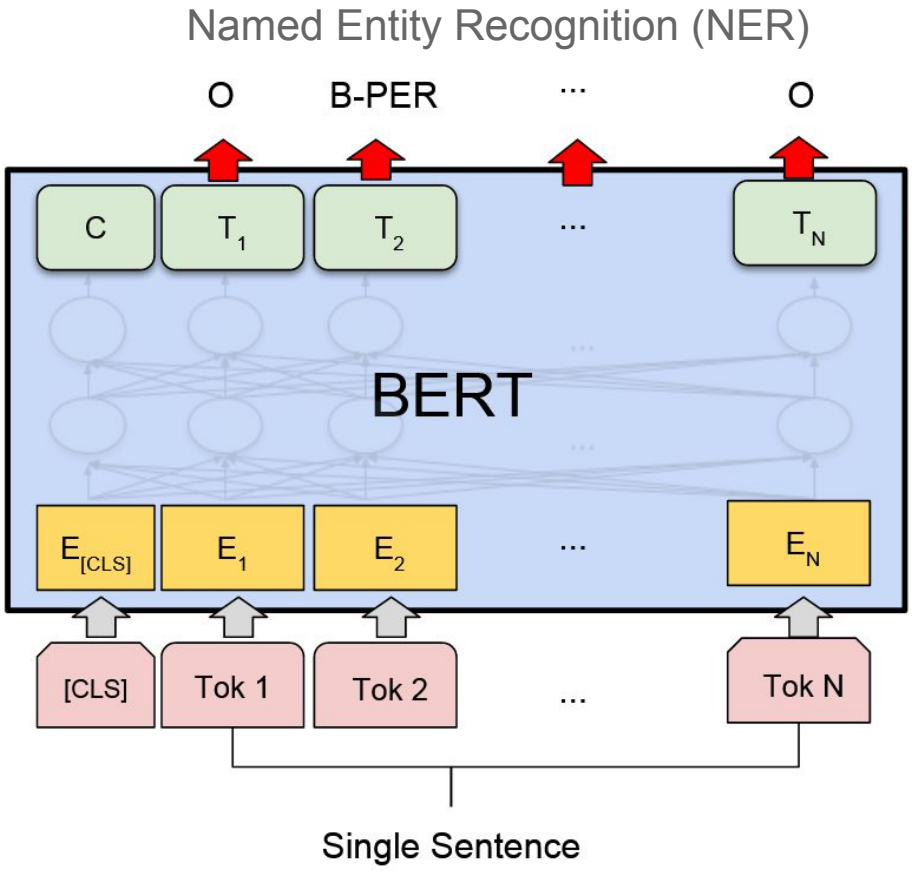
BERT Fine-tuning



Use [CLS] token embedding as model output for classification Devlin et al. (2018)

BERT Fine-tuning





But more types of tasks are also supported

Devlin et al. (2018)

Freeze or Fine-tune?

Frozen embeddings

- ✓ Fewer parameters to train (only task-specific parameters)
- ✓ Can be pre-computed for faster learning
- ✘ Worse performance generally
- ✘ More parameters overall (more task-specific parameters)

Fine-tuning

- ✓ Higher performance
- ✓ Smaller overall parameter count
- ✘ More parameters to learn
- ✘ Need to be more careful about hyper-parameters to avoid over-fitting

Pretraining	Adaptation	NER CoNLL 2003	SA SST-2	Nat. lang. inference MNLI	Nat. lang. inference SICK-E	Semantic textual similarity SICK-R	Semantic textual similarity MRPC	Semantic textual similarity STS-B
Skip-thoughts	❄	-	81.8	62.9	-	86.6	75.8	71.8
ELMo	❄	91.7	**91.8**	**79.6**	**86.3**	**86.1**	**76.0**	**75.9**
	🔥	**91.9**	91.2	76.4	83.3	83.3	74.7	75.5
	Δ=🔥-❄	0.2	-0.6	-3.2	-3.3	-2.8	-1.3	-0.4
BERT-base	❄	92.2	93.0	**84.6**	84.8	86.4	78.1	82.9
	🔥	**92.4**	**93.5**	**84.6**	**85.8**	**88.7**	**84.8**	**87.1**
	Δ=🔥-❄	0.2	0.5	0.0	1.0	2.3	6.7	4.2

Peters, Ruder et al. (2019)

Fine-tuning Performance over Time

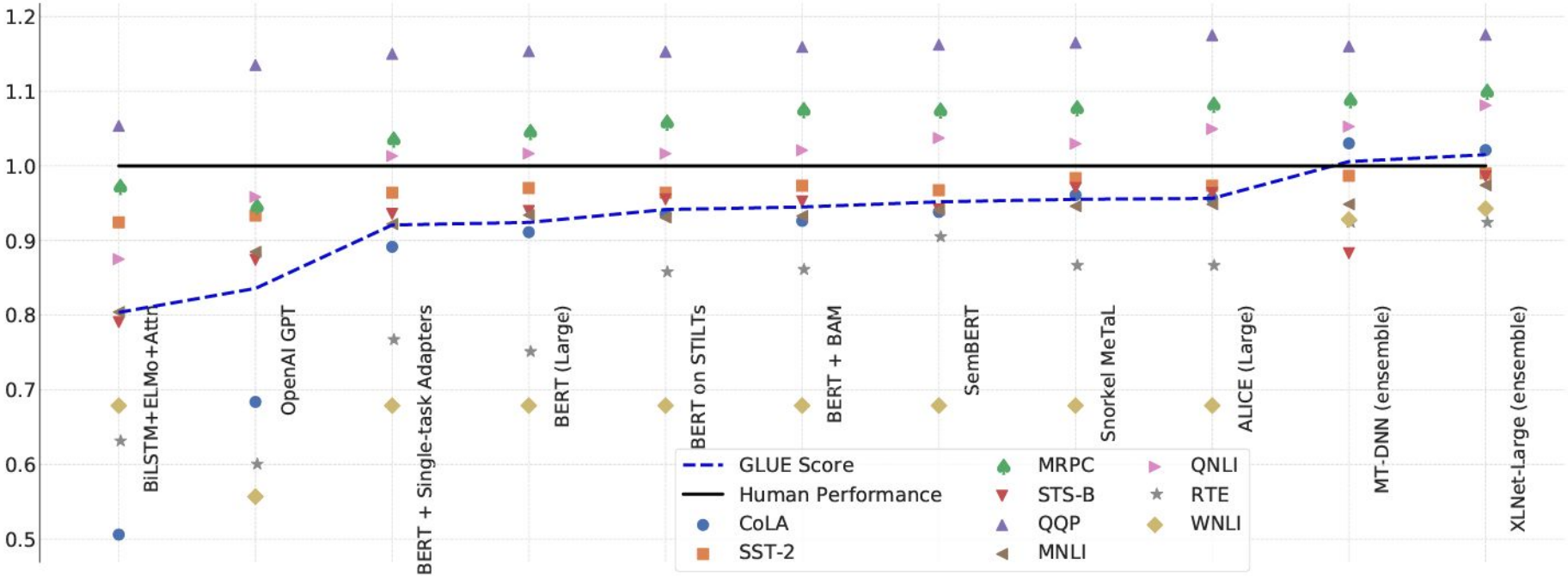
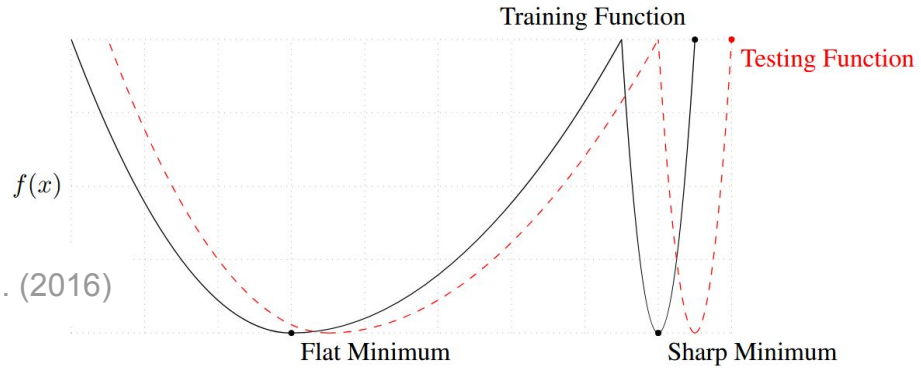


Figure from Wang et al. (2019)

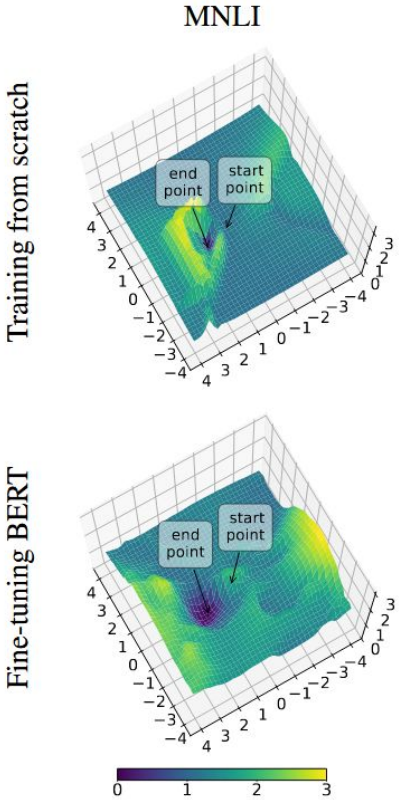
“BERTology”: Why Does Fine-tuning Work so Well?

- Why does fine-tuning a large pre-trained model not lead to overfitting?
- Some theory on minima found by SGD on pre-trained model
  - Flat minima -> better generalization (hypothesis by Keskar et al. 2016)



Keskar et al. (2016)

Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

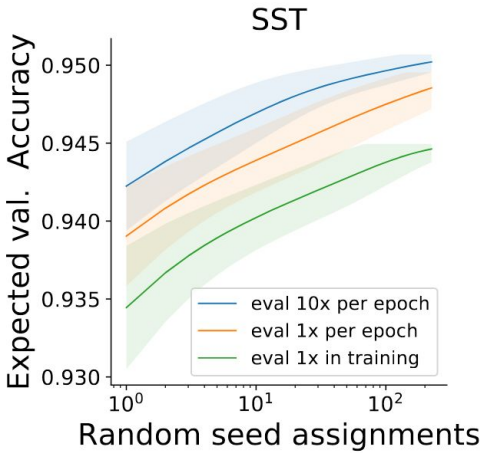


Hao et al. (2019)

“BERTology”: Why Does Fine-tuning Work so Well?

- Why does fine-tuning a large pre-trained model not lead to overfitting?
- Some theory on minima found by SGD on pre-trained model
  - Flat minima -> better generalization (hypothesis by Keskar et al. 2016)
- More prosaically: highly tuned fine-tuning procedure
  - Adam optimizer with good defaults
  - Few epochs, early stopping
  - Warmed up learning rate schedule

Influence of validation frequency and random seed on accuracy



Dodge et al. (2020)

See “A Primer in BERTology: What We Know About How BERT Works” Rogers et al. (2020) for a more complete overview

“BERTology”: What Does BERT Learn?

- Does BERT leverage some kind of syntactic information?
  - Some positive evidence

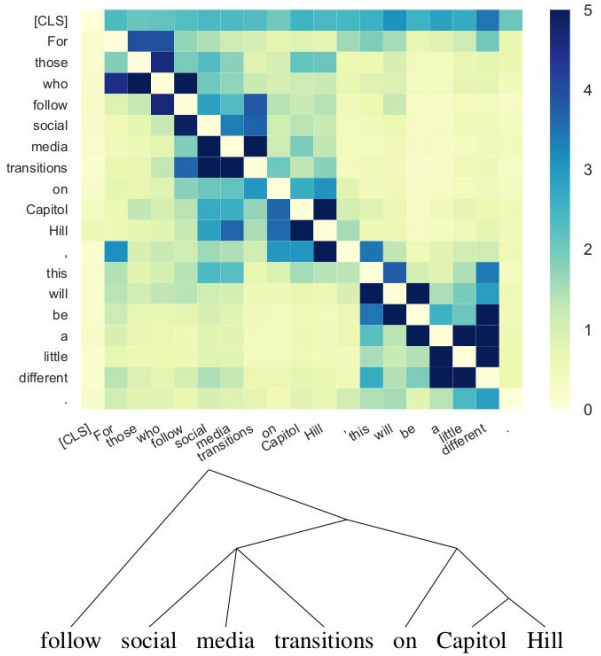


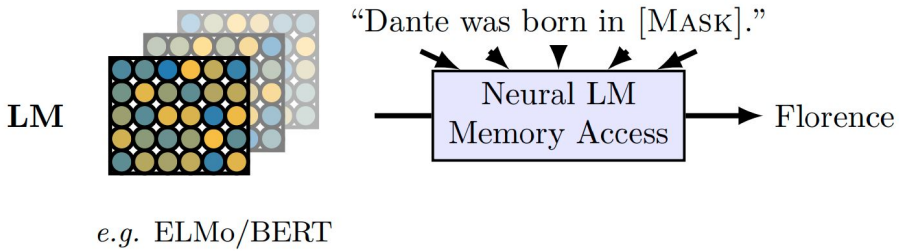
Figure 1: Parameter-free probe for syntactic knowledge: words sharing syntactic subtrees have larger impact on each other in the MLM prediction

Wu et al. (2020)

See “A Primer in BERTology: What We Know About How BERT Works” Rogers et al. (2020) for a more complete overview

“BERTology”: What Does BERT Learn?

- Does BERT leverage some kind of syntactic information?
  - Some positive evidence
- Does BERT learn world knowledge?
  - More on that later…



Petroni et al. (2019)

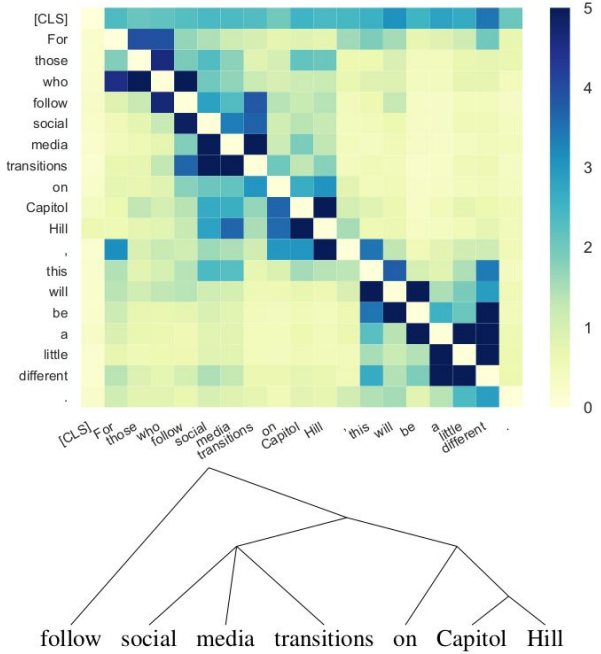


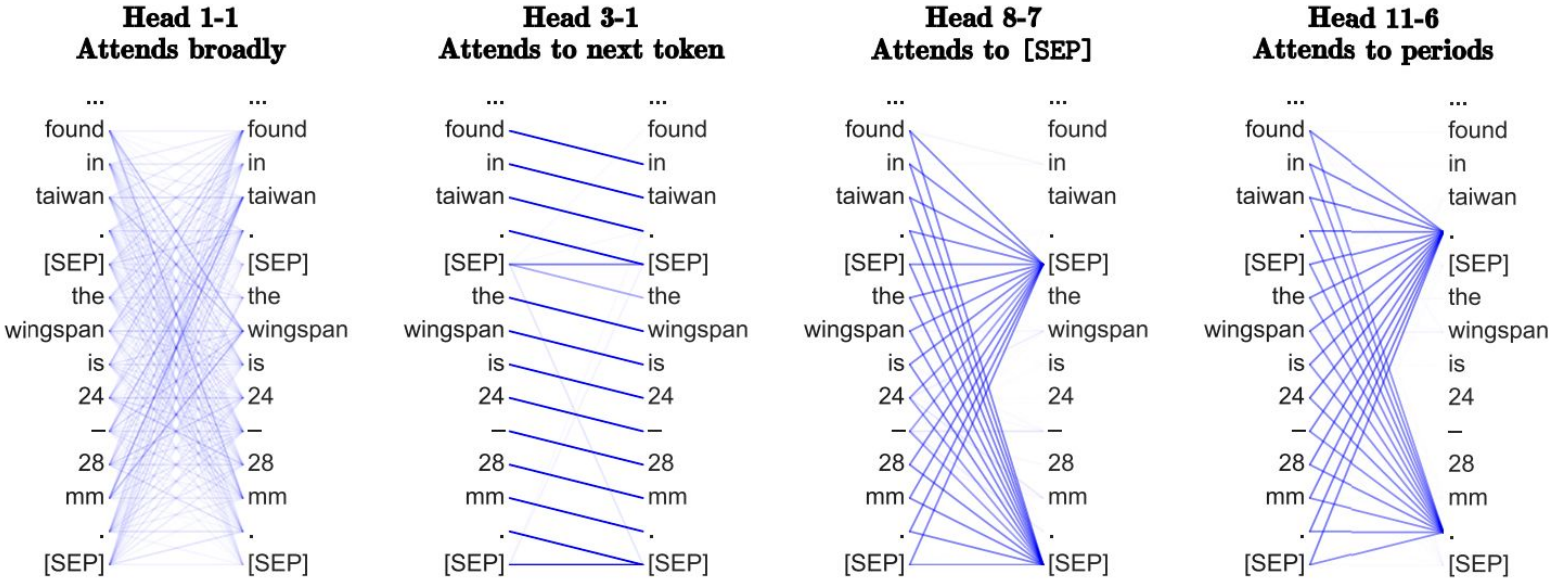
Figure 1: Parameter-free probe for syntactic knowledge: words sharing syntactic subtrees have larger impact on each other in the MLM prediction

Wu et al. (2020)

See “A Primer in BERTology: What We Know About How BERT Works” Rogers et al. (2020) for a more complete overview

“BERTology”: A Closer look at Attention

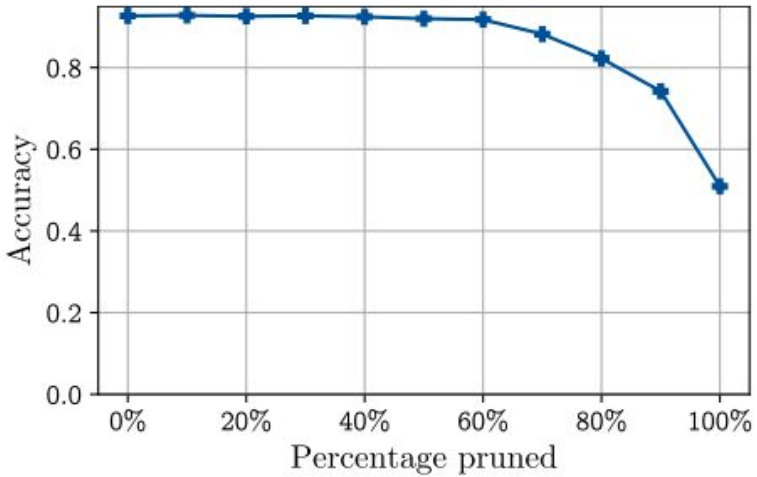
- What do attention heads pay look at?



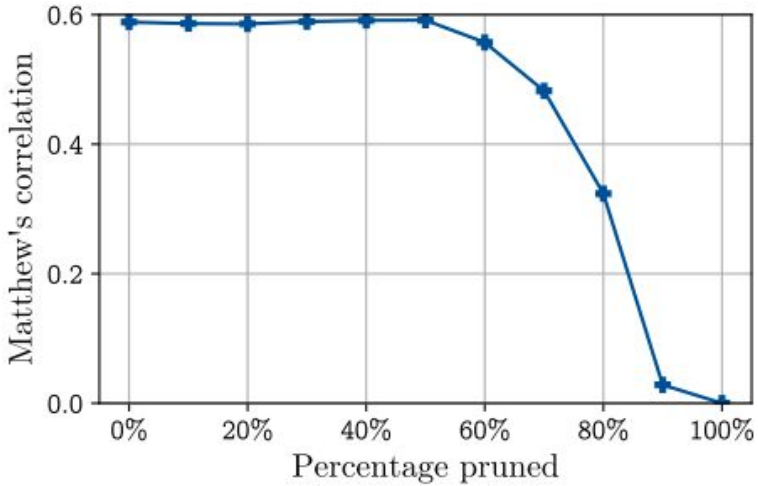
Clark et al. (2019)

“BERTology”: A Closer look at Attention

- What do attention heads pay look at?
- Do we even need all attention heads?



Sentiment Classification



Linguistic Acceptability

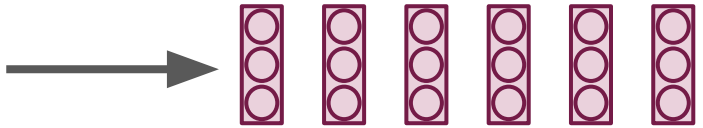
Large number of heads can be removed without impacting performance!

Michel et al. (2019)

Sentence Representations

From Words To Sentence Representations

- Word-level representations
  - One representation per word
  - Good early in the pipeline

These pretzels are making me thirsty 

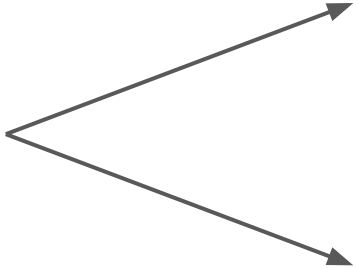
- Sentence/utterance-level representations
  - One representation per utterance

These pretzels are making me thirsty 

Why Sentence Representations?

- Sentence classification

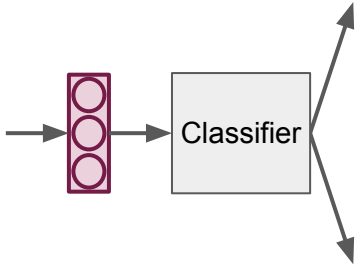
“Seinfeld comes fully into its own as a divine comedy of petty grievances”



Why Sentence Representations?

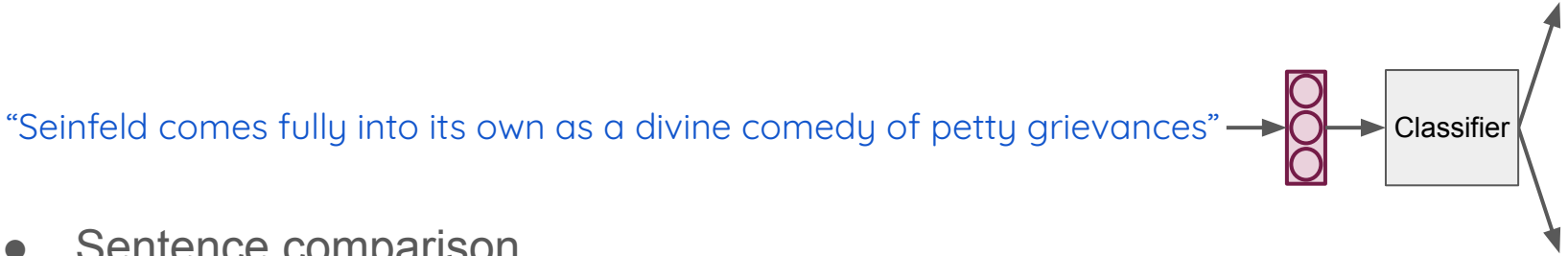
- Sentence classification

"Seinfeld comes fully into its own as a divine comedy of petty grievances"

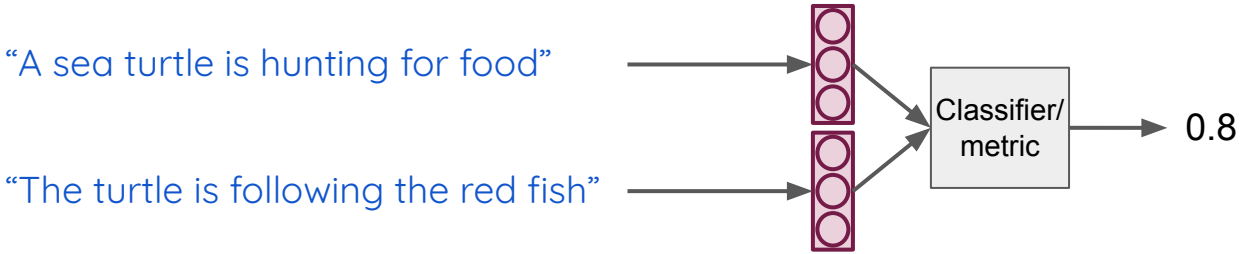


Why Sentence Representations?

- Sentence classification



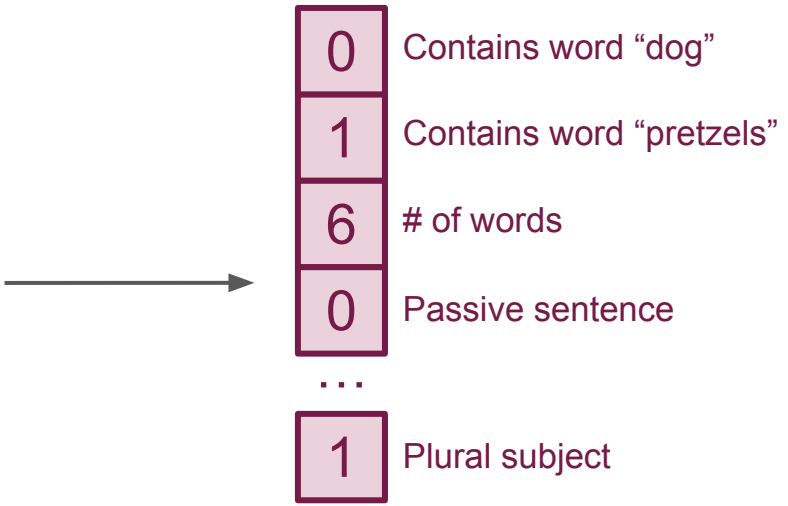
- Sentence comparison
  - Semantic similarity/Paraphrase detection
  - Retrieval



Encoding Sentences: Pre-neural

- Features

These pretzels are making me thirsty



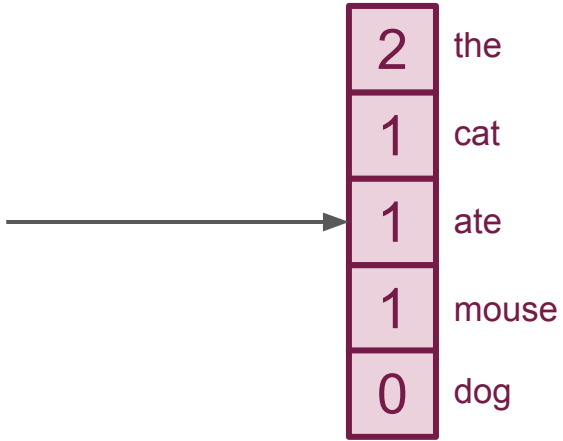
Encoding Sentences: Pre-neural

- Features

These pretzels are making me thirsty

- TF-IDF
  - Coordinate = frequency of word

The cat ate the mouse



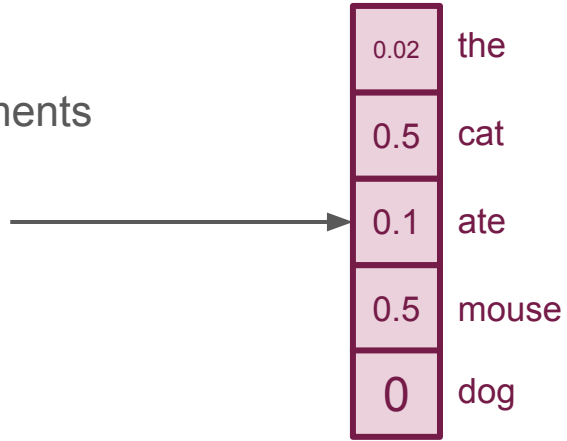
Encoding Sentences: Pre-neural

- Features

These pretzels are making me thirsty

- TF-IDF
  - Coordinate = frequency of word
  - Renormalize by frequency of word across documents

The cat ate the mouse



Encoding Sentences: Word-embedding based

- Average of word embeddings
  - Very simple yet strong baseline!

These pretzels are making me thirsty

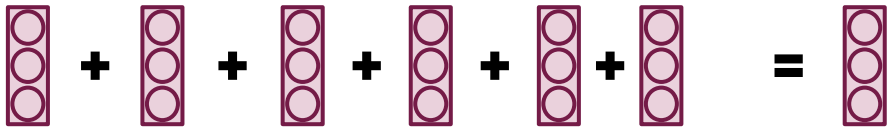
Encoding Sentences: Word-embedding based

- Average of word embeddings
  - Very simple yet strong baseline!

These pretzels are making me thirsty

Encoding Sentences: Word-embedding based

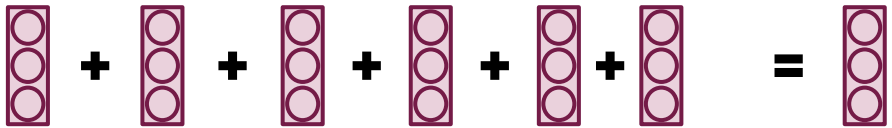
- Average of word embeddings
  - Very simple yet strong baseline!



These pretzels are making me thirsty

Encoding Sentences: Word-embedding based

- Average of word embeddings
  - Very simple yet strong baseline!



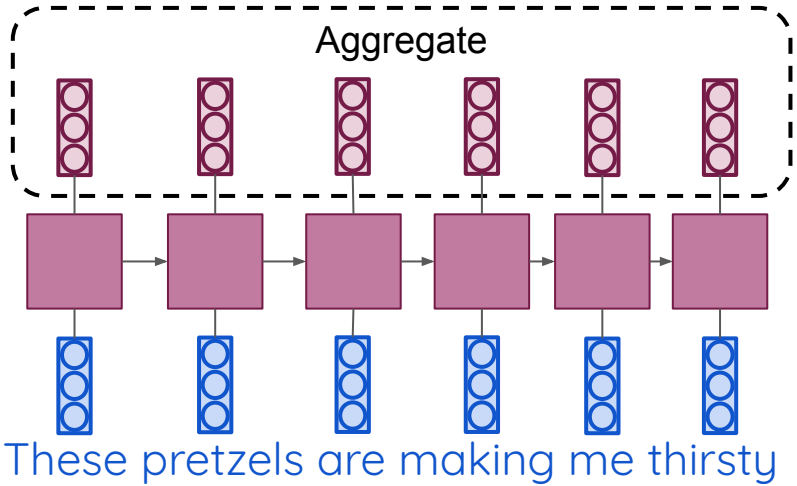
These pretzels are making me thirsty

- Still order-independent

Encoding Sentences: Neural Based

Output of Neural Network

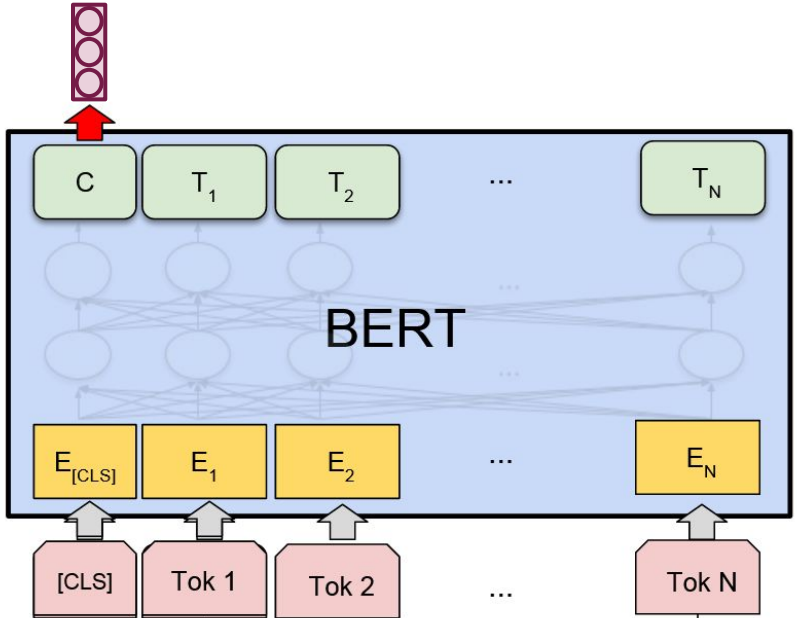
1. Run RNN/LSTM/Bidirectional LSTM/ Transformer…
2. Pool the resulting embeddings in one vector
   - Average/max pooling
   - Take first/last encoding



Encoding Sentences: Neural Based

Output of Neural Network

1. Run RNN/LSTM/Bidirectional LSTM/ Transformer…
2. Pool the resulting embeddings in one vector
   - Average/max pooling
   - Take first/last encoding
   - Encoding of CLS token (BERT)



Evaluating Sentence Representations

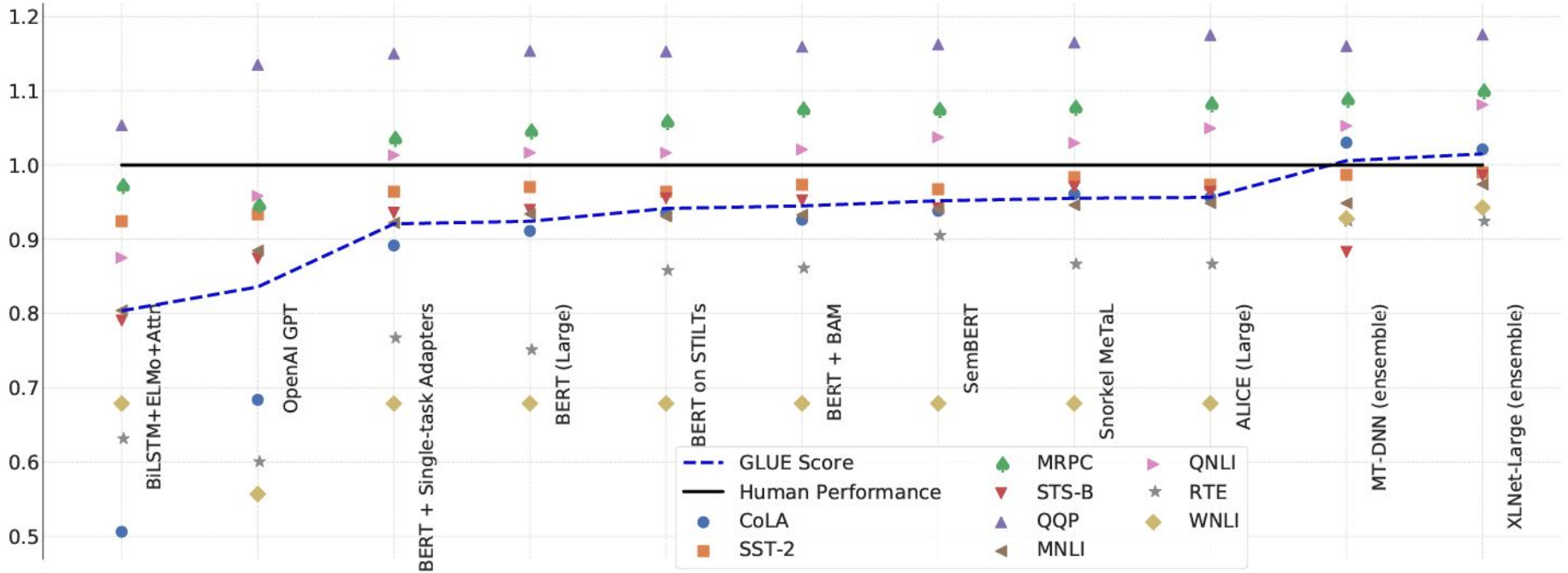
Multitask Benchmarks: GLUE (Wang et al. 2019)

- **Linguistic acceptability**
  - Is this sentence a valid English sentence
- **Sentiment Classification**
  - Is this sentence a valid
- **Natural Language Inference/Textual Entailment**
  - Does sentence 1 entail/contradict sentence 2
- **Semantic similarity**
  - How similar are sentence 1 and sentence 2
- **Paraphrase Detection**
  - Is sentence 1 a paraphrase of sentence 2?

Evaluating Sentence Representations

Model	Avg	Single Sentence CoLA	Single Sentence SST-2	Similarity and Paraphrase MRPC	Similarity and Paraphrase QQP	Similarity and Paraphrase STS-B	Natural Language Inference MNLI	Natural Language Inference QNLI	Natural Language Inference RTE	Natural Language Inference WNLI
Single-Task Training										
BiLSTM	63.9	15.7	85.9	69.3/79.4	81.7/61.4	66.0/62.8	70.3/70.8	75.7	52.8	**65.1**
+ELMo	66.4	**35.0**	90.2	69.0/80.8	85.7/65.6	64.0/60.2	72.9/73.4	71.7	50.1	**65.1**
+CoVe	64.0	14.5	88.5	73.4/81.4	83.3/59.4	67.2/64.1	64.5/64.8	75.4	53.5	**65.1**
+Attn	63.9	15.7	85.9	68.5/80.3	83.5/62.9	59.3/55.8	74.2/73.8	77.2	51.9	**65.1**
+Attn, ELMo	66.5	**35.0**	90.2	68.8/80.2	**86.5/66.1**	55.5/52.5	**76.9/76.7**	76.7	50.4	**65.1**
+Attn, CoVe	63.2	14.5	88.5	68.6/79.7	84.1/60.1	57.2/53.6	71.6/71.5	74.5	52.7	**65.1**
Multi-Task Training										
BiLSTM	64.2	11.6	82.8	74.3/81.8	84.2/62.5	70.3/67.8	65.4/66.1	74.6	57.4	**65.1**
+ELMo	67.7	32.1	89.3	**78.0/84.7**	82.6/61.1	67.2/67.9	70.3/67.8	75.5	57.4	**65.1**
+CoVe	62.9	18.5	81.9	71.5/78.7	84.9/60.6	64.4/62.7	65.4/65.7	70.8	52.7	**65.1**
+Attn	65.6	18.6	83.0	76.2/83.9	82.4/60.1	72.8/70.5	67.6/68.3	74.3	58.4	**65.1**
+Attn, ELMo	**70.0**	33.6	**90.4**	**78.0**/84.4	84.3/63.1	74.2/72.3	74.1/74.5	**79.8**	58.9	**65.1**
+Attn, CoVe	63.1	8.3	80.7	71.8/80.0	83.4/60.5	69.8/68.4	68.1/68.6	72.9	56.0	**65.1**
Pre-Trained Sentence Representation Models										
CBoW	58.9	0.0	80.0	73.4/81.5	79.1/51.4	61.2/58.7	56.0/56.4	72.1	54.1	**65.1**
Skip-Thought	61.3	0.0	81.8	71.7/80.8	82.2/56.4	71.8/69.7	62.9/62.8	72.9	53.1	**65.1**
InferSent	63.9	4.5	85.1	74.1/81.2	81.7/59.1	75.9/75.3	66.1/65.7	72.7	58.0	**65.1**
DisSent	62.0	4.9	83.7	74.1/81.7	82.6/59.5	66.1/64.8	58.7/59.1	73.9	56.4	**65.1**
GenSen	66.2	7.7	83.1	76.6/83.0	82.9/59.8	**79.3/79.2**	71.4/71.3	78.6	**59.2**	**65.1**

Evaluating Sentence Representations



Probing Tasks

Test for certain properties of the encoded sentence

1. Train classifier to predict certain properties
2. Test on evaluation set

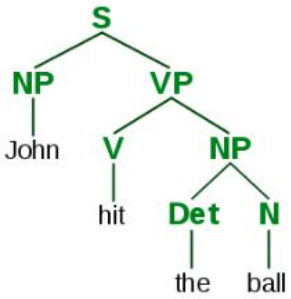
- High accuracy => information is encoded in the representation
- Low accuracy => ?
  - Information is difficult to to extract with (linear classifier)
  - Doesn’t mean it isn’t there

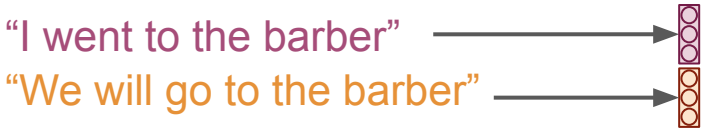
Probing Tasks

Test for certain properties of the encoded sentence

Examples from SentEval (Conneau et al. 2018)

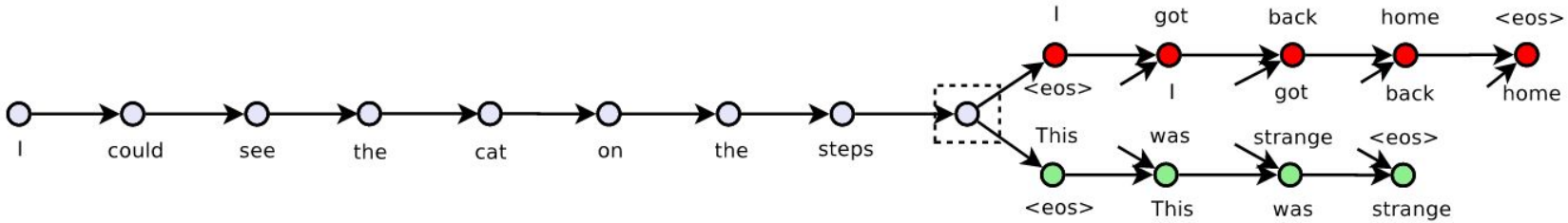
- **Surface Information**
  - Sentence length
  - Word content
- **Syntactic Information**
  - Sensitivity to word order
  - Depth of syntax tree (are there many subordinate/coordinate clauses)
  - Top-level constituent
- **Semantic Information**
  - Number of subject/object
  - Tense





Training Sentence Encodings with **Skip-Thought**

Predict next and previous sentences

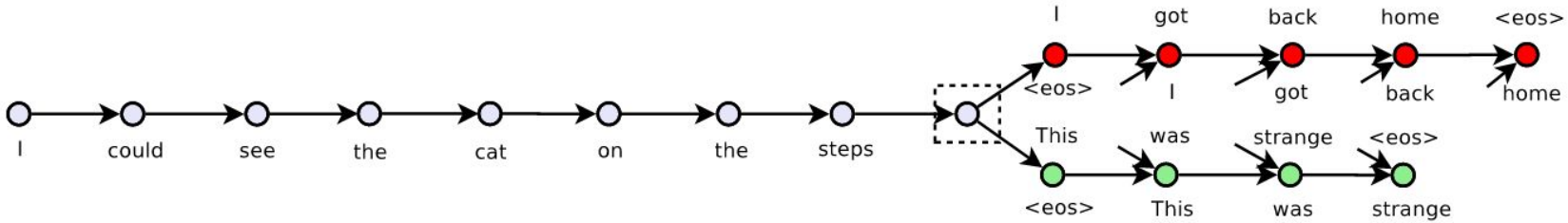


Kiros et al. (2015)

- Analogous to distributional semantics seen earlier
  - But at the sentence level
  - Precursor of BERT next sentence prediction task

Training Sentence Encodings with **Skip-Thought**

Predict next and previous sentences



Kiros et al. (2015)

- Analogous to distributional semantics seen earlier
  - But at the sentence level
  - Precursor of BERT next sentence prediction task

- Example usage: aligning text with other modalities
  - Books and text

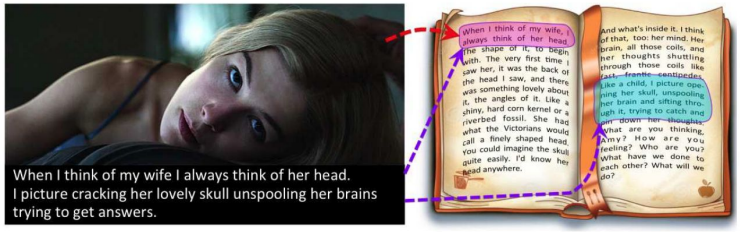


Figure 1: Shot from the movie *Gone Girl*, along with the subtitle, aligned with the book. We reason about the visual and dialog (text) alignment between the movie and a book.

Zhu et al. (2015)

Training Sentence Encodings with **Paraphrasing**

Train encoder to maximize **similarity** between embeddings of **paraphrases**

Paraphrases Negative example

$$\sum_{\langle x_1, x_2 \rangle \in X} \max(0, \delta - \cos(g(x_1), g(x_2)) + \cos(g(x_1), g(t_1))) + \max(0, \delta - \cos(g(x_1), g(x_2)) + \cos(g(x_2), g(t_2)))$$

Wieting et al. (2015)

Training Sentence Encodings with **Textual Entailment**

Textual Entailment/Natural Language Inference

Entailment: A => B

I went to the barber to get a haircut. I have shorter hair.

Contradiction: A and (not B)

I went to the barber to get a haircut. I have longer hair.

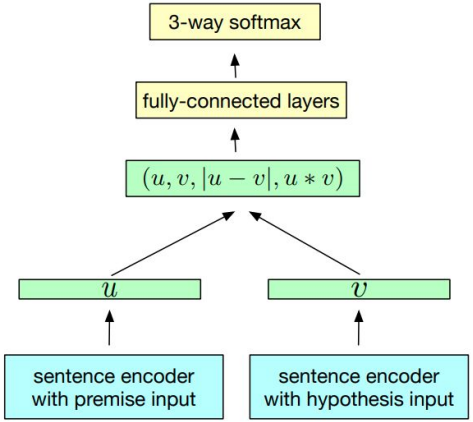
Neutral: nothing we can say

I went to the barber to get a haircut. Whales are mammals.

Training Sentence Encodings with **Textual Entailment**

InferSent Conneau et al. (2018)

- Train model on large amounts of NLI data
  - 3-way classification task
  - Datasets: SNLI (Bowman et al. 2015), MNLI (Williams et al. 2017)



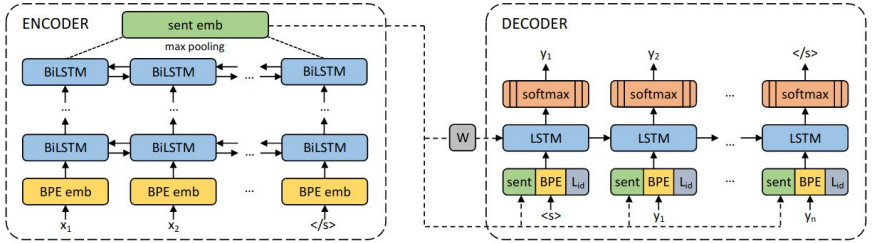
- Good results on many tasks
  - At the time

Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	SICK-R	SICK-E	STS14
Unsupervised representation training (ordered sentences)										
FastSent	70.8	78.4	88.7	80.6	-	76.8	72.2/80.3	-	-	**.63/.64**
FastSent+AE	71.8	76.7	88.8	81.5	-	80.4	71.2/79.1	-	-	.62/.62
SkipThought	76.5	80.1	93.6	87.1	82.0	**92.2**	**73.0/82.0**	**0.858**	82.3	.29/.35
SkipThought-LN	**79.4**	**83.1**	**93.7**	**89.3**	82.9	88.4	-	**0.858**	79.5	.44/.45
Supervised representation training										
CaptionRep (bow)	61.9	69.3	77.4	70.8	-	72.2	73.6/81.9	-	-	.46/.42
DictRep (bow)	76.7	78.7	90.7	87.2	-	81.0	68.4/76.8	-	-	**.67/.70**
NMT En-to-Fr	64.7	70.1	84.9	81.5	-	82.8	69.1/77.1	-		.43/.42
Paragram-phrase	-	-	-	-	79.7	-	-	0.849	83.1	**.71**/ -
BiLSTM-Max (on SST)†	(*)	83.7	90.2	89.5	(*)	86.0	72.7/80.9	0.863	83.1	.55/.54
BiLSTM-Max (on SNLI)†	79.9	84.6	92.1	**89.8**	83.3	**88.7**	75.1/82.3	**0.885**	**86.3**	.68/.65
BiLSTM-Max (on AllNLI)†	**81.1**	**86.3**	**92.4**	**90.2**	**84.6**	88.2	**76.2/83.1**	**0.884**	**86.3**	**.70/.67**

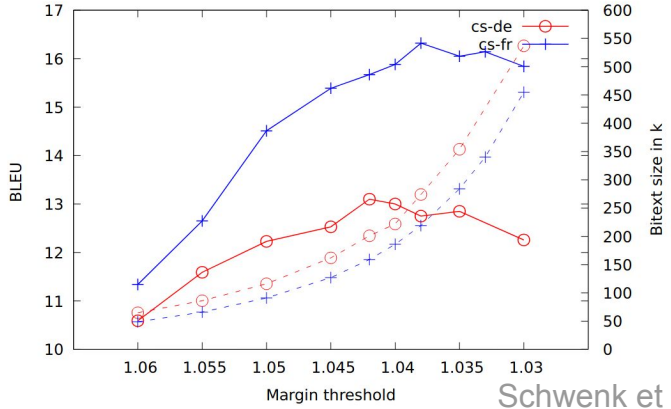
Training Sentence Encodings with **Machine Translation**

LASER

1. Train encoder-decoder machine translation system on many language pairs
2. Use embeddings output by encoder



Artexte & Schwenk (2018)



Schwenk et al. (2019)

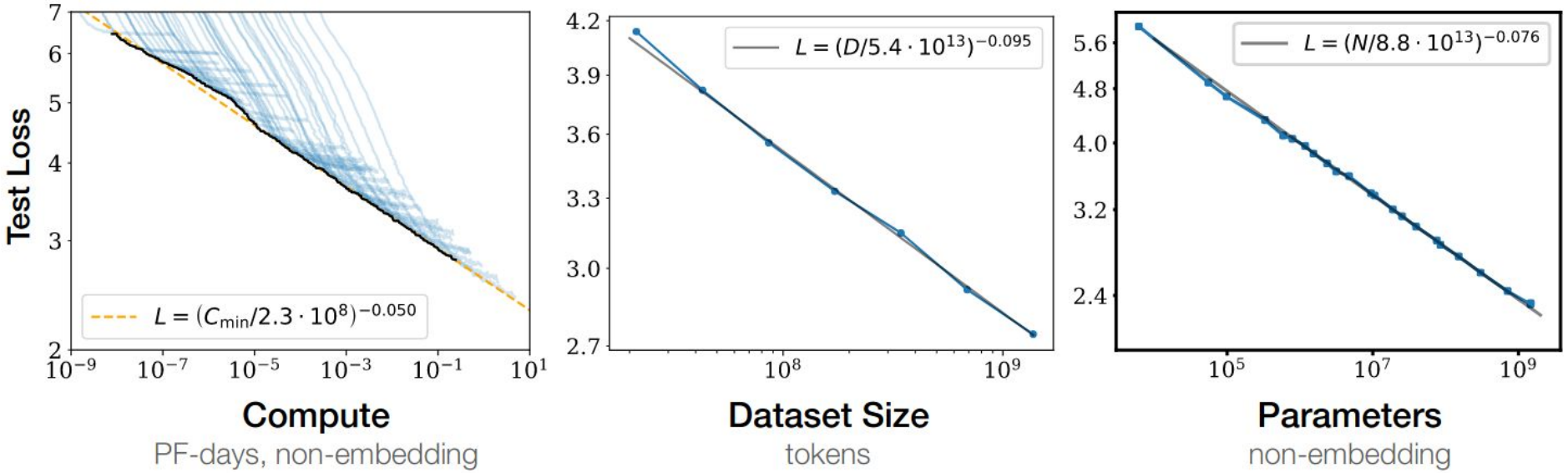
- Good multilingual encodings
- Used to mine parallel sentences for MT training

Frontiers in Text Representation Learning

Scaling Laws for NLP Models

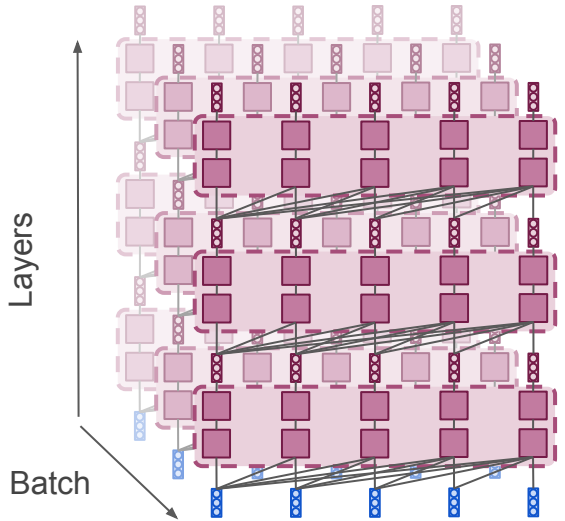
- More data is better
- More parameters are better
- Bigger batch sizes are better

Kaplan, McCandlish et al. (2020)



How to Train Large NLP Models

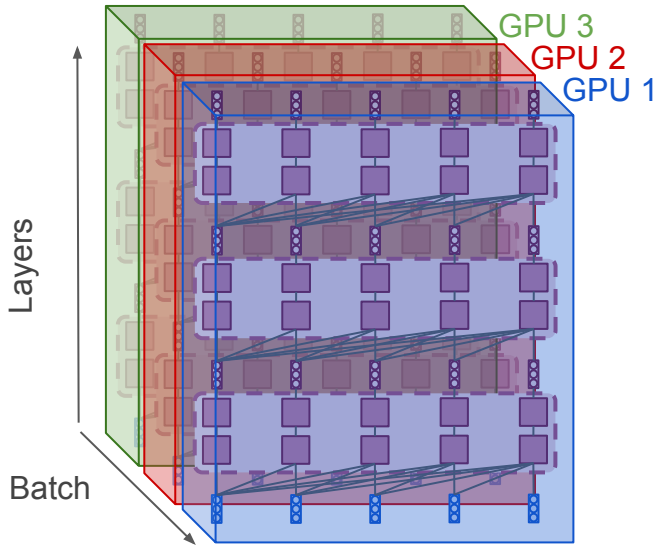
- How do we train a model that doesn’t fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

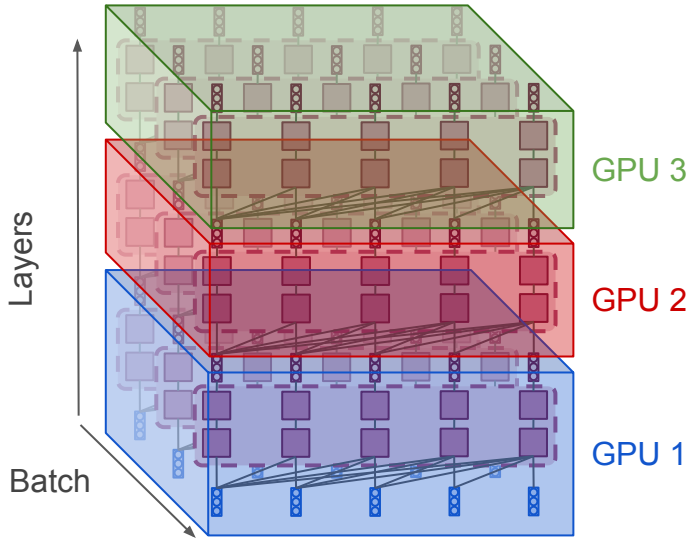
- Data parallelism
  - Split batch across each device
  - Copy model across devices
  - Good for larger batch sizes
  - Limited by model size



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

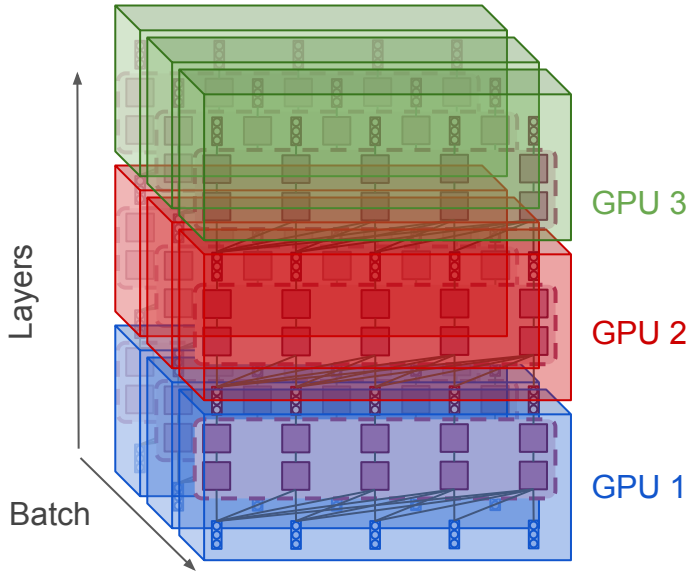
- Model parallelism
  - Split model across devices
  - Process batch sequentially
  - Good for large models
  - Need to copy data over from device to device
  - Sequential processing



Efficiency of Large Pre-trained Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

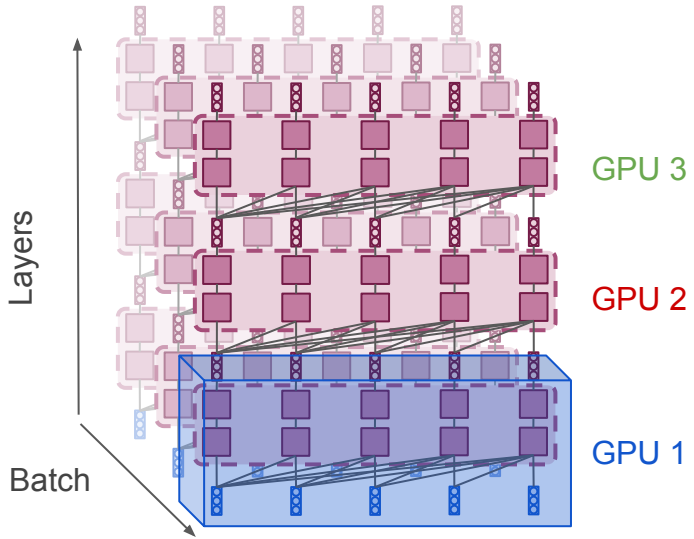
- Pipelining
  - Split model across devices
  - Process sub-batches sequentially
  - Avoids downtime



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

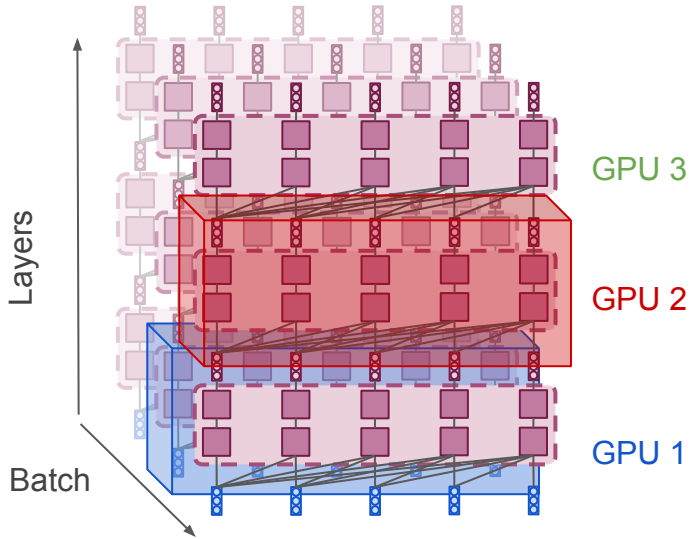
- Pipelining
  - Split model across devices
  - Process sub-batches sequentially
  - Avoids downtime



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

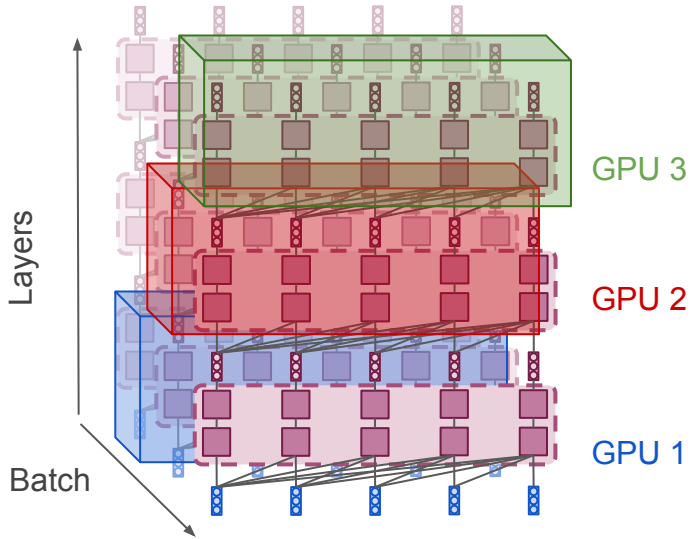
- Pipelining
  - Split model across devices
  - Process sub-batches sequentially
  - Avoids downtime



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

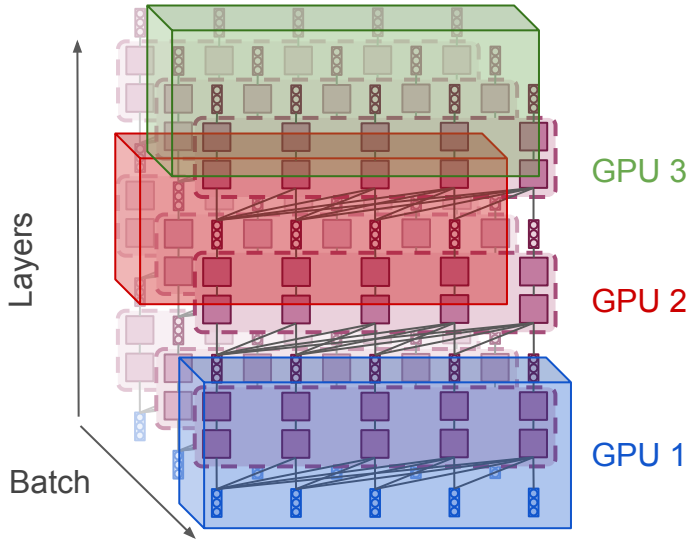
- Pipelining
  - Split model across devices
  - Process sub-batches sequentially
  - Avoids downtime



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

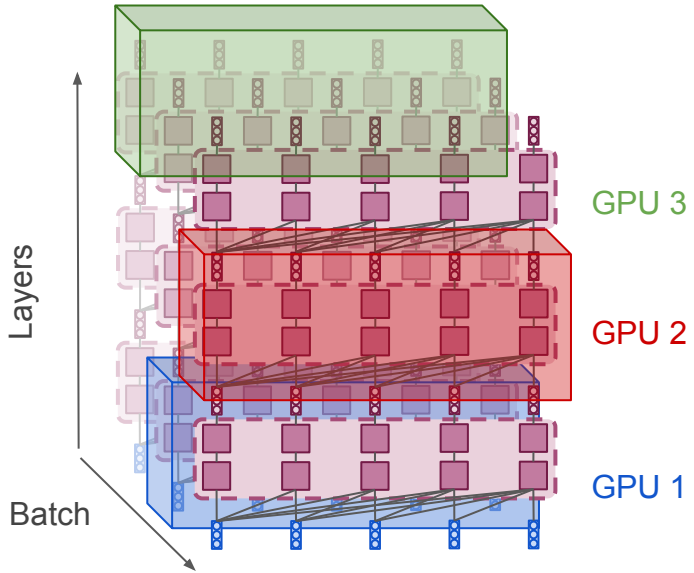
- Pipelining
  - Split model across devices
  - Process sub-batches sequentially
  - Avoids downtime



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

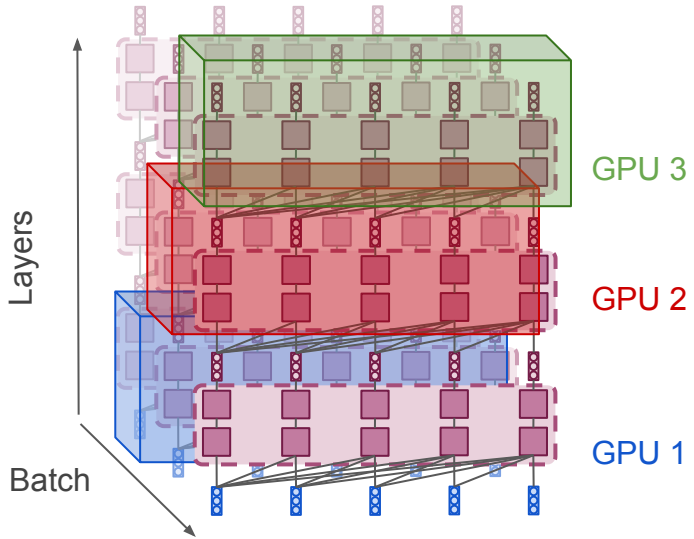
- Pipelining
  - Split model across devices
  - Process sub-batches sequentially
  - Avoids downtime



How to Train Large NLP Models

- How do we train a model that doesn't fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

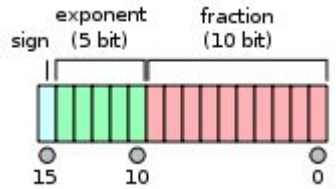
- Pipelining
  - Split model across devices
  - Process sub-batches sequentially
  - Avoids downtime



How to Train Large NLP Models

- How do we train a model that doesn’t fit on GPU memory?
  - BERT-large: 335M parameters
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters (~700GB in float32)

- 16-bit Floats
  - Halves memory costs
  - More instability (floating point error)
  - Needs supported device



Environmental Impact of Large NLP Models

- Training a large model has a significant financial and environmental cost
  - Not even considering hyperparameter search, tuning, etc…

Model	Hardware	Power (W)	Hours	kWh·PUE	CO_2e	Cloud compute cost
$\text{Transformer}_{base}$	P100x8	1415.78	12	27	26	$41–$140
Transformer_{big}	P100x8	1515.43	84	201	192	$289–$981
ELMo	P100x3	517.66	336	275	262	$433–$1472
BERT_{base}	V100x64	12,041.51	79	1507	1438	$3751–$12,571
BERT_{base}	TPUv2x16	—	96	—	—	$2074–$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	$942,973–$3,201,722
NAS	TPUv2x1	—	32,623	—	—	$44,055–$146,848
GPT-2	TPUv3x32	—	168	—	—	$12,902–$43,008

Table 3: Estimated cost of training a model in terms of CO_2 emissions (lbs) and cloud compute cost (USD).[7] Power and carbon footprint are omitted for TPUs due to lack of public information on power draw for this hardware.

Strubell et al. (2019)

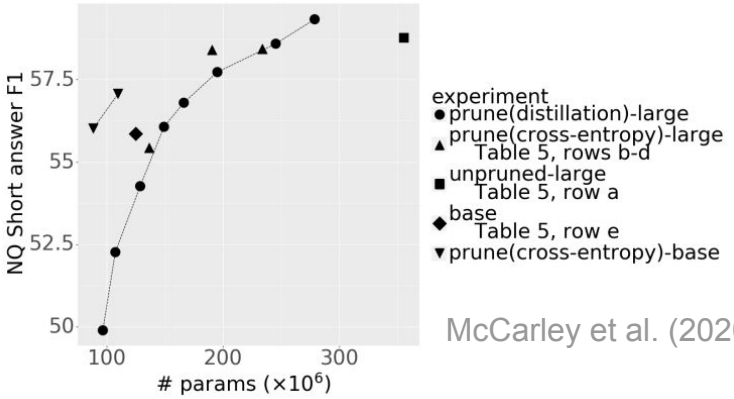
Avg person over 1 year in France: ~11,000

More Efficient Models: Pruning

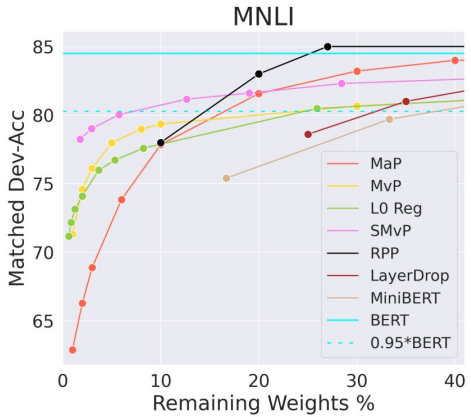
A significant number of parameters can be **removed after training**

- Structured Pruning
  - Remove dimensions, attention heads, etc…
  - Faster models
- Unstructured Pruning
  - Remove individual parameters
    - eg. parameters close to 0
  - Not necessarily faster
  - Allows for much higher percentage

Structured pruning of RoBERTa on question answering



McCarley et al. (2020)



Unstructured pruning of BERT (NLI)

Sanh et al. (2020)

More Efficient Models: Distillation

Train a **smaller** model to imitate a big model

- Better performance than small model trained from scratch
- Smaller model can be fine-tuned

		Compression	Performance	Speedup	Model	Evaluation
	BERT-base (Devlin et al., 2019)	×1	100%	×1	$BERT_{12}$	All GLUE tasks, SQuAD
	BERT-small	×3.8	91%	-	$BERT_4$†	All GLUE tasks
Distillation	DistilBERT (Sanh et al., 2019a)	×1.5	90%§	×1.6	$BERT_6$	All GLUE tasks, SQuAD
	$BERT_6$-PKD (Sun et al., 2019a)	×1.6	98%	×1.9	$BERT_6$	No WNLI, CoLA, STS-B; RACE
	$BERT_3$-PKD (Sun et al., 2019a)	×2.4	92%	×3.7	$BERT_3$	No WNLI, CoLA, STS-B; RACE
	Aguilar et al. (2019), Exp. 3	×1.6	93%	-	$BERT_6$	CoLA, MRPC, QQP, RTE
	BERT-48 (Zhao et al., 2019)	×62	87%	×77	$BERT_{12}$*†	MNLI, MRPC, SST-2
	BERT-192 (Zhao et al., 2019)	×5.7	93%	×22	$BERT_{12}$*†	MNLI, MRPC, SST-2
	TinyBERT (Jiao et al., 2019)	×7.5	96%	×9.4	$BERT_4$†	No WNLI; SQuAD
	MobileBERT (Sun et al., 2020)	×4.3	100%	×4	$BERT_{24}$†	No WNLI; SQuAD
	PD (Turc et al., 2019)	×1.6	98%	×2.5‡	$BERT_6$†	No WNLI, CoLA and STS-B
	WaLDORf (Tian et al., 2019)	×4.4	93%	×9	$BERT_8$†‖	SQuAD
	MiniLM (Wang et al., 2020b)	×1.65	99%	×2	$BERT_6$	No WNLI, STS-B, $MNLI_{mm}$; SQuAD
	MiniBERT(Tsai et al., 2019)	×6**	98%	×27**	$mBERT_3$†	CoNLL-18 POS and morphology
	BiLSTM-soft (Tang et al., 2019)	×110	91%	×434‡	$BiLSTM_1$	MNLI, QQP, SST-2

TinyBERT (Jiao et al., 2019)

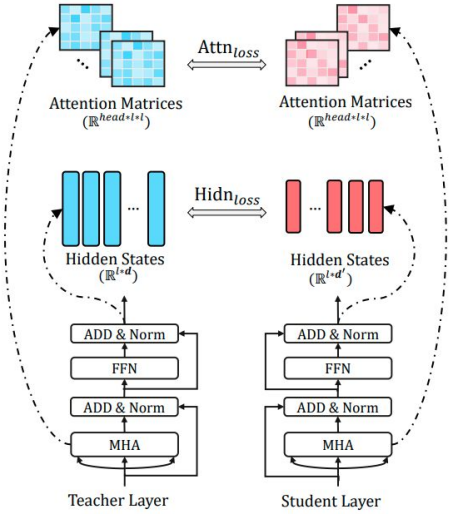
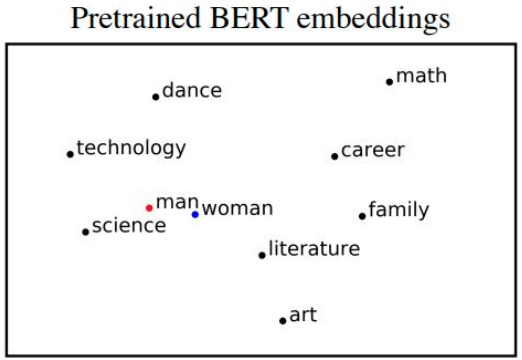


Table from Rogers et al. (2020)

Bias in Large Pre-trained Models

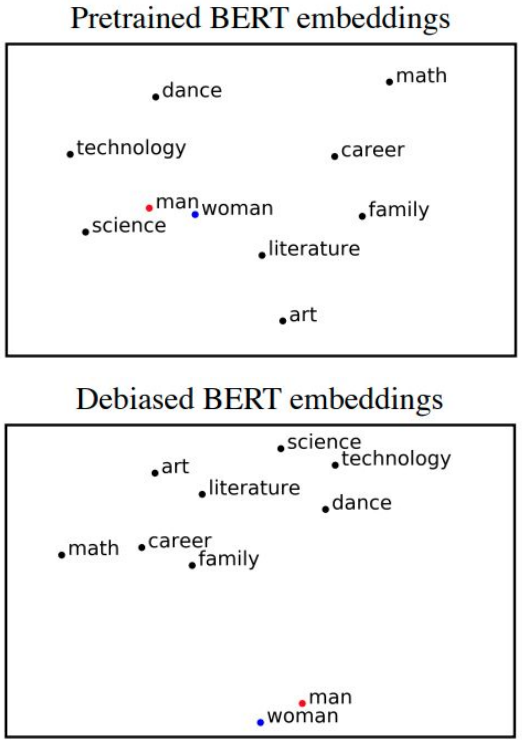
- Large models are trained on biased dataset
  - Internet, reddit, books, etc…
- Leads to biased representations
  - Eg. Gender bias, racial bias, etc.



Bias in Large Pre-trained Models

- Large models are trained on biased dataset
  - Internet, reddit, books, etc…
- Leads to biased representations
  - Eg. Gender bias, racial bias, etc.

- Some active work on debiasing large NLP models
  - Post-hoc debiasing



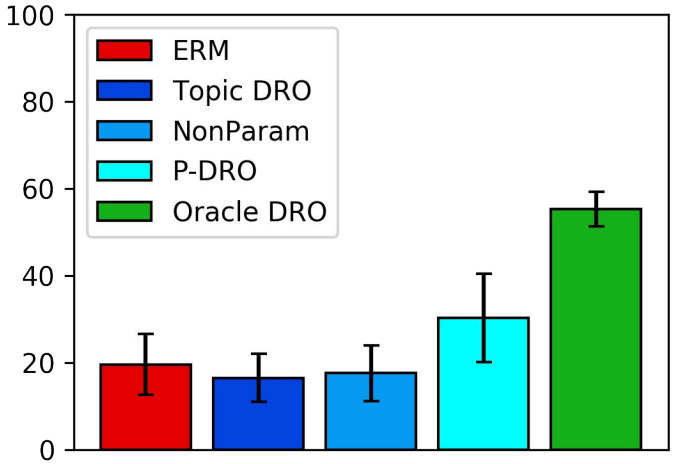
Liang et al. (2020)

Bias in Large Pre-trained Models

- Large models are trained on biased dataset
  - Internet, reddit, books, etc…
- Leads to biased representations
  - Eg. Gender bias, racial bias, etc.

- Some active work on debiasing large NLP models
  - Post-hoc debiasing
  - Robustness to demographic shifts

Worst accuracy among demographic groups in Toxicity Detection task



Michel et al. (2021)

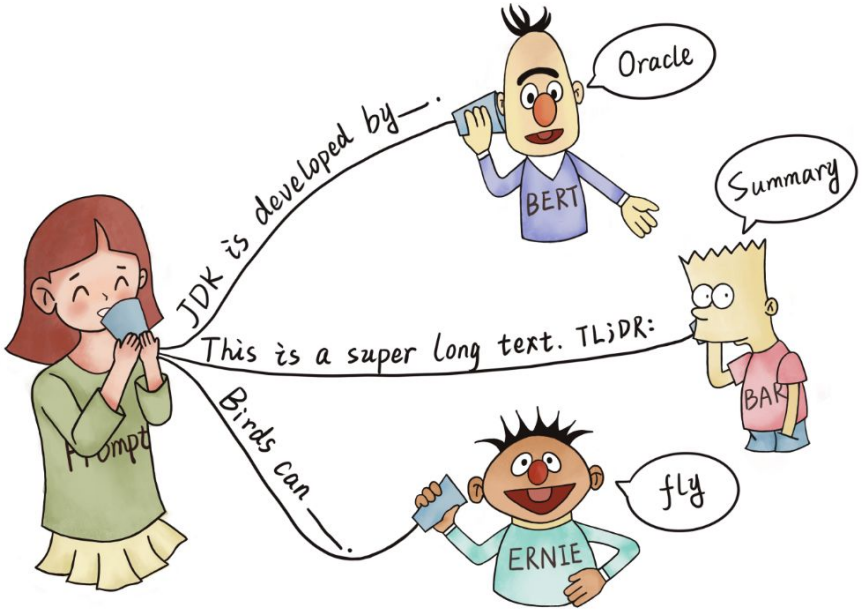
Beyond Representation Learning

Do we **need** to frame tasks as functions of text representations?

Natural Language as an Interface

Tasks can be reformulated as **natural language queries/statements**

Natural language statements can be evaluated by **language models**



Drawing from Liu et al. (2021)

Natural Language as an Interface: BERT

Reformulate task as cloze test

(Taylor, 1954)

Find the missing word in a sentence

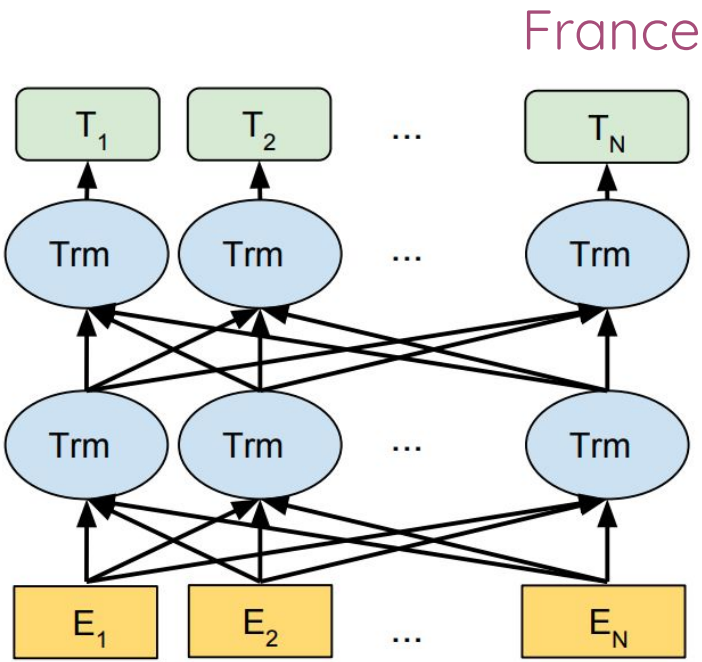


Figure from Devlin et al. (2018)

Natural Language as an Interface: T5

Reformulate tasks as sequence to sequence prediction

Raffel et al. (2020)

Natural Language as an Interface: GPT-3

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese =>                           ← prompt
```

One-shot

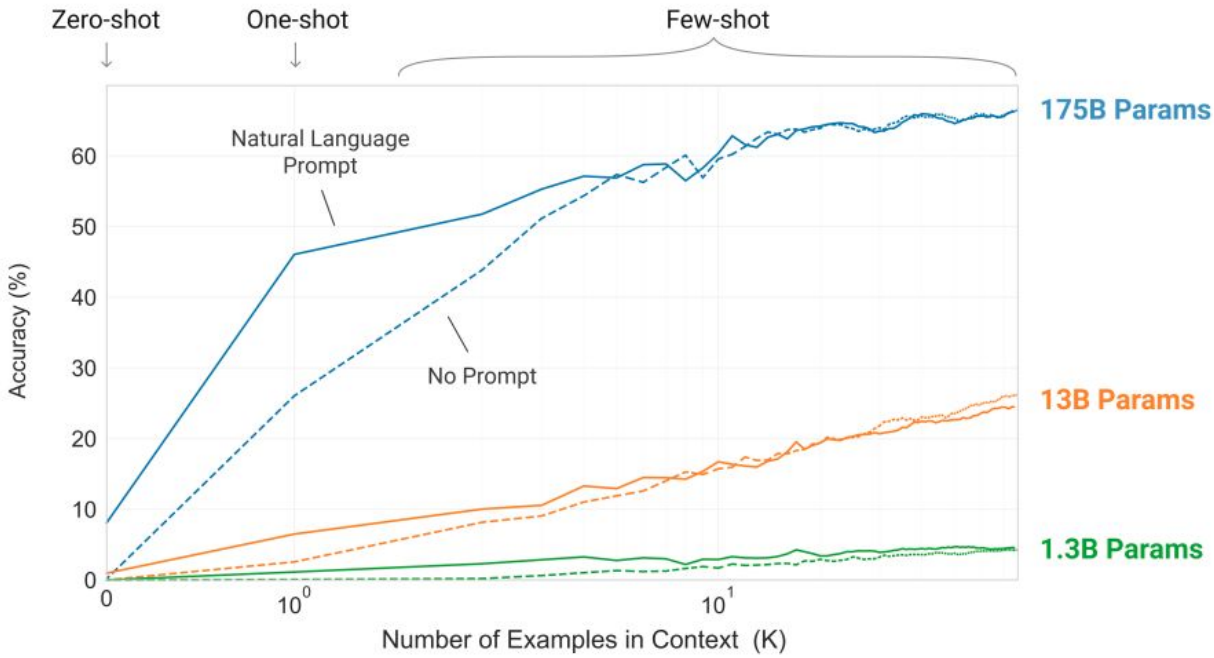
In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                           ← prompt
```



Brown et al. (2020)

Prompt/Answer Tuning

Find the optimal prompt/prefix for a given task and model

- Prompt mining

Prompts	
manual	`DirectX` *is developed by* y_{man}
mined	y_{mine} *released the* `DirectX`
paraphrased	`DirectX` *is created by* y_{para}

Top 5 predictions and log probabilities

	y_{man}		y_{mine}		y_{para}	
1	`Intel`	-1.06	`Microsoft`	-1.77	`Microsoft`	-2.23
2	`Microsoft`	-2.21	`They`	-2.43	`Intel`	-2.30
3	`IBM`	-2.76	`It`	-2.80	`default`	-2.96
4	`Google`	-3.40	`Sega`	-3.01	`Apple`	-3.44
5	`Nokia`	-3.58	`Sony`	-3.19	`Google`	-3.45

Jiang et al. (2020)

Prompt/Answer Tuning

Find the optimal prompt/prefix for a given task and model

- Prompt mining
- Prompting + fine-tuning

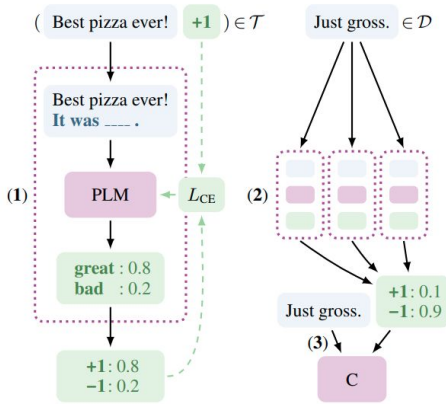


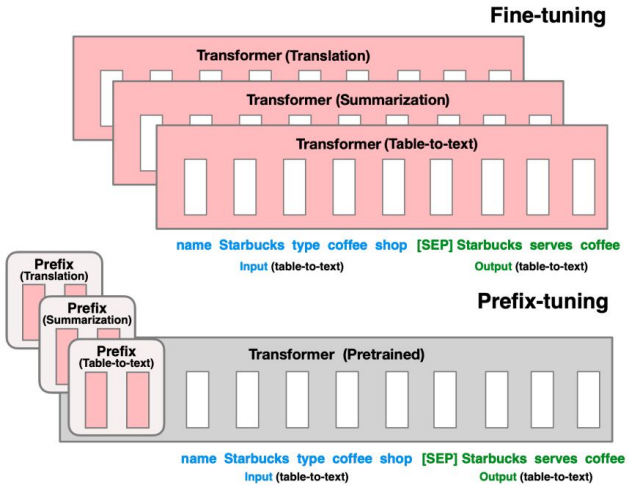
Figure 1: PET for sentiment classification. **(1)** A number of patterns encoding some form of task description are created to convert training examples to cloze questions; for each pattern, a pretrained language model is finetuned. **(2)** The ensemble of trained models annotates unlabeled data. **(3)** A classifier is trained on the resulting soft-labeled dataset.

Schick & Schütze (2021)

Prompt/Answer Tuning

Find the optimal prompt/prefix for a given task and model

- Prompt mining
- Prompting + fine-tuning
- Continuous prompt tuning



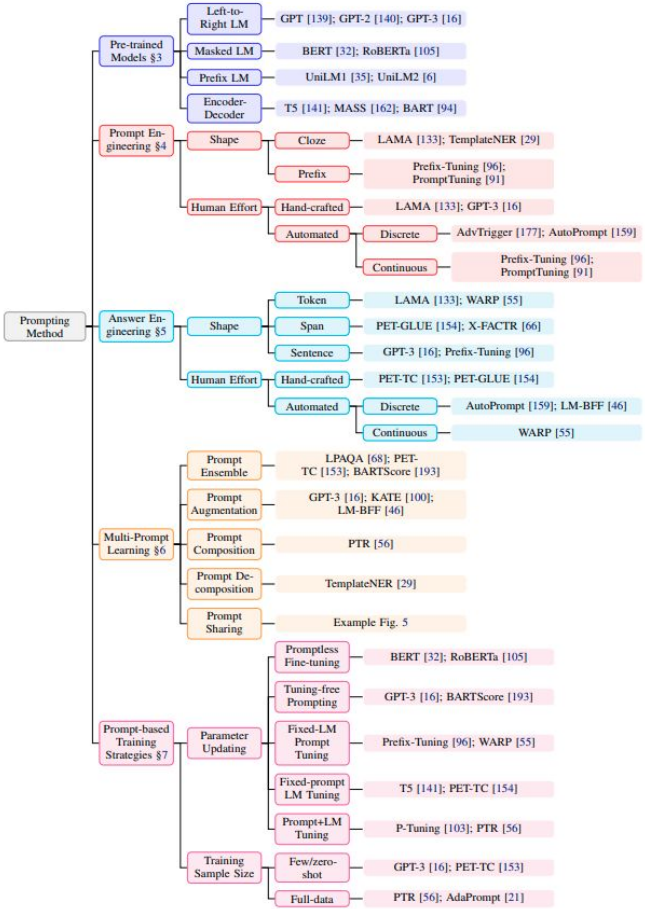
Li & Liang (2021)

Prompt/Answer Tuning

Find the optimal prompt/prefix for a given task and model

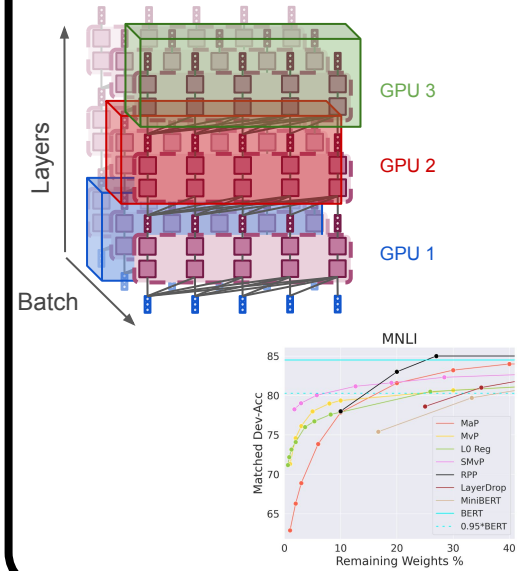
- Prompt mining
- Prompting + fine-tuning
- Continuous prompt tuning
- …

Very active research area!

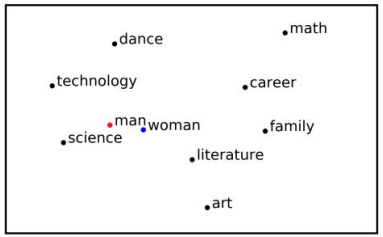


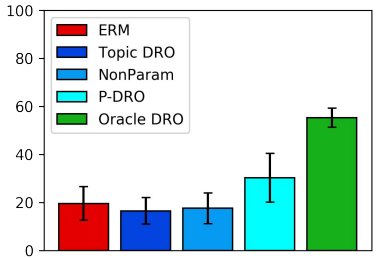
Conclusion

Computational efficiency

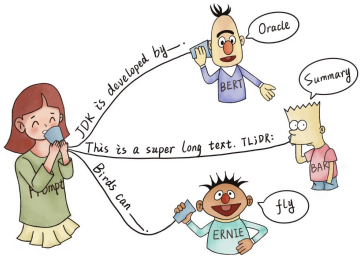


Bias and Fairness





Prompting



T5

Many more! Low resource languages, domain shift, adversarial attacks… stay tuned!