Date Vector Semantics Subject

- Vector semantic is the standard way to represent word meaning in nlp.

- Suppose the meaning of the borrowed ongchio is unknown but we see it in a context:
  - "ongchoi is a delicious sauteed with garlic"
  - "ongchoi leaves with salty sauce"
  - "ongchoi is superb over rice."

Because ongchoi appears with words like rice, garlic, delicious, salty, we can infer it is also a leafy green. ⇩
computationally, this is done by counting word contexts.

- vector semantics = representing the meaning of words as mathematical vectors each representing the point in a multidimensional space.

- Embeddings: the actual vector representation of a word.

Date vectors and documents Subject

- Tf : [redacted]
  - a term-document matrix has: (vocabulary)
    - Rows = words
    - columns = documents
    - cells = frequency count
  - The idea is: "Similar documents have similar word distribution."
  - Since the vocabulary is huge, these vectors are sparse.

- Term-Term Matrix : ((Size $|V| \times |V|$))
  - Row = ((target word)) column = ((context word))
  - cells = ((the number of times the target word would happen in some context))

- "Cosine Similarity" for measuring Similarity:
  - a word or a document is represented by a vector.

$$w = (w_1, w_2, \dots, w_n),\ v = (v_1, v_2, \dots, v_n) \Rightarrow w \cdot v = \sum_{i=1}^{n} w_i \cdot v_i$$

??What is the problem with Raw Dot Product??

(I) it depends on vector length

(II) Frequent words ((like 'good', 'the', ...)) will have unfairly high values.

→ this means if we consider a $w_1 \cdot v_1 \geq M$, there will be frequent words w_2, v_2 that $w_2 \cdot v_2 \gg M$. this questions the validity of M as an optimized criteria

→ the ide is to divide by magnitude θ = angle between w and v $\cos(\theta) = \frac{v \cdot w}{|v||w|}$

Date ~~Cosine Similarity~~ Subject TF-IDF

- What's happening before ??
  - we build a coocurrance matrice ((words x words
  - each cell is just raw frequency.
- what is the problem ??
  - Raw counts are skewed. Some words appear everywhere
  - common words are not useful for distinguishing meaning and context
  - as a result, similarity matrices get dominated by frequently used uninformative words. which reduces accuracy when comparing meanings.

TF-IDF: make vectors more informative by reducing the weight of common words and increasing weight of rare but discriminative words.

- TF-IDF stands for:

$$\text{Term Frequency} = TF(w,d) = \frac{\text{count of } w \text{ in } d}{\text{total words in } d}$$

$$\text{Inverse Frequency document} = IDF(w) = \lg\left(\frac{N}{df(w)}\right)$$

N = total number of documents, $df(w)$ = number of documents that w appeared in

- $TF\_IDF(w,d) = TF(w,d) \times IDF(w)$
  - if a word appears a lot in a document, also appears a lot in all documents, it is meaningless stop words.

JABAN

Date

Subject PMI / PPMI

- We want a way to measure how strongly two words are associated???

- That's what PMI does??
  "do these words occure together more than we would expect by chance???"

- PMI = pointwise mutual information??

$$PMI(w,c) = \lg\left(\frac{P(w,c)}{P(w)P(c)}\right)$$

P(w,c) = probability that target word w and context-word c occure
P(w) = word occurs P(c) = context occurs

  - if w and c occure more than chance ⟹ PMI is positive
  - if w and c occure less than chance ⟹ PMI is negative
  - if w and c are independent ⟹ PMI = 0

- PPMI = positive PMI
  PMI can be negative, but negative association are often not useful for building semantic vectors.

$$PPMI(w,c) = \max(PMI(w,c), 0)$$

$$P(w,c) = \frac{count(w,c)}{\text{total cooccurances}} \qquad P(w) = \frac{count(w)}{\text{total cooccurances}}$$

- context distribution smoothing
  - Instead of using P(c), raise it to the power α (often α = 0.75)
  - this makes rare context words less document

$$PPMI_\alpha(w,c) = \max\left(\lg\left(\frac{P(w,c)}{P(w)P(c)^\alpha}\right), 0\right)$$

Date word-2vec Subject

- Sparse vectors problems
  - very high dimensions - harder to train classifiers
  - classifiers will need to many parameters.
  - Synonyms are far apart. eg: car have unrelated dimension automobile.

- Dense word embeddings.
  - Dense vectors: every number can be non-zero and real (eg: -03)
  - Low-dimensional: only 50-1000 numbers per word.
  - no explicit meaning per dimension - but they capture semantic similarity.
  - words in similar contexts get similar embeddings, even if they never appear together directly.

- word2vec: two main training algorithms.
  - skip-gram with negative samplings (SGNS)
  - continuous Bag of words (CBOW)
  - pretrained models are available and widely used
  - produces static embeddings; one fixed vector per word.

Date SGNS works Subject

• How skip-gram NS works??

① Suppose we have a corpus " The cat sat on the mat" with a window of size 2, the target context pairs are:

target	context	
cat	the, sat	⇒ the context is n words before and after target, here n=2
sat	cat, on	
on	the, sat	
the	on, mat	
mat	the	

these pairs are called "positive pairs". Because they happened next to each other:

(cat, the), (cat, sat), (sat, cat), (sat, on), ___

② Prediction task: given a target word, predict which words are nearby. this is a binary classification task.

- input: ((target word vector, also candidate context word embedding
- predict one for real context words. 0 for real words we don't care about this model's prediction only that while learning to predict, the model adjusts the word vectors.

but this will be very expensive. the vocabulary might have 100000 words.

③ Negative sampling:

- For real (target, context) pair ((positive ones)), we also make K random (target, random-word-context)) pair.
- this model learns to push positive vectors closer, and push negative pairs apart.

JAHAN

Date

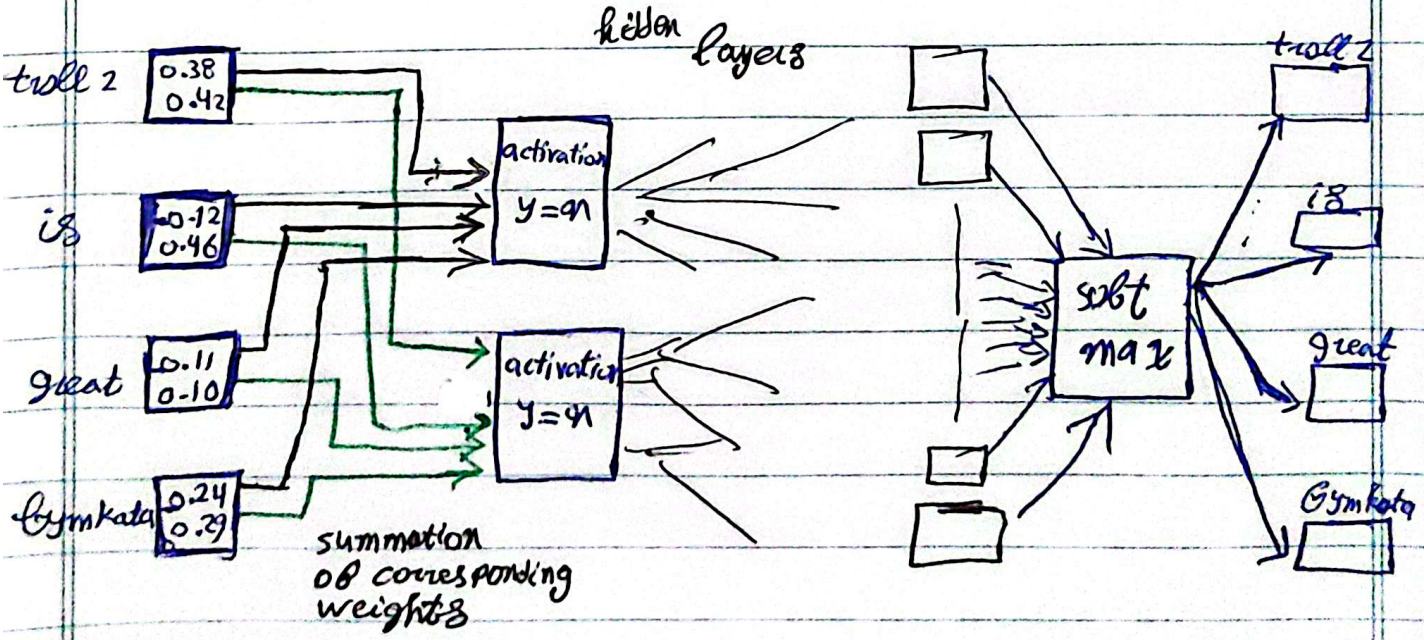
how (word2vec works (by me))

Subject

- how we can get a super simple Neural net that figures out what numbers should go with different words. Lets imagine we have 2 phrases.
  - Troll 2 is great
  - Gymkata is great

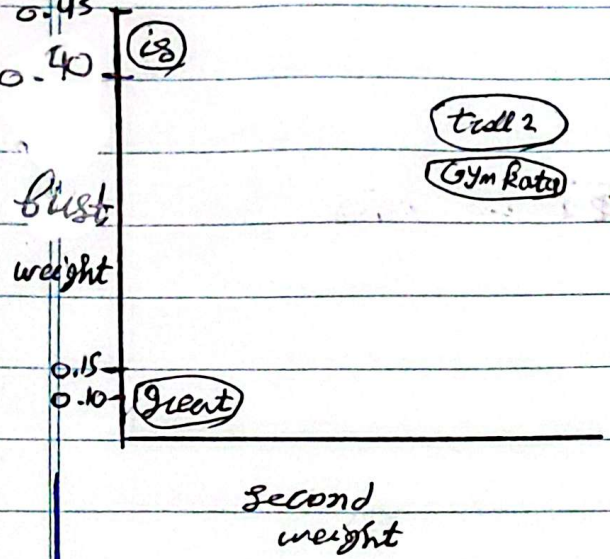
troll 2 and Gymkata are name of movies.

both phrases are about two movies. in order to create a neural network to figure out what numbers should associate with each word, the first thing we do is create inputs for each unique word. In this case we have 4 unique words, so we have 4 inputs. we assign random weights to each word. ((for simplicity, each word is in 2-Dimensional space)).

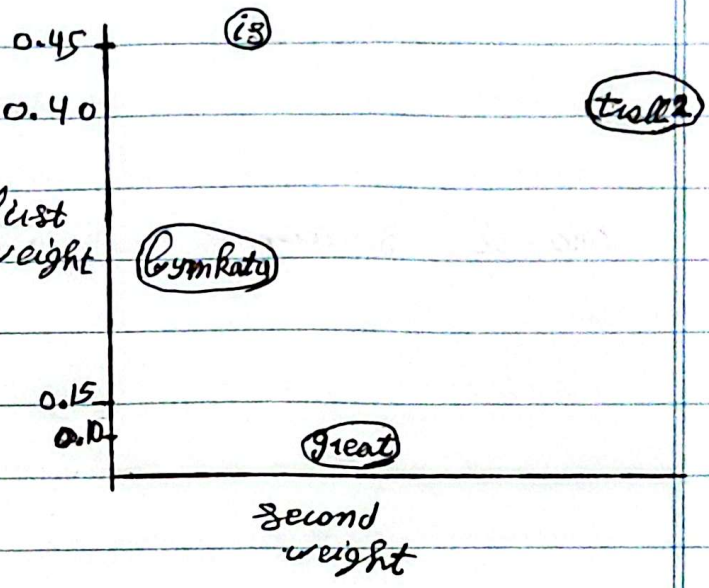


- using back-propagation, the weights of each word will be adjusted, such that similar words in similar contexts represent similar embeddings

After training



Before training



→ now, our model, knows both "troll 2" and "GymKata" are names of movies and that they should lie within same context.

Note: that model is designed to predict next word. for example feeding "is" into our NN, it will predict next word is "great". However, the main goal with this model, is to adjust weights, such that similar words close together.

after the creation of this model, we will create our batch data for training. we may use "SGNS" or "CBOW"

Skip-gram
- generate positive pairs
- geate k random negative pairs
- train with NN

CBOW
- instead of neighbour given center, Predicts a center given it's neighbours.

JAHAN

- The core idea is:
"we want the embedding vector of word w be more similar to the vectors of words that actually appear near it and less similar to random unrelated words."

- در صفحات قبل، توضیح دادم SGNS چجوری کار می‌کنه اما این پرسش وجود داره و وقتی داریم مدل nn رو برای SGNS «که توضیح دادم» train می‌کنه چجوری با فهمیدن تفاوت و نامربوط بودن، پارامترهای مدل nn آپدیت می‌شوند.»

(I) First we see that with a sliding window of size of 2 we generate context words, how?? we will create positive words pairs that are in the form (target, context1), (target, context2) with a window of size 2 that is [context1, target, context2]

(II) now we generate k random negative pairs that is a target word with all words that are not in context.
((target, and a word out of window))

important note:
we dont pick those samples uniformly. we pick them according to their unigram frequency $p(w)$ raise to power α
this makes common words, less likely to be dominant. makes rare words a bit more likely as negative.

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

Date Subject

(III) Prediction task:

- input : a pair (w, c) (target, context)
- Label : (1) if it's real positive pair, 0 if it's negative.

(IV) update embedding:

inside the model, each input pair has two learnable vectors:

(1) a target word vector embeddings

(2) a context word embedding.

we comput dot product of these vectors, pass it through a sigmoid function to get probability:

$$P(\text{context is real} \mid w, c) = \sigma(\bar{v}_w \cdot v_c)$$

then we use loss function(binary cross entropy) to adjust:

$$L = -\lg\left(\sigma(v_w \cdot v_c)\right) - \sum_{i=1}^{K} \lg\left(\sigma(-v_w \cdot v_{c_i})\right)$$

this says:

- Increase the dot product with real context words.
- decrease the dot product with negative words

Make vectors more or less similar.

- dot product as similarity signal:

if $\vec{v}_w$ and $\vec{v}_c$ point same direction, their dot product is large positive.

- if they are near orthoginal(unrelated), dot product is near zero
- if they point oppositely, dot product is negative.