

CS224N : Lecture 1 - Intro & Word Vectors



Word meaning can be representing rather well by a large vector of real numbers.

1. Human language and word meaning

- How do we represent the meaning of a word?
 - **WordNet** : A thesaurus containing lists of **synonym sets and hypernyms**
 - ▼ Problems with resources like WordNet
 - (1) Missing nuance
 - (2) Missing new meanings of words
 - (3) Subjective
 - (4) Requires human labor to create and adapt
 - (5) Can't compute accurate word similarity
 - ▼ Problems of the traditional NLP
 - Regard words as discrete symbols → Represent words by **one-hot vectors**
 - Very huge vector dimension (= number of words in vocabulary)
 - If vectors of the word are orthogonal → There is no natural notion of similarity for one-hot vectors
 - Solution : Learn to encode similarity in the vectors themselves

- **Distributional semantics** : A word's meaning is given by the words that frequently appear close-by (**contexts**)

2. Word2vec algorithm introduction

- **Word vectors**
 - **Distributed** Representation
 - = Word embeddings
 - = (Neural) Word representations
 - **Similar contexts** appears as **similar vector of words**
- **Word2vec** : framework for learning word vectors
 - Use the **similarity of the word vectors** for certain words to **calculate the probability** of them
 - Keep adjusting the word vectors to **maximize the probability**

3. Word2vec objective function gradients

- **Likelihood** : **predict context words** within a window of fixed size m
- **Objective function** : (average) **negative log likelihood**

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

▼ *Slides*

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m , given center word w_j . Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The **objective function** $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} | w_t; \theta)$?
- **Answer:** We will use two vectors per word w :
 - v_w when w is a center word
 - u_w when w is a context word
- Then for a center word c and a context word o :

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

② Exponentiation makes anything positive

① Dot product compares similarity of o and c .
 $u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$
 Larger dot product = larger probability

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$ Open region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
 - "max" because amplifies probability of largest x_i
 - "soft" because still assigns some probability to smaller x_i
 - Frequently used in Deep Learning

But sort of a weird name because it returns a distribution!

4. Optimization basics

- Optimization**
 - Adjust parameters to **minimize a loss**
 - Optimize parameters by walking down the gradient (compute all vector gradient)

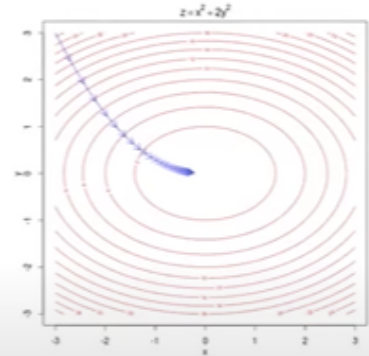
▼ Slides

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d -dimensional vectors and V -many words, we have:
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

5. Looking at word vectors

