[draft] Note 1: Introduction and Word2Vec [1] [2] [3]

CS 224n: Natural Language Processing with Deep Learning

Winter 2023

[1] Course Instructors: Christopher Manning, John Hewitt

[2] Author: John Hewitt
`johnhew@cs.stanford.edu`

[3] Contributors to past notes: Francois Chaubard, Michael Fang, Guillaume Genthial, Rohit Mundra, Richard Socher

Summary. This note introduces the field of Natural Language Processing (NLP) briefly, and then discusses word2vec and the fundamental, beautiful idea of representing words as low-dimensional real-valued vectors learned from distributional signal.

1 Introduction to Natural Language Processing

Natural language processing is a field of science and engineering focused on the development and study of automatic systems that understand and generate natural (that is, human,) languages.

1.1 Humans and language

Human languages are communicative devices enabling the efficient sharing and storage of complex ideas, facts, and intents. As [Manning, 2022] argues, the complexity of communication enabled by language is a uniquely human intelligence among species. As scientists and engineers interested in the creation and study of intelligent systems, human language is to us both a fascinating object of study—after all, it has *evolved to be learnable and useful*—and a great enabler for interacting with humans even in contexts where other modalities (e.g., vision) are also of interest.

1.2 Language and machines

Human children, interacting with a rich multi-modality world and various forms of feedback, acquire language with exceptional sample efficiency (not observing that much language) and compute efficiency (brains are efficient computing machines!) With all the (impressive!) advances in NLP in the last decades, we are still nowhere close to developing learning machines that have a fraction of acquisition ability of children. One fundamental (and still quite open) problem in building language-learning machines is the question of **representation**; how should we represent language in a computer such that the computer can robustly process and/or generate it? This is where this course focuses on the tools provided by **deep learning**, a highly effective toolkit for representing both the wild variety of natural language and some of the rules and structures it sometimes adheres to. Much of this course will be dedicated to this question of representation, and the rest of this note will talk about a basic subquestion: **how do we represent words?** Before that, though, let's briefly discuss some of the applications you can hope to build after learning modern NLP techniques.

1.3 *A few uses of NLP*

Natural language processing algorithms are increasingly useful and deployed, but their failures and limitations are still largely opaque and sometimes hard to detect. Here are a few of the major applications; this list is intended to pique your interest, not to be exhaustive:

Machine translation. Perhaps one of the earliest and most successful applications and driving uses of natural language processing, MT systems learn to translate between languages and are ubiquitous in the digital world. Still, failures of these systems for most of the world's 7000 languages, difficulties in translating long text, and ensuring contextual correctness of translations make this still a fruitful field of research.

Question answering and information retrieval. The concept of "question answering" should seem overly broad—can't we express any problem as question answering?—but in NLP, question answering has tended to be related to information-seeking questions ("Who is the emir of Abu Dhabi?", "What is the process by which I can get an intern visa for the United Kingdom?"). Continually broadening the scope of answerable questions, providing provenance for answers, answering questions in an interactive dialogue—this is one of the fastest-evolving research directions.

Summarization and analysis of text. There are myriad reasons to want to understand (1) what people are talking about and (2) what they think about those things. Companies want to do market research, politicians want to know peoples' opinions, individuals want summaries of complex topics in digestible form. NLP tools can be powerful for both the increase of access to information to the public, as well as surveillance, corporate or governmental. Bear this aspect of "dual use" in mind as you progress and decide what you are building.

Note: speech(or sign)-to-text. The process of automatic transcription of spoken or signed language (audio or video) to textual representations is a massive and useful application, but one we'll largely avoid in this course. Partly, this is historical and methodological; the raw signal processing methods and expertise are generally covered in other courses (224s!) and other research communities, though there has been some convergence of techniques of late.

In all aspects of NLP, most existing tools work for precious few (usually one, maybe up to 100) of the world's roughly 7000 languages, and fail disproportionately much on lesser-spoken and/or marginalized dialects, accents, and more. Beyond this, recent successes in building better systems have far outstripped our ability to characterize and audit these systems. Biases encoded in text, from race to gender to religion and more, are reflected and often amplified by NLP systems. With these challenges and considerations in mind, but with the desire to do good science and build trustworthy systems that improve peoples' lives, let's take a look at a fascinating first problem in NLP.

2 *Representing words*

2.1 *Signifier and signified*

Consider the sentence

Zuko makes the tea for his uncle.

The word *Zuko* is a sign, a symbol that represents an entity ZUKO in some (real of imagined) world. The word *tea* is also a symbol that refers to a signified thing—perhaps a specific instance of tea. If one were instead to say *Zuko likes to make tea for his uncle*, note that the symbol *Zuko* still refers to ZUKO, but now *tea* refers to a broader class—tea in general, not a *specific* bit of hot delicious water. Consider the two following sentences:

Zuko makes the coffee for his uncle.
Zuko makes the drink for his uncle.

Which is "more like" the sentence about tea? The *drink* may be tea (or it may be quite different!) and *coffee* definitely isn't tea, but is yet similar, no? And is *Zuko* similar to *uncle* because they both describe people? And is *the* similar to *his* because they both pick out specific instances of a class?

Word meaning is endlessly complex, deriving from humans' goals of communicating with each other and achieving goals in the world. People use continuous media—speech, signing—but produce signs in a discrete, symbolic structure—language—to express complex meanings. Expressing and processing the nuance and wildness of language—while achieving the strong transfer of information that language is intended to achieve—makes representing words an endlessly fascinating problem. Let's move to some methods.

2.2 *Independent words, independent vectors*

What is a word? I cannot define a word for you, but I can give some examples in English: *tea, coffee, abbreviate, gumption*. The word *anti-radate* I hereby define to mean the action of looking wistfully at an inedible decoration, wishing it were as tasty as it looked. If I use this sign to communicate with others my longing, that's good enough to me to be a word.

Perhaps the simplest way to represent words is as independent, unrelated entities. You might think of this as a set,

$$\{\ldots, \textit{tea}, \ldots, \textit{coffee}, \ldots, \textit{antiridate}\}.$$

Here let's introduce a bit of terminology. We will refer to a word **type** as an element of a finite vocabulary, independent of actually observing the word in context. So, we've just written a set of types. A word **token** is an instance of the type, e.g., observed in some context. Our word representations right now provides a single representation for each word type, and we might use that same representation for any occurence of the word token in context.

A (word) **type** is an element of a vocabulary; a word in abstract. A (word) **token** is an instance of a type in context.

We will often be working with *vectors* in this course; the conventional vector representation of independent components is the set of **1-hot**, or standard basis, vectors. Thus, maybe

$$v_{\text{tea}} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \qquad v_{\text{coffee}} = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \tag{1}$$

where $v_{\text{tea}} = e_3$, the third standard basis vector, and $v_{\text{coffee}} = e_j$, the j^{th} standard basis vector.

Why do we represent words as vectors? To better compute with them. And when computing with 1-hot vectors, we do achieve the crucial fact that different words are different, but alas, we encode no meaningful notion of **similarity or other relationship**. This is because, for example, if we take the dot product as a notion of similarity (or the L2 distance, or the L1 distance, or. . .) we compute:

$$v_{\text{tea}}^{\top} v_{\text{coffee}} = v_{\text{tea}}^{\top} v_{\text{the}} = 0, \tag{2}$$

all words are equally dissimilar from each other. Note as well that in the diagram, words are not ordered, e.g., alphabetically—this is an important note; there is no (explicit) character-level information in these strings, beyond the strict notion of identity (is this word the same sequence of characters/bytes as this other word. If so, they have the same vector; if not, they have independent vectors.)

Since it is of course not the case that all words are equally dissimilar from each other, we'll move to some alternatives.

2.3 *Vectors from annotated discrete properties.*

Should we represent word semantics not as one-hot vectors, but instead as a collection of features and relationships to linguistic categories and other words?

For any word, say *runners*, there is a wealth of information we can annotate about that word. There is grammatical information, like PLURALITY, there's derivational information, like how the *runners* is something like the verb TO RUN plus a notion of "doer", or AGENT (think *one who runs*.) There's also semantic information, like how *runners* might be a hyponym of *humans*, or *animals*, or *entities*. (A hyponym is a member of an is-a relationship; e.g., *a runner is a human.*)

There are substantial existing resources in English and a few other languages for various kinds of annotated information about words. WordNet [Miller, 1995] annotates for synonyms, hyponyms, and other semantic relations; UniMorph [Batsuren et al., 2022] annotates for morphology (subword structure) information across many languages. With such resources, one could build word vectors that look something like

$$v_{\text{tea}} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \begin{array}{c} \text{(plural noun)} \\ \text{(3rd singular verb)} \\ \text{(hyponym-of-beverage)} \\ \vdots \\ \text{(synonym-of-chai)} \end{array} \tag{3}$$

In 2023, word vectors resulting from these methods are not the norm, and they won't be the focus of this course. One main failure is that human-annotated resources are always lacking in vocabulary compared to methods that can draw a vocabulary from a naturally occuring text source—updating these resources is costly and they're always incomplete. Another failure is a tradeoff between dimensionality and utility of the embedding—it takes a very high-dimensional vector (think much larger than the vocabulary size) to represent all of these categories, and modern neural methods that tend to operate on dense vectors do not behave well with such vectors. Finally, a continual theme we'll see in this course is that human ideas of what the right representations should be for text tend to underperform methods that allow data to determine more aspects—at least when one has a lot of data to learn from.

3 *Distributional semantics and Word2vec*

A promise of deep learning is to learn rich representations of complex objects from data. Increasingly relevant in NLP is the idea that we can *unsupervisedly* learn rich representations from data. Unsupervised (or lately, "self-supervised") learning takes data and attempts to learn learn properties of the elements of that data, often by taking part of the data (maybe a word in a sentence) and attempting to predict other parts of the data (other words) with it. In language, this idea was captured well years ago by Firth [Firth, 1957], who famously said

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

At a high level, you can think of the distribution of words that show up around the word *tea* as a way to define the meaning that word. So, *tea* shows up around *drank, the, pot, kettle, bag, delicious, oolong, hot, steam,. . .* , It should become clear that words similar to *tea* (like *coffee*) will have similar distributions of surrounding words. While simple, this is **one of the most influential and successful ideas in all of modern NLP**, and analogues of it have taken hold in myriad learning-related fields.

The **distributional hypothsis**: the meaning of a word can be derived from the distribution of contexts in which it appears.

That's the high level. But as always, the details matter. What does it mean for a word to be near another word? (Right next to it? Two away? In the same document?) How does one represent this encoding, and learn it? Let's go through some options.

3.1 *Co-occurrence matrices and document contexts*

If you were asked to code up the idea "represent a word by the distribution of words it appears near", you might immediately have the following idea:

1. Determine a vocabulary $\mathcal{V}$.

2. Make a matrix of size $|\mathcal{V}| \times |\mathcal{V}|$ of zeros.

3. Walk through a sequence of documents. For each document, for each word w in the document, add all the counts of the other words w' in the document to the row corresponding to w at the column corresponding to w'.

4. Normalize the rows by the sum.

You've just made a document-level co-occurrence matrix for your vocabulary! Call this matrix $\mathbf{X}$. The word embedding $\mathbf{X}_{\text{tea}} \in \mathbb{R}^{|\mathcal{V}|}$ (a row of $\mathbf{X}$) is substantially more immediately useful than the old 1-hot e_{tea} that we had.

One decision we made was **document-level** co-occurrence. We could instead say that a word w' is only co-occurring with w if that w' appears much closer, say, within a few words. Here's an example, where a few relevant windows for co-occurrence are labeled:

Larger notions of co-occurrence (e.g., large windows or documents) lead to more semantic or even topic-encoding representations; shorter windows lead to more syntax-encoding representations

$$\text{[It's hot and delicious. [I poured [the } \underbrace{\text{tea}}_{\text{center word}} \text{ for]}_1 \text{ my uncle]}_3\text{.]}_{\text{document}}$$

In brief, shorter windows (like the one word window above, labeled 1) seem to encode syntactic properties. For example, nouns tend to appear right next to *the* or *is*. Plural nouns don't appear right next to *a*. Larger windows tend to encode more semantic (and at extremes, topic-like) properties. Note how *poured* or *delicious* may occur farther from tea but still be relevant. Document-level windows, for large documents (1000s of words) intuitively represent words by what kinds of documents they appear in (sports, law, medicine, etc.)

Another design decision we made was to represent explicit counts of words in $|\mathcal{V}|$-sized vectors. This ends up being a big mistake. We've already stated that high-dimensional vectors tend to be unwieldy in today's neural systems. But another issue is that raw counts of words end up over-emphasizing the importance of very common words like *the*. Taking the **log token frequency** ends up being much more useful. A very influential paper on word representation taught us much more about what is wrong with the raw co-occurrence method by introducing **GloVe** (Pennington et al., 2014) a co-occurence-based word representation algorithm that works as well as word2vec, the method we'll introduce in the next section. However, many of the details of word2vec will hold true in methods that we'll proceed to further in the course, so we'll focus our time on that.

3.2 *Word2vec model and objective*

The word2vec model represents each word in a fixed vocabulary as a low-dimensional (much smaller than vocabulary size) vector. It learns the value of each word's vector to be predictive via a simple function of the distribution of words in a (usually short; 2-4 words) context. The model we'll describe here is called the *skipgram* word2vec algorithm.

Skipgram word2vec. As usual, we have a finite vocabulary $\mathcal{V}$. Let C, O be random variables representing an (unknown) pair of $C \in \mathcal{V}$ (a *center word*), and $O \in \mathcal{V}$ (an *outside word*, appearing in the context of the center word). We'll use c, o to refer to specific values of the random variables. Let $U \in \mathbb{R}^{|\mathcal{V}|\times d}$ and $V \in \mathbb{R}^{|\mathcal{V}|\times d}$. Note that each word in $\mathcal{V}$ is associated with a single row of U and one of V; we think of this as resulting from an arbitrary ordering of $\mathcal{V}$. The word2vec model is a probabilistic model specified as follows, where u_w refers to the row of U corresponding to word $w \in \mathcal{V}$ (and likewise for V):

$$p_{U,V}(o|c) = \frac{\exp u_o^\top v_c}{\sum_{w\in\mathcal{V}} \exp u_w^\top v_c} \tag{4}$$

This may be familiar to you as the **softmax** function, which takes arbitrary scores (here, one for each word in the vocabulary, resulting from dot products) and produces a probability distribution where larger-scored things get higher probability. Note that the vector of probabilities over all words given a center word $p_{U,V}(\cdot \mid c) \in \mathcal{R}^{|\mathcal{V}|}$ is a lot like a row of our old co-occurrence matrix $\mathbf{X}_c$.

The story isn't over yet; this is just a model. How do we estimate the values of the parameters U, V? We learn to minimize the **cross-entropy loss** objective with the true distribution $P^*(O \mid C)$:

$$\min_{U,V} \ \mathbb{E}_{o,c}\big[-\log p_{U,V}(o \mid c)\big]. \tag{5}$$

This equation should be read as "minimize with respect to parameters U and V the expectation over values of o and c drawn from the distributions of O and V the negative-log probability under the (U, V)-model of that value of o given that value of c".

There is so much rich detail to get into here. How do we perform the min operation? How do we "get" the random variables O and C? Why the negative-log of the probability? Why is this so much better than co-occurrence counting? Can you tell why *not all distributions over o given c can be represented by this model?* (Should this be good? Bad? Surprising? Obvious?) For now, let's go through a few details about how to implement this in practice.

3.3 *Estimating a word2vec model from a corpus*

How do we train word2vec in practice? Specifying the word2vec model is relatively transparent from the math we've given above: one constructs matrices U and V and can write out the math of the probability. However, it may not yet be evident how to estimate the parameters: (1) how to calculate the expectation in Eqn 5 for a given value of U and V, and then (2) how to perform the minimization operation. Let's start with 1.

Word2vec empirical loss. Let D be a set of documents $\{d\}$ where each document is a sequence of words $w_1^{(d)}, \ldots, w_m^{(d)}$, with all $w \in \mathcal{V}$. Let $k \in \mathbb{N}_{++}$ be a positive-integer window size. Let's define how our center word random variable C and outside word r.v. O relate to this concrete dataset. O takes on the value of each word w_i in each document, and for each such w_i, the outside words are $\{w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}\}$. So, our Eqn 5 objective becomes:

$$L(U, V) = \sum_{d \in D} \sum_{i=1}^{m} \sum_{j=1}^{k} -\log p_{U,V}(w_{i-j}^{(d)} \mid w_i^{(d)}), \tag{6}$$

where you'll note we're taking the sum over (1) all documents of the sum over (2) all words in the document of the sum over (3) all words occuring in the window of the likelihood of the outside word given the center word.

Now, how do we do the minimization?

Gradient-based estimation At a high level, we try to find "good" U and V for the objective we've specified by starting with a relatively uninformed guess, and iteratively moving in the direction that locally best-improves the guess. This is done by *gradient-based* methods, a full description of which is out of scope for this note. Briefly, the gradient (think: derivative) $\nabla_U f$ of a scalar function f with respect to a parameter matrix U represents the direction to (locally) move U in in order to maximally increase the value of f. So, in practice, we do something like drawing the initial $U^{(0)}$ and $V^{(0)}$ randomly as $U, V \sim \mathcal{N}(0, 0.001)^{|\mathcal{V}| \times d}$ (matrices of independent draws from a zero-centered normal distribution with small variance), and then perform some number of iterations of the following process:

$$U^{(i+1)} = U^{(i)} - \alpha \nabla_U L(U^{(i)}, V^{(i)}). \tag{7}$$

This should be read as setting the value of U at iteration $i + 1$ as the value of U at the previous iteration, plus a small (α small) step in the direction that locally-best improves U with respect to the objective $L(U, V)$ we specified in Eqn 6.

Stochastic gradients. There's a crucial detail remaining here (beyond how to compute the gradient function $\nabla_U(\cdot)$, which we'll discuss later): computing $L(U, V)$ is exceptionally expensive, as it walks over the entire dataset. Instead of computing the objective exactly, we instead perform *stochastic* gradient-based optimization, approximating $L(U, V)$ using a few samples for each step of Eqn 7. We might do this by sampling documents $d_1, \dots, d_\ell \sim D$ and computing

$$\hat{L}(U,V) = \sum_{d_1,\dots,d_\ell} \sum_{i=1}^{m} \sum_{j=1}^{k} -\log p_{U,V}(w_{i-j}^{(d)} \mid w_i^{(d)}), \tag{8}$$

3.4 *Working through a gradient.*

How does a word2vec gradient step affect the parameters? Let's work out the math and build some intuition. In particular, we'll write out the partial gradient of the loss with respect to the parameter v_c for a single instance of a center word c. We start by writing out the gradient and passing the gradient operator through the sums:

$$\nabla_{v_c} \hat{L}(U,V) = \sum_{d \in D} \sum_{i=1}^{m} \sum_{j=1}^{k} -\nabla_{v_c} \log p_{U,V}\left(w_{i-j}^{(d)} \mid w_i^{(d)}\right), \tag{9}$$

intuitively, the gradient of all these terms in the sum is just the sum of their gradients. Let's work out the gradient of the probability, and for concision of notation we'll write $w_{i-j}^{(d)}$ as o again, and $w_i^{(d)}$ as c.

So let's take a single term of the sum and break it up with logarithm rules:

$$\begin{aligned} \nabla_{v_c} \log p_{U,V}(o \mid c) &= \nabla_{v_c} \log \frac{\exp u_o^\top v_c}{\sum_{i=1}^{n} u_w^\top v_c} && (10) \\ &= \underbrace{\nabla_{v_c} \log \exp u_o^\top v_c}_{PartA} - \underbrace{\nabla_{v_c} \log \sum_{i=1}^{n} \exp u_w^\top v_c}_{PartB} && (11) \end{aligned}$$

Part A. Let's differentiate Part A first, since it's easier.

$$\begin{aligned} \nabla_{v_c} \log \exp u_o^\top v_c &= \nabla_{v_c} u_o^\top v_c && \text{inverse operations} && (12) \\ &= u_o && \text{why?} && (13) \end{aligned}$$

To see why the last equality (marked "why?") holds, consider each individual dimension of v_c. The partial derivative of the output of $u_o^\top v_c$ is $\nabla_{v_{c,i}} u_o^\top v_c = \nabla_{v_{c,i}} \sum_i u_{o,i} v_{c,i} = u_{o,i}$. This is in turn because only one term of the sum depends on $v_{c,i}$, and $\nabla_{v_{c,i}} u_{o,i} v_{c,i} = u_{o,i}$ by single-variable calculus. When we stack a bunch of these single variable derivatives together, we get $[u_{o,1}, \dots, u_{o,d}] = u_o$, as written. The *shape* of the gradient is u_o, not $u_o^\top$ because of convention; by convention, we set the gradient of any object to be the shape of that object, which may involve some re-shaping.

Part B. Now let's differentiate Part B.

$$
\begin{aligned}
-\nabla_{v_c} \log \sum_{w=1}^{n} \exp u_w^\top v_c &= \frac{1}{\sum_{w=1}^{n} \exp u_w^\top v_c} \nabla_{v_c} \sum_{x=1}^{n} \exp u_x^\top v_c && \text{derivative of log; chain rule} \\
&= \frac{1}{\sum_{w=1}^{n} \exp u_w^\top v_c} \sum_{x=1}^{n} \nabla_{v_c} \exp u_x^\top v_c && \text{linearity of derivative} \\
&= \frac{1}{\sum_{w=1}^{n} \exp u_w^\top v_c} \sum_{x=1}^{n} \exp(u_x^\top v_c) \nabla_{v_c} u_w^\top v_c && \text{derivative of exponential; chain rule} \\
&= \frac{1}{\sum_{w=1}^{n} \exp u_w^\top v_c} \sum_{x=1}^{n} \exp(u_x^\top v_c) u_w && \text{derivative of dot product}
\end{aligned}
$$

Now we'll use this to come to a bit of insight. Let's put Part A and Part B together, and do a little bit of algebra:

$$
\begin{aligned}
u_o - \underbrace{\frac{1}{\sum_{w=1}^{n} \exp u_w^\top v_c}}_{\text{pull this under the sum}} \sum_{x=1}^{n} \exp(u_x^\top v_c) u_w &= u_o - \sum_{x=1}^{n} \underbrace{\frac{\exp(u_x^\top v_c)}{\sum_{w=1}^{n} \exp u_w^\top v_c}}_{\text{This is } p_{U,V}(x \mid c)} u_w \\
&= u_o - \underbrace{\sum_{x=1}^{n} p_{U,V}(x \mid c)}_{\text{This is an expectation}} \; u_w \\
&= u_o - \mathbb{E}[u_w] \\
&= \text{"observed" - "expected"},
\end{aligned}
$$

Intuitively, this all comes down to the last equation here; we have the vector for the word actually observed: u_c. We subtract from that the vector, intuitively, that the model expected—in the sense that it's the sum over all the vocabulary of the probability the model assigned to that word, multiplied by the vector that was assigned to that word. So, the v_c vector is updated to be "more like" the word vector that was actually observed than the word vector it expected.

If you didn't follow the math above, don't fear; I'd suggest going through it a few times and not rushing yourself. And if this was all dreadfully boring to you because you understood it quickly, use your newfound free time to help teach others!

3.5 Skipgram-negative-sampling

Now that we're doing stochastic estimates of our gradients, one remaining bottleneck of efficiency in estimating our word2vec model is computing the exact model probability $-\log p(U, V)(o \mid c)$ when computing $\hat{L}(U, V)$. For a given word, it is cheap to compute the **unnormalized score** $\exp(u_c \top v_o)$. However, it is expensive to compute the **partition function** (sum of scores for all words) in the denominator, since it requries a term for each word in the vocabulary.

Intuitively, what is the partition function doing, such that we might understand how to remove it? Let's repeat the softmax here:

$$p_{U,V}(o|c) = \frac{\overbrace{\exp u_o^\top v_c}^{\text{Affinity of word } c \text{ for context } o}}{\underbrace{\sum_{w\in\mathcal{V}} \exp u_o^\top v_w}_{\text{Partition function, or, normalization}}} \tag{14}$$

From a probabilistic perspective, the partition function guarantees a probability by normalizing the scores to sum to 1. (The exponential guarantees that the scores are non-negative.) From a learning perspective, the partition function "pushes down" on all the words other than the observed words. Put another way, the numerator of this equation encourages the model to make u_o more like v_c; the denominator encourages all the other u_w for $w \neq o$ less like v_c. The intuition of negative sampling is that **we don't need to push down on all the u_w all the time**, since that's where most of the cost comes from.

However, the actual skip-gram with negative sampling (SGNS) objective ends up being a bit more different; we'll write it here:

$$\overbrace{\log \sigma(u_o^\top v_c)}^{\text{Affinity of word } c \text{ for context } o} + \underbrace{\sum_{\ell=1}^{k} \left[\log \sigma(-u_\ell^\top v_c)\right]}_{\text{Push down everything else in expectation (replacement for partition fn)}} \tag{15}$$

where, where σ is the logistic function, and $u_\ell \sim p_{\text{neg}}$, which means u_ℓ is drawn from a distribution we haven't defined, called p_{neg}, Think of this for now like the uniform distribution over $\mathcal{V}$. What is this objective doing? It has two terms, just like how we've described the orginal skipgram. The first term encourages v_c and u_o to be more like each other, and the second term encourages v_c and u_ℓ for k random samples from the vocabulary to be less like each other. The intuition here is that if we randomly push down a few words at each step, then on average, things will work out sort of as if we always pushed down every word.

A Extra notes

A.1 Continuous Bag-of-Words

A.2 Singular Value Decomposition

References

[Batsuren et al., 2022] Batsuren, K., Goldman, O., Khalifa, S., Habash, N., Kieraś, W., Bella, G., Leonard, B., Nicolai, G., Gorman, K., Ate, Y. G., Ryskina, M., Mielke, S., Budianskaya, E., El-Khaissi, C., Pimentel, T., Gasser, M., Lane, W. A., Raj, M., Coler, M., Samame, J. R. M., Camaiteri, D. S., Rojas, E. Z., López Francis, D., Oncevay, A., López Bautista, J., Villegas, G. C. S., Hennigen, L. T., Ek, A., Guriel, D., Dirix, P., Bernardy, J.-P., Scherbakov, A., Bayyr-ool, A., Anastasopoulos, A., Zariquiey, R., Sheifer, K., Ganieva, S., Cruz, H., Karahóğa, R., Markantonatou, S., Pavlidis, G., Plugaryov, M., Klyachko, E., Salehi, A., Angulo, C., Baxi, J., Krizhanovsky, A., Krizhanovskaya, N., Salesky, E., Vania, C., Ivanova, S., White, J., Maudslay, R. H., Valvoda, J., Zmigrod, R., Czarnowska, P., Nikkarinen, I., Salchak, A., Bhatt, B., Straughn, C., Liu, Z., Washington, J. N., Pinter, Y., Ataman, D., Wolinski, M., Suhardijanto, T., Yablonskaya, A., Stoehr, N., Dolatian, H., Nuriah, Z., Ratan, S., Tyers, F. M., Ponti, E. M., Aiton, G., Arora, A., Hatcher, R. J., Kumar, R., Young, J., Rodionova, D., Yemelina, A., Andrushko, T., Marchenko, I., Mashkovtseva, P., Serova, A., Prud'hommeaux, E., Nepomniashchaya, M., Giunchiglia, F., Chodroff, E., Hulden, M., Silfverberg, M., McCarthy, A. D., Yarowsky, D., Cotterell, R., Tsarfaty, R., and Vylomova, E. (2022). UniMorph 4.0: Universal Morphology. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 840–855, Marseille, France. European Language Resources Association.

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. (2011). Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398.

[Firth, 1957] Firth, J. R. (1957). Applications of general linguistics. *Transactions of the Philological Society*, 56(1):1–14.

[Manning, 2022] Manning, C. D. (2022). Human Language Understanding & Reasoning. *Daedalus*, 151(2):127–138.

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

[Miller, 1995] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[Rong, 2014] Rong, X. (2014). word2vec parameter learning explained. *CoRR*, abs/1411.2738.

[Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.