Singular value Decomposition:

$$A = U \Sigma V^T$$

$\Rightarrow$ The columns of U are orthonormal eigenvectors of AA^T

$\Rightarrow$ The columns of V are orthogonal eigenvectors of A^TA

$\Rightarrow$ Eigenvalues $\lambda_1, \lambda_2 \cdots \lambda_r$ of AA^T are the eigenvalues of A^TA.

$$\sigma_i = \sqrt{\lambda_i}$$

$$\Sigma^i = \text{diag}(\sigma_1, \sigma_2 \cdots \sigma_n)$$

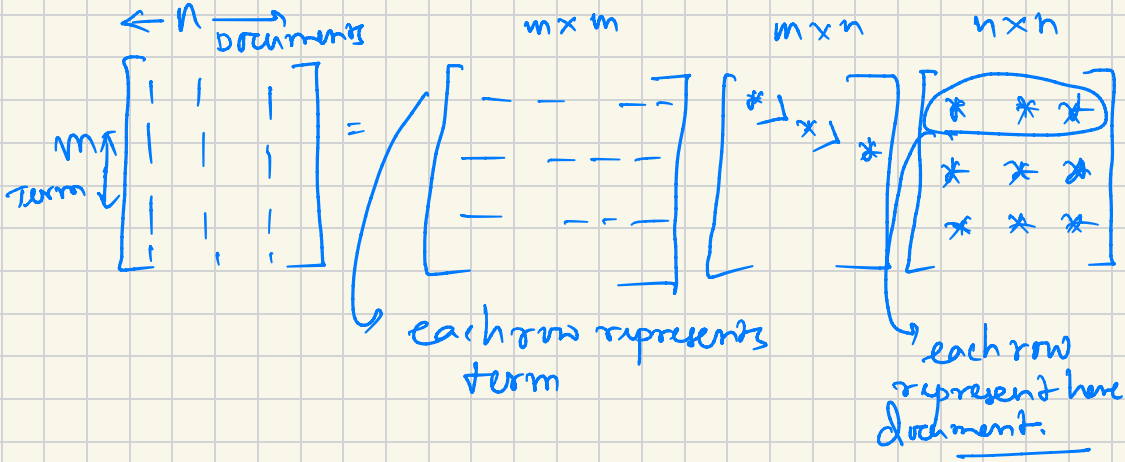
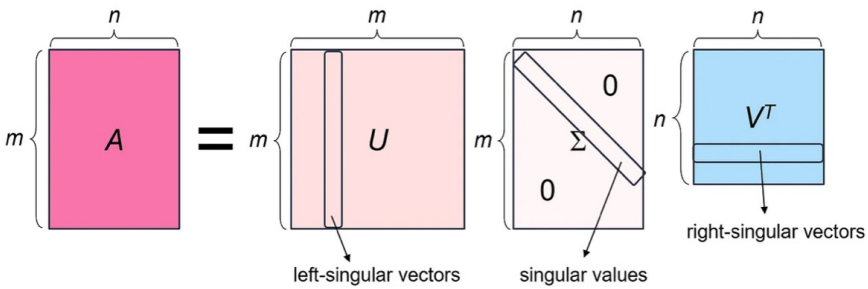


Image by Peggy und Marco Lachmann-Anke from Pixabay

Mathematical Definition

The singular value decomposition of an $m \times n$ real matrix A is a factorization of the form $A = U\Sigma V^t$, where:

- U is an $m \times m$ **orthogonal matrix** (i.e., its columns and rows are orthonormal vectors). The columns of U are called the **left-singular vectors** of A.
- Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal. The diagonal entries $\sigma_i = \Sigma_{ii}$ are known as the **singular values** of A and are typically arranged in descending order, i.e., $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$. The number of the non-zero singular values is equal to the rank of A.
- V is an $n \times n$ orthogonal matrix. The columns of V are called the **right-singular vectors** of A.



SVD of a matrix

Every matrix has a singular value decomposition (a proof of this statement can be found here). This is unlike eigenvalue decomposition, for example, which can be applied only to squared diagonalizable matrices.

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

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

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

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

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

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

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

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

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:

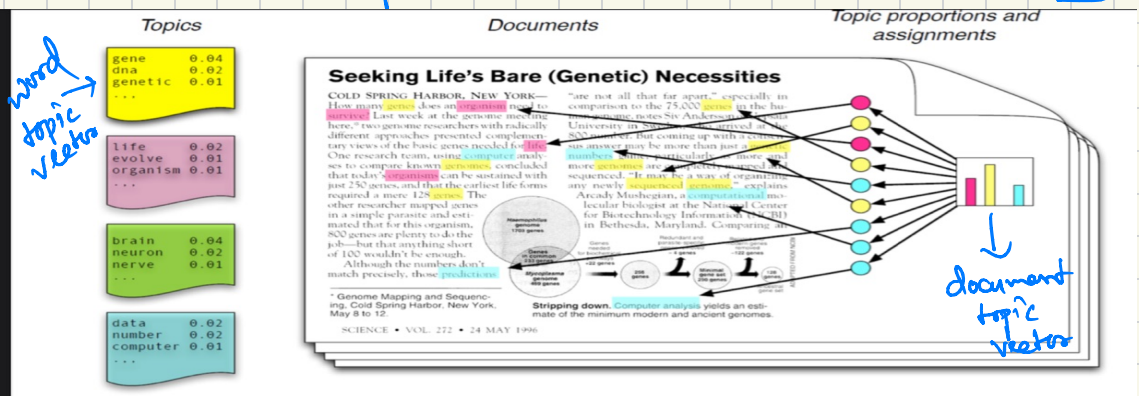
Documents with similar topics use similar groups of words

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.

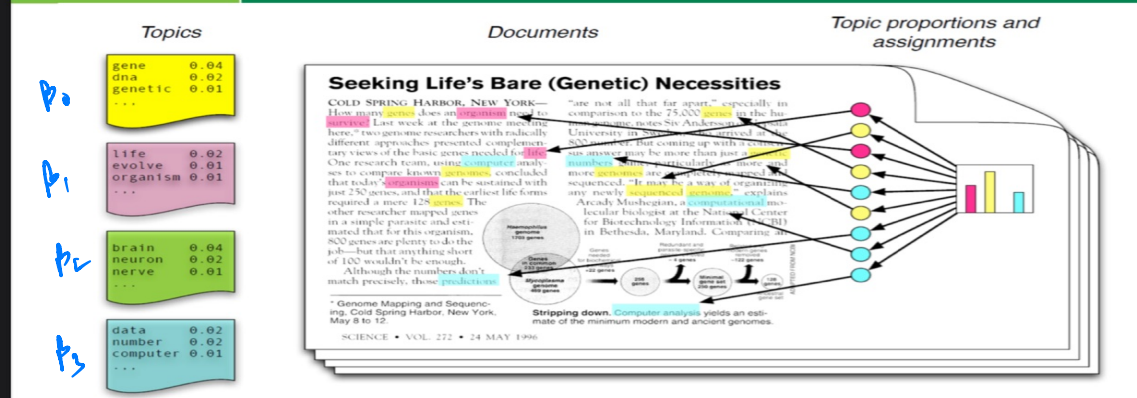
→ map words to topic.

→ 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.

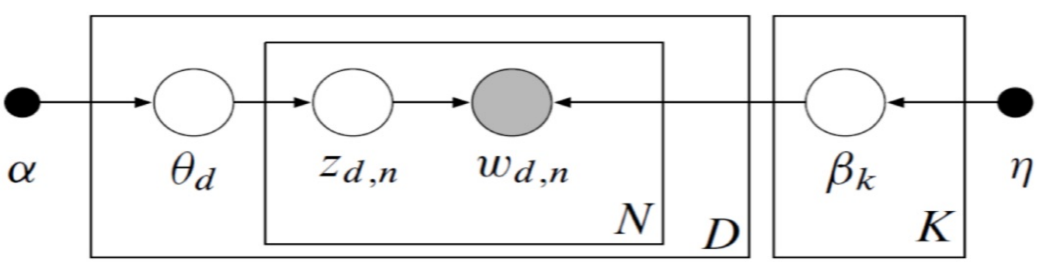
2.1 select topic $z_i \sim \theta_d$ (multinomial dist.)

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

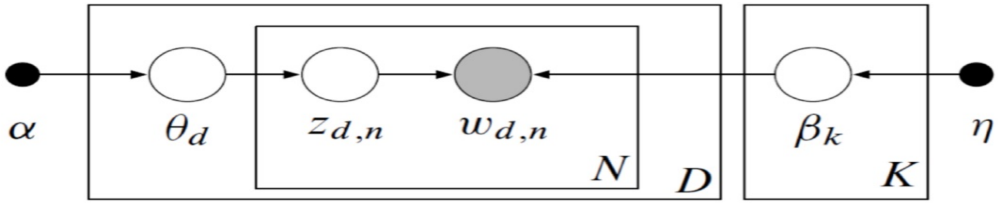
repeat for n words

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)

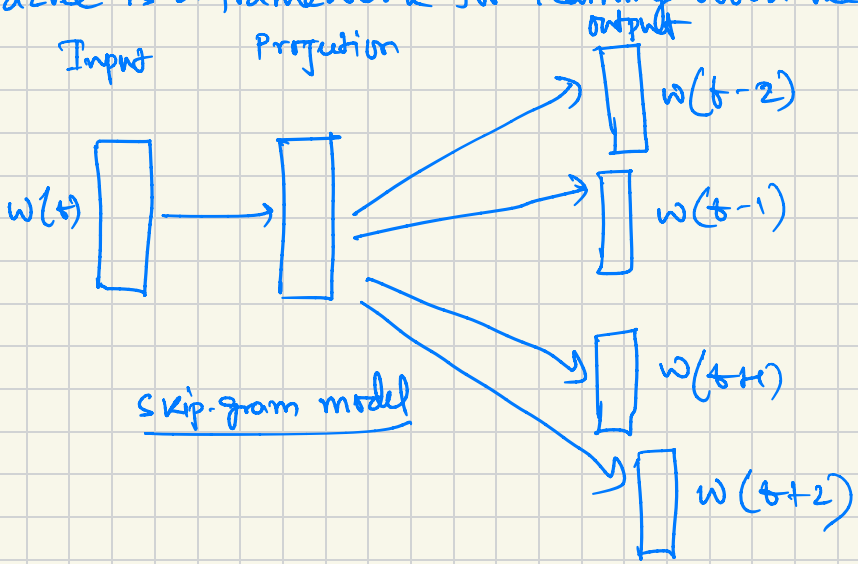
	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

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

** TF-IDF is the alternate to PMI.

Word2Vec

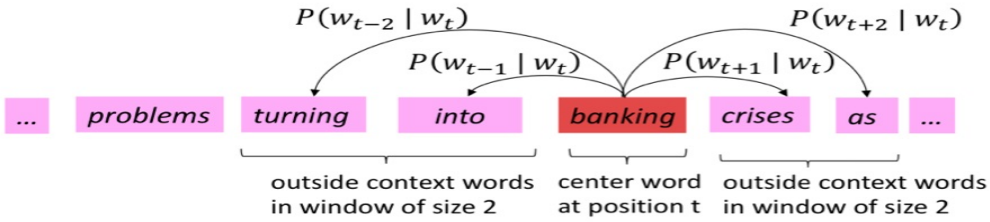
word2vec is a framework for learning word vectors.



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:

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

W = O:

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

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 increase 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:

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

Affinity of word c for context o

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:

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

Affinity of word c for context o

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 Cooccurrence of words one at a time.

Why not to capture Cooccurrence Counts directly?

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