Word2Vec, Seq2seq, NMT

Milan Straka

April 27, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Unsupervised Word Embeddings

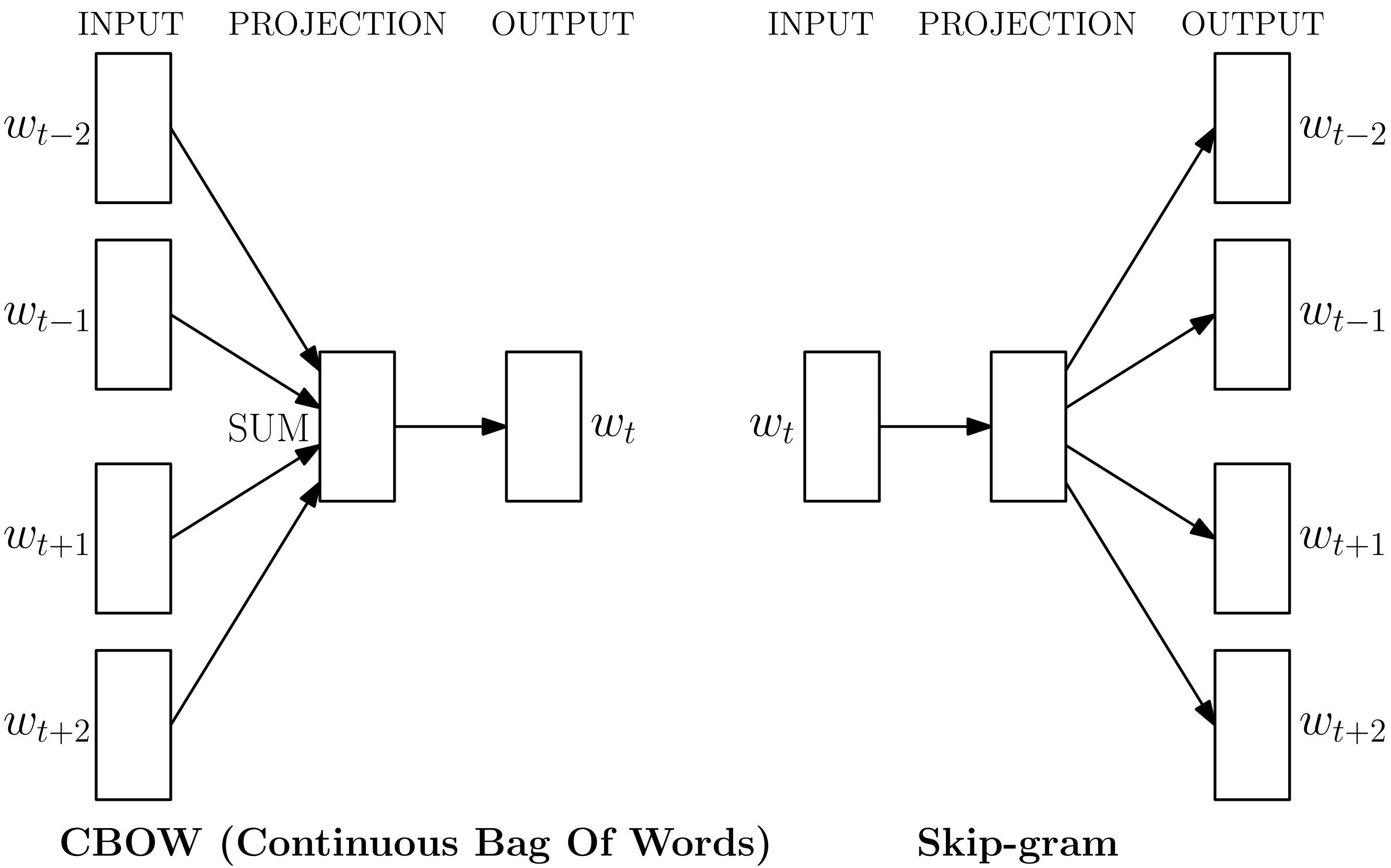
The embeddings can be trained for each task separately.

However, a method of precomputing word embeddings have been proposed, based on *distributional hypothesis*:

Words that are used in the same contexts tend to have similar meanings.

The distributional hypothesis is usually attributed to Firth (1957):

> *You shall know a word by a company it keeps.*

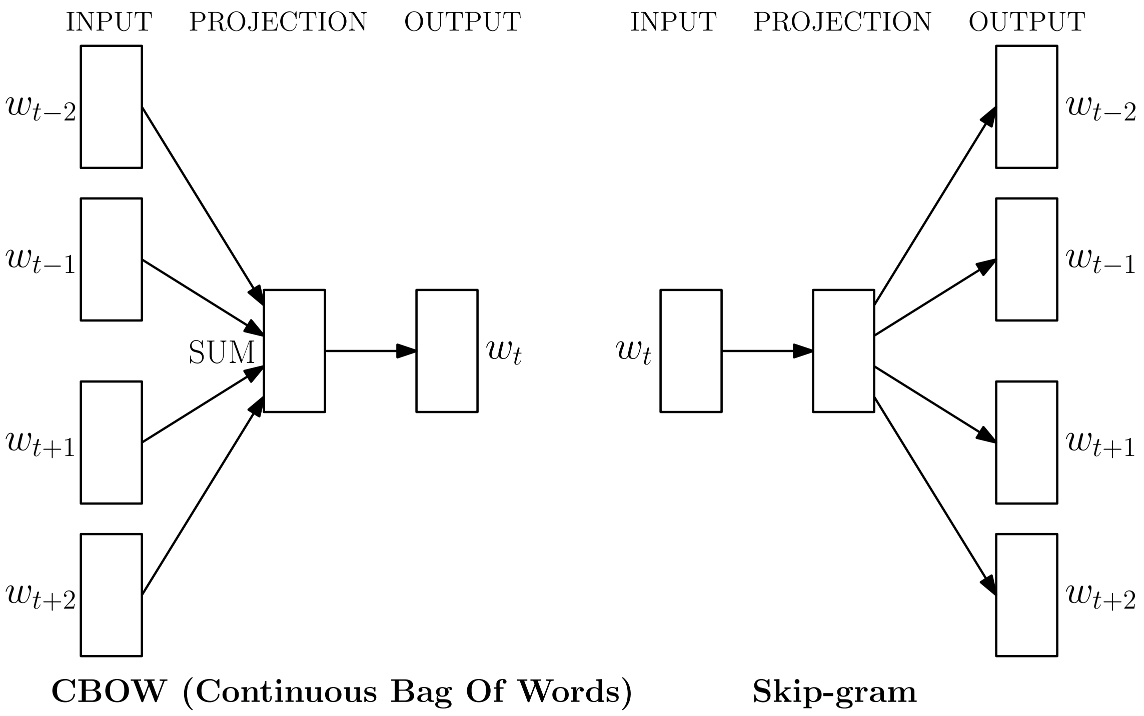


Mikolov et al. (2013) proposed two very simple architectures for precomputing word embeddings, together with a C multi-threaded implementation `word2vec`.

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Table 8 of paper "Efficient Estimation of Word Representations in Vector Space", https://arxiv.org/abs/1301.3781.



Considering input word w_i and output w_o, the Skip-gram model defines

$$p(w_o|w_i) \stackrel{\text{def}}{=} \frac{e^{\boldsymbol{W}_{w_o}^\top \boldsymbol{V}_{w_i}}}{\sum_w e^{\boldsymbol{W}_w^\top \boldsymbol{V}_{w_i}}}.$$

Instead of a large softmax, we construct a binary tree over the words, with a sigmoid classifier for each node.

If word w corresponds to a path $n_1, n_2, \ldots, n_L$, we define

$$p_{\mathrm{HS}}(w|w_i) \stackrel{\text{def}}{=} \prod_{j=1}^{L-1} \sigma([+1 \text{ if } n_{j+1} \text{ is right child else -1}] \cdot \boldsymbol{W}_{n_j}^\top \boldsymbol{V}_{w_i}).$$

Word2Vec – Negative Sampling

Instead of a large softmax, we could train individual sigmoids for all words.

We could also only sample the *negative examples* instead of training all of them.

This gives rise to the following *negative sampling* objective:

$$l_\mathrm{NEG}(w_o, w_i) \stackrel{\text{def}}{=} \log \sigma(\boldsymbol{W}_{w_o}^\top \boldsymbol{V}_{w_i}) + \sum_{j=1}^k \mathbb{E}_{w_j \sim P(w)} \log\big(1 - \sigma(\boldsymbol{W}_{w_j}^\top \boldsymbol{V}_{w_i})\big).$$

For $P(w)$, both uniform and unigram distribution $U(w)$ work, but

$$U(w)^{3/4}$$

outperforms them significantly (this fact has been reported in several papers by different authors).

increased	*John*	*Noahshire*	*phding*
reduced	Richard	Nottinghamshire	mixing
improved	George	Bucharest	modelling
expected	James	Saxony	styling
decreased	Robert	Johannesburg	blaming
targeted	Edward	Gloucestershire	christening

Table 2: Most-similar in-vocabular words under the C2W model; the two query words on the left are in the training vocabulary, those on the right are nonce (invented) words.

Figure 1 of paper "Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation", https://arxiv.org/abs/1508.02096.

Convolutional Character-level WEs

	In Vocabulary					Out-of-Vocabulary		
	while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*	–	–	–
	letting	*her*	*we*	*robert*	*advertising*	–	–	–
	though	*my*	*guys*	*neil*	*turnover*	–	–	–
	minute	*their*	*i*	*nancy*	*turnover*	–	–	–
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*	*computer-guided*	*informed*	*look*
	whole	*hhs*	*young*	*rich*	*training*	*computerized*	*performed*	*cook*
	meanwhile	*is*	*four*	*richer*	*reading*	*disk-drive*	*transformed*	*looks*
	white	*has*	*youth*	*richter*	*leading*	*computer*	*inform*	*shook*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*	*computer-guided*	*informed*	*look*
	whole	*this*	*your*	*gerard*	*training*	*computer-driven*	*performed*	*looks*
	though	*their*	*doug*	*edward*	*traded*	*computerized*	*outperformed*	*looked*
	nevertheless	*your*	*i*	*carl*	*trader*	*computer*	*transformed*	*looking*

Table 6: Nearest neighbor words (based on cosine similarity) of word representations from the large word-level and character-level (before and after highway layers) models trained on the PTB. Last three words are OOV words, and therefore they do not have representations in the word-level model.

Table 6 of paper "Character-Aware Neural Language Models", https://arxiv.org/abs/1508.06615.

Character N-grams

Another simple idea appeared simultaneously in three nearly simultaneous publications as Charagram, Subword Information or SubGram.

A word embedding is a sum of the word embedding plus embeddings of its character *n*-grams. Such embedding can be pretrained using same algorithms as `word2vec`.

The implementation can be

- dictionary based: only some number of frequent character *n*-grams is kept;
- hash-based: character *n*-grams are hashed into K buckets (usually $K \sim 10^6$ is used).

Charagram WEs

query	tiling	tech-rich	english-born	micromanaging	eateries	dendritic
`sisg`	tile	tech-dominated	british-born	micromanage	restaurants	dendrite
	flooring	tech-heavy	polish-born	micromanaged	eaterie	dendrites
`sg`	bookcases	technology-heavy	most-capped	defang	restaurants	epithelial
	built-ins	.ixic	ex-scotland	internalise	delis	p53

Table 7: Nearest neighbors of rare words using our representations and `skipgram`. These hand picked examples are for illustration.

Table 7 of paper "Enriching Word Vectors with Subword Information", https://arxiv.org/abs/1607.04606.

Charagram WEs

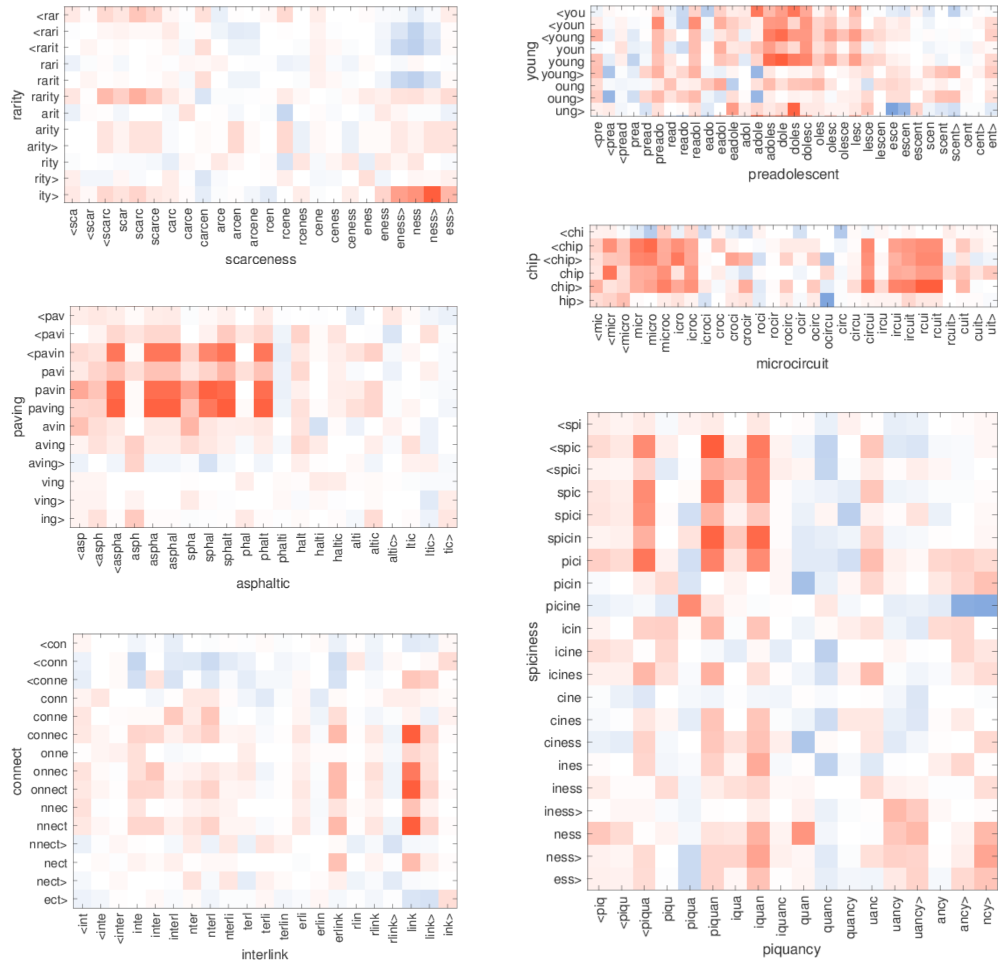


Figure 2: Illustration of the similarity between character n-grams in out-of-vocabulary words. For each pair, only one word is OOV, and is shown on the x axis. Red indicates positive cosine, while blue negative.

Figure 2 of paper "Enriching Word Vectors with Subword Information", https://arxiv.org/abs/1607.04606.

Sequence-to-Sequence Architecture

Sequence-to-Sequence Architecture

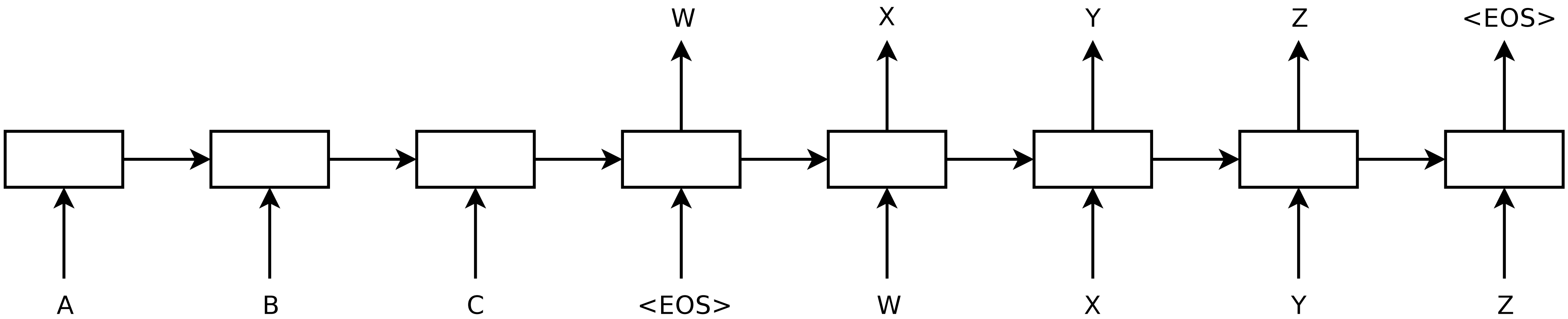


Figure 1 of paper "Sequence to Sequence Learning with Neural Networks", https://arxiv.org/abs/1409.0473.

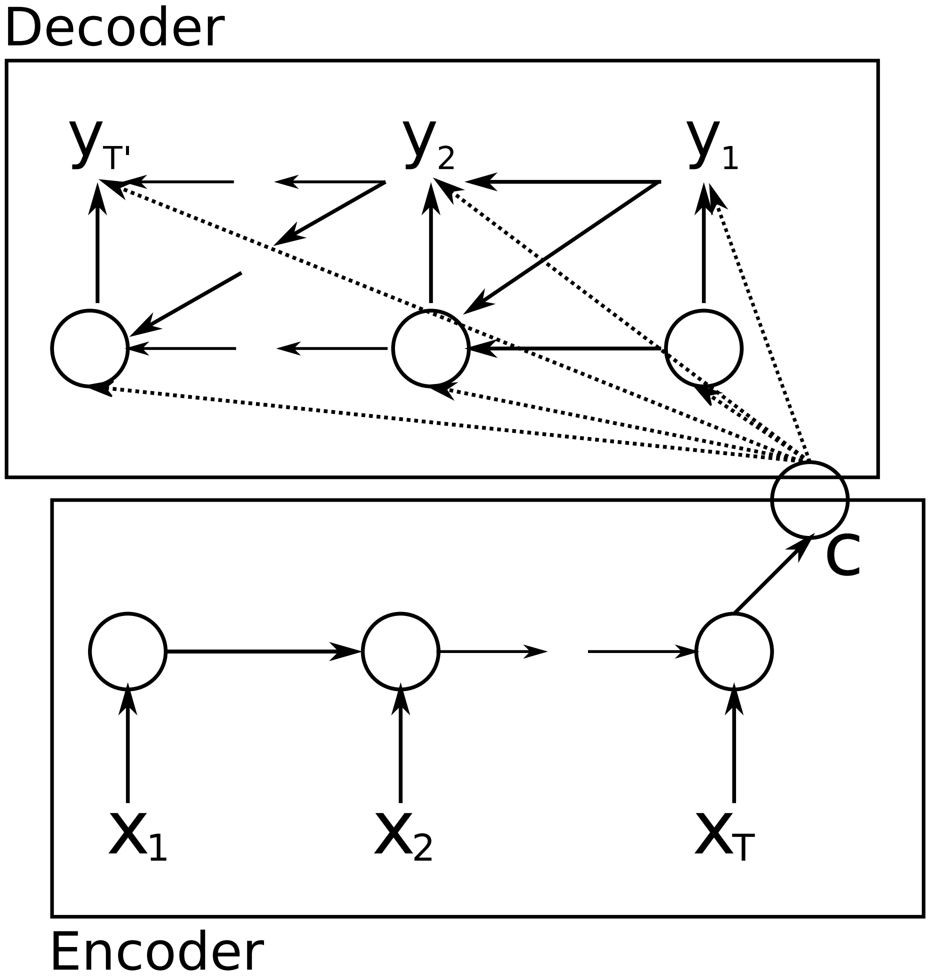
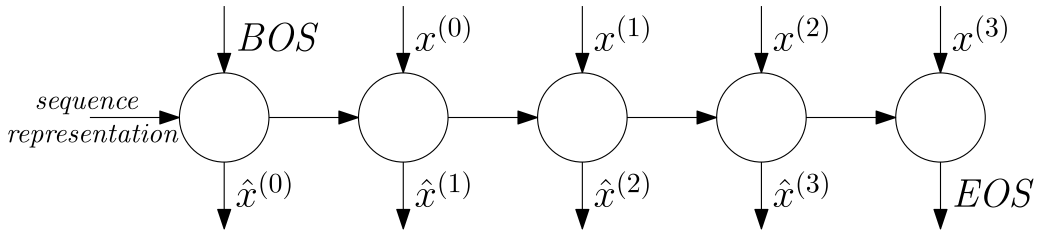


Figure 1 of paper "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", https://arxiv.org/abs/1406.1078.

Training

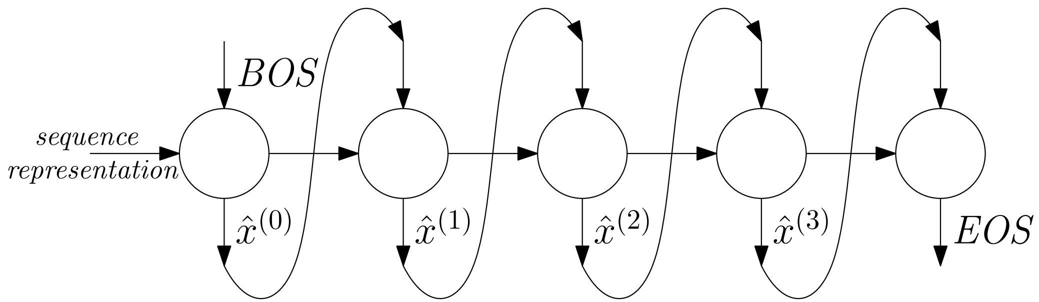
The so-called *teacher forcing* is used during training – the gold outputs are used as inputs during training.



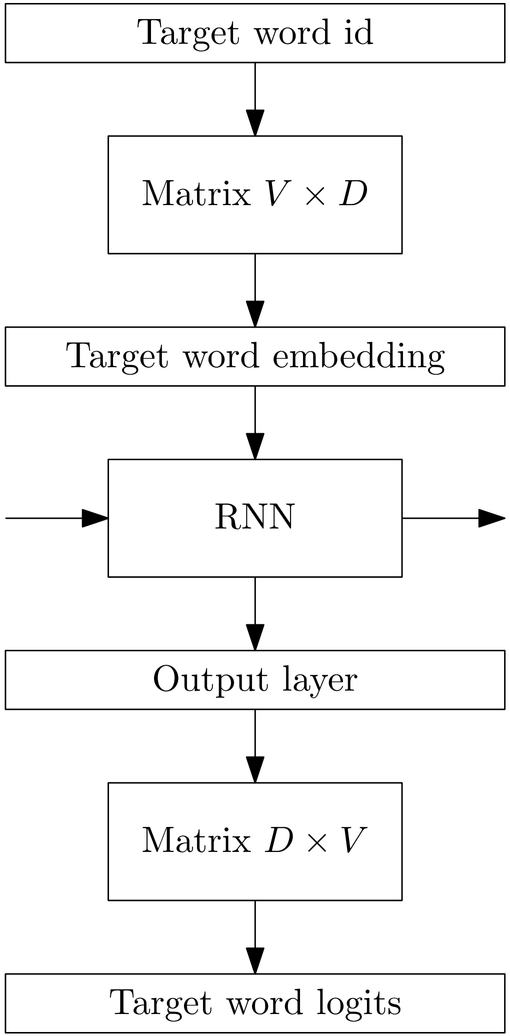
Inference

During inference, the network processes its own predictions.

Usually, the generated logits are processed by an $\arg\max$, the chosen word embedded and used as next input.



Tying Word Embeddings



As another input during decoding, we add *context vector* c_i:

$$\boldsymbol{s}_i = f(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i).$$

We compute the context vector as a weighted combination of source sentence encoded outputs:

$$\boldsymbol{c}_i = \sum_j \alpha_{ij} \boldsymbol{h}_j$$

The weights α_{ij} are softmax of e_{ij} over j,

$$\boldsymbol{\alpha}_i = \operatorname{softmax}(\boldsymbol{e}_i),$$

with e_{ij} being

$$e_{ij} = \boldsymbol{v}^\top \tanh(\boldsymbol{V}\boldsymbol{h}_j + \boldsymbol{W}\boldsymbol{s}_{i-1} + \boldsymbol{b}).$$

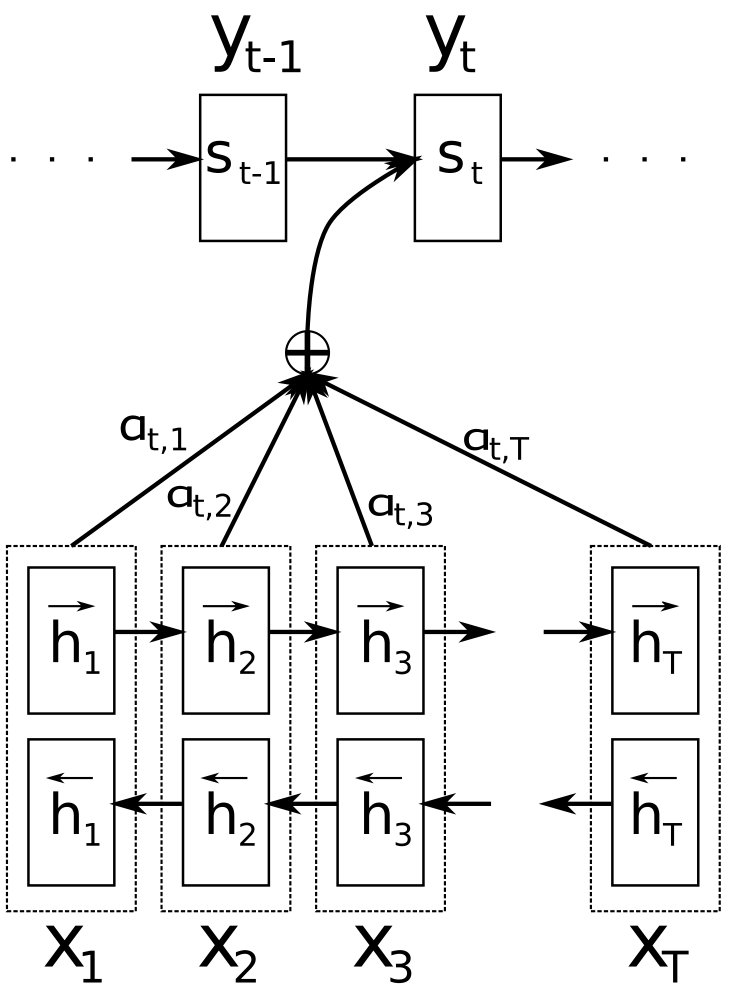


Figure 1 of paper "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473.

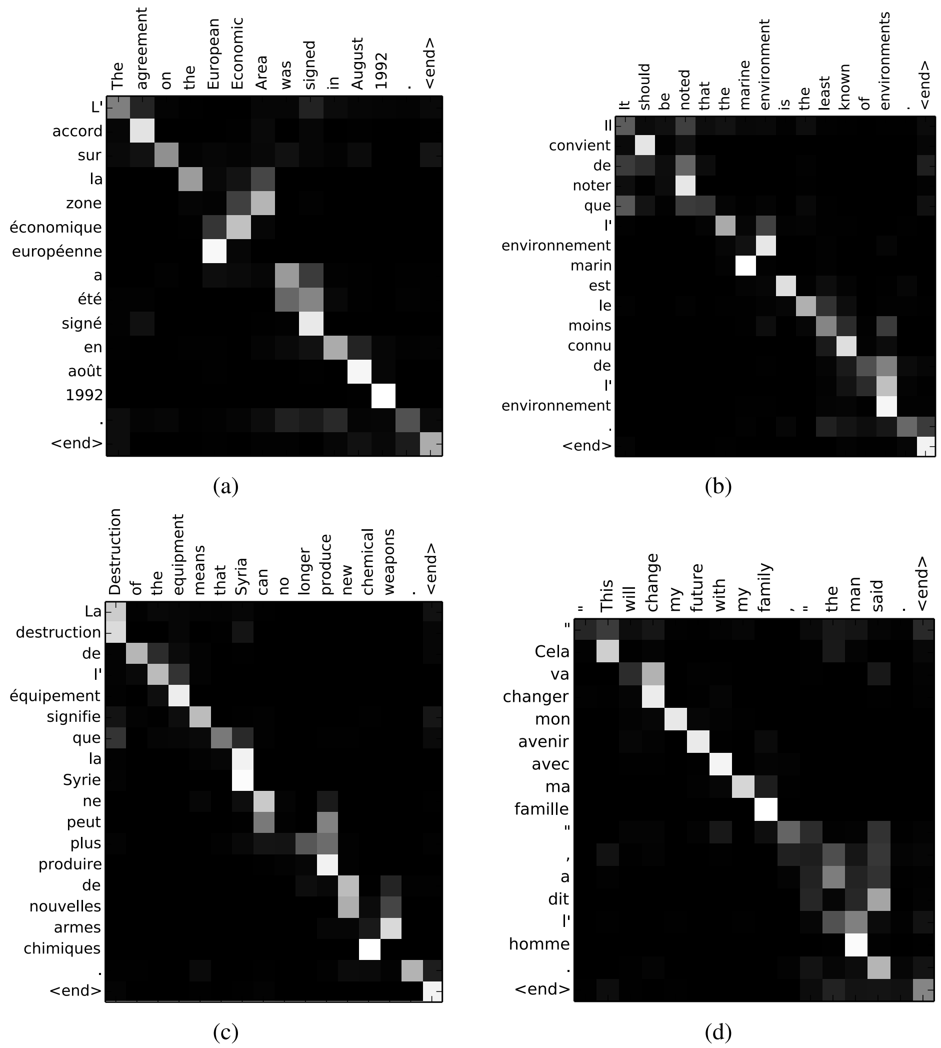


Figure 3 of paper "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473.

Subword Units

Translate *subword units* instead of words. The subword units can be generated in several ways, the most commonly used are:

- **BPE**: Using the *byte pair encoding* algorithm. Start with individual characters plus a special end-of-word symbol $\cdot$. Then, merge the most occurring symbol pair A, B by a new symbol AB, with the symbol pair never crossing word boundary (so that the end-of-word symbol cannot be inside a subword).

  Considering a dictionary with words *low, lowest, newer, wider*, a possible sequence of merges:

$$r \ \ \cdot \rightarrow r\cdot$$
$$l \ \ o \rightarrow lo$$
$$lo \ \ w \rightarrow low$$
$$e \ \ r\cdot \rightarrow er\cdot$$

- **Wordpieces**: Given a text divided into subwords, we can compute unigram probability of every subword, and then get the likelihood of the text under a unigram language model by multiplying the probabilities of the subwords in the text.

  When we have only a text and a subword dictionary, we divide the text in a greedy fashion, iteratively choosing the longest existing subword.

  When constructing the subwords, we again start with individual characters, and then repeatedly join such a pair of subwords, which increases the unigram language model likelihood the most.

Both approaches give very similar results; a biggest difference is that during the inference:

- for BPE, the sequence of merges must be performed in the same order as during the construction of the BPE;
- for Wordpieces, it is enough to find longest matches from the subword dictionary.

Usually quite little subword units are used (32k-64k), often generated on the union of the two vocabularies (the so-called *joint BPE* or *shared wordpieces*).

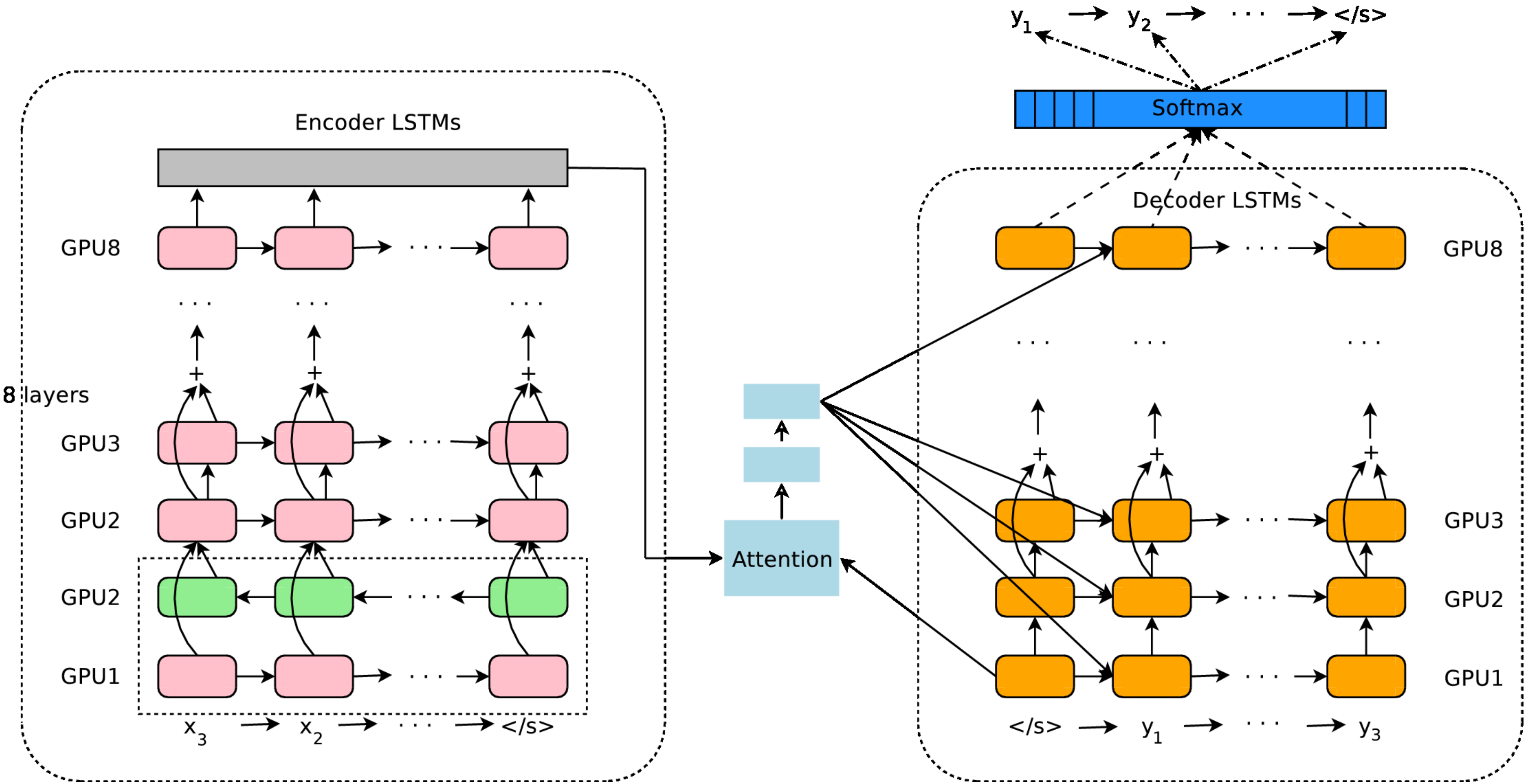


Figure 1 of paper "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144.

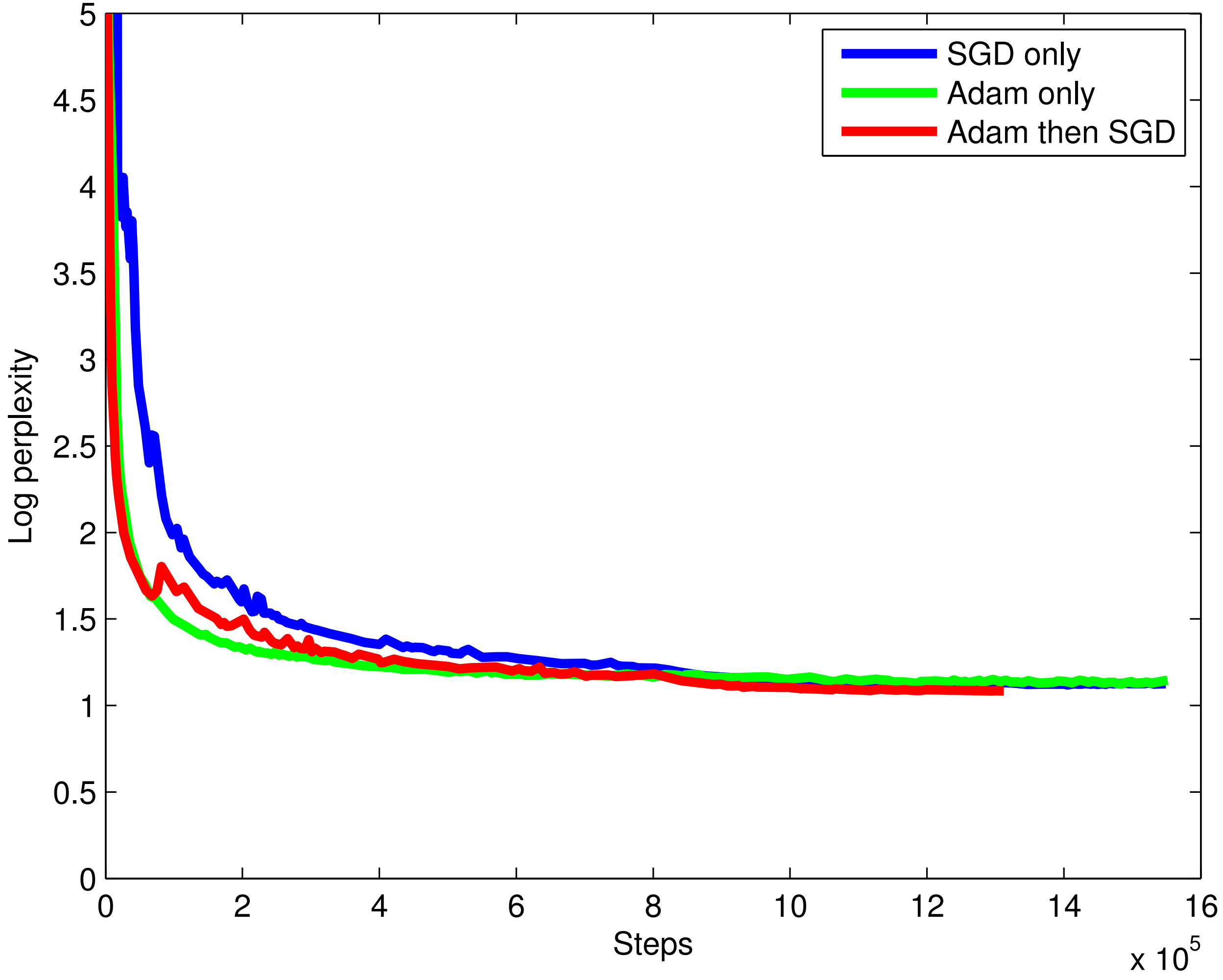


Figure 5 of paper "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144.

Google NMT

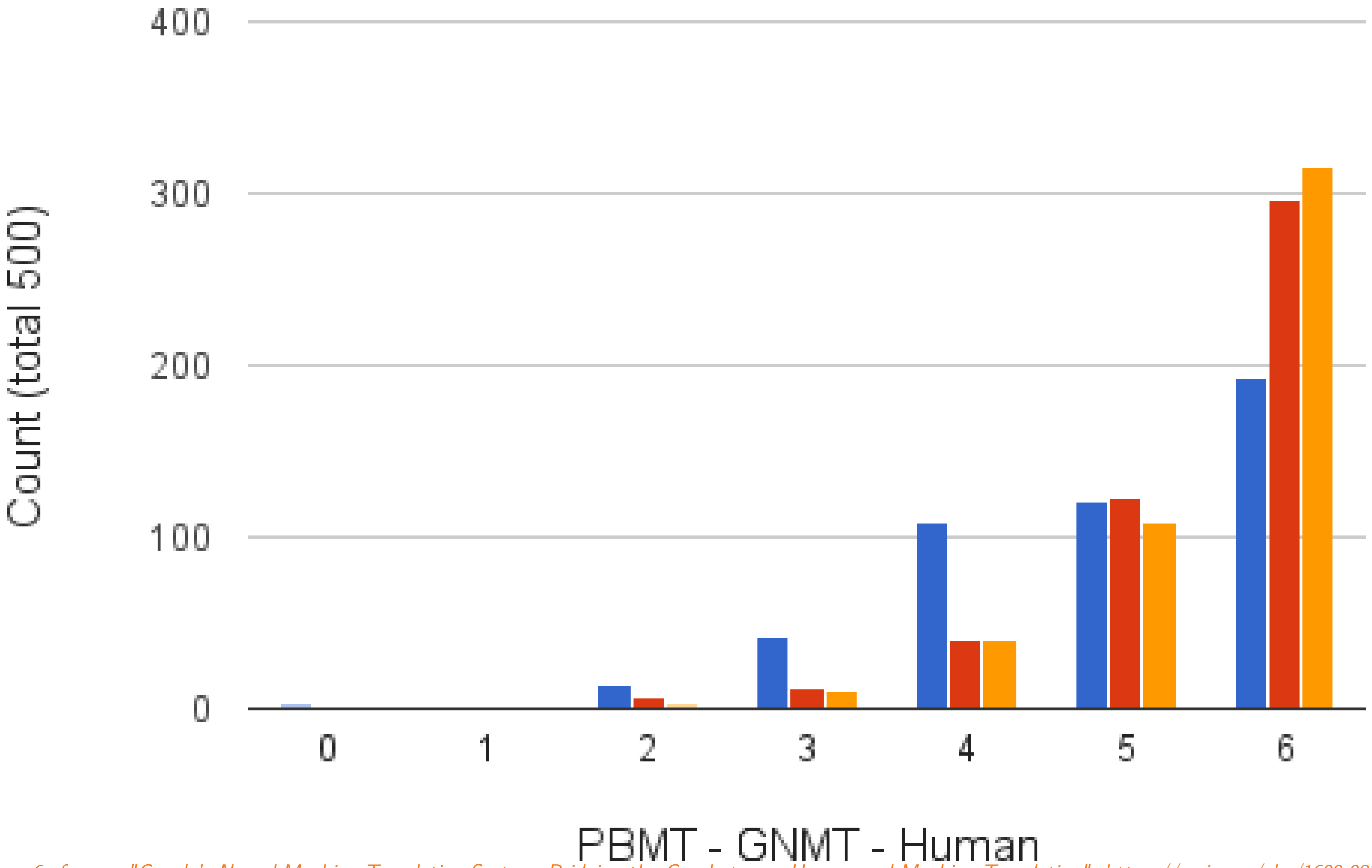
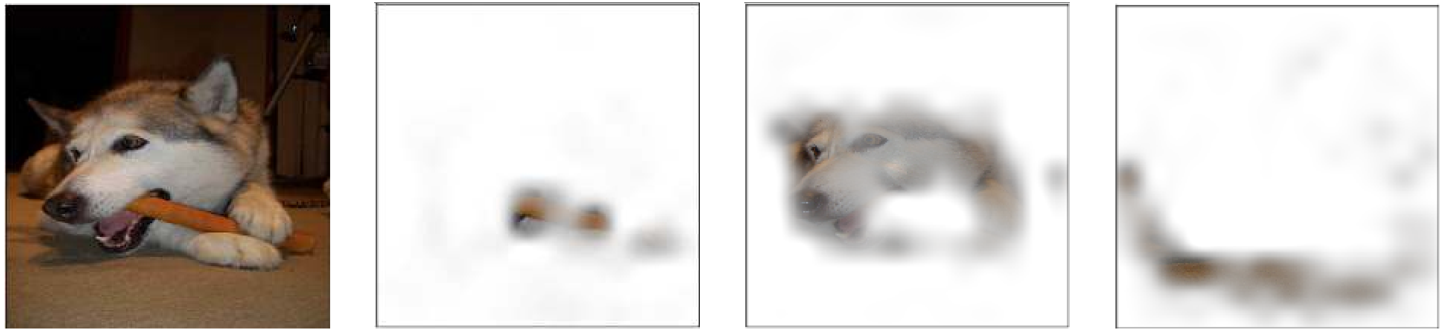


Figure 6 of paper "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144.

Fig. 5. A selection of evaluation results, grouped by human rating.

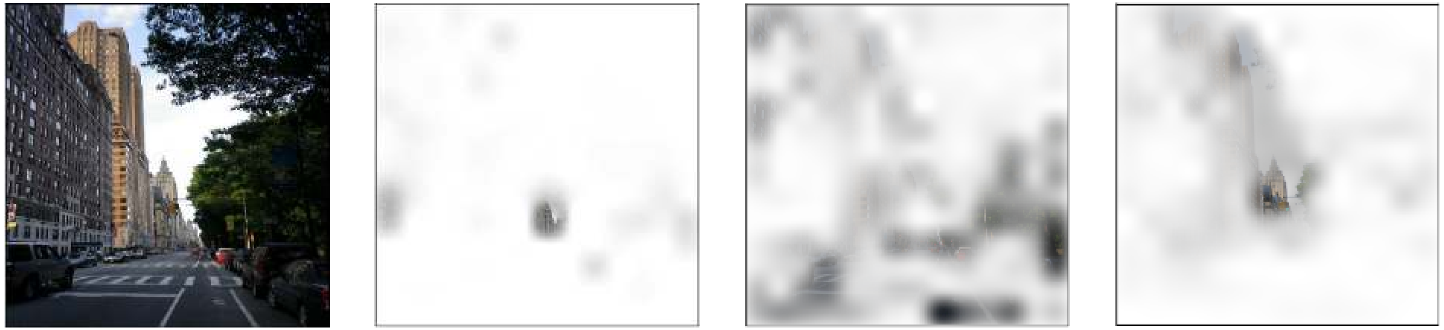
Figure 5 of "Show and Tell: Lessons learned from the 2015 MSCOCO...", https://arxiv.org/abs/1609.06647.



What vegetable is the dog chewing on?
MCB: carrot
GT: carrot

What kind of dog is this?
MCB: husky
GT: husky

What kind of flooring does the room have?
MCB: carpet
GT: carpet



What color is the traffic light?
MCB: green
GT: green

Is this an urban area?
MCB: yes
GT: yes

Where are the buildings?
MCB: in background
GT: on left

Figure 6 of "Multimodal Compact Bilinear Pooling for VQA and Visual Grounding", https://arxiv.org/abs/1606.01847.

Multilingual and Unsupervised Translation

Many attempts at multilingual translation.

- Individual encoders and decoders, shared attention.
- Shared encoders and decoders.

Surprisingly, even unsupervised translation is attempted lately. By unsupervised we understand settings where we have access to large monolingual corpora, but no parallel data.

In 2019, the best unsupervised systems were on par with the best 2014 supervised systems.

		WMT-14			
		fr-en	en-fr	de-en	en-de
Unsupervised	Proposed system	33.5	36.2	27.0	22.5
	*detok. SacreBLEU**	33.2	33.6	26.4	21.2
Supervised	WMT best*	35.0	35.8	29.0	20.6†
	Vaswani et al. (2017)	-	41.0	-	28.4
	Edunov et al. (2018)	-	45.6	-	35.0

Table 3: Results of the proposed method in comparison to different supervised systems (BLEU).

Table 3 of paper "An Effective Approach to Unsupervised Machine Translation", https://arxiv.org/abs/1902.01313.