

**Transformer-based Encoders.**  
Masked language models based on the  
Transformer architecture.  
BERT and related models.

# Agenda: BERT and related models

## Tokenization strategies

- *BPE*
- *WordPiece*
- *SentencePiece*

# Agenda: BERT and related models

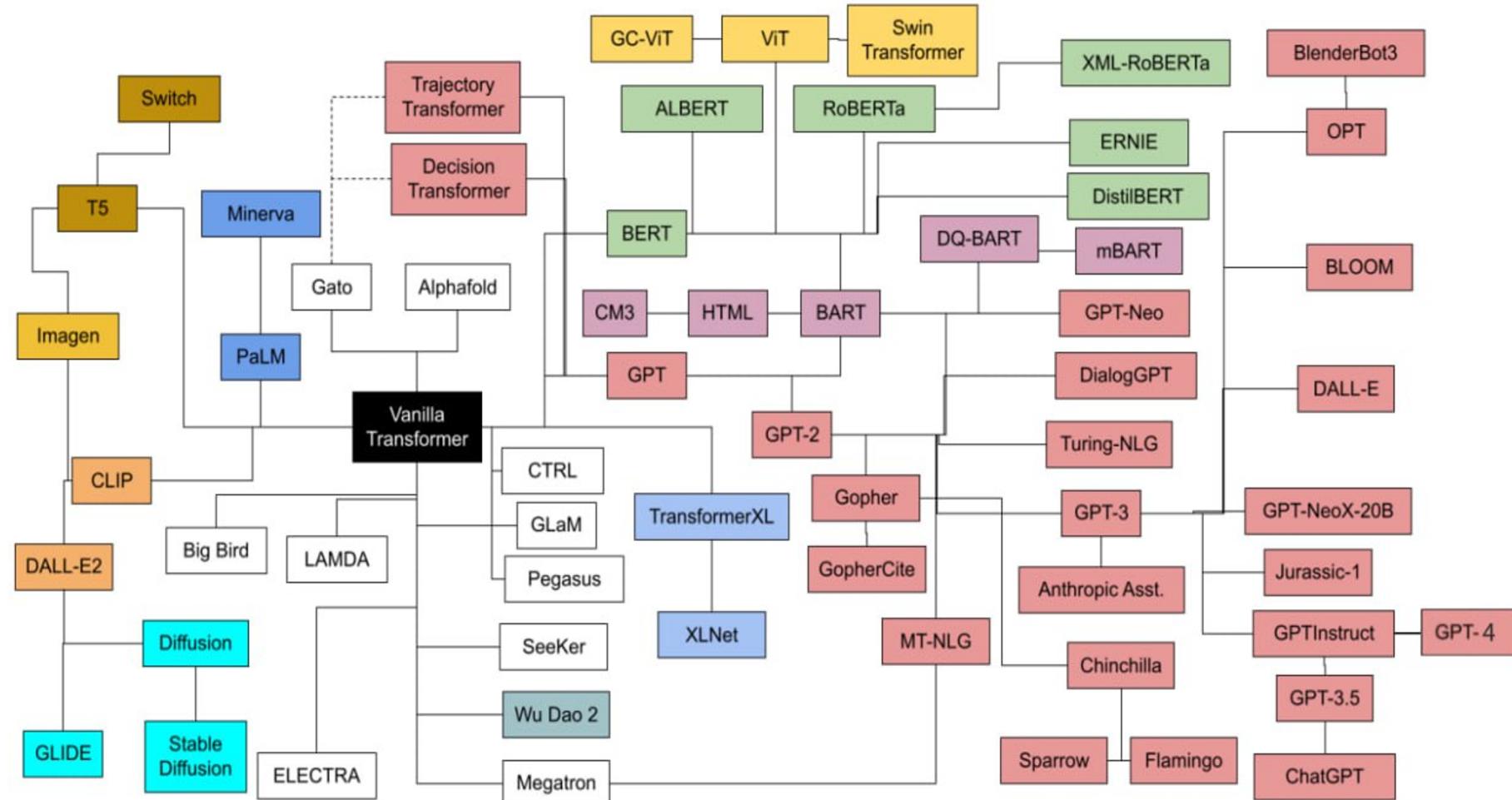
## Tokenization strategies

- *BPE*
- *WordPiece*
- *SentencePiece*

## Popular encoder-based models:

- *BERT, XLNet*
- *RoBERTa*
- *XLM-R*

# Transformer model Zoo



Transformer Family Tree

# Options for a pre-trained model

- The pretrained models vary by many parameters
  - *training corpus*
  - *vocabulary*
  - *pretraining task*
  - *architecture and model size*
  - *training hyperparameters*

# Options for a pre-trained model

- The pretrained models vary by many parameters
  - *training corpus*
  - *vocabulary*
  - *pretraining task*
  - *architecture and model size*
  - *training hyperparameters*
- How to choose the best pretrained model for a downstream task?
  - *Mostly experimentally*
  - *By similarity of pretraining corpus and task to the target problem*
  - *By performance on related tasks (e.g. <https://paperswithcode.com>)*

# Tokenization strategies

# How to split a text to pieces

An extreme: traditional linguistically-motivated tokenization ( “*by word*” )

- *It leads to unlimited vocabularies*
  - *A typical transformer model has limited vocabulary*
    - *It has to learn the matrix of input embeddings (but this can be fixed with e.g. FastText)*
    - *It has to predict distribution of the next (or missing) word*
  - *Out-of-vocabulary words have to be replaced with e.g. <UNK> token*
- *Many words are very rare => difficult to learn good representations*

# How to split a text to pieces

An extreme: traditional linguistically-motivated tokenization ( “by word” )

- *It leads to unlimited vocabularies*
  - *A typical transformer model has limited vocabulary*
    - *It has to learn the matrix of input embeddings (but this can be fixed with e.g. FastText)*
    - *It has to predict distribution of the next (or missing) word*
    - *Out-of-vocabulary words have to be replaced with e.g. <UNK> token*
  - *Many words are very rare => difficult to learn good representations*

Another extreme: split by character (or even by Unicode byte)

- *Small vocabularies (not for all scripts), but very long texts*
- *The model has to recover word meaning from very small units, which is difficult*

```
>>> list('cat'.encode('utf-8'))
[99, 97, 116]
>>> list('кот'.encode('utf-8'))
[208, 186, 208, 190, 209, 130]
>>> list('猫'.encode('utf-8'))
[232, 178, 147]
```

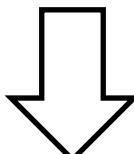
# How to split a text to pieces

An extreme: traditional linguistically-motivated tokenization ( “by word” )

- *It leads to unlimited vocabularies*
  - *A typical transformer model has limited vocabulary*
    - *It has to learn the matrix of input embeddings (but this can be fixed with e.g. FastText)*
    - *It has to predict distribution of the next (or missing) word*
    - *Out-of-vocabulary words have to be replaced with e.g. <UNK> token*
  - *Many words are very rare => difficult to learn good representations*

Another extreme: split by character (or even by Unicode byte)

- *Small vocabularies (not for all scripts), but very long texts*
- *The model has to recover word meaning from very small units, which is difficult*



```
>>> list('cat'.encode('utf-8'))
[99, 97, 116]
>>> list('кот'.encode('utf-8'))
[208, 186, 208, 190, 209, 130]
>>> list('猫'.encode('utf-8'))
[232, 178, 147]
```

**Compromise: subword embeddings**

# Byte-pair encoding

**The goal is compression:** encode the text corpus into the shortest possible sequence of tokens, with the fixed vocabulary size

BPE is a simple greedy algorithm:

- Start with the vocabulary of characters (or Unicode bytes)
- Merge the most frequent bigram into a new token
  - in practice, mergers across word boundaries are usually prohibited
  - space is often prepended to the next word
- Repeat for N times

**Result:**

- For frequent words, there are special tokens
- Rare words are represented by word pieces

```
vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
print(best)
```

---

r .	→	r.
l o	→	lo
lo w	→	low
e r.	→	er.

Figure 1: BPE merge operations learned from dictionary {‘low’, ‘lowest’, ‘newer’, ‘wider’}.

# WordPiece

Very similar to BPE

The difference: ***it is probabilistic***

- BPE chooses the pair  $A+B$  with maximal frequency (equivalently,  $P(A,B)$ )
- WordPiece chooses the pair with **maximal gain**:  $P(A,B) / [P(A) * P(B)]$ 
  - By doing this, it maximizes likelihood of a 1-gram language model on the corpus

# WordPiece

Very similar to BPE

The difference: ***it is probabilistic***

- BPE chooses the pair  $A+B$  with maximal frequency (equivalently,  $P(A,B)$ )
- WordPiece chooses the pair with **maximal gain**:  $P(A,B) / [P(A) * P(B)]$ 
  - By doing this, it maximizes likelihood of a 1-gram language model on the corpus

During inference, go from left to right, and pick the longest existing token on each step.

This algorithm is used in BERT

- Tokens that do not start a word are prepended with ##
  - E.g. *Мама мыла раму* => *M ##ама мы ##ла р ##ам ##у*

# SentencePiece

This algorithm does not use pre-tokenization

- *It makes the algorithm useful for scripts that do not use spaces (e.g. Japanese)*

It performs lossless tokenization (preserves spaces and other details)

- *Space is escaped as “\_” and treated as an ordinary character*

Uses BPE algorithm under the hood

# Is there the best tokenizer?

- Vocabulary size matters much more than the algorithm
- Vocabulary contents matters a lot
  - Most of the words might be useless for the target domain
- When you use a pre-trained model, the tokenizer comes with it 
- For a new model, **SentencePiece is a good idea**
  - *Reversible tokenization is a good idea for models that generate text*
  - *Subword regularization is a good idea with noisy texts*

BERT

# BERT

BERT = Bidirectional *Encoder* Representations from Transformers

*Presented in NAACL (June, 2019), best long paper award*

## **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**

**Jacob Devlin    Ming-Wei Chang    Kenton Lee    Kristina Toutanova**

Google AI Language

{jacobdevlin, mingweichang, kentonl, kristout}@google.com

[v1] Thu, 11 Oct 2018 00:50:01 UTC (227 KB)

[v2] Fri, 24 May 2019 20:37:26 UTC (309 KB)



# Results on the GLUE benchmark

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>

## MultiNLI

Premise: Hills and mountains are especially sanctified in Jainism.

Hypothesis: Jainism hates nature.

Label: Contradiction

## CoLa

Sentence: The wagon rumbled down the road.

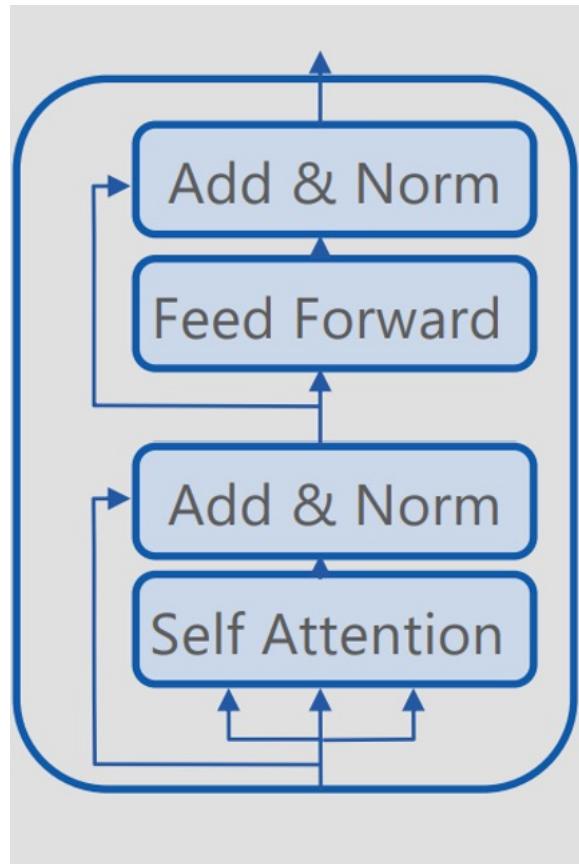
Label: Acceptable

Sentence: The car honked down the road.

Label: Unacceptable

# BERT architecture

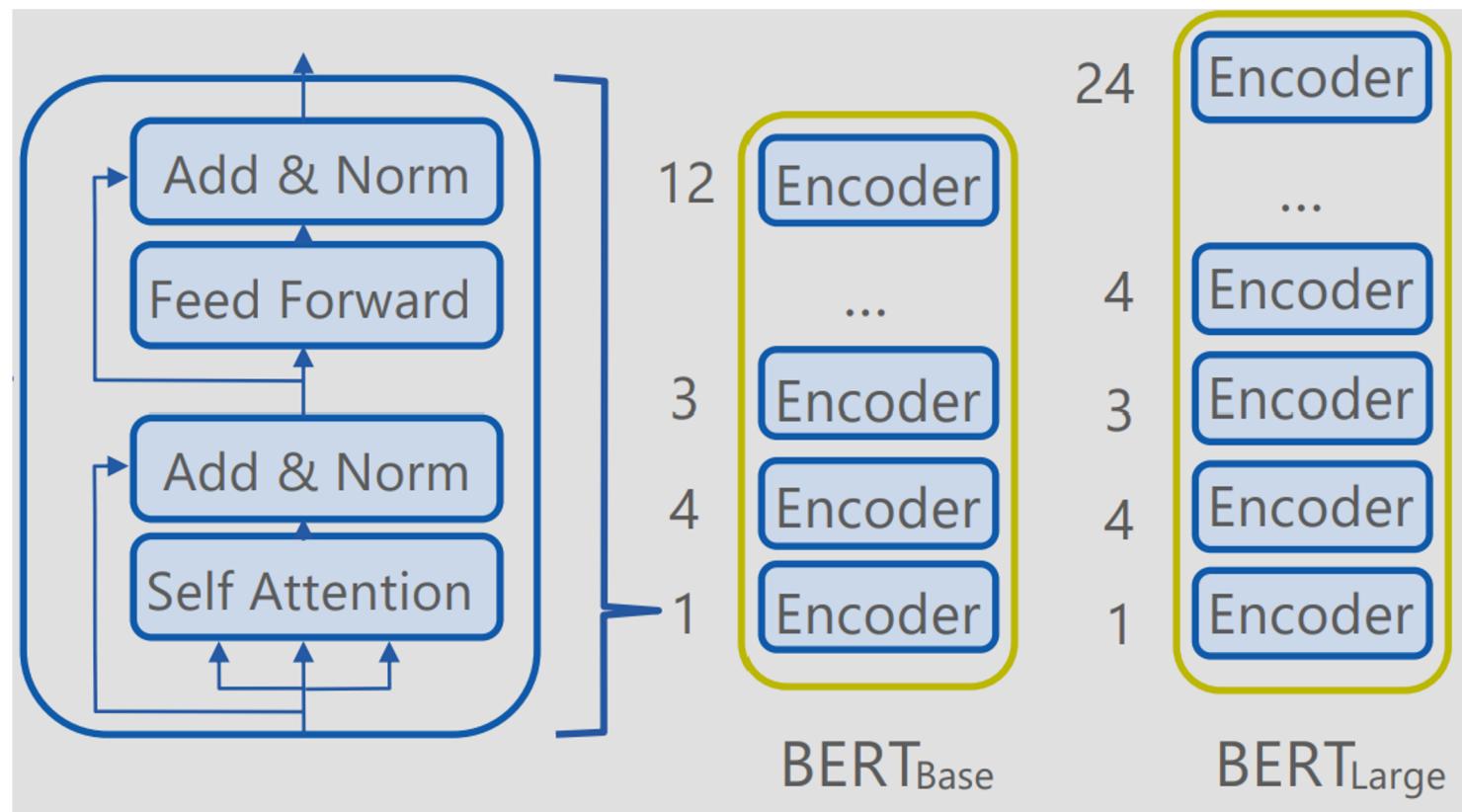
The model is based on **Encoder part** of Transformer



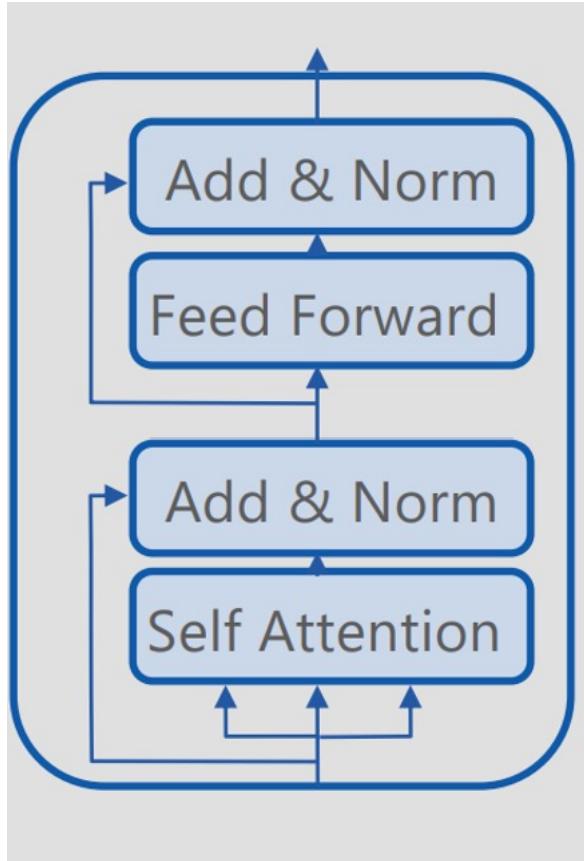
# BERT architecture

The model is based on **Encoder part** of Transformer

Two main configurations: base & large

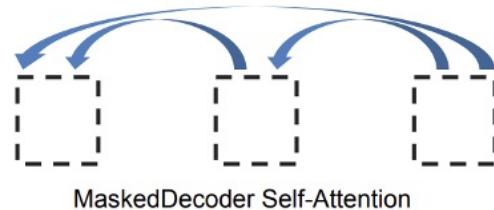
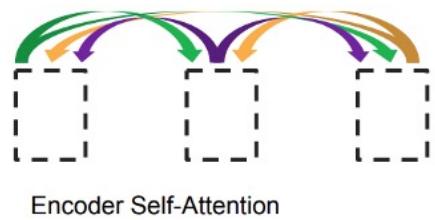


# Recap: What is the key difference between Encoder & Decoder?



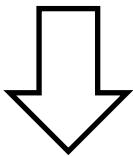
# Recap: What is the key difference between Encoder & Decoder?

**Answer:** in decoder we use Masked Self-Attention, which prohibits looking into the future, while in Encoder all tokens interact with each other.



# Recap: What is the key difference between Encoder & Decoder?

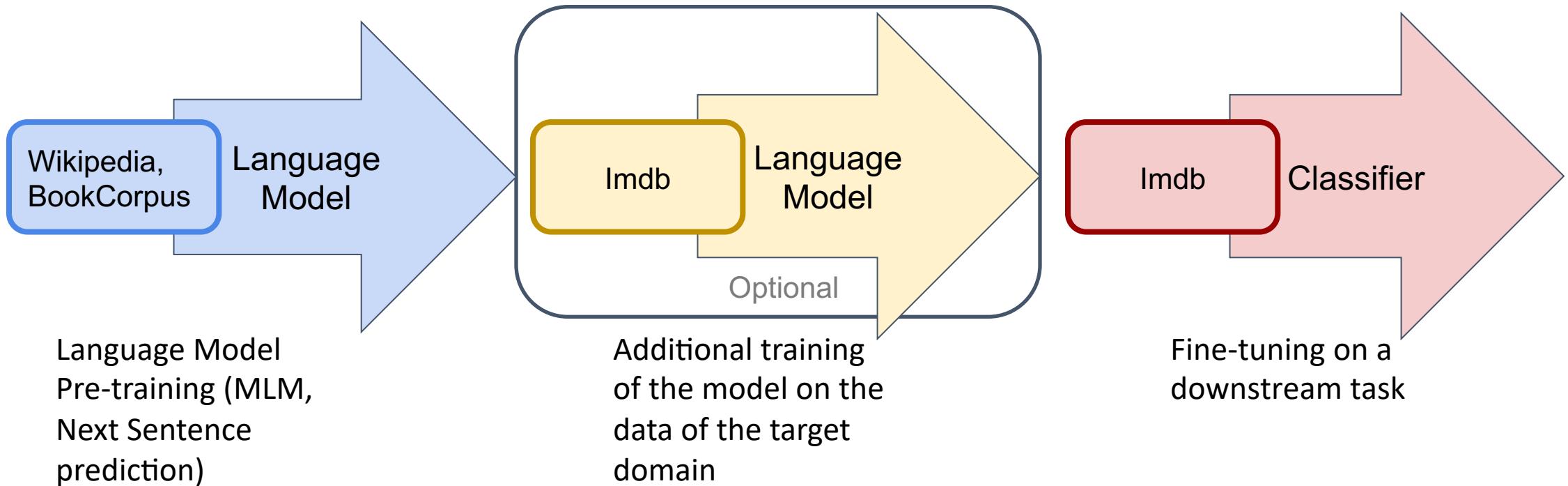
**Answer:** in decoder we use *Masked Self-Attention*, which prohibits looking into the future, while in Encoder *all tokens interact with each other.*



Being an Encoder BERT "looks" at all context tokens (left & right)



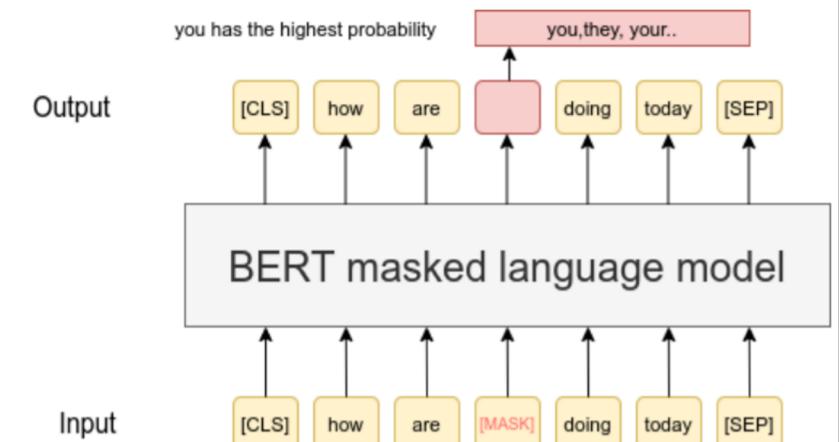
# General scheme for using BERT



# BERT pre-training

## Two pre-training tasks:

- Masked Language Modeling Task (**MLM**)
- Next Sentence Prediction (**NSP**)



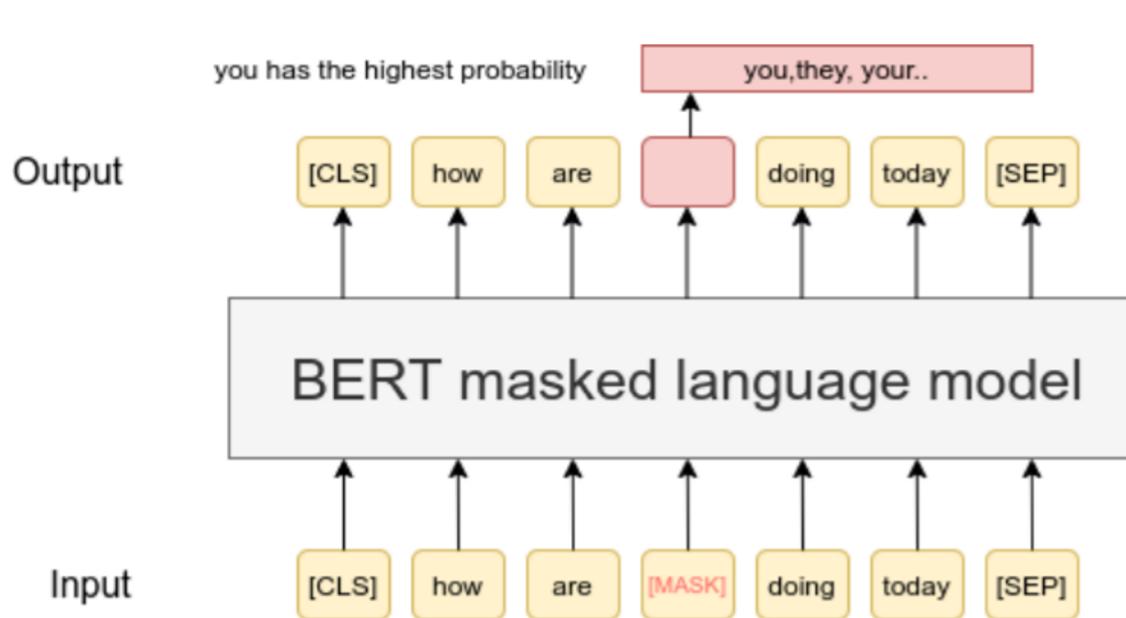
**Sentence A** = The man went to the store.  
**Sentence B** = He bought a gallon of milk.  
**Label** = IsNextSentence

**Sentence A** = The man went to the store.  
**Sentence B** = Penguins are flightless.  
**Label** = NotNextSentence

# BERT pre-training

Masked Language Modeling Task (**MLM**): mask k% of the input words and then predict the masked words (usually k=15%)

- *Too little masking: Too expensive to train*
- *Too much masking: Not enough content*



# Masked LM objective

Problem: don't want masking for downstream tasks, but only outputs from masked timesteps affect MLM loss  
=> bad representations for all other timesteps?

Solution: CE on (some) non-masked timesteps also

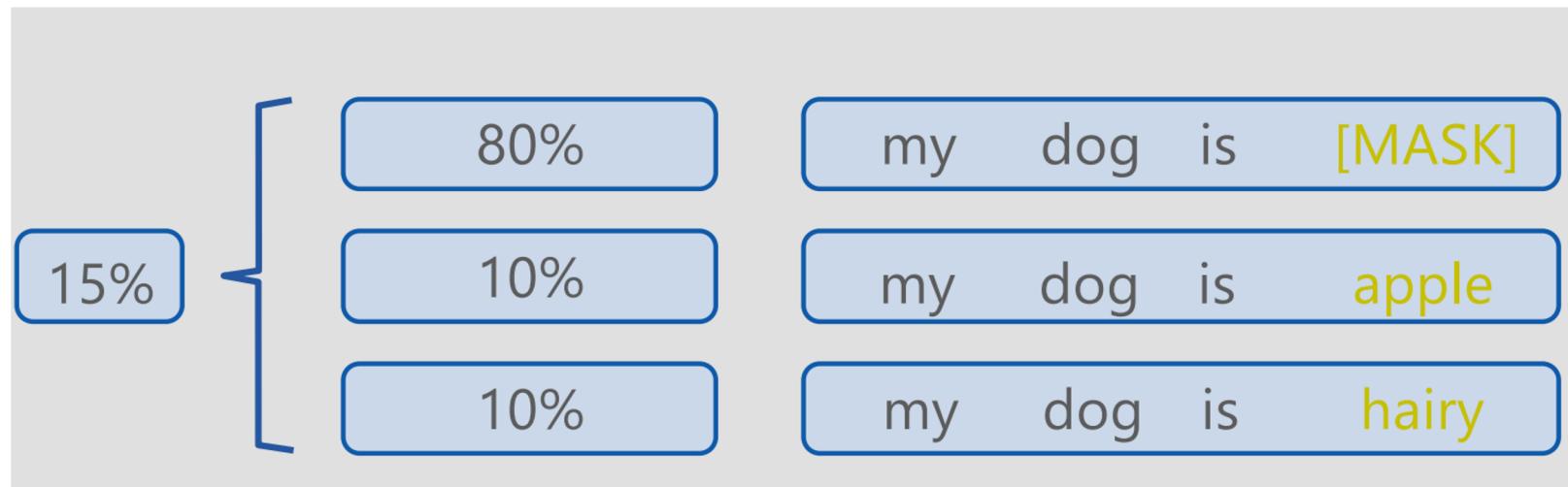
Problem: can learn simply to copy non-masked timesteps without even looking at context

Solution: replace non-masked timesteps with random words sometimes

This is **denoising autoencoder**: replace some tokens from a text fragment with random tokens / [MASK] token and try to reconstruct initial fragment!

# MLM Task

- 80 % of time, replace with [MASK]
- 10% of time, replace with random words
- 10% of time keep same



# Next Sentence Prediction (NSP) objective

Input 2 sentences: A,B; predict if B is next sentence after A

- *Sample B following A (with p=0.5) or random sentence from the corpus*
- *Learn to extract relations between 2 sentences (for tasks like paraphrase detection, NLI)*

NSP was shown to worsen results in the following research (too simple objective?)

- More difficult alternative: always sample 2 adjacent sentences and predict their order

**Sentence A** = The man went to the store.  
**Sentence B** = He bought a gallon of milk.  
**Label** = IsNextSentence

**Sentence A** = The man went to the store.  
**Sentence B** = Penguins are flightless.  
**Label** = NotNextSentence

# How to get sentence embedding from BERT?

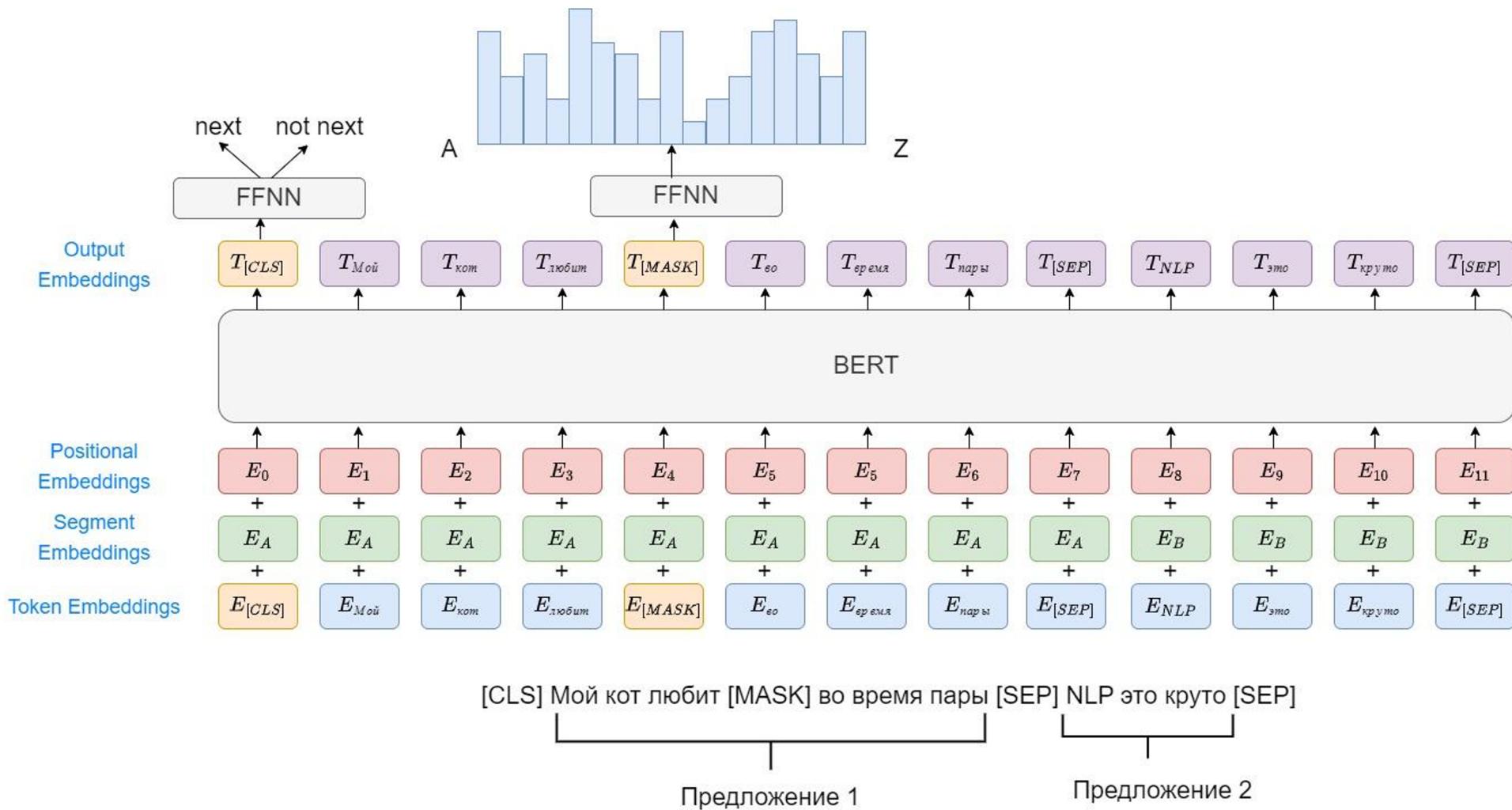
- [CLS] - CLS token embedding from the last layer
- [MEAN] - averaging the word vectors on the last layer
- [MAX] - componentwise maximum of word vectors on the last layer  
(maxpooling)

# Inputs

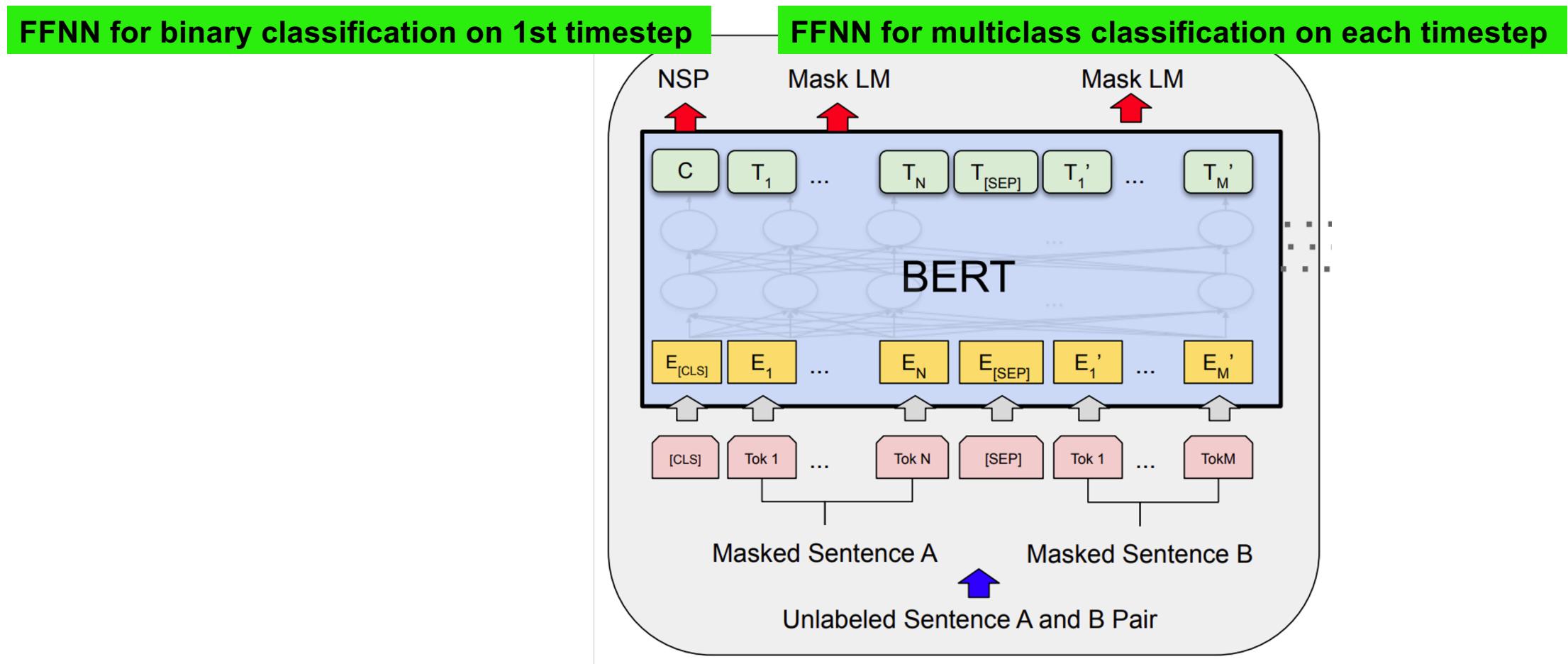
Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	# #ing	[SEP]
Token Embeddings	$E_{[CLS]}$	$E_{my}$	$E_{dog}$	$E_{is}$	$E_{cute}$	$E_{[SEP]}$	$E_{he}$	$E_{likes}$	$E_{play}$	$E_{##ing}$	$E_{[SEP]}$
Segment Embeddings	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_B$	$E_B$	$E_B$	$E_B$	$E_B$
Position Embeddings	$E_0$	$E_1$	$E_2$	$E_3$	$E_4$	$E_5$	$E_6$	$E_7$	$E_8$	$E_9$	$E_{10}$

- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

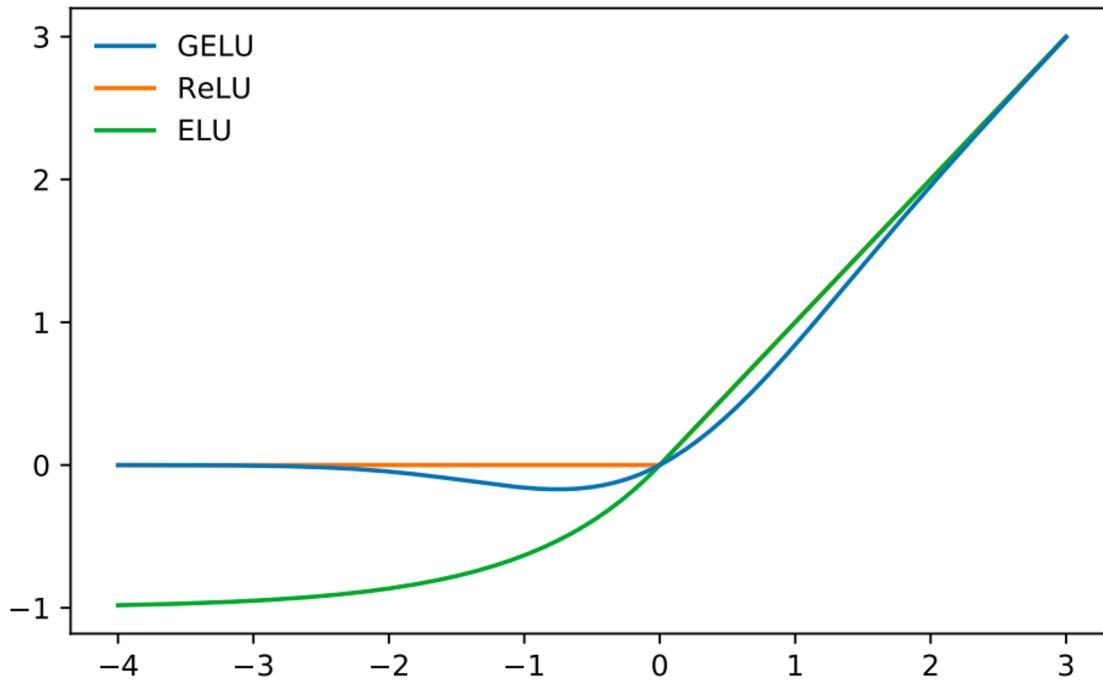
# MLM and NSP together



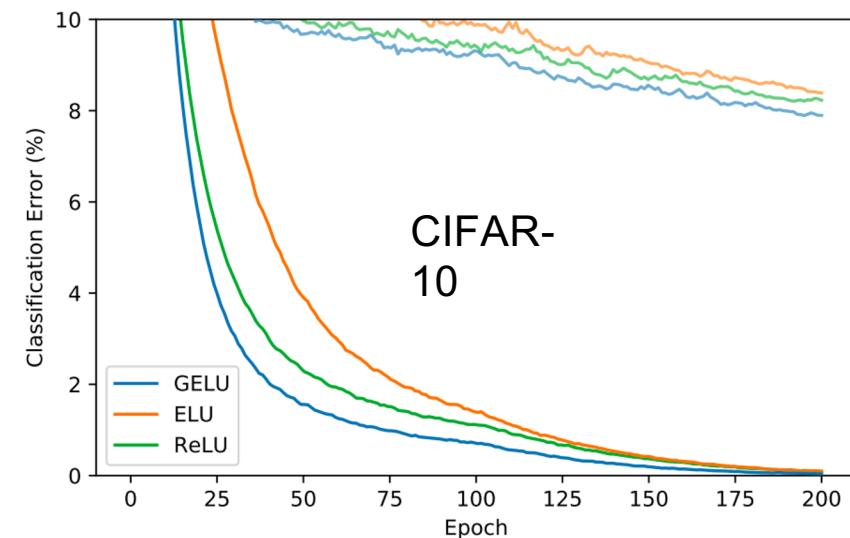
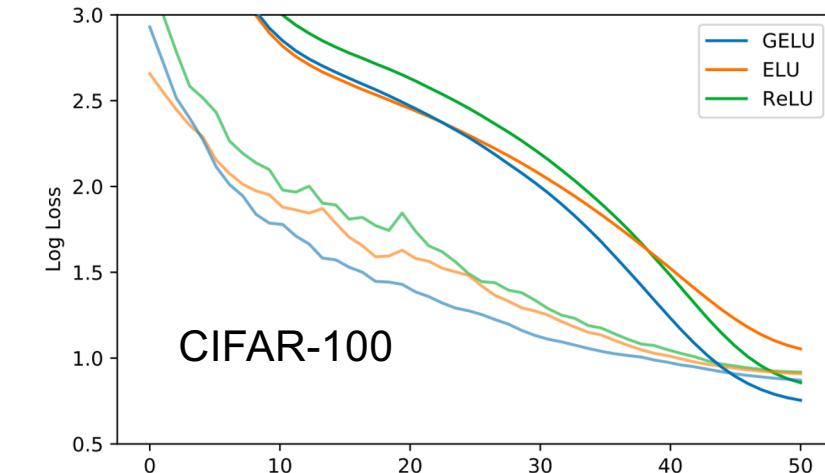
# BERT Pre-training



# GELU activation



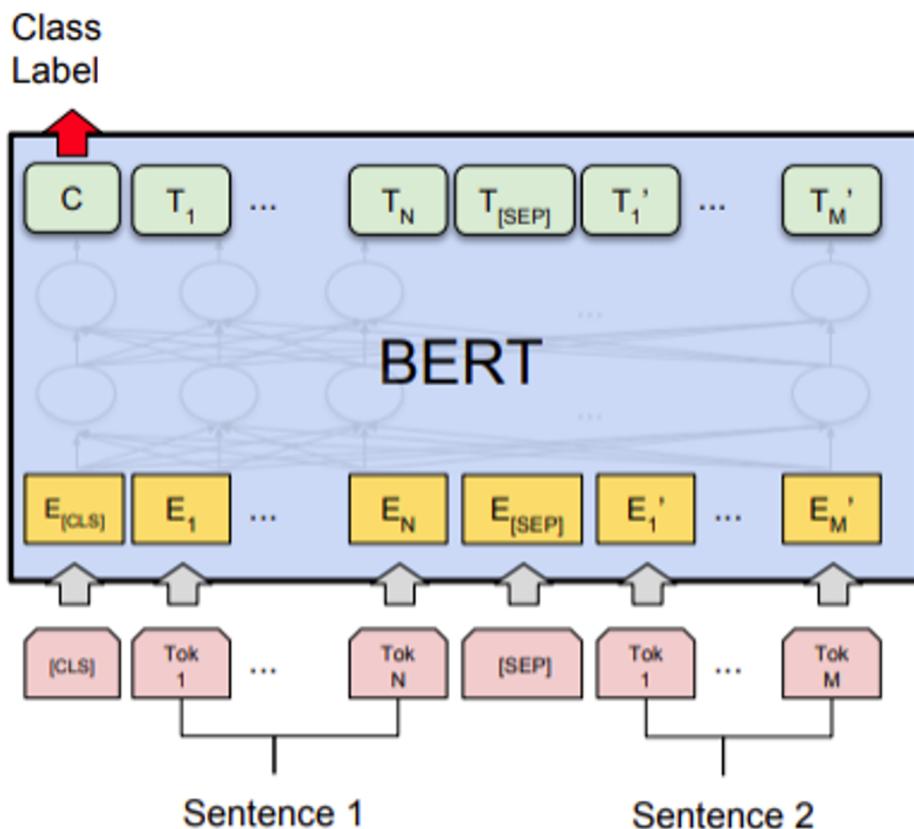
both GPT and BERT changed ReLU to GELU



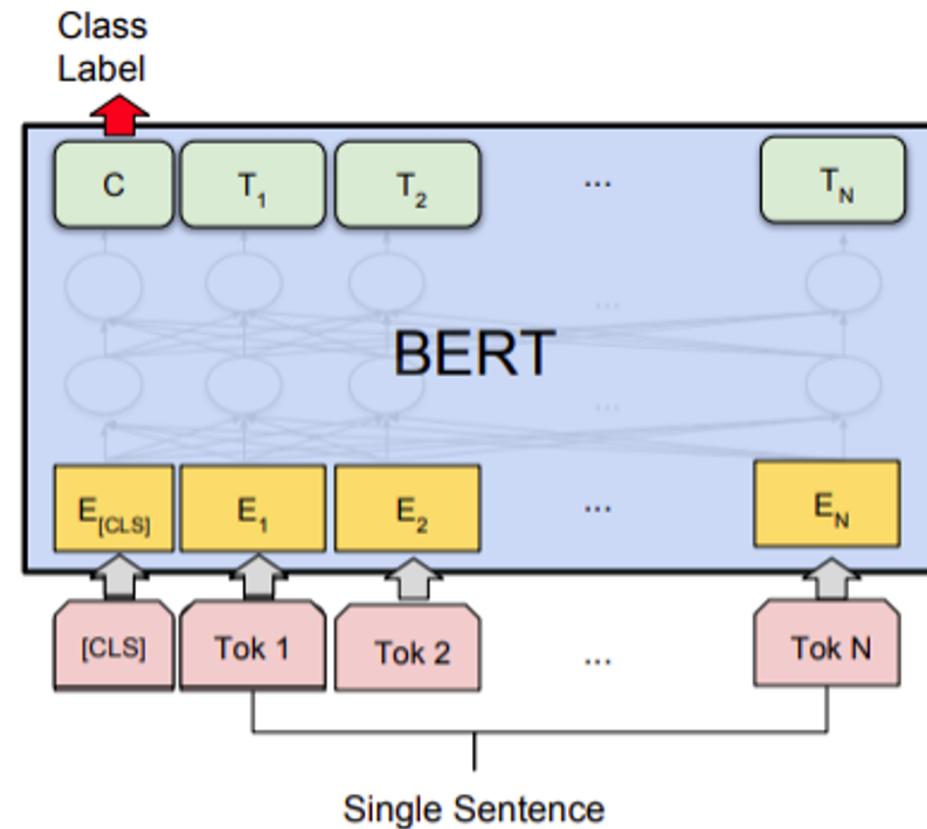
# BERT Pre-training details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences \* 128 length or 256 sequences \* 512 length)
- Training Time: 1M steps (~40 epochs) +10k steps lr warmup
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head 110M params
- BERT-Large: 24-layer, 1024-hidden, 16-head 340M params
- Trained on 4x4 or 8x8 TPU slice for 4 days

# BERT Fine-tuning

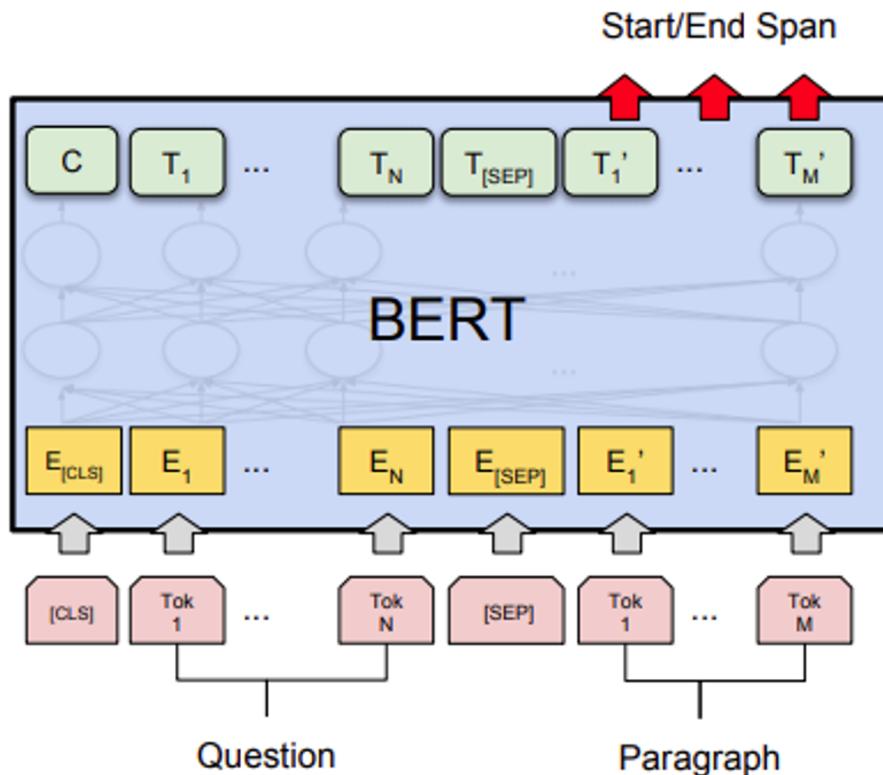


(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG

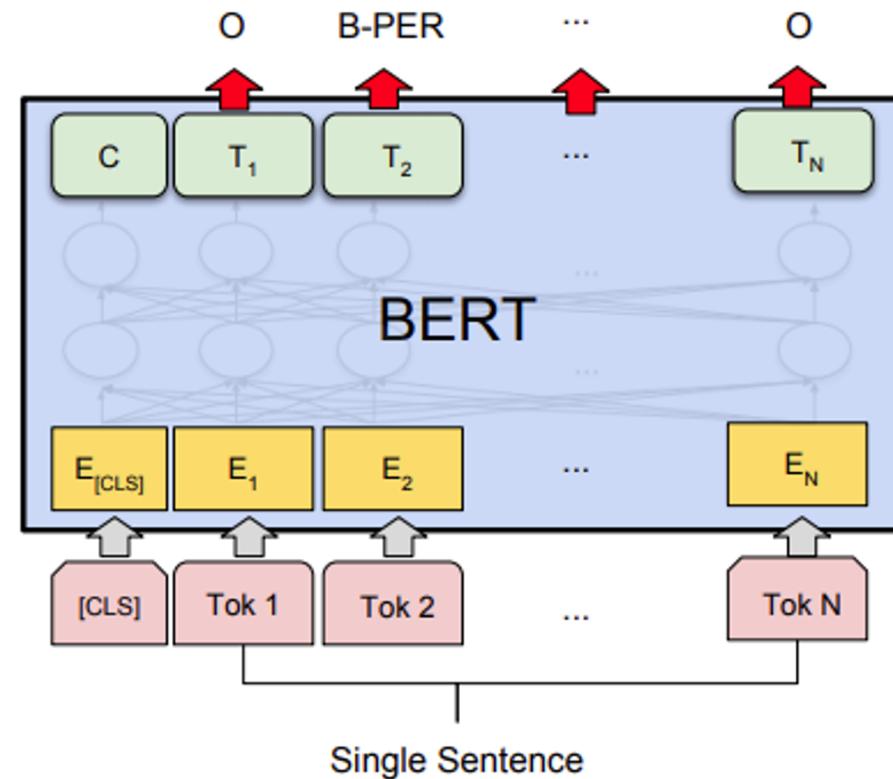


(b) Single Sentence Classification Tasks:  
SST-2, CoLA

# BERT Fine-tuning



(c) Question Answering Tasks:  
SQuAD v1.1



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

# Results on the GLUE benchmark

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>

## MultiNLI

Premise: Hills and mountains are especially sanctified in Jainism.

Hypothesis: Jainism hates nature.

Label: Contradiction

## CoLa

Sentence: The wagon rumbled down the road.

Label: Acceptable

Sentence: The car honked down the road.

Label: Unacceptable

# SQuAD 1.1 Results

What was another term used for the oil crisis?  
Ground Truth Answers: first oil shock shock shock first oil  
shock shock  
Prediction: shock

The 1973 oil crisis began in October 1973 when the members of the Organization of Arab Petroleum Exporting Countries (OAPEC, consisting of the Arab members of OPEC plus Egypt and Syria) proclaimed an oil embargo. By the end of the embargo in March 1974, the price of oil had risen from US\$3 per barrel to nearly \$12 globally; US prices were significantly higher. The embargo caused an oil crisis, or "shock", with many short- and long-term effects on global politics and the global economy. It was later called the "first oil shock", followed by the 1979 oil crisis, termed the "second oil shock."

- Only new parameters:  
Start vector and end vector.
- Softr

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar et al. '16)	82.304	91.221
1	BERT (ensemble) Google AI Language <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>	87.433	93.160
2	BERT (single model) Google AI Language <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>	85.083	91.835
2	nlnet (ensemble) Microsoft Research Asia	85.954	91.677
5	nlnet (single model) Microsoft Research Asia	83.468	90.133
3	QANet (ensemble) Google Brain & CMU	84.454	90.490

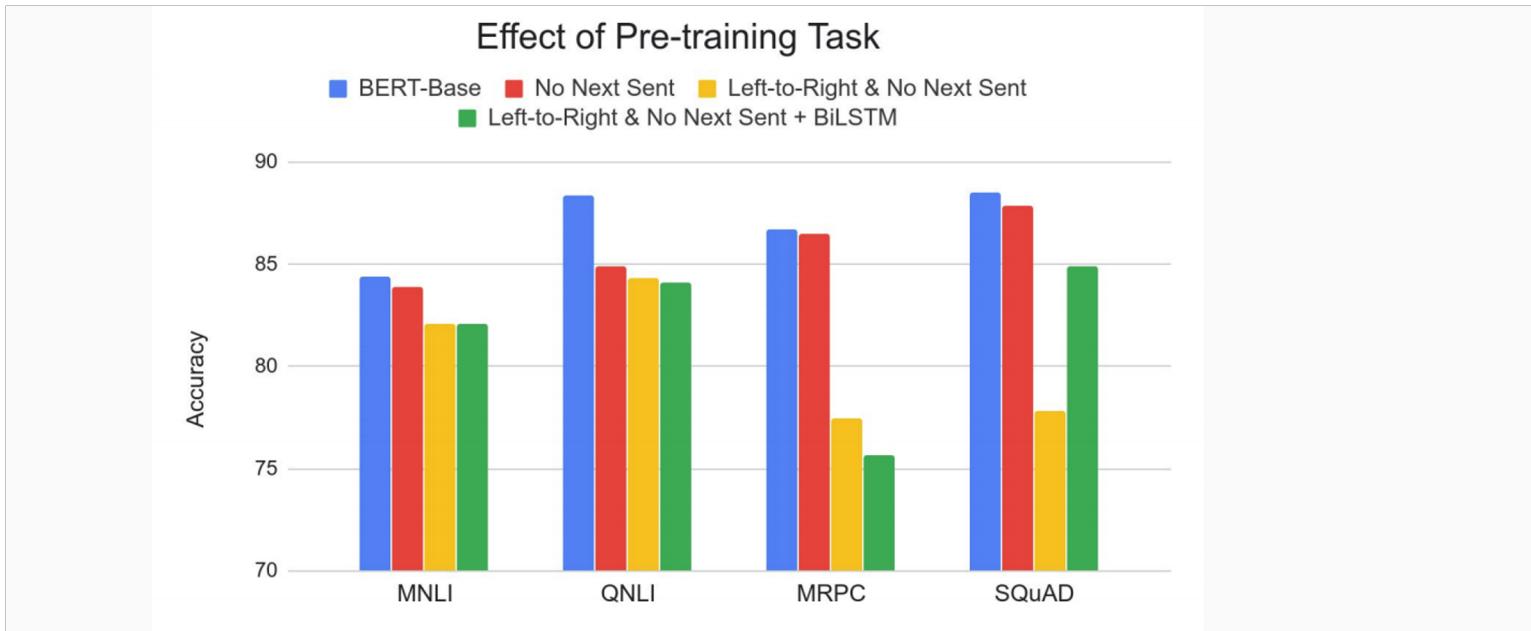
# BERT secrets

- Very large model & lots of compute
- Moderately large corpora (3.2B words)
- Deep bidirectional contextualized word representation  
(seeing all words when encoding each word in each layer)

ULMfit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	Feb 2019
Training:	Training	Training	Training
1 GPU day	240 GPU days	256 TPU days ~320–560 GPU days	~2048 TPU v3 days according to <a href="#">a reddit thread</a>

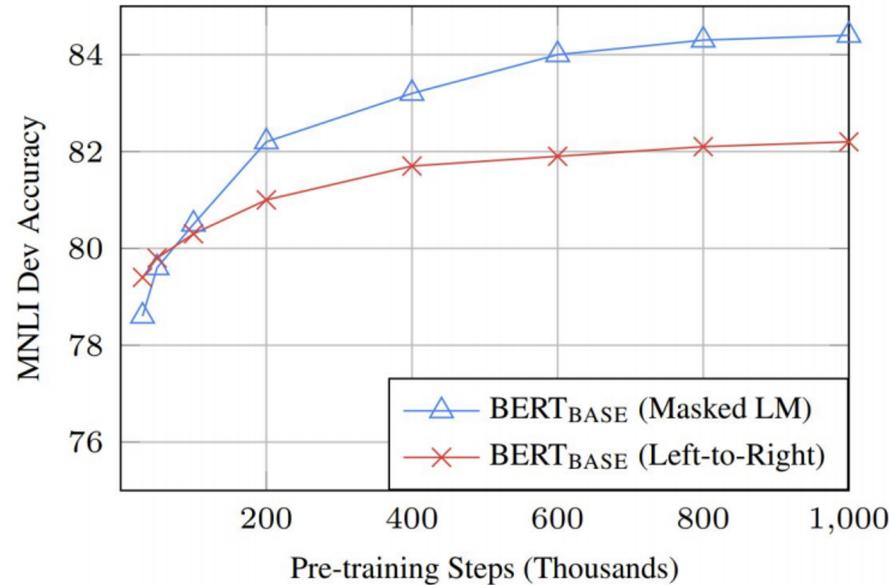


# Pre-training objectives effect



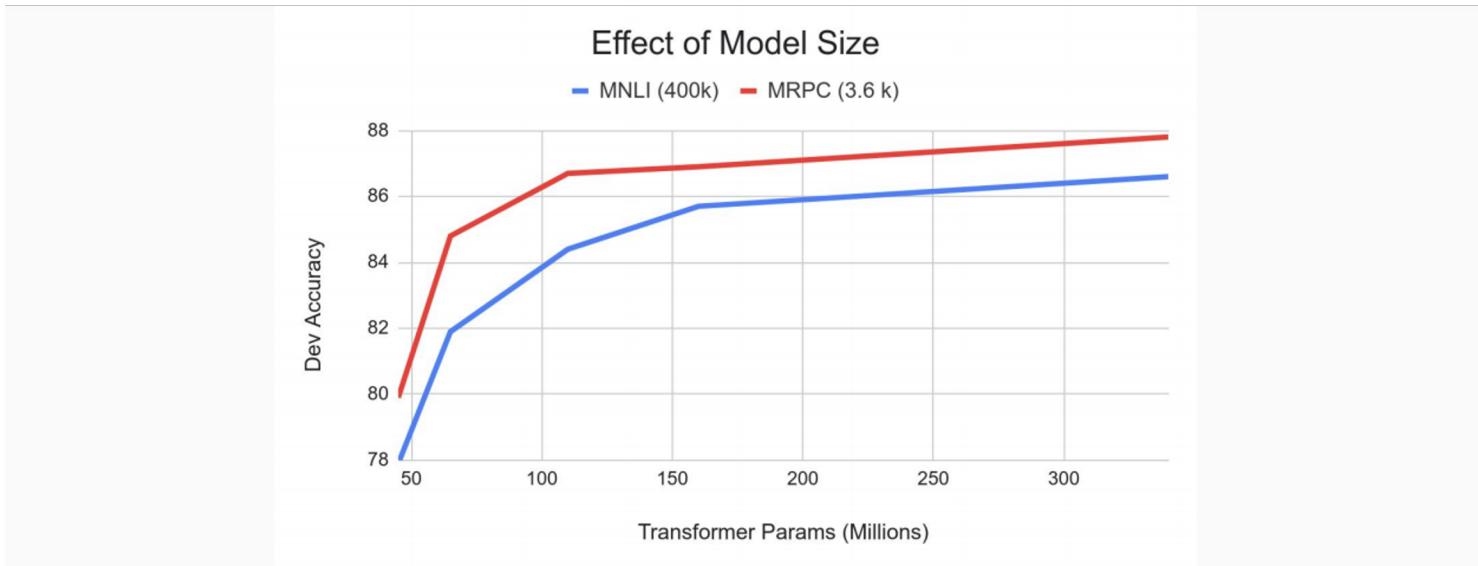
Masked LM (compared to left-to-right LM) is very important on some tasks, Next Sentence Prediction is important on other tasks. Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM

# MLM vs. LM pretraining



Masked LM takes slightly longer to converge because we only predict 15% instead of 100%  
But absolute results are much better almost immediately

# Do we need huge models?



Big models help a *lot*

Going from 110M → 340M params helps even on datasets with 3,600 labeled examples

Improvements have not asymptoted

# RuBERT

Pre-Trained on the Russian-language texts

Created by SberDevices

- Tokenizer: bpe
- Dict size: 120 138
- Training Data Volume 30 GB



Base: <https://huggingface.co/ai-forever/rubert-base>

Large: <https://huggingface.co/ai-forever/rubert-large>



# Conclusion

- A new paradigm in NLP: fine-tune large pre-trained model for particular tasks (Transfer learning)
- The pretrained model already knows a lot about language
- The use of pre-trained models opens up new perspectives: less data, higher quality indicators, less fine-tuning time

# BERT "relatives"

- mBERT
- RoBERTa
- XLM-RoBERTa (aka XLM-R)
- Electra
- DeBERTa
- and many-many others!



RoBERTa

# RoBERTa

- RoBERTa: Robustly optimized BERT approach, 2019
- Was not accepted to ICLR-2020 due to the limited novelty and despite of SOTA results on GLUE, SQuAD, RACE
- Google sub-optimally selected optimization hyperparameters / decisions and trained BERT poorly, so Facebook fixed it  
=> much better than BERT

## RoBERTa: A Robustly Optimized BERT Pretraining Approach

**Yinhan Liu<sup>\*§</sup> Myle Ott<sup>\*§</sup> Naman Goyal<sup>\*§</sup> Jingfei Du<sup>\*§</sup> Mandar Joshi<sup>†</sup>  
Danqi Chen<sup>§</sup> Omer Levy<sup>§</sup> Mike Lewis<sup>§</sup> Luke Zettlemoyer<sup>†§</sup> Veselin Stoyanov<sup>§</sup>**

<sup>†</sup> Paul G. Allen School of Computer Science & Engineering,  
University of Washington, Seattle, WA

{mandar90, lsz}@cs.washington.edu

<sup>§</sup> Facebook AI

{yinhanliu, myleott, naman, jingfeidu,  
danqi, omerlevy, mikelewis, lsz, ves}@fb.com

# Dynamic Masking

- BERT: data was duplicated 10 times and perturbed (masked, etc.) before training for 40 epochs  
*=> inputs/targets are seen 4 times*
- RoBERTa: randomly perturbs inputs during training.
  - No need to duplicate large datasets
  - More diverse data when training more epochs
  - Natural (like dropout)
  - Similar or better results

<b>Masking</b>	<b>SQuAD 2.0</b>	<b>MNLI-m</b>	<b>SST-2</b>
reference	76.3	84.3	92.8
<i>Our reimplementation:</i>			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Table 1: Comparison between static and dynamic masking for BERT<sub>BASE</sub>. We report F1 for SQuAD and accuracy for MNLI-m and SST-2. Reported results are medians over 5 random initializations (seeds). Reference results are from Yang et al. (2019).

# Inputs and NSP

- **segment-pair+NSP (BERT)**: a pair of text fragments, combined length is  $\leq 512$
- **sentence-pair+NSP**: a pair of sentences, much shorter than 512 => dynamic batch size to obtain same number of tokens in a batch
- **full-sentences (no NSP)**: a sequence of whole sentences taken continuously from the corpus while length is  $\leq 512$ . Insert special symbol for document boundaries.
- **doc-sentences (no NSP)**: similarly to previous, but examples do not cross document boundaries. Increase batch size dynamically to obtain same number of tokens in a batch as in full-sentences.

# Inputs and NSP

Doc-sentences is best. But results in variable size batch.  
=> Full-sentences is used for all other experiments.

Sentences are worse than text fragments  
(cannot learn long-range deps?)

Removing NSP helps (contrary to BERT paper)

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
<i>Our reimplementation (with NSP loss):</i>				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
<i>Our reimplementation (without NSP loss):</i>				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
BERT <sub>BASE</sub>	88.5/76.3	84.3	92.8	64.3
XLNet <sub>BASE</sub> (K = 7)	-/81.3	85.8	92.7	66.1
XLNet <sub>BASE</sub> (K = 6)	-/81.0	85.6	93.4	66.7

Table 2: Development set results for base models pretrained over BOOKCORPUS and WIKIPEDIA. All models are trained for 1M steps with a batch size of 256 sequences. We report F1 for SQuAD and accuracy for MNLI-m, SST-2 and RACE. Reported results are medians over five random initializations (seeds). Results for BERT<sub>BASE</sub> and XLNet<sub>BASE</sub> are from Yang et al. (2019).

# Inputs and NSP

- Compare different batch sizes for the fixed number of epochs (different number of steps).
- =>  $bsz=2K$  (8x larger than in BERT) is better even with 8x less training steps;  $bsz=8K$  is worse (too few steps?)

<b>bsz</b>	<b>steps</b>	<b>lr</b>	<b>ppl</b>	<b>MNLI-m</b>	<b>SST-2</b>
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	<b>3.68</b>	<b>85.2</b>	<b>92.9</b>
8K	31K	1e-3	3.77	84.6	92.8

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

# Subwords

- BERT: *30K character level BPE* (WordPiece) vocabulary, heuristic preprocessing and tokenization.
- RoBERTa: **50K byte-level BPE**, no preprocessing or tokenization (much simpler and more universal, but slightly worse results for some tasks in the preliminary experiments)
  - *Too many unicode characters (~140K), a significant part of which occurs in a diverse corpora, can occupy a large part of the vocabulary leaving too few space for words and large parts of words. The model can degenerate to character level model.*
  - *Byte-level BPE starts from bytes, not unicode symbols. With 50K vocabulary it can encode any input string without using <unk> token.*
  - *Still looks at unicode character categories to prevent merging different categories (otherwise, wastes vocabulary: dog. dog! dog?)*

# Data

- BERT→RoBERTa: 16GB→**160GB**
- 16GB from BERT: BookCorpus + En Wikipedia
- 76GB CC-News – En part of CommonCrawl News
- 38GB OpenWebText – crawled URLs from Reddit with  $\geq 3$  upvotes
- 31GB Stories – subset of CommonCrawl matching story-like style of Winograd schemas
- Training large models: 4days on 64 TPU chips<sup>13</sup>→  
1 day on 1024 V100 GPUs

# BERT→RoBERTa

- 10x larger dataset
- 32x larger batch size and longer training
  - but 2x less steps
- Dynamic masking
- Removed Next sentence prediction loss and segment-pairs input format
- 30k WordPiece→**50k byte-level BPE vocabulary**
- Careful selection of optimization hyperparameters for pre-training and fine-tuning

# GLUE test results

- Single-task finetuning (unlike some other top models);  
for RTE, STS, MRPC start fine-tuning from model fine-tuned on MNLI
- Ensemble 5-7 models per task

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT <sub>LARGE</sub>	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet <sub>LARGE</sub>	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	<b>90.2/90.2</b>	<b>94.7</b>	<b>92.2</b>	<b>86.6</b>	<b>96.4</b>	<b>90.9</b>	<b>68.0</b>	<b>92.4</b>	<b>91.3</b>	-
<i>Ensembles on test (from leaderboard as of July 25, 2019)</i>										
ALICE	88.2/87.9	95.7	<b>90.7</b>	83.5	95.2	92.6	<b>68.6</b>	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	<b>96.8</b>	<b>93.0</b>	67.8	91.6	<b>90.4</b>	88.4
RoBERTa	<b>90.8/90.2</b>	<b>98.9</b>	90.2	<b>88.2</b>	96.7	92.3	67.8	<b>92.2</b>	89.0	<b>88.5</b>

Table 5: Results on GLUE. All results are based on a 24-layer architecture. BERT<sub>LARGE</sub> and XLNet<sub>LARGE</sub> results are from Devlin et al. (2019) and Yang et al. (2019), respectively. RoBERTa results on the development set are a median over five runs. RoBERTa results on the test set are ensembles of *single-task* models. For RTE, STS and MRPC we finetune starting from the MNLI model instead of the baseline pretrained model. Averages are obtained from the GLUE leaderboard.

# Other datasets

Model	SQuAD 1.1		SQuAD 2.0	
	EM	F1	EM	F1
<i>Single models on dev, w/o data augmentation</i>				
BERT <sub>LARGE</sub>	84.1	90.9	79.0	81.8
XLNet <sub>LARGE</sub>	<b>89.0</b>	94.5	86.1	88.8
RoBERTa	88.9	<b>94.6</b>	<b>86.5</b>	<b>89.4</b>
<i>Single models on test (as of July 25, 2019)</i>				
XLNet <sub>LARGE</sub>			86.3 <sup>†</sup>	89.1 <sup>†</sup>
RoBERTa			86.8	89.8
XLNet + SG-Net Verifier			<b>87.0<sup>†</sup></b>	<b>89.9<sup>†</sup></b>

Table 6: Results on SQuAD. <sup>†</sup> indicates results that depend on additional external training data. RoBERTa uses only the provided SQuAD data in both dev and test settings. BERT<sub>LARGE</sub> and XLNet<sub>LARGE</sub> results are from Devlin et al. (2019) and Yang et al. (2019), respectively.

Model	Accuracy	Middle	High
<i>Single models on test (as of July 25, 2019)</i>			
BERT <sub>LARGE</sub>	72.0	76.6	70.1
XLNet <sub>LARGE</sub>	81.7	85.4	80.2
RoBERTa	<b>83.2</b>	<b>86.5</b>	<b>81.3</b>

Table 7: Results on the RACE test set. BERT<sub>LARGE</sub> and XLNet<sub>LARGE</sub> results are from Yang et al. (2019).

SOTA among models w/o data augmentation

XLM-RoBERTa

# XLM-R (XLM-RoBERTa)

- A single RoBERTa model trained on **2.5TB** of **filtered** texts from CommonCrawl on **100 languages**
- Shows zero-shot cross-lingual transfer ability!
- Takes best from XLM (multilingual) and RoBERTa (English) models

**Unsupervised Cross-lingual Representation Learning at Scale**

**Alexis Conneau\*** **Kartikay Khandelwal\***

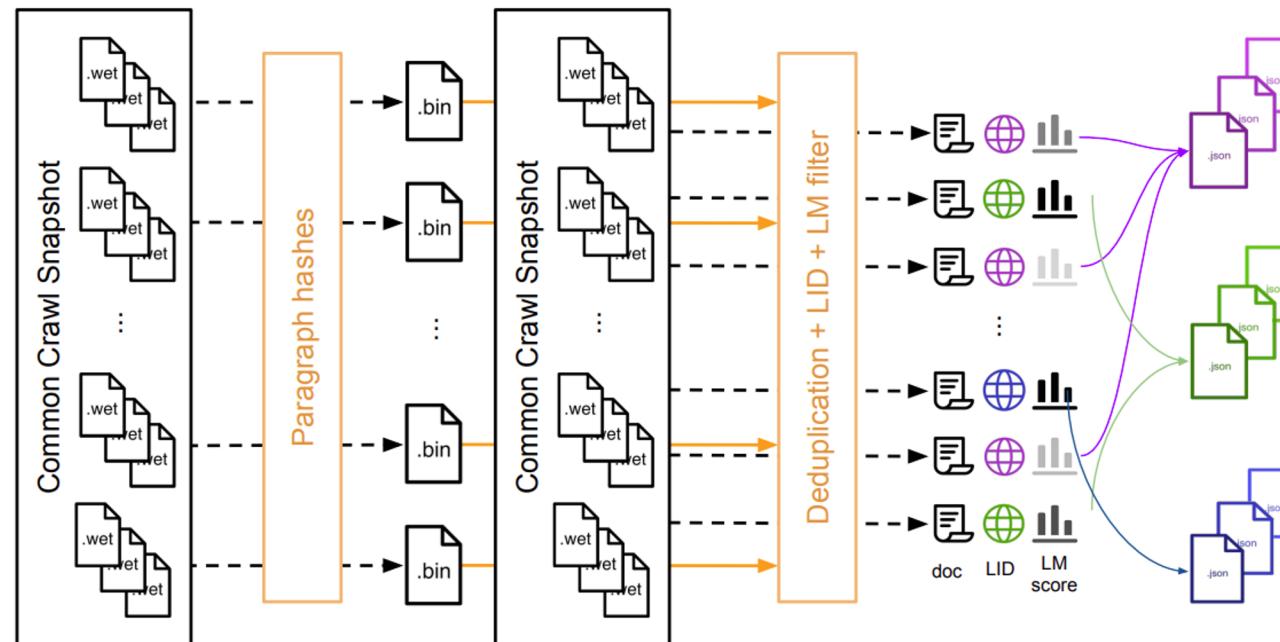
**Naman Goyal** **Vishrav Chaudhary** **Guillaume Wenzek** **Francisco Guzmán**

**Edouard Grave** **Myle Ott** **Luke Zettlemoyer** **Veselin Stoyanov**

**Facebook AI**

# XLM-R data

- Build clean CommonCrawl in 100 languages with CCNet:
  - *Partially deduplicate paragraphs (also removes boilerplate)*
  - *Run lang id and filter by language*
  - *Filter by perplexity (for each language a Kneser-Ney 5-gram LM is trained on Wikipedia in this language)*



# XLM-R data

- Upsampling low-resource languages
  - Alleviates bias towards high-resource languages
  - Prevents words in low-resource languages from being split into characters
  - Alpha=0.3 (0.5 in XLM):
    - Ex: English / Uzbek = 300GB / 0.7GB = 430;  $430 \cdot 0.3 = 6.16$
  - N is number of languages,  $n_i$  is number of sentences in language i

$$q_i = \frac{p_i^\alpha}{\sum_{j=1}^N p_j^\alpha} \quad \text{with} \quad p_i = \frac{n_i}{\sum_{k=1}^N n_k}$$

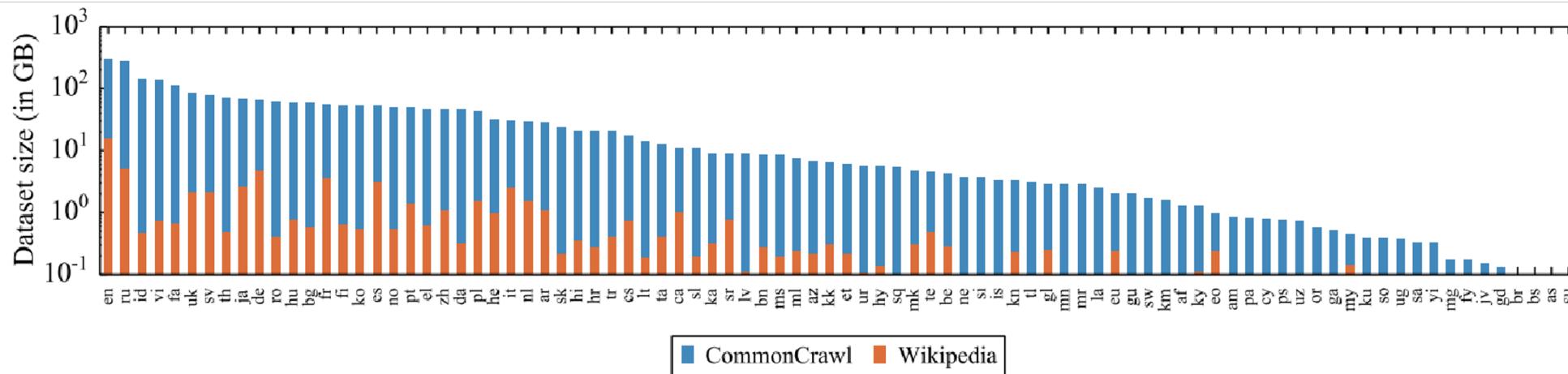


Figure 1: Amount of data in GiB (log-scale) for the 88 languages that appear in both the Wiki-100 corpus used for mBERT and XLM-100, and the CC-100 used for XLM-R. CC-100 increases the amount of data by several orders of magnitude, in particular for low-resource languages.

# The curse of multilinguality

- With fixed capacity as we add more langs, the quality:
  - *When 93 langs added, acc. for 7 eval langs:  $0.718 \rightarrow 0.677$*
  - *for high-res. langs decrease (capacity dilution)*
  - *for low-res. – first increases (due to cross-ling. transfer), then decreases (due to capacity dilution)*
- Increase capacity with number of languages?

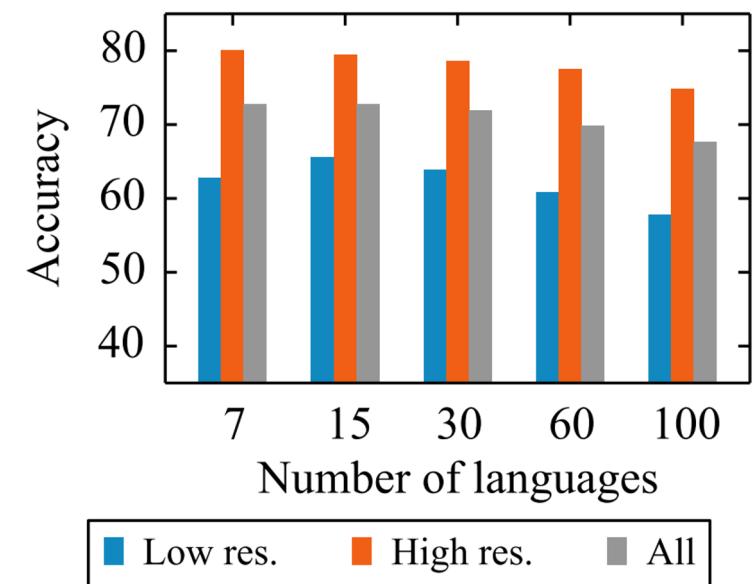
For ablations: base model, pre-training on Wikipedia, fine-tuning on XNLI.

Accuracy on XNLI (zero-shot from en?)

High res.: (English + French) / 2

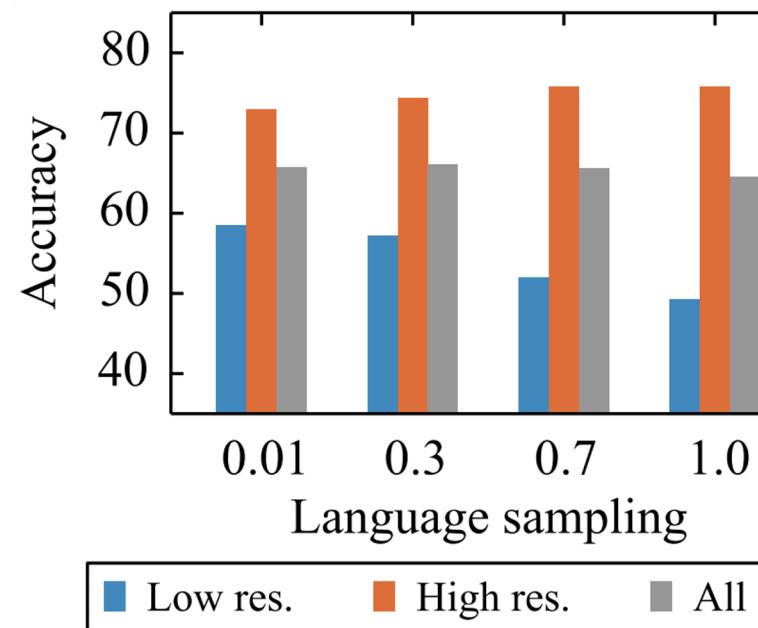
Low res.: (Swahili + Urdu) / 2

All: + German, Russian, Chinese



# The curse of multilinguality

- XLM-100 on pre-trained Wikipedia
- Upsampling helps low-res. langs a lot, but hurts high-res. langs a little
- alpha=0.3 results in best average accuracy on 7 eval. langs



# XLM-R large

- As BERT/RoBERTa large: 24 layers of Transformer, hidden size 1024
  - 270M/550M in base/large model
  - about half of parameters are subword embeddings
- Vocabulary: 250K subwords shared between 100 languages
  - Sentence Piece with a unigram LM directly on raw text
  - MLM pre-trained with full softmax
- 1.5M updates, bs=8192
- 500 V100 GPUs
  - How many days?
  - Compared to RoBERTa: 2x less GPUs, 3x more updates, same bs => about 1 week?

# XNLI accuracy (on 6/15 langs)

Model	D	#M	#lg	4 high-res				2 high-res Avg across 15		
				en	fr	es	de	sw	ur	Avg
<i>Fine-tune multilingual model on English training set (Cross-lingual Transfer)</i>										
Lample and Conneau (2019)	Wiki+MT	N	15	85.0	78.7	78.9	77.8	68.4	67.3	75.1
Huang et al. (2019)	Wiki+MT	N	15	85.1	79.0	79.4	77.8	69.7	66.7	75.4
Devlin et al. (2018)	Wiki	N	102	82.1	73.8	74.3	71.1	50.4	58.0	66.3 +14% cmp. to mBERT
Lample and Conneau (2019)	Wiki	N	100	83.7	76.2	76.6	73.7	58.0	62.4	71.3
Lample and Conneau (2019)	Wiki	1	100	83.2	76.7	77.7	74.0	58.2	62.4	70.7
<b>XLM-R<sub>Base</sub></b>	CC	1	100	85.8	79.7	80.7	78.7	66.5	68.3	76.2
<b>XLM-R</b>	CC	1	100	<b>89.1</b>	<b>84.1</b>	<b>85.1</b>	<b>83.9</b>	<b>73.9</b>	<b>73.8</b>	<b>80.9</b>
<i>Translate everything to English and use English-only model (TRANSLATE-TEST)</i>										
BERT-en	Wiki	1	1	88.8	81.4	82.3	80.1	65.8	65.8	76.2
RoBERTa	Wiki+CC	1	1	<b>91.3</b>	82.9	84.3	81.2	66.7	66.8	77.8
<i>Fine-tune multilingual model on each training set (TRANSLATE-TRAIN)</i>										
Lample and Conneau (2019)	Wiki	N	100	82.9	77.6	77.9	77.9	66.5	62.4	74.2
<i>Fine-tune multilingual model on all training sets (TRANSLATE-TRAIN-ALL)</i>										
Lample and Conneau (2019) <sup>†</sup>	Wiki+MT	1	15	85.0	80.8	81.3	80.3	72.8	68.5	77.8
Huang et al. (2019)	Wiki+MT	1	15	85.6	81.1	82.3	80.9	73.8	69.6	78.5
Lample and Conneau (2019)	Wiki	1	100	84.5	80.1	81.3	79.3	69.2	67.7	76.9
<b>XLM-R<sub>Base</sub></b>	CC	1	100	85.4	81.4	82.2	80.3	73.1	73.0	79.1
<b>XLM-R</b>	CC	1	100	<b>89.1</b>	<b>85.1</b>	<b>86.6</b>	<b>85.7</b>	<b>78.0</b>	<b>78.1</b>	<b>83.6</b>

# XLM-R (Summary)

- Pre-training a single MLM on 100 languages:
  - requires high capacity (large vocab., hidden size)
  - requires long training with low-res. langs upsampling
- Fine-tuning multilingual MLM:
  - enables zero-shot cross-lingual transfer
  - but better translate (MT) into all target languages and fine-tune on multilingual train set

**Bonus: ELECTRA**

# ELECTRA

- Instead of masking random tokens, replace them using a MLM
  - For each token, predict whether it was replaced
  - This objective is more sample-efficient, because it involves all tokens
  - Smaller discrepancy between pretraining and fine-tuning
- “Efficiently Learning an Encoder that Classifies Token Replacements Accurately”

## ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS

**Kevin Clark**

Stanford University

[kevclark@cs.stanford.edu](mailto:kevclark@cs.stanford.edu)

**Minh-Thang Luong**

Google Brain

[thangluong@google.com](mailto:thangluong@google.com)

**Quoc V. Le**

Google Brain

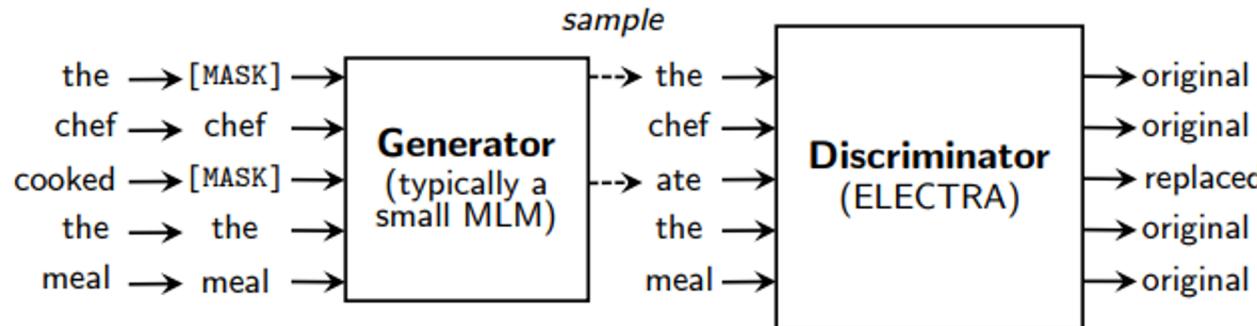
[qvl@google.com](mailto:qvl@google.com)

**Christopher D. Manning**

Stanford University & CIFAR Fellow

[manning@cs.stanford.edu](mailto:manning@cs.stanford.edu)

# Replaced token detection

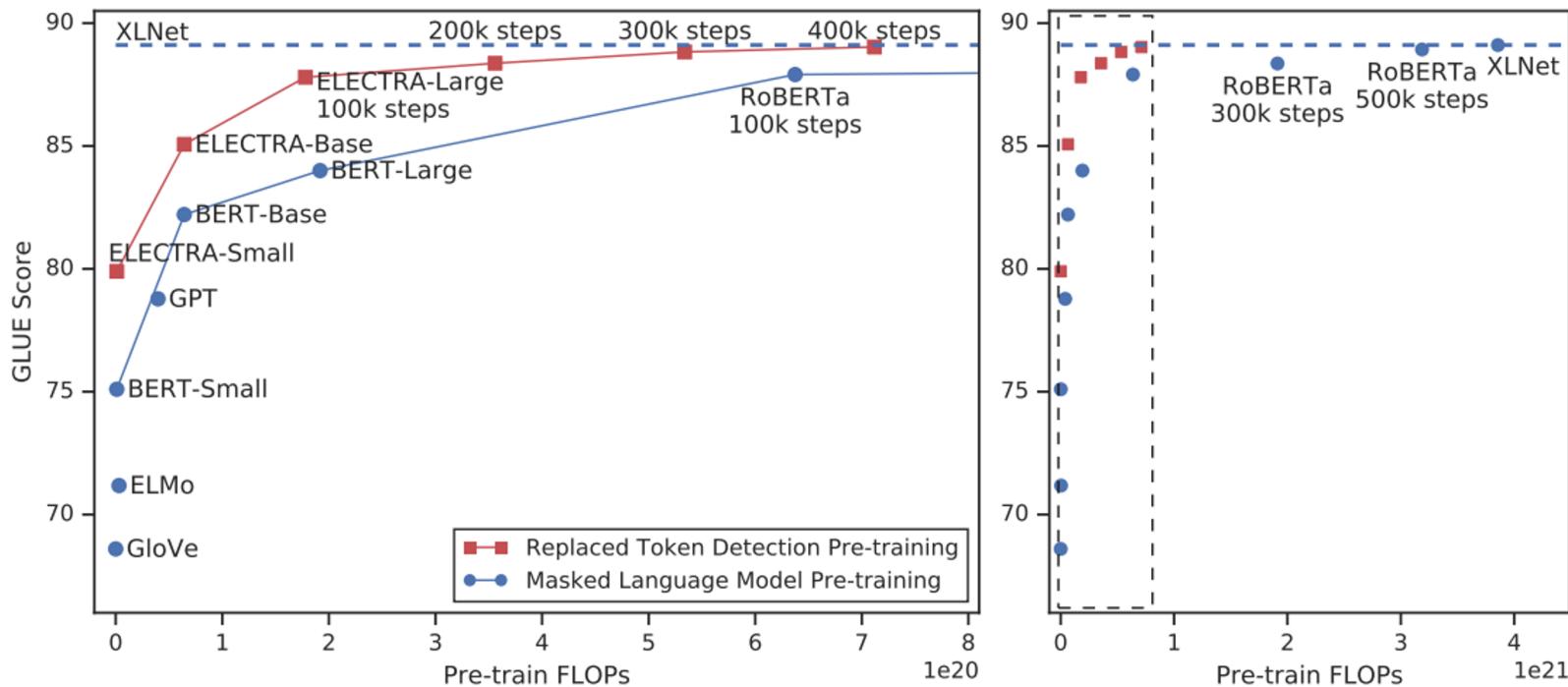


- The generator is a small transformer MLM
  - Trained jointly with the discriminator, but not adversarially
  - Embeddings can be tied with the discriminator
- The discriminator predicts for each token, whether it has changed

$$\mathcal{L}_{\text{MLM}}(\mathbf{x}, \theta_G) = \mathbb{E} \left( \sum_{i \in \mathbf{m}} -\log p_G(x_i | \mathbf{x}^{\text{masked}}) \right)$$

$$\mathcal{L}_{\text{Disc}}(\mathbf{x}, \theta_D) = \mathbb{E} \left( \sum_{t=1}^n -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\mathbf{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\mathbf{x}^{\text{corrupt}}, t)) \right)$$

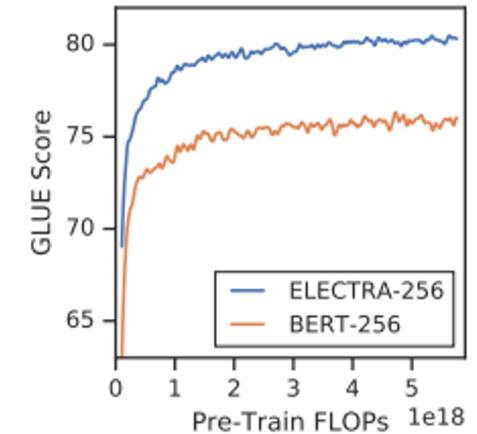
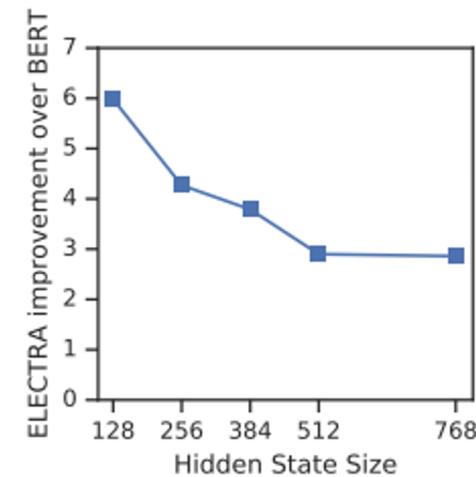
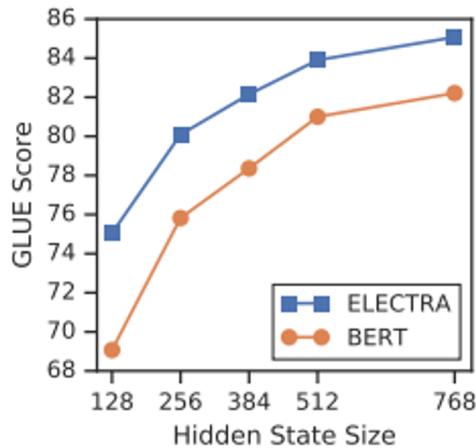
# Pretraining efficiency



- Pre-training ELECTRA for the same number of steps as BERT is costlier
  - because two models are trained
- But the model trains faster and converges to a better accuracy

# Ablations

ELECTRA outperforms BERT for all model sizes, especially with small models



Efficiency comes mostly out of working with all tokens

Predict token identity for all tokens

Predict token identity for replaced tokens

Discriminate only the 15% tokens that were masked

Model	ELECTRA	All-Tokens MLM	Replace MLM	ELECTRA 15%	BERT
GLUE score	85.0	84.3	82.4	82.4	82.2

Bonus: DeBERTa

# DeBERTa

- Improvements over RoBERTa:
  - Disentangle attention to content and to (relative) position of tokens
  - Incorporate absolute positions to the MLM decoder layer
- Improvements over ELECTRA:
  - Share generator embeddings with the discriminator, but not vice versa
  - Train a cross-lingual model (like XLM-R)

Published as a conference paper at ICLR 2021

## DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION

Pengcheng He<sup>1</sup>, Xiaodong Liu<sup>2</sup>, Jianfeng Gao<sup>2</sup>, Weizhu Chen<sup>1</sup>

<sup>1</sup> Microsoft Dynamics 365 AI

<sup>2</sup> Microsoft Research

{penhe, xiaodl, jfgao, wzchen}@microsoft.com

## DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing

Pengcheng He<sup>1</sup>, Jianfeng Gao<sup>2</sup>, Weizhu Chen<sup>1</sup>

<sup>1</sup> Microsoft Azure AI

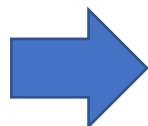
<sup>2</sup> Microsoft Research

{penhe, jfgao, wzchen}@microsoft.com

# Disentangled attention

- Separate attentions for content (H) and position (P)
  - In transformers, attention often depends on position; let's simplify the job!
  - P is represented with relative position embeddings (same for all layers)
  - All attention scores are added together before softmax

$$\begin{aligned} \mathbf{Q} &= \mathbf{HW}_q, \mathbf{K} = \mathbf{HW}_k, \mathbf{V} = \mathbf{HW}_v, \mathbf{A} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} \\ \mathbf{H}_o &= \text{softmax}(\mathbf{A})\mathbf{V} \end{aligned}$$



$$\begin{aligned} \mathbf{Q}_c &= \mathbf{HW}_{q,c}, \mathbf{K}_c = \mathbf{HW}_{k,c}, \mathbf{V}_c = \mathbf{HW}_{v,c}, \mathbf{Q}_r = \mathbf{PW}_{q,r}, \mathbf{K}_r = \mathbf{PW}_{k,r} \\ \tilde{A}_{i,j} &= \underbrace{\mathbf{Q}_i^c \mathbf{K}_j^{c\top}}_{\text{(a) content-to-content}} + \underbrace{\mathbf{Q}_i^c \mathbf{K}_{\delta(i,j)}^r}_{\text{(b) content-to-position}}^\top + \underbrace{\mathbf{K}_j^c \mathbf{Q}_{\delta(j,i)}^r}_{\text{(c) position-to-content}}^\top \end{aligned}$$

$$\mathbf{H}_o = \text{softmax}\left(\frac{\tilde{\mathbf{A}}}{\sqrt{3d}}\right)\mathbf{V}_c$$

Relative position:

$$\delta(i, j) = \begin{cases} 0 & \text{for } i - j \leq -k \\ 2k - 1 & \text{for } i - j \geq k \\ i - j + k & \text{others.} \end{cases}$$

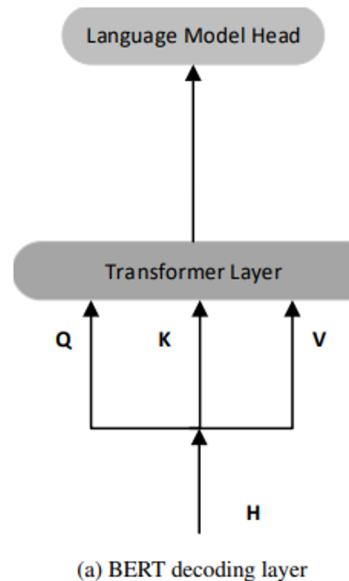
# Enhanced mask decoder

How to incorporate absolute positions?

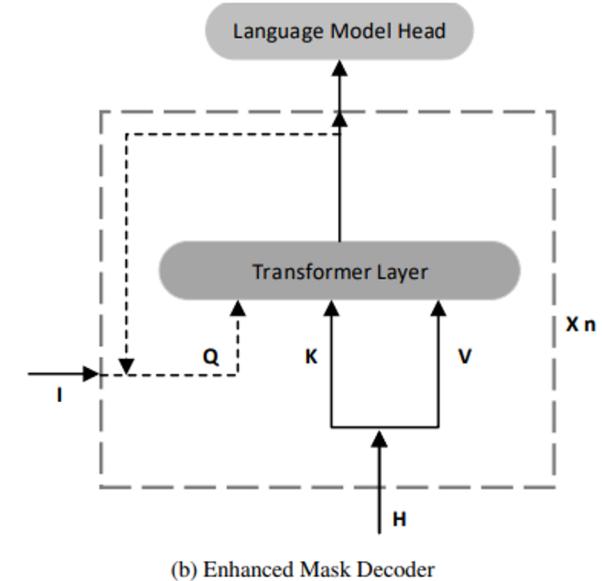
- BERT:
  - add their embeddings to the input token embeddings
- DeBERTa:
  - use their embeddings / as the query to the last transformer layer
  - Loop through this layer n=2 times
    - First use / for the queries, then the previous output of this layer

Motivation:

- Absolute positions matter most on the last layer, because syntax depends on this, and syntax matters for MLM
- “We conjecture that the early incorporation of absolute positions used by BERT might undesirably hamper the model from learning sufficient information of relative positions.”



(a) BERT decoding layer



(b) Enhanced Mask Decoder

# DeBERTa ablations

- Both parts of disentangled attention and EMD increase the quality

- -EMD is the DeBERTa base model without EMD.
- -C2P is the DeBERTa base model without the content-to-position term ((c) in Eq. 4).
- -P2C is the DeBERTa base model without the position-to-content term ((b) in Eq. 4). As XLNet also uses the relative position bias, this model is close to XLNet plus EMD.

Model	MNLI-m/mm Acc	SQuAD v1.1 F1/EM	SQuAD v2.0 F1/EM	RACE Acc
BERT <sub>base</sub> Devlin et al. (2019)	84.3/84.7	88.5/81.0	76.3/73.7	65.0
RoBERTa <sub>base</sub> Liu et al. (2019c)	84.7/-	90.6/-	79.7/-	65.6
XLNet <sub>base</sub> Yang et al. (2019)	85.8/85.4	-/-	81.3/78.5	66.7
RoBERTa-ReImp <sub>base</sub>	84.9/85.1	91.1/84.8	79.5/76.0	66.8
DeBERTa <sub>base</sub>	<b>86.3/86.2</b>	<b>92.1/86.1</b>	<b>82.5/79.3</b>	<b>71.7</b>
-EMD	86.1/86.1	91.8/85.8	81.3/78.0	70.3
-C2P	85.9/85.7	91.6/85.8	81.3/78.3	69.3
-P2C	86.0/85.8	91.7/85.7	80.8/77.6	69.6
-(EMD+C2P)	85.8/85.9	91.5/85.3	80.3/77.2	68.1
-(EMD+P2C)	85.8/85.8	91.3/85.1	80.2/77.1	68.5

# Scale invariant fine-tuning

- This is an additive regularization loss used at fine-tuning
  - Add a small adversarial noise to the normalized input embeddings
  - Penalize for changes in the distribution
  - This loss induces smoothness
- Adds  $\approx 1$  accuracy point

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta),$$

$$\mathcal{R}_s(\theta) = \frac{1}{n} \sum_{i=1}^n \max_{\|\tilde{x}_i - x_i\|_p \leq \epsilon} \ell_s(f(\tilde{x}_i; \theta), f(x_i; \theta)),$$

$$\ell_s(P, Q) = D_{\text{KL}}(P\|Q) + D_{\text{KL}}(Q\|P);$$

Model	CoLA Mcc	QQP Acc	MNLI-m/mm Acc	SST-2 Acc	STS-B Corr	QNLI Acc	RTE Acc	MRPC Acc	Avg.
DeBERTa <sub>large</sub>	70.5	92.3	91.1/91.1	96.8	92.8	95.3	88.3	91.9	90.00
DeBERTa <sub>900M</sub>	71.1	92.3	91.7/91.6	<b>97.5</b>	92.0	95.8	93.5	93.1	90.86
DeBERTa <sub>1.5B</sub>	72.0	92.7	91.7/91.9	97.2	92.9	96.0	93.9	92.0	91.17
DeBERTa <sub>1.5B</sub> +SiFT	<b>73.5</b>	<b>93.0</b>	<b>92.0/92.1</b>	97.5	<b>93.2</b>	<b>96.5</b>	<b>96.5</b>	<b>93.2</b>	<b>91.93</b>

Table 12: Comparison results of DeBERTa models with different sizes on the GLUE development set.

# SuperGLUE results

- Some more tricks for scaling the model to 1.5B parameters
  - Share projection matrices for relative position embeddings
  - Add a convolution layer to enhance the word embeddings with subword information
- As a result of all hacks, DeBERTa was the first model to surpass the average human scores on the SuperGLUE benchmark

Model	BoolQ Acc	CB F1/Acc	COPA Acc	MultiRC F1a/EM	ReCoRD F1/EM	RTE Acc	WiC Acc	WSC Acc	Average Score
RoBERTa <sub>large</sub>	87.1	90.5/95.2	90.6	84.4/52.5	90.6/90.0	88.2	69.9	89.0	84.6
NEXHA-Plus	87.8	94.4/96.0	93.6	84.6/55.1	90.1/89.6	89.1	74.6	93.2	86.7
T5 <sub>11B</sub>	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	89.3
T5 <sub>11B</sub> +Meena	<b>91.3</b>	<b>95.8/97.6</b>	97.4	88.3/63.0	94.2/93.5	92.7	<b>77.9</b>	95.9	90.2
Human	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	89.8
DeBERTa <sub>1.5B</sub> +SiFT	90.4	94.9/97.2	96.8	<b>88.2/63.7</b>	<b>94.5/94.1</b>	<b>93.2</b>	76.4	<b>95.9</b>	89.9
DeBERTa <sub>Ensemble</sub>	90.4	95.7/97.6	<b>98.4</b>	88.2/63.7	94.5/94.1	93.2	77.5	95.9	<b>90.3</b>

Table 5: SuperGLUE test set results scored using the SuperGLUE evaluation server. All the results are obtained from <https://super.gluebenchmark.com> on January 6, 2021.

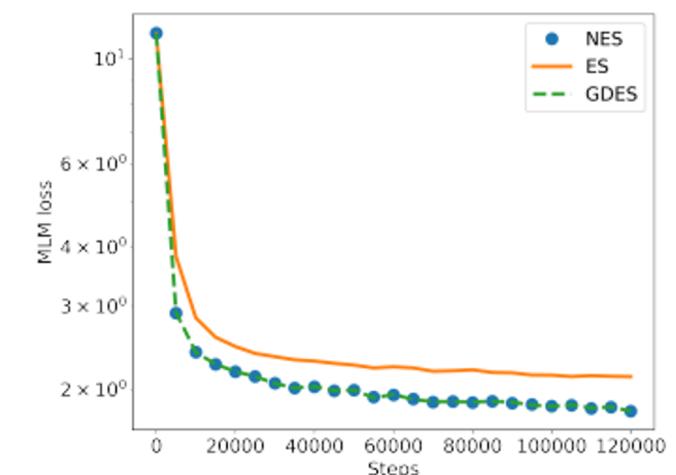
# DeBERTaV3

- ELECTRA-like discriminative pretraining is better than MLM
- ELECTRA shares generator and discriminator embeddings
  - this is suboptimal for the generator
    - Generator pulls embeddings of semantically similar words together, while discriminator pulls them apart
    - However, embeddings with impact from the MLM task improve the discriminator performance on downstream tasks
  - We want to share embeddings only in one direction (generator  $\bowtie$  discriminator)

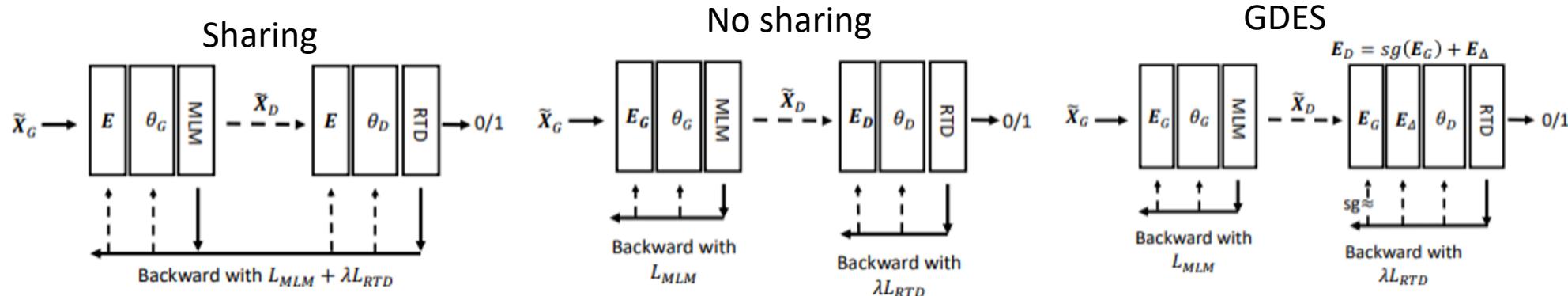
NES: no embedding sharing

ES: shared embeddings

GDES: a new method for partially sharing them



# Gradient-Disentangled Embedding Sharing



- Solution: in discriminator, learn only a matrix that is added to the generator embeddings
- This improves performance
  - We have better good generator, and weights from it are shared with the discriminator

Model	MNLI-m/mm Acc	SQuAD v2.0 F1/EM
BERT <sub>base</sub> [9]	84.3/84.7	76.3/73.7
ELECTRA <sub>base</sub> [8]	85.8/-	-/-
DeBERTa <sub>base</sub> [4]	86.3/86.2	82.5/79.3
DeBERTa+RTD <sub>base</sub>		
① ES	88.8/88.4	86.3/83.5
② NES	88.3/87.9	85.3/82.7
③ GDES	<b>89.3/89.0</b>	<b>87.2/84.5</b>

# Results

- DeBERTaV3 = DeBERTa architecture with ELECTRA training (with GDES)
- Outperforms DeBERTa and ELECTRA on various model sizes

Model	CoLA	QQP	MNLI-m/mm	SST-2	STS-B	QNLI	RTE	MRPC	Avg.
	Mcc	Acc	Acc	Acc	Corr	Acc	Acc	Acc	
#Train	8.5k	364k	393k	67k	7k	108k	2.5k	3.7k	
BERT <sub>large</sub>	60.6	91.3	86.6/-	93.2	90.0	92.3	70.4	88.0	84.05
RoBERTa <sub>large</sub>	68.0	92.2	90.2/90.2	96.4	92.4	93.9	86.6	90.9	88.82
XLNet <sub>large</sub>	69.0	92.3	90.8/90.8	97.0	92.5	94.9	85.9	90.8	89.15
ELECTRA <sub>large</sub>	69.1	92.4	90.9/-	96.9	92.6	95.0	88.0	90.8	89.46
DeBERTa <sub>large</sub>	70.5	92.3	91.1/91.1	96.8	92.8	95.3	88.3	91.9	90.00
DeBERTaV3 <sub>large</sub>	<b>75.3</b>	<b>93.0</b>	<b>91.8/91.9</b>	96.9	<b>93.0</b>	<b>96.0</b>	<b>92.7</b>	<b>92.2</b>	<b>91.37</b>

Table 3: Comparison results on the GLUE development set.

Model	Vocabulary Size(K)	Backbone #Params(M)	MNLI-m/mm ACC	SQuAD v2.0 F1/EM
Base models: 12 layers, 768 hidden size, 12 heads				
BERT <sub>base</sub>	30	86	84.3/84.7	76.3/73.7
RoBERTa <sub>base</sub>	50	86	87.6/-	83.7/80.5
XLNet <sub>base</sub>	32	92	86.8/-	-/80.2
ELECTRA <sub>base</sub>	30	86	88.8/-	-/80.5
DeBERTa <sub>base</sub>	50	100	88.8/88.5	86.2/83.1
DeBERTaV3 <sub>base</sub>	128	86	<b>90.6/90.7</b>	<b>88.4/85.4</b>
Small models: 6 layers, 768 hidden size, 12 heads				
TinyBERT <sub>small</sub>	30	44	84.5/-	77.7/-
MiniLMv2 <sub>small</sub>	30	44	87.0/-	81.6/-
BERT <sub>small</sub>	30	44	81.8/-	73.2/-
DeBERTaV3 <sub>small</sub>	128	44	<b>88.2/87.9</b>	<b>82.9/80.4</b>
XSmall models: 12 layers, 384 hidden size, 6 heads				
MiniLMv2 <sub>xsmall</sub>	30	22	86.9/-	82.3/-
DeBERTaV3 <sub>xsmall</sub>	128	22	<b>88.1/88.3</b>	<b>84.8/82.0</b>

# Multilingual DeBERTa

- Train the model on the CC100 dataset (as XLM-R)
- Vocabulary from mT5 (250K tokens)
- Pretrain like XLM-R, but for 0.5M steps instead of 1.5M
- Result: SOTA on the XNLI dataset

Model	en	fr	es	de	el	bg	ru	tr	ar	vi	th	zh	hi	sw	ur	Avg
Cross-lingual transfer																
XLM	83.2	76.7	77.7	74.0	72.7	74.1	72.7	68.7	68.6	72.9	68.9	72.5	65.6	58.2	62.4	70.7
mT5 <sub>base</sub>	84.7	79.1	80.3	77.4	77.1	78.6	77.1	72.8	73.3	74.2	73.2	74.1	70.8	69.4	68.3	75.4
XLM-R <sub>base</sub>	85.8	79.7	80.7	78.7	77.5	79.6	78.1	74.2	73.8	76.5	74.6	76.7	72.4	66.5	68.3	76.2
mDeBERTa <sub>base</sub>	<b>88.2</b>	<b>82.6</b>	<b>84.4</b>	<b>82.7</b>	<b>82.3</b>	<b>82.4</b>	<b>80.8</b>	<b>79.5</b>	<b>78.5</b>	<b>78.1</b>	<b>76.4</b>	<b>79.5</b>	<b>75.9</b>	<b>73.9</b>	<b>72.4</b>	<b>79.8</b>
Translate train all																
XLM	84.5	80.1	81.3	79.3	78.6	79.4	77.5	75.2	75.6	78.3	75.7	78.3	72.1	69.2	67.7	76.9
mT5 <sub>base</sub>	82.0	77.9	79.1	77.7	78.1	78.5	76.5	74.8	74.4	74.5	75.0	76.0	72.2	71.5	70.4	75.9
XLM-R <sub>base</sub>	85.4	81.4	82.2	80.3	80.4	81.3	79.7	78.6	77.3	79.7	77.9	80.2	76.1	73.1	73.0	79.1
mDeBERTa <sub>base</sub>	<b>88.9</b>	<b>84.4</b>	<b>85.3</b>	<b>84.8</b>	<b>84.0</b>	<b>84.5</b>	<b>83.2</b>	<b>82.0</b>	<b>81.6</b>	<b>82.0</b>	<b>79.8</b>	<b>82.6</b>	<b>79.3</b>	<b>77.3</b>	<b>73.6</b>	<b>82.2</b>

Table 6: Results on XNLI test set under cross-lingual transfer and translate train all settings.

# Conclusions

- With BERT, large pretrained transformers have conquered most NLP benchmarks
- After BERT, there were many improvements to the idea of a pretrained transformer encoder
  - But since RoBERTa, most such improvements have been rather tiny
- With XLM-R and similar models, such models went multilingual
- In the next lecture, we will discuss fine-tuning BERT in detail

# Bonus: XLNet

# XLNet

- Combines best from LM and MLM pre-training
  - An alternative to MLM: deeply bidirectional model, which is still autoregressive!
  - Improves backbone architecture by integrating ideas from Transformer-XL
- 

## XLNet: Generalized Autoregressive Pretraining for Language Understanding

---

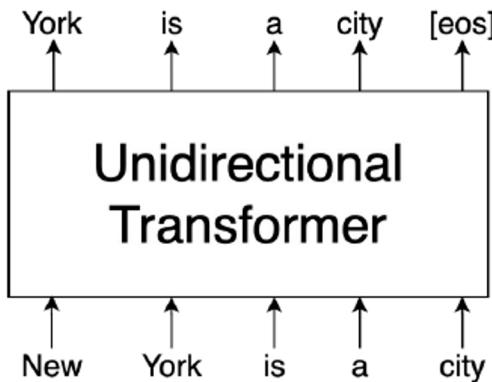
Zhilin Yang<sup>\*1</sup>, Zihang Dai<sup>\*12</sup>, Yiming Yang<sup>1</sup>, Jaime Carbonell<sup>1</sup>,  
Ruslan Salakhutdinov<sup>1</sup>, Quoc V. Le<sup>2</sup>

<sup>1</sup>Carnegie Mellon University, <sup>2</sup>Google AI Brain Team

{zhiliny,dzihang,yiming,jgc,rsalakhu}@cs.cmu.edu, qvl@google.com

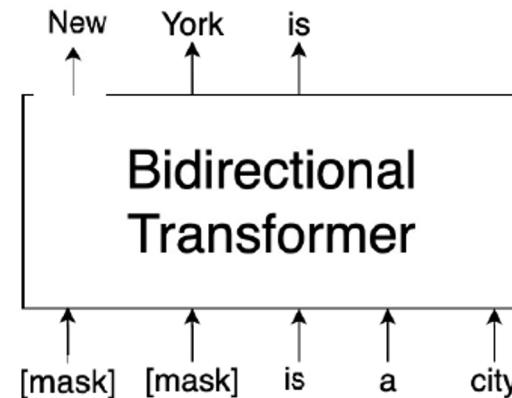
# LM vs. MLM pre-training

**Auto-Regressive (AR)**



$$\max_{\theta} \sum_{t=1}^T \log p_{\theta}(x_t | \mathbf{x}_{<t})$$

**Denoising Auto-Encoder (AE)**



$$\max_{\theta} \sum_{t=1}^T m_t \log p_{\theta}(x_t | \tilde{\mathbf{x}})$$



Natural auto-regressive dependence



Masked tokens are treated as independent given context (predicted all at once)



Similar distributions of inputs during pre-training and fine-tuning



Discrepancy between distributions of inputs during pre-training and fine-tuning ([mask] token, random replacement)

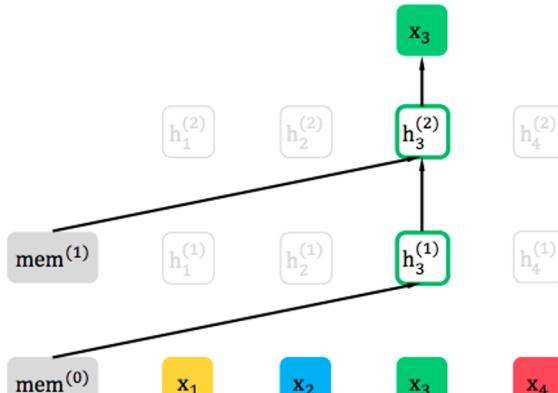


No Bidirectional Context



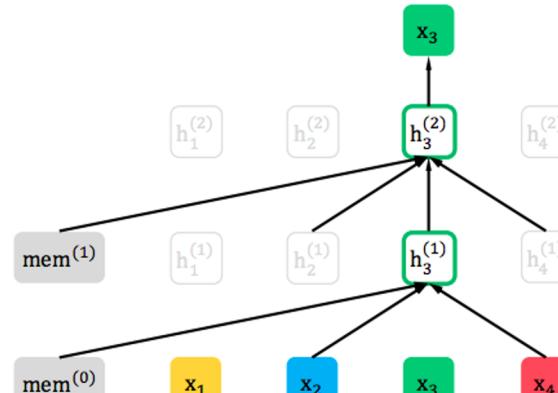
Natural Bidirectional Context

# Permutation language model



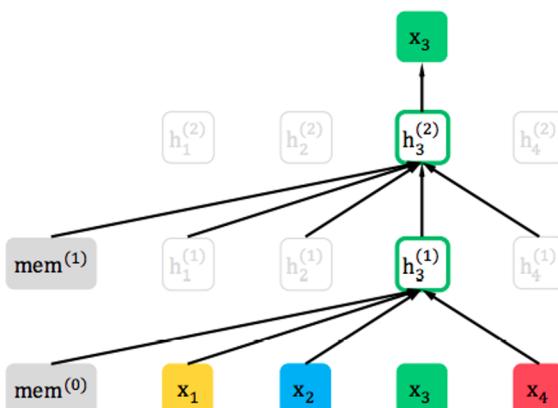
Factorization order:  $3 \rightarrow 2 \rightarrow 4 \rightarrow 1$

$$P(X) = \mathbf{P}(\mathbf{x}_3)P(x_2/x_3)P(x_4/x_3, x_2)P(x_1/x_3, x_2, x_4)$$



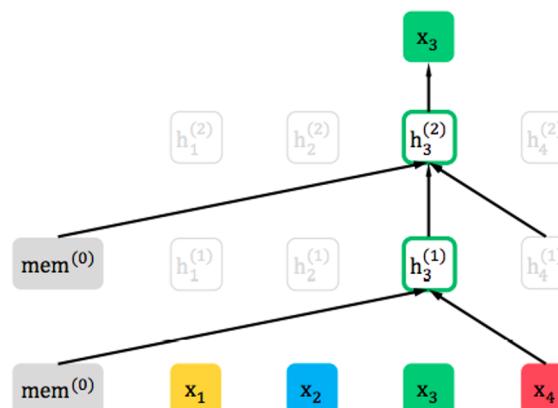
Factorization order:  $2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

$$P(X) = P(x_2)P(x_4/x_2)\mathbf{P}(\mathbf{x}_3/\mathbf{x}_2, \mathbf{x}_4)P(x_1/x_2, x_4, x_3)$$



Factorization order:  $1 \rightarrow 4 \rightarrow 2 \rightarrow 3$

$$P(X) = P(x_1)P(x_4/x_1)P(x_2/x_1, x_4)\mathbf{P}(\mathbf{x}_3/\mathbf{x}_1, \mathbf{x}_4, \mathbf{x}_2)$$



Factorization order:  $4 \rightarrow 3 \rightarrow 1 \rightarrow 2$

$$P(X) = P(x_4)\mathbf{P}(\mathbf{x}_3/\mathbf{x}_4)P(x_1/x_4, x_3)P(x_2/x_4, x_3, x_1)$$

# Permutation language model

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[ \sum_{t=1}^T \log p_{\theta}(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}}) \right]$$

- Maximize the expected log likelihood of training sequences w.r.t. all possible permutations of factorization order.
- Each position doesn't know in advance which other positions will be available. Thus, each position learns to utilize information from all other positions (bidirectional context).
- In practice, loss is calculated only for the last 14-17% of the positions in a permutation.

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[ \log p_{\theta}(\mathbf{x}_{\mathbf{z}_{>c}} \mid \mathbf{x}_{\mathbf{z}_{\leq c}}) \right] = \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[ \sum_{t=c+1}^{|\mathbf{z}|} \log p_{\theta}(x_{z_t} \mid \mathbf{x}_{\mathbf{z}_{<t}}) \right]$$

# Two streams

- In a traditional LM we always predict the content of the next position. The model naturally doesn't see this content.
- In a MLM we predict the content at the current position. The model (usually) doesn't see this content because it is masked.
- In a permutational LM we shall specify the position for which we want the content to be predicted, but without showing the content at that position!

# Two streams

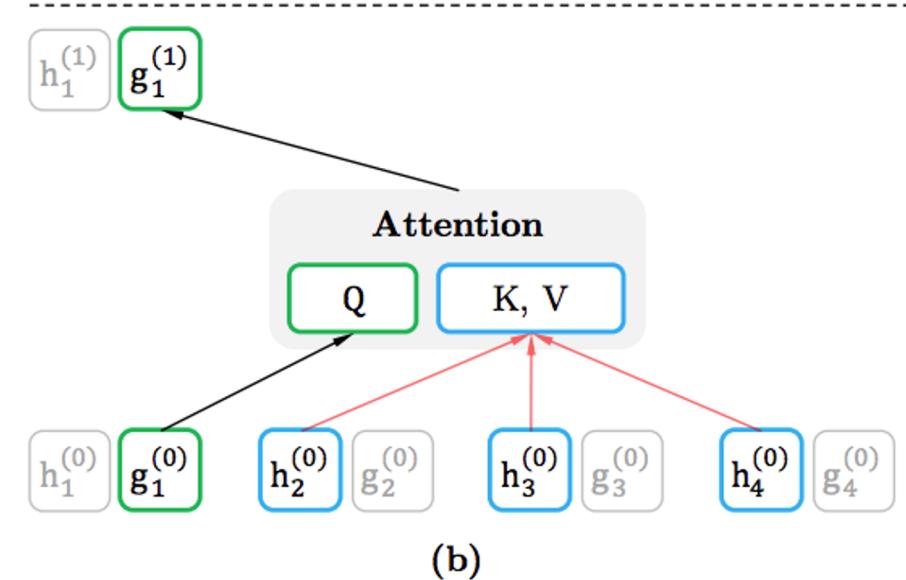
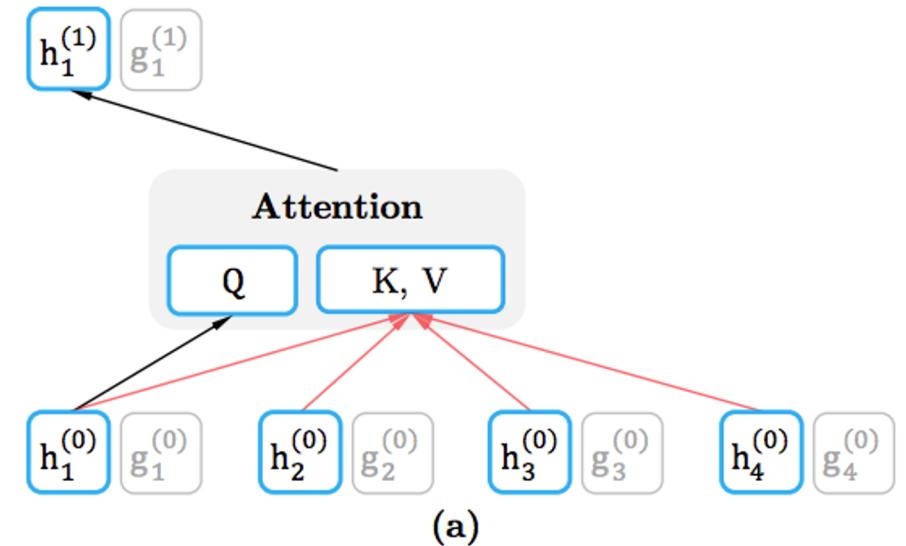
- Content stream  $h$  is similar to transformer's decoder
- Query stream  $g$  doesn't see content at the current position, only positional information
  - At fine-tuning, it can be dropped

$$g_i^{(0)} = w \quad \text{a trainable vector (independent of } x_i)$$

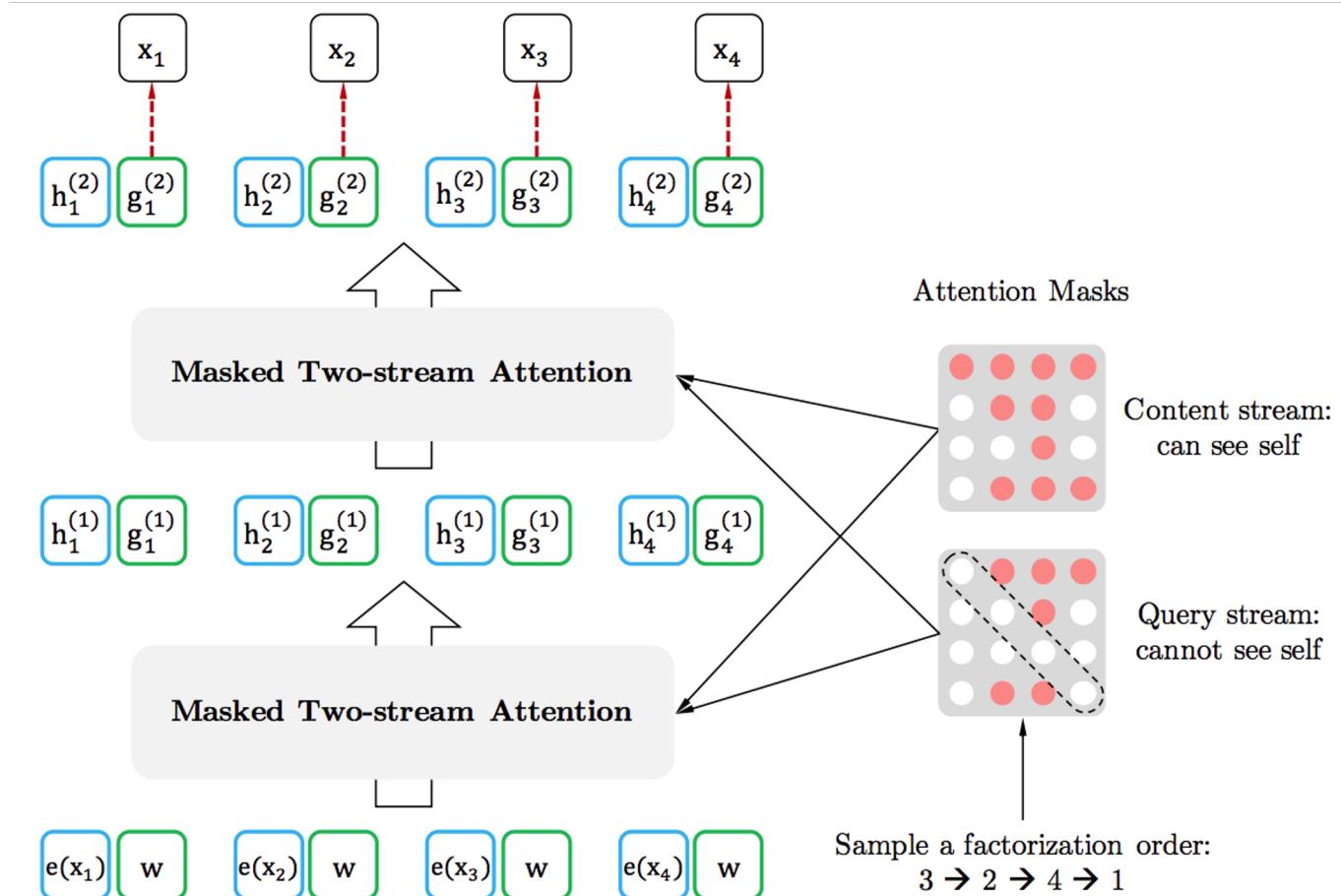
$$h_i^{(0)} = e(x_i)$$

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(Q = g_{z_t}^{(m-1)}, KV = h_{\mathbf{z}_{<t}}^{(m-1)}; \theta)$$

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(Q = h_{z_t}^{(m-1)}, KV = h_{\mathbf{z}_{\leq t}}^{(m-1)}; \theta)$$



# Two streams



# Other details

- Improvements from Transformer-XL
  - Relative positional embeddings
  - Segment recurrence
- Drops query stream when fine-tuning
- Similarly to BERT, segment-pair inputs from same or different documents, but no NSP.
- SentencePiece vocabulary of 32K subwords
- Following BERT, XLNet has 110M / 340M parameters for base / large
- Trained on 126GB of texts
  - bs=2048, 500K steps
  - 512 TPU v3 chips, 2.5 days
  - Underfitted the data, but continuing training didn't help for downstream tasks.

# XLNet vs. BERT

- Same training data as in BERT: Wikipedia + BooksCorpus
- Same hyperparameters for pretraining as in BERT:
- Model size: L = 24, H = 1024, A = 16
- Batch size: 256
- Number of steps: 1M

