Word Vectors Representations

Cristian Cardellino • Milagro Teruel

Aprendizaje Profundo • Diplodatos 2019
Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

Credits

Slides adapted from the Stanford University course
CS224d - Deep Learning for Natural Language Processing

Word Representations

How to represent the meaning of a word?

- **Meaning:** Idea represented by a word (or phrase).
- Represent meaning using a *taxonomy*, like WordNet, with hypernyms (is-a) relationships and synonyms.
  - New words need update of the resource.
  - Needs human labor to create or adapt.
  - Word similarity is hard to compute.

How to represent a word?

- **Common denominator** in any NLP task (from synonym finding to question answering).
- Any model in NLP needs an **input representation of a word**.
- Much of the NLP work treats words as **atomic symbols**.
- To perform well on most NLP tasks we need to have some **notion of similarity and difference between words**.
- **Word vectors can encode this ability** in the vectors themselves (using distance measures such as Euclidean, Cosine, etc).

Discrete Word Vectors

Discrete Word Vectors

- First approximation: **one-hot vector**.
- Treats the words as **atomic symbols**.
- For each word in the vocabulary we have one vector.
- In vector space terms, is a vector with one **1** and a lot of zeroes.

Discrete Word Vectors: Example

Given a corpus: "The cat sat on the mattress", it has a vocabulary $V = \{cat, mattress, on, sat, the\}$. Then, we can encode $|V|$ number of vectors for each word $w^{(i)} \in \mathbb{R}^{|V|}$ ($|V| = 5$ in this example).

Vectors for: "The cat sat on the mattress"

$$w^{cat} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, w^{mattress} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, w^{on} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, w^{sat} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, w^{the} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

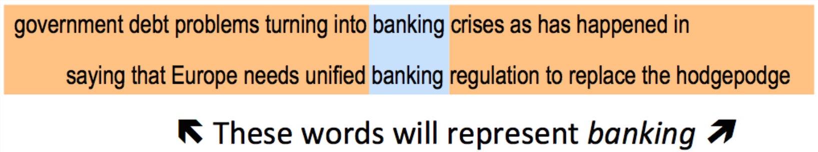
Problems with Discrete Representations

- English has an estimated **13 million tokens** for the language.
- The **dimensionality** of the vector can be extremely large.
- All the **vectors are sparse**, then:
  $(w^{cat})^T w^{feline} = (w^{cat})^T w^{hotel} = 0$

Distributional Representations

Distributional similarity based representations

- Idea: Represent a word by **means of its neighbours**.
- One of the **most successful ideas** in modern statistical NLP.



How to make neighbours represent a word?

- Use a **co-occurrence matrix** X.
- Can be of the **full document or of a window**.
- Word-Document Matrix: Gives general topics (all sport terms will have similar entries). Leads to "Latent Semantic Analysis".
- Word-Word Matrix: An affinity matrix. It captures syntactic (PoS) and semantic information of the words.

Word-Word Co-occurrence Matrix

- Window length is usually in $[5, 10]$.
- Symmetric (irrelevant whether the context is to the left or right).
- $X \in \mathbb{R}^{|V| \times |V|}$
- X_{ij} represents the co-occurrence between $w^{(i)}$ and $w^{(j)}$ within the window.

Word-Word Co-occurrence Matrix Example

Given the corpus:

- I enjoy flying.
- I like NLP.
- I like deep learning.

We have a word-word co-occurrence matrix with window of size 1:

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Problems with Co-Occurrence Matrix

- This first approximation for co-occurrence matrix **increases with the vocabulary**.
- It ends up being **very high-dimensional** and requires a lot of storage.
- We have to deal with **sparsity issues** (due to Zipf Law) $\rightarrow$ Models are less robust.
- Idea: *store most of the important information* in a fixed, small number of dimensions: **a dense vector**.
  - How to reduce dimensionality?

Word Vectors via Matrix Factorization

Singular Value Decomposition on X

- We generate the **co-occurrence matrix** X.
- We apply SVD (**Singular Value Decomposition**) on X to get $X = USV^T$
- We observe the **singular values** (the diagonal matrix S) and **cut them off at some index** k.
- We take $U_{1:|V|,1:k}$ as the **word embedding matrix** of k-dimensional word vectors.
- Each word is represented by a row in the matrix. The rows of V represent the words as context.
- In all subsequent models (e.g. deep learning models), **the word is represented by a dense vector**.

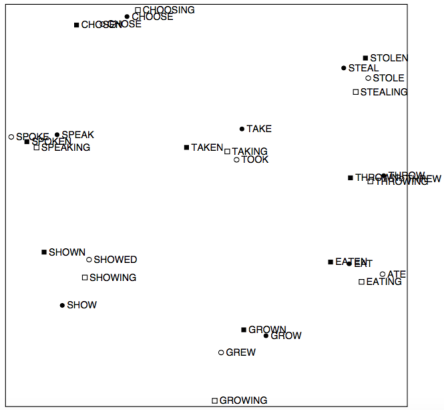
Applying SVD to X

$$\begin{array}{cc} & |V| \\ |V| & \begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix} \end{array} = \begin{array}{cc} & |V| \\ |V| & \begin{bmatrix} - & u_1 & - \\ - & u_2 & - \\ & \vdots & \end{bmatrix} \end{array} \begin{array}{cc} & |V| \\ |V| & \begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \end{array} \begin{array}{cc} & |V| \\ |V| & \begin{bmatrix} | & | & \\ v_1 & v_2 & \cdots \\ | & | & \end{bmatrix} \end{array}$$

We reduce by selecting the first k singular vectors.

$$\begin{array}{cc} & k \\ |V| & \begin{bmatrix} & & \\ & \hat{X} & \\ & & \end{bmatrix} \end{array} = \begin{array}{cc} & k \\ |V| & \begin{bmatrix} - & u_1 & - \\ - & u_2 & - \\ & \vdots & \end{bmatrix} \end{array} \begin{array}{cc} & k \\ k & \begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

Semantic Patterns in Vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence Rohde et al. 2005.

Problems with SVD

- **Function words** (the, he, has) are too frequent.
  - **Ignore them** or use an upper bound.
  - Use **positive pointwise mutual information**.
  - Use **positive Pearson correlation**.
- The **dimension of the matrix changes** as new words are added to the corpus. This changes the size of the vocabulary $|V|$.
- The **matrix is extremely sparse** (most of the words don't co-occur).
- The **matrix is very high-dimensional** in general.
- The cost to perform SVD is **quadratic**.

Probabilistic Word Vectors

Directly Learn Low Dimensional Vectors

- Don't store global information.
- Create a model **that learns one iteration at a time**.
- The model **encodes the probability** of a word given a context C.
- Possible approach: **Language models**.
    - This will give a **correct sentence high probability**.

Language models

- In the **unigram model** we assume each word occurrence is independent.

$$P(w^{(1)}, w^{(2)}, \ldots, w^{(n)}) = \prod_{i=1}^{n} P(w^{(i)})$$

  - **Naive solution**. Next word is contingent upon previous sequence of words.
- In the **bigram model**, the probability of a word depends on the previous one.

$$P(w^{(1)}, w^{(2)}, \ldots, w^{(n)}) = \prod_{i=2}^{n} P(w^{(i)}|w^{(i-1)})$$

  - Still naive, but **works well**. We are getting more on what we want to learn.

Neural Word Vectors

word2vec: Using Neural Networks to Learn Word Vectors

- Idea: **Predict surroundings** of every word.
- Is faster, **can easily incorporate new sentences/documents** to the vocabulary.
- Two possible variations.
  - Continuous Bag of Words (CBOW): Given a context **predict the center word**.
  - Skip-Gram Model: Given a center word **predict its context**.

Parameters of the Models

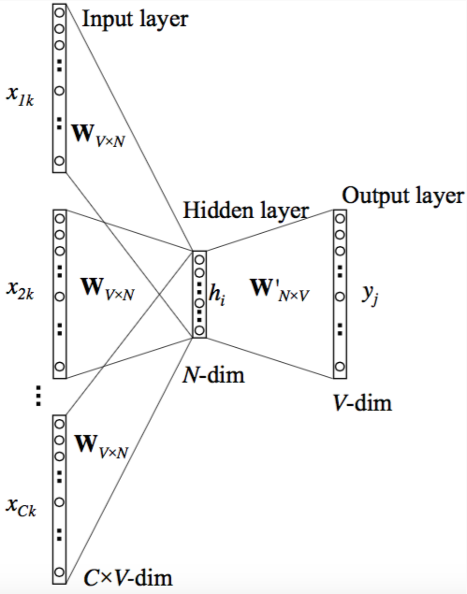
- We have a vocabulary $V = w^{(1)}, w^{(2)}, \ldots, w^{(m)}$ ($|V| = m$), and a symmetric context C.
- For each word $w^{(i)}$ we have a one-hot vector $x^{(i)} \in \mathbb{R}^m$.
- We create two matrices $W^{(1)} \in \mathbb{R}^{m \times n}$ and $W^{(2)} \in \mathbb{R}^{n \times m}$. In this case n is the dimension of our learned vectors.
- $u^{(i)} \in \mathbb{R}^n$ is the i-th row of matrix $W^{(1)}$, and represents the *embedded input vector* of the word $w^{(i)}$.
- $v^{(j)} \in \mathbb{R}^n$ is the j-th column of matrix $W^{(2)}$, and represents the *embedded output vector* of the word $w^{(j)}$.
- We learn 2 vector representations for every word $w^{(i)}$.

Continuous Bag of Words Model (CBOW)

Continuous Bag of Words: Forward Step

- We want to get the center word $w^{(i)}$ given the context words $(w^{(i-C)}, \ldots, w^{(i-1)}, w^{(i+1)}, \ldots, w^{(i+C)})$.
- Generate the one-hot word vectors $(x^{(i-C)}, \ldots, x^{(i-1)}, x^{(i+1)}, \ldots, x^{(i+C)})$.
- Get the embedded input vectors for each one-hot encoding $(u^{(i-C)} = x^{(i-C)T}W^{(1)}, \ldots, u^{(i-1)} = x^{(i-1)T}W^{(1)}, u^{(i+1)} = x^{(i+1)T}W^{(1)}, \ldots, u^{(i+C)} = x^{(i+C)T}W^{(1)})$.
- Average the vectors to get $h = \frac{u^{(i-C)}+u^{(i-C+1)}+\cdots+u^{(i+C)}}{2C}$.
- Generate a score vector $z = h^T W^{(2)}$.
- Turn the scores into a probability array, $\hat{y} = \text{softmax}(z)$.
- We desire our probabilities generated, $\hat{y}^{(i)}$, to match the true probabilities, $y^{(i)}$. That is, our one-hot vector of the center word $w^{(i)}$.

Continuous Bag of Words: Visualization



Continuous Bag of Words: Learning the Weight Matrices

- Question: How do we **learn the values of the matrices $W^{(1)}$ and $W^{(2)}$**?
  - We use an **objective function** (also known as cost/loss function).
- What do we want? **Maximize the probability of a true event**.
- Use cross-entropy, i.e. the negative log-likelihood of the true labels given a probabilistic classifier's prediction.

$$H(y, \hat{y}) = -\sum_{j=1}^{m} y_j \log(\hat{y_j})$$

Continuous Bag of Words: Objective Function

- Given a center word $w^{(i)}$ we want to predict, and a one-hot vector $y^{(i)}$ for the word, cross entropy simplifies to:

$$H(y, \hat{y}) = -y_i \log(\hat{y}_i)$$

- Then, we want to minimize our loss function:

$$\begin{aligned} J &= -\log P(w^{(i)}|w^{(i-C)}, \ldots, w^{(i-1)}, w^{(i+1)}, \ldots, w^{(i+C)}) \\ &= -\log P(v^{(i)}|h) \\ &= -\log \frac{\exp(v^{(i)T} h)}{\sum_{k=1}^{m} \exp(v^{(k)T} h)} \\ &= -v^{(i)T} h + \log \sum_{k=1}^{m} \exp(v^{(k)T} h) \end{aligned}$$

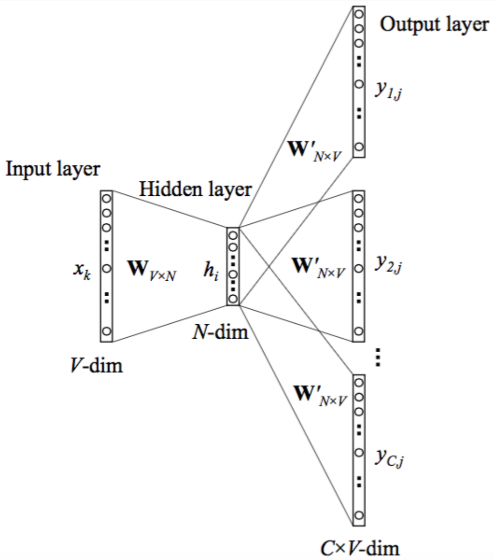
- We need to get the gradients of each of the vectors $v^{(i)}$ and $u^{(j)}$ for the gradient descent algorithm with backpropagation.

Skip-Gram Model

Skip-Gram: Forward Step

- We want to get the context words $(w^{(i-C)}, \dots, w^{(i-1)}, w^{(i+1)}, \dots, w^{(i+C)})$ given the center word $w^{(i)}$.
- Generate the one-hot word vector $x^{(i)}$.
- Get the embedded input vector for the one-hot encoding $u^{(i)} = x^{(i)T} W^{(1)}$.
- There's no average, we set $h = u^{(i)}$.
- Generate $2C$ score vectors $(z^{(i-C)}, \dots, z^{(i-1)}, z^{(i+1)}, \dots, z^{(i+C)})$ with $z^{(j)} = h^{(T)} W^{(2)}$.
- Turn each of the scores into probabilities arrays, $\hat{y}^{(j)} = \text{softmax}(z^{(j)})$.
- We desire our probabilities vectors generated, $(\hat{y}^{(i-C)}, \dots, \hat{y}^{(i-1)}, \hat{y}^{(i+1)}, \dots, \hat{y}^{(i+C)})$, to match the true probabilities, $(y^{(i-C)}, \dots, y^{(i-1)}, y^{(i+1)}, \dots, y^{(i+C)})$. That is, our one-hot vectors for each of the context words.

Skip-Gram: Visualization



Skip-Gram: Learning the Weight Matrices

- We also need an **objective function** to learn the weight matrices $W^{(1)}$ and $W^{(2)}$.
- A key difference is to invoke a **Naive Bayes** assumption.
- Each of the **probabilities** of the outputs given a center word is **independent**.
- We want to **maximize the log probability** of any context word given the current center word.

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

Skip-Gram: Objective Function

- A way to maximize the function $J(\theta)$ is to minimize the negative log-likelihood.

$$
\begin{aligned}
J &= -\log P(w^{(i-C)}, \dots, w^{(i-1)}, w^{(i+1)}, \dots, w^{(i+C)} | w^{(i)}) \\
&= -\log \prod_{-C \le j \le C, j \ne 0} P(w^{(i+j)} | w^{(i)}) \\
&= -\log \prod_{-C \le j \le C, j \ne 0} P(v^{(i+j)} | u^{(i)}) \\
&= -\log \prod_{-C \le j \le C, j \ne 0} \frac{\exp(v^{(i+j)T} h)}{\sum_{k=1}^{m} \exp(v^{(k)T} h)} \\
&= - \sum_{-C \le j \le C, j \ne 0} v^{(i+j)T} h + 2C \log \sum_{k=1}^{m} \exp(v^{(k)T} h)
\end{aligned}
$$

- As in CBOW we need to get the gradients of each of the vectors $v^{(j)}$ and $u^{(i)}$.

Negative Sampling

Negative Sampling

- If we look at the objective function, it **sums over all** $|V|$.
- **Any update or evaluation we do is** $O(|V|)$, which is in the millions (if not billions).
- Idea: For every training step, instead of loop over the entire vocabulary, we **sample several negative examples**.
- Use a **noise distribution** $P_n(w)$ whose probabilities match the ordering of the frequency of the vocabulary.

Mikolov's Negative Sampling

- Based on Skip-Gram, **changing the objective**.
- Considering a pair (w, c) of word and context. Does it comes from the training data?
  - $P(D = 1|w, c)$ is the probability that (w, c) came from the corpus data.
  - $P(D = 0|w, c)$ is the probability that (w, c) didn't come from the training data.
  - We model $P(D = 1|w, c)$ with the sigmoid function:

$$P(D = 1|w, c) = \frac{1}{1 + \exp(-v_c^T v_w)}$$

- We build a new objective function.
  - Maximize $P(D = 1|w, c)$ if (w, c) are in the corpus.
  - Maximize $P(D = 0|w, c)$ if (w, c) are not in the corpus.
- Take a maximum log-likelihood approach of these two probabilities.

Mikolov's Negative Sampling: Parameter Estimation

Given θ, the parameters of the model, we have:

$$
\begin{aligned}
\theta &= \mathsf{argmax}_\theta \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta) \\
&= \mathsf{argmax}_\theta \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1 - P(D=1|w,c,\theta)) \\
&= \mathsf{argmax}_\theta \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1 - P(D=1|w,c,\theta)) \\
&= \mathsf{argmax}_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c^T v_w}} + \sum_{(w,c)\in \tilde{D}} \log\left(1 - \frac{1}{1+e^{-v_c^T v_w}}\right) \\
&= \mathsf{argmax}_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c^T v_w}} + \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1+e^{v_c^T v_w}}\right)
\end{aligned}
$$

Mikolov's Negative Sampling: Objective Function

- In the previous equations $\tilde{D}$ is a **false corpus**.
- **Unnatural sentences** should have **low probability** of occurrence.
- The final objective function is defined:

$$J = -\log \sigma(v^{(j)} h) + \sum_{k=1}^{K} \log \sigma(\tilde{v}^{(k)} h)$$

- Where $\{\tilde{v}^{(k)} | k = 1, \ldots, K\}$ are sampled from $P_n(w)$.
- What is $P_n(w)$?
  - The approach taken by Mikolov: Unigram Model raised to the power of $\frac{3}{4}$. Why is this?
    - is: $0.9^{\frac{3}{4}} = 0.92$.
    - Constitution: $0.09^{\frac{3}{4}} = 0.16$.
    - bombastic: $0.01^{\frac{3}{4}} = 0.032$.
  - "bombastic" is 3 times more likely to be sampled now, but "is" only went up marginally.

Learning the Word Vectors

Learning the Vectors

- The idea of getting the gradient of any parameter of the previous loss functions is to use them in **gradient descent to train the input/output vectors**.
- Usually, the set of **all parameters** in a model is **defined in terms of one long vector** θ.
- E.g. with d-dimensional vectors and V vocabulary size:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ v'_{aardvark} \\ v'_a \\ \vdots \\ v'_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Gradient Descent

- To minimize $J(\theta)$ over the full batch, requires to compute gradients for all windows.
- Updates for each element of θ:

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- With step size α
- In matrix notation for all parameters:

$$\theta^{new} = \theta^{old} - \alpha \frac{\partial}{\partial \theta^{old}} J(\theta)$$
$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

Stochastic Gradient Descent

- Corpus may have **billions of tokens and windows**.
- Is a **bad idea for neural nets to wait that much** for a single update.
- The parameters should be **updated after each window t**.
- This is call **Stochastic Gradient Descent**
- Each **gradient can be very sparse** (only contains information of the vectors of the window)
    - **Keep a hash** for word vectors or **update certain columns** of full embedding matrices $W^{(1)}$ and $W^{(2)}$.

Final Vectors

- We end up with **two matrices of vectors** $W^{(1)}$ and $W^{(2)}$.
- Each matrix has the **input and output representation** of each vector.
- Both **capture co-occurrence information**.
- How do we get a final vector?

Word2Vec Vectors' Characteristics

Linear Relationships

- The embeddings are very good at encoding dimensions of similarity.
- Analogies testing dimensions of similarity can be solved quite well by doing vector subtraction.
- Syntactically:
  - $w^{apple} - w^{apples} \approx w^{car} - w^{cars} \approx w^{family} - w^{families}$
  - Similarly for verb and adjectives morphological forms
- Semantically:
  - $w^{shirt} - w^{clothing} \approx w^{char} - w^{furniture}$
  - $w^{king} - w^{man} \approx w^{queen} - w^{woman}$

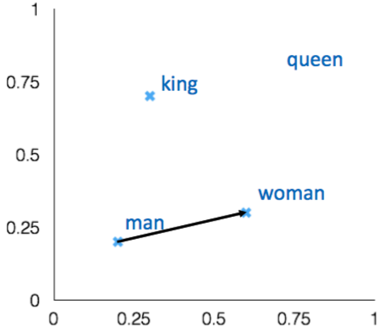
Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:? → $$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Count Based vs. Direct Prediction

LSA, HAL (Lund & Burgess),
COALS (Rohde et al),
Hellinger-PCA (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to small counts

- NNLM, HLBL, RNN, Skip-gram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity