Efficient Contextual Representation Learning With Continuous Outputs

Liunian Harold Li
UCLA

Patrick H. Chen
UCLA

Cho-Jui Hsieh
UCLA

Kai-Wei Chang
UCLA

Motivation: Efficient Contextual Representation Learning

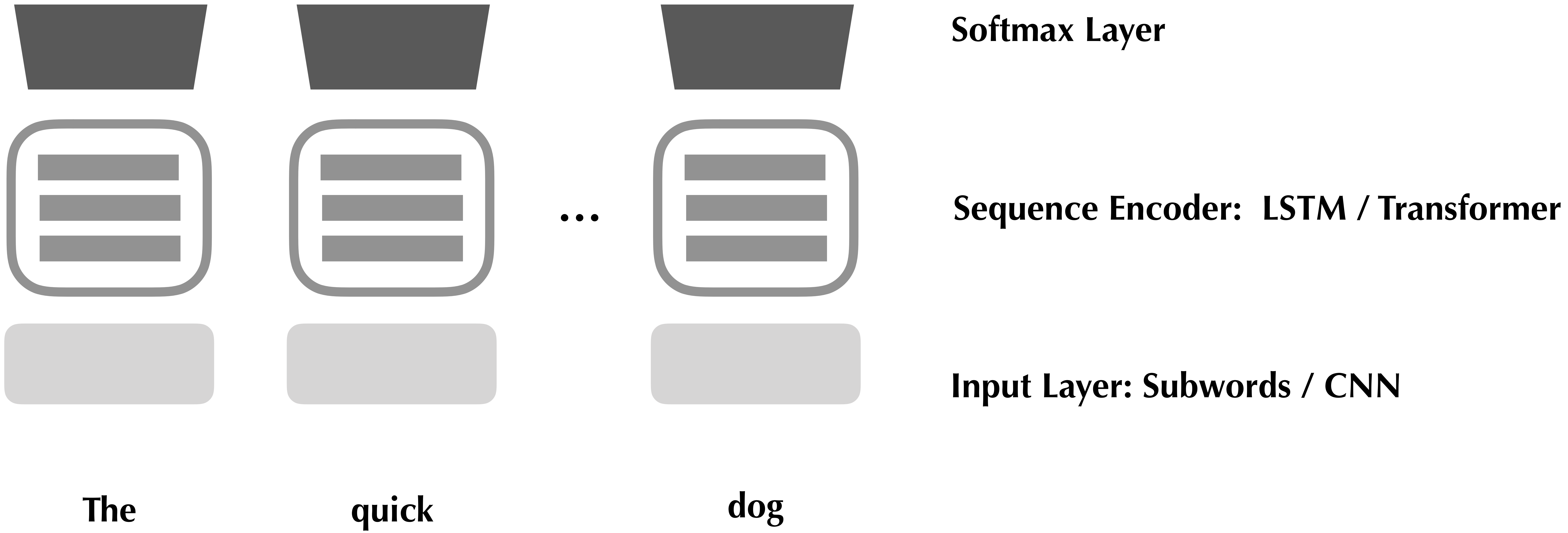
Model	CO_2e	Cloud compute cost
ELMo	262	\$433–\$1472
BERT_{base}	1438	\$3751–\$12,571
GPT-2	—	\$12,902–\$43,008

Consumption	**CO_2e (lbs)**
American life, avg, 1 year	36,156
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468

Energy implication of popular NLP models (Strubell et al., 2019).

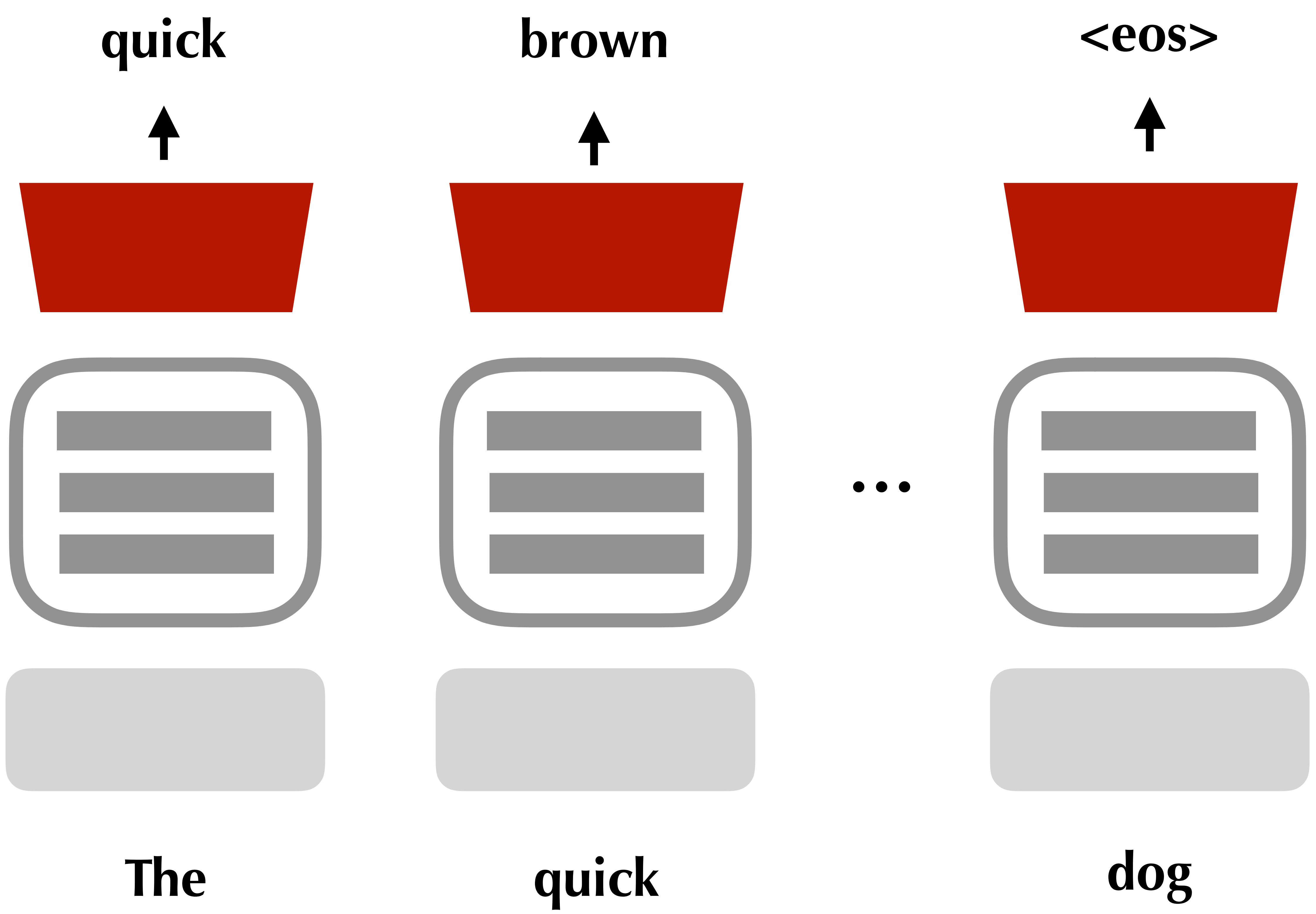
Background: Language Model Pre-training

Language Model Objectives: forward / backward / masked



An illustration of popular pre-trained language models, such as ELMo, GPT, and BERT.

Background: Softmax Layer



Forward language modeling of ELMo

Loss function with a softmax layer:

$$
\begin{aligned}
l(c, w) &= -\log p(w|c) \\
&= -\log softmax(\boldsymbol{cW^T})
\end{aligned}
$$

c: context vector from the sequence encoder

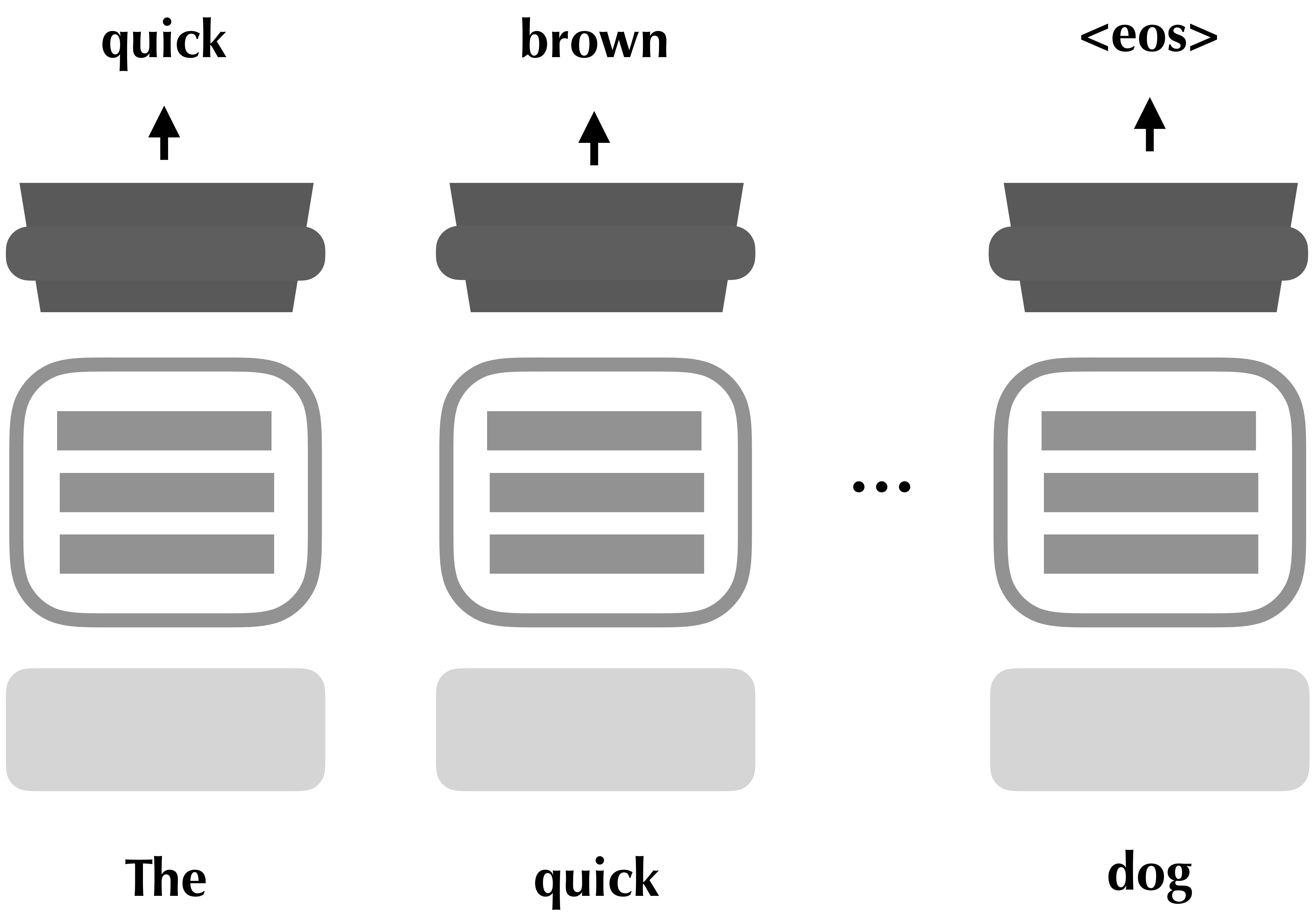
W: V x m matrix, with V being the vocabulary size

V could become extremely large (800K for ELMo)

W takes up 80% of parameters of ELMo

Softmax layer becomes the speed bottleneck!

Approach: Accelerating Language Model Training with Continuous Output



Forward language modeling of ELMo

Loss function with a continuous output layer*:

$$l(c, w) = d(\boldsymbol{c}, \boldsymbol{w}).$$

c: context vector from the sequence encoder

w: pre-trained word embedding of w

d: distance function such as cosine distance

Predicting the word embedding instead of the word!

*Von mises-fisher loss for training sequence to sequence models with continuous outputs. Sachin Kumar and Yulia Tsvetkov. 2018.

Approach: Computational Efficiency

Time complexity:

O(|vocabulary|) -> O(|embedding|)

Negligible

Trainable parameter size:

Hundreds of Millions -> 0

80% parameter reduction for ELMo

Related work

Sampling

Adaptive softmax

Subword

…

Significant efficiency improvement over existing methods

Approach: Computational Efficiency

Time complexity:

O(|vocabulary|) -> O(|embedding|)

Negligible

Trainable parameter size:

Hundreds of Millions -> 0

80% parameter reduction for ELMo

Efficiency improvement of the output layer

Optimizer overhead

GPU memory consumption

Communication cost

Efficiency improvement for the entire model

ELMo training: 14 days x 3 GPUs -> 2.5 days x 4 GPUs

Approach: Open-vocabulary Training

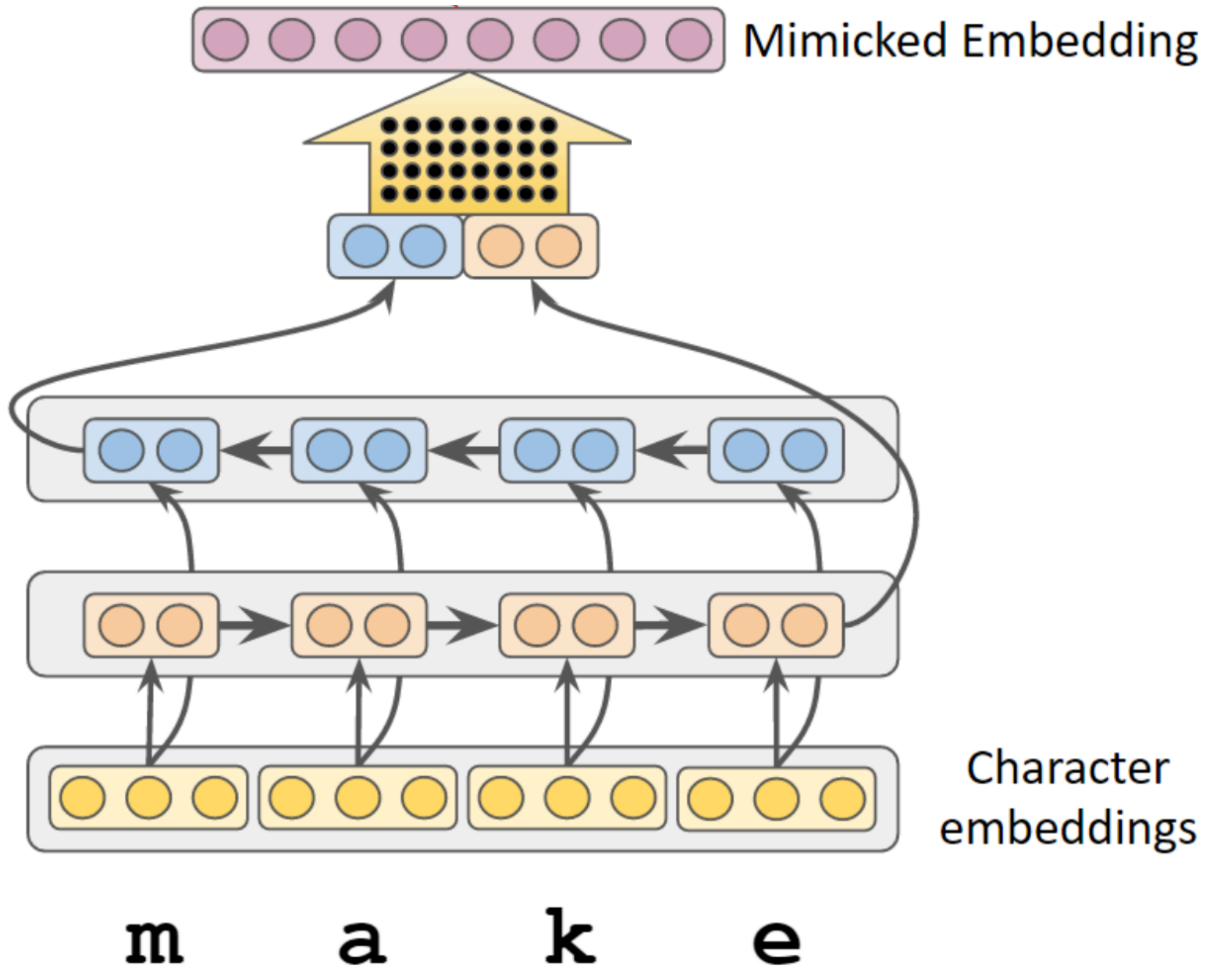
Loss function with a continuous output layer:

$$l(c, w) = d(\boldsymbol{c}, \boldsymbol{w}).$$

w: pre-trained word embedding of w

What if w **is not in the vocabulary?**

Open-vocabulary word embedding such as FastText / MIMICK:



MIMICK (Pinter et al., 2017)

Experiment

Model	Input	Sequence Encoder	Output
ELMo	CNN	LSTM	Sampled Softmax
ELMo-*C* (ours)	FastText$_{CC}$	LSTM w/ LN	Cont w/ FastText$_{CC}$
ELMo-*A*	FastText$_{CC}$	LSTM w/ LN	Adaptive Softmax
ELMo-*Sub*	Subword	LSTM w/ LN	Softmax

All models pre-trained on One Billion Word Benchmark for 10 epochs.

ELMo-C, ELMo-A, and ELMo-Sub trained with the exact same hyper-parameters.

ELMo-A achieves a perplexity of 35.8, lower than 39.7 of the original ELMo.

Experiment

	ELMo	ELMo-*A*	ELMo-*Sub*	ELMo-*C*
Time	14 x 3	5.7 x 4	3.9 x 4	2.5 x 4
Batch	128	256	320	768
Params	499M	196M	92M	76M

Training time (Day x GPU), batch size (per GPU), trainable parameters of four ELMo variants

ELMo-C is 4.2x faster and 6x more memory efficient than ELMo

Experiment

	ELMo	ELMo-*A*	ELMo-*Sub*	ELMo-*C*
Time	14 x 3	5.7 x 4	3.9 x 4	2.5 x 4
Batch	128	256	320	768
Params	499M	196M	92M	76M

Training time (Day x GPU), batch size (per GPU), trainable parameters of four ELMo variants

ELMo-A and ELMo-Sub are more efficient than ELMo

ELMo-C is still 1.6x - 2.3x faster

Experiment

	ELMo	ELMo-*A*	ELMo-*Sub*	ELMo-*C*
SNLI	88.5	88.9	87.1	88.8
Coref	72.9	72.9	72.4	72.9
SST-5	52.96 ± 2.26	53.58 ± 0.77	53.02 ± 2.08	53.80 ± 0.73
NER	92.51 ± 0.28	92.28 ± 0.20	92.17 ± 0.56	92.24 ± 0.10
SRL	83.4	82.7	82.4	82.4

Performance on five downstream tasks following settings of the original ELMo

ELMo-C is comparable with ELMo on four tasks except SRL.

Experiment

	ELMo	ELMo-*A*	ELMo-*Sub*	ELMo-*C*
SNLI	88.5	88.9	87.1	88.8
Coref	72.9	72.9	72.4	72.9
SST-5	52.96 ± 2.26	53.58 ± 0.77	53.02 ± 2.08	53.80 ± 0.73
NER	92.51 ± 0.28	92.28 ± 0.20	92.17 ± 0.56	92.24 ± 0.10
SRL	83.4	82.7	82.4	82.4

Performance on five downstream tasks following settings of the original ELMo

ELMo-C rivals or outperforms ELMo-A and ELMo-Sub.

Analysis: The Continuous Output Layer with Different Sequence Encoders

	LSTM	LSTMx2	Trans Base	ELMo	Trans Large	GPT
Cont	3.97s	10.42s	15.87s	34.58s	48.55s	43.53s
Subword	2.32x	1.49x	1.78x	1.55x	1.72x	1.44x
Adaptive	4.58x	2.20x	2.62x	1.89x	3.28x	2.33x
Sampled	2.50x	1.60x	2.91x	1.91x	OOM	8.31x

Time needed to finish training on one million words using 4 GPUs.

Consistent efficiency improvement over other variants (1.44x - 8.31x), even when the sequence encoder is very large.

Conclusion

Predicting word embedding instead of softmaxing accelerates ELMo training

The resulting model ELMo-C retains comparable performance as ELMo

Computational efficiency sustains when applied to large transformers

https://github.com/uclanlp/ELMO-C