Chapter 5

Lexical-level and Morphological-level Extensions of Word2Vec

Keywords: glove, lexvec, swivel, morpheme, inflectional morphology, derivational morphology, n-grams, character embeddings, fasttext.

5.1 Lexical-level extensions

This chapter will be briefly overviewing models came after Word2Vec that exploit the same idea of representing words with vectors.

5.1.1 Shifted PPMI

While the PMI matrix emerges from SGNS with $k = 1$, it was shown that different values of k can substantially improve the resulting embedding. With $k > 1$, the association metric in the implicitly factorized matrix is $PMI_{w,c} - \log k$. This suggests the use of **Shifted PPMI (SPPMI)**, a novel association metric which, to the best of our knowledge, was not explored in the NLP and word-similarity communities:

$$SPPMI^k_{w,c} = \max(PMI_{w,c} - \log k, 0) = (PMI_{w,c} - \log k)^+.$$

As with SGNS, certain values of k can improve the performance on different tasks.

5.1.2 GloVe

The **Global Vector (GloVe)** model proposed in Pennington et al. (2014) aims to combine the count-based matrix factorization and the context-based skip-gram model together.

We all know the counts and co-occurrences can reveal the meanings of words. We will define the co-occurrence probability as:

$$P(w_k \mid w_i) = \frac{\#(w_i, w_k)}{\#(w_i)}.$$

Say, we have two words, w_i = "ice" and w_j = "steam". The third word w_k = "solid" is related to "ice" but not "steam", and thus we expect $P(w_k \mid w_i)$ to be much larger than $P(w_k \mid w_j)$ and therefore $P(w_k \mid w_i)/P(w_k \mid w_j)$ to be very large. If the third word w_k = "water" is related to both or w_k = "fashion" is unrelated to either of them, the equation above is expected to be close to one.

The intuition here is that the word meanings are captured by the ratios of co-occurrence probabilities rather than the probabilities themselves. The global vector models the relationship between two words regarding to the third context word as:

$$F(w_i, w_j, w_k) = \frac{P(w_k \mid w_i)}{P(w_k \mid w_j)}.$$

Further, since the goal is to learn meaningful word vectors, F is designed to be a function of the linear difference between two words w_i and w_j. We denote vectors for "central" words as $\boldsymbol{u}$ (with indices), and vectors for context words as $\boldsymbol{v}$:

$$F(w_i, w_j, w_k) = F((\boldsymbol{u}_{w_i} - \boldsymbol{u}_{w_j})^\top \boldsymbol{v}_{w_k}).$$

With the consideration of F being symmetric between target words and context words, the final solution is to model F as an exponential function. And we obtain the following equations:

$$F(\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k}) = \exp(\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k}) = P(w_k \mid w_i),$$

$$F((\boldsymbol{u}_{w_i} - \boldsymbol{u}_{w_j})^\top \boldsymbol{v}_{w_k}) = \exp((\boldsymbol{u}_{w_i} - \boldsymbol{u}_{w_j})^\top \boldsymbol{v}_{w_k}) = \frac{\exp(\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k})}{\exp(\boldsymbol{u}_{w_j}^\top \boldsymbol{v}_{w_k})} = \frac{P(w_k \mid w_i)}{P(w_k \mid w_j)}.$$

Finally,

$$\boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k} = \log P(w_k \mid w_i) = \log \frac{\#(w_i, w_k)}{\#(w_i)} = \log \#(w_i, w_k) - \log \#(w_i).$$

Since the second term $-\log \#(w_i)$ is independent of k, we can add bias term $b_{w_i}^u$ for w_i to capture $-\log \#(w_i)$. To keep the symmetric form, we also add in a bias $b_{w_k}^v$ for w_k. After that, we obtain

$$\log \#(w_i, w_k) = \boldsymbol{u}_{w_i}^\top \boldsymbol{v}_{w_k} + b_{w_i}^u + b_{w_k}^v.$$

The loss function for the GloVe model is designed to preserve the above formula by minimizing the sum of the squared errors

$$L = \sum_{w \in \mathcal{W}} \sum_{c \in \mathcal{W}} f\left(\#(w, c)\right) \left(\boldsymbol{u}_w^\top \boldsymbol{v}_c + b_w^u + b_c^v - \log \#(w, c)\right)^2.$$

The weighting schema $f(x)$ is a function of the co-occurrence of words w (as a central word) and c (as a context word) and it is an adjustable model configuration. It should be close to zero as $x \to 0$; should be non-decreasing as higher co-occurrence should have more impact; should saturate when x become extremely large. The paper proposed the following weighting function:

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^\alpha, & \text{if } x < x_{\max}, \\ 1, & \text{otherwise} \end{cases}$$

with optimal values $\alpha = 0.75$ and $x_{\max} = 100$.

5.1.3 LexVec

Just like GloVe, **LexVec** (Salle et al. (2016)) also tries to factorize PPMI matrices, emerging characteristics of both count-based and prediction-based models. Unlike GloVe, it penalizes errors of frequent co-occurrences more heavily, while still treating negative co-occurrences.

Moreover, given that using PPMI results in better performance than PMI on semantic tasks, the authors propose keeping the SGNS weighting scheme by using window sampling and negative sampling, but explicitly factorizing the PPMI matrix rather than implicitly factorizing the shifted PMI matrix. The LexVec loss function has two terms:

$$L^{WC}(w,c) = \frac{1}{2}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PPMI^*_{w,c}\right)^2,$$
$$L^{W}(w) = \frac{1}{2}\sum_{j=1}^{k} \mathbb{E}_{c_{n,j}\sim P_n}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_{c_{n,j}} - PPMI^*_{w,c_{n,j}}\right)^2$$

where $PPMI^*$ is an improved (e.g. by using context-distribution smoothing (Levy et al., 2015) or subsampling the corpus (Mikolov et al., 2013b)) PPMI matrix.

These terms could be minimized using two alternative approaches. The first one is Mini-Batch. This variant executes gradient descent in exactly the same way as SGNS. Every time a pair (w,c) is observed by window sampling and pairs $(w,c_{n,1}),\ldots,(w,c_{n,k})$ drawn by negative sampling, $\boldsymbol{u}_w$, $\boldsymbol{v}_c$ and $\boldsymbol{v}_{c_{n,1}},\ldots,\boldsymbol{v}_{c_{n,k}}$ are updated by gradient descent on the sum of two loss function terms.

Another approach for minimization is Stochastic. In this case every context window is extended with k negative samples $c_{n,1},\ldots,c_{n,k}$. Iterative gradient descent of the first equation is then run on pairs (w_t,c), for $c \in \mathcal{C}_t = \cup_{i=-win,i\neq 0}^{win} w_{t+i}$ and $(w_t,c_{n,j})$, $j=1,\ldots,k$ for each window.

5.1.4 Swivel

Submatrix-wise vector embedding learner (Swivel) was introduced in Shazeer et al. (2016). This model is based on applying SVD for PMI matrix, and the main idea is to use a loss function which penalties depend on whether the word-context pair co-occurs in the corpus or not, so the algorithm could be trained to not to over-estimate PMI of common values whose co-occurrence is unobserved. Notably, Word2Vec with a negative sampling is also capable of taking unobserved co-occurrences into account, but it is done indirectly.

The central claim of the authors of Swivel is that none of the mainstream word embeddings provide any special treatment to unobserved word-context co-occurrences, so the ability to capture unobserved word-context co-occurrences helped to outperform other embedding training algorithms in word similarity and word analogy tasks.

For co-occurrences that have been observed ($\#(w,c)>0$), we'd like $\boldsymbol{u}_w^\top \boldsymbol{v}_c$ to accurately estimate $PMI_{w,c}$ subject to how confident we are in the observed count $\#(w,c)$. Swivel computes the weighted squared error between the embedding dot product and the PMI of w and c:

$$L^+(w,c) = \frac{1}{2} f(\#(w,c))\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PMI_{w,c}\right)^2 =$$
$$= \frac{1}{2} f(\#(w,c))\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - \log \#(w,c) - \log T + \log \#(w) + \log \#(c)\right)^2.$$

This encourages $\boldsymbol{u}_w^\top \boldsymbol{v}_c$ to correctly estimate the observed PMI. The loss is modulated by a monotonically increasing confidence function $f(x)$: the more frequently a co-occurrence

is observed, the more the model is required to accurately approximate $PMI_{w,c}$. The authors experimented with several different variants for $f(x)$, and discovered that a linear transformation of $x^{\frac{1}{2}}$ produced good results.

Unfortunately, if w and c are never observed together, $\#(w,c) = 0$, $PMI_{w,c} = `\infty$, and the squared error cannot be computed. What would we like the model to do in this case? Treating $\#(w,c)$ as a sample, we can ask: how significant is it that its observed value is zero? If the two words w and c are rare, their co-occurrence could plausibly have gone unobserved due to the fact that we simply haven't seen enough data. On the other hand, if words w and c are common, this is less likely: it becomes significant that a co-occurrence hasn't been observed, so perhaps we ought to consider that the features are truly anti-correlated. In either case, we certainly don't want the model to over-estimate the PMI between features, and so we can encourage the model to respect an upper bound on its PMI estimate $\boldsymbol{u}_w^\top \boldsymbol{v}_c$.

We address this by smoothing the PMI value as if a single co-occurrence had been observed (i.e., computing PMI as if $\#(w,c) = 1$), and using an asymmetric cost function that penalizes over-estimation of the smoothed PMI. The following "soft hinge" cost function accomplishes this:

$$\begin{aligned} L^0(w,c) &= \log\left(1 + \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PMI^*_{w,c}\right)\right) = \\ &= \log\left(1 + \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - \log T + \log \#(w) + \log \#(c)\right)\right). \end{aligned}$$

Here, PMI^* refers to the smoothed PMI computation where $\#(w,c)$'s actual count of 0 is replaced with 1. This loss penalizes the model for over-estimating the objective value; however, it applies negligible penalty — i.e., is noncommittal—if the model under-estimates it.

5.2 Morphology. Inflectional and Derivational Morphology

In this section, we will introduce several basic definitions fom morphology that will be needed to understand further models.

Morphology is a study of internal structure of words.

Morpheme is the smallest linguistic unit which has a meaning or grammatical function. Words are composed of morphemes (one or more). There are some complications with this simple definition: *sing/er/s, moon/light, un/kind/ly, talk/s, ten/th, de/nation/al/iz/ation.* The order of morphemes matters: *talk/ed* $\neq$ **ed/talk, re/write* $\neq$ **write/re.*

Morph. The term morpheme is used both to refer to an abstract entity and its concrete realization(s) in speech or writing. When it is needed to maintain the signified and signifier distinction, the term morph is used to refer to the concrete entity, while the term morpheme is reserved for the abstract entity only.

Allomorphs are morphemes having the same function but different form. Unlike the synonyms they usually cannot be replaced one by the other. Here are

1. (a) indefinite article: ***an*** *orange* — ***a*** *building*
   (b) plural morpheme: *cat/**s*** — *dog/**s*** — *judg/**es***
2. (a) Czech: *matk/**a*** *(mother$_{nom}$)* — *mat/ek/ (mothers$_{gen}$)* — *matc/**e*** *(mother$_{dat}$)* — *matč/**in*** *(mother's)*

5.2.1 Classification of Morphemes

Bound and Free

- **Bound** morpheme cannot appear as a word by itself: *-s (dog/s), -ly (quick/ly), -ed (walk/ed).*
- **Free** morpheme can appear as a word by itself; often can combine with other morphemes too: *dog (dog/s), walk (walked), of, the, or.*

Root and Affixes

- **Root** is a nucleus of the word that affixes attach too. In English, most of the roots are free. In some languages that is less common (Lithuanian: *Bill/as Clinton/as*). Compounds contain more than one root: *home/work.*
- **Affix** is a morpheme that is not a root; it is always bound. There are several kinds of affixes. The most common are the following:
  - **suffix**: *talk — talk/**ing**, quick — quick/**ly***;
  - **prefix**: *happy — **un**/happy, existing — **pre**/existing*;
  - **circumfix**: Dutch: *berg (mountain) — **ge**/berg/**te** (mountains), *__ge__/berg, *berg/__te__.*
  - **infix**: Tagalog: *basa (read) — b/**um**/asa (read$_{past}$)*;
  - **transfix**: Arabic: *k-t-b (write) — k/**a**/t/**a**/b/**a** (he wrote), **ya**/kt/**u**/b/**u** (he is writing)*;
  - **interfix**[1]: *speed/**o**/meter.*

  Suffixes are more common than prefixes which are more common than infixes/circumfixes.

Content and Functional

- **Content** morphemes carry some semantic content *car, -able-, un-, -ness.*
- **Functional** morphemes provide grammatical information *the, and, -s (plural), -s (3rd sg).*

Inflection and Derivation

- **Inflection** is a process of creating various forms of the same word: *big — bigger, biggest.* **Lexeme** is an abstract entity; the set of all forms related by inflection (but not derivation). **Lemma** is a form from a lexeme chosen by convention (e.g., nom.sg. for nouns, infinitive for verbs) to represent that set. Also called the canonical / base / dictionary / citation form. E.g., *break, breaks, broke, broken, breaking* have the same lemma *break.* **Ending** is an inflectional suffix.
- **Derivation** is a process of creating new words: *slow — slowly, slowness.*

  Derivation tends to affects the meaning of the word, while inflection tends to affect only its syntactic function. Derivation tends to be more irregular — there are more

[1] Traditionally, interfix is not considered as morpheme because it has no individual value.

gaps, the meaning is more idiosyncratic and less compositional. However, the boundary between derivation and inflection is often fuzzy and unclear.

For any word we can build a tree with a "history" of a word: *unbelievable = un + (believe + able)*, but *unbelievable ≠ *(un + believe) + able*. At the same time, some words can be ambiguous: *unlockable = (un + lock) + able, unlockable = un + (lock + able)*.

5.2.2 Morphological processes

- **Concatenation** (adding continuous affixes) is the most common process. Often phonological changes on morpheme boundaries.
- **Reduplication** is a process when a part of the word or the entire word is doubled. Afrikaans: *amper (nearly) — amper/amper (very nearly)*; Tagalog: *basa (read) — ba/basa (will read)*; English: *humpty/dumpty*.
- **Templates** Both the roots and affixes are discontinuous. Only Semitic languages (Arabic, Hebrew). A root (3 or 4 consonants, e.g., *l-m-d (learn)*) is interleaved with a (mostly) vocalic pattern. Hebrew: *lomed (learn$_{masc}$) — lomad (learnt$_{masc.sg.3rd}$) — limed (taught$_{masc.sg.3rd}$) — lumad (was taught$_{masc.sg.3rd}$), shatak (be quiet$_{pres.masc}$) — shatak (was quiet$_{masc.sg.3rd}$) — shitek (made sb to be quiet$_{masc.sg.3rd}$) — shutak (was made to be quiet$_{masc.sg.3rd}$)*
- **Morpheme internal changes (apophony, ablaut)** are the processes when the word changes internally. English: *sing — sang — sung, man — men, goose — geese*; German: *Hund (dog) — Hünd/chen (small dog)*.
- **Subtraction (Deletion)**: some material is deleted to create another form. Papago (a native American language in Arizona): *him (walking$_{imperf.}$) — hi (walking$_{perf.}$)*; French: *grande (big$_f$) — grand (big$_m$), fausse (false$_f$) — faux (false$_m$)*.
- **Suppletion**: English: *be — am — is — was, go — went, good — better*.

5.2.3 Word formation

- **Affixation**. Words are formed by adding affixes: *write — writer, productive — unproductive*.
- **Compounding**. Words are formed by combining two or more words: *rain + bow — rainbow, over + do — overdo*.
- **Acronyms** are like abbreviations, but act as a normal words: *light amplification by simulated emission of radiation — laser*.
- **Blending** parts of two different words are combined: *breakfast + lunch — brunch, motor + hotel — motel*.
- **Clipping** longer words are shortened: *doctor — doc, laboratory — lab, advertisement — ad, examination — exam*.

5.2.4 Morphological Types of Languages

- **Analytic** languages have only free morphemes, sentences are sequences of single morpheme words. A grammatical meaning is usually expressed with separate words.
- **Synthetic** languages have both free and bound morphemes. Unlike analytic languages, a grammatical meaning is usually expressed inside words.
  - **Agglutinating** — each morpheme has a single function, it is easy to separate them. E.g., Uralic lgs (Estonian, Finnish, Hungarian), Turkish, Basque, Dravidian languages (Tamil, Kannada, Telugu), Esperanto.
  - **Fusional** — like agglutinating, but affixes tend to "fuse together", one affix has more than one function. Common homonymy of inflectional affixes. E.g., Slavic, Romance languages, Greek.
  - **Polysynthetic** — extremely complex, many roots and affixes combine together, often one word corresponds to a whole sentence in other languages. E. g. Eskimo: *angyaghllangyugtuq (he wants to acquire a big boat).*

  Usually languages combine several types in different proportions.

5.3 N-grams. FastText

In this section, we describe a model to learn word representations while taking into account morphology. It was proposed by Bojanowski et al. (2017). A morphology is modeled by considering subword units, and representing words by a sum of its character n-grams. We will begin by presenting the general framework that is used to train word vectors, then present a subword model and eventually describe how to handle the dictionary of character n-grams.

5.3.1 General Model

We start by briefly reviewing the continuous skip-gram model introduced by Mikolov et al. (2013), from which this model is derived. Given a word vocabulary $\mathcal{W}$, where a word is associated with its index $w \in \{1, \ldots, |\mathcal{W}|\}$, the goal is to learn a vectorial representation for each word $w \in \mathcal{W}$. Inspired by the distributional hypothesis, word representations are trained to *predict well* words that appear in its context. More formally, given a large training corpus represented as a sequence of words $w_1, \ldots, w_T$, the objective of the skipgram model is to maximize the following log-likelihood:

$$\sum_{t=1}^{T} \sum_{c \in \mathcal{C}_t} \log P(c \mid w_t),$$

where the context $\mathcal{C}_t$ is the set of words surrounding word w_t. The probability of observing a context word c given w_t will be parameterized using the aforementioned word vectors. For now, let us consider that we are given a scoring function s which maps pairs of (word, context) to scores in $\mathbb{R}$. One possible choice to define the probability of a context word is the softmax:

$$P(c \mid w_t) = \frac{\exp(s(w_t,\ c))}{\sum_{c' \in \mathcal{W}} \exp(s(w_t,\ c'))}.$$

However, such a model is not adapted to our case as it implies that, given a word w_t, we only predict one context word c.

The problem of predicting context words can instead be framed as a set of independent binary classification tasks. Then the goal is to independently predict the presence (or absence) of context words. For the word at position t we consider all context words as positive examples and sample negatives at random from the dictionary. For a chosen context word c, using the binary logistic loss, we obtain the following negative log-likelihood:

$$\log\left(1+e^{-s(w_t,\ c)}\right)+\sum_{c_n\in\mathcal{N}_{t,c}}\log\left(1+e^{s(w_t,\ c_n)}\right),$$

where $\mathcal{N}_{t,c}$ is a set of negative examples sampled from the vocabulary. By using the sigmoid function, we can re-write the objective as:

$$\sum_{t=1}^{T}\left[\sum_{c\in\mathcal{C}_t}\log\sigma(s(w_t,\ c))+\sum_{c_n\in\mathcal{N}_{t,c}}\log\sigma(-s(w_t,\ c_n))\right].$$

A natural parameterization for the scoring function s between a word w_t and a context word c is to use word vectors.

Let us define for each word w in the vocabulary two vectors $\boldsymbol{u}_w$ and $\boldsymbol{v}_w$ in $\mathbb{R}^d$. These two vectors are sometimes referred to as *input* and *output* vectors in the literature. In particular, we have vectors $\boldsymbol{u}_{w_t}$ and $\boldsymbol{v}_c$, corresponding, respectively, to words w_t and c. Then the score can be computed as the scalar product between word and context vectors as $s(w_t, c) = \boldsymbol{u}_{w_t}^\top \boldsymbol{v}_c$.

5.3.2 FastText Model

By using a distinct vector representation for each word, the skipgram model ignores the internal structure of words. In this section, we propose a different scoring function s, in order to take into account this information.

Each word w is represented as a bag of character n-grams. We add special boundary symbols `<` and `>` at the beginning and end of words, allowing to distinguish prefixes and suffixes from other character sequences. We also include the word w itself in the set of its n-grams, to learn a representation for each word (in addition to character n-grams). Taking the word *where* and $n = 3$ as an example, it will be represented by the character n-grams:

```
<wh, whe, her, ere, re>
```

and the special sequence

```
<where>.
```

Note that the sequence `<her>`, corresponding to the word *her* is different from the tri-gram `her` from the word *where*. In practice, we extract all the n-grams for $3 \leq n \leq 6$. This is a very simple approach, and different sets of n-grams could be considered, for example taking all prefixes and suffixes.

Suppose that you are given a dictionary of n-grams. Given a word w, let us denote by $\mathcal{G}_w^n$ the set of n-grams appearing in w. We associate a vector representation $\boldsymbol{z}_{g^n}$ to each n-gram g^n. We represent a word by the mean of the vector representations of its n-grams. We thus obtain the scoring function:

$$s(w,c)=\frac{1}{|\mathcal{G}_w^n|}\sum_{g^n\in\mathcal{G}_w^n}\boldsymbol{z}_{g^n}^\top\boldsymbol{v}_c.$$

The embedding of a word w, then, can be expressed as

$$\boldsymbol{u}_w = \frac{1}{|\mathcal{G}_w^n|} \sum_{g^n \in \mathcal{G}_w^n} \boldsymbol{z}_{g^n}.$$

This simple model allows sharing the representations across words, thus allowing to learn reliable representation for rare words.

5.4 Subword LexVec

The LexVec model factorizes the PPMI-weighted word-context co-occurrence matrix using stochastic gradient descent. LexVec adjusts the PPMI matrix using *context distribution smoothing*.

With the PPMI matrix calculated, the sliding window process is repeated and the following loss functions are minimized for every observed (w, c) pair and target word w:

$$L^{WC}(w, c) = \frac{1}{2}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c - PPMI^*_{w,c}\right)^2,$$
$$L^{W}(w) = \frac{1}{2}\sum_{j=1}^{k} \mathbb{E}_{c_{n,j} \sim P_n}\left(\boldsymbol{u}_w^\top \boldsymbol{v}_{c_{n,j}} - PPMI^*_{w,c_{n,j}}\right)^2$$

where $\boldsymbol{u}_w$ and $\boldsymbol{v}_c$ are d-dimensional word and context vectors. The second loss function describes how, for each target word, k *negative samples* are drawn from the smoothed context unigram distribution.

Given a set of subwords $\mathcal{G}_w^s$ for a word w, we follow fastText and replace $\boldsymbol{u}_w$ in loss function equations by $\boldsymbol{u}'_w$ such that:

$$\boldsymbol{u}'_w = \frac{1}{1 + |\mathcal{G}_w^s|}\left(\boldsymbol{u}_w + \sum_{g^s \in \mathcal{G}_w^s} \boldsymbol{q}_{hash(g^s)}\right)$$

such that a word is the sum of its word vector and its d-dimensional subword vectors $\boldsymbol{q}_x$. The number of possible subwords is very large so the function $hash(g^s)$ hashes a subword to the interval $[1, buckets]$. For OOV words,

$$\boldsymbol{u}'_w = \frac{1}{|\mathcal{G}_w^s|} \sum_{g^s \in \mathcal{G}_w^s} \boldsymbol{q}_{hash(g^s)}.$$

Subword LecVec was proposed in Salle and Villavicencio (2018). They compared two types of subwords: simple n-grams (like FastText) and unsupervised morphemes.

5.5 Byte Pair Embeddings

5.5.1 Byte Pair Encoding

Byte pair encoding (BPE) (Gage (1994)) is a variable-length encoding that views text as a sequence of symbols and iteratively merges the most frequent symbol pair into a new symbol. E.g., encoding an English text might consist of first merging the most frequent symbol pair *t h* into a new symbol *th*, then merging the pair *th e* into *the* in the next

Merge ops	Byte-pair encoded text
5000	豊 田 （ と よ だ え き ） は 、 東京都 日 野 市 豊 田 四 丁目 にある
10000	豊 田 （ と よ だ えき ） は 、 東京都 日 野市 豊 田 四 丁目にある
25000	豊 田 (とよ だ えき) は 、 東京都 日 野市 豊田 四 丁目にある
50000	豊 田 (とよ だ えき) は 、 東京都 日 野市 豊田 四丁目にある
Tokenized	豊田 （ と よ だ え き ） は 、 東京 都 日野 市 豊田 四 丁目 に ある
10000	田 站 是 東 日本 旅 客 鐵 道 (JR 東 日本) 中央 本 線 的 鐵路 車站
25000	田 站是 東日本旅客鐵道 (JR 東日本) 中央 本 線的鐵路車站
50000	田 站是 東日本旅客鐵道 (JR 東日本) 中央 本線的鐵路車站
Tokenized	田站 是 東日本 旅客 鐵道 (JR 東日本) 中央本線 的 鐵路車站
1000	to y od a _station is _a _r ail way _station _on _the _ch ū ō _main _l ine
3000	to y od a _station _is _a _railway _station _on _the _ch ū ō _main _line
10000	toy oda _station _is _a _railway _station _on _the _ch ū ō _main _line
50000	toy oda _station _is _a _railway _station _on _the _chū ō _main _line
100000	toy oda _station _is _a _railway _station _on _the _chūō _main _line
Tokenized	toyoda station is a railway station on the chūō main line

Table 5.1: Effect of the number of BPE merge operations on the beginning of the Japanese (top), Chinese (middle), and English (bottom) Wikipedia article TOYODA_STATION. Since BPE is based on frequency, the resulting segmentation is often, but not always meaningful. E.g. in the Japanese text, 豊 (toyo) and 田 (ta) are correctly merged into 豊田 (Toyoda, a Japanese city) in the second occurrence, but the first 田 is first merged with (eki, *train station*) into the meaningless 田 (ta-eki).

iteration, and so on. The number of merge operations o determines if the resulting encoding mostly creates short character sequences (e.g. $o = 1000$) or if it includes symbols for many frequently occurring words, e.g. $o = 30000$ (cf. Table 5.1). Since the BPE algorithm works with any sequence of symbols, it requires no preprocessing and can be applied to untokenized text.

5.5.2 BPEmb

BPEmb (Heinzerling and Strube (2018)) is a collection of pre-trained subword unit embeddings in 275 languages, based on byte-pair encoding. In an evaluation using fine-grained entity typing as testbed, BPEmb performs competitively, and for some languages better than alternative subword approaches, while requiring vastly fewer resources and no tokenization. BPEmb is available at `https://github.com/bheinzerling/bpemb`.

The authors applied BPE to all Wikipedias of sufficient size with various o and pre-trained embeddings for the resulting BPE symbol using GloVe, resulting in byte-pair embeddings for 275 languages. To allow studying the effect the number of BPE merge operations and of the embedding dimensionality, the authors provide embeddings for 1000, 3000, 5000, 10000, 25000, 50000, 100000 and 200000 merge operations, with dimensions 25, 50, 100, 200, and 300.

5.6 Morphological Embeddings

5.6.1 Capturing Morphology in Existing Models

As we have seen in 4.6, one of the criteria in word embedding model evaluation is its ability to solve the analogical reasoning problem, and the Skip-gram architecture proved to work on

a group of relations. But the deeper analysis in Linzen (2016) showed that a lot of particular (not only) morphological relations are not captured in the existing models.

The logical improvement is then to build the models that do not have this disadvantage by including morphological information explicitly, i. e. learning individual vector representations for morphological units. Moreover, morphological embeddings are theoretically grounded, unlike the heuristics such as FastText or BPE, which are, however, rather good for some languages.

In the following sections, we are going to show approaches based on a-priori language knowledge.

5.6.2 Subword-level Embeddings for Korean

We start with a FastText modification introduced in Park et al. (2018) for Korean language.

Decomposition of Korean Words

Korean words are formed by an explicit hierarchical structure which can be exploited for better modeling. Every word can be decomposed into a sequence of characters, which in turn can be decomposed into *jamo*s, the smallest lexicographic units representing the consonants and vowels of the language. Unlike English which has a more flexible sequences of consonants and vowels making up syllables (e.g., "straight"), a Korean "character" which is similar to a syllable in English has a rigid structure of three *jamo*s. They have names that reflect the position in a character: 1) chosung (syllable onset), 2) joongsung (syllable nucleus), and 3) jongsung (syllable coda). Each component indicates how the character should be pronounced. With the exception of empty consonants, chosung and jongsung are consonants while joongsung are vowels. The *jamo*s are written with the chosung on top, with joongsung on the right of or below chosung, and jongsung on the bottom (see Fig. 5.1). This combination of *jamo*s completes a character as a syllable.

(a) chosung (b) joongsung (c) jongsung

Figure 5.1: Example of the composition of a Korean character. Each character is comprised of 3 parts as shown in example of '달Moon'. On the other hand, as in the top case '해Sun', some characters lack the last component, 'jongsung'.

As shown in the top of Fig. 5.1, some characters such as '해Sun' lack jongsung. In this case, we add an empty jongsung symbol `e` such that a character always has three (*jamo*s). Thus, the character '달Moon' is decomposed into {ㄷ, ㅏ, ㄹ}, and '해Sun' into {ㅎ, ㅐ, e}.

When decomposing a word, we keep the order of the characters and the order of *jamo*s (chosung, joongsung, and jongsung) within the character. By following this rule, we ensure

that a Korean word with N characters will have $3N$ *jamo*s in order. Lastly, the symbols for start of a word < and end of a word > are added to the sequence. For example, the word '강아지puppy' will be decomposed to a sequence of *jamo*s: {<, ㄱ, ㅏ, ㅇ, ㅇ, ㅏ, e, ㅈ, ㅣ, e, >}.

Extracting n-grams from *jamo* Sequence

We extract the following *jamo*-level and character-level n-grams from the decomposed Korean words: 1) character-level n-grams, and 2) inter-character *jamo*-level n-grams. These two levels of subword features can be successfully integrated into *jamo*-level n-grams by ensuring a character has three *jamo*s, adding empty jongsung symbol to the sequence. For better understanding, we start with the word '먹었다ate'.

Character-level n-grams

Since we add the empty jongsung symbol e when decomposing characters, we can find *jamo*-level trigrams representing a single character in the decomposed *jamo* sequence of a word. For example, there are three character-level unigrams in the word '먹었다ate':

{ㅁ, ㅓ, ㄱ}, {ㅇ, ㅕ, ㅆ}, {ㄷ, ㅏ, e}

Next, we find character-level n-grams by using the extracted unigrams. Adjacent unigrams are attached to construct n-grams. There are two character-level bigrams, and one trigram in the example:

{ㅁ, ㅓ, ㄱ, ㅇ, ㅕ, ㅆ}, {ㅇ, ㅕ, ㅆ, ㄷ, ㅏ, e}
{ㅁ, ㅓ, ㄱ, ㅇ, ㅕ, ㅆ, ㅇ, ㅕ, ㅆ, ㄷ, ㅏ, e}

Lastly, we add the total *jamo* sequence of a word including < and > to the set of extracted character-level n-grams.

Inter-character *jamo*-level n-grams

Since Korean is a member of the agglutinative languages, a syntactic character is attached to the semantic part in the word, and this generates many variations. These variations are often determined by *jamo*-level information. For example, usage of the subjective case '이' or '가' is determined by the existence of jongsung in the previous character. In order to learn these regularities, we consider *jamo*-level n-grams across adjacent characters as well. For instance, there are 6 inter-character *jamo*-level trigrams in the example:

{<, ㅁ, ㅓ}, {ㅓ, ㄱ, ㅇ}, {ㄱ, ㅇ, ㅕ},
{ㅆ, ㄷ, ㅏ}, {ㅓ, ㅆ, ㄷ}, {ㅏ, e, >}

Vector Representation

Similarly to FastText, we construct a vector representation of a Korean word by using the extracted two types of n-grams. We compute the sum of *jamo*-level n-grams, sum of character-level n-grams, and compute mean of the vectors. Let us denote character-level n-grams of w to $\mathcal{G}_w^c$, and inter-character *jamo*-level n-grams $\mathcal{G}_w^j$, then we obtain the equation for embedding of word w as follows:

$$\boldsymbol{u}_w = \frac{1}{|\mathcal{G}_w^c| + |\mathcal{G}_w^j|} \left(\sum_{g^c \in \mathcal{G}_w^c} \boldsymbol{z}_{g^c} + \sum_{g^j \in \mathcal{G}_w^j} \boldsymbol{z}_{g^j} \right)$$

where $\boldsymbol{z}_{g^j}$ is the vector representation of the *jamo*-level n-gram g^j, and $\boldsymbol{z}_{g^c}$ is that of the character-level n-gram g^c.

5.6.3 Morpheme Segmentation

Morpheme segmentation is a process of deriving words into morphemes. One may think that a morpheme segmentation based model is the same as for FastText, and the only difference is in using morphemes instead of n-grams. An embedding for a word w can be computed as

$$\boldsymbol{u}_w = \frac{1}{|\mathcal{G}_w^m|} \sum_{g^m \in \mathcal{G}_w^m} \boldsymbol{z}_{g^m},$$

where $\mathcal{G}_w^m$ represents the set of morphemes of word w.

But the segmentation process itself can be a significantly difficult task, especially for fusional and polysynthetic languages. For various languages segmentation algorithms are developed. Such algorithms in the most cases use statistical information, but in recent years neural network based model were proposed.

5.6.4 Compositionality on a Morphological Level

Lazaridou et al. (2013) were the first to predict distributional vectors for derived words using CDSMs and experimented with a range of established CDSMs. In their paper, all models are supervised, i.e., some word pairs for each pattern are used as training instances, and others serve for evaluation. Also, all models assume that the base word (input) b and derived word (output) d are represented as vectors in some underlying distributional space.

Simple Additive Model

The simple additive model predicts the derived word from the base word as

$$\boldsymbol{v}_d = \boldsymbol{v}_b + \boldsymbol{v}_p$$

where $\boldsymbol{v}_p$ is a vector representing the semantic shift accompanying the derivation pattern p.

Simple Multiplicative Model

The simple multiplicative model is very similar, but uses component-wise multiplication

$$\boldsymbol{v}_d = \boldsymbol{v}_b \odot \boldsymbol{v}_p$$

instead of addition to combine the base and pattern vectors (Mitchell and Lapata, 2010).

Weighted Additive Model

The third model, the weighted additive model, enables a simple reweighting of the contributions of basis and pattern

$$\boldsymbol{v}_d = \alpha \boldsymbol{v}_b + \beta \boldsymbol{v}_p.$$

Basic Additive Model

The basic additive model (introduced in (Mitchell and Lapata, 2008)) computes the distibutional semantics of a pair of words according to formula

$$\boldsymbol{v}_d = (1 - \alpha)\boldsymbol{v}_b + \alpha \boldsymbol{v}_p.$$

Dilation Model

In the dilation model, the output vector is obtained by first decomposing one of the input vectors, say $\boldsymbol{v}_b$, into a vector parallel to $\boldsymbol{v}_p$ and an orthogonal vector. Following this, the parallel vector is dilated by a factor α before re-combining. This results in:

$$\boldsymbol{v}_d = (\alpha - 1)(\boldsymbol{v}_p^\top \boldsymbol{v}_b)\boldsymbol{v}_p + (\boldsymbol{v}_p^\top \boldsymbol{v}_p)\boldsymbol{v}_b.$$

Lexical Function Model

The lexical function model (Baroni and Zamparelli, 2010) represents the pattern as a matrix $\boldsymbol{P}_p$ that is multiplied with the basis vector:

$$\boldsymbol{v}_d = \boldsymbol{P}_p \boldsymbol{v}_b,$$

essentially modelling derivation as linear mapping. This model is considerably more powerful than the others, however its number of parameters is quadratic in the number of dimensions of the underlying space, whereas the additive and multiplicative models only use a linear number of parameters.

The main advance of this model is that a commutative law generally does not hold for matrices, and it allows us to derive words taking into account the order of applying derivational patterns. E.g., $\boldsymbol{P}_{able}(\boldsymbol{P}_{un}(\boldsymbol{v}_{lock})) \neq \boldsymbol{P}_{un}(\boldsymbol{P}_{able}(\boldsymbol{v}_{lock}))$.

Note that we can defeine pattern in a various ways. For instance, if we define them like $\boldsymbol{P}_{V+able}(\boldsymbol{P}_{un+V}(\boldsymbol{v}_{lock})) \neq \boldsymbol{P}_{un+Adj}(\boldsymbol{P}_{V+able}(\boldsymbol{v}_{lock}))$ then we will have different matrices (vectors in other models) for the same patterns applied to different parts of speech.

Full Additive Model

The full additive model computes the compositional vector of a pair using two linear tranformations $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively applied to the vectors of the base and the pattern:

$$\boldsymbol{v}_d = \boldsymbol{A}\boldsymbol{v}_b + \boldsymbol{B}\boldsymbol{v}_p.$$

5.6.5 Derivational Morphology Resources

To construct and evaluate a complicated model based on derivational morphology, one needd first to somehow build a lexical resource with the information about derivation relations between words.

CELEX

For English, Lazaridou et al. (2013) obtained a list of stem/derived-form pairs from the **CELEX** English Lexical Database, a widely used 100K-lemma lexicon containing, among other things, information about the derivational structure of words (Baayen et al., 1995). For each derivational affix present in CELEX, they extracted from the database the full list of stem/derived pairs matching its most common part-of-speech signature (e.g., for *-er* they only considered pairs having a verbal stem and nominal derived form). Since CELEX was populated by semi-automated morphological analysis, it includes forms that are probably not synchronically related to their stems, such as *crypt+ic* or *re+form*. However, they did not manually intervene on the pairs, since they are interested in training and testing their methods in realistic, noisy conditions.

DErivBase

Zeller et al. (2013) described a rule-based framework for inducing derivational families (i.e., clusters of lemmas in derivational relationships) and its application to create a high-coverage German resource, **DErivBase**, mapping over 280k lemmas into more than 17k non-singleton clusters.

They implemented the derivational rules from Hoeppner (1980) for verbs, nouns, and adjectives, covering such processes as zero derivation, prefixation, suffixation, circumfixation, and stem changes. DErivBase is available at `https://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/DErivBase.html`.

5.7 Seminar

TODO

Bibliography

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association of Computational Linguistics*, 5(1):135–146.

Gage, P. (1994). A new algorithm for data compression. *C Users J.*, 12(2):23–38.

Heinzerling, B. and Strube, M. (2018). BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In chair), N. C. C., Choukri, K., Cieri, C., Declerck, T., Goggi, S., Hasida, K., Isahara, H., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., Piperidis, S., and Tokunaga, T., editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Kisselew, M., Padó, S., Palmer, A., and Šnajder, J. (2015). Obtaining a better understanding of distributional models of german derivational morphology. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 58–63.

Lazaridou, A., Marelli, M., Zamparelli, R., and Baroni, M. (2013). Compositional-ly derived representations of morphologically complex words in distributional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1517–1526. Association for Computational Linguistics.

Linzen, T. (2016). Issues in evaluating semantic spaces using word analogies. *CoRR*, abs/1606.07736.

Marelli, M. and Baroni, M. (2015). Affixation in semantic space: Modeling morpheme meanings with compositional distributional semantics. *Psychological review*, 122(3):485.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Park, S., Byun, J., Baek, S., Cho, Y., and Oh, A. (2018). Subword-level word vector representations for korean. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2429–2438. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Salle, A., Idiart, M., and Villavicencio, A. (2016). Matrix factorization using window sampling and negative sampling for improved word representations. *CoRR*, abs/1606.00819.

Salle, A. and Villavicencio, A. (2018). Incorporating subword information into matrix factorization word embeddings. *CoRR*, abs/1805.03710.

Shazeer, N., Doherty, R., Evans, C., and Waterson, C. (2016). Swivel: Improving embeddings by noticing what's missing. *CoRR*, abs/1602.02215.

Zeller, B., Šnajder, J., and Padó, S. (2013). DErivBase: Inducing and evaluating a derivational morphology resource for German. In *Proceedings of ACL 2013*, pages 1201–1211, Sofia, Bulgaria.