CS 194-129 Deep Neural Networks
Spring 2018 Canny Lecture 12

1 Reading

Word2vec

The goal is to learn a representation for words in which similar words have similar representations. We can infer the similarity of words by the similarity of their contexts, i.e. if two words A and B often appear next to the same types of words, then it is likely that A and B are close in meaning.

Skip-gram

A Skip-gram model is optimized to find a representation for each word that will enable the identification of neighboring words. Given a sequence of words $w_1, ..., w_T$, the Skip-gram objective is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j} \mid w_t)$$

where c essentially denotes the window size around each word.
In its most basic form, $p(w_{t+j} \mid w_t)$ is defined via the softmax function

$$p(w_o \mid w_i) = -\frac{e^{(v'_{w_o})^T v_{w_i}}}{\sum_{w=1}^{W} e^{(v'_w)^T v_{w_i}}}$$

where v_w and v'_w are the "input" and "output" vector representations of w, and W is the number of words in the vocabulary. By "input" we refer to the central word; by "output" we refer to a potential neighbor of that central word.

However, it is intractable to deal with this formulation (gradient calculations scale linearly with W). Therefore, in practice we can approximate the full softmax with the *hierarchical* softmax. The hierarchical softmax only needs to evaluate about $\log_2(W)$ nodes in order to obtain the probability distribution. It represents the output layer as a binary tree, where the W words are the leaves and each node tracks the relative probabilities of its child nodes.

Alternatively, we can approximate the full softmax through Noise Contrastive Estimation (NCE). This technique is based on the idea that a good model should be able to distinguish data from noise by using logistic regression. As a simplification to NCE, we can replace every $\log p(w_o \mid w_i)$ in the Skip-gram objective with the negative sampling objective, such that the task becomes "distinguish the target word w_o from draws from the noise distribution $P_n(w)$ using logistic regression, where there are k negative samples for each data sample."

Skip-gram Extensions

Some words occur extremely frequently (think "the"), and end up providing a lot less information per-sample than rarer words. The model will benefit much more from the observation that "computer science" co-occurs with "Berkeley" than from the observation that "computer science" co-occurs with "the." After all, articles like "the" co-occur all the time with everything. Meanwhile, vector representations of common words don't change very much having seen them a lot.

Therefore, we can *subsample* these frequent words, i.e. don't include a word in the training with a probability that is based on that word's frequency.

Learning Phrases

Word2vec methods often struggle with idiomatic expressions and phrases; generally, it is not clear that "San Francisco" and "Chronicle" should have anything to do with each other. Therefore, we can use vectors to represent the whole phrases instead of splitting them into individual words. We shall first identify phrases using a data-driven approach, and thereafter treat the entire phrases as tokens.

A phrase has "a meaning that is not a simple composition of the meanings of its individual words." To find a phrase, we look for words that appear frequently right next to each other, but infrequently in other contexts.

Compositionality

We can additively combine our vector representations and attain meaningful output. Let `vec` be a function mapping words to vectors. Then it is the case that `vec`("Madrid") - `vec`("Spain") + `vec`("France") is very close to `vec`("Paris"). It is also the case that `vec`("Germany") + `vec`("capital") is close to `vec`("Berlin"). These results suggest that the encoding has managed to capture at least some of the language's underlying meaning!

This works because each vector represents a context distribution for a word. Each value is related logarithmically to the probability computed by the output layer (because it gets exponentiated during the softmax function). Therefore the sum of two word vectors equates to the product of the two context distributions, and acts as the AND function. After the sum, only words that were assigned high probabilities by *both* vectors will still have high probabilities.

For example, if "Volga River" occurs frequently in the context of both "Russian" and "river," the sum of the "Russian" and "river" feature vectors should produce something close to the "Volga River" vector.

As an aside, we can evaluate vectors for analogical reasoning by checking whether, e.g., `vec`("Berlin") - `vec`("Germany") + `vec`("France") = `vec`("Paris") according to the cosine distance. There are both syntactic analogies ("quick" is to "quickly" as "slow" is to "slowly") and semantic analogies ("Los Angeles" is to "Los Angeles Lakers" as "Cleveland" is to "Cleveland Cavaliers"). Note: this all works because the representations exhibit a linear structure.

Skip-Thought Vectors

The goal of this paper is to learn generic sentence representations (i.e. representations that are not tuned for a specific task). To do so, the authors abstract the skip-gram model to the sentence level. Instead of defining a representation by predicting the context of words, they define a representation by predicting the context of sentences (i.e. the sentences around a sentence). Their model is called **skip-thoughts**; the vectors produced are called **skip-thought vectors**.

Generalization stems from the use of a highly varied training corpus of books. The idea is that the training set will not be biased toward any particular domain.

Approach

Skip-thoughts follows an encoder-decoder architecture, where the end result is the sentence encoder (which maps an English sentence into a vector). The *decoder* attempts to generate the surrounding sentences of an encoded sentence. Note: a good encoder should map sentences that are similar semantically and syntactically to vectors that are similar distance-wise (according to cosine similarity, for example).

2 Lecture

Today we'll focus on text semantics and embedding methods (methods that take text and produce a corresponding distributed representation – i.e. a vector representation).

Text Semantics

Semantics is the study of the *meaning* of language. Note: there are other things to study, such as structure, but those things are not the focus of this course. As it stands, there are two main approaches to semantics:

- **Propositional/formal semantics:** a mapping from natural language text to formal predicates (propositional logic) intended to capture the meaning of the sentence. As an example, "dog bites man" might translate to "bites(dog, man)" where "bites($\cdot$, $\cdot$)" is a binary relation.
- **Vector representation:** where texts are *embedded* into a high-dimensional space. As an example, "dog bites man" might translate to $(0.2, -0.3, 1.5, ...) \in \mathbb{R}^n$. Sentences similar in meaning should be geometrically close as vectors in the embedded space.

In deep networks, we usually use distributed representations. "Distributed" means that the representation is spread out across the activations of some neurons. This is pretty much universally the case for NLP and vision models.

Our goal is to produce these representations from the input text.

Vector Embedding of Words

We can begin by embedding individual words (and for now just add them up to produce the semantics of a longer string). We will take each word and map it to a real-valued vector. All vectors are the same dimension, so we can compose them. Note: bag-of-words representations, i.e. *unordered* representations, are lacking because `vec`("dog bites man") would have the same representation as `vec`("man bites dog").

How do we define word similarity? We use the idea of *paradigmatic similarity*, which posits that words are similar in meaning if you can take a collection of text and replace one word with the other word. If you can do that some of the time, then the words are at least similar in meaning. Observe that this definition supports unsupervised learning: we can cluster or embed words according to their contexts only.

The context becomes the feature vector for a word.

Matrix Factorization

Latent semantic analysis (LSA) processes documents in a bag-of-words format. It constructs a count matrix T, where T_{ij} is the count of word j in document i. The matrix will be extremely sparse; most entries will be 0. We can then apply a form of PCA by approximating T as a product of two matrices $U^T V$.

(Note: $T \in \mathbb{R}^{N,M}$, $U^T \in \mathbb{R}^{N,K}$, and $V \in \mathbb{R}^{K,M}$. N is the number of documents, M is the number of word features, and K is the latent dimension size. K will be much less than M and N.)

So we end up with two matrices: one a mapping from the documents to the latent space, and the other a mapping from the word features to the latent dimensions. By PCA, this factorization is the best L_2 approximation to T. The factors are the rows of V, and encode similar *entire document contexts*.

If U and V are orthogonal, S is a diagonal matrix of singular values, and t is a document (a row of T), then $v = Vt$ is an embedding of the document in the latent space and $t' = U^T v = U^T V t$ is the decoding of the document from its embedding. Such an SVD factorization gives the best possible approximation to T in the squared error sense, and hence the best possible document reconstructions t' (as bags of words i.e. counts).

LSA is therefore an autoencoding method, and is trained on the loss for recreating the inputs.

Word2vec

To consider an entire document as the context is to lose too much of the meaning of the words.

Instead of using entire documents, Word2vec uses words only a few positions away from each center word. These pairs of (center word, context word) are called **skip-grams**. Word2vec considers all words as center words *and* all of their context words. This allows us to consider only the most relevant context for each word (i.e. its local surroundings).

In this setup, we are separately predicting output words. We treat the center word as the input and the surrounding context words as the output. Typically, there is variable weighting for context words in the loss based on distance; the model will try harder to predict closer words than faraway words.

Word2vec does a softmax prediction of the output word given the input word (using the skip-gram model just described). The probability is therefore just

$$p(j \mid i) = \frac{e^{u_j^T v_i}}{\sum_{k=1}^{V} e^{u_k^T v_i}}$$

where j is the output word, i is the input word, u is the output embedding vector, and v is the input embedding vector. j ranges over a context of $\pm$ 3-5 positions around the input word. (When we say "input word," we mean the center word, and when we say "output word" we mean the context word.)

We use different embedding vectors (u & v) because center words and context words behave differently. Also, since it's very expensive to normalize the softmax over all words, approximations are almost always used.

Local contexts capture more information about relations and properties than LSA. This allows us to do analogies based on short sequence/attribute/type compositions. If we start with a couple of words such as Japan and Tokyo, we can take the vector from Japan's embedding to Tokyo's embedding and this vector should encode the relationship between "country" and "capital." So if we add this relationship vector to `vec`("Greece"), for example, we should get `vec`("Athens").

More compositions:

- `vec`("woman") - `vec`("man") $\approx$ `vec`("aunt") - `vec`("uncle")
- `vec`("woman") - `vec`("man") $\approx$ `vec`("queen") - `vec`("king")

`vec`("woman") - `vec`("man") can be thought of as the gender change vector.

GloVe

Let C_{ij} be the number of times that the word j occurs in the context of word i (so C is the local context co-occurrence matrix). GloVe minimizes

$$J(\theta) = \sum_{i,j=1}^{V} f(C_{ij})(u_i^T v_j + b_i + \tilde{b}_j - \log C_{ij})^2$$

(i.e. an L2 loss which uses the log of the counts), in a way **optimizing specifically for analogies and vector-offset relationships between words**. u_i is the word embedding, v_j is the context word embedding, b_i and $\tilde{b}_j$ are bias vectors, and $f(\cdot)$ is a function which satisfies

- $f(0) = 0$ (so we can treat the entire result as 0 when $C_{ij} = 0$, and not have to worry about $\log C_{ij}$)
- $f(x)$ is non-decreasing
- $f(x)$ saturates (not too large for large x)

Skip-Thought Vectors

So far, our models have embedded texts as the sum of their words (i.e. via *lexical semantics*, which focuses on the meaning of individual words, and not *compositional semantics*, where the meaning depends both on the words and how they're combined). We want to model text structure as well as word meanings! For this we turn to skip-thought vectors.

Skip-thought vectors is a method for producing a semantic representation of an entire sentence, or even a paragraph. It's another kind of autoencoding method where we take a sequence-to-sequence RNN and use it to predict the next and previous sentences. It encodes an input sentence, and decodes into the next and previous sentences.

The idea is that adjacent sentences are semantically similar. However, further than one sentence away the similarity tends to drop off too much, so we typically only look at the direct neighbors of a sentence.

Intuition: the model should be trying as hard is it can to predict the before and after sentence. Therefore it should be learning to capture (in the encoding) as much of the meaning of the sentence as it can; this is the entirety of the information that's passed to the decoders.

Siamese Models

A Siamese model is a deep representation that's explicitly geared toward matching two different sets of text. This is done instead of training the sequence model to just predict adjacent sentences (which has nothing to do with comparing the sentences).

Specifically, the method has us build a corpus of (a, b) similar pairs of sentences. Then we run these pairs of sentences through two LSTMs with tied weights (both of which compute an embedding) and exponentiate a distance measure to form the loss. Hence we are explicitly optimizing for comparing sentences. The additional challenge is that we suddenly require all of these similar sentence pairs.

References

[1] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In Advances on Neural Information Processing Systems, 2013.

[2] R. Kiros, Y. Zhu, R.R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-Thought Vectors. In NIPS, 2015.