Contextual word representations

Christopher Potts

Stanford Linguistics

CS 224U: Natural language understanding
May 11

Overview

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. RoBERTa: **R**obustly **o**ptimized **BERT** **a**pproach
6. ELECTRA: **E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately
7. XLNet
8. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

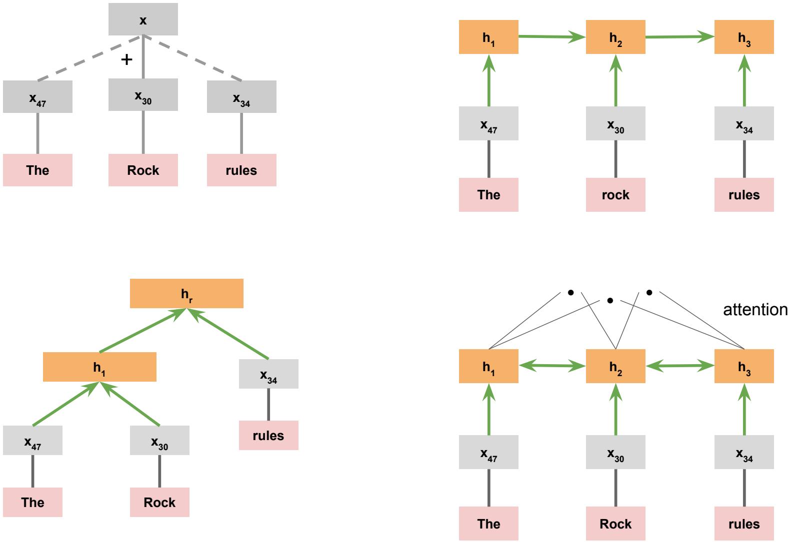
Associated materials

- Notebook: `contextualreps.ipynb`
- Smith 2019
- ELMo: Peters et al. 2018; [project site]
- Transformer
  1. Vaswani et al. 2017
  2. Alexander Rush: The Annotated Transformer [link]
  3. Hugging Face `transformers`: project site
     a. BERT: Devlin et al. 2019; project site
     b. RoBERTa: Liu et al. 2019; project site
     c. ELECTRA: Clark et al. 2019; project site
     d. XLNet: Yang et al. 2019; project site

Word representations and context

1. a. The vase broke.
   b. Dawn broke.
   c. The news broke.
   d. Sandy broke the world record.
   e. Sandy broke the law.
   f. The burgler broke into the house.
   g. The newscaster broke into the movie broadcast.
   h. We broke even.
2. a. flat tire/beer/note/surface
   b. throw a party/fight/ball/fit
3. a. A crane caught a fish.
   b. A crane picked up the steel beam.
   c. I saw a crane.
4. a. Are there typos? I didn't see any.
   b. Are there bookstores downtown? I didn't see any.

Model structure and linguistic structure



Guiding idea: Attention (from the NLI slides)

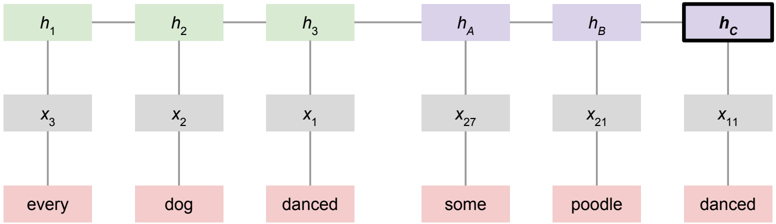
classifier $y = \textbf{softmax}(\tilde{h}W + b)$

attention combo $\tilde{h} = \tanh([\kappa; h_C]W_\kappa)$

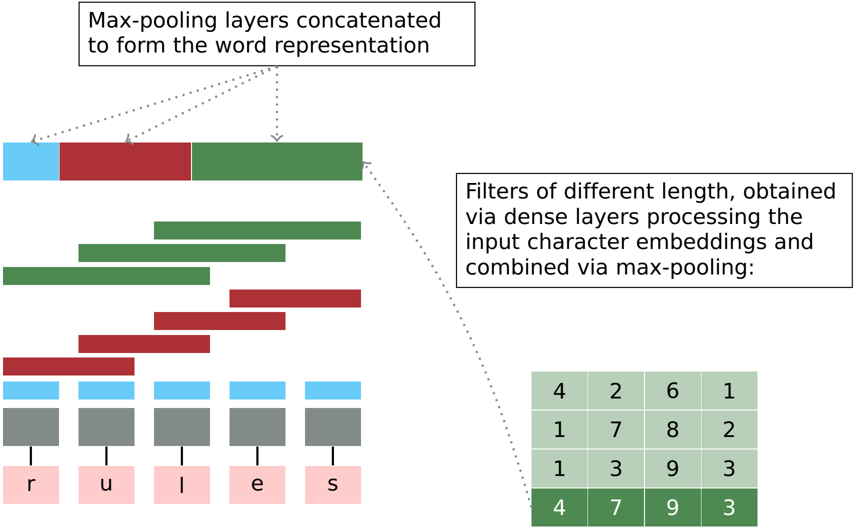
context $\kappa = \textbf{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3)$

attention weights $\alpha = \textbf{softmax}(\tilde{\alpha})$

scores $\tilde{\alpha} = \begin{bmatrix} h_C^\top h_1 & h_C^\top h_2 & h_C^\top h_3 \end{bmatrix}$



Guiding idea: Subword modeling

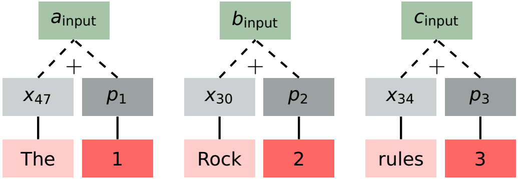


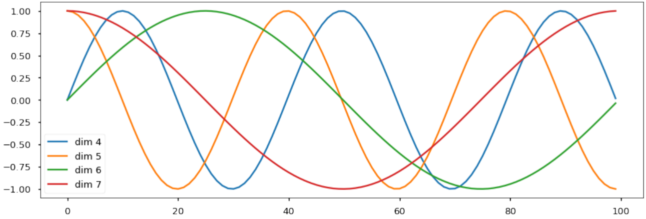
Guiding idea: Word piece tokenization

```
[1]: from transformers import BertTokenizer

[2]: tokenizer = BertTokenizer.from_pretrained('bert-base-cased')

[3]: tokenizer.tokenize("This isn't too surprising.")

[3]: ['This', 'isn', "'", 't', 'too', 'surprising', '.']

[4]: tokenizer.tokenize("Encode me!")

[4]: ['En', '##code', 'me', '!']

[5]: tokenizer.tokenize("Snuffleupagus?")

[5]: ['S', '##nu', '##ffle', '##up', '##agu', '##s', '?']

[6]: tokenizer.vocab_size

[6]: 28996
```

Sennrich et al. 2016,
https://github.com/google/sentencepiece

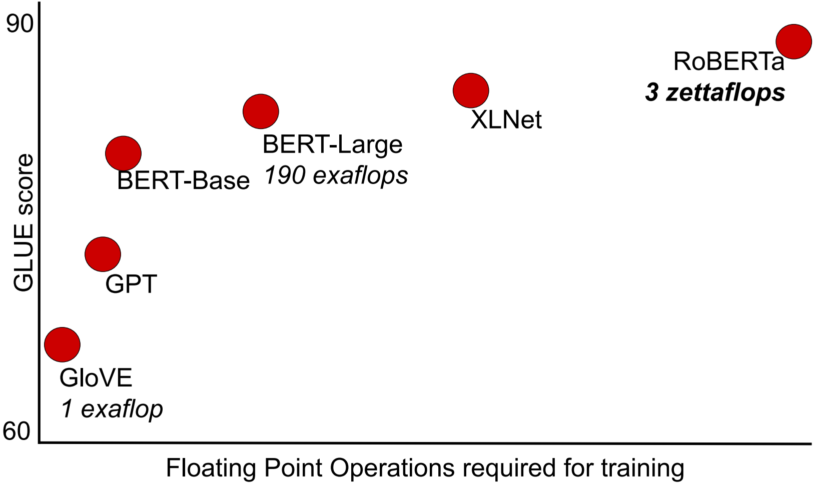
Guiding idea: Positional encoding





From 'The Annotated Transformer'

Current issues and efforts



Clark et al. 2019

Current issues and efforts

https://twitter.com/artetxem/status/1178794889229864962

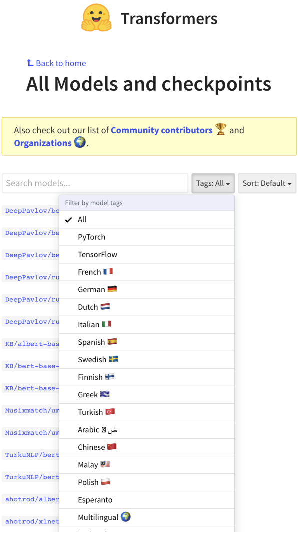
Current issues and efforts

Consumption	**CO_2e (lbs)**
Air travel, 1 person, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experiments	78,468
Transformer (big)	192
w/ neural arch. search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

Strubell et al. 2019

Current issues and efforts



https://huggingface.co

Current issues and efforts

Compressing Large-Scale Transformer-Based Models: A Case Study on BERT

Prakhar Ganesh[1], **Yao Chen**[1], **Xin Lou**[1], **Mohammad Ali Khan**[1], **Yin Yang**[2], **Deming Chen**[3], **Marianne Winslett**[3], **Hassan Sajjad**[4,2] and **Preslav Nakov**[4,2]
[1]Advanced Digital Sciences Center
[2]Hamad Bin Khalifa University
[3]University of Illinois at Urbana-Champaign
[4]Qatar Computing Research Institute
{prakhar.g, yao.chen, lou.xin, mohammad.k}@adsc-create.edu.sg,
{yyang, hsajjad, pnakov}@hbku.edu.qa, {dchen, winslett}@illinois.edu

Mitchell A. Gordon — About Blog Bookshelf

All The Ways You Can Compress BERT

Nov 18, 2019

Model compression reduces redundancy in a trained neural network. This is useful, since BERT barely fits on a GPU (BERT-Large does not) and definitely won't fit on your smart phone. Improved memory and inference speed efficiency can also save costs at scale.

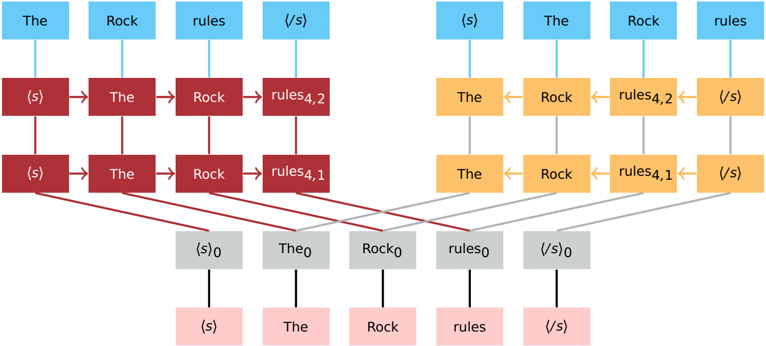
http://mitchgordon.me/

ELMo

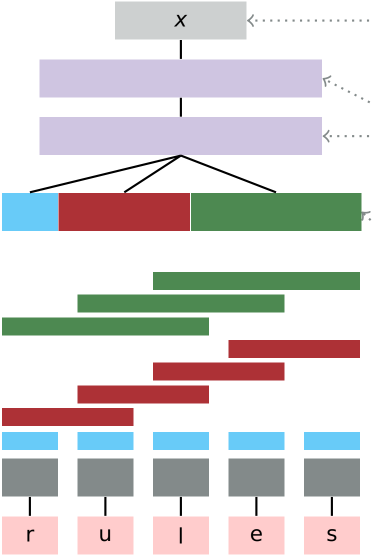
1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. RoBERTa: **R**obustly **o**ptimized **BERT** **a**pproach
6. ELECTRA: **E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately
7. XLNet
8. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

Core model structure

$$\boxed{\text{rules}} = s_0^{\text{task}} \cdot \text{rules}_0 + s_1^{\text{task}} \cdot \text{rules}_{4,1}\,\text{rules}_{4,1} + s_2^{\text{task}} \cdot \text{rules}_{4,2}\,\text{rules}_{4,2}$$



Word embeddings



A final linear projection into the embedding dimensionality, which must be twice the RNN hidden dimensionality

Highway layers introduce gating information between layers

A series of convolutional filters with max pooling, concatenated to form the initial representation.

ELMo model releases

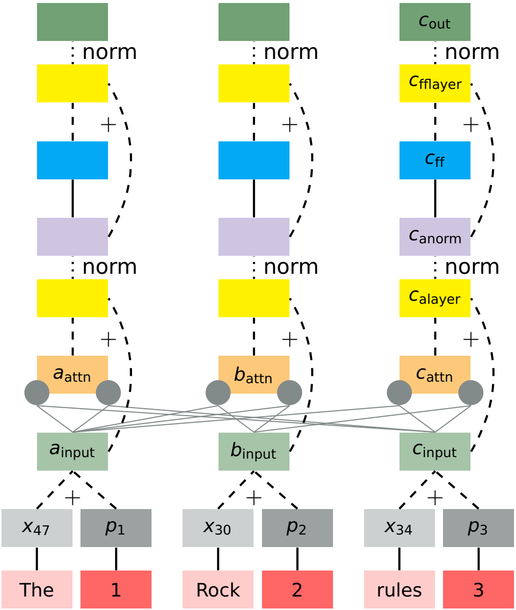
		LSTM		
Model	**Parameters**	**Hidden size**	**Output size**	**Highway layers**
Small	13.6M	1024	128	1
Medium	28.0M	2048	256	1
Original	93.6M	4096	512	2
Original (5.5B)	93.6M	4096	512	2

Additional details at `https://allennlp.org/elmo`; the options files reveal additional information about the subword convolutional filters, activation functions, thresholds, and layer dimensions.

Transformers

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. RoBERTa: **R**obustly **o**ptimized **BERT** **a**pproach
6. ELECTRA: **E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately
7. XLNet
8. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

Core model structure



$$c_{out} = \frac{c_{fflayer} - \textbf{mean}(c_{fflayer})}{\textbf{std}(c_{fflayer}) + \varepsilon}$$

$$c_{fflayer} = c_{anorm} + \textbf{Dropout}(c_{ff})$$

$$c_{ff} = \textbf{ReLU}(c_{anorm} W_1 + b_1) W_2 + b_2$$

$$c_{anorm} = \frac{c_{alayer} - \textbf{mean}(c_{alayer})}{\textbf{std}(c_{alayer}) + \varepsilon}$$

$$c_{alayer} = \textbf{Dropout}\left(c_{attn} + c_{input}\right)$$

$$c_{attn} = \textbf{sum}\left(\left[\alpha_1 a_{input}, \alpha_2 b_{input}\right]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{{c_{input}}^{\top} a_{input}}{\sqrt{d_k}}, \frac{{c_{input}}^{\top} b_{input}}{\sqrt{d_k}}\right]$$

$$c_{input} = x_{34} + p_3$$

Computing the attention representations

Calculation as previously given

$$c_{\text{attn}} = \textbf{sum}\left(\left[\alpha_1 a_{\text{input}}, \alpha_2 b_{\text{input}}\right]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{{c_{\text{input}}}^{\top} a_{\text{input}}}{\sqrt{d_k}}, \frac{{c_{\text{input}}}^{\top} b_{\text{input}}}{\sqrt{d_k}}\right]$$

Matrix format

$$\textbf{softmax}\left(\frac{c_{\text{input}} \begin{bmatrix} a_{\text{input}} \\ b_{\text{input}} \end{bmatrix}^{\top}}{\sqrt{d_k}}\right) \begin{bmatrix} a_{\text{input}} \\ b_{\text{input}} \end{bmatrix}$$

Computing the attention representations

```
[1]: import numpy as np

[2]: seq_length = 3
     d_k = 4

[3]: inputs = np.random.uniform(size=(seq_length, d_k))
     inputs

[3]: array([[0.31436922, 0.66969307, 0.270804  , 0.72023504],
            [0.87180132, 0.27637445, 0.43091867, 0.34138704],
            [0.20292054, 0.6345131 , 0.01058343, 0.22846636]])

[4]: a_input = inputs[0]
     b_input = inputs[1]
     c_input = inputs[2]
```

Computing the attention representations

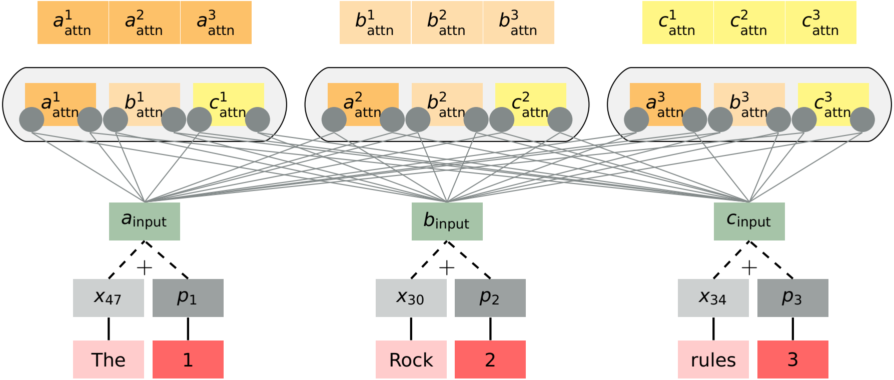
```
[5]: def softmax(X):
         z = np.exp(X)
         return (z / z.sum(axis=0)).T

[6]: c_alpha = softmax([
         (c_input.dot(a_input) / np.sqrt(d_k)),
         (c_input.dot(b_input) / np.sqrt(d_k))])

[7]: c_attn = sum([c_alpha[0]*a_input, c_alpha[1]*b_input])
     c_attn

[7]: array([0.57768027, 0.48390338, 0.34643646, 0.54128076])

[8]: ab = inputs[:-1]

[9]: softmax(c_input.dot(ab.T) / np.sqrt(d_k)).dot(ab)

[9]: array([0.57768027, 0.48390338, 0.34643646, 0.54128076])

[10]: # If we allow every input to attend to itself:
      softmax(inputs.dot(inputs.T) / np.sqrt(d_k)).dot(inputs)

[10]: array([[0.4614388 , 0.53204444, 0.2451212 , 0.45136127],
             [0.50173123, 0.50618272, 0.26184404, 0.43678288],
             [0.45493467, 0.5332328 , 0.23643403, 0.4388242 ]])
```

Multi-headed attention

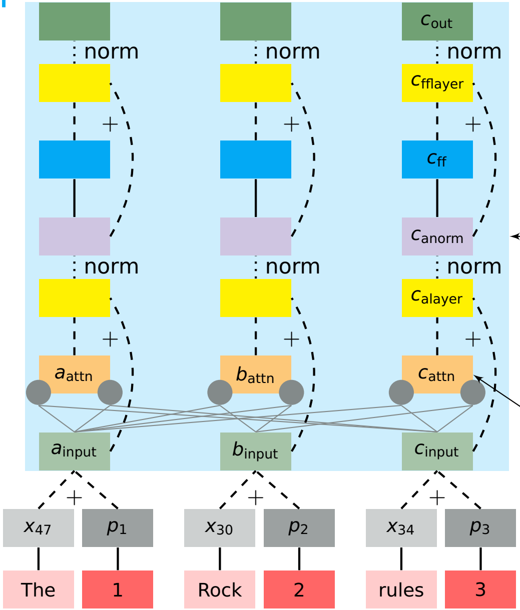
$$c^3_{\text{attn}} = \textbf{sum}\left(\left[\alpha_1(a_{\text{input}} W^V_3), \alpha_2(b_{\text{input}} W^V_3)\right]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{(c_{\text{input}} W^Q_3)^\top (a_{\text{input}} W^K_3)}{\sqrt{d_k}}, \frac{(c_{\text{input}} W^Q_3)^\top (b_{\text{input}} W^K_3)}{\sqrt{d_k}}\right]$$



Repeated transformer blocks



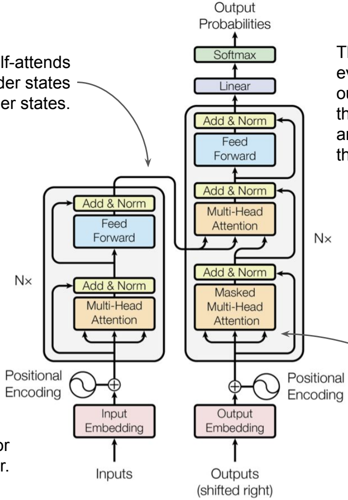
Repeated 6 times with c_{out} serving as c_{input} at each successive layer.

Includes multi-headed attention in each block

The architecture diagram

Each decoder state self-attends with all of its fellow decoder states and with all the encoder states.

The right side is repeated for every decoder state, with outputs for each state that has them (all of them for dialogue and machine translation, only the final one for NLI).



The left side is repeated for every state in the encoder.

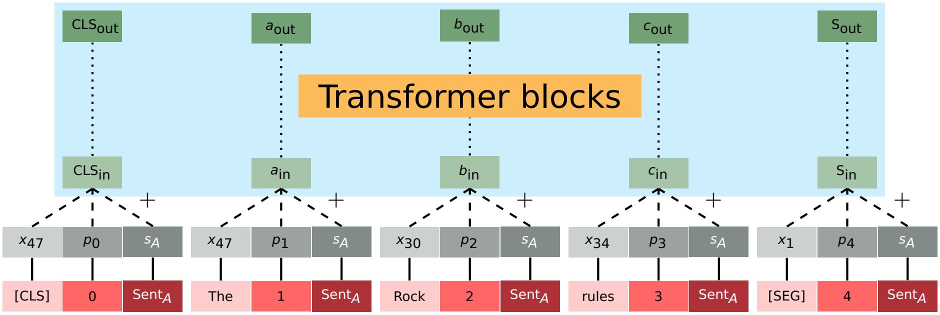
In the decoder, self-attention is limited to preceding words.

Figure 1: The Transformer - model architecture.

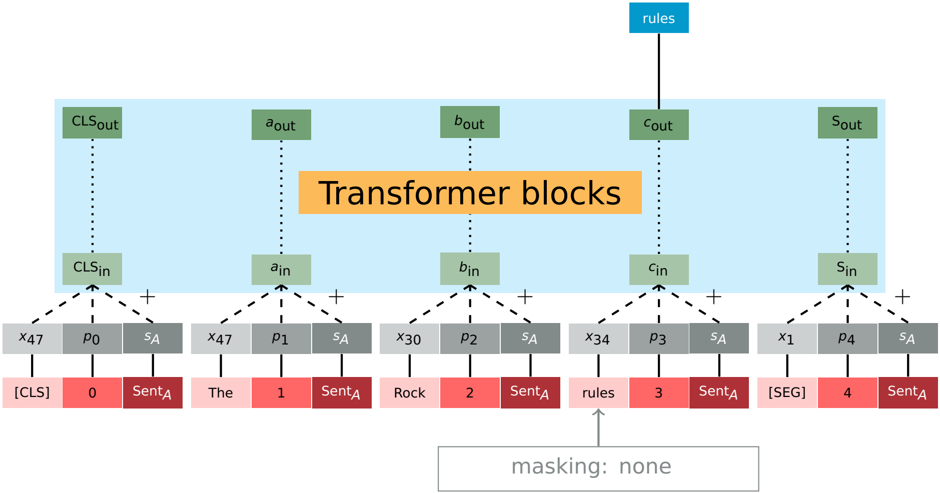
BERT

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. RoBERTa: **R**obustly **o**ptimized **BERT** **a**pproach
6. ELECTRA: **E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately
7. XLNet
8. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

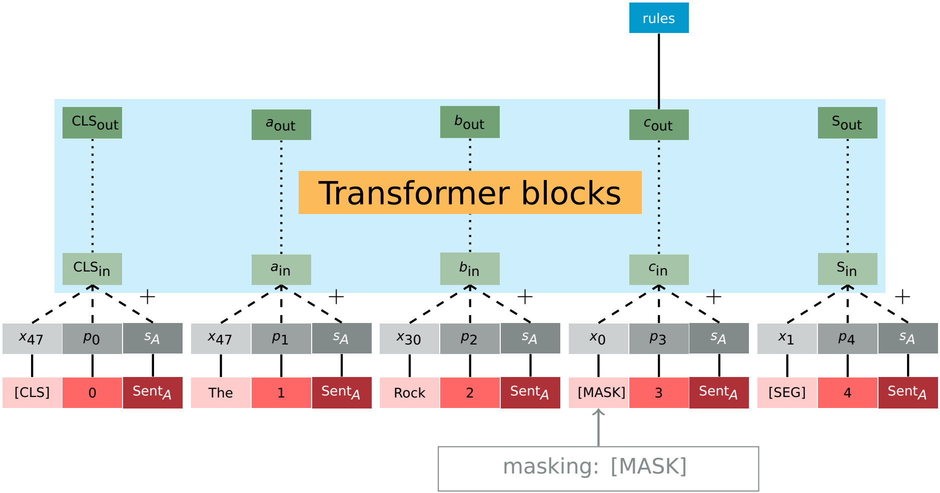
Core model structure



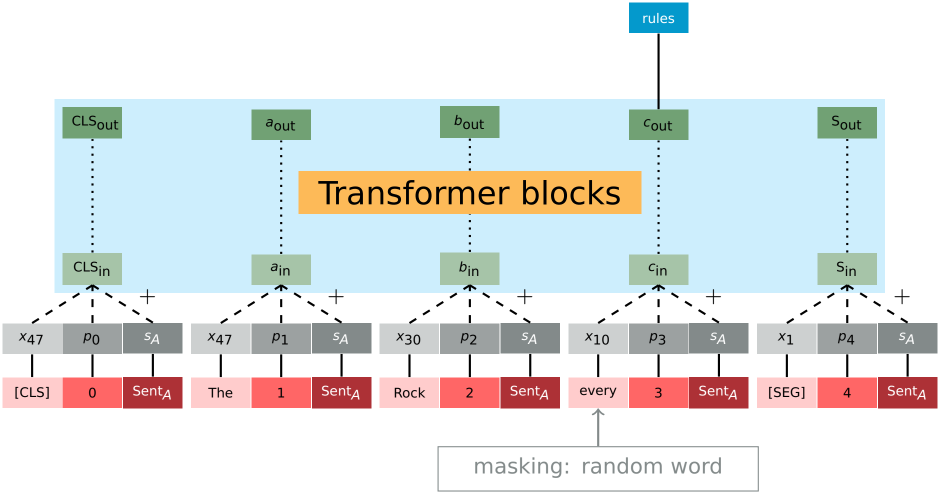
Masked Language Modeling (MLM)



Masked Language Modeling (MLM)



Masked Language Modeling (MLM)



MLM loss function

For Transformer parameters H_θ and sequence $\mathbf{x} = [x_1, \ldots, x_T]$ with masked version $\hat{\mathbf{x}}$:

$$\max_\theta \sum_{t=1}^{T} m_t \log \frac{\exp\left(e(x_t)^\top H_\theta(\hat{\mathbf{x}})_t\right)}{\sum_{x' \in \mathcal{V}} \exp\left(e(x')^\top H_\theta(\hat{\mathbf{x}})_t\right)}$$

where $\mathcal{V}$ is the vocabulary, x_t is the actual token at step t, $m_t = 1$ if token t was masked, else 0, and $e(x)$ is the embedding for x.

Binary sentence prediction pretraining

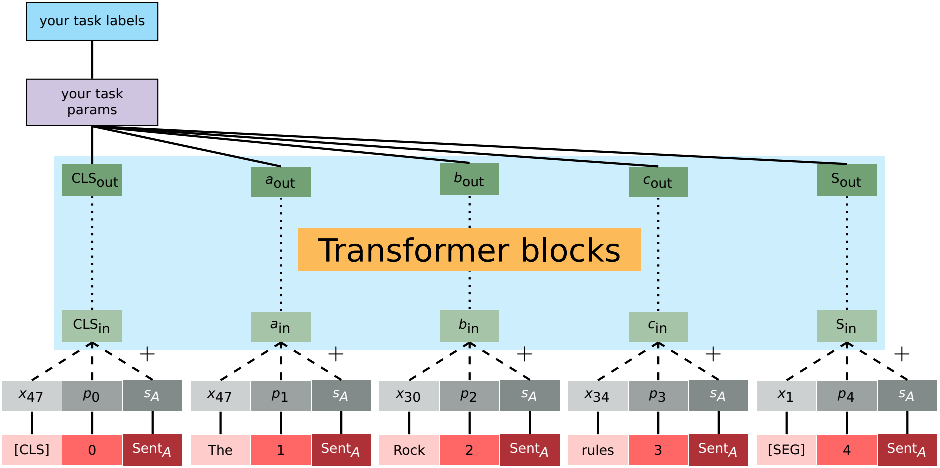
Positive: Actual sentence sequences

- [CLS] the man went to [MASK] store [SEP]
- he bought a gallon [MASK] milk [SEP]
- Label: `IsNext`

Negative: Randomly chosen second sentence

- [CLS] the man went to [MASK] store [SEP]
- penguin [MASK] are flight ##less birds [SEP]
- Label: `NotNext`

Transfer learning and fine-tuning



Tokenization and the BERT embedding space

```
[1]: from transformers import BertTokenizer

[2]: tokenizer = BertTokenizer.from_pretrained('bert-base-cased')

[3]: tokenizer.tokenize("This isn't too surprising.")

[3]: ['This', 'isn', "'", 't', 'too', 'surprising', '.']

[4]: tokenizer.tokenize("Encode me!")

[4]: ['En', '##code', 'me', '!']

[5]: tokenizer.tokenize("Snuffleupagus?")

[5]: ['S', '##nu', '##ffle', '##up', '##agu', '##s', '?']

[6]: tokenizer.vocab_size

[6]: 28996
```

Initial BERT model releases

Base

- Transformer layers: 12
- Hidden representations: 768 dimensions
- Attention heads: 12
- Total parameters: 110M

Large

- Transformer layers: 24
- Hidden representations: 1024 dimensions
- Attention heads: 16
- Total parameters: 340M

Limited to sequences of 512 tokens due to dimensionality of the positional embeddings.

Many new releases at the project site and on Hugging Face.

Efforts to make BERT smaller

Efforts to make BERT smaller

Mitchell A. Gordon

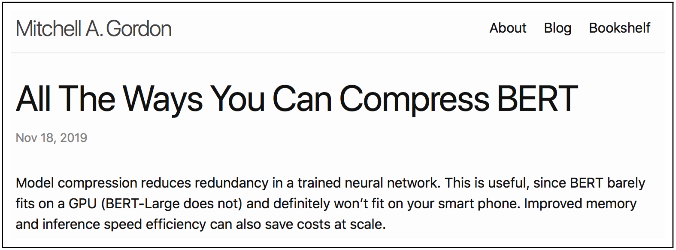
About Blog Bookshelf

All The Ways You Can Compress BERT

Nov 18, 2019

Model compression reduces redundancy in a trained neural network. This is useful, since BERT barely fits on a GPU (BERT-Large does not) and definitely won't fit on your smart phone. Improved memory and inference speed efficiency can also save costs at scale.

Efforts to make BERT smaller


Mitchell A. Gordon

About Blog Bookshelf

All The Ways You Can Compress BERT

Nov 18, 2019

Model compression reduces redundancy in a trained neural network. This is useful, since BERT barely fits on a GPU (BERT-Large does not) and definitely won't fit on your smart phone. Improved memory and inference speed efficiency can also save costs at scale.

Particularly relevant to this lecture:

- Sanh et al. (2019): DistilBERT
- Michel et al. (2019): Fewer attention heads
- Lan et al. (2019): ALBERT

Known limitations with BERT

1. Devlin et al. (2019:§5): admirably detailed but still partial ablation studies and optimization studies.
2. Devlin et al. (2019): “The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning.”
3. Devlin et al. (2019): “The second downside of using an MLM is that only 15% of tokens are predicted in each batch”
4. Yang et al. (2019): “BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language”

RoBERTa

Addressing the known limitations with BERT

1. Devlin et al. (2019:§5): admirably detailed but still partial ablation studies and optimization studies.
2. Devlin et al. (2019): "The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning."
3. Devlin et al. (2019): "The second downside of using an MLM is that only 15% of tokens are predicted in each batch"
4. Yang et al. (2019): "BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language"

Robustly optimized BERT approach

BERT	RoBERTa
Static masking/substitution	Dynamic masking/substitution
Inputs are two concatenated document segments	Inputs are sentence sequences that may span document boundaries
Next Sentence Prediction (NSP)	No NSP
Training batches of 256 examples	Training batches of 2,000 examples
Word-piece tokenization	Character-level byte-pair encoding
Pretraining on BooksCorpus and English Wikipedia	Pretraining on BooksCorpus, CC-News, OpenWebText, and Stories
Train for 1M steps	Train for up to 500K steps
Train on short sequences first	Train only on full-length sequences

Additional differences in the optimizer and data presentation (sec 3.1).

RoBERTa results informing final system design

Masking	**SQuAD 2.0**	**MNLI-m**	**SST-2**
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Table 1: Comparison between static and dynamic masking for $\text{BERT}_{\text{BASE}}$. 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).

RoBERTa results informing final system design

RoBERTa choice for efficient batching, and comparisons with related work.

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (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 $\text{BERT}_{\text{BASE}}$ and $\text{XLNet}_{\text{BASE}}$ are from Yang et al. (2019).

RoBERTa results informing final system design

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
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.

RoBERTa results informing final system design

Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of $\text{BERT}_{\text{LARGE}}$. Results for $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.

Related work

A Primer in BERTology: What we know about how BERT works

Anna Rogers, Olga Kovaleva, Anna Rumshisky
Department of Computer Science, University of Massachusetts Lowell
Lowell, MA 01854
{arogers, okovalev, arum}@cs.uml.edu

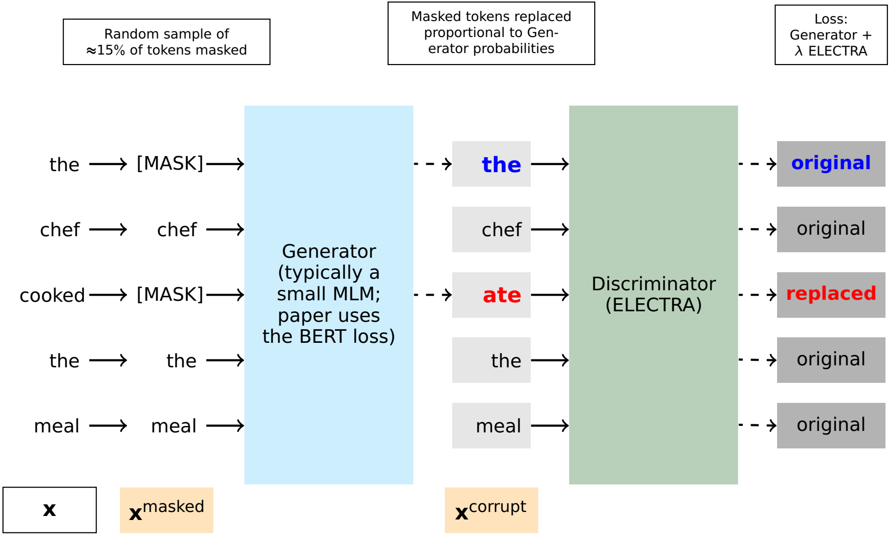
ELECTRA

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. RoBERTa: **R**obustly **o**ptimized **BERT** **a**pproach
6. ELECTRA: **E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately
7. XLNet
8. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

Addressing the known limitations with BERT

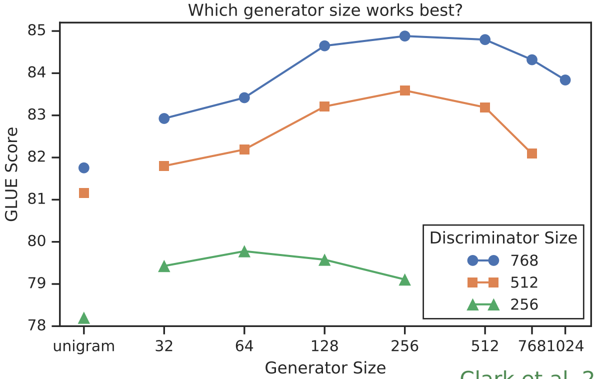
1. Devlin et al. (2019:§5): admirably detailed but still partial ablation studies and optimization studies.
2. Devlin et al. (2019): "The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning."
3. Devlin et al. (2019): "The second downside of using an MLM is that only 15% of tokens are predicted in each batch"
4. Yang et al. (2019): "BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language"

Core model structure (Clark et al. 2019)



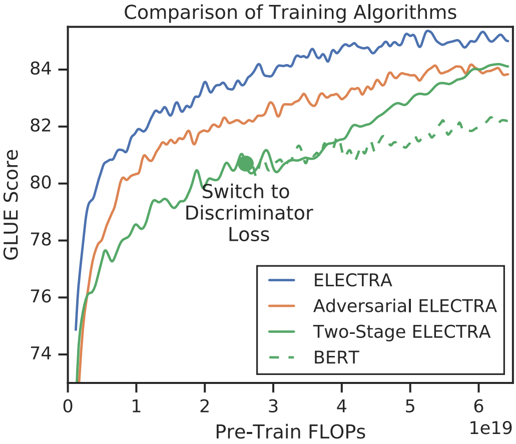
Generator/Discriminator relationships

Where Generator and Discriminator are the same size, they can share Transformer parameters, and more sharing is better. However, the best results come from having a Generator that is small compared to the Discriminator:



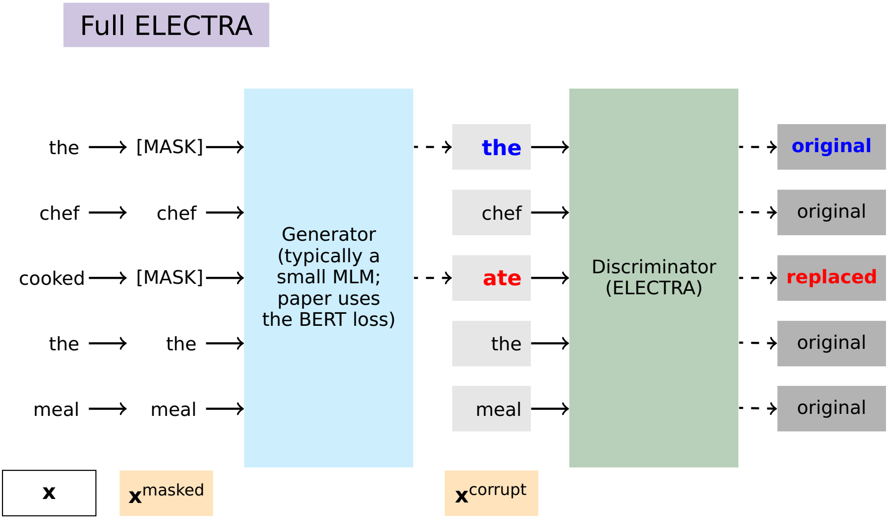
Clark et al. 2019, Figure 3

Efficiency



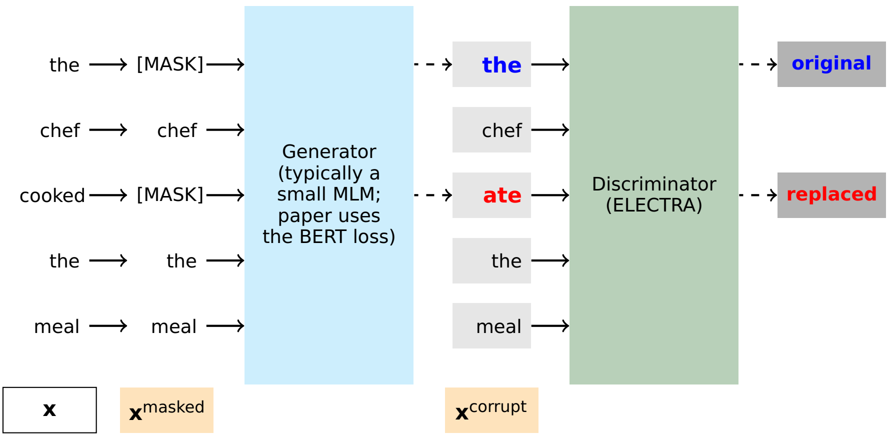
Clark et al. 2019, Figure 3

ELECTRA efficiency analyses



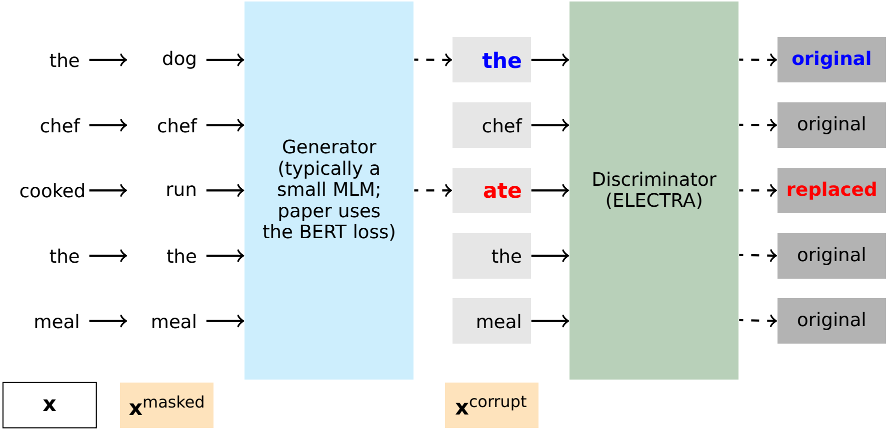
ELECTRA efficiency analyses

ELECTRA 15%

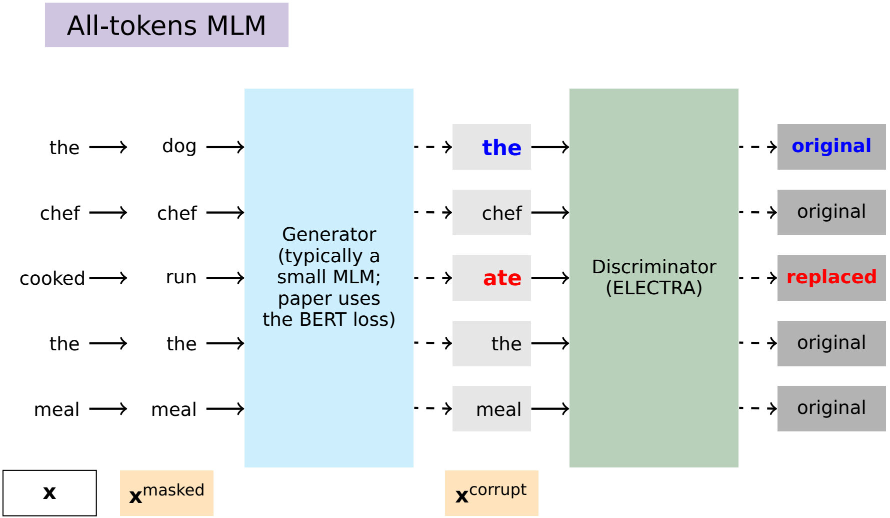


ELECTRA efficiency analyses

Replace MLM



ELECTRA efficiency analyses



ELECTRA efficiency analyses

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

ELECTRA model releases

Available from the project site:

Model	Layers	Hidden Size	Params	GLUE test
Small	12	256	14M	77.4
Base	12	768	110M	82.7
Large	24	1024	335M	85.2

'Small' is the model designed to be "quickly trained on a single GPU".

XLNet

Addressing the known limitations with BERT

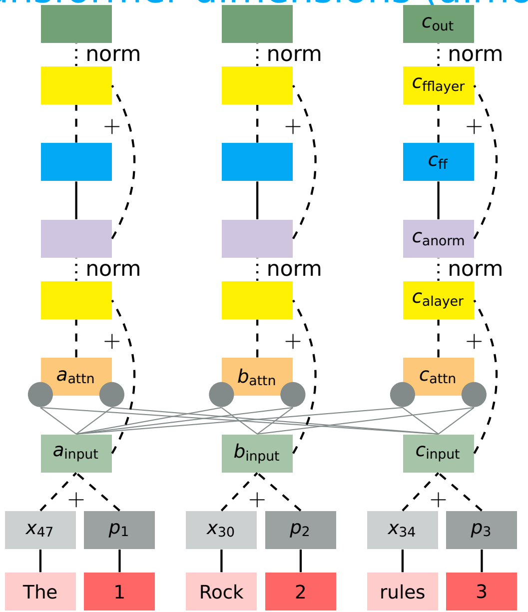
1. Devlin et al. (2019:§5): admirably detailed but still partial ablation studies and optimization studies.

2. Devlin et al. (2019): "The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning."

3. Devlin et al. (2019): "The second downside of using an MLM is that only 15% of tokens are predicted in each batch"

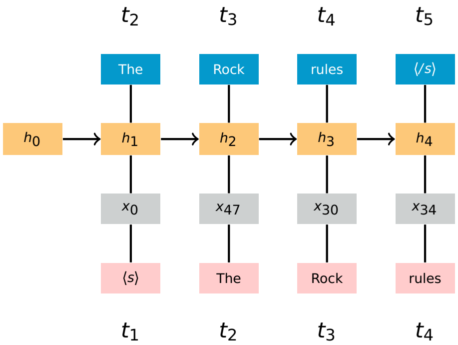
4. Yang et al. (2019): "BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language"

Transformer dimensions (almost) independent



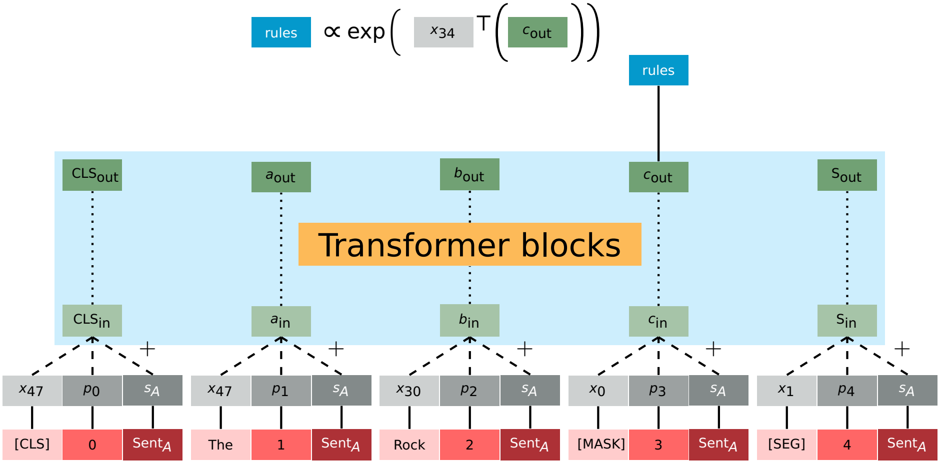
The order of the positions doesn't matter except for the positional encodings at the bottom.

Conditional language modeling



$$\text{Rock} \propto \exp\left(x_{30}^{\top}\left(h_2\right)\right)$$

Comparison with BERT



The two objective functions

For vocabulary $\mathcal{V}$, sequence $\mathbf{x} = [x_1, \ldots, x_T]$, and word-level embedding e:

Language model

$$\max_{\theta} \sum_{t=1}^{T} \log \frac{\exp\left(e(x_t)^\top h_\theta(\mathbf{x}_{1:t-1})\right)}{\sum_{x' \in \mathcal{V}} \exp\left(e(x')^\top h_\theta(\mathbf{x}_{1:t-1})\right)}$$

for RNN parameters h_θ.

BERT

$$\max_{\theta} \sum_{t=1}^{T} m_t \log \frac{\exp\left(e(x_t)^\top H_\theta(\hat{\mathbf{x}})_t\right)}{\sum_{x' \in \mathcal{V}} \exp\left(e(x')^\top H_\theta(\hat{\mathbf{x}})_t\right)}$$

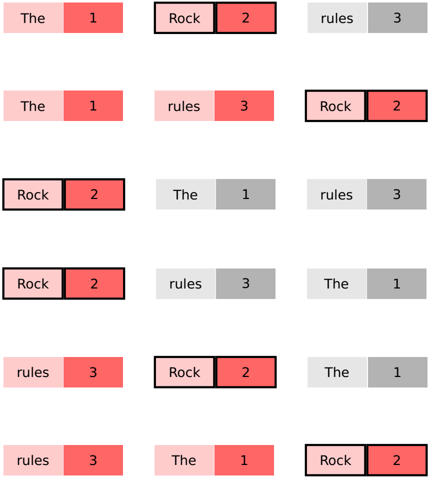
for Transformer parameters H_θ, with $m_t = 1$ if token t was masked, else 0.

Permutation orders

The	1	Rock	2	rules	3
The	1	rules	3	Rock	2
Rock	2	The	1	rules	3
Rock	2	rules	3	The	1
rules	3	Rock	2	The	1
rules	3	The	1	Rock	2

Yang et al. 2019:§2.2

Permutation orders



Yang et al. 2019:§2.2

XLNet permutation orders

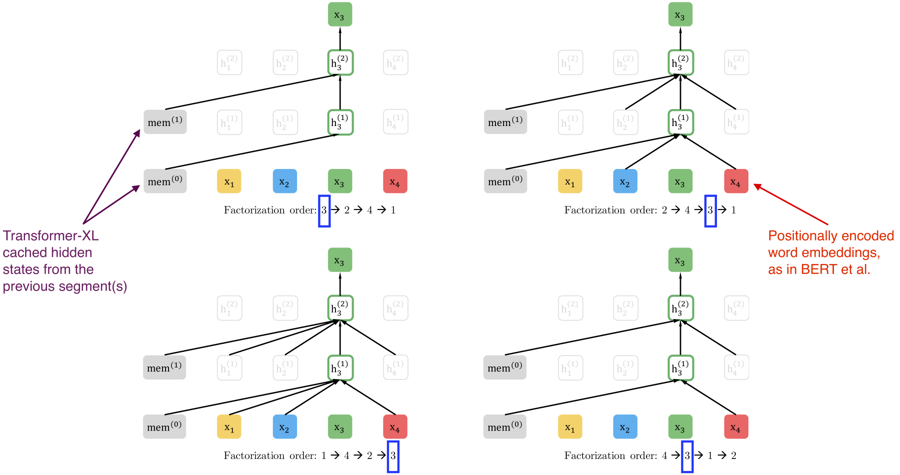
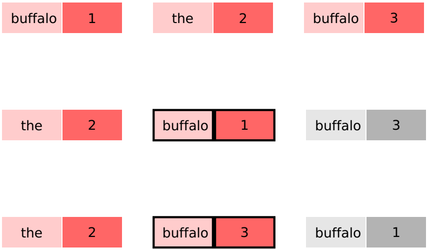


Figure 4: Illustration of the permutation language modeling objective for predicting x_3 given the same input sequence $\mathbf{x}$ but with different factorization orders.

Yang et al. 2019:§A.7

Lack of sensitivity to the target position

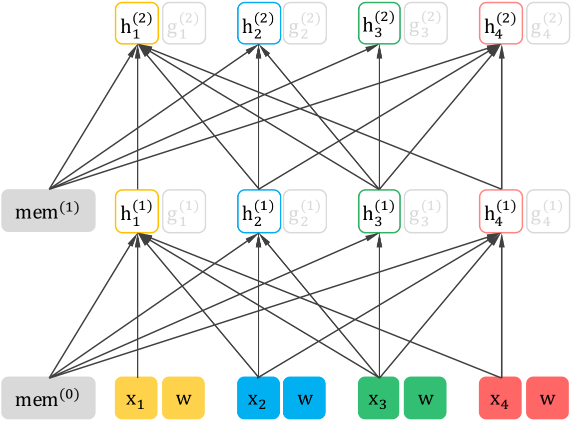


$$\max_{\theta} \sum_{t=1}^{T} \log \frac{\exp\left(e(x_t)^{\top} h_{\theta}(\mathbf{x}_{1:t-1})\right)}{\sum_{x' \in \mathcal{V}} \exp\left(e(x')^{\top} h_{\theta}(\mathbf{x}_{1:t-1})\right)}$$

Yang et al. 2019:§2.2, A.1

Two-stream attention: order 3 → 2 → 4 → 1

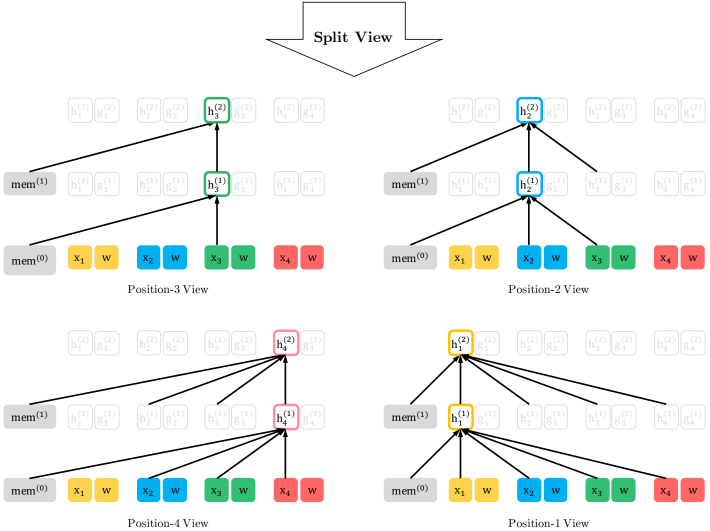
Content stream



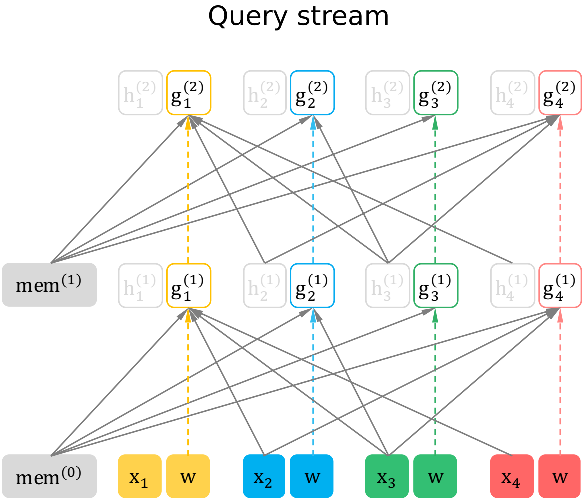
Joint View of the Content Stream
(Factorization order: 3 → 2 → 4 → 1)

Two-stream attention: order 3 → 2 → 4 → 1

Content stream

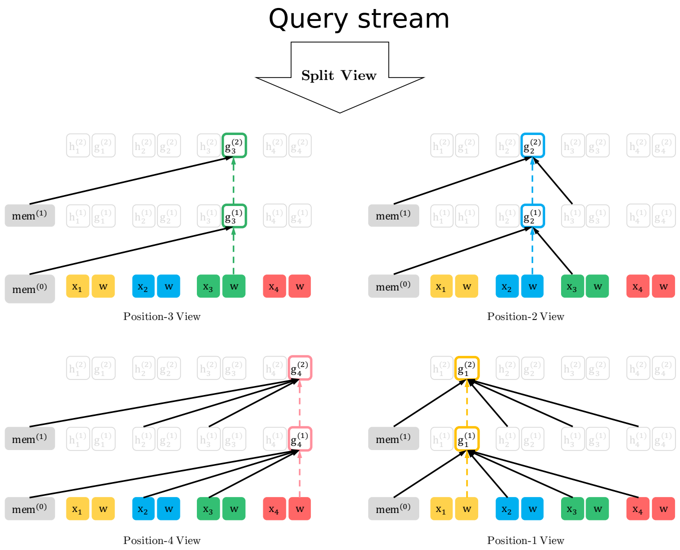


Two-stream attention: order 3 → 2 → 4 → 1

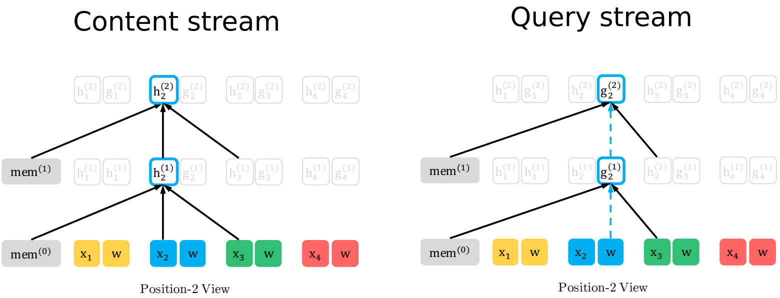


Joint View of the Query Stream
(Factorization order: 3 → 2 → 4 → 1)

Two-stream attention: order 3 → 2 → 4 → 1



Two-stream attention: order 3 → 2 → 4 → 1



Yang et al. 2019:§2.2, A.7

XLNet model releases

From `https://github.com/zihangdai/xlnet`:

Model	Layers	Hidden Size	Heads
Large, Cased	24	1024	16
Base, Cased	12	768	12

See also `https://huggingface.co/models?search=xlnet`

Conditional dependencies

For sampled permutation order [is, a, city, New, York] and prediction targets {New, York}:

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}),$$

$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city}).$$

Yang et al. 2019:§2.6

contextualreps.ipynb

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. RoBERTa: **R**obustly **o**ptimized **BERT a**pproach
6. ELECTRA: **E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately
7. XLNet
8. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

Guiding idea

1. Your existing architecture can benefit from contextual representations.

2. `contextualreps.ipynb` shows you how to bring in ELMo and BERT representations:
   - Simple featurization
   - Fine-tuning

3. By extending existing PyTorch modules for this course, you can create *customized* fine-tuning models with just a few lines of code.

4. (This is possible only because of the amazing work that the Hugging Face and AllenNLP groups have done.)!

Standard RNN dataset preparation

Examples

[a, b, a]
[b, c]

$\Downarrow$

Embedding			
1	−0.42	0.10	0.12
2	−0.16	−0.21	0.29
3	−0.26	0.31	0.37

Indices

[1, 2, 1]
[2, 3]

$\Downarrow$

Vectors

$$\Big[[-0.42\ 0.10\ 0.12], [-0.16\ -0.21\ 0.29], [-0.42\ 0.10\ 0.12]\Big]$$

$$\Big[[-0.16\ -0.21\ 0.29], [-0.26\ 0.31\ 0.37]\Big]$$

RNN contextual representation inputs

Examples

[a, b, a]
[b, c]

$$\Downarrow$$

Vectors

$$\Big[[-0.41\ -0.08\ 0.27], [0.17\ -0.22\ 0.78][-0.46\ 0.24\ 0.12]\Big]$$

$$\Big[[-0.02\ -0.56\ 0.11][-0.45\ 0.43\ 0.32]\Big]$$

Code snippet: ELMo RNN inputs

[1]:
```
from allennlp.commands.elmo import ElmoEmbedder
from torch_rnn_classifier import TorchRNNClassifier
import os, sst
```

[2]:
```
s3="https://allennlp.s3.amazonaws.com/models/elmo/2x4096_512_2048cnn_2xhighway/"
options_file = s3 + "elmo_2x4096_512_2048cnn_2xhighway_options.json"
weights_file = s3 + "elmo_2x4096_512_2048cnn_2xhighway_weights.hdf5"
```

[3]:
```
SST_HOME = os.path.join("data", "trees")
```

[4]:
```
elmo_embedder = ElmoEmbedder(options_file, weights_file)
```

[5]:
```
def elmo_sentence_phi(tree):
    vecs = elmo_embedder.embed_sentence(tree.leaves())
    return vecs[-1]
```

[6]:
```
def fit_prefeaturized_rnn(X, y):
    mod = TorchRNNClassifier(
        vocab=[],
        max_iter=50,
        use_embedding=False)
    mod.fit(X, y)
    return mod
```

[7]:
```
_ = sst.experiment(
    SST_HOME,
    elmo_sentence_phi,
    fit_prefeaturized_rnn,
    train_reader=sst.train_reader,
    assess_reader=sst.dev_reader,
    class_func=sst.ternary_class_func,
    vectorize=False)
```

Code snippet: BERT RNN inputs

```
[1]: import torch
     from torch_rnn_classifier import TorchRNNClassifier
     from transformers import BertModel, BertTokenizer
     import os, sst

[2]: SST_HOME = os.path.join("data", "trees")

[3]: hf_weights_name = 'bert-base-cased'

[4]: hf_tokenizer = BertTokenizer.from_pretrained(hf_weights_name)

[5]: hf_model = BertModel.from_pretrained(hf_weights_name)

[6]: def hugging_face_bert_phi(tree):
         s = " ".join(tree.leaves())
         input_ids = hf_tokenizer.encode(s, add_special_tokens=True)
         X = torch.tensor([input_ids])
         with torch.no_grad():
             final_hidden_states, cls_output = hf_model(X)
             return final_hidden_states.squeeze(0).numpy()

[7]: def fit_prefeaturized_rnn(X, y):
         mod = TorchRNNClassifier(
             vocab=[],
             max_iter=50,
             use_embedding=False)
         mod.fit(X, y)
         return mod

[8]: experiment = sst.experiment(
         SST_HOME,
         hugging_face_bert_phi,
         fit_prefeaturized_rnn,
         train_reader=sst.train_reader,
         assess_reader=sst.dev_reader,
         class_func=sst.ternary_class_func,
         vectorize=False)  # Pass in the BERT hidden states directly!
```

Code snippet: ELMo fine-tuning with AllenNLP

[1]:
```python
from allennlp.modules.elmo import Elmo, batch_to_ids
import torch
import torch.nn as nn
from torch_rnn_classifier import TorchRNNClassifier, TorchRNNClassifierModel
import os, sst
```

[2]:
```python
s3="https://allennlp.s3.amazonaws.com/models/elmo/2x4096_512_2048cnn_2xhighway/"
options_file = s3 + "elmo_2x4096_512_2048cnn_2xhighway_options.json"
weights_file = s3 + "elmo_2x4096_512_2048cnn_2xhighway_weights.hdf5"
```

[3]:
```python
class ElmoRNNClassifierModel(TorchRNNClassifierModel):
    def __init__(self, options_file, weights_file,
            hidden_dim, output_dim, bidirectional, device):
        super().__init__(vocab_size=0,
            embed_dim=1024, # self.elmo.get_output_dim()
            use_embedding=False, embedding=None,
            hidden_dim=hidden_dim, output_dim=output_dim,
            bidirectional=bidirectional, device=device)
        self.options_file = options_file
        self.weights_file = weights_file
        self.elmo = Elmo(
            self.options_file,
            self.weights_file,
            num_output_representations=2,
            dropout=0)

    def forward(self, X, seq_lengths):
        X = X.to(self.device, non_blocking=True)
        result = self.elmo(X)
        X = result['elmo_representations'][-1]
        state = self.rnn_forward(X, seq_lengths, self.rnn)
        logits = self.classifier_layer(state)
        return logits
```

Code snippet: ELMo fine-tuning with AllenNLP

```
[4]: class ElmoRNNClassifier(TorchRNNClassifier):
         def __init__(self, options_file, weights_file, *args, **kwargs):
             self.options_file = options_file
             self.weights_file = weights_file
             vocab = []
             super().__init__(
                 vocab, *args, use_embedding=False, embedding=None, **kwargs)

         def build_graph(self):
             elmo = ElmoRNNClassifierModel(
                 options_file=self.options_file,
                 weights_file=self.weights_file,
                 hidden_dim=self.hidden_dim,
                 output_dim=self.n_classes_,
                 bidirectional=self.bidirectional,
                 device=self.device)
             elmo.train()
             return elmo

         def _prepare_dataset(self, X):
             seq_lengths = [sum([1 for w in ex if w.sum() > 0]) for ex in X]
             return X, torch.tensor(seq_lengths)

         @staticmethod
         def encode(X):
             return batch_to_ids(X)
```

```
[5]: mod = ElmoRNNClassifier(
         options_file,
         weights_file,
         batch_size=16,
         max_iter=10,  # More iters improves things. How many did the ELMo team do?
         eta=0.0001,
         l2_strength=0.0001)
```

Code: BERT fine-tuning with Hugging Face

[1]:
```
import torch
import torch.nn as nn
from torch_shallow_neural_classifier import TorchShallowNeuralClassifier
from transformers import BertModel, BertTokenizer
```

[2]:
```
class HfBertClassifierModel(nn.Module):
    def __init__(self, n_classes, weights_name='bert-base-cased'):
        super().__init__()
        self.n_classes = n_classes
        self.weights_name = weights_name
        self.bert = BertModel.from_pretrained(self.weights_name)
        self.hidden_dim = self.bert.embeddings.word_embeddings.embedding_dim
        self.W = nn.Linear(self.hidden_dim, self.n_classes)

    def forward(self, X):
        indices = X[: , 0, : ]
        indices = indices.long()
        mask = X[: , 1, : ]
        (final_hidden_states, cls_output) = self.bert(
            indices, attention_mask=mask)
        return self.W(cls_output)
```

Code: BERT fine-tuning with Hugging Face

```
[3]: class HfBertClassifier(TorchShallowNeuralClassifier):
         def __init__(self, weights_name, *args, **kwargs):
             self.weights_name = weights_name
             self.tokenizer = BertTokenizer.from_pretrained(self.weights_name)
             super().__init__(*args, **kwargs)

         def define_graph(self):
             bert = HfBertClassifierModel(
                 self.n_classes_, weights_name=self.weights_name)
             bert.train()
             return bert

         def encode(self, X, max_length=None):
             data = self.tokenizer.batch_encode_plus(
                 X,
                 max_length=max_length,
                 add_special_tokens=True,
                 pad_to_max_length=True,
                 return_attention_mask=True)
             indices = data['input_ids']
             mask = data['attention_mask']
             return [[i, m] for i, m in zip(indices, mask)]
```

```
[4]: mod = HfBertClassifier(
         'bert-base-cased',
         batch_size=16, # Crucial; large batches will eat up all your memory!
         max_iter=4,
         eta=0.00002)
```

References I

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2019. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Deming Chen, Marianne Winslett, Hassan Sajjad, and Preslav Nakov. 2020. Compressing large-scale Transformer-based models: A case study on BERT. ArXiv:2002.11985.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. ArXiv:1909.11942.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. ROBERTa: A robustly optimized BERT pretraining approach. ArXiv:1907.11692.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 14014–14024. Curran Associates, Inc.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. ArXiv:2002.12327.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. ArXiv:1910.01108.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Noah A. Smith. 2019. Contextual word representations: A contextual introduction. ArXiv:1902.06006v2.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

References II

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5753–5763. Curran Associates, Inc.