Contextual word representations: Overview

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

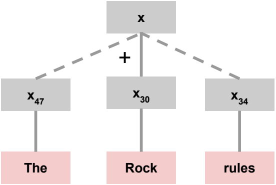
Associated materials

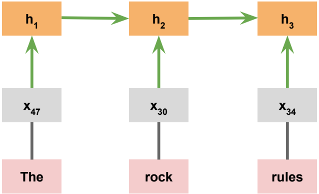
- Notebook: `finetuning.ipynb`
- Smith 2019
- Transformers
  1. Vaswani et al. 2017
  2. Alexander Rush: The Annotated Transformer [link]
- Hugging Face `transformers`: project site
- BERT: Devlin et al. 2019; project site
- RoBERTa: Liu et al. 2019; project site
- ELECTRA: Clark et al. 2019; project site

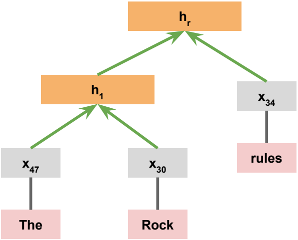
Word representations and context

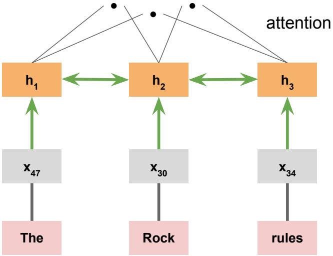
1. a. The vase broke.
   b. Dawn broke.
   c. The news broke.
   d. Sandy broke the world record.
   e. Sandy broke the law.
   f. The burgler broke into the house.
   g. The newscaster broke into the movie broadcast.
   h. We broke even.
2. a. flat tire/beer/note/surface
   b. throw a party/fight/ball/fit
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



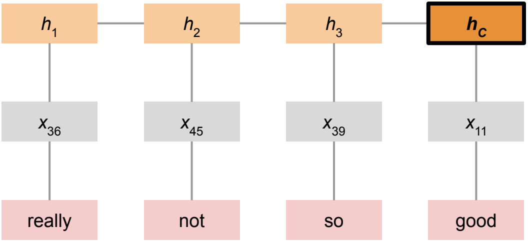






Guiding idea: Attention

classifier	$y = \textbf{softmax}(\tilde{h}W + b)$
attention combo	$\tilde{h} = \tanh([\kappa; h_C]W_\kappa)$
context	$\kappa = \textbf{mean}\left([\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3]\right)$
attention weights	$\alpha = \textbf{softmax}(\tilde{\alpha})$
scores	$\tilde{\alpha} = \begin{bmatrix} h_C^\top h_1 & h_C^\top h_2 & h_C^\top h_3 \end{bmatrix}$

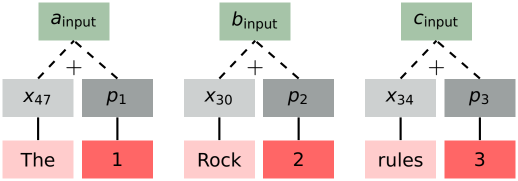


Guiding idea: Word pieces

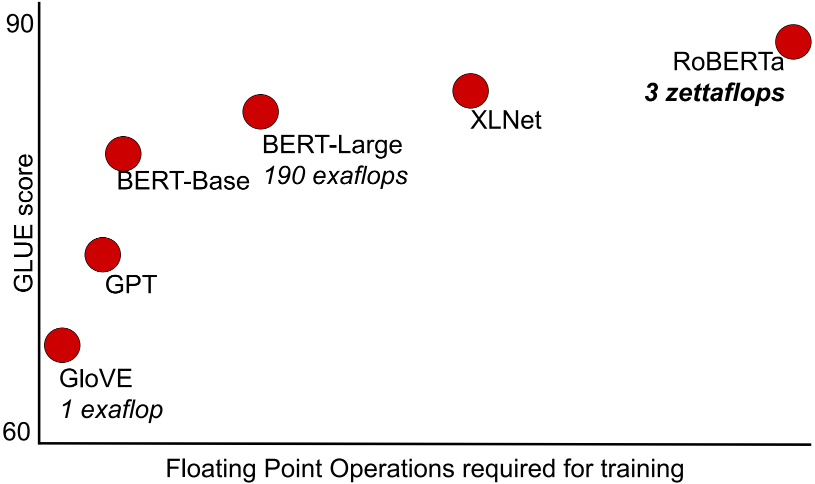
```
[1]: from transformers import BertTokenizer

[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



Current issues and efforts



Clark et al. 2019

Current issues and efforts

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

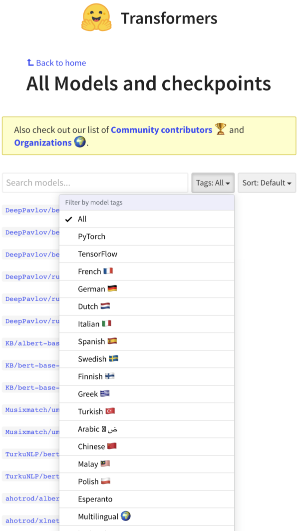
Current issues and efforts

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/

Current issues and efforts

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

Some other Transformer-based models

- SBERT (**S**entence-BERT; Reimers and Gurevych 2019)
- **G**enerative **P**re-trained **T**ransformer
    - GPT (Radford et al. 2018)
    - GPT-2 (Radford et al. 2019)
    - GPT-3 (Brown et al. 2020)
- XLNet (**X**tra **L**ong **T**ransfromer: Yang et al. 2019
- T5 (**T**ext-**T**o-**T**ext **T**ransfer **T**ransformer; Raffel et al. 2019)
- BART: Devlin et al. 2019

References I

T. Brown, B. Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, P. Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, G. Krüger, Tom Henighan, R. Child, Aditya Ramesh, D. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, E. Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, J. Clark, Christopher Berner, Sam McCandlish, A. Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

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.

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.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. Ms, OpenAI.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

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

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.

References II

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.

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.

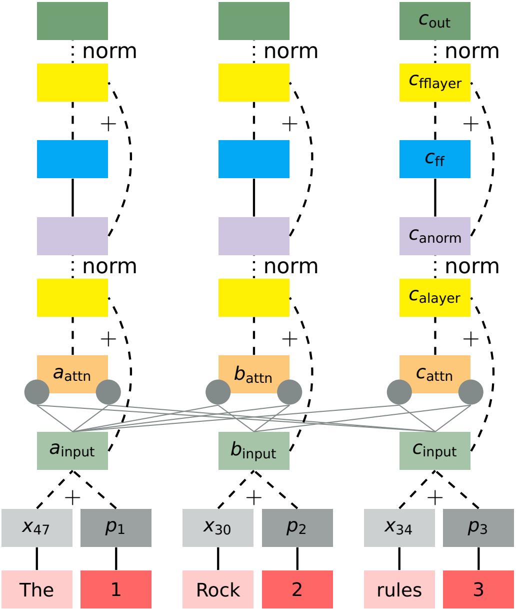
Contextual word representations: Transformers

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

Core model structure



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

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

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

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

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

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

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

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

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

Computing the attention representations

Calculation as previously given

$$
\begin{aligned}
c_{\text{attn}} &= \textbf{sum}\left(\left[\alpha_1 a_{\text{input}}, \alpha_2 b_{\text{input}}\right]\right) \\
\alpha &= \textbf{softmax}(\tilde{\alpha}) \\
\tilde{\alpha} &= \left[\frac{c_{\text{input}}{}^{\top} a_{\text{input}}}{\sqrt{d_k}}, \frac{c_{\text{input}}{}^{\top} b_{\text{input}}}{\sqrt{d_k}}\right]
\end{aligned}
$$

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)
```

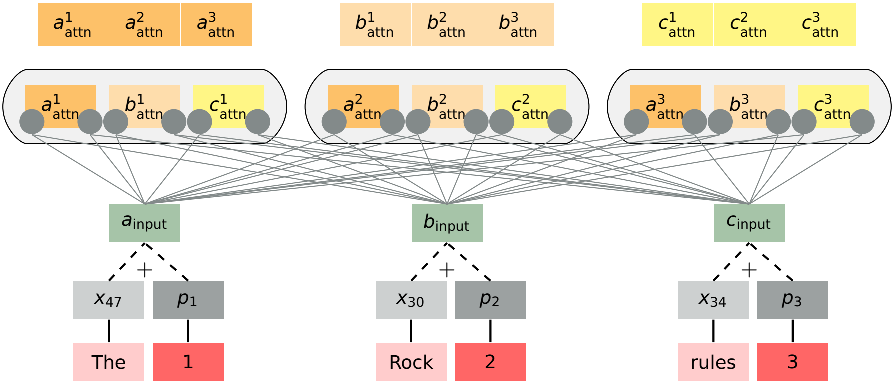
[10]:
```
array([[0.4614388 , 0.53204444, 0.2451212 , 0.45136127],
       [0.50173123, 0.50618272, 0.26184404, 0.43678288],
       [0.45493467, 0.5332328 , 0.23643403, 0.4388242 ]])
```

Multi-headed attention

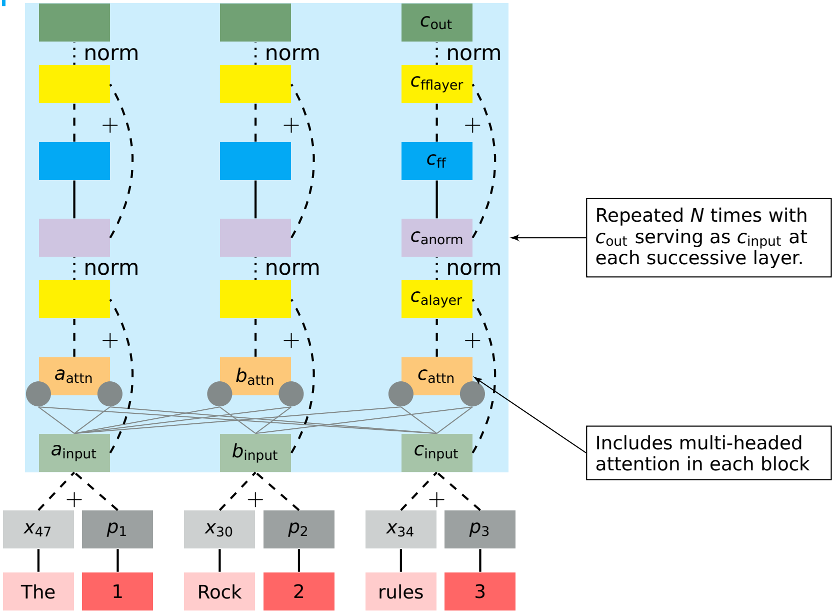
$$c^3_{\text{attn}} = \textbf{sum}\left(\left[\alpha_1(a_{\text{input}} W^V_3), \alpha_2(b_{\text{input}} W^V_3)\right]\right)$$

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

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



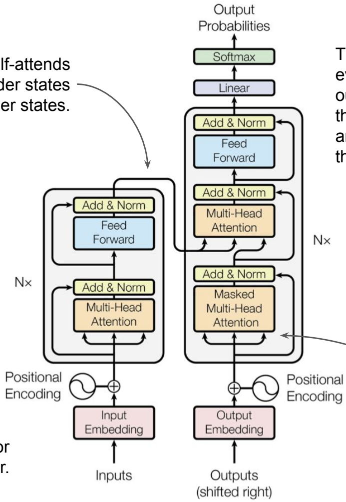
Repeated transformer blocks



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.

References I

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.

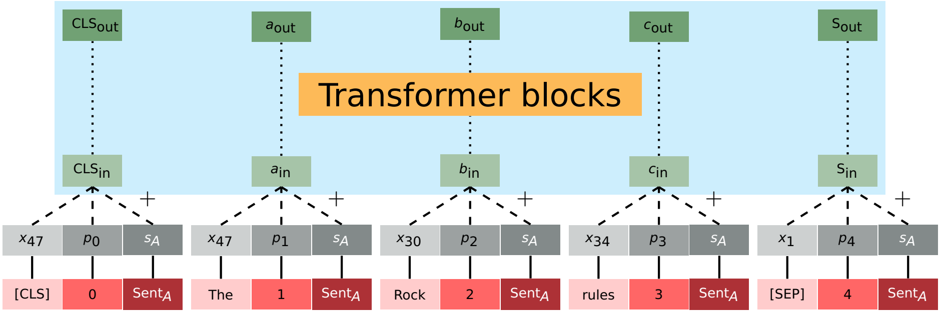
Contextual word representations: BERT

Christopher Potts

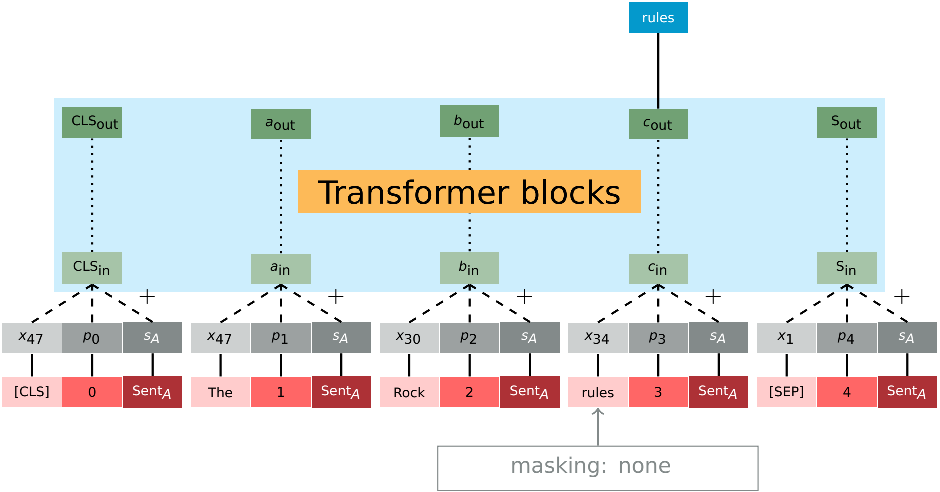
Stanford Linguistics

CS224u: Natural language understanding

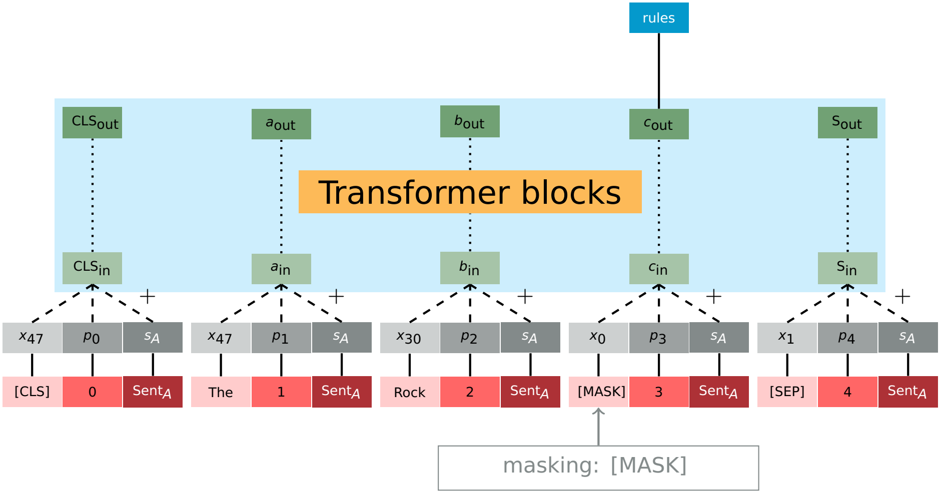
Core model structure



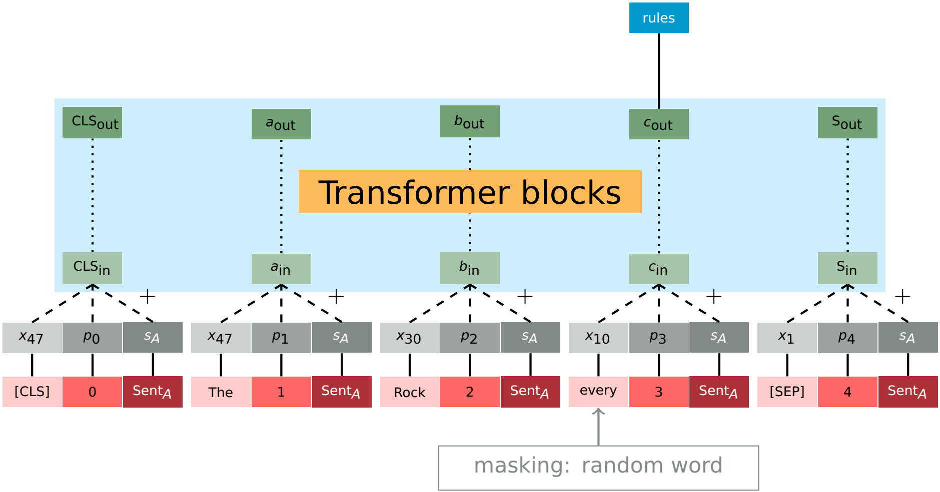
Masked Language Modeling (MLM)



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 next sentence prediction pretraining

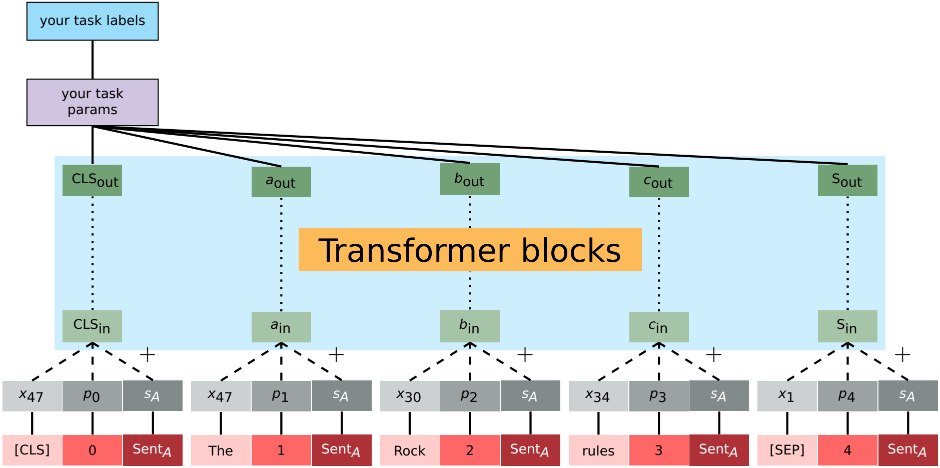
Positive: Actual sentence sequences

- [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.

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"

References I

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.

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.

Contextual word representations: RoBERTa

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

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 **o**ptimized **BERT** **a**pproach

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 $BERT_{BASE}$. 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

References I

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.

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.

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

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.

Contextual word representations: ELECTRA

(**E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately)

Christopher Potts

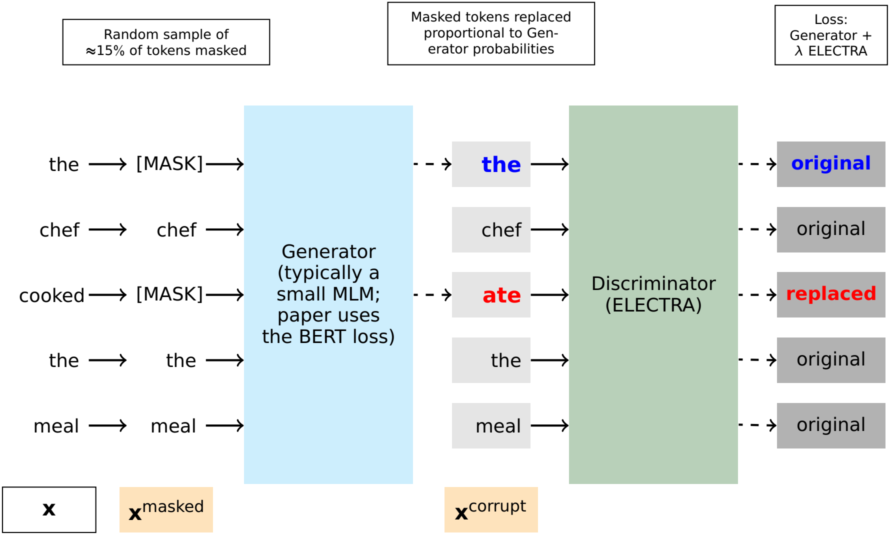
Stanford Linguistics

CS224u: Natural language understanding

Addressing the known limitations with BERT

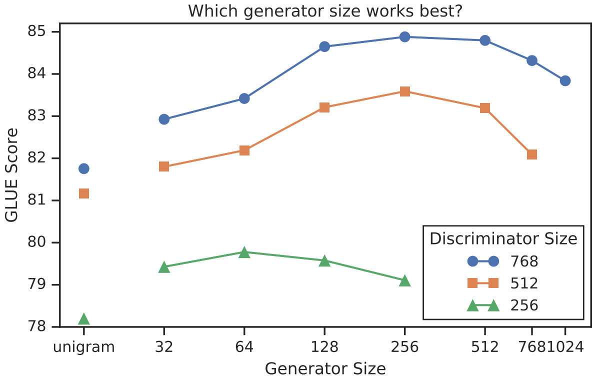
1. Devlin et al. (2019:§5): admirably detailed but still partial ablation studies and optimization studies.
2. Devlin et al. (2019): "The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning."
3. Devlin et al. (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)



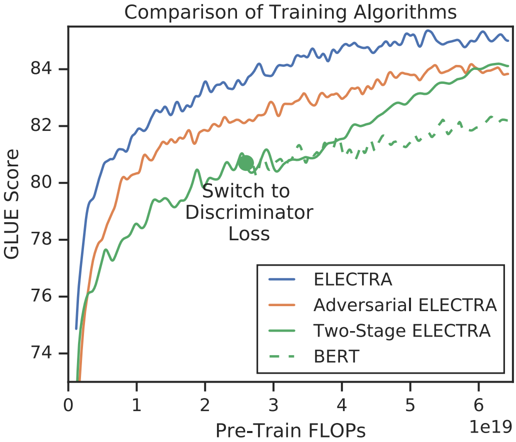
Generator/Discriminator relationships

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:



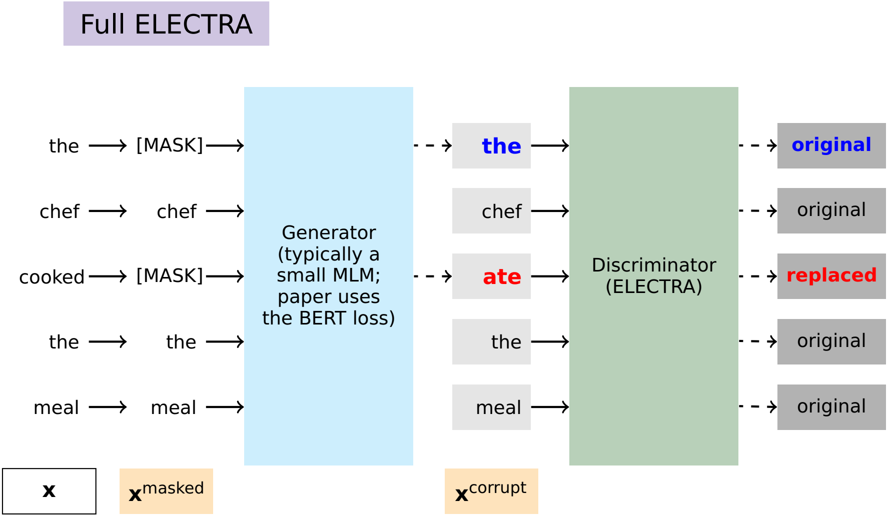
Clark et al. 2019, Figure 3

Efficiency

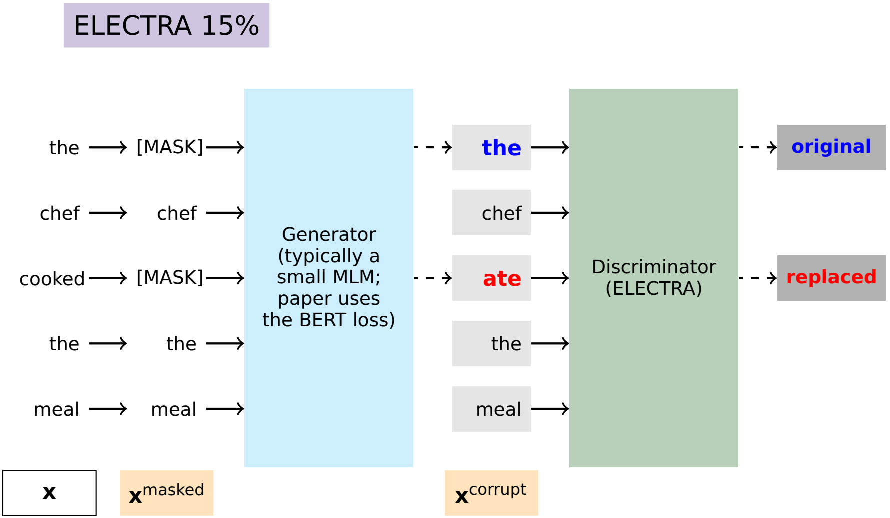


Clark et al. 2019, Figure 3

ELECTRA efficiency analyses

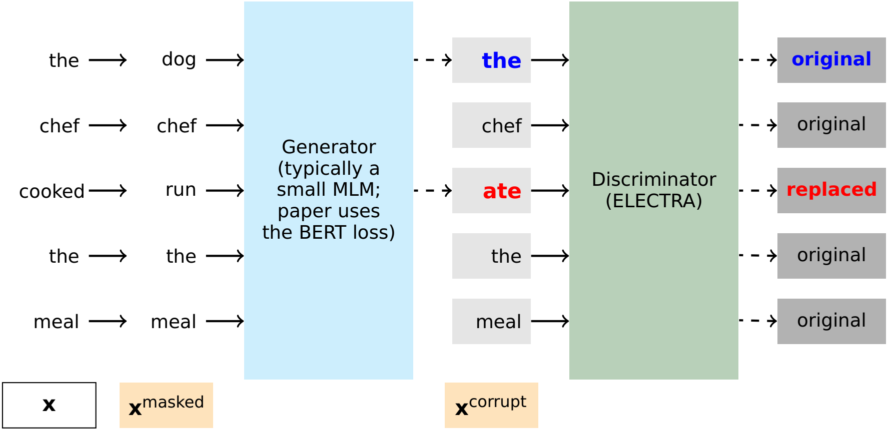


ELECTRA efficiency analyses



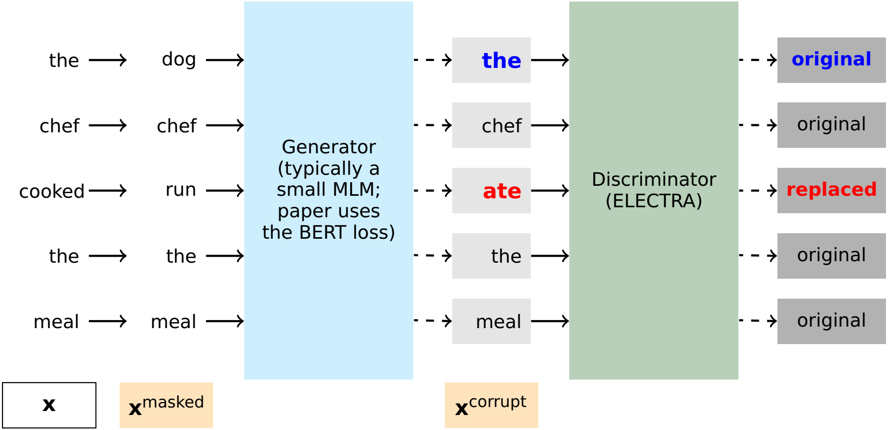
ELECTRA efficiency analyses

Replace MLM



ELECTRA efficiency analyses

All-tokens MLM



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”.

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.

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.

Contextual word representations: Practical fine-tuning

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding

Guiding idea

1. Your existing architecture can benefit from contextual representations.

2. `finetuning.ipynb` shows you how to bring in Transformer 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 team has done!

Standard RNN dataset preparation

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

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

$\Downarrow$

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

$$\left[[-0.41\ -0.08\ 0.27], [0.17\ -0.22\ 0.78][-0.46\ 0.24\ 0.12]\right]$$
$$\left[[-0.02\ -0.56\ 0.11][-0.45\ 0.43\ 0.32]\right]$$

Code snippet: BERT RNN inputs

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

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

[3]:
```
weights_name = 'bert-base-cased'
```

[4]:
```
bert_tokenizer = BertTokenizer.from_pretrained(weights_name)
```

[5]:
```
bert_model = BertModel.from_pretrained(weights_name)
```

[6]:
```
def bert_phi(text):
    input_ids = bert_tokenizer.encode(text, add_special_tokens=True)
    X = torch.tensor([input_ids])
    with torch.no_grad():
        reps = bert_model(X)
        return reps.last_hidden_state[0].squeeze(0).numpy()
```

[7]:
```
def fit_prefeaturized_rnn(X, y):
    mod = TorchRNNClassifier(
        vocab=[],  # No notion of a vocab; the model deals only with vectors.
        early_stopping=True,
        use_embedding=False)  # Feed in vectors directly.
    mod.fit(X, y)
    return mod
```

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

Simple custom models

[7]:
```
class TorchSoftmaxClassifier(TorchShallowNeuralClassifier):

    def build_graph(self):
        return nn.Sequential(
            nn.Linear(self.input_dim, self.n_classes_))
```

tutorial_pytorch_models.ipynb

Simple custom models

[14]:
```
class TorchDeeperNeuralClassifier(TorchShallowNeuralClassifier):
    def __init__(self, hidden_dim1=50, hidden_dim2=50, **base_kwargs):
        super().__init__(**base_kwargs)
        self.hidden_dim1 = hidden_dim1
        self.hidden_dim2 = hidden_dim2
        # Good to remove this to avoid confusion:
        self.params.remove("hidden_dim")
        # Add the new parameters to support model_selection using them:
        self.params += ["hidden_dim1", "hidden_dim2"]

    def build_graph(self):
        return nn.Sequential(
            nn.Linear(self.input_dim, self.hidden_dim1),
            self.hidden_activation,
            nn.Linear(self.hidden_dim1, self.hidden_dim2),
            self.hidden_activation,
            nn.Linear(self.hidden_dim2, self.n_classes_))
```

tutorial_pytorch_models.ipynb

Simple custom models

[24]:
```
class TorchLinearRegressionModel(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.input_dim = input_dim
        self.w = nn.Parameter(torch.zeros(self.input_dim))
        self.b = nn.Parameter(torch.zeros(1))

    def forward(self, X):
        return X.matmul(self.w) + self.b
```

tutorial_pytorch_models.ipynb

Simple custom models

```
[25]: class TorchLinearRegresson(TorchModelBase):
          def __init__(self, **base_kwargs):
              super().__init__(**base_kwargs)
              self.loss = nn.MSELoss(reduction="mean")

          def build_graph(self):
              return TorchLinearRegressionModel(self.input_dim)

          def build_dataset(self, X, y=None):
              """
              This function will be used in training (when there is a `y`)
              and in prediction (no `y`). For both cases, we rely on a
              `TensorDataset`.
              """
              X = torch.FloatTensor(X)
              self.input_dim = X.shape[1]
              if y is None:
                  dataset = torch.utils.data.TensorDataset(X)
              else:
                  y = torch.FloatTensor(y)
                  dataset = torch.utils.data.TensorDataset(X, y)
              return dataset

          def predict(self, X, device=None):
              """
              The `_predict` function of the base class handles all the
              details around data formatting. In this case, the
              raw output of `self.model`, as given by
              `TorchLinearRegressionModel.forward` is all we need.
              """
              return self._predict(X, device=device).cpu().numpy()

          def score(self, X, y):
              """
              Follow sklearn in using `r2_score` as the default scorer.
              """
              preds = self.predict(X)
              return r2_score(y, preds)
```

tutorial_pytorch_models.ipynb

Code: BERT fine-tuning with Hugging Face

[31]:
```
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.bert.train()
        self.hidden_dim = self.bert.embeddings.word_embeddings.embedding_dim
        # The only new parameters -- the classifier:
        self.classifier_layer = nn.Linear(
            self.hidden_dim, self.n_classes)

    def forward(self, indices, mask):
        reps = self.bert(
            indices, attention_mask=mask)
        return self.classifier_layer(reps.pooler_output)
```

Code: BERT fine-tuning with Hugging Face

[32]:

```python
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)
        self.params += ['weights_name']

    def build_graph(self):
        return HfBertClassifierModel(self.n_classes_, self.weights_name)

    def build_dataset(self, X, y=None):
        data = self.tokenizer.batch_encode_plus(
            X,
            max_length=None,
            add_special_tokens=True,
            padding='longest',
            return_attention_mask=True)
        indices = torch.tensor(data['input_ids'])
        mask = torch.tensor(data['attention_mask'])
        if y is None:
            dataset = torch.utils.data.TensorDataset(indices, mask)
        else:
            self.classes_ = sorted(set(y))
            self.n_classes_ = len(self.classes_)
            class2index = dict(zip(self.classes_, range(self.n_classes_)))
            y = [class2index[label] for label in y]
            y = torch.tensor(y)
            dataset = torch.utils.data.TensorDataset(indices, mask, y)
        return dataset
```