ANLP Lecture 22
Lexical Semantics with Dense Vectors

Shay Cohen
(Based on slides by Henry Thompson and Dorota Glowacka)

4 November 2019

Last class

Represent a word by a context vector

- Each word x is represented by a vector $\vec{v}$. Each dimension in the vector corresponds to a context word type y
- Each v_i measures the level of association between the word x and context word y_i

Pointwise Mutual Information

- Set each v_i to $\log_2 \frac{p(x,y_i)}{p(x)p(y_i)}$
- Measures "colocationness'
- Vectors have many dimensions and very sparse (when PMI < 0 is changed to 0)

Similarity metric between $\vec{v}$ and another context vector $\vec{w}$:

- The cosine of the angle between $\vec{v}$ and $\vec{w}$: $\frac{\vec{v}\vec{w}}{|\vec{v}||\vec{w}|}$

Today's Lecture

- How to represent a word with vectors that are short (with length of 50 – 1,000) and *dense* (most values are non-zero)
- Why short vectors?
  - Easier to include as features in machine learning systems
  - Because they contain fewer parameters, they generalise better and are less prone to overfitting

Roadmap for Main Course of Today

- Skip-gram models - relying on the idea of pairing words with dense context and target vectors. If a word co-occurs with a context word w^c, then its target vector should be similar to the context vector of w^c
- The computational problem with skip-gram models
- An example solution to this problem: negative sampling skip-grams

Before the Main Course, on PMI and TF-IDF

- PMI is one way of trying to detect *important* co-occurrences based on divergence between observed and predicted (from unigram MLEs) bigram probabilities
- A different take: a word that is common in only *some* contexts carries more information than one that is common everywhere

How to formalise this idea?

TF-IDF: Main Idea

Key Idea: Combine together the frequency of a term in a context (such as document) with its relative frequency overall in all documents.

- This is formalised under the name **tf-idf**
  - **tf** Term frequency
  - **idf** Inverse document frequency
- Originally from Information Retrieval, where there a lots of documents, often with lots of words in them
- Gives an "importance" level of a term in a specific context

TF-IDF: Combine Two Factors

- tf: term frequency of a word t in document d:

$$\mathrm{tf}(t,d) = \begin{cases} 1 + \log \mathrm{count}(t,d) & \textit{if } \mathrm{count}(t,d) > 0 \\ 0 & \textit{otherwise} \end{cases}.$$

frequency count of term i in document d

- Idf: inverse document frequency:

$$\mathrm{idf}(t) = \log\left(\frac{N}{\mathrm{df}_t}\right)$$

- N is total # of docs in collection
- df_t is # of docs that have term t
- Terms such as *the* or *good* have very low idf
  - because $\mathrm{df}_t \approx N$
- tf-idf value for word t in document d:

$$\mathrm{tfidf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Summary: TF-IDF

- Compare two words using tf-idf cosine to see if they are similar
- Compare two documents
  - Take the centroid of vectors of all the terms in the document
  - Centroid document vector is:

$$d = \frac{t_1 + t_2 + \cdots + t_k}{k}$$

TF-IDF and PMI are Sparse Representations

- TF-IDF and PMI vectors
  - have many dimensions (as the size of the vocabulary)
  - are sparse (most elements are zero)
- Alternative: dense vectors, vectors which are
  - short (length 50–1000)
  - dense (most elements are non-zero)

Neural network-inspired dense embeddings

- Methods for generating dense embeddings inspired by neural network models
  Key idea: *Each word in the vocabulary is associated with two vectors: a context vector and a target vector. We try to push these two types of vectors such that the target vector of a word is close to the context vectors of words with which it co-occurs.*
- This is the main idea, and what is important to understand. Now to the details to make it operational...

Skip-gram modelling (or Word2vec)

- Instead of counting how often each word occurs near "apricot"
- Train a classifier on a binary prediction task:
  - Is the word likely to show up near "apricot"?
  - A by-product of learning this classifier will be the context and target vectors discussed.
  - These are the *parameters* of the classifier, and we will use these parameters as our word embeddings.
- No need for hand-labelled supervision - use text with co-occurrence

Prediction with Skip-Grams

- Each word *type* w is associated with two dense vectors: $v(w)$ (target vector) and $c(w)$ (context vector)
- Skip-gram model predicts each neighbouring word in a context window of L words, e.g. context window $L = 2$ the context is $[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$
- Skip-gram calculates the probability $p(w_k|w_j)$ by computing dot product between context vector $c(w_k)$ of word w_k and target vector $v(w_j)$ for word w_j
- The higher the dot product between two vectors, the more similar they are

Prediction with Skip-grams

- We use softmax function to normalise the dot product into probabilities:

  $p\left(w_k|w_j\right) = \frac{\exp\left(c(w_k)\cdot v(w_j)\right)}{\sum_{w \in V} \exp\left(c(w)\cdot v(w_j)\right)}$

  where V is our vocabulary.
- If both fruit and apricot co-occur with delicious, then v(*fruit*) and v(*apricot*) should be similar both to c(*delicious*), and as such, to each other
- Problem: Computing the denominator requires computing dot product between each word in V and the target word w_j, which may take a long time

Skip-gram with Negative Sampling

- Problem with skip-grams: Computing the denominator requires computing dot product between each word in V and the target word w_j, which may take a long time

Instead:

- Given a pair of target and context words, predict + or - (telling whether they co-occur together or not)
- This changes the classification into a binary classification problem, no issue with normalisation
- It is easy to get example for the + label (words co-occur)
- Where do we get examples for - (words do not co-occur)?
- Solution: randomly sample "negative" examples

Skip-gram with Negative Sampling

- Training sentence for example word *apricot*:
  lemon, a tablespoon of apricot preserves or jam
- Select $k = 2$ noise words for each of the context words:

cement	bacon	dear	coaxial	apricot	ocean	hence	never	puddle
n_1	n_2	n_3	n_4	w	n_5	n_6	n_7	n_8

- We want noise words w^{n_i} to have a low dot-product with target embedding w
- We want the context word to have high dot-product with target embedding w

Skip-Gram Goal

To recap:

- Given a pair (w^t, w^c) = target, context
  - (apricot, jam)
  - (apricot, aardvark)

  return probability that w^c is a real context word:
  - $P(+|w^t, w^c)$
  - $P(-|w^t, w^c) = 1 - P(+|w^t, w^c)$
- Learn from examples (w^t, w^c, ℓ) where $\ell \in \{+, -\}$ and the negative examples are obtained through sampling

How to Compute $p(+|w^t, w^c)$?

Intuition:

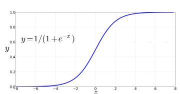
- Words are likely to appear near similar words
- Again use dot-product to indicative positive/negative label, coupled with logistic regression. This means

$$P(+|w^t, w^c) = \frac{1}{1 + \exp\left(-v(w^t) \cdot c(w^c)\right)}$$

$$P(-|w^t, w^c) = 1 - P(+|w^t, w^c) = \frac{\exp\left(-v(w^t) \cdot c(w^c)\right)}{1 + \exp\left(-v(w^t) \cdot c(w^c)\right)}$$

The function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



is also referred to as "the sigmoid"

Skip-gram with Negative Sampling

So, given the learning objective is to maximise:

$\log P(+|w^t, w^c) + \sum_{i=1}^{k} \log P(-|w^t, w^{n_i})$

where we have k negative-sampled words $w^{n_1}, \cdots, w^{n_k}$

- We want to maximise the dot product of a word target vector with a true context word context vector
- We want to minimise over all the dot products of a target word with all the untrue contexts
- How do we maximise this learning objective? Using gradient descent

How to Use the Context and Target Vectors?

- After this learning process, use:
  - $v(w)$ as the word embedding, discarding $c(w)$
  - Or the concatenation of $c(w)$ with $v(w)$

A good example of representation learning: through our classifier setup, we learned how to represent words to fit the classifier model to the data

Food for thought: are $c(w)$ and $v(w)$ going to be similar for each w? Why?

$$v(\textit{fruit}) \rightarrow c(\textit{delicious}) \rightarrow v(\textit{apricot}) \rightarrow c(\textit{fruit})$$

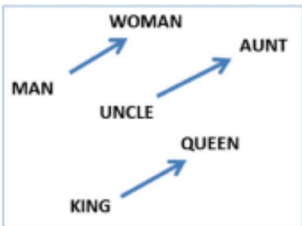
Some Real Embeddings

Examples of the closest tokens to some target words using a phrase-based extension of the skip-gram algorithm (Mikolov et al. 2013):

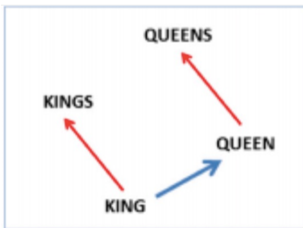
Redmond	Havel	ninjutsu	graffiti	capitulate
Redmond Wash	Vaclav Havel	ninja	spray paint	capitulation
Redmond Washington	President Vaclav Havel	Martial arts	graffiti	capitulated
Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

Properties of Embeddings

Offsets between embeddings can capture relations between words, e.g. vector(king)+ (vector(woman) − vector(man)) is close to vector(queen)



Offsets can also capture grammatical number



Summary

skip-grams (and related approaches such as **continuous bag of words** (CBOW)) are often referred to as **word2vec**

- Code available online - try it!
- Very fast to train
- Idea: predict rather than count