Structured Prediction, CTC, Word2Vec

Milan Straka

April 15, 2024

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

Structured Prediction

Structured Prediction

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

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

$$\begin{tikzcd}
\boldsymbol{x}_1 \arrow[d] & \boldsymbol{x}_2 \arrow[d] & \boldsymbol{x}_3 \arrow[d] & \cdots & \boldsymbol{x}_N \arrow[d] \\
y_1 & y_2 & y_3 & \cdots & y_N
\end{tikzcd}$$

However, there may be dependencies among the y_i themselves, in the sense that not all sequences of y_i are valid; but when generating each y_i independently, the model might not be capable of generating only valid sequences.

Structured Prediction – Span Labeling

Consider for example **named entity recognition**, whose goal is to locate *named entities*, which are single words or sequences of multiple words denoting real-world objects, concepts, and events. The most common types of named entities include:

- `PER`: *people*, including names of individuals, historical figures, and even fictional characters;
- `ORG`: *organizations*, incorporating companies, government agencies, educational institutions, and others;
- `LOC`: *locations*, encompassing countries, cities, geographical features, addresses.

Compared to part-of-speech tagging, locating named entities is much more challenging – named entity mentions are generally multi-word spans, and arbitrary number of named entities can appear in a sentence (consequently, we cannot use accuracy for evaluation; F1-score is commonly used).

Named entity recognition is an instance of a **span labeling** task, where the goal is to locate and classify spans in the input sequence.

Span Labeling – BIO Encoding

A possible approach to a span labeling task is to classify every sequence element using a specialized tag set. A common approach is to use the **BIO** encoding, which consists of

- `O`: *outside*, the given element is not part of any span;
- `B-PER`, `B-ORG`, `B-LOC`, …: *beginning*, the element is first in a new span;
- `I-PER`, `I-ORG`, `I-LOC`, …: *inside*, a continuation element of an existing span.

In a **valid** sequence, the `I-TYPE` **must follow** either `B-TYPE` or `I-TYPE`.

(Formally, the described scheme is IOB-2 format; there exists quite a few other possibilities like IOB-1, IEO, BILOU, …)

The described encoding can represent any set of continuous typed spans (when no spans overlap, i.e., a single element can belong to at most one span).

Span Labeling – BIO Encoding

However, when predicting each of the element tags independently, invalid sequences might be created.

- We can decide to ignore it and use heuristics capable of recovering the spans from invalid sequences of BIO tags.
- We can employ a decoding algorithm producing the most probable **valid sequence** of tags during prediction.
  - However, during training we do not consider the BIO tags validity.
- We might use a different loss enabling the model to consider only valid BIO tag sequences also during training.

Span Labeling – Decoding Algorithm

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ be an input sequence.

Our goal is to produce an output sequence $y_1, \ldots, y_N$, where each $y_t \in \mathcal{Y}$ with Y classes.

Assume we have a model predicting $p(y_t = k | \boldsymbol{X}; \boldsymbol{\theta})$, a probability that the t-th output element y_t is the class k.

However, only some sequences $\boldsymbol{y}$ are valid. We now make an assumption that the validity of a sequence depends only on the validity of **neighboring** output classes. In other words, if all neighboring pairs of output elements are valid, the whole sequence is.

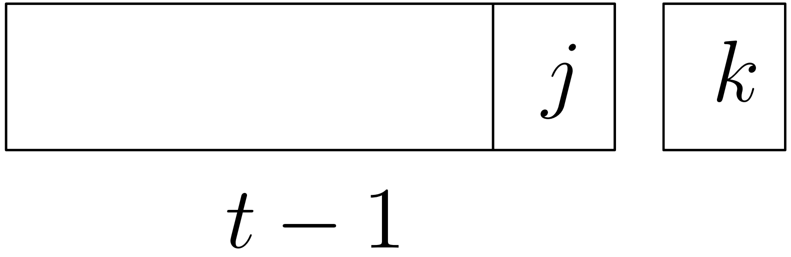
- The validity of neighboring pairs can be described by a transition matrix $\boldsymbol{A} \in \{0,1\}^{Y \times Y}$.
- Such an approach allows expressing the (in)validity of a BIO tag sequence.
  - However, the current formulation does not enforce conditions on the first and the last tag.

    If needed (for example to disallow `I-TYPE` as the first tag), we can add fixed y_0 and/or y_{N+1} imposing conditions on y_1 and/or y_N, respectively.

Span Labeling – Decoding Algorithm

Let us denote $\alpha_t(k)$ the log probability of the most probable output sequence of t elements with the last one being k.

We can compute $\alpha_t(k)$ efficiently using dynamic programming. The core idea is the following:



$$\alpha_t(k) = \log p(y_t = k | \boldsymbol{X}; \boldsymbol{\theta}) + \max\nolimits_{j,\ \text{such that } A_{j,k} \text{ is valid}} \alpha_{t-1}(j).$$

If we consider $\log A_{j,k}$ to be $-\infty$ when $A_{j,k} = 0$, we can rewrite the above as

$$\alpha_t(k) = \log p(y_t = k | \boldsymbol{X}; \boldsymbol{\theta}) + \max\nolimits_j \big(\alpha_{t-1}(j) + \log A_{j,k}\big).$$

The resulting algorithm is also called the **Viterbi algorithm**, and it is also a search for the path of maximum length in an acyclic graph.

Span Labeling – Decoding Algorithm

Inputs: Input sequence of length N, tag set with Y tags.

Inputs: Model computing $p(y_t = k|\boldsymbol{X};\boldsymbol{\theta})$, a probability that y_t should have the class k.

Inputs: Transition matrix $\boldsymbol{A} \in \mathbb{R}^{Y\times Y}$ indicating *valid* and *invalid* transitions.

Outputs: The most probable sequence $\boldsymbol{y}$ consisting of valid transitions only.

Time Complexity: $\mathcal{O}(N \cdot Y^2)$ in the worst case.

```
- For $t = 1, \ldots, N$:
    ○ For $k = 1, \ldots, Y$ :
        ■ $\alpha_t(k) \leftarrow \log p(y_t = k|\boldsymbol{X};\boldsymbol{\theta})$   *logits (unnormalized log probs) can also be used*
        ■ If $t > 1$:
            ■ $\beta_t(k) \leftarrow \arg\max_{j,\text{ such that } A_{j,k} \text{ is valid}} \alpha_{t-1}(j)$
            ■ $\alpha_t(k) \leftarrow \alpha_t(k) + \alpha_{t-1}\big(\beta_t(k)\big)$
```

- The most probable sequence has the log probability $\max \alpha_N$, and its elements can be recovered by traversing β from $t = N$ downto $t = 1$.

Span Labeling – Other Approaches

With deep learning models, constrained decoding is usually sufficient to deliver high performance.

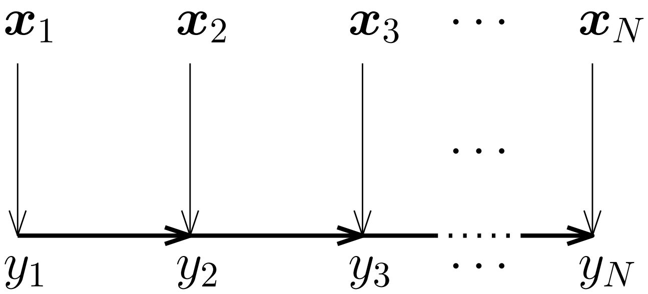
Historically, there have been also other approaches:

- **Maximum Entropy Markov Models**

  We might model the dependencies by explicitly conditioning on the previous label:

  $$P(y_i | \boldsymbol{X}, y_{i-1}).$$

  Then, each label is predicted by a softmax from a hidden state and a *previous label*.

  The decoding can still be performed by a dynamic programming algorithm.

Span Labeling – Other Approaches

- **Conditional Random Fields (CRF)**

  In the simplest variant, Linear-chain CRF, usually abbreviated only to CRF, can be considered an extension of softmax – instead of a sequence of independent softmaxes, it is a sentence-level softmax, with additional weights for neighboring sequence elements.

  We start by defining a score of a label sequence $\boldsymbol{y}$ as

  $$s(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{A}) = f(y_1|\boldsymbol{X}; \boldsymbol{\theta}) + \sum\nolimits_{i=2}^{N} \left(\boldsymbol{A}_{y_{i-1}, y_i} + f(y_i|\boldsymbol{X}; \boldsymbol{\theta})\right),$$

  and define the probability of a label sequence $\boldsymbol{y}$ using $\operatorname{softmax}$:

  $$p(\boldsymbol{y}|\boldsymbol{X}) = \operatorname{softmax}_{\boldsymbol{z} \in Y^N} \big(s(\boldsymbol{X}, \boldsymbol{z})\big)_{\boldsymbol{y}}.$$

  The probability $\log p(\boldsymbol{y}_\text{gold}|\boldsymbol{X})$ can be efficiently computed using dynamic programming in a differentiable way, so it can be used in NLL computation.

  For more details, see Lecture 8 of NPFL114 2022/23 slides.

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.

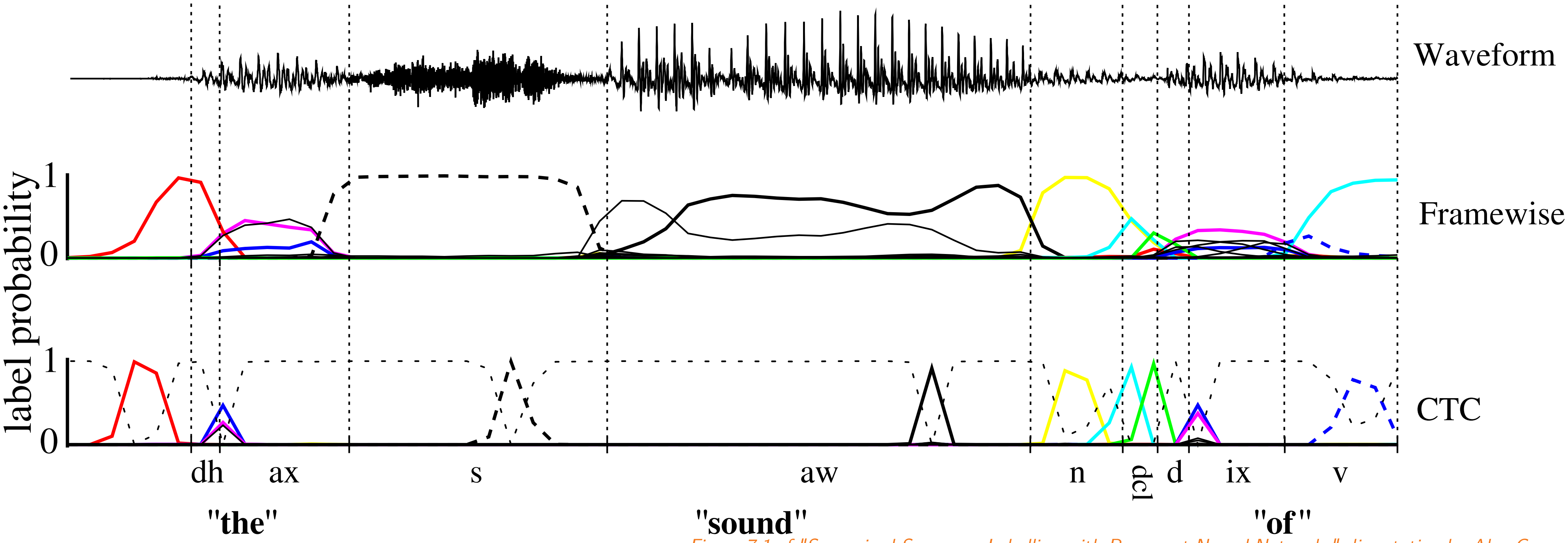


Figure 7.1 of "Supervised Sequence Labelling with Recurrent Neural Networks" dissertation by Alex Graves

Connectionist Temporal Classification

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

1. We collapse multiple neighboring occurrences of the same symbol into one.
2. We remove the blank –.

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

$$\alpha^t(s) \stackrel{\text{def}}{=} \sum_{\substack{\text{extended}\\\text{labelings }\boldsymbol\pi:\\\mathcal{B}(\boldsymbol\pi_{1:t})=\boldsymbol y_{1:s}}} \prod_{i=1}^t p_{\pi_i}^i .$$

You can have a look at https://distill.pub/2017/ctc/ for a nice and detailed description.

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

$$\alpha_-^t(s) \overset{\text{def}}{=} \sum_{\substack{\text{extended} \\ \text{labelings } \boldsymbol\pi: \\ \mathcal{B}(\boldsymbol\pi_{1:t})=\boldsymbol y_{1:s}, \pi_t=-}} \prod_{i=1}^t p_{\pi_i}^i$$

$$\alpha_*^t(s) \overset{\text{def}}{=} \sum_{\substack{\text{extended} \\ \text{labelings } \boldsymbol\pi: \\ \mathcal{B}(\boldsymbol\pi_{1:t})=\boldsymbol y_{1:s}, \pi_t\neq-}} \prod_{i=1}^t p_{\pi_i}^i$$

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

Connectionist Temporal Classification

Computation – Initialization

We initialize α^1 as follows:

- $\alpha_-^1(0) \leftarrow p_-^1$
- $\alpha_*^1(1) \leftarrow p_{y_1}^1$
- all other α^1 to zeros

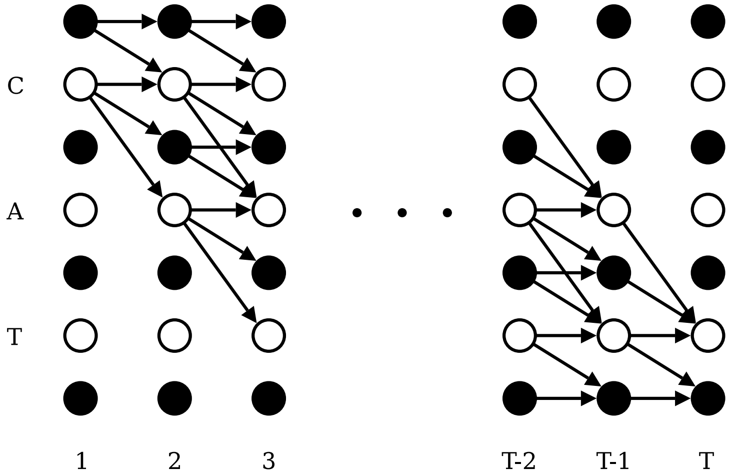


Figure 7.3 of "Supervised Sequence Labelling with Recurrent Neural Networks" dissertation by Alex Graves

Computation – Induction Step

We then proceed recurrently according to:

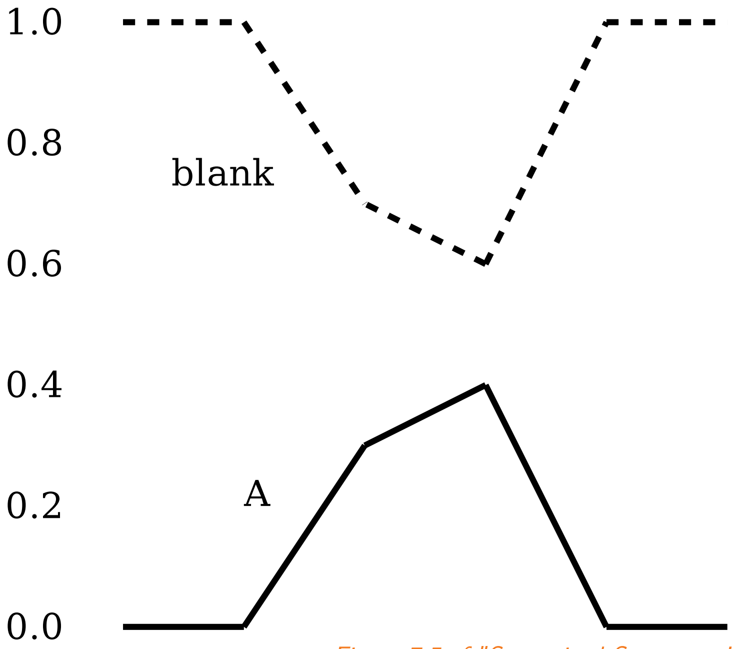
- $\alpha_-^t(s) \leftarrow p_-^t\left(\alpha_*^{t-1}(s) + \alpha_-^{t-1}(s)\right)$
- $\alpha_*^t(s) \leftarrow \begin{cases} p_{y_s}^t\left(\alpha_*^{t-1}(s) + \alpha_-^{t-1}(s-1) + \alpha_*^{t-1}(s-1)\right), \text{if } y_s \neq y_{s-1} \\ p_{y_s}^t\left(\alpha_*^{t-1}(s) + \alpha_-^{t-1}(s-1) + \cancel{\alpha_*^{t-1}(s-1)}\right),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

  We can write the update as $p_{y_s}^t\left(\alpha_*^{t-1}(s) + \alpha_-^{t-1}(s-1) + [y_s \neq y_{s-1}] \cdot \alpha_*^{t-1}(s-1)\right)$.

CTC Decoding

Unlike BIO-tag structured prediction, nobody knows how to perform CTC decoding optimally in polynomial time.

The key observation is that while an optimal extended labeling can be extended into an optimal labeling of a greater length, the same does not apply to a regular labeling. The problem is that regular labeling corresponds to many extended labelings, which are modified each in a different way during an extension of the regular labeling.



p(l=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 "Supervised Sequence Labelling with Recurrent Neural Networks" dissertation by Alex Graves

Beam Search

To perform a beam search, we keep k best **regular** (non-extended) labelings. Specifically, for each regular labeling $\boldsymbol{y}$ we keep both $\alpha^t_-(\boldsymbol{y})$ and $\alpha^t_*(\boldsymbol{y})$, which are probabilities of all (modulo beam search) extended labelings of length t which produce the regular labeling $\boldsymbol{y}$; we therefore keep k regular labelings with the highest $\alpha^t_-(\boldsymbol{y}) + \alpha^t_*(\boldsymbol{y})$.

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

- adding a *blank* symbol, i.e., contributing to $\alpha^{t+1}_-(\boldsymbol{y})$ both from $\alpha^t_-(\boldsymbol{y})$ and $\alpha^t_*(\boldsymbol{y})$;
- adding a non-blank symbol, i.e., contributing to $\alpha^{t+1}_*(\bullet)$ from $\alpha^t_-(\boldsymbol{y})$ and contributing to a possibly different $\alpha^{t+1}_*(\bullet)$ from $\alpha^t_*(\boldsymbol{y})$.

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.

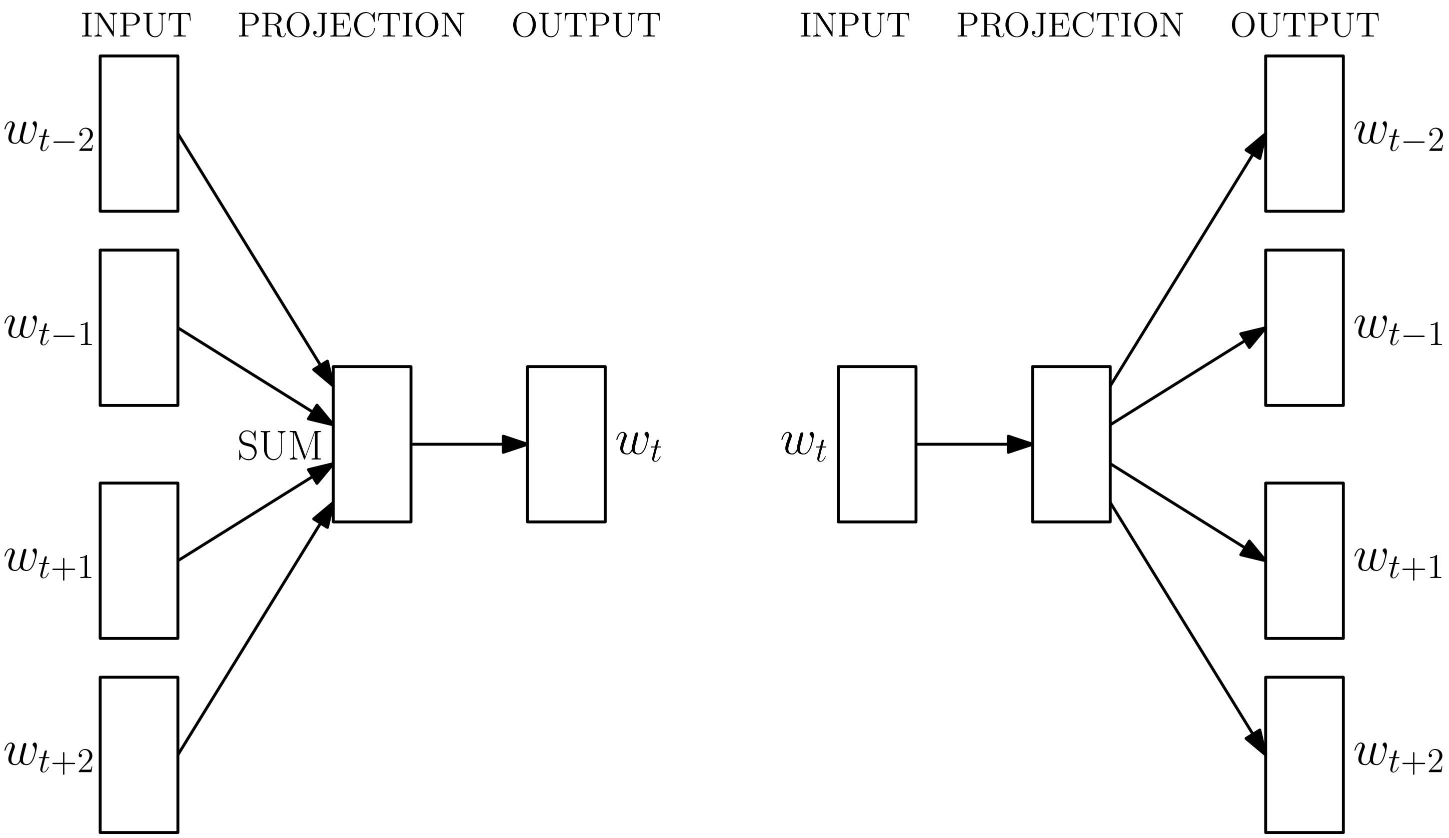
However, a method of precomputing word embeddings have been proposed, based on *distributional hypothesis*:

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

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

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

Word2Vec



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`.

Word2Vec

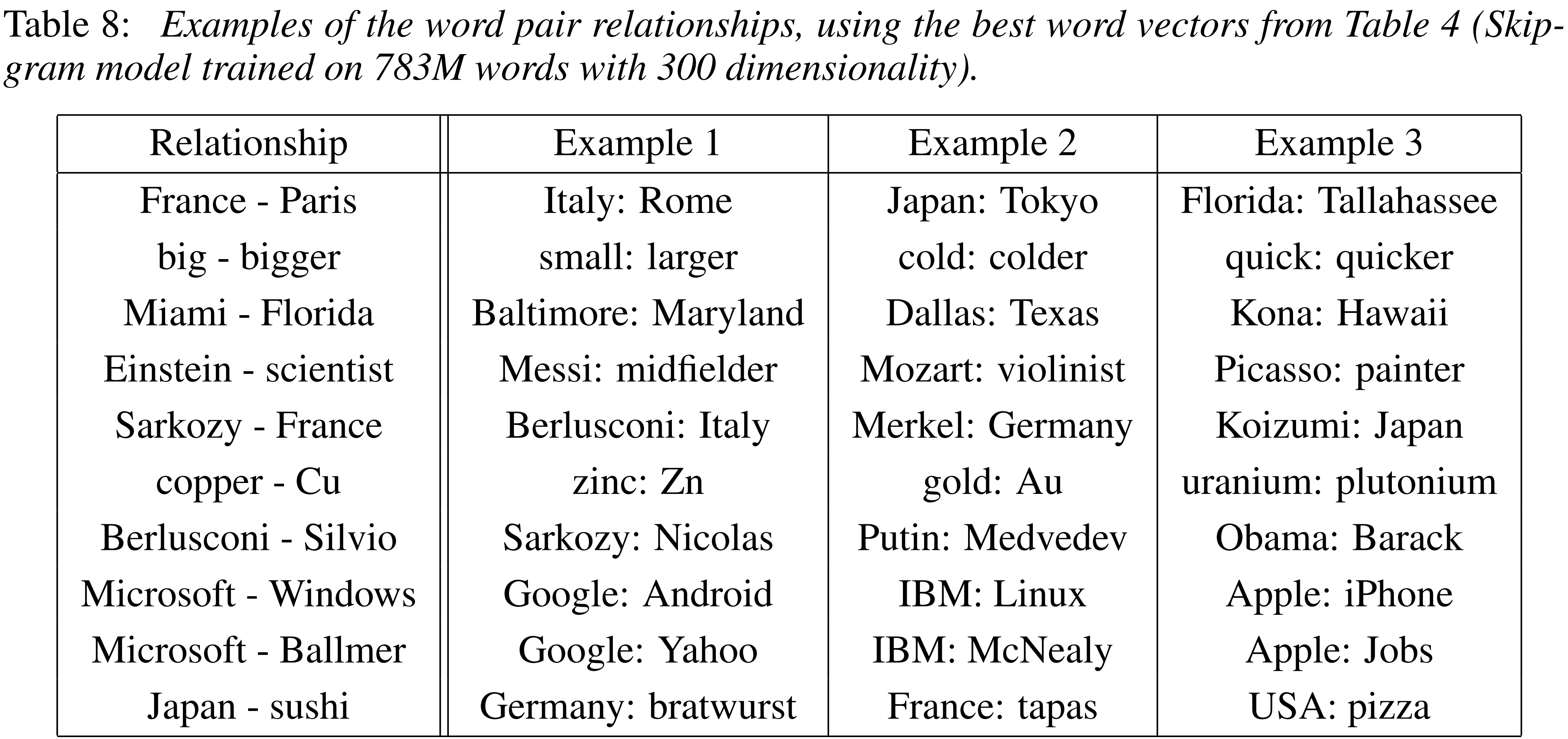
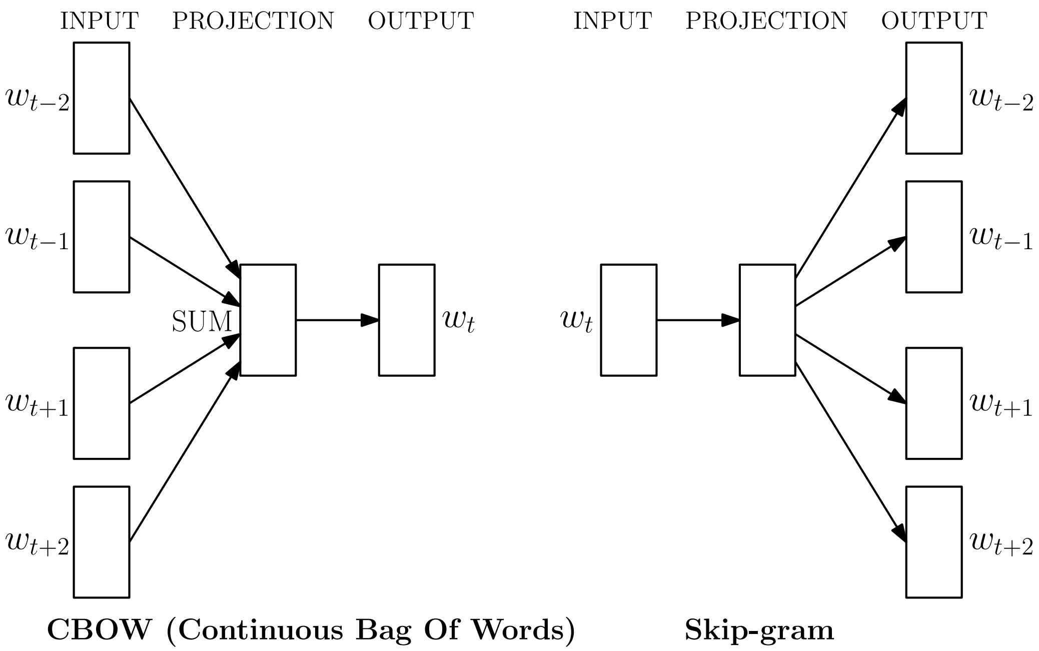


Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

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

Word2Vec – SkipGram Model



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

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

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

After training, the final embeddings are the rows of the $\boldsymbol{V}$ matrix.

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

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

Word2Vec – Negative Sampling

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

We could also only sample several *negative examples*. This gives rise to the following *negative sampling* objective (instead of just summing all the sigmoidal losses):

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

The usual value of negative samples k is 5, but it can be even 2 for extremely large corpora.

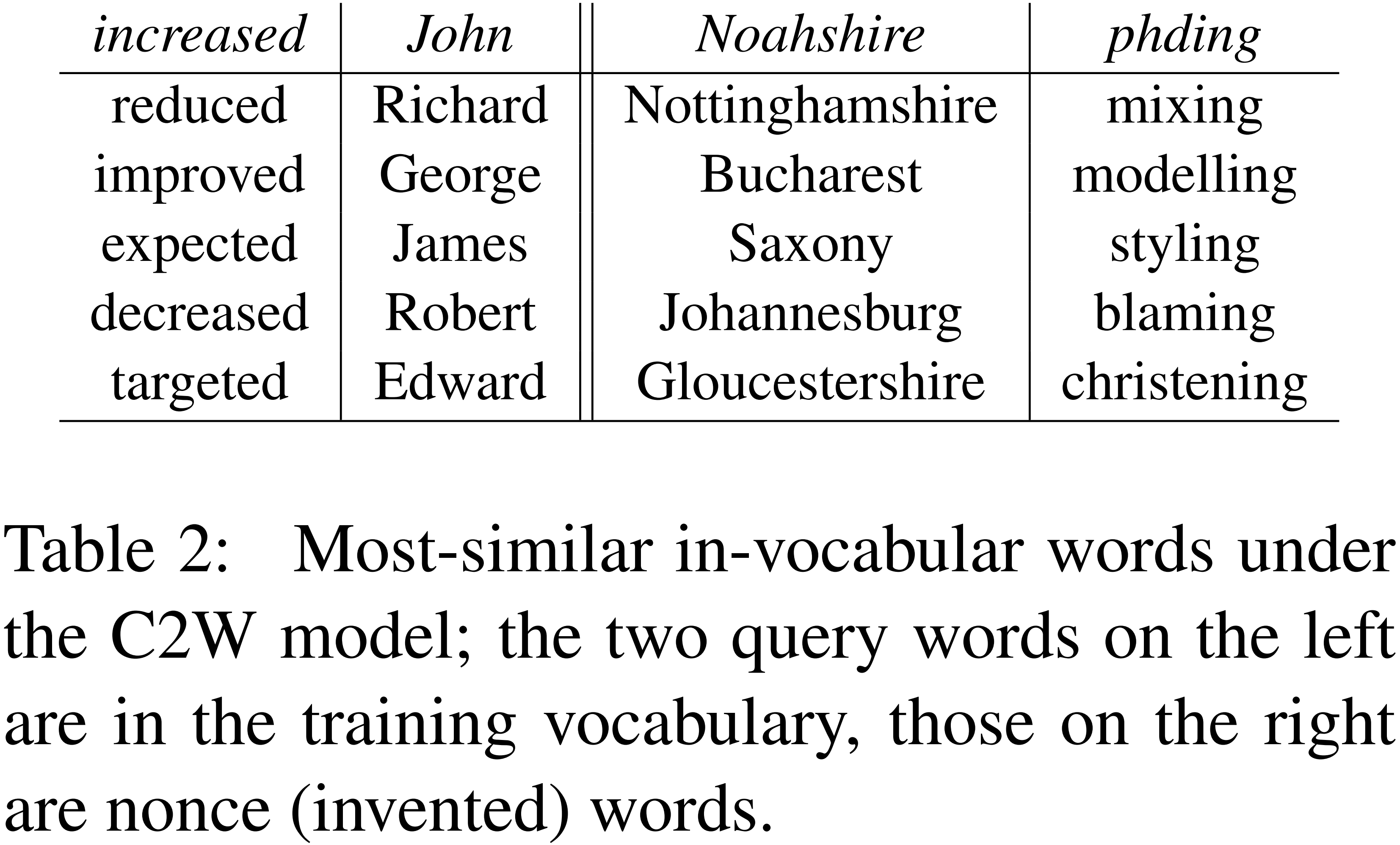
Each expectation in the loss is estimated using a single sample.

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

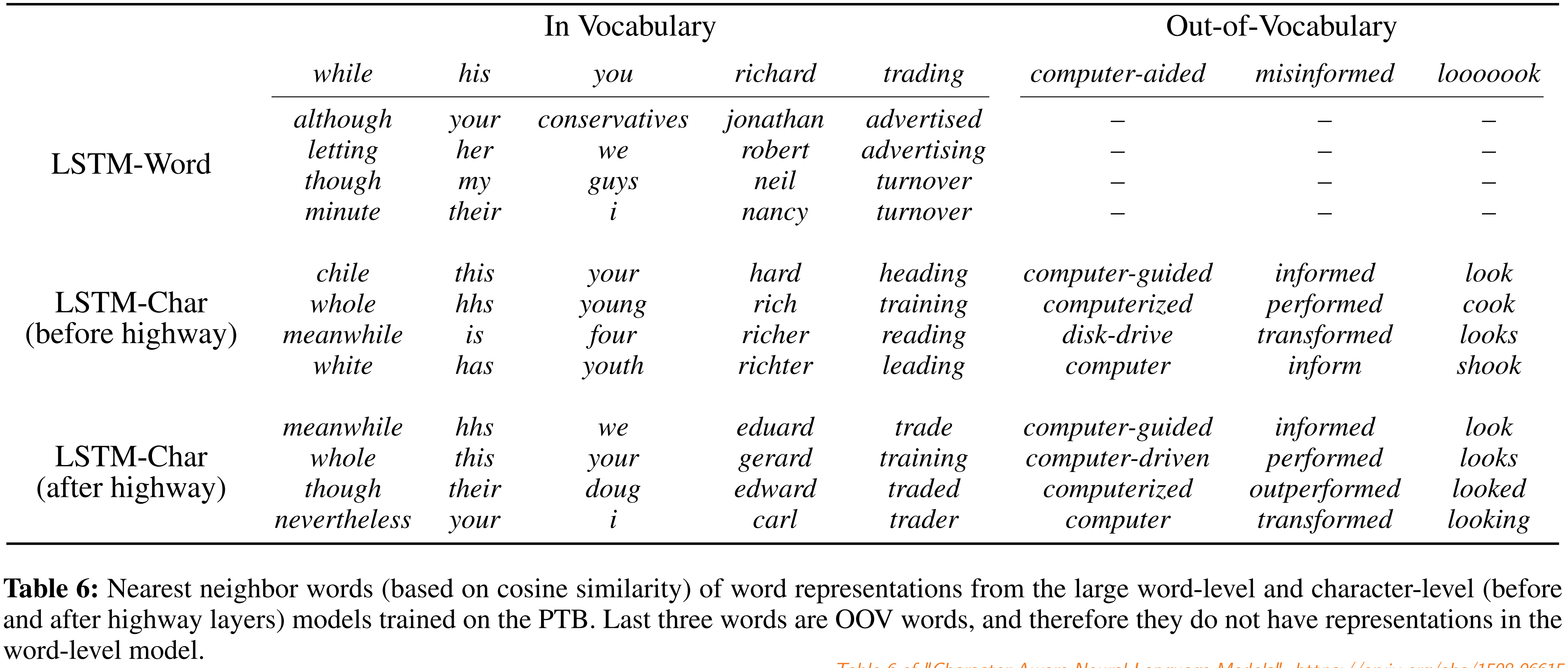


increased	*John*	*Noahshire*	*phding*
reduced	Richard	Nottinghamshire	mixing
improved	George	Bucharest	modelling
expected	James	Saxony	styling
decreased	Robert	Johannesburg	blaming
targeted	Edward	Gloucestershire	christening

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.

Table 2 of "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 "Character-Aware Neural Language Models", https://arxiv.org/abs/1508.06615

Character N-grams

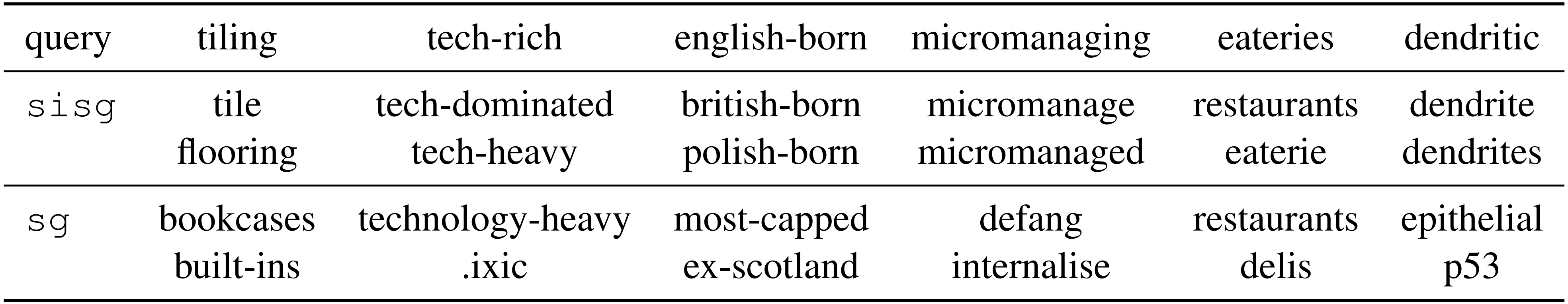
Another simple idea appeared simultaneously in three nearly simultaneous publications as Charagram, Subword Information or SubGram.

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

Charagram WEs

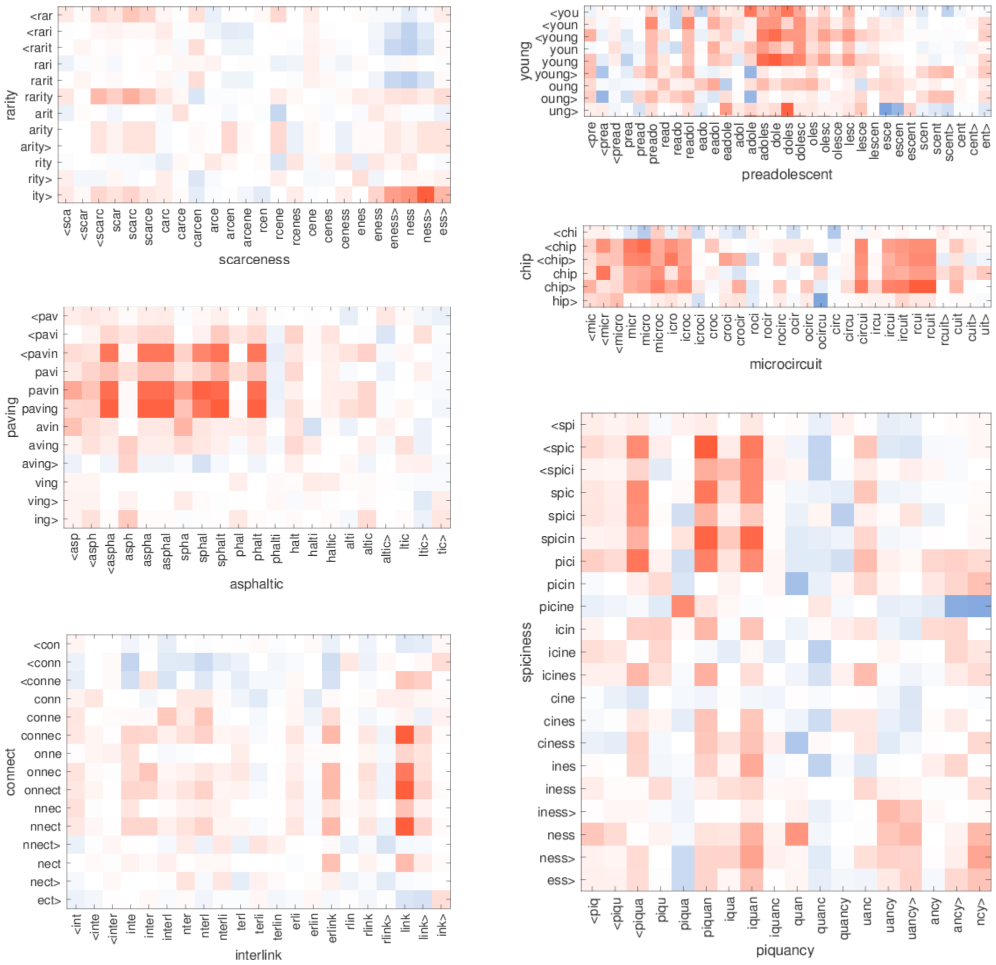


Figure 2: Illustration of the similarity between character n-grams in out-of-vocabulary words. For each pair, only one word is OOV, and is shown on the x axis. Red indicates positive cosine, while blue negative.

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

FastText

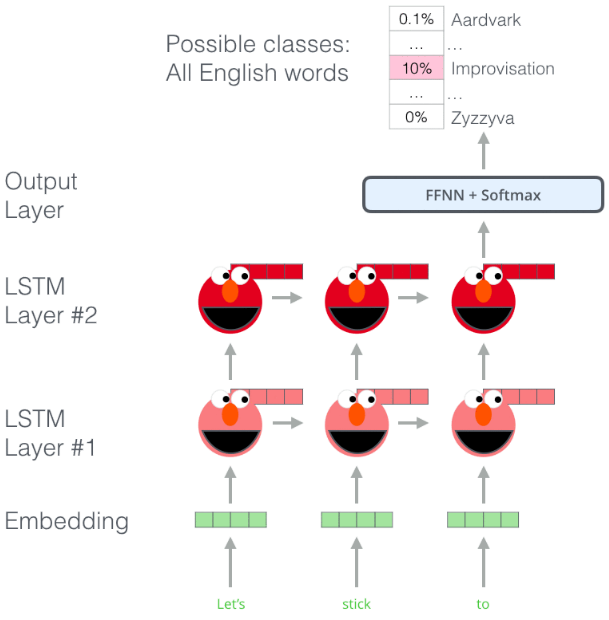
The word2vec enriched with subword embeddings is implemented in publicly available `fastText` library https://fasttext.cc/.

Pre-trained embeddings for 157 languages (including Czech) trained on Wikipedia and CommonCrawl are also available at https://fasttext.cc/docs/en/crawl-vectors.html.

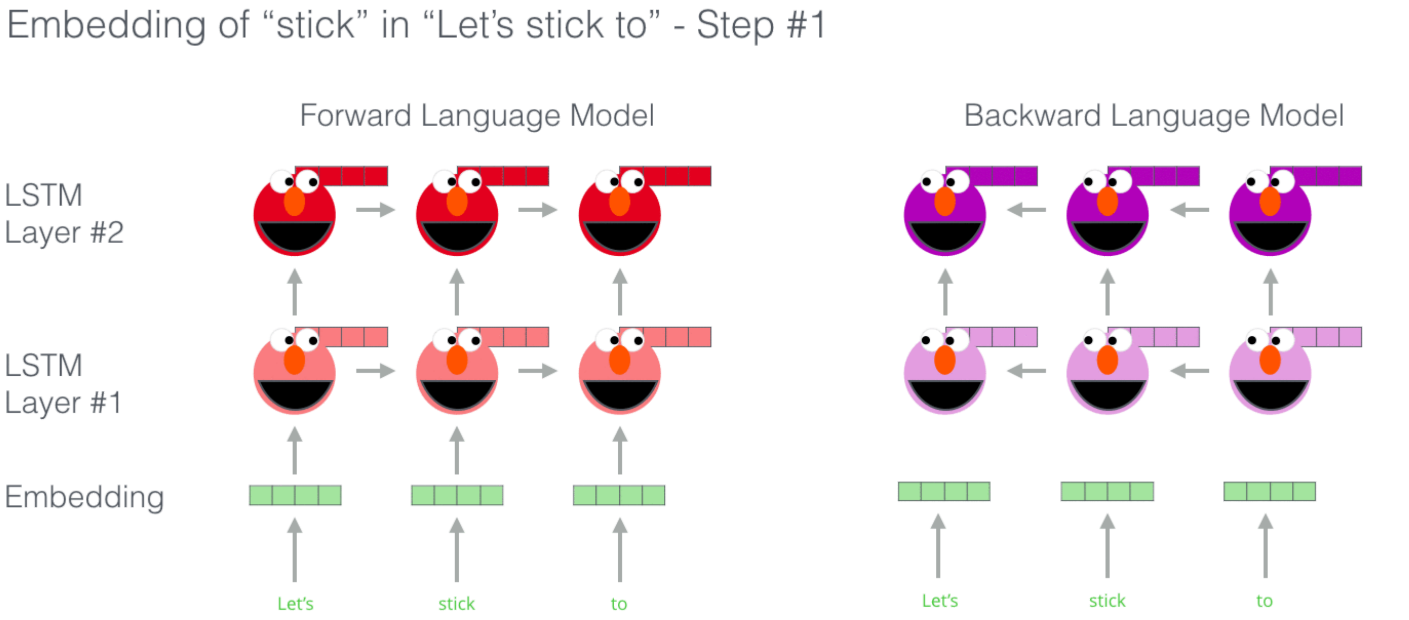
ELMo

At the end of 2017, a new type of *deep contextualized* word representations was proposed by Peters et al., called ELMo, **E**mbeddings from **L**anguage **Mo**dels.

The ELMo embeddings were based on a two-layer pre-trained LSTM language model, where a language model predicts following word based on a sentence prefix. Specifically, two such models were used, one for the forward direction and the other one for the backward direction.



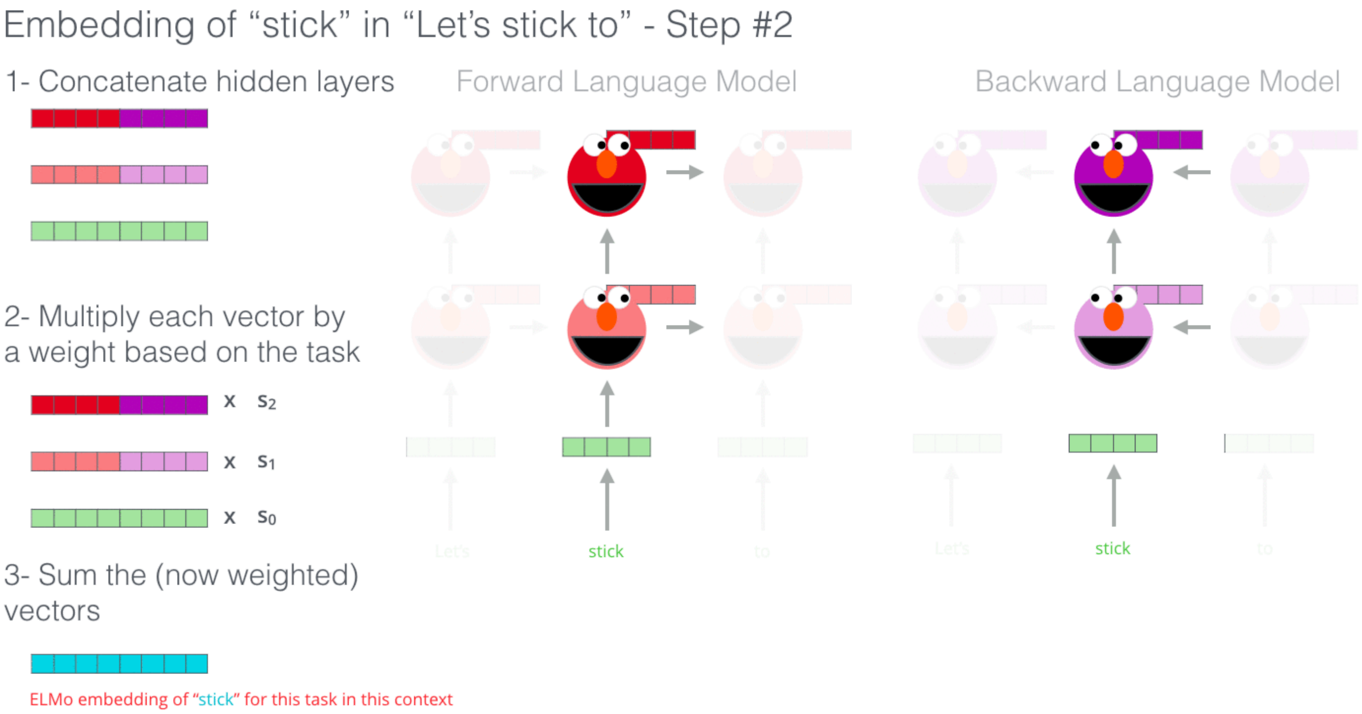
http://jalammar.github.io/images/Bert-language-modeling.png



http://jalammar.github.io/images/elmo-forward-backward-language-model-embedding.png

ELMo

To compute an embedding of a word in a sentence, the concatenation of the two language model's hidden states is used.



Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

3- Sum the (now weighted) vectors

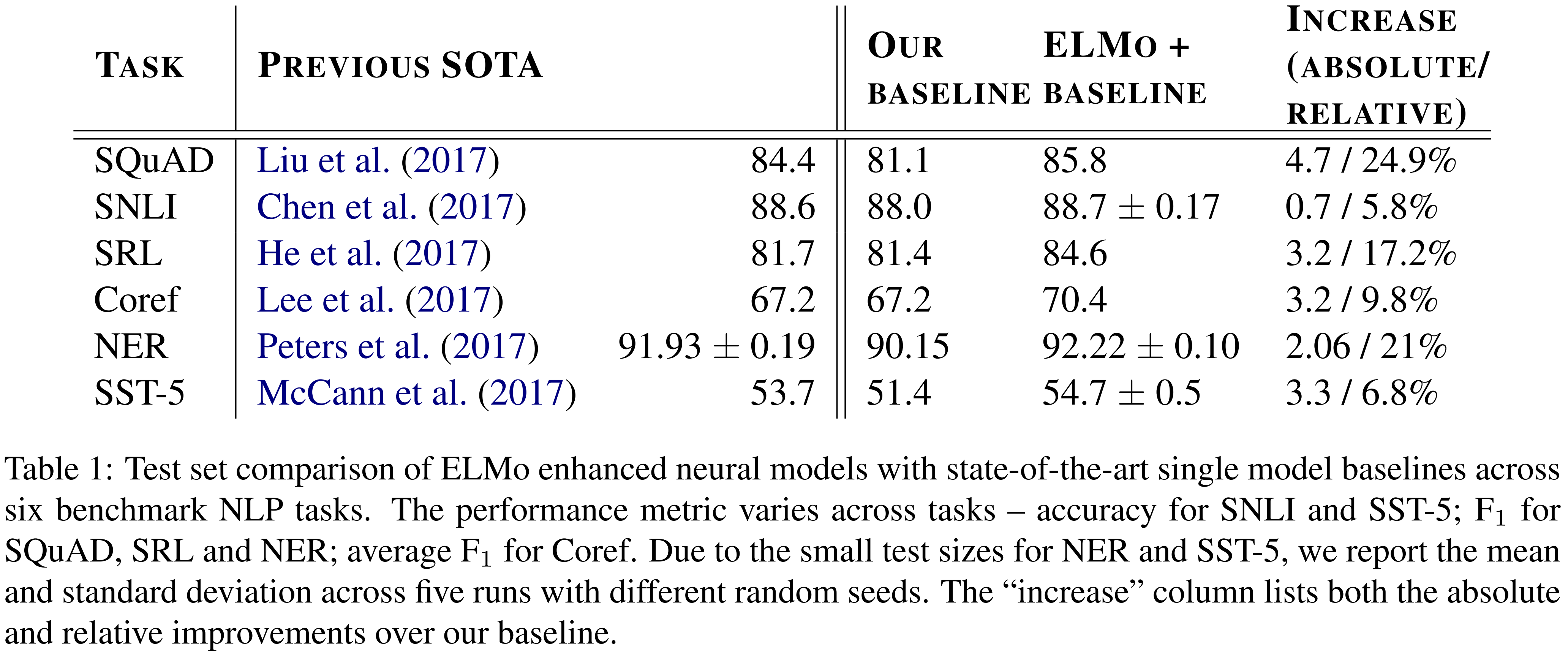
ELMo embedding of “stick” for this task in this context

http://jalammar.github.io/images/elmo-embedding.png

To be exact, the authors propose to take a (trainable) weighted combination of the input embeddings and outputs on the first and second LSTM layers.

ELMo Results

Pre-trained ELMo embeddings substantially improved several NLP tasks.



Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

Table 1 of "Deep contextualized word representations", https://arxiv.org/abs/1802.05365