CRF, CTC, Word2Vec

Milan Straka

April 4, 2022

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

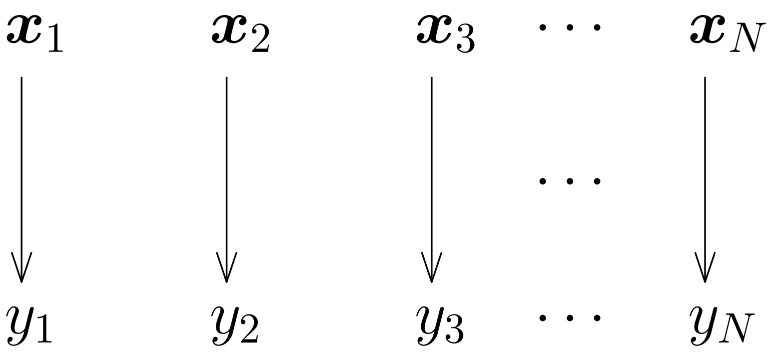
Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Structured Prediction

Structured Prediction

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

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



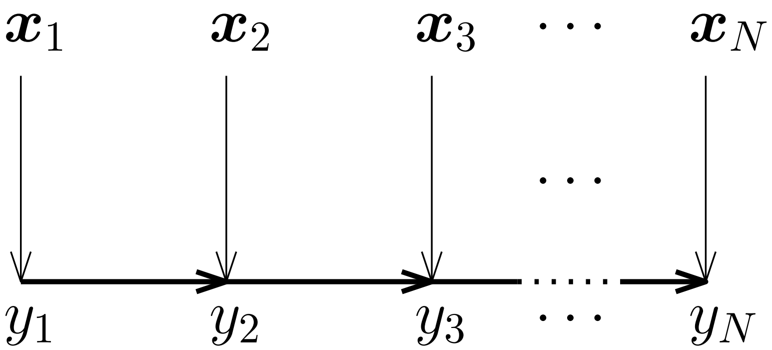
However, there may be dependencies among the y_i themselves, which is difficult to capture by independent element classification.

Maximum Entropy Markov Models

We might model the dependencies by assuming that the output sequence is a Markov chain, and model it as

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

Each label would be predicted by a softmax from the hidden state and the *previous label*.



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

However, MEMMs suffer from a so-called **label bias** problem. Because the probability is factorized, each $P(y_i | \boldsymbol{X}, y_{i-1})$ is a distribution and **must sum to one**.

Imagine there was a label error during prediction. In the next step, the model might "realize" that the previous label has very low probability of being followed by any label – however, it cannot express this by setting the probability of all following labels to a low value, it has to "conserve the mass".

Conditional Random Fields

Let $G = (V, E)$ be a graph such that $\boldsymbol{y}$ is indexed by vertices of G. Then $(\boldsymbol{X}, \boldsymbol{y})$ is a **conditional random field**, if the random variables $\boldsymbol{y}$ conditioned on $\boldsymbol{X}$ obey the Markov property with respect to the graph, i.e.,

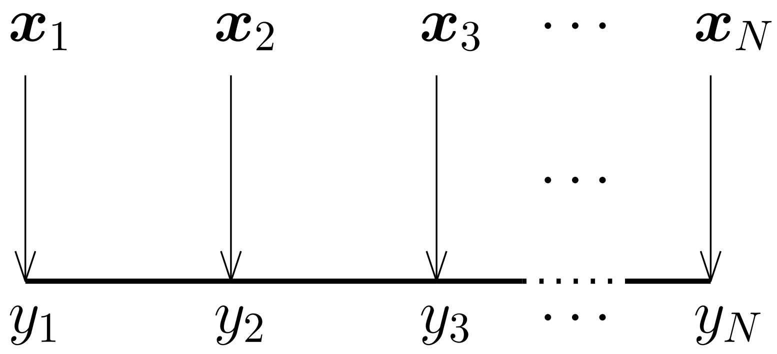
$$P\big(y_i|\boldsymbol{X}, \{y_j \,|\, \forall j \neq i\}\big) = P\big(y_i|\boldsymbol{X}, \{y_j \,|\, \forall j : (i, j) \in E\}\big).$$

By a *fundamental theorem of random fields (the Hammersley–Clifford theorem)*, the density of a conditional random field can be factorized over the *cliques* (complete subgraphs) of the graph G:

$$P(\boldsymbol{y}|\boldsymbol{X}) = \prod_{\text{clique } C \text{ of } G} P(\boldsymbol{y}_C|\boldsymbol{X}).$$

Linear-Chain Conditional Random Fields (CRF)

Most often, we assume that dependencies of $\boldsymbol{y}$, conditioned on $\boldsymbol{X}$, form a chain.



Then, the cliques are *nodes* and *edges*, and we can factorize the probability as:

$$P(\boldsymbol{y}|\boldsymbol{X}) \propto \exp\left(\sum_{i=1}^N \log P(y_i|\boldsymbol{X}) + \sum_{i=2}^N \log P(y_i, y_{i-1})\right).$$

Linear-Chain Conditional Random Fields (CRF)

Linear-chain Conditional Random Field, usually abbreviated only to CRF, acts as an output layer. It 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

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

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

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

For cross-entropy (and also to avoid underflow), we need a logarithm of the probability:

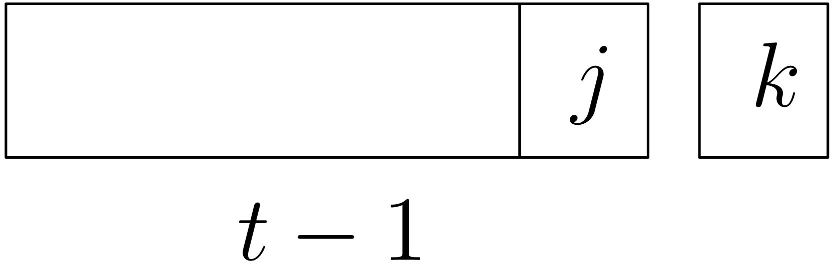
$$\log p(\boldsymbol{y} | \boldsymbol{X}) = s(\boldsymbol{X}, \boldsymbol{y}) - \operatorname{logsumexp}_{\boldsymbol{z} \in Y^N} \big(s(\boldsymbol{X}, \boldsymbol{z})\big), \text{ where}$$
$$\operatorname{logsumexp}_x \big(f(x)\big) = \log\big(\sum\nolimits_x e^{f(x)}\big).$$

Linear-Chain Conditional Random Fields (CRF)

Computation

We can compute $p(\boldsymbol{y}|\boldsymbol{X})$ efficiently using dynamic programming. We denote $\alpha_t(k)$ the logarithmic probability of all t-element sequences with the last label y being k.

The core idea is the following:



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

For efficient implementation, we use the fact that

$$\ln(a+b) = \ln a + \ln(1 + e^{\ln b - \ln a}),\ \text{so}$$
$$\operatorname{logsumexp}_x\big(f(x)\big) = \max_x\big(f(x)\big) + \log\big(\textstyle\sum_x e^{f(x) - \max_x(f(x))}\big).$$

Conditional Random Fields (CRF)

Inputs: Network computing $f_{\boldsymbol\theta}(y_t = k | \boldsymbol X)$, which is a logit (unnormalized log-probability) of output sequence label being k at time t.
Inputs: Transition matrix $\boldsymbol A \in \mathbb{R}^{Y \times Y}$.
Inputs: Input sequence $\boldsymbol X$ of length N, gold labeling $\boldsymbol g \in Y^N$.
Outputs: Value of $\log p(\boldsymbol g | \boldsymbol X)$.
Time Complexity: $\mathcal{O}(N \cdot Y^2)$.

- For $t = 1, \ldots, N$:
  - For $k = 1, \ldots, Y$:
    - $\alpha_t(k) \leftarrow f_{\boldsymbol\theta}(y_t = k | \boldsymbol X)$
    - If $t > 1$:
      - $\alpha_t(k) \leftarrow \alpha_t(k) + \operatorname{logsumexp}\left(\alpha_{t-1}(j) + \boldsymbol A_{j,k} \,\middle|\, j = 1, \ldots, Y\right)$
- Return $\sum_{t=1}^N f_{\boldsymbol\theta}(y_t = g_t | \boldsymbol X) + \sum_{t=2}^N \boldsymbol A_{g_{t-1}, g_t} - \operatorname{logsumexp}_{k=1}^Y(\alpha_N(k))$

Decoding

We can perform decoding optimally, by using the same algorithm, only replacing $\operatorname{logsumexp}$ with $\max$, and tracking where the maximum was attained.

Applications

The CRF output layer is useful for **span labeling** tasks, like

- named entity recognition,
- dialog slot filling.

It can be also useful for image segmentation.

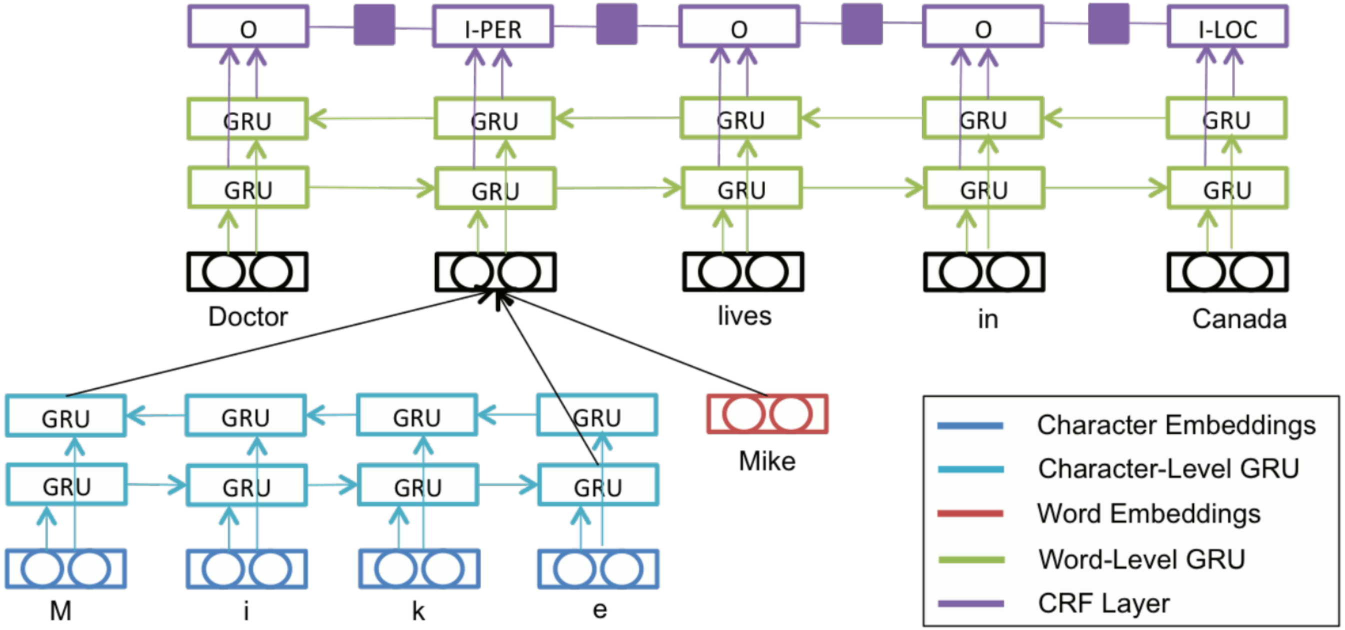


Figure 1 of "Multi-Task Cross-Lingual Sequence Tagging from Scratch", https://arxiv.org/abs/1603.06270

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.

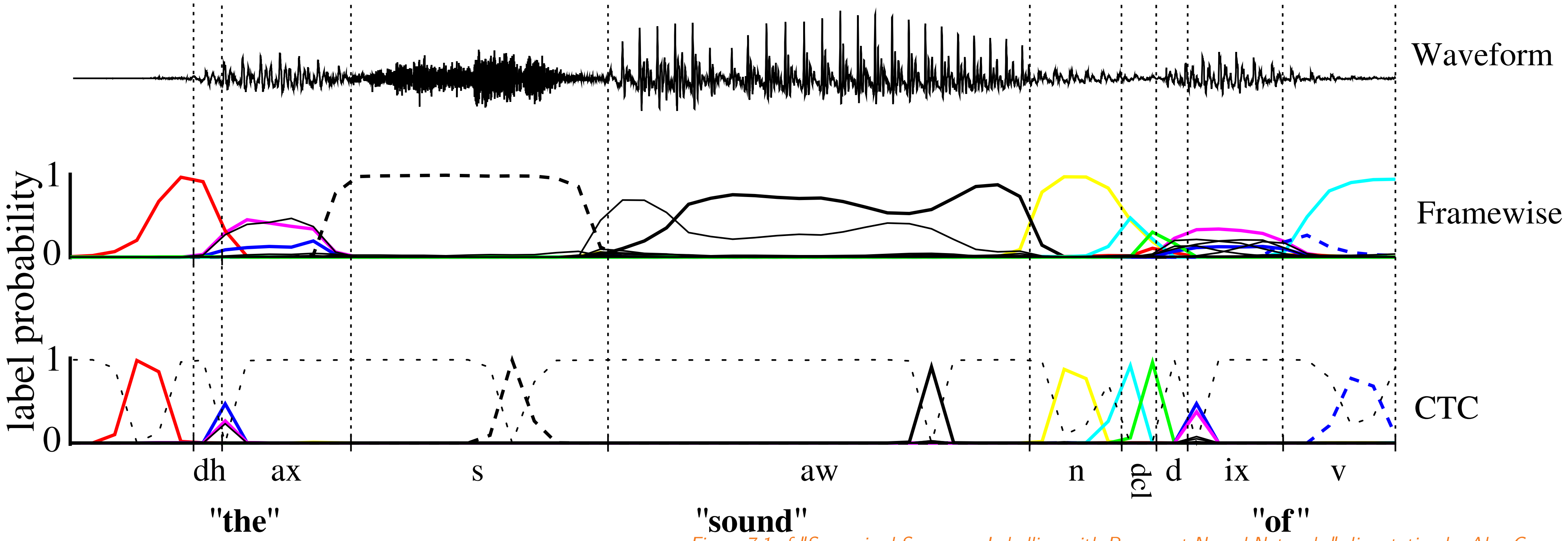


Figure 7.1 of "Supervised Sequence Labelling with Recurrent Neural Networks" dissertation by Alex Graves

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**. 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

$$\alpha^t(s) \stackrel{\text{def}}{=} \sum_{\substack{\text{extended} \\ \text{labelings } \boldsymbol\pi: \\ \mathcal{B}(\boldsymbol\pi_{1:t}) = \boldsymbol y_{1:s}}} \prod_{t'=1}^t p_{\boldsymbol\pi_{t'}}^{t'}.$$

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

$$\alpha_-^t(s) \stackrel{\text{def}}{=} \sum_{\substack{\text{extended}\\ \text{labelings } \boldsymbol\pi:\\ \mathcal{B}(\boldsymbol\pi_{1:t})=\boldsymbol y_{1:s}, \pi_t=-}} \prod_{t'=1}^{t} p_{\boldsymbol\pi_{t'}}^{t'}$$

$$\alpha_*^t(s) \stackrel{\text{def}}{=} \sum_{\substack{\text{extended}\\ \text{labelings } \boldsymbol\pi:\\ \mathcal{B}(\boldsymbol\pi_{1:t})=\boldsymbol y_{1:s}, \pi_t\neq-}} \prod_{t'=1}^{t} p_{\boldsymbol\pi_{t'}}^{t'}$$

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

Computation – Initialization

We initialize α as follows:

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

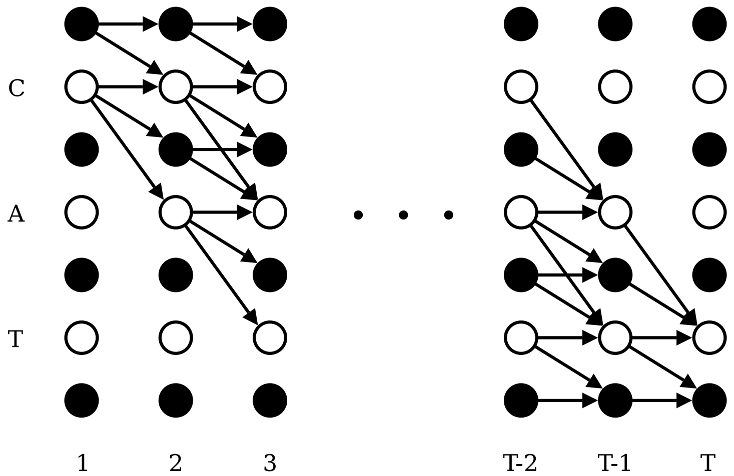


Figure 7.3 of "Supervised Sequence Labelling with Recurrent Neural Networks" dissertation by Alex Graves

Computation – Induction Step

We then proceed recurrently according to:

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

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 a regular (non-extended) 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.

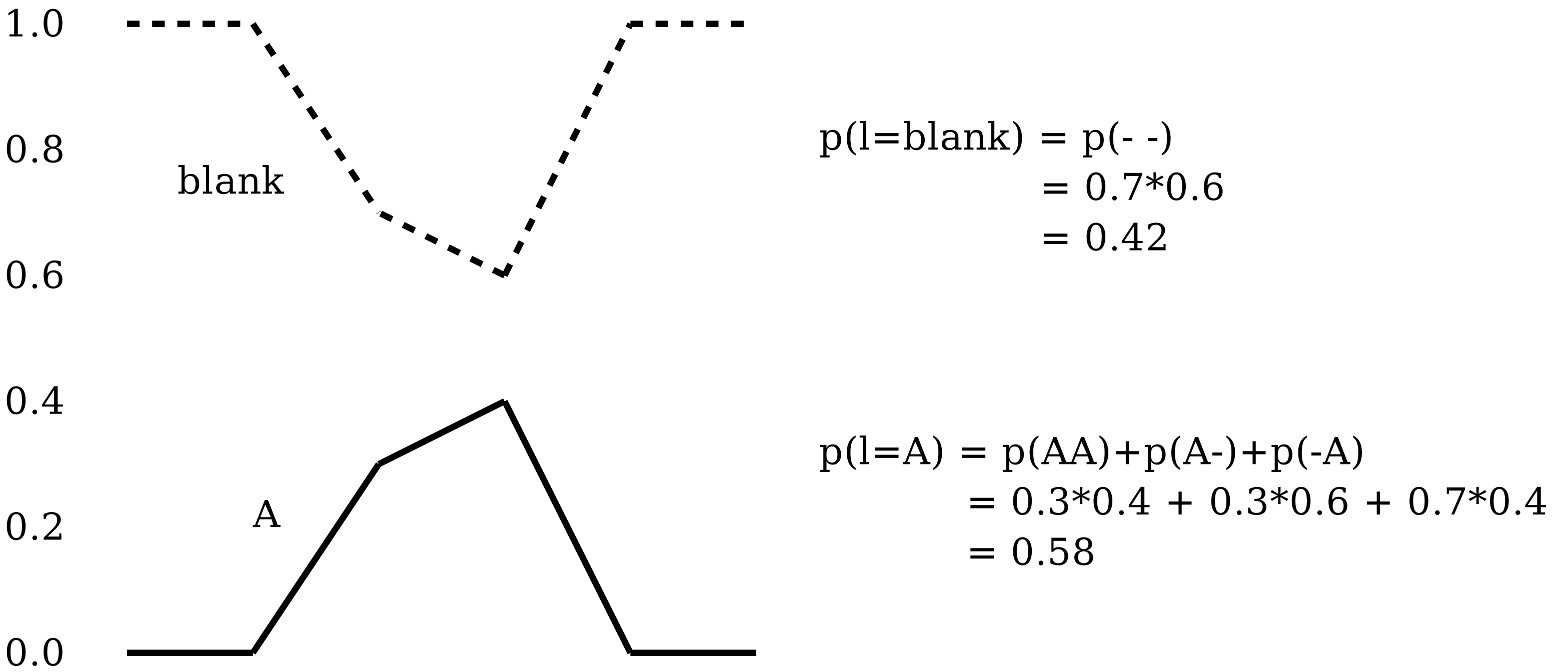


Figure 7.5 of "Supervised Sequence Labelling with Recurrent Neural Networks" dissertation by Alex Graves

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 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 $\alpha^{t+1}_*(\cdot)$ from $\alpha^t_-(\boldsymbol y)$ and to possibly different $\alpha^{t+1}_*(\cdot)$ 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

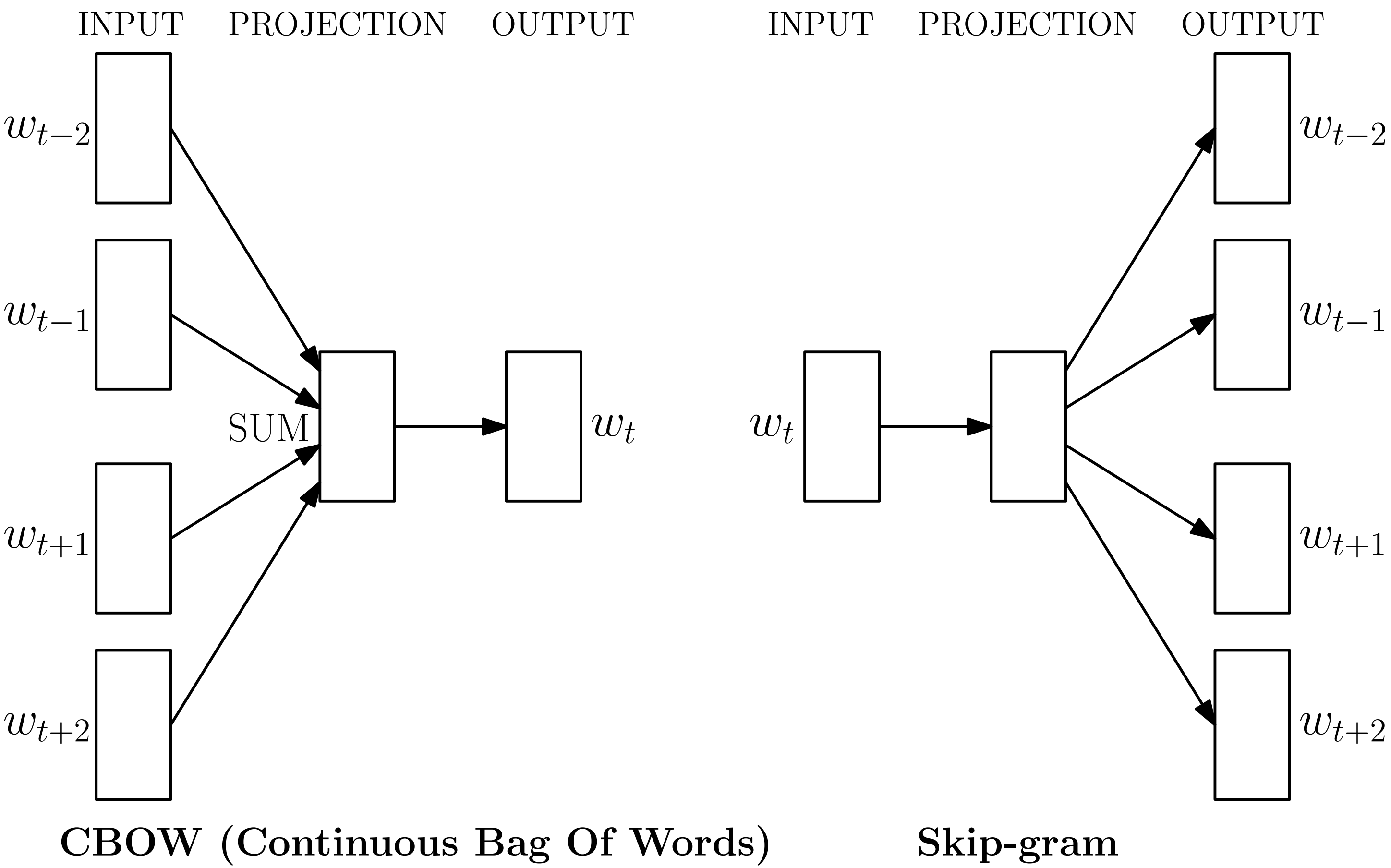
The embeddings can be trained for each task separately.

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.*



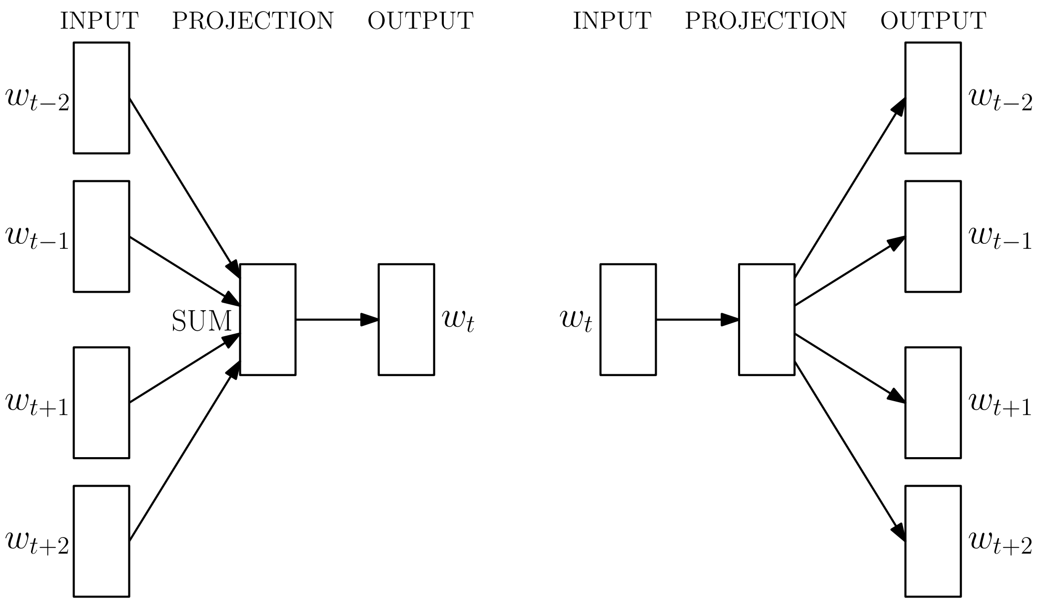
Mikolov et al. (2013) proposed two very simple architectures for precomputing word embeddings, together with a C multi-threaded implementation `word2vec`.

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 "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

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

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):

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

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).

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

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

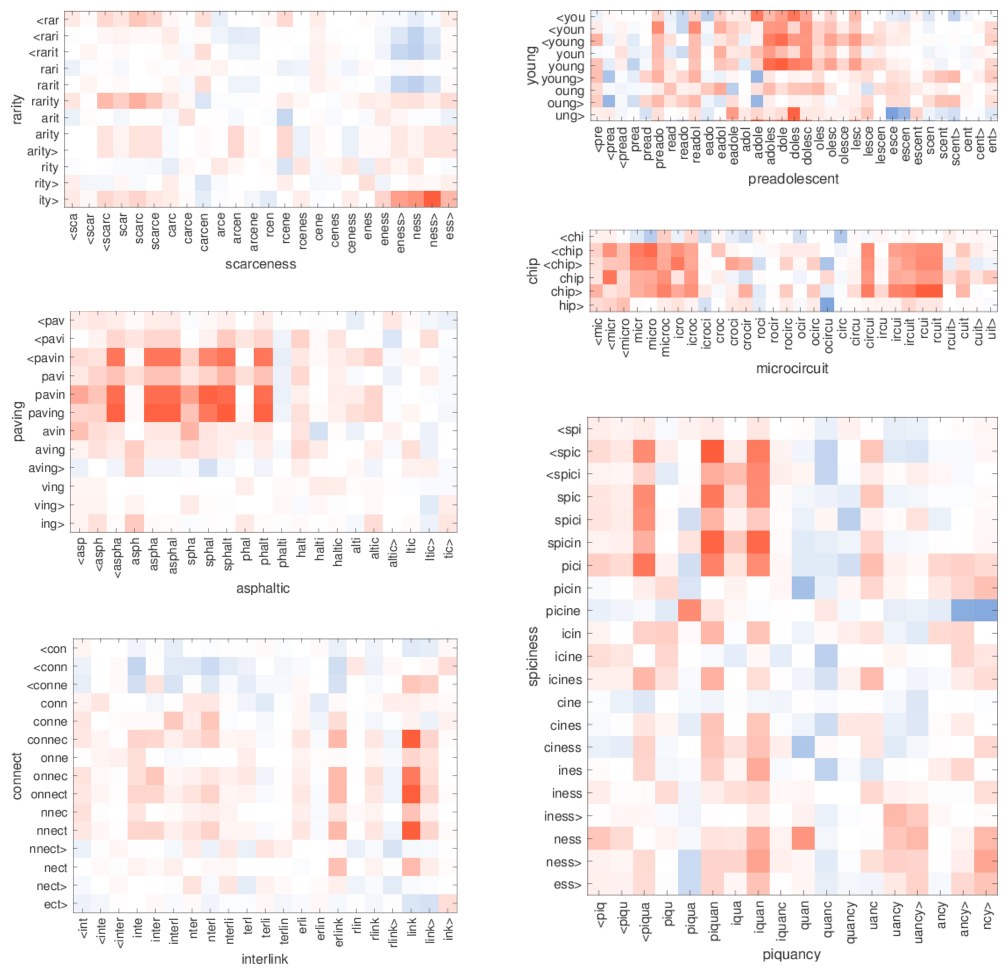


Figure 2: Illustration of the similarity between character n-grams in out-of-vocabulary words. For each pair, only one word is OOV, and is shown on the x axis. Red indicates positive cosine, while blue negative.

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.