Find the matrices U, Σ, V for $A = \begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix}$

Step 1 — find an orthogonal diagonalization

$$A^TA = \begin{bmatrix} 3 & 4 \\ 0 & 5 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix} = X$$

find eigen value & eigen vectors —

$$X - \lambda I = 0$$

$$\begin{bmatrix} 25-\lambda & 20 \\ 20 & 25-\lambda \end{bmatrix} = 0 \Rightarrow (25-\lambda)^2 - 400 = 0$$

$$\Rightarrow 625 - 50\lambda + \lambda^2 - 400 = 0$$

$$\Rightarrow \lambda^2 - 50\lambda + 225 = 0$$

$$\Rightarrow \lambda - 45\lambda - 5\lambda + 225 = 0$$

$$\Rightarrow \lambda(\lambda - 45) - 5(\lambda - 45) = 0$$

$$\Rightarrow \lambda = 45 / 5$$

So, $\lambda_1 = 45 \mid \lambda_2 = 5$

eigen vector

$$\begin{bmatrix} -20 & 20 \\ 20 & -20 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow v_1$$

$$\Rightarrow v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 20 & 20 \\ 20 & 20 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

v_1 & v_2 are these (orthogonal) eigen vectors rescaled to length 1.

$v_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$ $v_2 = \frac{1}{\sqrt{2}}\begin{bmatrix}-1\\1\end{bmatrix}$

So, $V = \begin{bmatrix}1/\sqrt{2} & -1/\sqrt{2}\\ 1/\sqrt{2} & 1/\sqrt{2}\end{bmatrix}$

$D^2 \begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix} = \begin{bmatrix}\sqrt{\lambda_1} & 0\\ 0 & \sqrt{\lambda_2}\end{bmatrix} = \begin{bmatrix}\sqrt{45} & 0\\ 0 & \sqrt{5}\end{bmatrix}$

$\Sigma = [D] = \begin{bmatrix}\sqrt{45} & 0\\ 0 & \sqrt{5}\end{bmatrix}$

$u_1 = \frac{1}{\sigma_1} A v_1 = \frac{1}{\sqrt{2}\sqrt{45}} \begin{bmatrix}3 & 0\\ 4 & 5\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \frac{3}{3\sqrt{5}\sqrt{2}}\begin{bmatrix}1\\3\end{bmatrix}$

$= 1/\sqrt{10}\begin{bmatrix}1\\3\end{bmatrix}$

$u_2 = \frac{1}{\sigma_2} A v_2 = \frac{1}{\sqrt{2}}\frac{1}{\sqrt{5}}\begin{bmatrix}3 & 0\\ 4 & 5\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} = \frac{1}{\sqrt{10}}\begin{bmatrix}-3\\1\end{bmatrix}$

$U = [u_1\ u_2] = \frac{1}{\sqrt{10}}\begin{bmatrix}1 & -3\\ 3 & 1\end{bmatrix}$

$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$

Limitations:

Computational complexity: The computational complexity of SVD can be high, especially for large matrices, making it less suitable for real-time or online scenarios.

Interpretability: The resulting singular vectors might not have a direct interpretation, as they represent abstract concepts or latent factors.

Sensitive to outliers: Outliers in the data can impact the quality of the decomposition, potentially leading to distorted results.

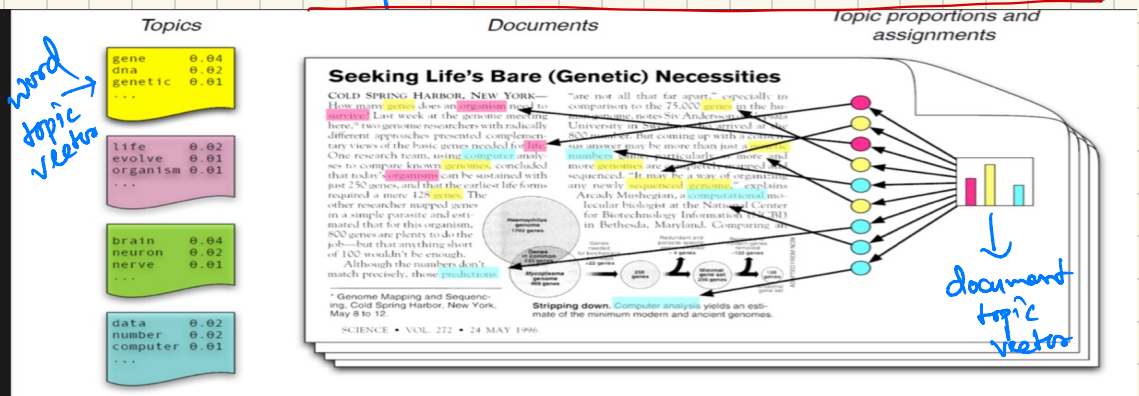
Latent Diricilate Allocation

Latent Dirichlet allocation is a technique to map sentences to topics. LDA extracts certain sets of topic according to topic we fed to it. Before generating those topic there are numerous process that are carried out by LDA. Before applying that process we have certain amount of rules, facts that we considered.
Assumptions of LDA for Topic Modelling:

- Documents with similar topics use similar groups of words
- Latent topics can then be found by searching for groups of words that frequently occur together in documents across the corpus
- Documents are probability distributions over latent topics which signifies certain document will contain more words of a specific topic.

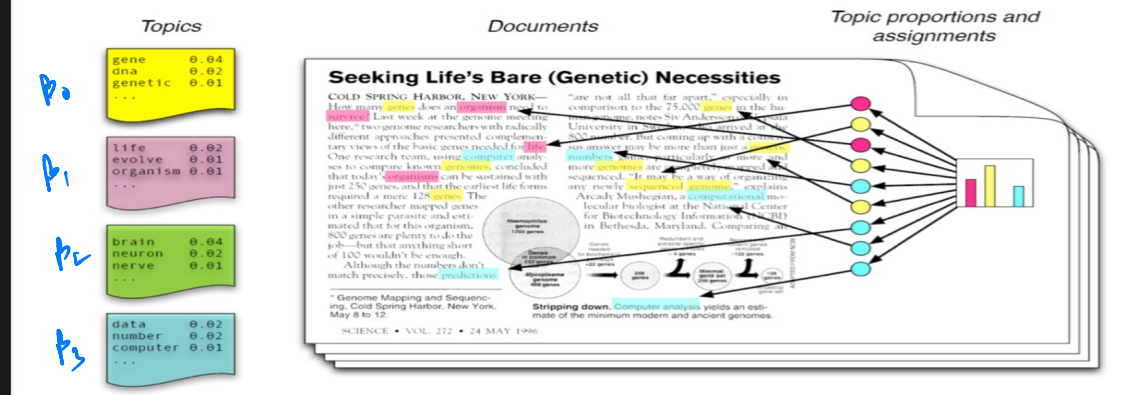
→ map words to topic.

→ each topic can be considered as a cluster.



- Each topic is a distribution over words
- Each document is a mixture of topics
- Each word is drawn from the mixture

Latent Dirichlet Allocation (LDA)



- Only documents are observed
- Topics, mixtures, etc. are all hidden and need to be learned/predicted from data

θ → topic distribution

$\theta = [.8, .05, .15]$

↳ topic 2.

Step 1: get topic vector for document θ_d

Step 2: if our document has n words.

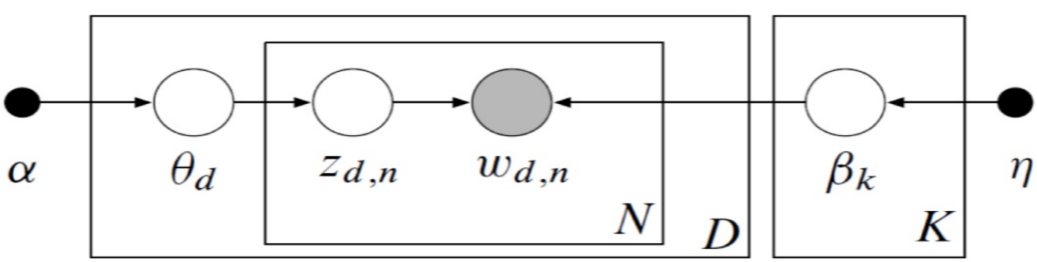
2.1 select topic $z_i \sim \theta_d$ (multinomial dist.)

2.2 generate $w_i \sim \beta_{z_i}$ (multinomial dist.)

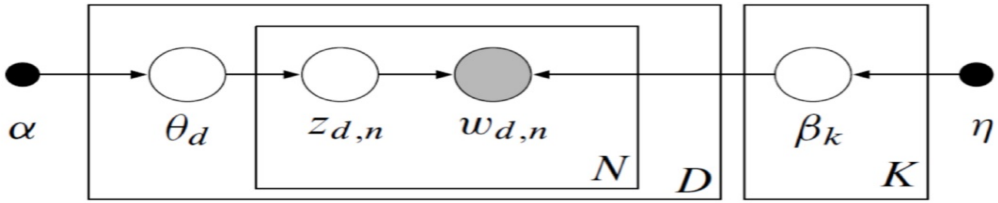
repeat for n words

Here, we don't know θ_d, z_i & β_{z_i}

we only know w_i



- α and η are parameters of the prior distributions over θ and β respectively
- θ_d is the distribution of topics for document d (real vector of length K)
- β_k is the distribution of words for topic k (real vector of length V)
- $z_{d,n}$ is the topic for the n^{th} word in the d^{th} document
- $w_{d,n}$ is the n^{th} word of the d^{th} document



- **Plate notation**
  - There are $N \cdot D$ different variables that represent the observed words in the different documents
  - There are K total topics (assumed to be known in advance)
  - There are D total documents

The only observed variables are the words in the document.

- the topic for each word, the distribution over topics for each document, the distribution of words per topic are all latent variables in the model.

⇒ The model contains both continuous & discrete random variables.

→ θ_d & β_K are vectors of probabilities.

→ $Z_{d,n}$ is an integer in $\{1, \ldots K\}$ that indicates the topic of the n^{th} word in the d^{th} document.

→ $W_{d,n}$ is an integer in $\{1, \ldots V\}$ which indexes over all possible words.

⇒ $\theta_d \sim Dir(\alpha)$ where $Dir(\alpha)$ is the Dirichlet distribution with parameter vector $\alpha > 0$

⇒ $\beta_k \sim Dir(\eta)$ with parameter vector $\eta > 0$

⇒ Dirichlet distribution over $x_1, \dots x_K$ such that $x_1, \dots x_K \geq 0$ & $\sum_i x_i = 1$

$$f(x_1, x_2 \dots x_K; \alpha_1, \alpha_2 \dots \alpha_K) \propto \prod_i x_i^{\alpha_i - 1}$$

⇒ The Dirichlet distribution is a distribution over probability distributions over K elements.

→ α controls sparsity: lower α's make sparse distribution more likely.

- The joint distribution is then

$$p(w, z, \theta, \beta | \alpha, \eta) = \prod_k p(\beta_k | \eta) \prod_d \left[p(\theta_d | \alpha) \prod_n p(z_{d,n} | \theta_d) \, p(w_{d,n} | z_{d,n}, \beta) \right]$$

- Inference in this model is NP-hard
- Given the D documents, want to find the parameters that best maximize the joint probability
  - Can use an EM based approach called variational EM

Co-occurrence Matrix:

Namely, consider the expression 'social media': both the words can have independent meaning, however, when they are together, they express a precise, unique concept.
Nevertheless, it is not an easy task, since if both words are frequent by themselves, their co-occurrence might be just a chance. Namely, consider the name 'Las Vegas': it is not that frequent to read only 'Las' or 'Vegas' (in English corpora of course). The only way we see them is in the bigram Las Vegas, hence it is likely for them to form a unique concept. On the other hand, if we think of 'New York', it is easy to see that the word 'New' will probably occur very frequently in different contexts. How can we assess that the co-occurrence with York is meaningful and not as vague as 'new dog, new cat…'?
The answer lies in the Pointwise Mutual Information (PMI) criterion. The idea of PMI is that we want to quantify the likelihood of co-occurrence of two words, taking into account the fact that it might be caused by the frequency of the single words. Hence, the algorithm computes the (log) probability of co-occurrence scaled by the product of the single probability of occurrence as follows:

Pointwise Mutual Information

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

$$PMI(X = x, Y = y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)

Do words x and y co-occur more than if they were independent?

$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Positive Pointwise Mutual Information

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
  - Things are co-occurring **less than** we expect by chance
  - Unreliable without enormous corpora
    - Imagine w1 and w2 whose probability is each 10^{-6}
    - Hard to be sure p(w1,w2) is significantly different than 10^{-12}
  - Plus it's not clear people are good at "unrelatedness"
- So we just replace negative PMI values by 0

- Positive PMI (PPMI) between word1 and word2:

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

Count(w,context)

	computer	data	pinch	result	sugar
apricot	0	0	1	0	1
pineapple	0	0	1	0	1
digital	2	1	0	1	0
information	1	6	0	4	0

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N}$$

p(w=information,c=data) = 6/19 = .32

p(w=information) = 11/19 = .58

p(c=data) = 7/19 = .37

p(w,context) **p(w)**

	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

$$\text{PMI}_{ij} = \log \frac{p_{ij}}{p_{i*}p_{*j}}$$

	p(w,context)					p(w)
	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

- pmi(information,data) = $\log_2$ (.32 / (.37*.58)) = .58

(.57 using full precision)

PPMI(w,context)

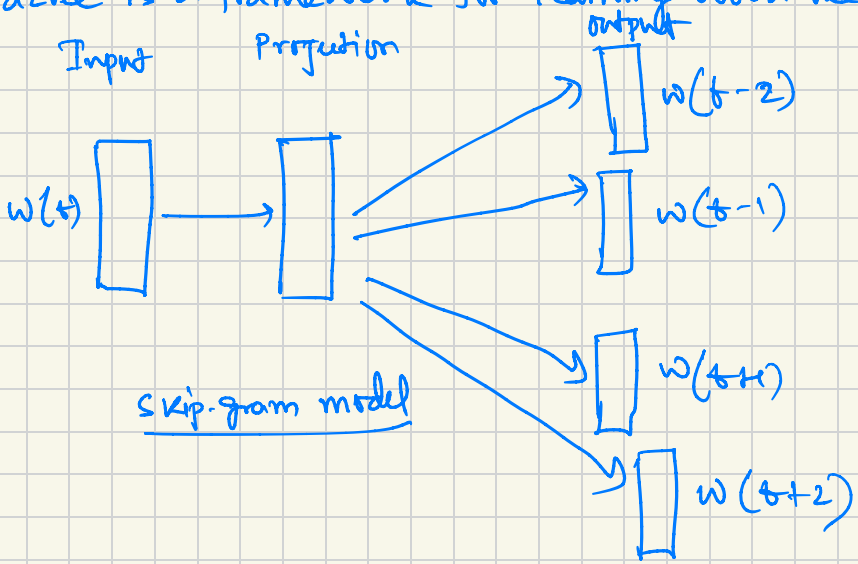
	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

** PMI is biased towards infrequent event.
→ very rear word will have very high PMI value.

** TF-IDF is the alternate to PMI.

Word2Vec

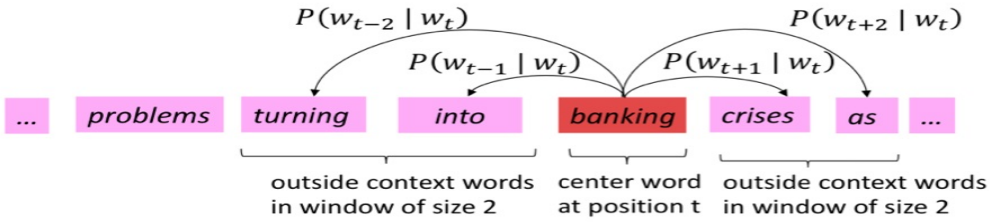
word2vec is a framework for learning word vectors.



Idea:

1. we have a large corpus of text: a long list of words.
2. Every word in a fixed vocabulary is represented by a vector.
3. Go through each position t in the text, which has a center word c & context ("outside") words o.
4. use the similarity of the word vectors for c & o to calculate the probability of o given c (or vice versa)
5. Keep adjusting the word vectors to maximize this probability.

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: objective function

No of tokens

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

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

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:

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

Minimizing objective function ⟺ Maximizing predictive accuracy

Negative log likelihood

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

Avg of Negative log likelihood.

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

Now, How to calculate $P(w_{t+j} \mid w_t; \theta)$?

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **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*:

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

Word2vec: prediction function

② Exponentiation makes anything positive

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
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** $\mathbb{R}^n \to (0,1)^n$ ← 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!

We know $J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log P(w_{t+j} \mid w_t)$

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

$$\Rightarrow \frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)$$

$$\Rightarrow u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \sum_{x=1}^{V} \frac{\partial}{\partial v_c} \exp(u_x^T v_c)$$

$$\Rightarrow u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x$$

$$\Rightarrow u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x$$

$$\Rightarrow u_o - \sum_{x=1}^{V} \boxed{\frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}}\, u_x$$

→ similar to $P(x|c)$

$$\Rightarrow u_o - \sum_{x=1}^{V} P(x|c)\, u_x$$

$$\frac{\partial J(\theta)}{\partial v_c} = -u_o + \sum_{x=1}^{V} P(x|c)\, u_x$$

u_o: observed

→ Avg over all context vectors weighted by their probability.

So basically, we are subtracting the actual and expected representations to get the direction in which I should move and change my weight vector Vc so that I can maximize the likelihood.

Moving forward in the same manner, we can also calculate the derivative of J(θ) wrt to Uw. There will be two cases for Uw, one when w is the context word and one when w is not the context word. This will be the output of derivatives in both cases:

W ≠ O:

$$\frac{\partial J(\theta)}{\partial u_w} = \sum_{x=1}^{V} P(x|c) * v_c$$

$$J(\theta) = u_o^T v_c - \log \sum_{w=1}^{V} \exp(u_w^T v_c)$$

$$\frac{\partial J(\theta)}{\partial u_w} = 0 - \sum_{x=1}^{V} P(x|c) v_c$$

W = O:

$$\frac{\partial J(\theta)}{\partial u_w} = -v_c + \sum_{x=1}^{V} P(x|c) * v_c$$

$$\frac{\partial J(\theta)}{\partial u_w} = v_c - \sum_{x=1}^{V} P(x|c) v_c$$

Once we have both the derivatives, we can use them in our SGD equation to update the weights.

However, there is one problem in this approach. As we can see, in the denominator, we have to take the exponential of the dot product of all our words and this is very time consuming when we have a huge vocabulary. We will need to train millions of weights which is not feasible.

So, in order to decrease the training time, a new method was used called negative sampling. In this method, we will be updating only a small percentage of weights in one step. We will select a few -ve words i.e. the words which are not in the context window and we will change our weights in such a way that it maximizes the probability of real context words and minimize the probability of random words appearing around the center word. This change the loss function and now we are trying to maximize the following equation:

Intuitively, what is the partition function doing, such that we might understand how to remove it? Let's repeat the softmax here:

Affinity of word c for context o

$$p_{U,V}(o|c) = \frac{\exp u_o^\top v_c}{\sum_{w \in \mathcal{V}} \exp u_w^\top v_c} \tag{14}$$

Partition function, or, normalization

From a probabilistic perspective, the partition function guarantees a probability by normalizing the scores to sum to 1. (The exponential guarantees that the scores are non-negative.) From a learning perspective, the partition function "pushes down" on all the words other than the observed words. Put another way, the numerator of this equation encourages the model to make u_o more like v_c; the denominator encourages all the other u_w for $w \neq o$ less like v_c. The intuition of negative sampling is that **we don't need to push down on all the u_w all the time**, since that's where most of the cost comes from.

However, the actual skip-gram with negative sampling (SGNS) objective ends up being a bit more different; we'll write it here:

Affinity of word c for context o

$$\log \sigma(u_o^\top v_c) + \sum_{\ell=1}^{k} \left[\log \sigma(-u_\ell^\top v_c)\right] \tag{15}$$

Push down everything else in expectation (replacement for partition fn

where, where σ is the logistic function, and $u_\ell \sim p_{\text{neg}}$, which means u_ℓ is drawn from a distribution we haven't defined, called p_{neg}, Think of this for now like the uniform distribution over $\mathcal{V}$. What is this objective doing? It has two terms, just like how we've described the orginal skipgram. The first term encourages v_c and u_o to be more like each other, and the second term encourages v_c and u_ℓ for k random samples from the vocabulary to be less like each other. The intuition here is that if we randomly push down a few words at each step, then on average, things will work out sort of as if we always pushed down every word.

$$J_{neg.sample}(\theta) = -\log\sigma(u_o^T v_c) - \sum_{k=1}^{K}\left[\log\sigma(-u_k^T v_c)\right]$$

K → no of negative sample.

To maximize the above term, we again need to take the derivative of the Loss function with respect to the weights, this case, it will be ***Uw, Uk, and Vc.*** Doing this in a similar way as we did above will give us the following three equations:

$$\frac{\partial J(\theta)}{\partial v_c} = -\sigma(-u_o^T v_c)u_o + \sum_{k=1}^{K} \sigma(u_k^T v_c)u_k$$

$$\frac{\partial J(\theta)}{\partial u_o} = -\sigma(-u_o^T v_c)v_c$$

$$\frac{\partial J(\theta)}{\partial u_k} = \sum_{k=1}^{K} \sigma(u_k^T v_c)v_c$$

With all the derivatives calculated, now we can update our weight vectors little by little and get a vector representation that will point words appearing in a similar context in the same direction.

Problems with word2vec:

This captures Coccurrence of words one at a time.

Why not to capture Coccurrence counts directly?

(d) In word2vec (skipgram) we compute the predicted probability distribution vector ($\hat{\mathbf{y}}$) on the vocabulary via a softmax over the output vector $\mathbf{z}$ i.e. $\hat{\mathbf{y}} = softmax(\mathbf{z})$. Given that the desired output is $\mathbf{y}$ and the error function $\mathcal{E}$ is the cross entropy function $\mathcal{E} = \sum_i -y_i ln(\hat{y}_i)$ derive the gradient for the first step in the backpropagation.

$z = w^T h + b$ (wxd) — $h \Rightarrow$ hidden layer, w_2 weights.

$\hat{y} = softmax(z)$ $\quad \hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{|V|} e^{z_j}}$ $\quad i = 1 \dots |V|$

$$\epsilon = -\sum_{i=1}^{|V|} y_i \ln(\hat{y}_i)$$

$$\mathcal{E} = -\sum_{i=1}^{|V|} y_i \left[z_i - \ln \sum_{j=1}^{|V|} e^{z_j} \right]$$

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}} \Rightarrow \frac{\partial \mathcal{E}}{\partial z_i} = -\left[y_i - \frac{e^{z_i}}{\sum_{j=1}^{|V|} e^{z_j}} \right]$$

$$= -\left[y_i - \hat{y}_i \right]$$

$$= \hat{y}_i - y_i$$

$\frac{\partial z_i}{\partial w_{ij}} = x_i$ $\quad$ so, $\boxed{\frac{\partial \mathcal{E}}{\partial w_{ij}} = (\hat{y}_i - y_i)^T x_i}$

3. Why not capture co-occurrence counts directly?

There's something weird about iterating through the whole corpus (perhaps many times); why don't we just accumulate all the statistics of what words appear near each other?!?

Building a co-occurrence matrix *X*

- 2 options: windows vs. full document
- Window: Similar to word2vec, use window around each word → captures some syntactic and semantic information ("word space")
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis" ("document space")

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)

- Example corpus:
  - I like deep learning
  - I like NLP
  - I enjoy flying

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

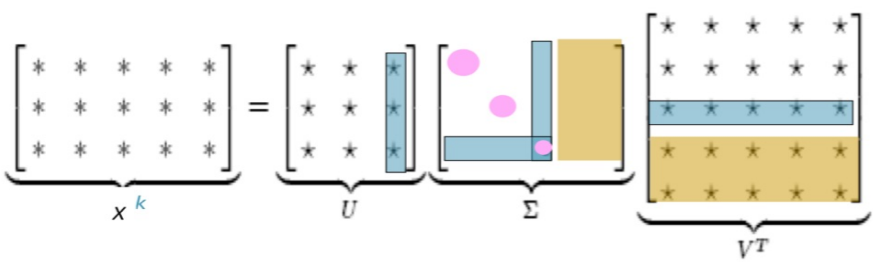
Co-occurrence vectors

- Simple count co-occurrence vectors
  - Vectors increase in size with vocabulary
  - Very high dimensional: require a lot of storage (though sparse)
  - Subsequent classification models have sparsity issues → Models are less robust
- Low-dimensional vectors
  - Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
  - Usually 25–1000 dimensions, similar to word2vec
  - How to reduce the dimensionality?

Classic Method: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix X

Factorizes X into $U\Sigma V^T$, where U and V are orthonormal (unit vectors and orthogonal)



Retain only k singular values, in order to generalize.
$\hat{X}$ is the best rank k approximation to X, in terms of least squares.
Classic linear algebra result. Expensive to compute for large matrices.

Hacks to X (several used in Rohde et al. 2005 in COALS)

- Running an SVD on raw counts doesn't work well!!!
- Scaling the counts in the cells can help ***a lot***
  - Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
    - log the frequencies
    - min(*X*, *t*), with $t \approx 100$
    - Ignore the function words
- Ramped windows that count closer words more than further away words
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Problems with SVD

Computational cost scales quadratically for $n \times m$ matrix:
$O(mn^2)$ flops (when $n < m$)

→ Bad for millions of words or documents.

→ Hard to incorporate new word or documents.

Combining the best from word2vec & SVD

word2vec → Window based

SVD → Matrix factorization

Glove — Global Vectors for Word Representation

Glove (Global vectors for word Embedding)

3. Dense word vectors learned through word2vec or GloVe have many advantages over using sparse one-hot word vectors. Which of the following is a NOT advantage dense vectors have over sparse vectors?

a) Models using dense word vectors generalize better to unseen words than those using sparse vectors.

b) Models using dense word vectors generalize better to rare words than those using sparse vectors.

c) Dense word vectors encode similarity between words while sparse vectors do not.

d) Dense word vectors are easier to include as features in machine learning systems than sparse vectors.

Answer: (a) Models using dense word vectors generalize better to unseen words than those using sparse vectors.

Just like sparse representations, word2vec or GloVe do not have representations for unseen words and hence do not help in generalization.

Table 1: Co-occurrence probabilities for target words *ice* and *steam* with selected context words from a 6 billion token corpus. Only in the ratio does noise from non-discriminative words like *water* and *fashion* cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Linear structure" in this context usually just refers to the addition and scalar multiplication of the vector space.

context of word i.

We begin with a simple example that showcases how certain aspects of meaning can be extracted directly from co-occurrence probabilities. Consider two words i and j that exhibit a particular aspect of interest; for concreteness, suppose we are interested in the concept of thermodynamic phase, for which we might take $i = ice$ and $j = steam$. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words, k. For words k related to ice but not steam, say $k = solid$, we expect the ratio P_{ik}/P_{jk} will be large. Similarly, for words k related to steam but not ice, say $k = gas$, the ratio should be small. For words k like *water* or *fashion*, that are either related to both ice and steam, or to neither, the ratio should be close to one. Table 1 shows these probabilities and their ratios for a large corpus, and the numbers confirm these expectations. Compared to the raw probabilities, the ratio is better able to distinguish relevant words (*solid* and *gas*) from irrelevant words (*water* and *fashion*) and it is also better able to discriminate between the two relevant words.

The above argument suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves. Noting that the ratio P_{ik}/P_{jk} depends on three words i, j, and k, the most general model takes the form,

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \tag{1}$$

where $w \in \mathbb{R}^d$ are word vectors and $\tilde{w} \in \mathbb{R}^d$ are separate context word vectors whose role will be discussed in Section 4.2. In this equation, the right-hand side is extracted from the corpus, and F may depend on some as-of-yet unspecified parameters. The number of possibilities for F is vast, but by enforcing a few desiderata we can select a unique choice. First, we would like F to encode the information present the ratio P_{ik}/P_{jk} in the word vector space. Since vector spaces are inherently linear structures, the most natural way to do this is with vector differences. With this aim, we can restrict our consideration to those functions F that depend only on the difference of the two target words, modifying Eqn. (1) to,

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}. \tag{2}$$

Next, we note that the arguments of F in Eqn. (2) are vectors while the right-hand side is a scalar. While F could be taken to be a complicated function parameterized by, e.g., a neural network, doing so would obfuscate the linear structure we are trying to capture. To avoid this issue, we can first take the dot product of the arguments,

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}}, \tag{3}$$

which prevents F from mixing the vector dimensions in undesirable ways. Next, note that for word-word co-occurrence matrices, the distinction between a word and a context word is arbitrary and that we are free to exchange the two roles. To do so consistently, we must not only exchange $w \leftrightarrow \tilde{w}$ but also $X \leftrightarrow X^T$. Our final model should be invariant under this relabeling, but Eqn. (3) is not. However, the symmetry can be restored in two steps. First, we require that F be a homomorphism between the groups $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$, i.e.,

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}, \tag{4}$$

which, by Eqn. (3), is solved by,

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}. \tag{5}$$

The solution to Eqn. (4) is $F = \exp$, or,

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i). \tag{6}$$

Next, if we assume F has a certain property (i.e. homomorphism between additive group and the multiplicative group) which gives,
F(w_i* u_k — w_j* u_k) = F(w_i* u_k)/F(w_j* u_k) = P_ik/P_jk
In other words this particular homomorphism ensures that the subtraction F(A-B) can also be represented as a division F(A)/F(B) and get the same result.

Next, we note that Eqn. (6) would exhibit the exchange symmetry if not for the $\log(X_i)$ on the right-hand side. However, this term is independent of k so it can be absorbed into a bias b_i for w_i. Finally, adding an additional bias $\tilde{b}_k$ for $\tilde{w}_k$ restores the symmetry,

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik}) \,. \tag{7}$$

Eqn. (7) is a drastic simplification over Eqn. (1), but it is actually ill-defined since the logarithm diverges whenever its argument is zero. One resolution to this issue is to include an additive shift in the logarithm, $\log(X_{ik}) \rightarrow \log(1 + X_{ik})$, which maintains the sparsity of X while avoiding the divergences. The idea of factorizing the log of the co-occurrence matrix is closely related to LSA and we will use the resulting model as a baseline in our experiments. A main drawback to this model is that it weighs all co-occurrences equally, even those that happen rarely or never. Such rare co-occurrences are noisy and carry less information than the more frequent ones — yet even just the zero entries account for 75–95% of the data in X, depending on the vocabulary size and corpus.

We propose a new weighted least squares regression model that addresses these problems. Casting Eqn. (7) as a least squares problem and introducing a weighting function $f(X_{ij})$ into the cost function gives us the model

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \,, \tag{8}$$

where V is the size of the vocabulary. The weighting function should obey the following properties:

1. $f(0) = 0$. If f is viewed as a continuous function, it should vanish as $x \rightarrow 0$ fast enough that the $\lim_{x \rightarrow 0} f(x) \log^2 x$ is finite.
2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.
3. $f(x)$ should be relatively small for large values of x, so that frequent co-occurrences are not overweighted.

Of course a large number of functions satisfy these properties, but one class of functions that we found to work well can be parameterized as,

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \,. \tag{9}$$

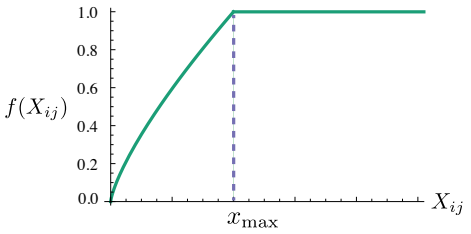


Figure 1: Weighting function f with $\alpha = 3/4$.

The performance of the model depends weakly on the cutoff, which we fix to $x_{\max} = 100$ for all our experiments. We found that $\alpha = 3/4$ gives a modest improvement over a linear version with $\alpha = 1$. Although we offer only empirical motivation for choosing the value 3/4, it is interesting that a similar fractional power scaling was found to give the best performance in (Mikolov et al., 2013a).

3.1 Relationship to Other Models

Because all unsupervised methods for learning word vectors are ultimately based on the occurrence statistics of a corpus, there should be commonalities between the models. Nevertheless, certain models remain somewhat opaque in this regard, particularly the recent window-based methods like skip-gram and ivLBL. Therefore, in this subsection we show how these models are related to our proposed model, as defined in Eqn. (8).

The starting point for the skip-gram or ivLBL methods is a model Q_{ij} for the probability that word j appears in the context of word i. For concreteness, let us assume that Q_{ij} is a softmax,

$$Q_{ij} = \frac{\exp(w_i^T \tilde{w}_j)}{\sum_{k=1}^{V} \exp(w_i^T \tilde{w}_k)} \,. \tag{10}$$

Most of the details of these models are irrelevant for our purposes, aside from the the fact that they attempt to maximize the log probability as a context window scans over the corpus. Training proceeds in an on-line, stochastic fashion, but the implied global objective function can be written as,

$$J = -\sum_{\substack{i \in \text{corpus} \\ j \in \text{context}(i)}} \log Q_{ij} \,. \tag{11}$$

Evaluating the normalization factor of the softmax for each term in this sum is costly. To allow for efficient training, the skip-gram and ivLBL models introduce approximations to Q_{ij}. However, the sum in Eqn. (11) can be evaluated much

more efficiently if we first group together those terms that have the same values for i and j,

$$J = -\sum_{i=1}^{V}\sum_{j=1}^{V} X_{ij} \log Q_{ij}\,, \tag{12}$$

where we have used the fact that the number of like terms is given by the co-occurrence matrix X.

Recalling our notation for $X_i = \sum_k X_{ik}$ and $P_{ij} = X_{ij}/X_i$, we can rewrite J as,

$$J = -\sum_{i=1}^{V} X_i \sum_{j=1}^{V} P_{ij} \log Q_{ij} = \sum_{i=1}^{V} X_i H(P_i, Q_i)\,, \tag{13}$$

where $H(P_i, Q_i)$ is the cross entropy of the distributions P_i and Q_i, which we define in analogy to X_i. As a weighted sum of cross-entropy error, this objective bears some formal resemblance to the weighted least squares objective of Eqn. (8). In fact, it is possible to optimize Eqn. (13) directly as opposed to the on-line training methods used in the skip-gram and ivLBL models. One could interpret this objective as a "global skip-gram" model, and it might be interesting to investigate further. On the other hand, Eqn. (13) exhibits a number of undesirable properties that ought to be addressed before adopting it as a model for learning word vectors.

To begin, cross entropy error is just one among many possible distance measures between probability distributions, and it has the unfortunate property that distributions with long tails are often modeled poorly with too much weight given to the unlikely events. Furthermore, for the measure to be bounded it requires that the model distribution Q be properly normalized. This presents a computational bottleneck owing to the sum over the whole vocabulary in Eqn. (10), and it would be desirable to consider a different distance measure that did not require this property of Q. A natural choice would be a least squares objective in which normalization factors in Q and P are discarded,

$$\hat{J} = \sum_{i,j} X_i (\hat{P}_{ij} - \hat{Q}_{ij})^2 \tag{14}$$

where $\hat{P}_{ij} = X_{ij}$ and $\hat{Q}_{ij} = \exp(w_i^T \tilde{w}_j)$ are the unnormalized distributions. At this stage another problem emerges, namely that X_{ij} often takes very large values, which can complicate the optimization. An effective remedy is to minimize the squared error of the logarithms of $\hat{P}$ and $\hat{Q}$ instead,

$$\begin{aligned}\hat{J} &= \sum_{i,j} X_i (\log \hat{P}_{ij} - \log \hat{Q}_{ij})^2 \\ &= \sum_{i,j} X_i (w_i^T \tilde{w}_j - \log X_{ij})^2\,. \end{aligned} \tag{15}$$

Finally, we observe that while the weighting factor X_i is preordained by the on-line training method inherent to the skip-gram and ivLBL models, it is by no means guaranteed to be optimal. In fact, Mikolov et al. (2013a) observe that performance can be increased by filtering the data so as to reduce the effective value of the weighting factor for frequent words. With this in mind, we introduce a more general weighting function, which we are free to take to depend on the context word as well. The result is,

$$\hat{J} = \sum_{i,j} f(X_{ij}) (w_i^T \tilde{w}_j - \log X_{ij})^2\,, \tag{16}$$

which is equivalent[1] to the cost function of Eqn. (8), which we derived previously.

3.2 Complexity of the model

As can be seen from Eqn. (8) and the explicit form of the weighting function $f(X)$, the computational complexity of the model depends on the number of nonzero elements in the matrix X. As this number is always less than the total number of entries of the matrix, the model scales no worse than $O(|V|^2)$. At first glance this might seem like a substantial improvement over the shallow window-based approaches, which scale with the corpus size, $|C|$. However, typical vocabularies have hundreds of thousands of words, so that $|V|^2$ can be in the hundreds of billions, which is actually much larger than most corpora. For this reason it is important to determine whether a tighter bound can be placed on the number of nonzero elements of X.

In order to make any concrete statements about the number of nonzero elements in X, it is necessary to make some assumptions about the distribution of word co-occurrences. In particular, we will assume that the number of co-occurrences of word i with word j, X_{ij}, can be modeled as a power-law function of the frequency rank of that word pair, r_{ij}:

$$X_{ij} = \frac{k}{(r_{ij})^\alpha}\,. \tag{17}$$

[1] We could also include bias terms in Eqn. (16).

Congratulations! You passed!
TO PASS 80% or higher

Keep Learning

GRADE
100%

Natural Language Processing & Word Embeddings

LATEST SUBMISSION GRADE
100%

1. Suppose you learn a word embedding for a vocabulary of 10000 words. Then the embedding vectors should be 10000 dimensional, so as to capture the full range of variation and meaning in those words. **1 / 1 point**

- ○ True
- ◉ False

Correct
The dimension of word vectors is usually smaller than the size of the vocabulary. Most common sizes for word vectors ranges between 50 and 400.

2. What is t-SNE? **1 / 1 point**

- ○ A linear transformation that allows us to solve analogies on word vectors
- ◉ A non-linear dimensionality reduction technique
- ○ A supervised learning algorithm for learning word embeddings
- ○ An open-source sequence modeling library

Correct
Yes

3. Suppose you download a pre-trained word embedding which has been trained on a huge corpus of text. You then use this word embedding to train an RNN for a language task of recognizing if someone is happy from a short snippet of text, using a small training set. **1 / 1 point**

x (input text)	y (happy?)
I'm feeling wonderful today!	1
I'm bummed my cat is ill.	0
Really enjoying this!	1

Then even if the word "ecstatic" does not appear in your small training set, your RNN might reasonably be expected to recognize "I'm ecstatic" as deserving a label $y = 1$.

- ◉ True
- ○ False

Correct
Yes, word vectors empower your model with an incredible ability to generalize. The vector for "ecstatic would contain a positive/happy connotation which will probably make your model classified the sentence as a "1".

4. Which of these equations do you think should hold for a good word embedding? (Check all that apply) **1 / 1 point**

- ☑ $e_{boy} - e_{girl} \approx e_{brother} - e_{sister}$

Correct
Yes!

- ☐ $e_{boy} - e_{girl} \approx e_{sister} - e_{brother}$
- ☑ $e_{boy} - e_{brother} \approx e_{girl} - e_{sister}$

Correct
Yes!

- ☐ $e_{boy} - e_{brother} \approx e_{sister} - e_{girl}$

5. Let E be an embedding matrix, and let o_{1234} be a one-hot vector corresponding to word 1234. Then to get the embedding of word 1234, why don't we call $E * o_{1234}$ in Python? **1 / 1 point**

- ◉ It is computationally wasteful.
- ○ The correct formula is $E^T * o_{1234}$.

- This doesn't handle unknown words (<UNK>).
- None of the above: calling the Python snippet as described above is fine.

Correct
Yes, the element-wise multiplication will be extremely inefficient.

6. When learning word embeddings, we create an artificial task of estimating $P(target \mid context)$. It is okay if we do poorly on this artificial prediction task; the more important by-product of this task is that we learn a useful set of word embeddings. **1 / 1 point**

- (•) True
- ( ) False

Correct

7. In the word2vec algorithm, you estimate $P(t \mid c)$, where t is the target word and c is a context word. How are t and c chosen from the training set? Pick the best answer. **1 / 1 point**

- ( ) c is the sequence of all the words in the sentence before t.
- ( ) c is the one word that comes immediately before t.
- ( ) c is a sequence of several words immediately before t.
- (•) c and t are chosen to be nearby words.

Correct

8. Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings. The word2vec model uses the following softmax function: **1 / 1 point**

$$P(t \mid c) = \frac{e^{\theta_t^T e_c}}{\sum_{t'=1}^{10000} e^{\theta_{t'}^T e_c}}$$

Which of these statements are correct? Check all that apply.

- [x] θ_t and e_c are both 500 dimensional vectors.

Correct

- [ ] θ_t and e_c are both 10000 dimensional vectors.
- [x] θ_t and e_c are both trained with an optimization algorithm such as Adam or gradient descent.

Correct

- [ ] After training, we should expect θ_t to be very close to e_c when t and c are the same word.

9. Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings.The GloVe model minimizes this objective: **1 / 1 point**

$$\min \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij})(\theta_i^T e_j + b_i + b_j' - log X_{ij})^2$$

Which of these statements are correct? Check all that apply.

- [ ] θ_i and e_j should be initialized to 0 at the beginning of training.
- [x] θ_i and e_j should be initialized randomly at the beginning of training.

Correct

- [x] X_{ij} is the number of times word j appears in the context of word i.

Correct

- [x] The weighting function $f(.)$ must satisfy $f(0) = 0$.

Correct
The weighting function helps prevent learning only from extremely common word pairs. It is not necessary that it satisfies this function.

10. You have trained word embeddings using a text dataset of m_1 words. You are considering using these word embeddings for a language task, for which you have a separate labeled dataset of m_2 words. Keeping in mind that using word embeddings is a form of transfer learning, under which of these circumstance would you expect the word embeddings to be helpful? **1 / 1 point**

- (•) $m_1 >> m_2$

$m_1 << m_2$

Correct

Identifying and quantifying bias in word embeddings

- Assumption: The aspect of bias is known. E.g. gender
- Find the "gender" dimension
  - Collect explicit gender-based word pairs (f, m): (woman, man), (mother, father), (gal, guy), (girl, boy), (she, he)
  - Get the gender dimension as (f-m) [How?]
- Collect a set N of gender neutral words

- Compute the gender component g in elements from N
  - DirectBias = $(1/|N|)\sum_{w \in N}|\cos(w,g)|$

Identifying and quantifying bias in word embeddings

- How to capture indirect bias?
- Direct bias: component along gender dimension
- Indirect bias: Component along its perpendicular

- Need to find the component to the perpendicular of the "gender" dimension
- Component of vector a along vector b:
  - Scalar Component: $comp_b(a) = (a.b)/|b|$
  - Vector component: $comp_b(a).b$
- $w_g = (w.g)g$, $w_\perp = w - w_g$

- IndirectBias $B(w,v) = (w.v - (w_\perp - v_\perp)/(|w_\perp| - |v_\perp|)) / (w.v)$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - log(X_{ij})\right)^2 + \lambda \cos(w_i, g) + \gamma \cos(\tilde{w}_j, g)$$

A simple technique for debiasing GloVE

(a) (2 points) Which of the following statement about Skip-gram are correct?

- ○ It predicts the center word from the surrounding context words
- √ **The final word vector for a word is the average or sum of the input vector v and output vector u corresponding to that word**
- ○ When it comes to a small corpus, it has better performance than GloVe
- ○ It makes use of global co-occurrence statistics

(c) (4 points) Word2Vec represents a family of embedding algorithms that are commonly used in a variety of contexts. Suppose in a recommender system for online shopping, we have information about co-purchase records for items $x_1, x_2, \ldots, x_n$ (for example, item x_i is commonly bought together with item x_j). Explain how you would use ideas similar to Word2Vec to recommend similar items to users who have shown interest in any one of the items.

Solution: We can treat items that are copurchased with x to be in the 'context' of item x (1 point). We can use those copurchase records to build item embeddings akin to Word2Vec. (2 points, you need to mention that item embeddings are created). Then we can use a similarity metric such as finding the items with the largest cosine similarity to the average basket to determine item recommendations for users (1 point).

g. (2 points) Consider the skip-gram model

$$P(\text{context} = y|\text{word} = x) = \frac{\exp(\mathbf{v}_x \cdot \mathbf{c}_y)}{\sum_{y'} \exp(\mathbf{v}_x \cdot \mathbf{c}_{y'})}$$

What is the big-O runtime of computing a single probability $P(\text{context} = c|\text{word} = w)$ under this model? Express this in terms of the dimensionality d of the vectors and the vocabulary size $|V|$.

$O(dv)$. The dot product is linear in d and not quadratic, which was one common mistake

h. (3 points) In skip-gram, we can efficiently approximate the denominator if for a given word we know what the largest terms are in its denominator. Suppose you are given a large corpus of text taken from the web. For a particular word w, how could you go about identifying 10 unique context words that are likely to have high weights in the denominator?

Count words, pick the words that most frequently occur with the given word. Some students gave answers that amounted to just computing all of the weight vectors explicitly. This is true, but not in the spirit of approximating the denominator without computing it explicitly; pre-computing counts from a large corpus is quite fast compared to doing explicit computation of all denominators.

3. Word Vectors (12 points)

(a) (3 points) Although pre-trained word vectors work very well in many practical downstream tasks, in some settings it's best to continue to learn (i.e. 'retrain') the word vectors as parameters of our neural network. Explain why retraining the word vectors may hurt our model if our dataset for the specific task is too small.

Solution: See TV/television/telly example from lecture 4. Word vectors in training data move around; word vectors not in training data don't move around. Destroys structure of word vector space. Could also phrase as an overfitting or generalization problem.

(b) (2 points) Give 2 examples of how we can evaluate word vectors. For each example, please indicate whether it is intrinsic or extrinsic.

Solution: Intrinsic: Word Vector Analogies ; Word vector distances and their correlation with human judgments.

Extrinsic: Name entity recognitions: finding a person, location or organization and so on.

(c) (4 points) In lectures, we saw how word vectors can alternatively be learned via co-occurrence count-based methods. How does Word2Vec compare with these methods? Please briefly explain one advantage and one disadvantage of the Word2Vec model.

Solution:

Advantage of word2vec: scale with corpus size; capture complex patterns beyond word similarity

Disadvantage: slower than occurrence count based model; efficient usage of statistics

(d) (3 points) Alice and Bob have each used the Word2Vec algorithm to obtain word embeddings for the same vocabulary of words V. In particular, Alice has obtained 'context' vectors $\mathbf{u}_w^A$ and 'center' vectors $\mathbf{v}_w^A$ for every $w \in V$, and Bob has obtained 'context' vectors $\mathbf{u}_w^B$ and 'center' vectors $\mathbf{v}_w^B$ for every $w \in V$.

Suppose that, for every pair of words $w, w' \in V$, the inner product is the same in both Alice and Bob's model: $(\mathbf{u}_w^A)^T \mathbf{v}_{w'}^A = (\mathbf{u}_w^B)^T \mathbf{v}_{w'}^B$. Does it follow that, for every word $w \in V$, $\mathbf{v}_w^A = \mathbf{v}_w^B$? Why or why not?

Solution: No. Word2Vec model only optimizes for the inner product between word vectors for words in the same context.

One can rotate all word vectors by the same amount and the inner product will still be the same. Alternatively one can scale the set of context vectors by a