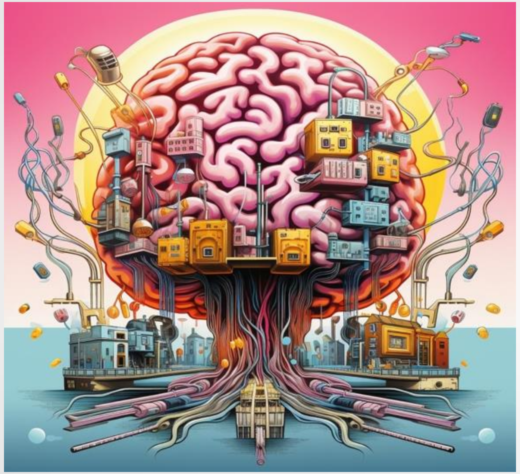


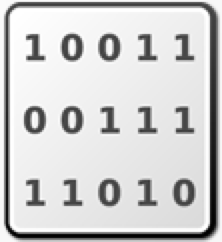
Chapter 3: Feature Extraction

Unit: Advanced Deep Learning

What is Feature Extraction ?

- **Definition:** Feature extraction is the process of transforming raw text into a structured format that can be used by machine learning models.
- **Importance:** Converts textual data into numerical vectors, capturing essential information for model training.
- **Overview:** Techniques such as TF-IDF, word embeddings (Word2Vec, GloVe, FastText) are methods used to represent text data.

‘I Love AI’ →



Text Representation

- Accurate text representation is crucial in NLP.
- By converting textual data into numerical format, machines can understand and analyze language.
- The choice of representation method directly affects the performance and accuracy of NLP models.
- Several text representation techniques are commonly used in NLP:
  - Bag-of-words,
  - TF-IDF,
  - Word embeddings

Bag-of-words Approach

- The bag-of-words treats each document as a collection of individual words and ignores grammar and word order.
- Words are converted into a numerical vector representation based on their frequency of occurrence.
- While this approach is simple and efficient, it lacks contextual understanding and cannot capture the semantic meaning of words.
- Example : the cat sat on the mat.

cat	sat	on	the	mat	quickly
1	1	1	2	1	0

Bag-of-words : Key Challenges

Sparse Input

- Each word is represented as a feature in a high-dimensional vector, with a 1-hot encoding indicating the presence of a word
- **Example** : "cat" = (1,0,0,...), "dog" = (0,1,0,...).
- **Issue**: For large vocabularies, most values in these vectors are 0, leading to sparse data.
- **Impact**: Sparse input can cause overfitting due to the high dimensionality of features compared to the number of examples.

Bag-of-words : Key Challenges

Lack of Semantic Generalization:

- The model treats each word as a distinct feature without understanding their relationships (e.g., "cat" and "dog" are treated independently).
- **Issue**: The model cannot generalize or apply knowledge from one feature to another, missing out on inherent similarities (e.g., both being animals).
- **Ideal Scenario**: A model would benefit from understanding relationships and similarities between words to handle tasks more effectively.

Reweighting

- Reweighting involves adjusting the importance or weight of certain features in a text to improve model performance and accuracy.
- **Purpose:** To address issues like imbalanced data, over-representation of common words, or under-representation of significant words.
- **Example:**
- "The cat sat on the mat"

cat	sat	on	the	mat	quickly
1	1	1	2	1	0

Techniques for Reweighting

- **Term Frequency-Inverse Document Frequency (TF-IDF):** Weighs words based on their frequency in a document relative to their frequency across the entire corpus.

- **Class Imbalance Reweighting:** Adjusts weights based on the frequency of classes in a classification problem.

- **Word Embedding Reweighting:** Alters the importance of words in embedding spaces based on context or semantic relevance.

- **Attention Mechanisms:** Uses neural networks to dynamically adjust the focus on different parts of the input text.

TF-IDF (Term Frequency-Inverse Document Frequency)

- **Term Frequency (TF):** Measures how often a word appears in a document.

$$TF(t,d) = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}$$

- **Inverse Document Frequency (IDF):** Measures how important a word is across all documents.

$$IDF(t,D) = \log \frac{\text{Total number of documents in corpus D}}{\text{Number of documents containing term t}}$$

- **TF-IDF**

$$TFIDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

TF-IDF (Term Frequency-Inverse Document Frequency)

- Example :
  - Doc1: ‘AI is transforming Technology’
  - Doc2: ‘Deep learning is a subset of AI’
- Step 1: Calculate TF values

$$TF(t,d) = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}$$

Doc1	Count	TF calculation	TF value
AI	1	1/4	**0,25**
is	1	1/4	**0,25**
transforming	1	1/4	**0,25**
Technology	1	1/4	**0,25**

Doc2	Count	TF calculation	TF value
Deep	1	1/7	**0,14**
Learning	1	1/7	**0,14**
is	1	1/7	**0,14**
a	1	1/7	**0,14**
Subset	1	1/7	**0,14**
Of	1	1/7	**0,14**
AI	1	1/7	**0,14**

TF-IDF (Term Frequency-Inverse Document Frequency)

- Step2: Calculate IDF (Inverse Document Frequency)

$$IDF(t, D) = \log \frac{\text{Total number of documents in corpus D}}{\text{Number of documents containing term t}}$$

Term	DF	IDF calculation	IDF value
AI	2	Log(2/2)	**0**
is	2	Log(2/2)	**0**
transforming	1	Log(2/1)	**0,30**
Technology	1	Log(2/1)	**0,30**
Deep	1	Log(2/1)	**0,30**
Learning	1	Log(2/1)	**0,30**
a	1	Log(2/1)	**0,30**
Subset	1	Log(2/1)	**0,30**
Of	1	Log(2/1)	**0,30**

TF-IDF (Term Frequency-Inverse Document Frequency)

- Step3: Calculate TF-IDF scores

$$TFIDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

Term (Doc1)	TF	IDF	TF-IDF	**TF-IDF Score**
AI	0.25	0	0.25×0	**0.00**
is	0.25	0	0.25×0	**0.00**
transforming	0.25	0.30	0.25×0.30	**0.075**
technology	0.25	0.30	0.25×0.30	**0.075**

Term (Doc2)	TF	IDF	TF-IDF	**TF-IDF Score**
Deep	0,14	0,30	0,14x0,30	**0.42**
Learning	0,14	0,30	0,14x0,30	**0.42**
is	0,14	0	0,14x0	**0**
A	0,14	0.30	0,14x0,30	**0.42**
subset	0,14	0,30	0,14x0,30	**0,42**
of	0,14	0,30	0,14x0,30	**0,42**
AI	0,14	0	0,14x0	**0**

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used technique for representing words in a document as vectors. It quantifies the importance of a word in a document relative to its frequency in a corpus of documents.

1 **Advantages**

TF-IDF is simple to implement and computationally inexpensive.

2 **Disadvantages**

It doesn't capture semantic relationships between words. It suffers from the curse of dimensionality, leading to sparse vectors.

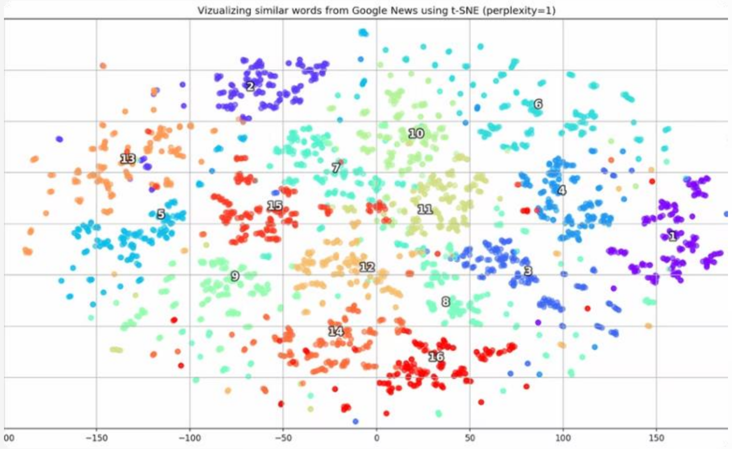
Word Embeddings

- Word embedding is a technique that maps words to continuous vectors in a lower-dimensional space. These vectors capture **semantic relationships** between words, allowing machines to understand their **meaning**.

- Word embeddings are learned from large text corpora, allowing them to capture **contextual** information and relationships between words. These embeddings are essential for various NLP tasks such as text classification, machine translation, and sentiment analysis.

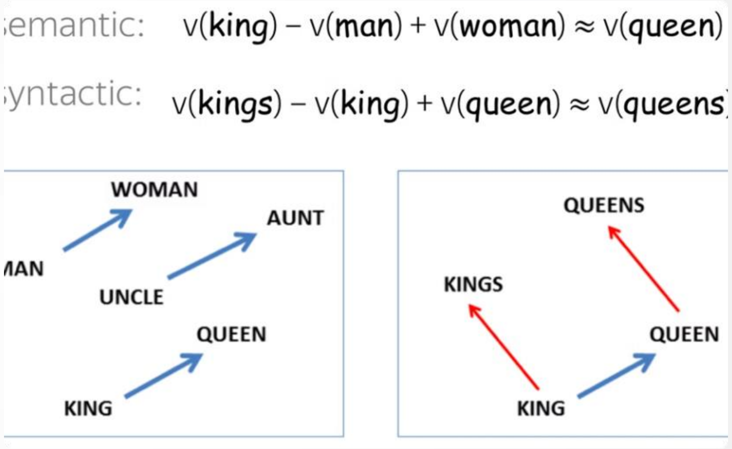
- Various embedding techniques like **Word2Vec**, **GloVe**, and **FastText**.

Word Embeddings



Semantic Similarity

Words with similar meanings are often located close together in the embedding space.



Analogical Reasoning

Word embeddings can be used to perform analogical reasoning. For example, "king" - "man" + "woman" = "queen."

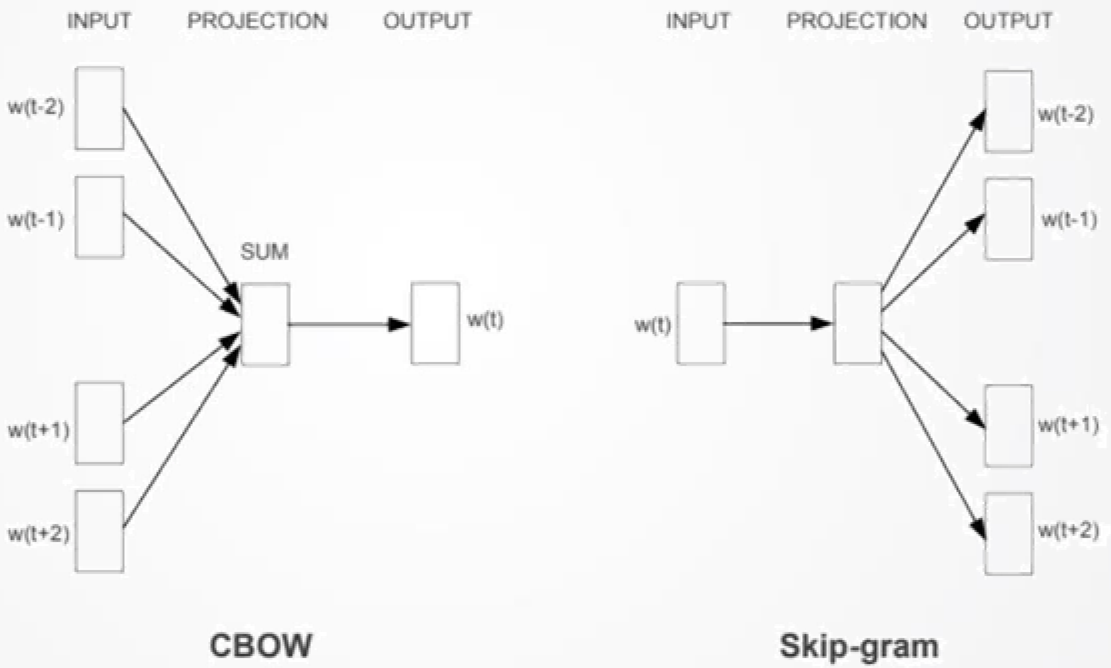
Basic word embedding methods

- Word2vec (Google, 2013)
  - Continuous bag-of-words (CBOW)
  - Skip-gram
- Global vectors (GLoVe) (Stanford, 2014)
- FastText (Facebook, 2016)
  - Supports out-of-vocabulary (OOV) words

Word2Vec

- Word2vec is a popular technique for learning word embeddings. It uses a neural network to capture semantic relationships between words.
- Word2vec consists of two main architectures:
  - **Continuous Bag of Words (CBOW):** Predicts a target word based on its surrounding context words. Useful for smaller datasets.
  - **Skip-gram:** Predicts context words based on a target word. Effective for larger datasets and capturing rare words.

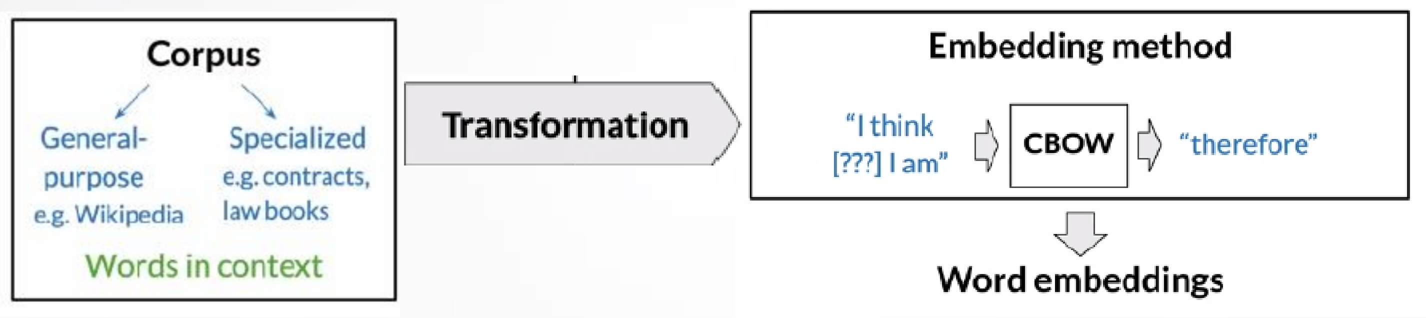
Word2Vec


INPUT
PROJECTION
OUTPUT
w(t-2)
w(t-1)
SUM
w(t)
w(t+1)
w(t+2)
CBOW
INPUT
PROJECTION
OUTPUT
w(t)
w(t-2)
w(t-1)
w(t+1)
w(t+2)
Skip-gram

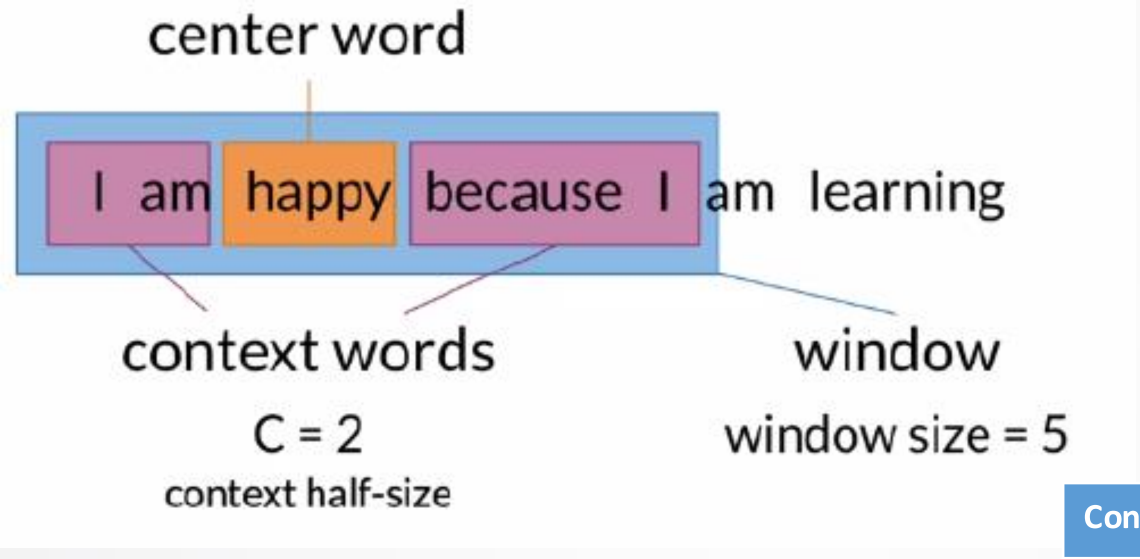
Word2vec

- **Training Process**:
  - Word2Vec learns word embeddings through an iterative training process using a large corpus of text data.
  - During training, the model adjusts the vector representations of words to maximize the likelihood of predicting words,
- **Semantic Nature of Word2Vec Embeddings:**
  - Word2Vec embeddings are renowned for their semantic properties, as they encode semantic relationships between words based on their contextual usage within text.

CBOW word embedding process



Creating training example



Context words	Center word
I am because I	happy
Am happy I am	because
Happy because am learning	I

Transforming center words into vectors

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

One-hot vector

$$
\begin{array}{r}
\\ \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning}
\end{array}
\begin{array}{c}
\text{am} \\ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{because} \\ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{happy} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{I} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{learning} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
\end{array}
$$

Average of individual one-hot vectors

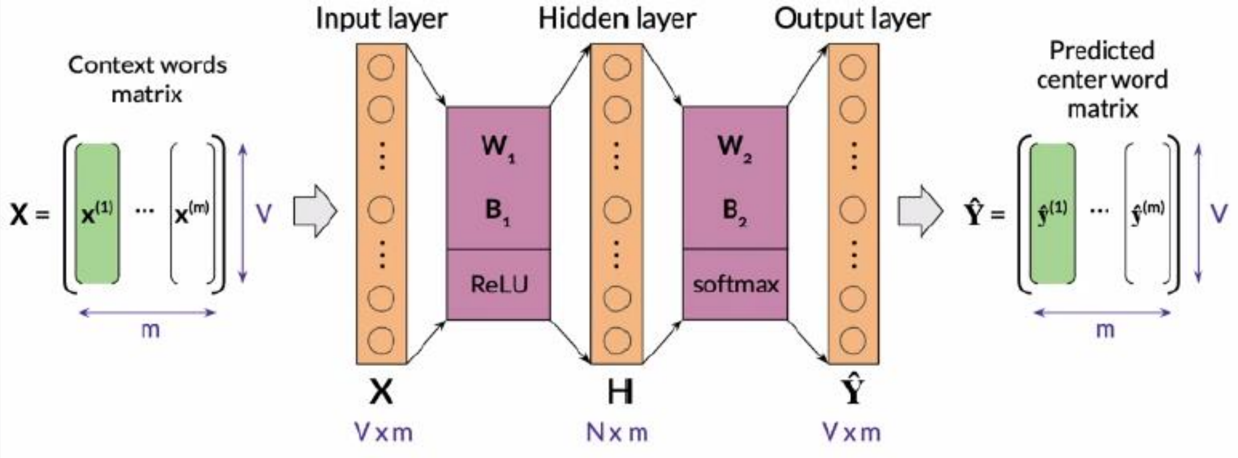
$$
\left(
\begin{array}{r}
\\ \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning}
\end{array}
\begin{array}{c}
\text{I} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}
\end{array}
+
\begin{array}{c}
\text{am} \\ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{array}
+
\begin{array}{c}
\text{because} \\ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{array}
+
\begin{array}{c}
\text{I} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}
\end{array}
\right) / 4 =
\begin{array}{c}
\text{I am because I} \\ \begin{pmatrix} 0.25 \\ 0.25 \\ 0 \\ 0.5 \\ 0 \end{pmatrix}
\end{array}
$$

training dataset

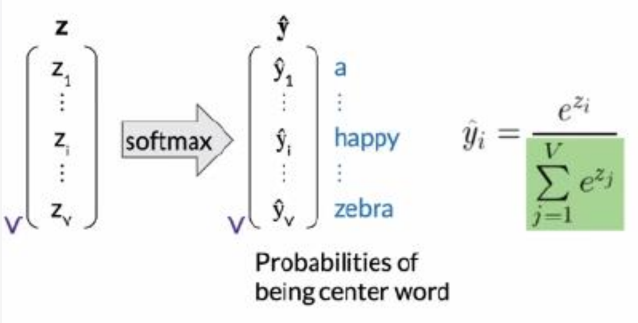
Context words	Context words vector	*Center word*	Center word vector
I am because I	[0.25; 0.25; 0; 0.5; 0]	*happy*	[0; 0; 1; 0; 0]
am happy I am	[0.5; 0; 0.25; 0.25; 0]	*because*	[0; 1; 0; 0; 0]
happy because am learning	[0.25; 0.25; 0.25; 0; 0.25]	*I*	[0; 0; 0; 1; 0]

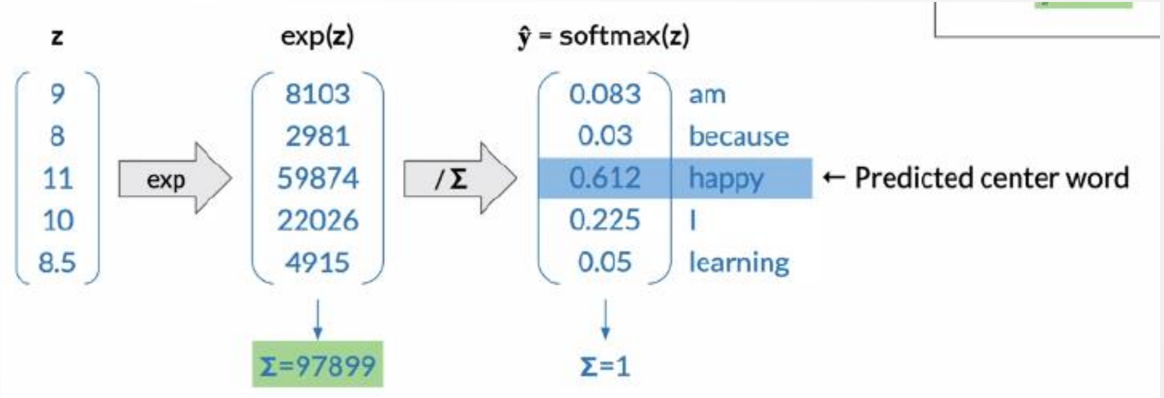
Architecture CBOW

- Hyperparameters : N : word embedding size

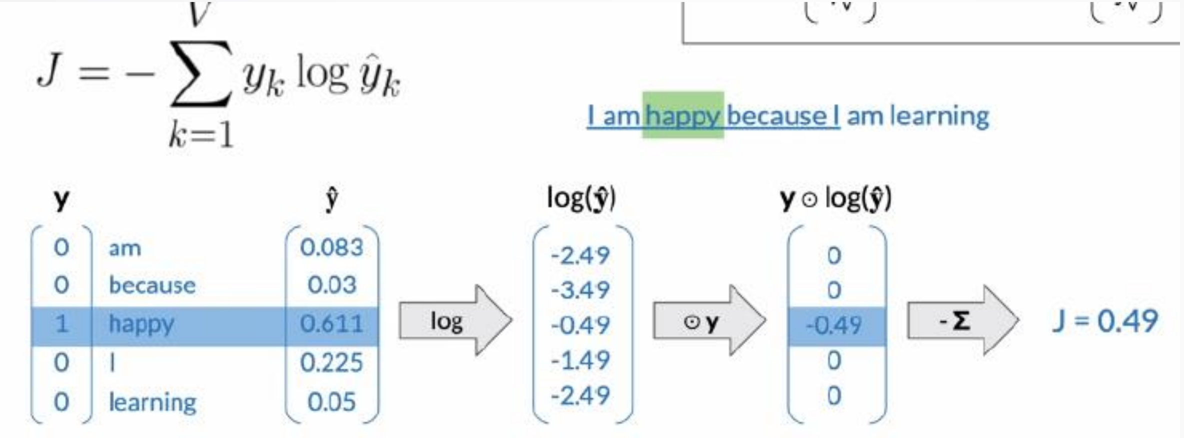


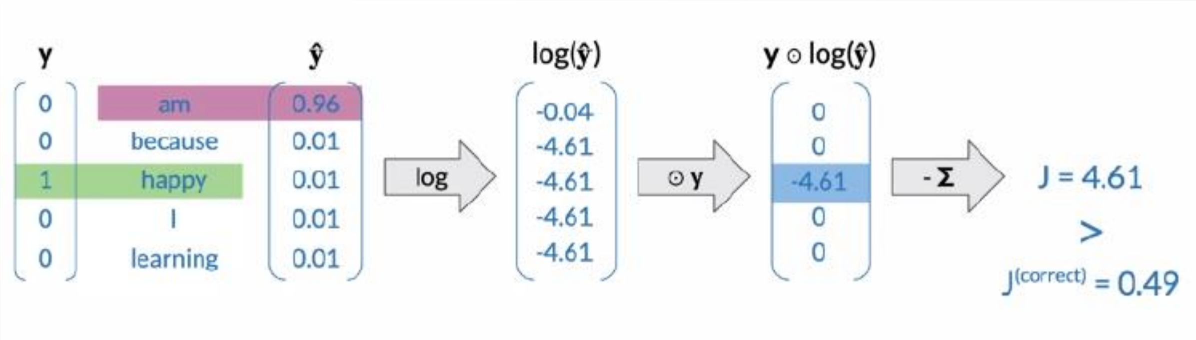
Softmax


z
z_1
z_i
z_V
V
softmax
ŷ
$\hat{y}_1$
$\hat{y}_i$
$\hat{y}_V$
a
happy
zebra
$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$
Probabilities of
being center word


z
9
8
11
10
8.5
exp
exp(z)
8103
2981
59874
22026
4915
Σ=97899
/ Σ
ŷ = softmax(z)
0.083
0.03
0.612
0.225
0.05
Σ=1
am
because
happy
I
learning
← Predicted center word

Cross-entropy loss





Cost

Cost: mean of losses

$$J_{batch} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{V} y_j^{(i)} \log \hat{y}_j^{(i)}$$

$$J_{batch} = -\frac{1}{m}\sum_{i=1}^{m} J^{(i)}$$

Predicted center word matrix

$$\hat{\mathbf{Y}} = \begin{bmatrix} \hat{\mathbf{y}}^{(1)} & \cdots & \hat{\mathbf{y}}^{(m)} \end{bmatrix}$$

Actual center word matrix

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^{(1)} & \cdots & \mathbf{y}^{(m)} \end{bmatrix}$$

- Minimizing the cost :
  - Backpropagation: calculate partial derivatives of cost respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W_1}}, \frac{\partial J_{batch}}{\partial \mathbf{W_2}}, \frac{\partial J_{batch}}{\partial \mathbf{b_1}}, \frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$

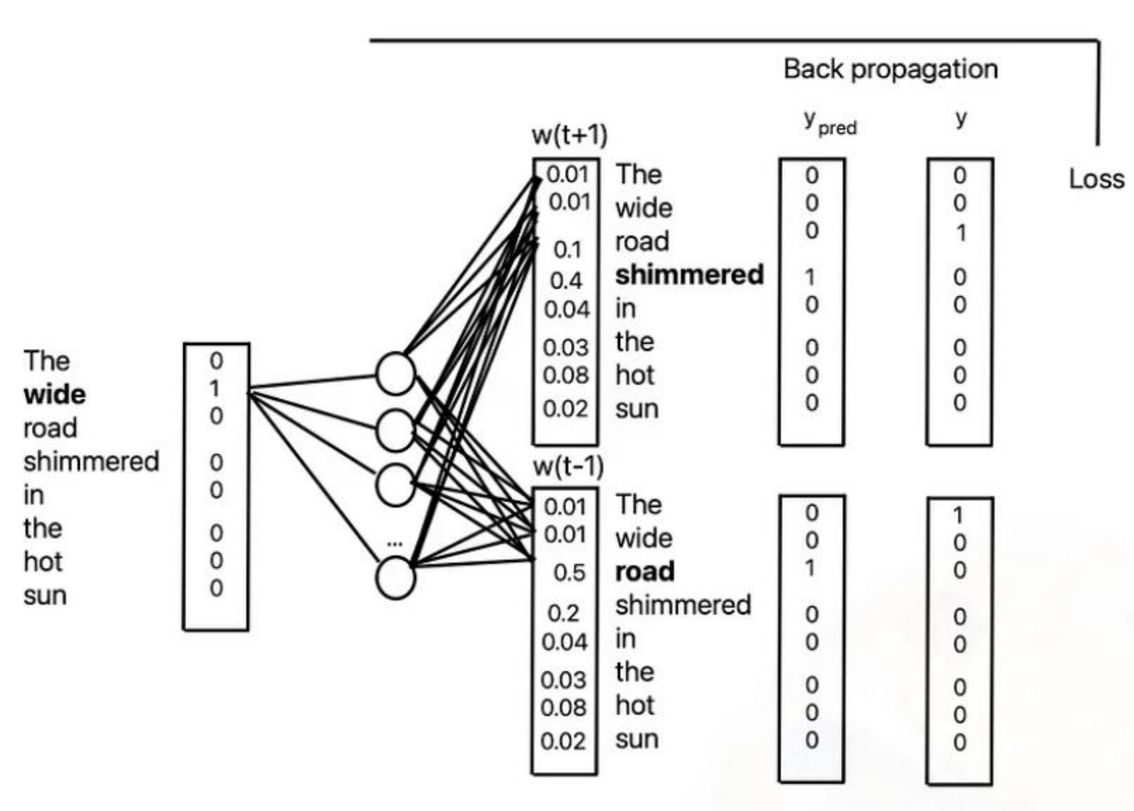
  - Gradient descent: update weights and biases.

Skip-Gram

- The skip-gram variant takes a target word and tries to predict the surrounding context words
- Example :
- The wide road shimmered in the hot sun.

Window Size	Text	Skip-grams
2	[The **wide** road shimmered] in the hot sun.	wide, the wide, road wide, shimmered
	The [wide road **shimmered** in the] hot sun.	shimmered, wide shimmered, road shimmered, in shimmered, the
	The wide road shimmered in [the hot **sun**].	sun, the sun, hot
3	[The **wide** road shimmered in] the hot sun.	wide, the wide, road wide, shimmered wide, in
	[The wide road **shimmered** in the hot] sun.	shimmered, the shimmered, wide shimmered, road shimmered, in shimmered, the shimmered, hot
	The wide road shimmered [in the hot **sun**].	sun, in sun, the sun, hot

Skip-Gram architecture



Skip-Gram algorithm

- For each word t = 1 … T, we predict the surrounding words in a window of "radius" m.
- We train a machine learning model to maximize the probability of any context word given the current centre word.

$$J' = \prod_{t=1}^{T} \prod_{-m \leq j \leq m} P(w_{t+j} | w_t)$$

minimize the negative log likelihood.

$$J = -\sum_{t=1}^{T} \sum_{-m \leq j \leq m} log(P(w_{t+j} | w_t))$$

Skip-Gram algorithm

- W_{t+j} : context , w_j : center
- P(context,center) can be formulated as a Softmax function.

$$P(context|center) = \frac{exp(u_{context}^T v_{center})}{\sum_{\omega \in vocab} exp(u_{\omega}^T v_{center})}$$

- Where

$$u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$$

GloVe (Global Vectors for Word Representation)

- GloVe generates word embeddings by factorizing the word co-occurrence matrix from a corpus.
- **Advantages:** Captures global statistical information about word usage.
- **Difference from Word2Vec:** GloVe models global co-occurrence, while Word2Vec models local context.
- The co-occurence matrix provides a quantitative measure of the semantic affinity between words by capturing the frequency with which they appear together in a given context.

How Glove works

- **Training Data: Sentences:**
  1. "AI is revolutionizing technology."
  2. "Technology is advancing rapidly.“

  ➢ **Step 1: Construct a Co-occurrence Matrix**

- **Window size =1**

Word	AI	IS	revolutionizing	Technology	Advancing	rapidly
AI	0	1	0	0	0	0
IS	1	0	1	1	1	0
revolutionizing	0	1	0	1	0	0
Technology	0	1	1	0	0	0
Advancing	0	1	0	0	0	1
rapidly	0	0	0	0	1	0

How Glove works

➢ **Step 2: Compute the Co-occurrence Ratios**

- Compute the ratios of co-occurrence counts to capture relative frequencies.
- The co-occurrence probability between two words i and j is defined as follows:

$$P_{ij} = \frac{X_{ij}}{\sum_k X_{ik}}$$

- Where
  - X_{ij} is the number of times word i appears in the context of word j,
  - $\sum_k X_{ik}$ is the sum of co-occurrences of i with all other words k in the corpus.

How Glove works

- GloVe Ratio Based on Co-occurrence Probability

$$\frac{P_{ik}}{P_{jk}} = \frac{X_{ik} / \sum_l X_{il}}{X_{jk} / \sum_l X_{jl}}$$

- P_{ik} is the probability that word k appears in the context of word i,
- P_{jk} is the probability that word k appears in the context of word j.

➢ **Step 3: Compute the Co-occurrence Ratios**

- Glove aims to minimize a loss function that measures the error between the dot product of the vectors and log(Xij)

$$J = \sum_{i,j} f(X_{ij}) \left(w_i^T w_j + b_i + b_j - \log(X_{ij})\right)^2$$

Where w_i and w_j are the word vectors for words i and j, b_i and b_j are bias terms, and $f(X_{ij})$ is a weighting function.

How Glove works

- Logarithm of Co-occurrences: Since co-occurrences can have a wide variability (some words are very frequent while others are very rare), we apply the logarithm to reduce the impact of very large values.

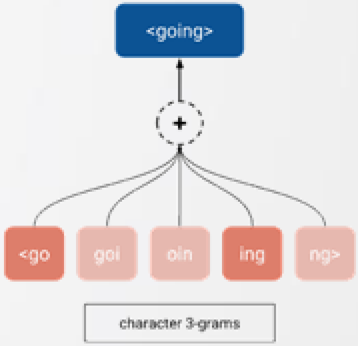
$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{X_{\max}}\right)^{\alpha} & \text{si} \quad X_{ij} < X_{\max} \\ 1 & \text{si} \quad X_{ij} \geq X_{\max} \end{cases}$$

- X_{max} is a hyperparameter that controls how much frequent co-occurrences are taken into account.
- α is another hyperparameter, usually set to 0.75.
  - For frequent co-occurrences (Xij≥Xmax), the function f(Xij)=1 : co-occurrences are fully taken into account.
  - For rare co-occurrences (Xij<Xmax), the function f(Xij) gradually decreases the weighting of co-occurrences as they become rarer.
- Training :
  - Use optimization techniques (e.g., stochastic gradient descent) to update the word vectors to minimize the objective function.
  - After training, each word will be represented by a dense vector capturing the semantic meaning based on co-occurrences in the corpus.

FastText

- FastText is an advanced word embedding technique developed by Facebook AI Research (FAIR) that extends the Word2Vec model.
- Unlike Word2Vec, FastText not only considers whole words but also incorporates subword information — parts of words like n-grams.
- This approach enables the handling of morphologically rich languages and captures information about word structure more effectively.

Each word is represented as a bag of character n-grams in addition to the word itself, so for example, for the word matter, with n = 3, the fastText representations for the character n-grams is <ma, mat, att, tte, ter, er>.



FastText

- FastText incorporates the subword information during training. The neural network in FastText is trained to predict words not just based on the target words but also based on these n-grams.
- **Advantages:**
  - Better representation of rare words.
  - Capable of handling out-of-vocabulary words.
  - Richer word representations due to subword information.
- **Disadvantages:**
  - Increased model size due to n-gram information.
  - Longer training times compared to Word2Vec.

Comparing TF-IDF, Word2Vec, GloVe, and FastText

- **TF-IDF:** Simple, interpretable, captures term importance but not semantics.
- **Word2Vec:** Captures word meanings, requires large datasets, local context.
- **GloVe:** Captures global co-occurrence information, good for larger corpora.
- **FastText:** Handles morphologically rich languages, improves rare word representation.

word embedding evaluation

- **Intrinsic Evaluation**
  - Word Similarity: Measure the similarity between word embeddings and compare them to human-annotated similarity scores.
  - Word Analogies: Test the ability of word embeddings to solve analogy tasks, such as “king – man + woman = queen,”
  - Word Clustering
- **Extrinsic Evaluation**
  - Sentiment Analysis
  - Named Entity Recognition
  - Machine Translation
- **Word Embedding Visualization :** Visualize word embeddings in a lower-dimensional space (e.g., 2D or 3D) using techniques like t-SNE or PCA. Examine the spatial relationships between words and inspect clusters or semantic groupings.

Intrinsic Evaluation of Word Embeddings

Visualization

Word Vectors are high dimensions (usually ~100)

➔ **Project** the word embedding vectors using PCA or T-SNE
➔ **Visualize** in 2D or 3D
➔ **Analyse** the clusters

Intrinsic Evaluation of Word Embeddings

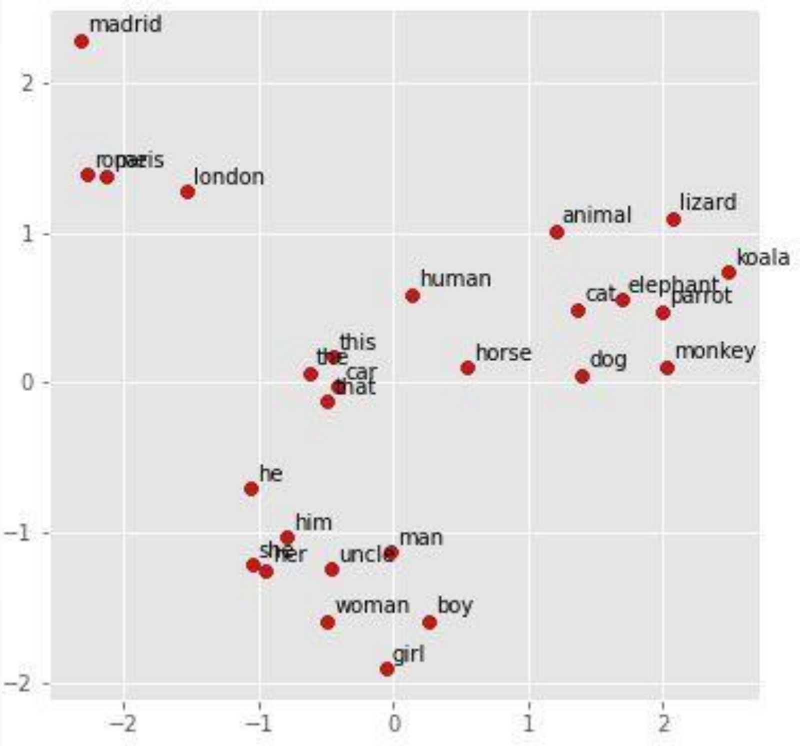


Figure: Visualize skip-gram trained on Wikipedia (1B tokens) (fastext.cc) vectors with PCA

Intrinsic Evaluation of Word Embeddings

We can compare the similarity between words in the embedding space with human judgment

1. Collect Human Judgment (or download dataset e.g. WordSim353) on a list of pairs of words
2. Compute similarity of the word vectors of those pairs
3. Measure correlation between both

Word 1	Word 2	Word2vec Cosine Similarity	Human Judgment
tiger	tiger	1.0	10
dollar	buck	0.3065	9.22
dollar	profit	0.3420	7.38
smart	stupid	0.4128	5.81

Intrinsic Evaluation of Word Embeddings

How to measure similarity in the word embedding space?

- **Cosine Similarity**

$$sim(w_1, w_2) = cos(x_{w_1}, x_{w_2}) = \frac{x_{w_1}^T x_{w_2}}{||x_{w_1}|| \, ||x_{w_2}||}$$

- **L2 Distance**

$$sim(w_1, w_2) = L2(x_{w_1}, x_{w_2}) = ||x_{w_1} - x_{w_2}||$$

Intrinsic Evaluation of Word Embeddings

Nearest-Neighbor with the cosine similarity (skip-gram trained on Wikipedia (1B tokens))

moon	score
mars	0.615
moons	0.611
lunar	0.602
sun	0.602
venus	0.583

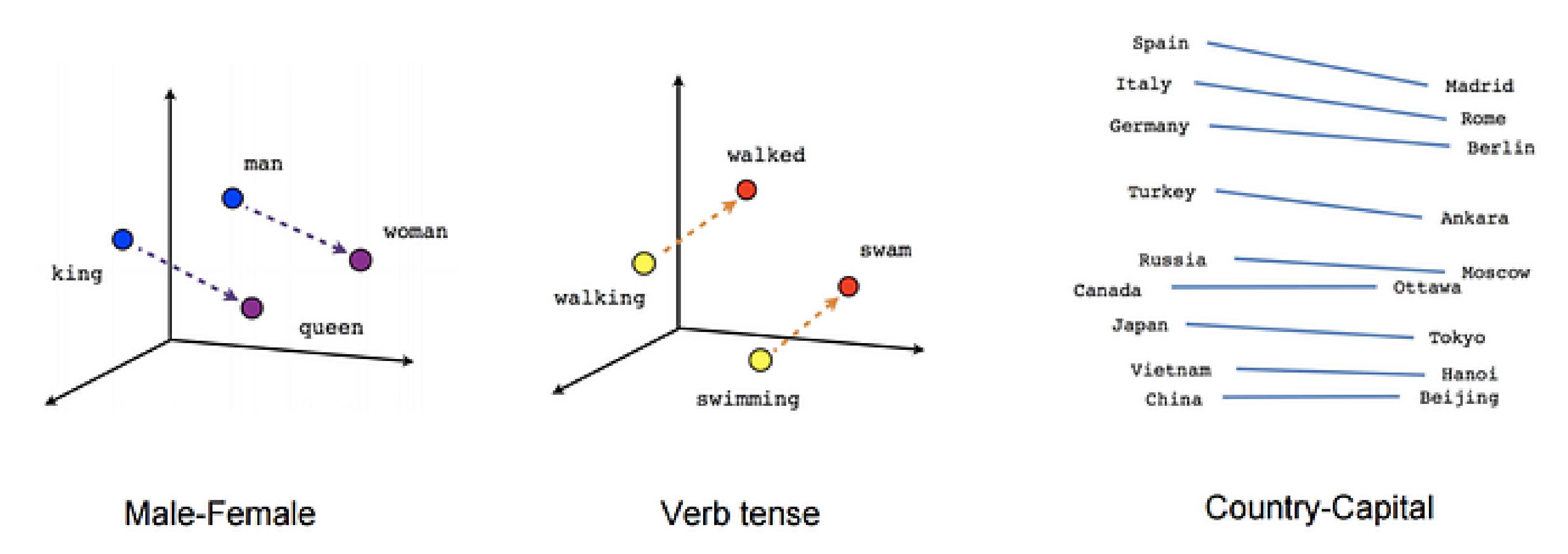
talking	score
discussing	0.663
telling	0.657
joking	0.632
thinking	0.627
talked	0.624

blue	score
red	0.704
yellow	0.677
purple	0.676
green	0.655
pink	0.612

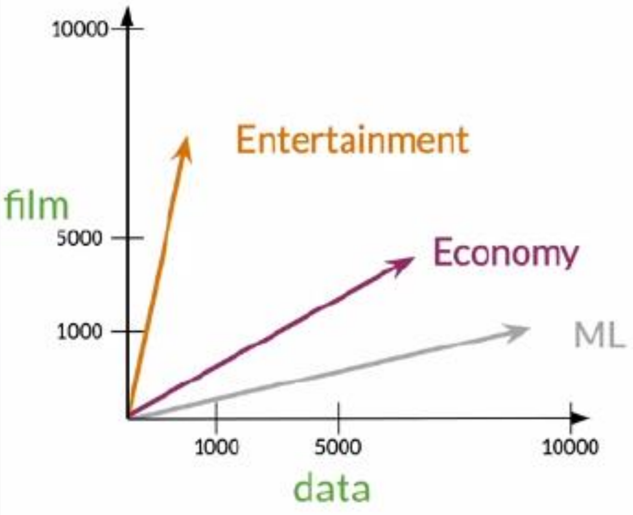
Vector comparison

- Vector comparison involves evaluating the similarity or difference between numerical representations (vectors) of text data
- Text is converted into vectors using methods like Bag of Words (BoW), TF-IDF, or embeddings.
- Measures used to compare vectors and evaluate similarity.
  - **Cosine Similarity:** measures the cosine of the angle between two vectors.
  - **Euclidean Distance:** measures the straight-line distance between two vectors in the vector space.
  - **Jaccard Similarity:** measures similarity based on the intersection over the union of feature sets.

Vector comparison



Vector comparison: vector space



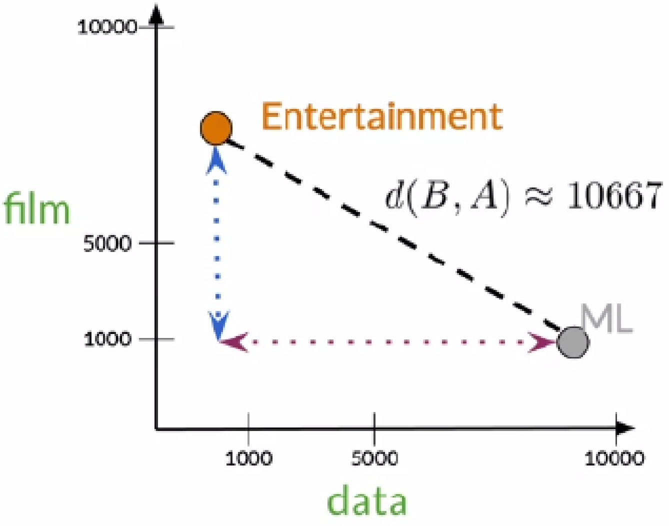
	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

Corpus

Measures of "similarity:"
Angle
Distance

Euclidean distance

Euclidean distance



Corpus A: (500,7000)

Corpus B: (9320,1000)

$$d(B,A) = \sqrt{\underline{(B_1 - A_1)}^2 + \underline{(B_2 - A_2)}^2}$$

$$c^2 = a^2 + b^2$$

$$d(B,A) = \sqrt{(-8820)^2 + (6000)^2}$$

Euclidean distance for n-dimensional vectors

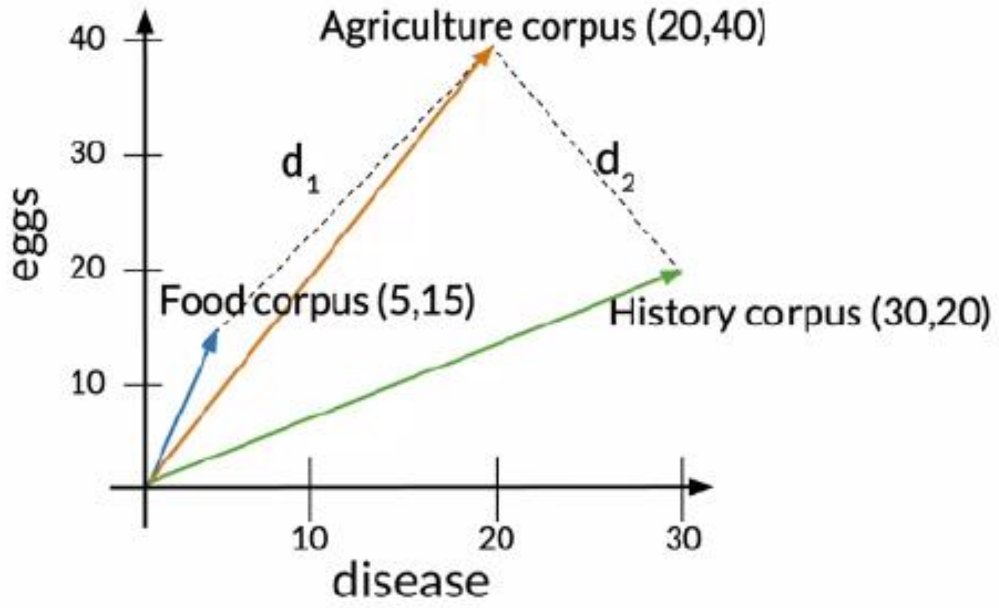
	data	boba	ice-cream
		$\vec{w}$	$\vec{v}$
AI	6	0	1
drinks	0	4	6
food	0	6	8

$$= \sqrt{(1-0)^2 + (6-4)^2 + (8-6)^2}$$

$$= \sqrt{1+4+4} = \sqrt{9} = 3$$

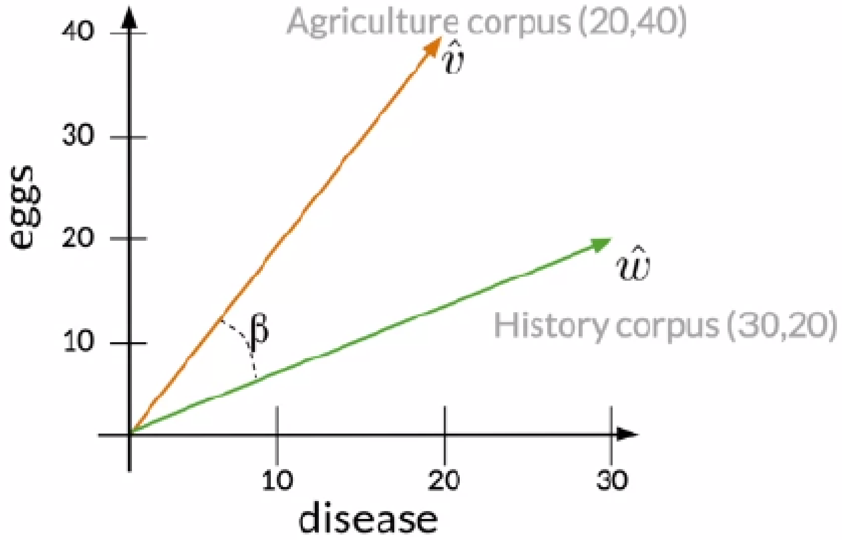
$$d(\vec{v}, \vec{w}) = \sqrt{\sum_{i=1}^{n} (v_i - w_i)^2} \longrightarrow \text{Norm of } (\vec{v} - \vec{w})$$

Problem with euclidean distance



Euclidean distance: $d_2 < d_1$

Cosine similarity



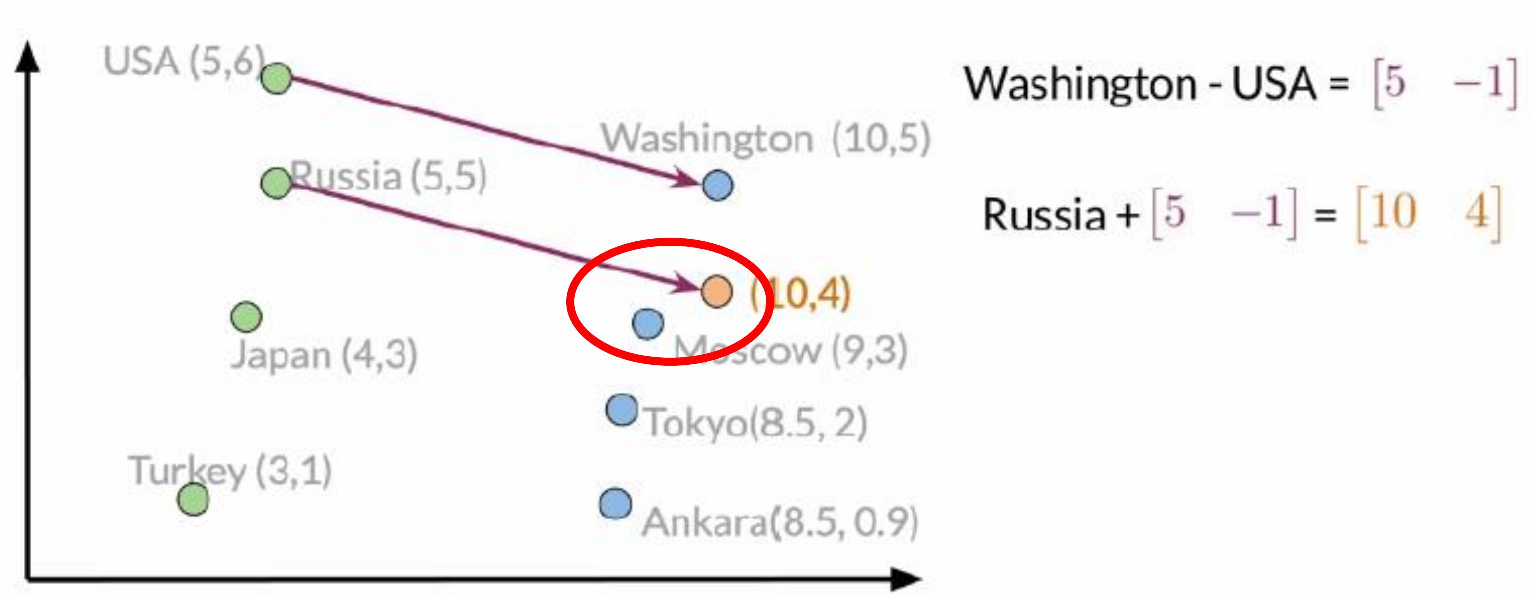
$$\hat{v} \cdot \hat{w} = \|\hat{v}\| \|\hat{w}\| \cos(\beta)$$

$$\boxed{\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}}$$

$$= \frac{(20 \times 30) + (40 \times 20)}{\sqrt{20^2 + 40^2} \times \sqrt{30^2 + 20^2}}$$

$$= 0.87$$

Manipulating word vectors



[Mikolov et al, 2013, Distributed Representations of Words and Phrases and their Compositionality]