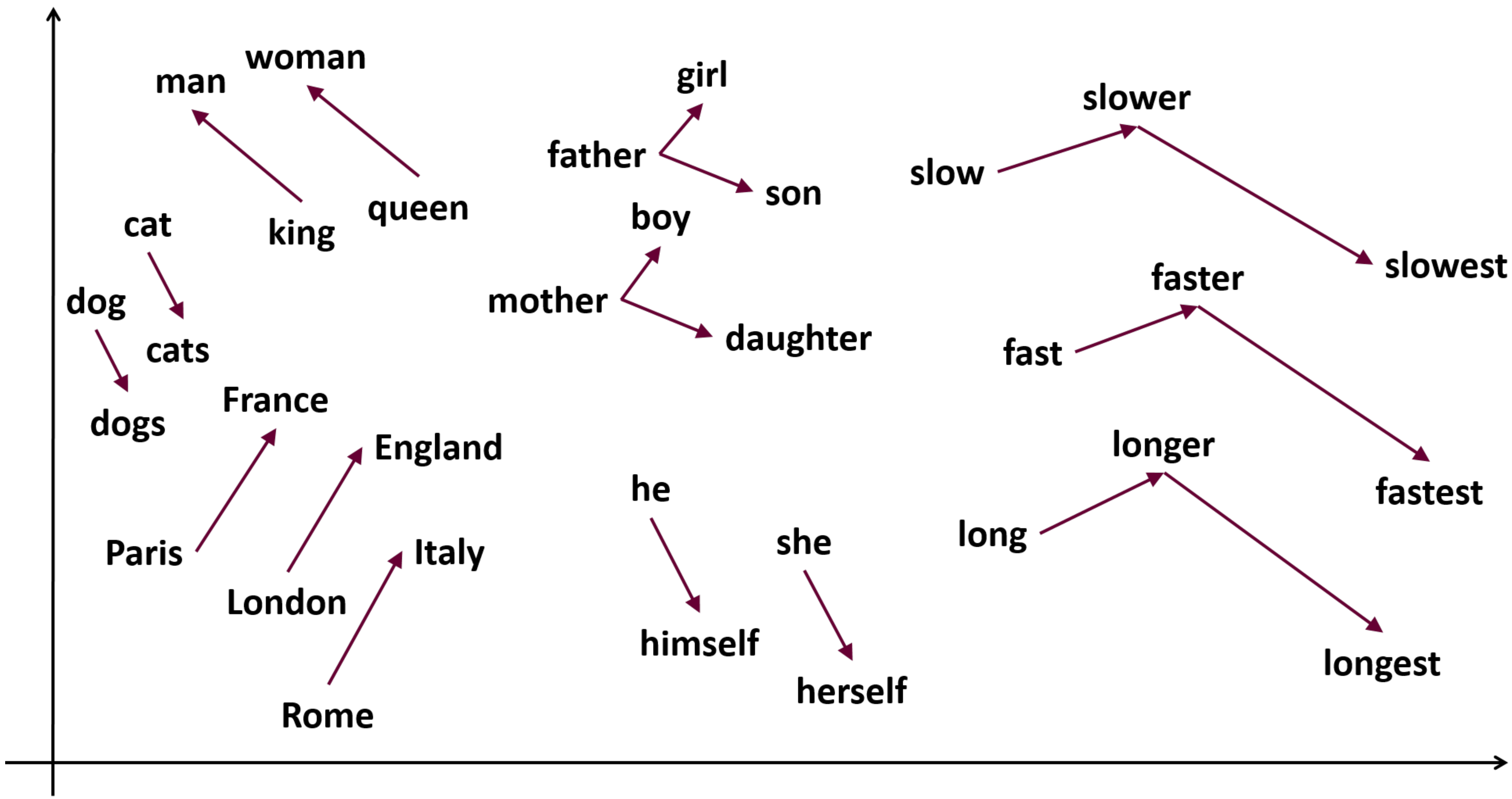
Compressing Word Embeddings via Deep Compositional Code Learning

Raphael Shu, Hideki Nakayama

Iurii Kemaev
16.02.2018

Word embeddings



http://www.samyzaf.com/ML/nlp/nlp.html

Compositional codes

- M codebooks $E_1,\ E_2,\ \ldots,\ E_M$, each containing K vectors
- each word w has code $C_w = (C_w^1, C_w^2, \ldots, C_w^M)$

C_w^i-th codeword in the codebook E_i

$$E(C_w) = \sum_{i=1}^{M} E_i(C_w^i)$$

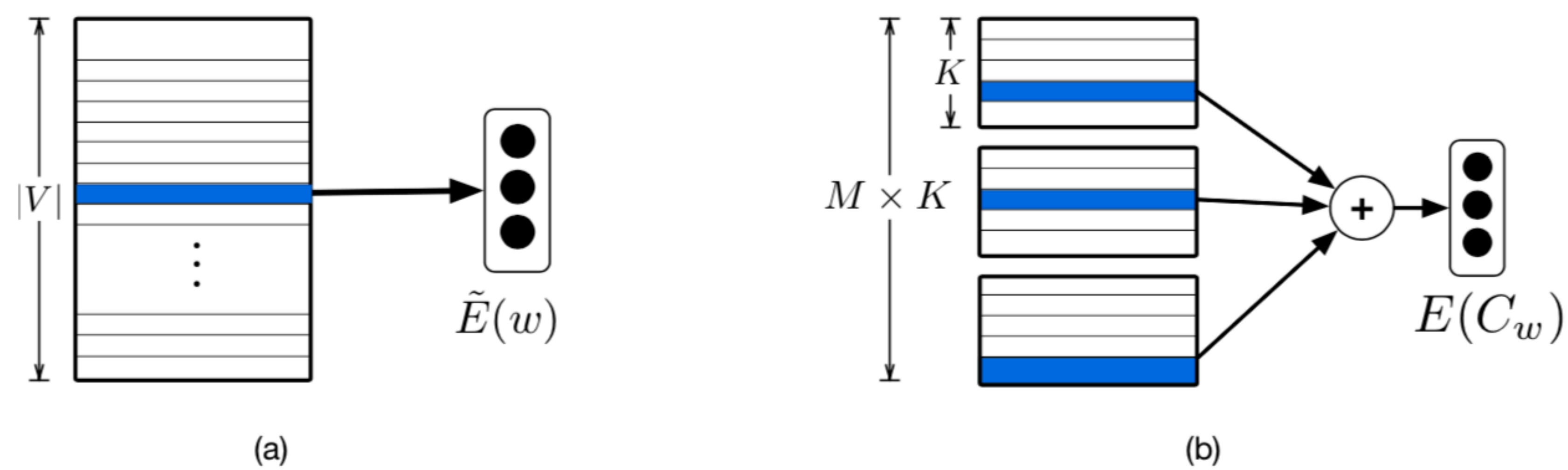


Figure 1: Comparison of embedding computations between the conventional approach (a) and compositional coding approach (b) for constructing embedding vectors

Advantages of compositional codes

	#vectors	computation	code length (bits)
conventional	$\|V\|$	1	-
binary	N	$N/2$	$N/2$
compositional	MK	M	$M \log_2 K$

Table 1: Comparison of different coding approaches. To support N basis vectors, a binary code will have $N/2$ bits and the embedding computation is a summation over $N/2$ vectors. For the compositional approach with M codebooks and K codewords in each codebook, each code has $M \log_2 K$ bits, and the computation is a summation over M vectors.

Trade-off between:

- embeddings matrix size
- model performance
- computational cost

Reconstruction loss

Optimal word codes

Baseline word embeddings

$$(\hat{C}, \hat{E}) = \underset{C,E}{\operatorname{argmin}} \frac{1}{|V|} \sum_{w \in V} \left|\left| E(C_w) - \tilde{E}(w) \right|\right|^2$$

Codebooks vectors

$$= \underset{C,E}{\operatorname{argmin}} \frac{1}{|V|} \sum_{w \in V} \left|\left| \sum_{i=1}^{M} E_i(C_w^i) - \tilde{E}(w) \right|\right|^2$$

Code learning

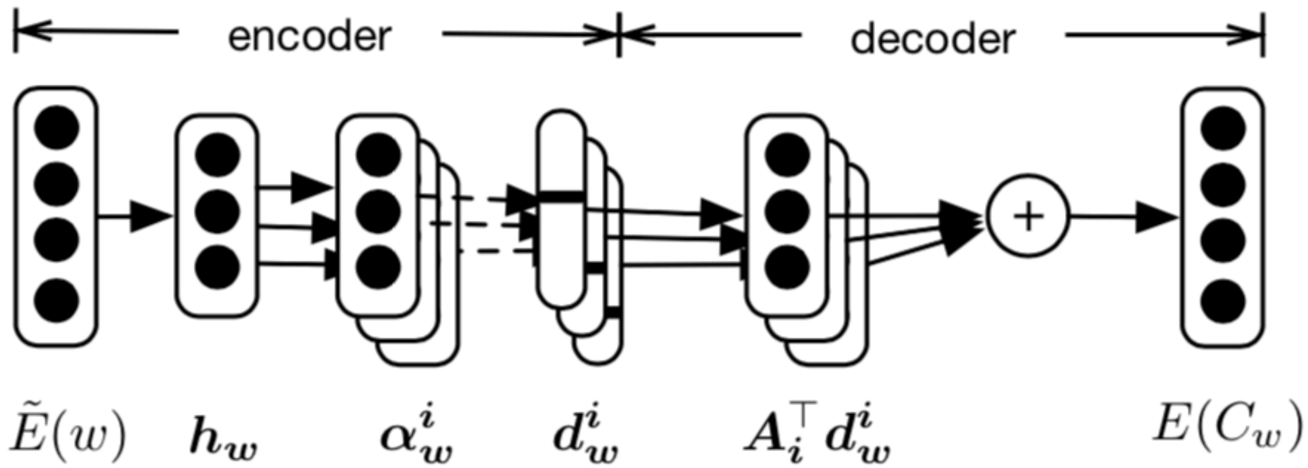


Figure 2: The network architecture for learning compositional compact codes. The Gumbel-softmax computation is marked with dashed lines.

i-th bucket basis vectors

$$E = \sum_{i=1}^{M} D^i A_i$$

Words one-hot codes for **i**-th bucket

Gumbel-softmax computation

$$(d_w^i)_k = \text{softmax}_\tau(\log \alpha_w^i + G)_k \ ,$$

$$G = -\log(-\log(\text{Uniform}[0,1]))$$

Experiments

- Minimize reconstruction loss, Adam optimizer (*lr=1e-4*)
- Sample uniformly batches of 128 elements
- 200k iterations for training
- Every 1k iterations check loss on validation set & update parameters
- 4 GPU, 15 min

Sentiment analysis

	#vectors	vector size	code len	code size	total size	accuracy
Glove baseline	75102	78 MB	-	-	78 MB	87.18
prune 80%	75102	21 MB	-	-	21 MB	86.25
prune 90%	75102	11 MB	-	-	11 MB	84.96
prune 95%	75102	5.31 MB	-	-	5.31 MB	83.88
8×8 coding	64	0.06 MB	24 bits	0.21 MB	0.27 MB	82.84
8×16 coding	128	0.13 MB	32 bits	0.28 MB	0.41 MB	83.77
16×8 coding	128	0.13 MB	48 bits	0.42 MB	0.55 MB	85.21
8×64 coding	512	0.52 MB	48 bits	0.42 MB	0.94 MB	86.66
16×32 coding	512	0.52 MB	80 bits	0.71 MB	**1.23 MB**	87.37
32×16 coding	512	0.52 MB	128 bits	1.14 MB	1.66 MB	87.80
64×8 coding	512	0.52 MB	192 bits	1.71 MB	2.23 MB	**88.15**

Table 3: Trade-off between the model performance and the size of embedding layer on IMDB sentiment analysis task

Machine translation

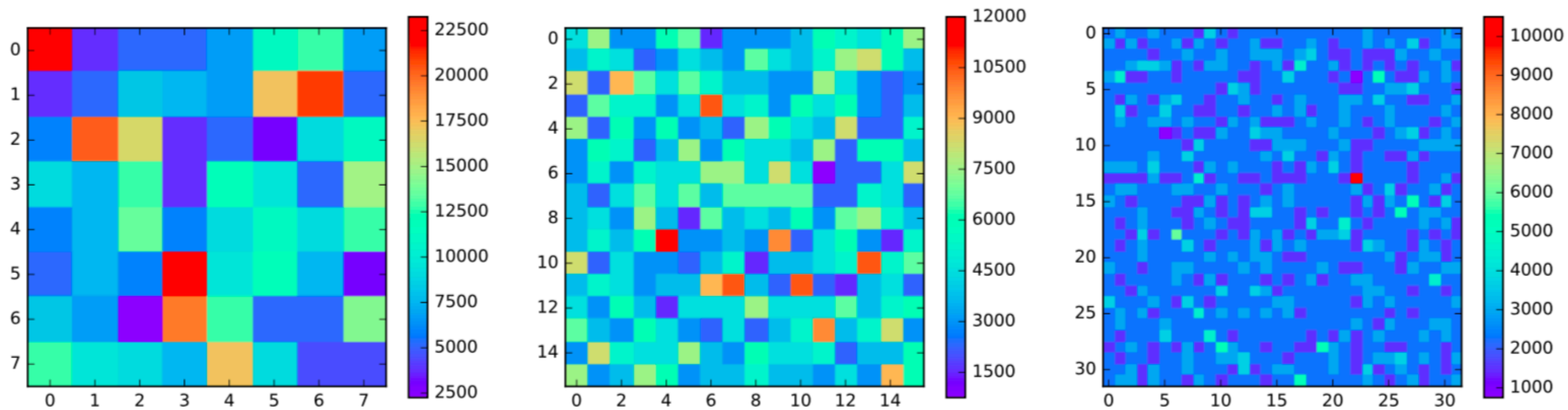
	coding	#vectors	vector size	code len	code size	total size	BLEU(%)
De $\rightarrow$ En	baseline	40000	35 MB	-	-	35 MB	29.45
	prune 90%	40000	5.21 MB	-	-	5.21 MB	29.34
	prune 95%	40000	2.63 MB	-	-	2.63 MB	28.84
	32×16	512	0.44 MB	128 bits	0.61 MB	1.05 MB	29.04
	64×16	1024	0.89 MB	256 bits	1.22 MB	**2.11 MB**	**29.56**
En $\rightarrow$ Ja	baseline	80000	274 MB	-	-	274 MB	37.93
	prune 90%	80000	41 MB	-	-	41 MB	38.56
	prune 98%	80000	8.26 MB	-	-	8.26 MB	37.09
	32×16	512	1.75 MB	128 bits	1.22 MB	**2.97 MB**	38.10
	64×16	1024	3.50 MB	256 bits	2.44 MB	5.94 MB	**38.89**

Table 4: Trade-off between the model performance and the size of embedding layer in machine translation tasks

Qualitative analysis

category	word	8×8 **code**	16×16 **code**
animal	dog	0 7 0 1 7 3 7 0	7 7 0 8 3 5 8 5 B 2 E E 0 B 0 A
	cat	7 7 0 1 7 3 7 0	7 7 2 8 B 5 8 C B 2 E E 4 B 0 A
	penguin	0 7 0 1 7 3 6 0	7 7 E 8 7 6 4 C F D E 3 D 8 0 A
verb	go	7 7 0 6 4 3 3 0	2 C C 8 2 C 1 1 B D 0 E 0 B 5 8
	went	4 0 7 6 4 3 2 0	B C C 6 B C 7 5 B 8 6 E 0 D 0 4
	gone	7 7 0 6 4 3 3 0	2 C C 8 0 B 1 5 B D 6 E 0 2 5 A

Table 5: Examples of learned compositional codes based on Glove embedding vectors



(a) 8 x 8 (b) 16 x 16 (c) 32 x 32

Figure 3: Visualization of code balance for different coding scheme. Each cell in the heat map shows the count of words containing a specific subcode. The results show that any codeword is assigned to more than 1000 words without wasting.

Summary

- Hight loss-free compression (reaches 94% ~ 99%)
- With the learned codes and basis vectors, the computation graph for composing embeddings is fairly easy to implement, and does not require modifications to other parts in the neural network
- Direct learning approach for the codes in an end-to-end neural network, with a Gumbel-softmax layer to encourage the discreteness

"Reviewers found the paper to be simple, clear, and effective." (C)

Sentiment analysis

- IMDB movie dataset: 25K reviews in train set & 25K in test
- public GloVe 300-d vectors as baseline
- single LSTM layer with 150 hidden units + softmax

Machine translation

- IWSLT 2014 German-to-English translation task (178K sentence pairs for training + 7K for test)
- ASPEC English-to-Japanese translation task (150M + 150M bilingual pairs)
- 2 NMT models (bidirectional encoder + 2-LSTM-layer decoder (256 & 1000 units) + Key-Value attention)
- Evaluate smoothed BLEU every 7K iterations on 50 batches
- The learning rate is reduced by a factor of 10 if no improvement is observed in 3 validation runs. The training ends after the learning rate is reduced three times
- Train baseline NMT to obtain task-specific embeddings, use them to get compositional codes and basis vectors, then fix reconstructed embeddings, plug them in NMT and retrain model

Thank you for attention