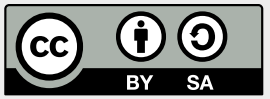
Recurrent Neural Networks III

Milan Straka

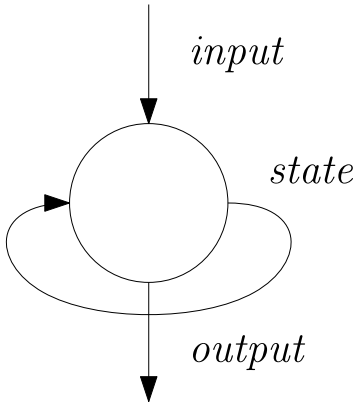
April 29, 2019

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

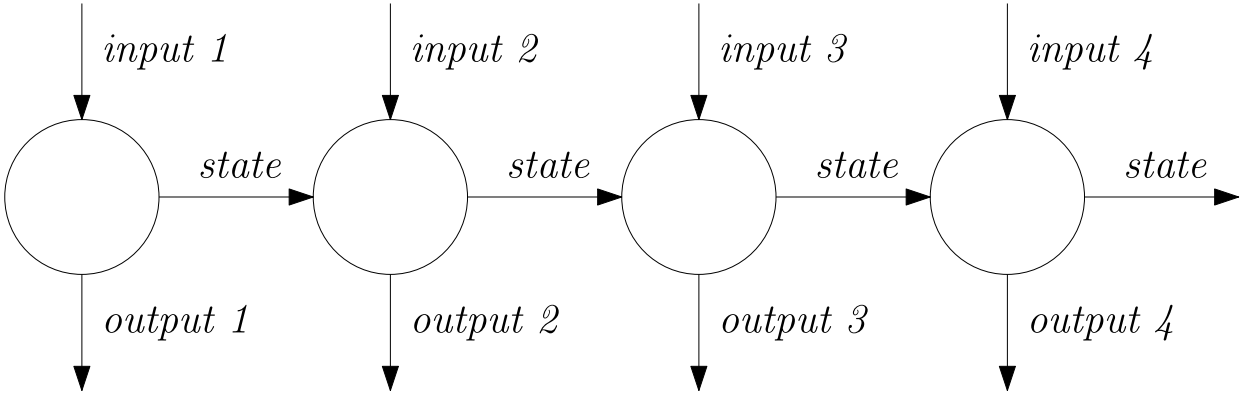
Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



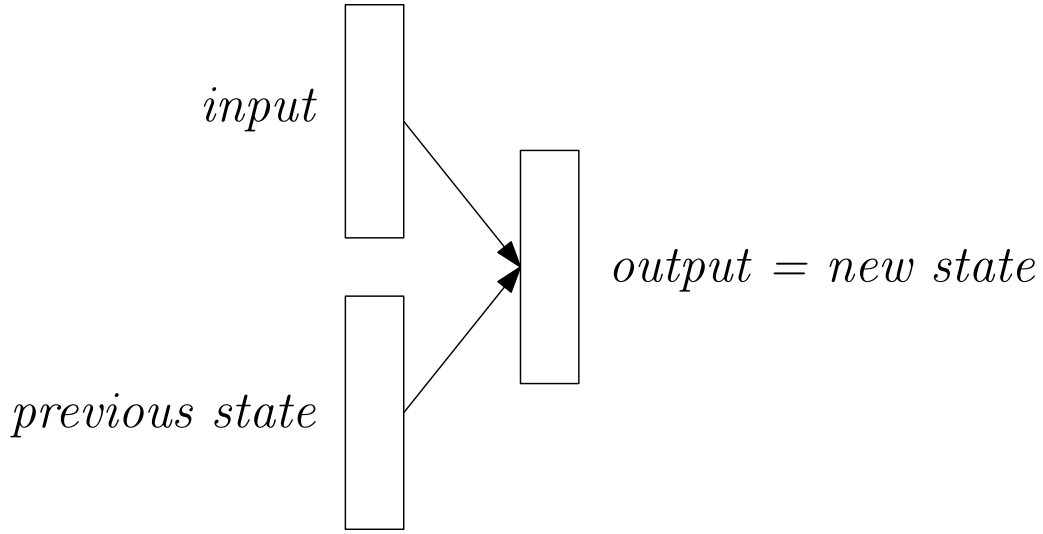
Single RNN cell



Unrolled RNN cells



Basic RNN Cell



Given an input $\boldsymbol{x}^{(t)}$ and previous state $\boldsymbol{s}^{(t-1)}$, the new state is computed as

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}).$$

One of the simplest possibilities is

$$\boldsymbol{s}^{(t)} = \tanh(\boldsymbol{U}\boldsymbol{s}^{(t-1)} + \boldsymbol{V}\boldsymbol{x}^{(t)} + \boldsymbol{b}).$$

Basic RNN Cell

Basic RNN cells suffer a lot from vanishing/exploding gradients (*the challenge of long-term dependencies*).

If we simplify the recurrence of states to

$$\boldsymbol{s}^{(t)} = \boldsymbol{U}\boldsymbol{s}^{(t-1)},$$

we get

$$\boldsymbol{s}^{(t)} = \boldsymbol{U}^t\boldsymbol{s}^{(0)}.$$

If U has eigenvalue decomposition of $\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$, we get

$$\boldsymbol{s}^{(t)} = \boldsymbol{Q}\boldsymbol{\Lambda}^t\boldsymbol{Q}^{-1}\boldsymbol{s}^{(0)}.$$

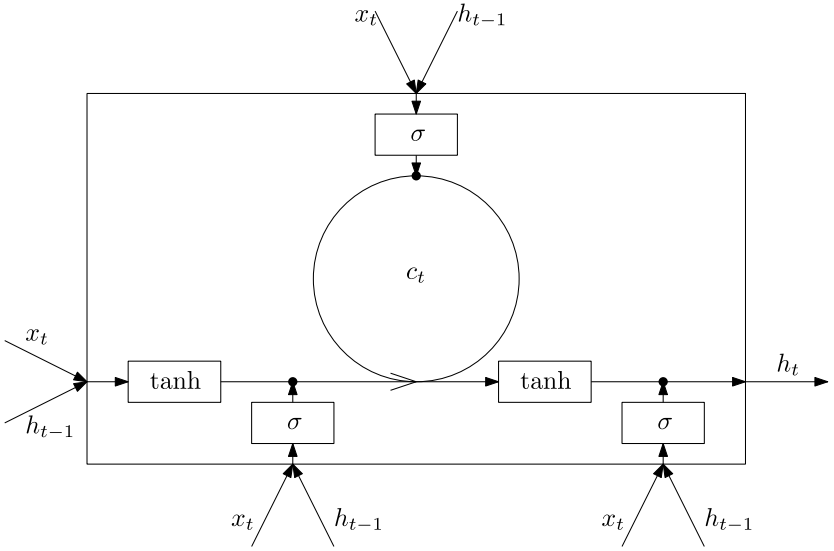
The main problem is that the *same* function is iteratively applied many times.

Several more complex RNN cell variants have been proposed, which alleviate this issue to some degree, namely **LSTM** and **GRU**.

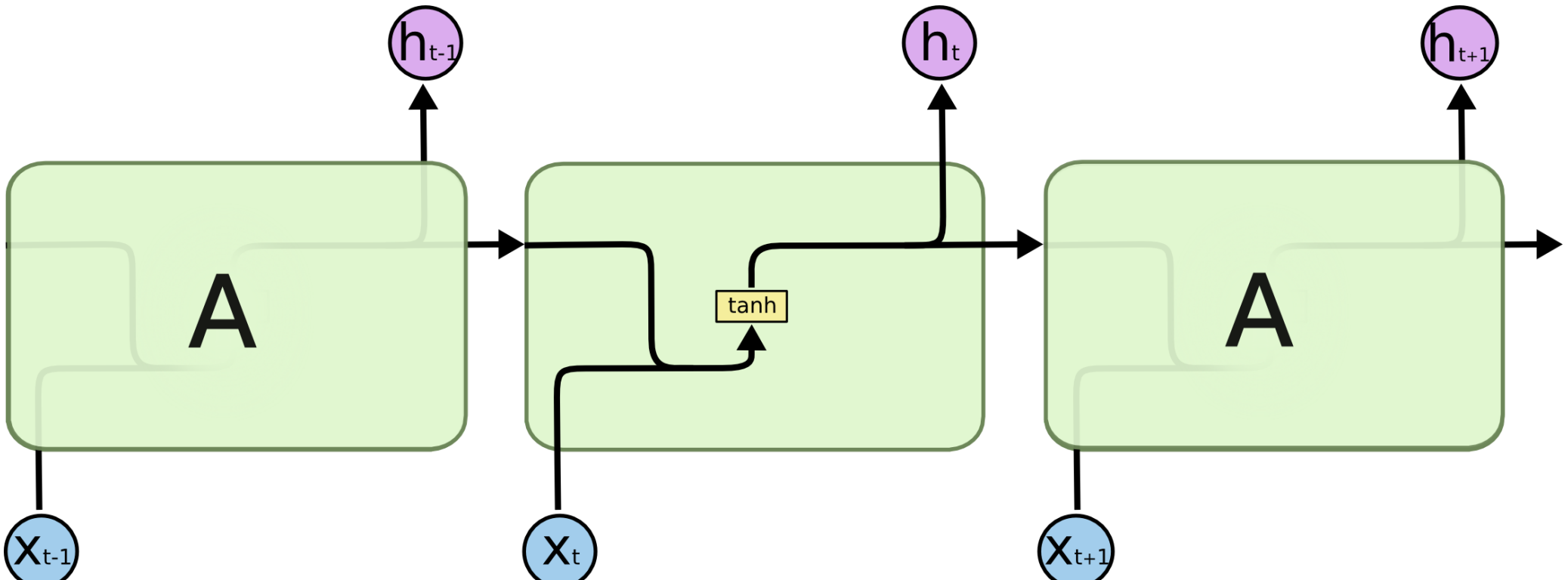
Long Short-Term Memory

Later in Gers, Schmidhuber & Cummins (1999) a possibility to *forget* information from memory cell $\boldsymbol{c}_t$ was added.

$$\begin{aligned}
\boldsymbol{i}_t &\leftarrow \sigma(\boldsymbol{W}^i \boldsymbol{x}_t + \boldsymbol{V}^i \boldsymbol{h}_{t-1} + \boldsymbol{b}^i) \\
\boldsymbol{f}_t &\leftarrow \sigma(\boldsymbol{W}^f \boldsymbol{x}_t + \boldsymbol{V}^f \boldsymbol{h}_{t-1} + \boldsymbol{b}^f) \\
\boldsymbol{o}_t &\leftarrow \sigma(\boldsymbol{W}^o \boldsymbol{x}_t + \boldsymbol{V}^o \boldsymbol{h}_{t-1} + \boldsymbol{b}^o) \\
\boldsymbol{c}_t &\leftarrow \boldsymbol{f}_t \cdot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \cdot \tanh(\boldsymbol{W}^y \boldsymbol{x}_t + \boldsymbol{V}^y \boldsymbol{h}_{t-1} + \boldsymbol{b}^y) \\
\boldsymbol{h}_t &\leftarrow \boldsymbol{o}_t \cdot \tanh(\boldsymbol{c}_t)
\end{aligned}$$

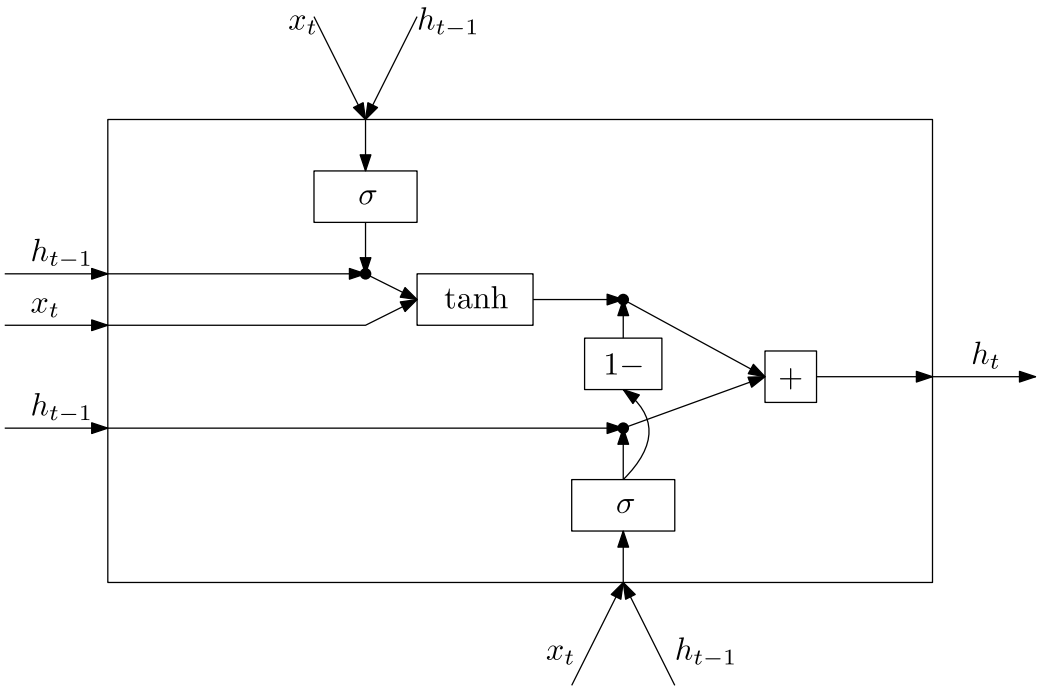


Long Short-Term Memory



http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-SimpleRNN.png

Gated Recurrent Unit



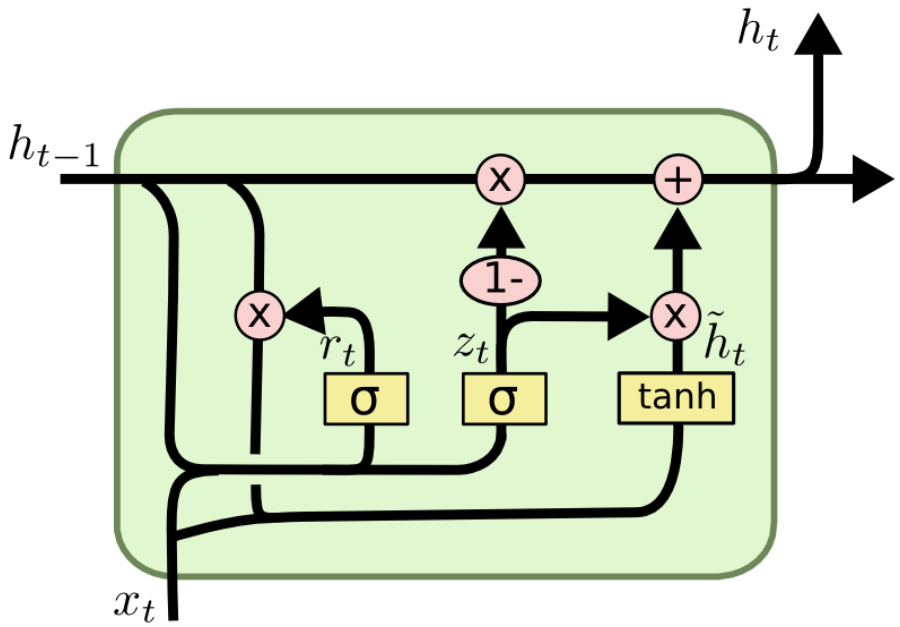
$$\boldsymbol{r}_t \leftarrow \sigma(\boldsymbol{W}^r \boldsymbol{x}_t + \boldsymbol{V}^r \boldsymbol{h}_{t-1} + \boldsymbol{b}^r)$$

$$\boldsymbol{u}_t \leftarrow \sigma(\boldsymbol{W}^u \boldsymbol{x}_t + \boldsymbol{V}^u \boldsymbol{h}_{t-1} + \boldsymbol{b}^u)$$

$$\hat{\boldsymbol{h}}_t \leftarrow \tanh(\boldsymbol{W}^h \boldsymbol{x}_t + \boldsymbol{V}^h (\boldsymbol{r}_t \cdot \boldsymbol{h}_{t-1}) + \boldsymbol{b}^h)$$

$$\boldsymbol{h}_t \leftarrow \boldsymbol{u}_t \cdot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}_t) \cdot \hat{\boldsymbol{h}}_t$$

Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-var-GRU.png

One-hot encoding considers all words to be independent of each other.

However, words are not independent – some are more similar than others.

Ideally, we would like some kind of similarity in the space of the word representations.

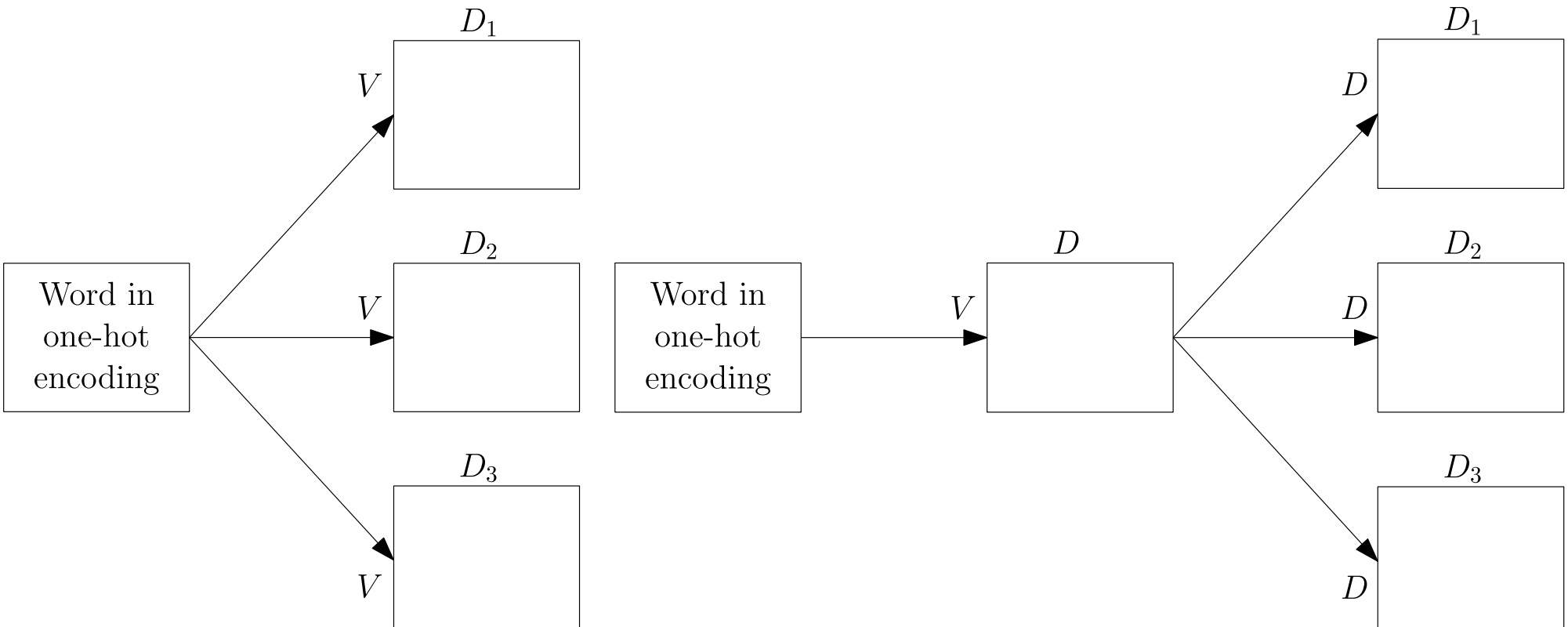
Distributed Representation

The idea behind distributed representation is that objects can be represented using a set of common underlying factors.

We therefore represent words as fixed-size *embeddings* into $\mathbb{R}^d$ space, with the vector elements playing role of the common underlying factors.

Word Embeddings

The word embedding layer is in fact just a fully connected layer on top of one-hot encoding. However, it is important that this layer is *shared* across the whole network.



Recurrent Character-level WEs

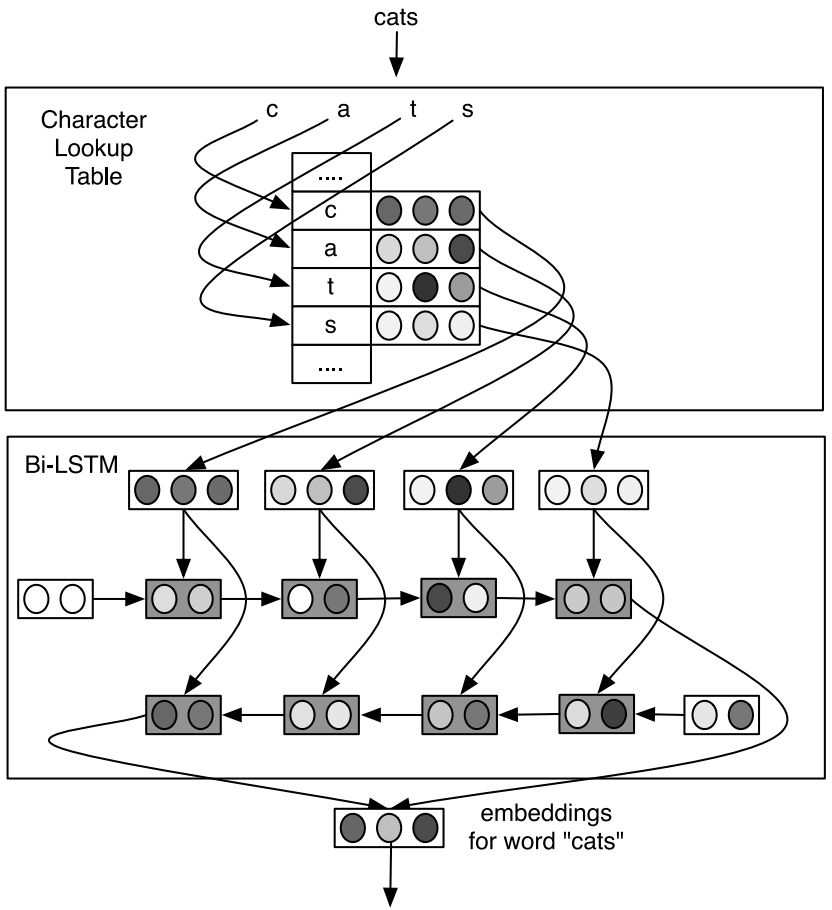


Figure 1 of paper "Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation", https://arxiv.org/abs/1508.02096.

Convolutional Character-level WEs

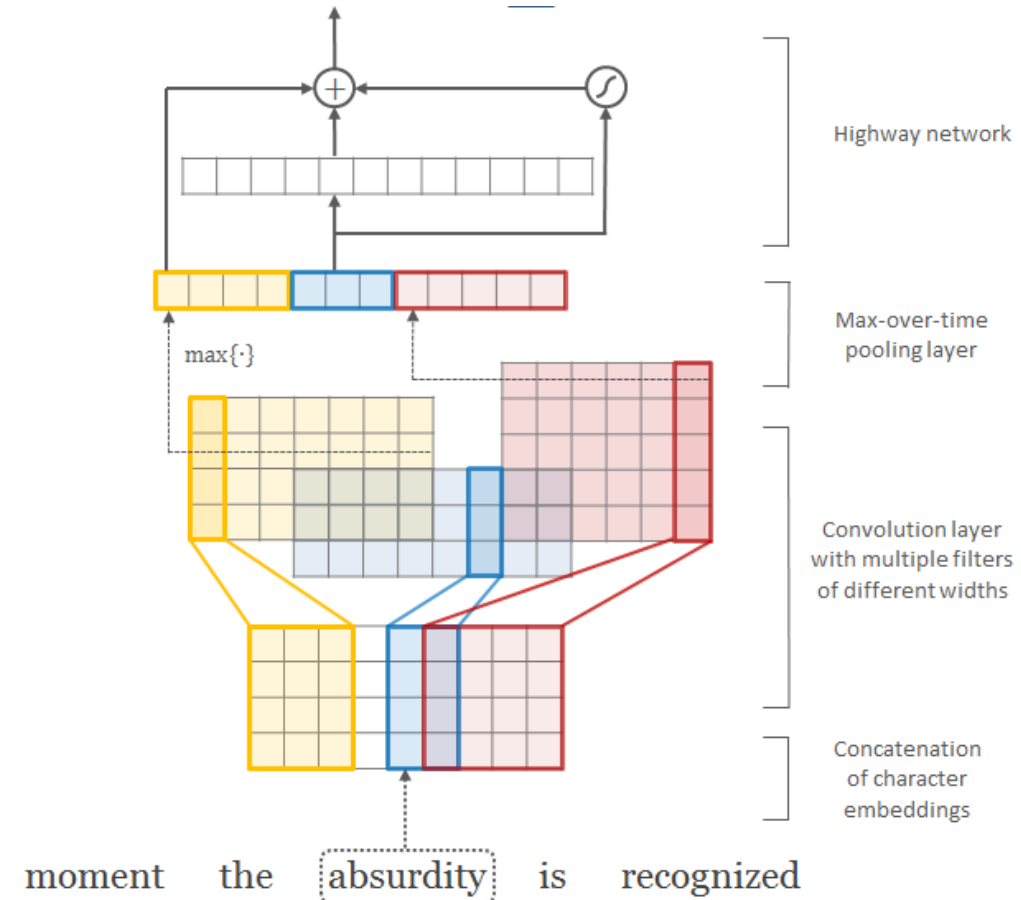
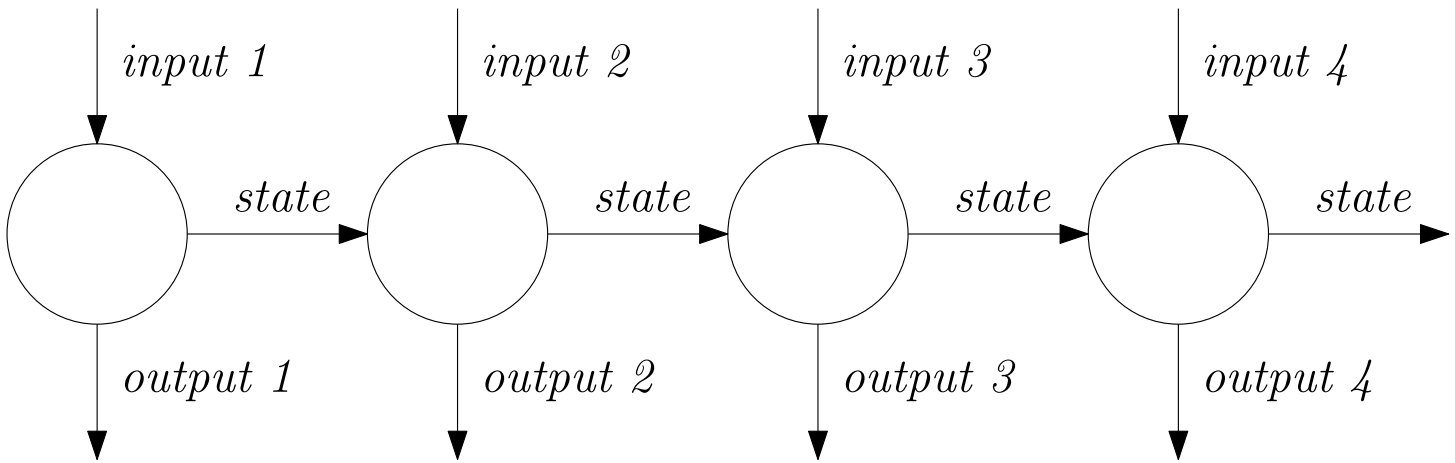


Figure 1 of paper "Character-Aware Neural Language Models", https://arxiv.org/abs/1508.06615.

Sequence Element Classification

Use outputs for individual elements.



Sequence Representation

Use state after processing the whole sequence (alternatively, take output of the last element).

Structured Prediction

Consider generating a sequence of $y_1, \ldots, y_N \in Y^N$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Predicting each sequence element independently models the distribution $P(y_i|\boldsymbol{X})$.

However, there may be dependencies among the y_i themselves, which is difficult to capture by independent element classification.

Linear-Chain Conditional Random Fields (CRF)

Linear-chain Conditional Random Fields, usually abbreviated only to CRF, acts as an output layer. It can be considered an extension of a softmax – instead of a sequence of independent softmaxes, CRF is a sentence-level softmax, with additional weights for neighboring sequence elements.

$$s(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{A}) = \sum_{i=1}^{N} \left(\boldsymbol{A}_{y_{i-1}, y_i} + f_{\boldsymbol{\theta}}(y_i | \boldsymbol{X}) \right)$$

$$p(\boldsymbol{y}|\boldsymbol{X}) = \mathrm{softmax}_{\boldsymbol{z} \in Y^N} \left(s(\boldsymbol{X}, \boldsymbol{z}) \right)_{\boldsymbol{z}}$$

$$\log p(\boldsymbol{y}|\boldsymbol{X}) = s(\boldsymbol{X}, \boldsymbol{y}) - \mathrm{logadd}_{\boldsymbol{z} \in Y^N}(s(\boldsymbol{X}, \boldsymbol{z}))$$

Computation

We can compute $p(\boldsymbol{y}|\boldsymbol{X})$ efficiently using dynamic programming. If we denote $\alpha_t(k)$ as probability of all sentences with t elements with the last y being k.

The core idea is the following:

$$\alpha_t(k) = f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X}) + \operatorname{logadd}_{j \in Y}(\alpha_{t-1}(j) + \boldsymbol{A}_{j,k}).$$

For efficient implementation, we use the fact that

$$\ln(a+b) = \ln a + \ln(1 + e^{\ln b - \ln a}).$$

Decoding

We can perform optimal decoding, by using the same algorithm, only replacing logadd with $\max$ and tracking where the maximum was attained.

Applications

CRF output layers are useful for *span labeling* tasks, like

- named entity recognition
- dialog slot filling

Connectionist Temporal Classification

Let us again consider generating a sequence of $y_1, \ldots, y_M$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, but this time $M \leq N$ and there is no explicit alignment of $\boldsymbol{x}$ and y in the gold data.

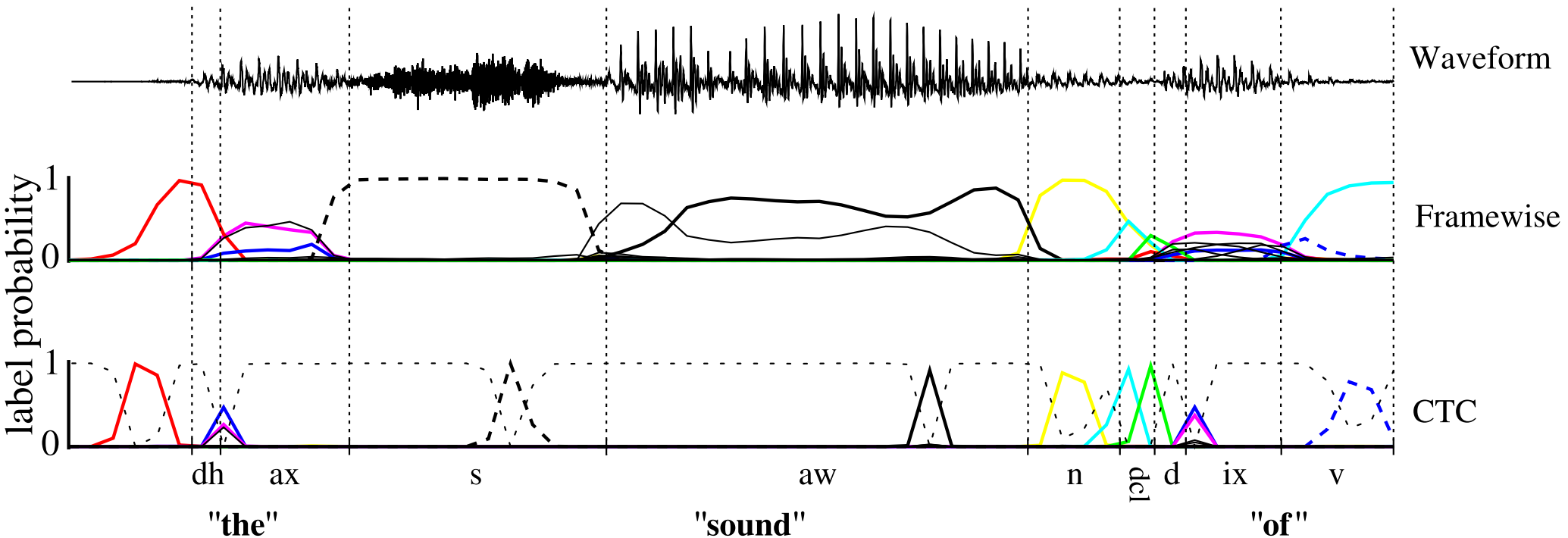


Figure 7.1 of the dissertation "Supervised Sequence Labelling with Recurrent Neural Networks" by Alex Graves.

Connectionist Temporal Classification

We enlarge the set of output labels by a – (*blank*) and perform a classification for every input element to produce an *extended labeling*. We then post-process it by the following rules (denoted $\mathcal{B}$):

1. We remove neighboring symbols.
2. We remove the –.

Because the explicit alignment of inputs and labels is not known, we consider *all possible* alignments.

Denoting the probability of label l at time t as p_l^t, we define

$$\alpha^t(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi}: \mathcal{B}(\boldsymbol{\pi}_{1:t}) = \boldsymbol{y}_{1:s}} \prod_{t'=1}^{t} p^{t'}_{\boldsymbol{\pi}_{t'}}.$$

CRF and CTC Comparison

In CRF, we normalize the whole sentences, therefore we need to compute unnormalized probabilities for all the (exponentially many) sentences. Decoding can be performed optimally.

In CTC, we normalize per each label. However, because we do not have explicit alignment, we compute probability of a labeling by summing probabilities of (generally exponentially many) extended labelings.

Computation

When aligning an extended labeling to a regular one, we need to consider whether the extended labeling ends by a *blank* or not. We therefore define

$$\alpha^t_-(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi}: \mathcal{B}(\boldsymbol{\pi}_{1:t}) = \boldsymbol{y}_{1:s}, \pi_t = -} \prod_{t'=1}^{t} p^{t'}_{\boldsymbol{\pi}_{t'}}$$

$$\alpha^t_*(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi}: \mathcal{B}(\boldsymbol{\pi}_{1:t}) = \boldsymbol{y}_{1:s}, \pi_t \neq -} \prod_{t'=1}^{t} p^{t'}_{\boldsymbol{\pi}_{t'}}$$

and compute $\alpha^t(s)$ as $\alpha^t_-(s) + \alpha^t_*(s)$.

Computation

We initialize αs as follows:

- $\alpha_-^1(0) \leftarrow p_-^1$
- $\alpha_*^1(1) \leftarrow p_{y_1}^1$

We then proceed recurrently according to:

- $\alpha_-^t(s) \leftarrow p_-^t(\alpha_-^{t-1}(s) + \alpha_*^{t-1}(s))$
- $\alpha_*^t(s) \leftarrow \begin{cases} p_{y_s}^t(\alpha_*^{t-1}(s) + \alpha_*^{t-1}(s-1) + a_-^{t-1}(s-1)), \text{if } y_s \neq y_{s-1} \\ p_{y_s}^t(\alpha_*^{t-1}(s) + a_-^{t-1}(s-1)), \text{if } y_s = y_{s-1} \end{cases}$

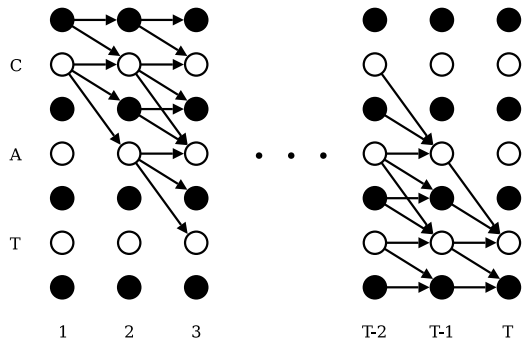
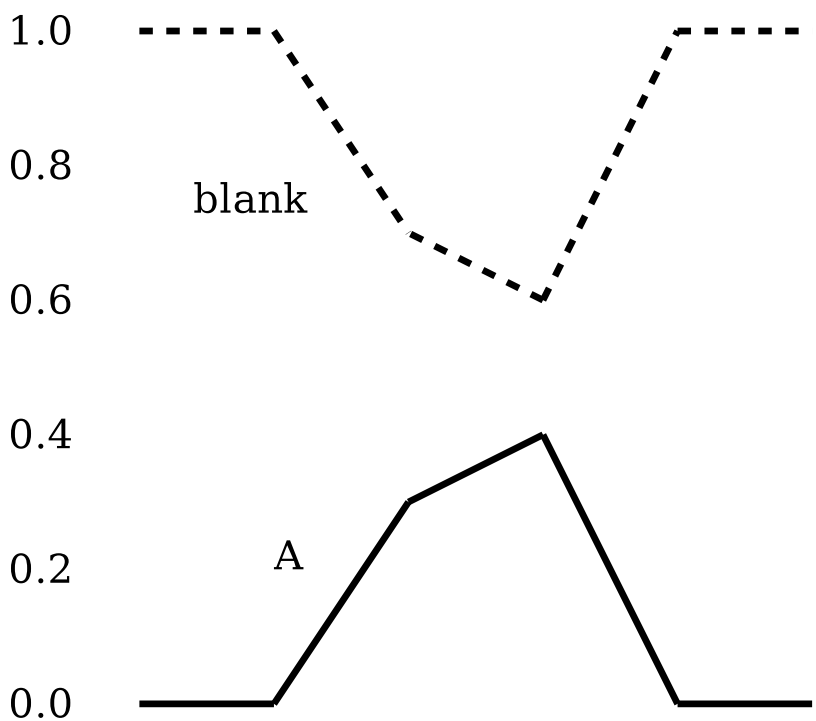


Figure 7.3 of the dissertation "Supervised Sequence Labelling with Recurrent Neural Networks" by Alex Graves.

CTC Decoding

Unlike CRF, we cannot perform the decoding optimally. The key observation is that while an optimal extended labeling can be extended into an optimal labeling of a larger length, the same does not apply to regular (non-extended) labeling. The problem is that regular labeling coresponds to many extended labelings, which are modified each in a different way during an extension of the regular labeling.



$$p(l=\text{blank}) = p(- -) = 0.7*0.6 = 0.42$$

$$p(l=A) = p(AA)+p(A-)+p(-A) = 0.3*0.4 + 0.3*0.6 + 0.7*0.4 = 0.58$$

Figure 7.5 of the dissertation "Supervised Sequence Labelling with Recurrent Neural Networks" by Alex Graves.

Beam Search

To perform beam search, we keep k best regular labelings for each prefix of the extended labelings. For each regular labeling we keep both $\alpha_{\_}$ and a_{*} and by *best* we mean such regular labelings with maximum $\alpha_{\_} + \alpha_{*}$.

To compute best regular labelings for longer prefix of extended labelings, for each regular labeling in the beam we consider the following cases:

- adding a *blank* symbol, i.e., updating both $\alpha_{\_}$ and α_{*};
- adding any non-blank symbol, i.e., updating α_{*}.

Finally, we merge the resulting candidates according to their regular labeling and keep only the k best.

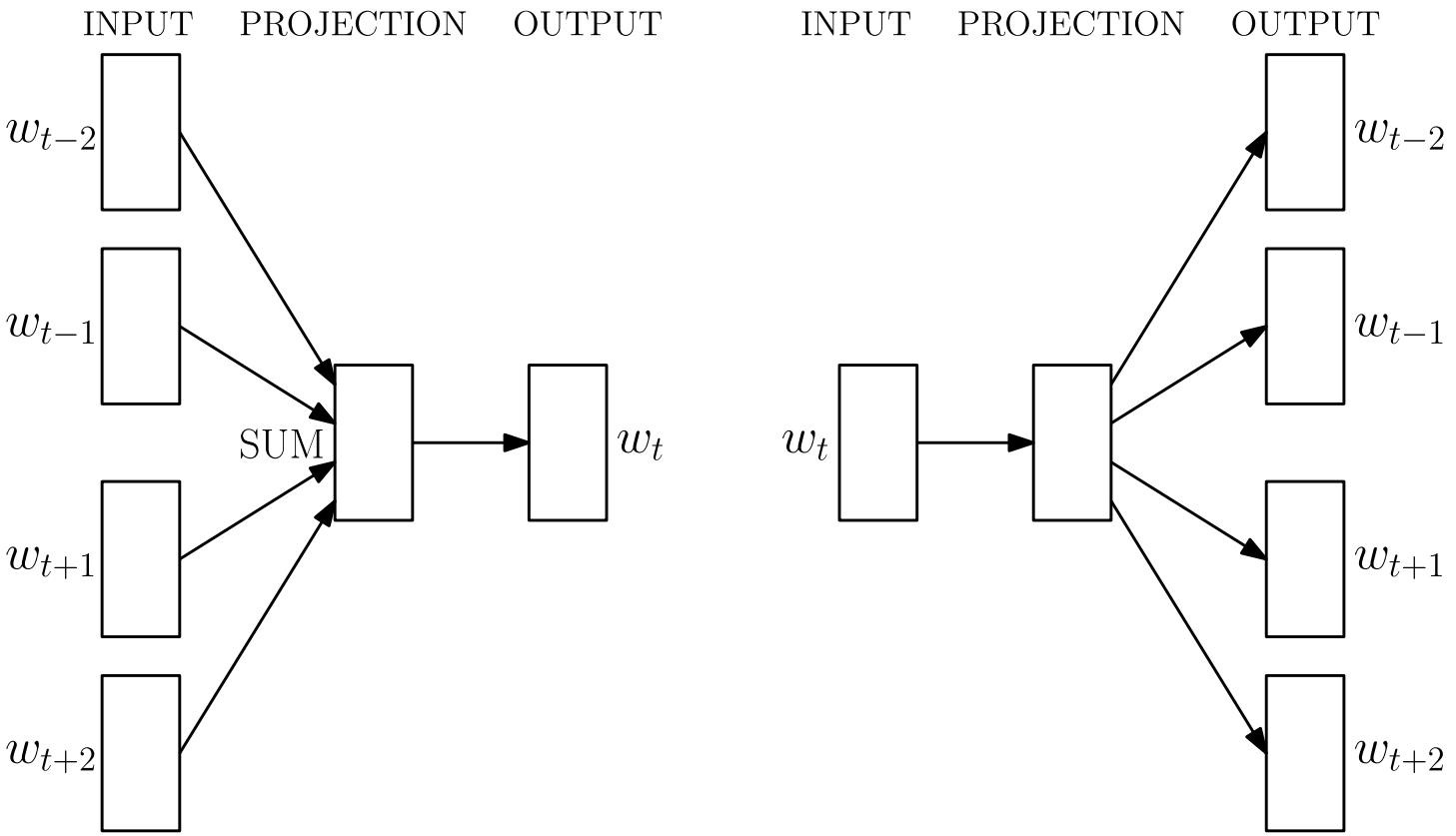
Unsupervised Word Embeddings

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).



CBOW (Continuous Bag Of Words) **Skip-gram**

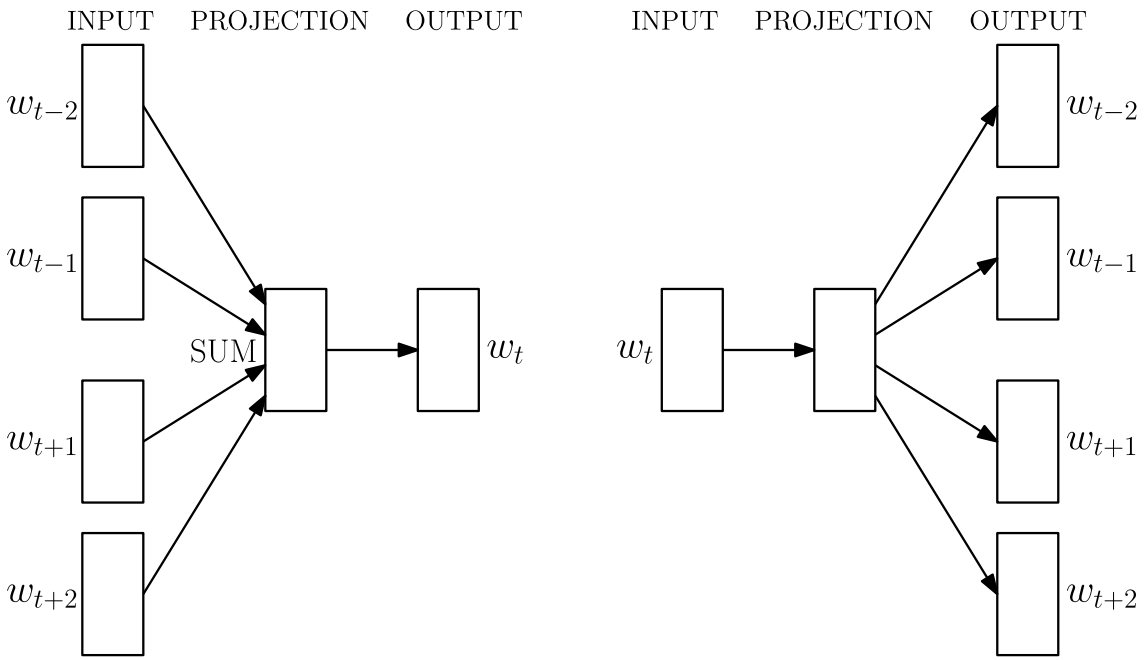
Mikolov et al. (2013) proposed two very simple architectures for precomputing word embeddings, together with a C multi-threaded implementation `word2vec`.

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.

Word2Vec – SkipGram Model



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

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

Word2Vec – Hierarchical Softmax

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

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

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:

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

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).

Recurrent Character-level WEs

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 flooring	tech-dominated tech-heavy	british-born polish-born	micromanage micromanaged	restaurants eaterie	dendrite dendrites
`sg`	bookcases built-ins	technology-heavy .ixic	most-capped ex-scotland	defang internalise	restaurants delis	epithelial 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

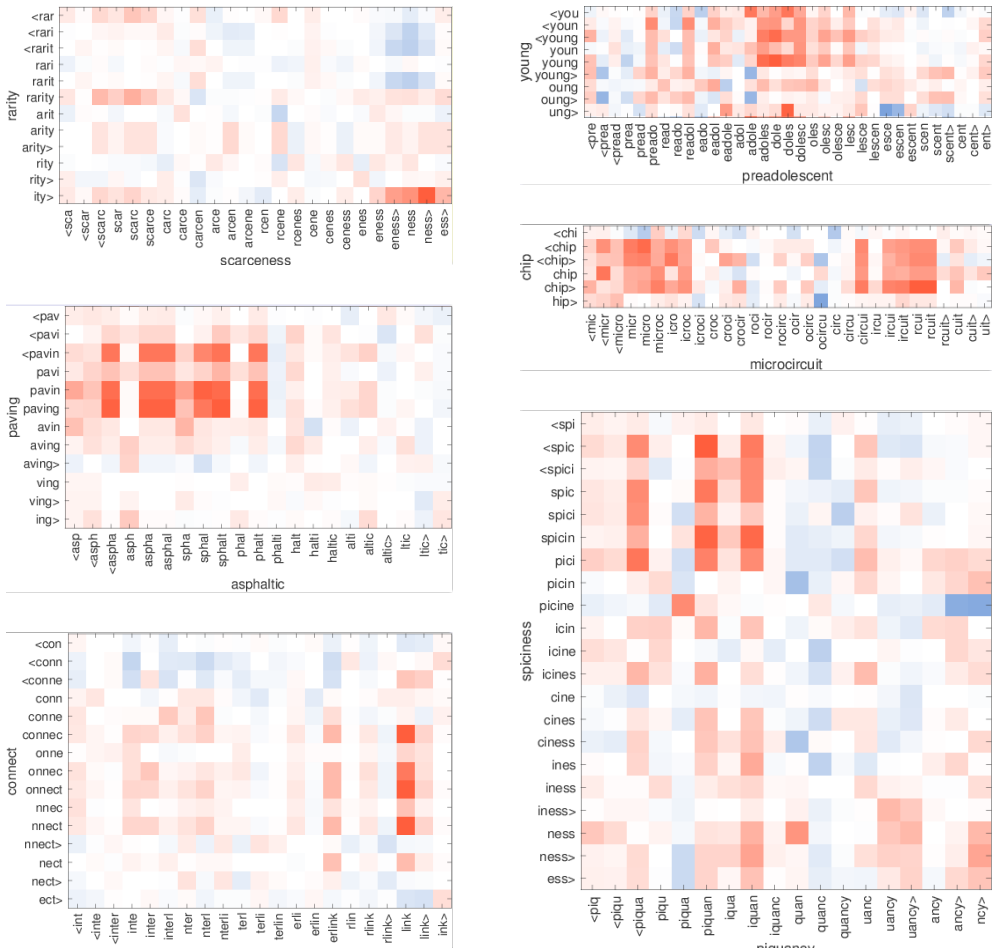


Figure 2: Illustration of the similarity between character n-grams in out-of-vocabulary words. For each pair, only one word is OOV, and is shown on the x axis. Red indicates positive cosine, while blue negative.

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

Sequence-to-Sequence Architecture

Sequence-to-Sequence Architecture

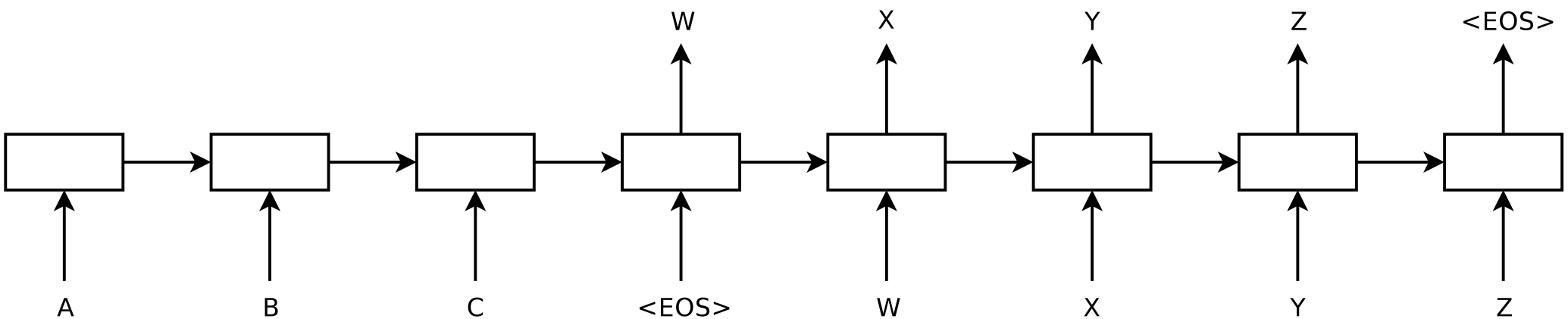


Figure 1 of paper "Sequence to Sequence Learning with Neural Networks", https://arxiv.org/abs/1409.0473.

Sequence-to-Sequence Architecture

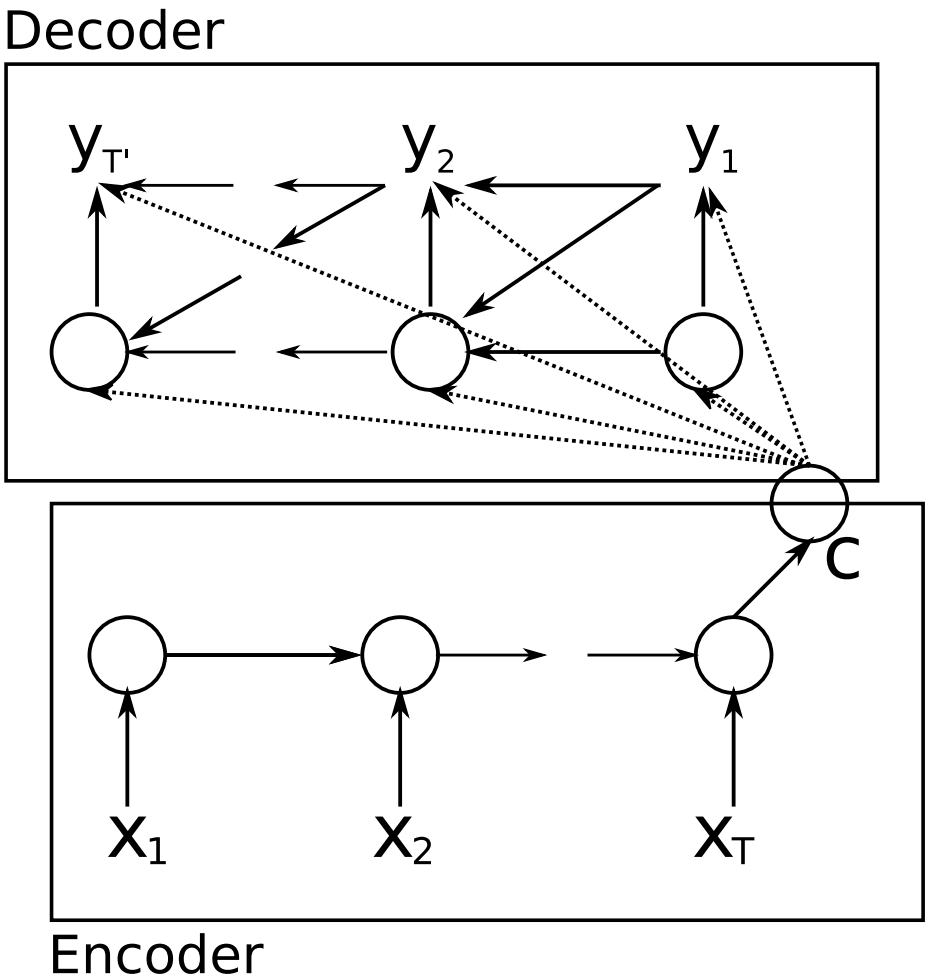


Figure 1 of paper "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", https://arxiv.org/abs/1406.1078.

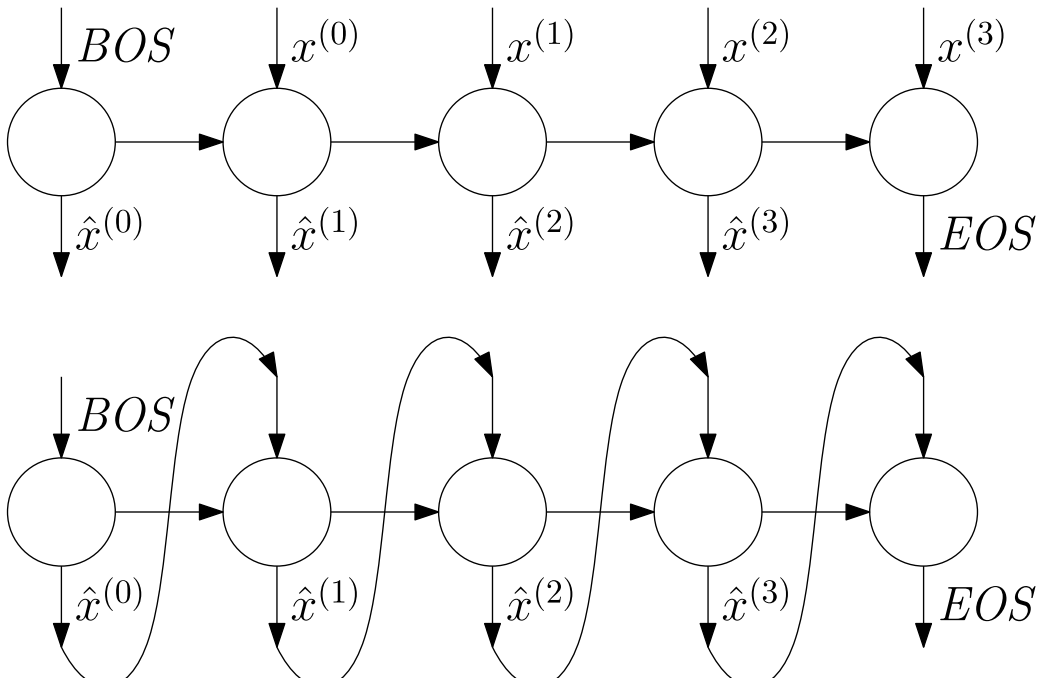
Training

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

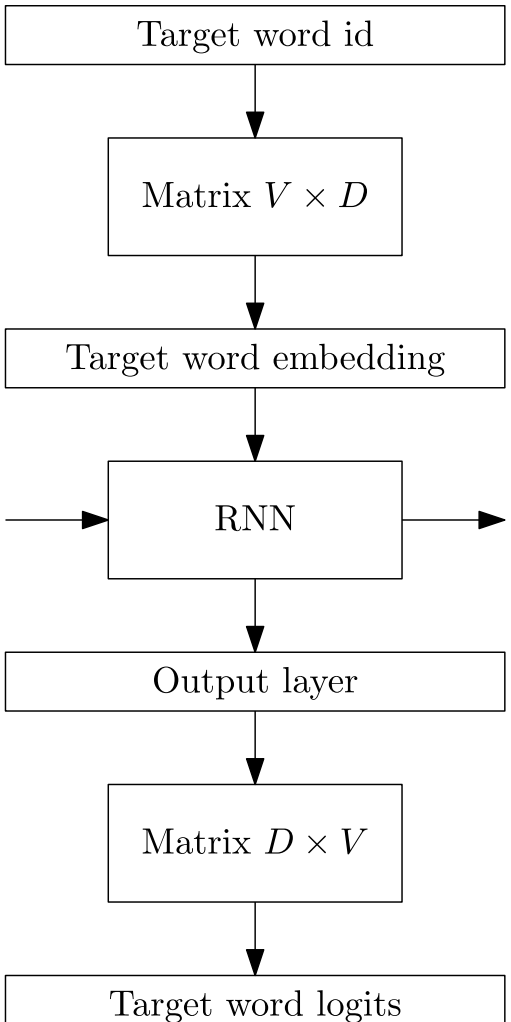
Inference

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



Attention

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

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

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

$$c_i = \sum_j \alpha_{ij} h_j$$

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

$$\alpha_i = \mathrm{softmax}(e_i),$$

with e_{ij} being

$$e_{ij} = v^\top \tanh(V h_j + W s_{i-1} + b).$$

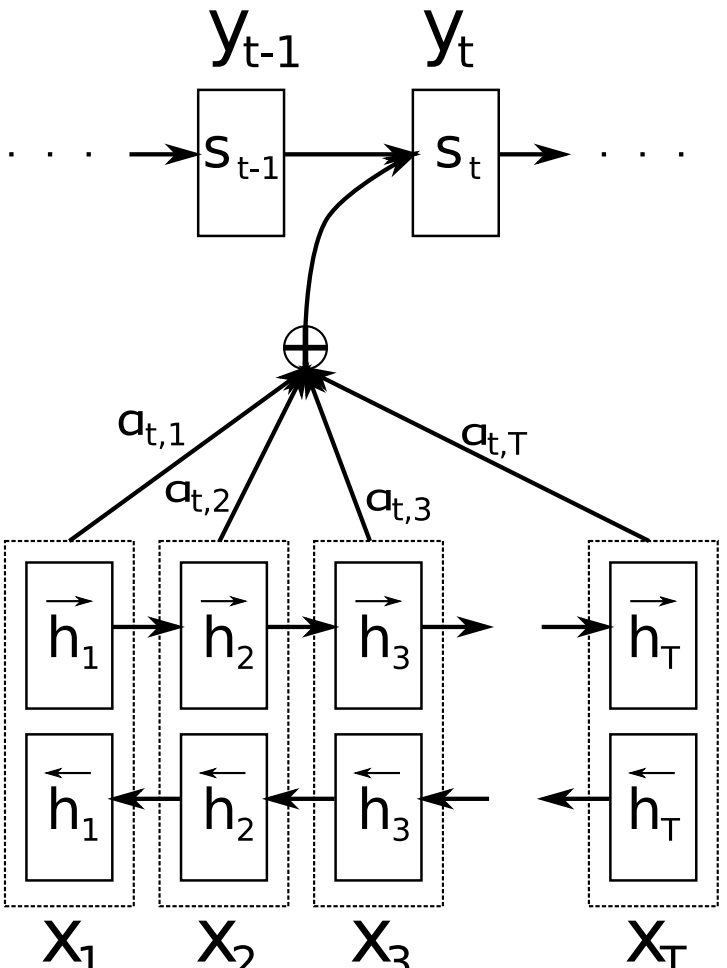


Figure 1 of paper "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473.

Attention

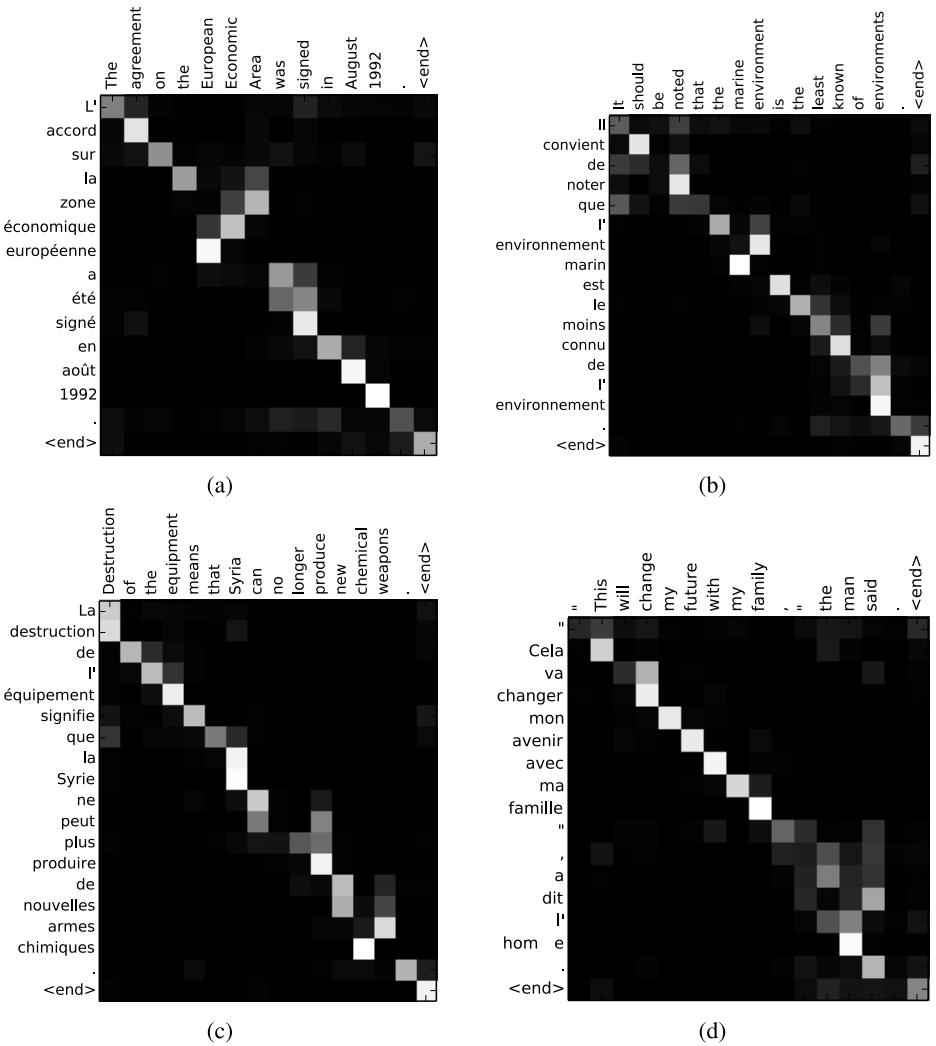


Figure 3 of paper "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473.

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 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.

  Considering a dictionary with words *low, lowest, newer, wider*:

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

- Wordpieces – Joining neighboring symbols to maximize unigram language model likelihood.

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*).

Google NMT

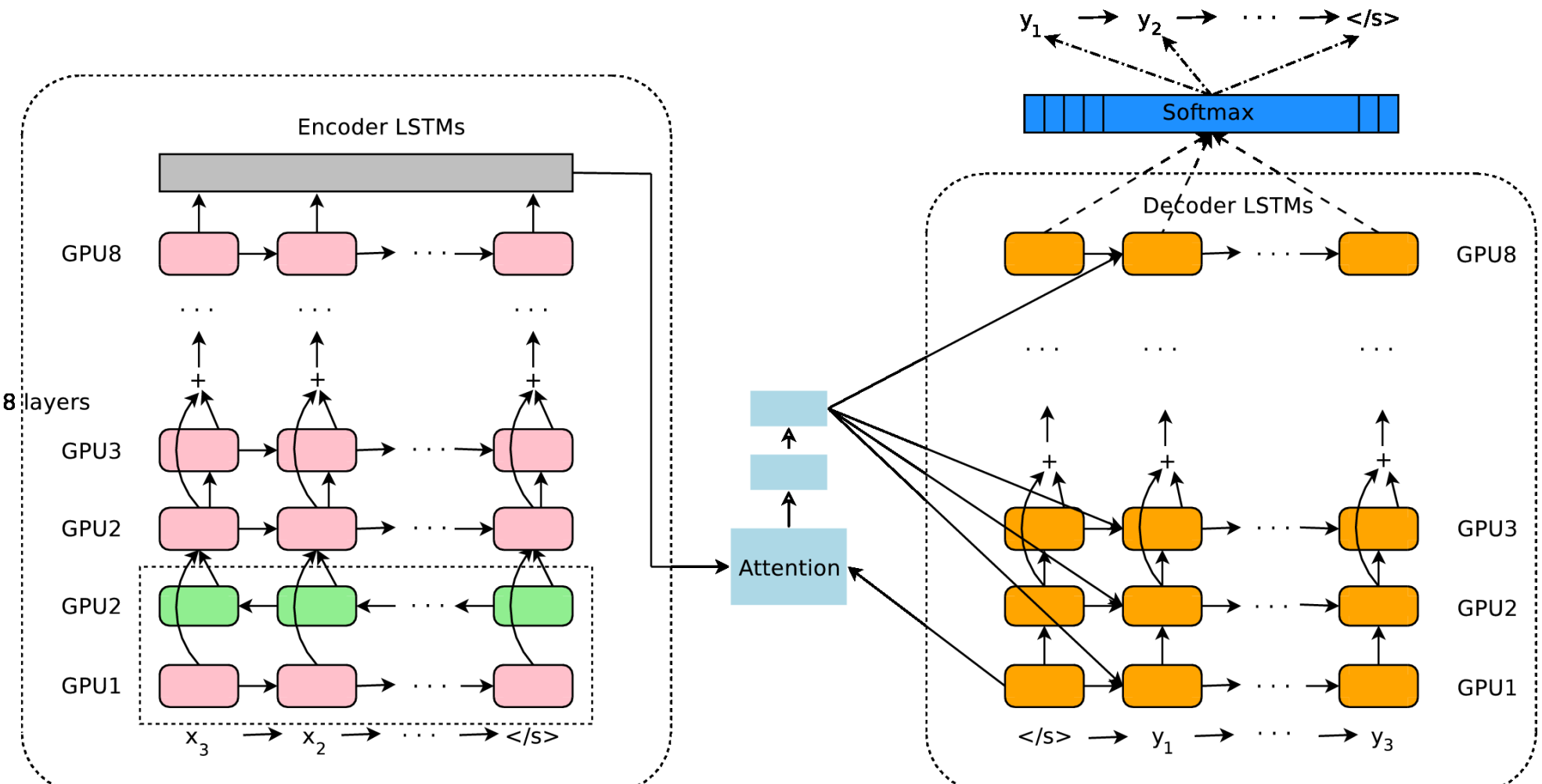


Figure 1 of paper "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144.

Google NMT

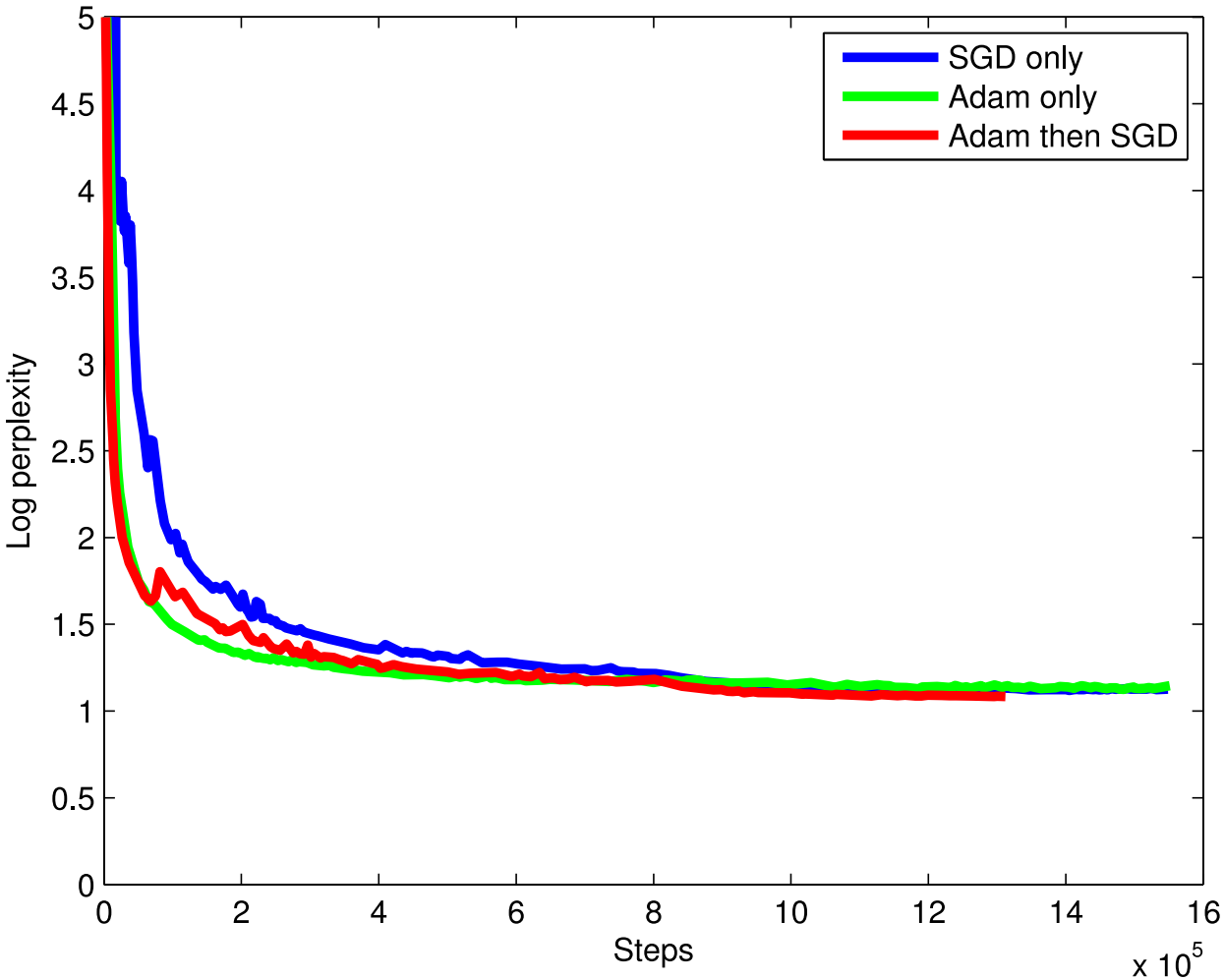


Figure 5 of paper "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144.

Google NMT

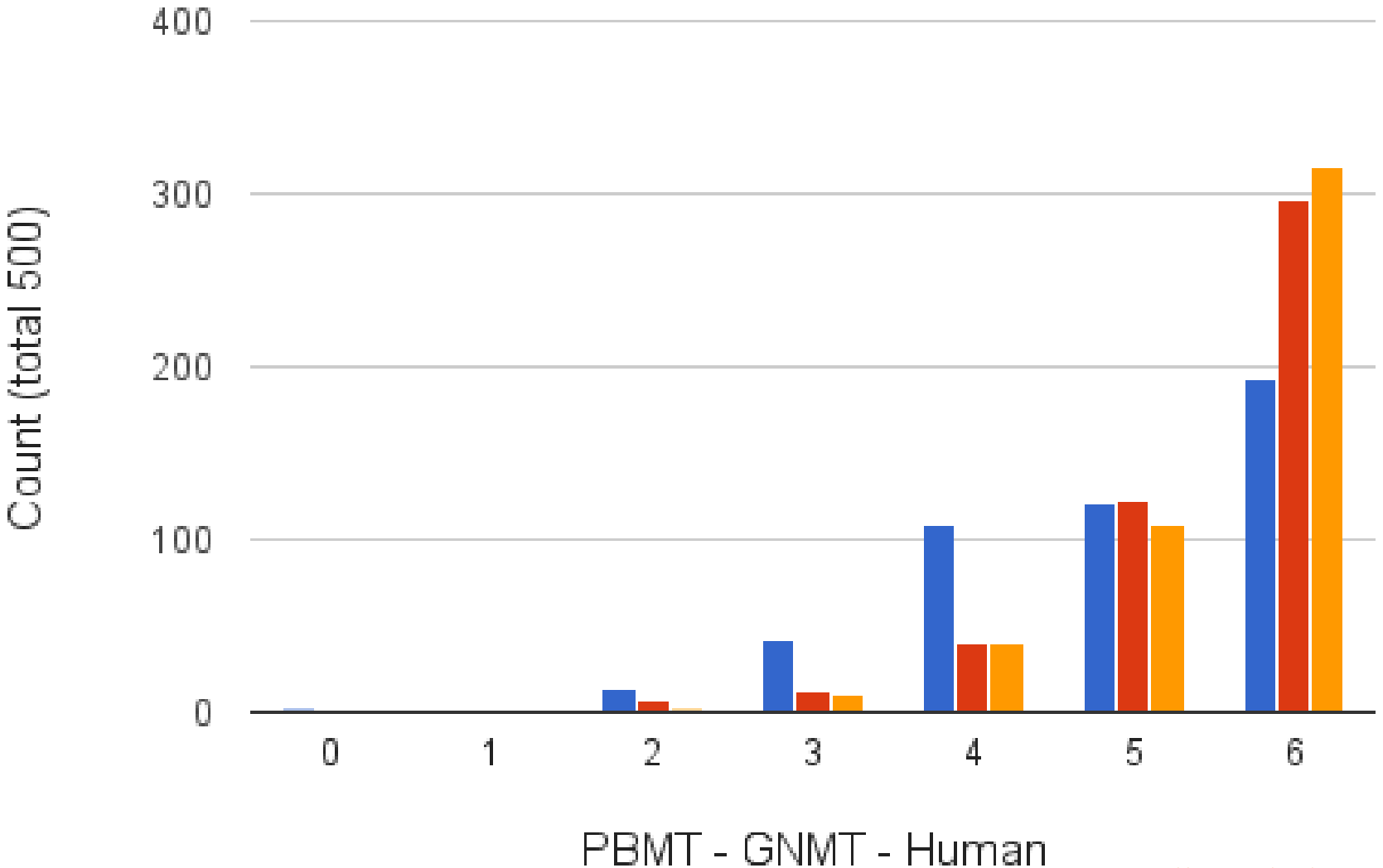


Figure 6 of paper "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144.

Beyond one Language Pair

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.

Beyond one Language Pair

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 Translation

Many attempts at multilingual translation.

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