Vector Semantics & Embeddings

Vector Semantics

Computational models of word meaning

Can we build a theory of how to represent word meaning, that accounts for at least some of the desiderata?

We'll introduce **vector semantics**

- The standard model in language processing!
- Handles many of our goals!

Ludwig Wittgenstein

PI #43:

"The meaning of a word is its use in the language"

Let's define words by their usages

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):

If A and B have almost identical environments we say that they are synonyms.

What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious **sautéed with garlic**.
- Ong choi is superb **over rice**
- Ong choi **leaves** with salty sauces

And you've also seen these:

- …spinach **sautéed with garlic over rice**
- Chard stems and **leaves** are **delicious**
- Collard greens and other **salty** leafy greens

Conclusion:

- Ongchoi is a leafy green like spinach, chard, or collard greens
  - We could conclude this based on words like "leaves" and "delicious" and "sauteed"

Ongchoi: *Ipomoea aquatica "Water Spinach"*

空心菜
kangkong
rau muống
…

Yamaguchi, Wikimedia Commons, public domain

Idea 1: Defining meaning by linguistic distribution

Let's define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.

Idea 2: Meaning as a point in space (Osgood et al. 1957)

3 affective dimensions for a word

- **valence**: pleasantness
- **arousal**: intensity of emotion
- **dominance**: the degree of control exerted

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

NRC VAD Lexicon
(Mohammad 2018)

Hence the connotation of a word is a vector in 3-space

Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space

Defining meaning as a point in space based on distribution

Each word = a vector (not just "good" or "w_{45}")

Similar words are "**nearby in semantic space**"

We build this space automatically by seeing which words are **nearby in text**

We define meaning of a word as a vector

Called an "embedding" because it's embedded into a space (see textbook)

The standard way to represent meaning in NLP

Every modern NLP algorithm uses embeddings as the representation of word meaning

Fine-grained model of meaning for similarity

Intuition: why vectors?

Consider sentiment analysis:

- With **words**, a feature is a word identity
  - Feature 5: 'The previous word was "terrible"'
  - requires **exact same word** to be in training and test
- With **embeddings**:
  - Feature is a word vector
  - 'The previous word was vector [35,22,17…]
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to **similar but unseen** words!!!

We'll discuss 2 kinds of embeddings

tf-idf

- Information Retrieval workhorse!
- A common baseline model
- **Sparse** vectors
- Words are represented by (a simple function of) the **counts** of nearby words

Word2vec

- **Dense** vectors
- Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**

From now on:
Computing with meaning representations instead of string representations

荃者所以在鱼，得鱼而忘荃　Nets are for fish;
Once you get the fish, you can forget the net.
言者所以在意，得意而忘言　Words are for meaning;
Once you get the meaning, you can forget the words

庄子(Zhuangzi), Chapter 26

Vector Semantics & Embeddings

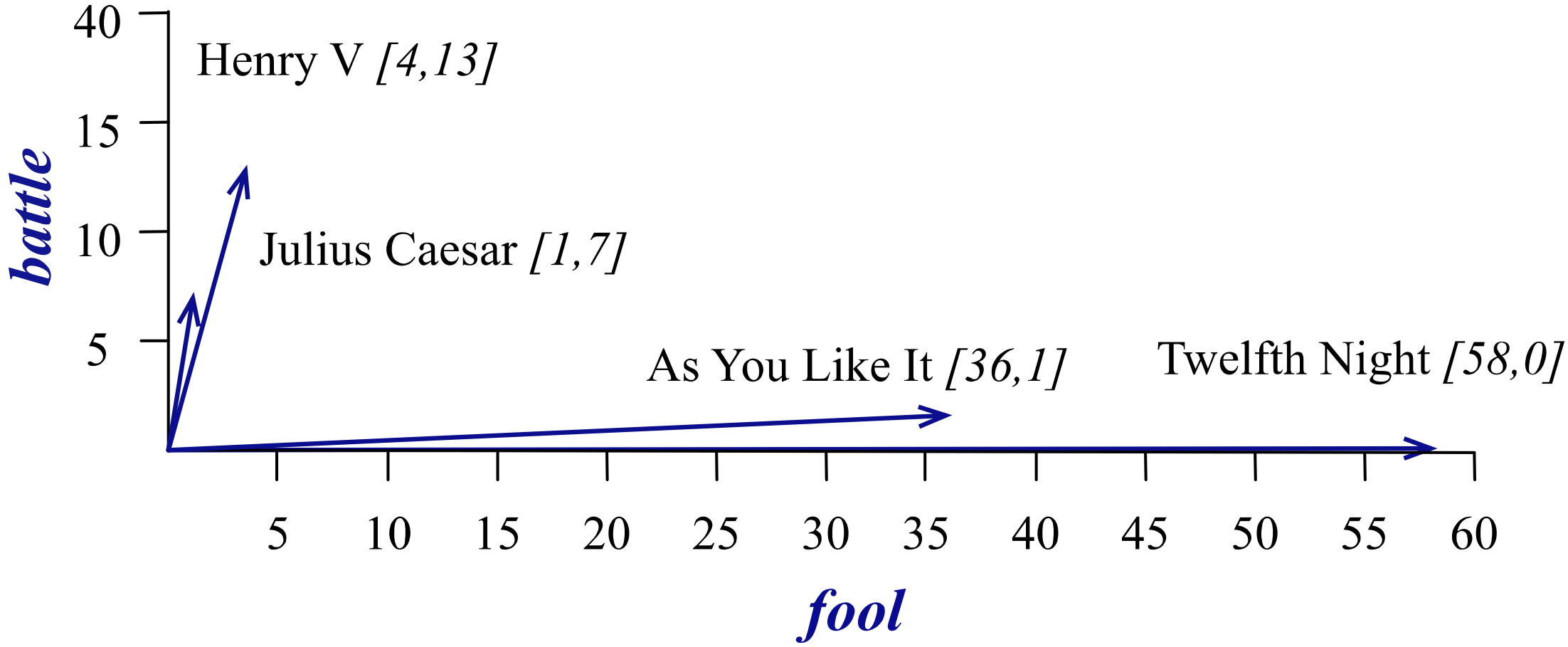
Words and Vectors

Term-document matrix

Each document is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Visualizing document vectors



Vectors are the basis of information retrieval

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Vectors are similar for the two comedies

But comedies are different than the other two

Comedies have more *fools* and *wit* and fewer *battles*.

Idea for word meaning: Words can be vectors too!!!

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

battle is "the kind of word that occurs in Julius Caesar and Henry V"

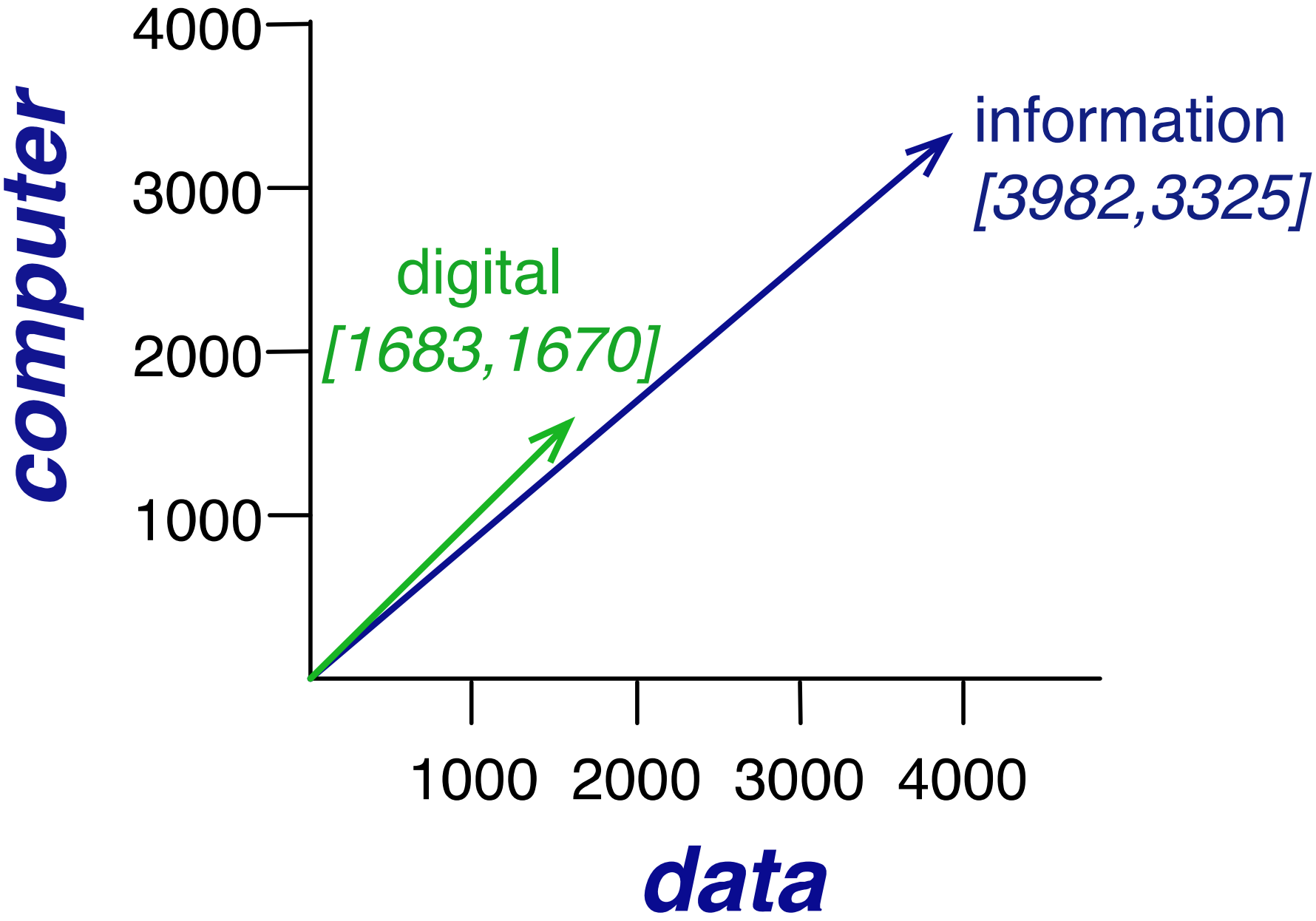
fool is "the kind of word that occurs in comedies, especially Twelfth Night"

More common: word-word matrix (or "term-context matrix")

Two **words** are similar in meaning if their context vectors are similar

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

	aardvark	...	**computer**	**data**	**result**	**pie**	**sugar**	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...


4000
3000
2000
1000
computer
information
[3982,3325]
digital
[1683,1670]
1000
2000
3000
4000
data

Vector Semantics & Embeddings

Cosine for computing word similarity

Computing word similarity: Dot product and cosine

The dot product between two vectors is a scalar:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

The dot product tends to be high when the two vectors have large values in the same dimensions

Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

Dot product favors long vectors

Dot product is higher if a vector is longer (has higher values in many dimension)

Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

Frequent words (of, the, you) have long vectors (since they occur many times with other words).

So dot product overly favors frequent words

Alternative: cosine for computing word similarity

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Based on the definition of the dot product between two vectors a and b

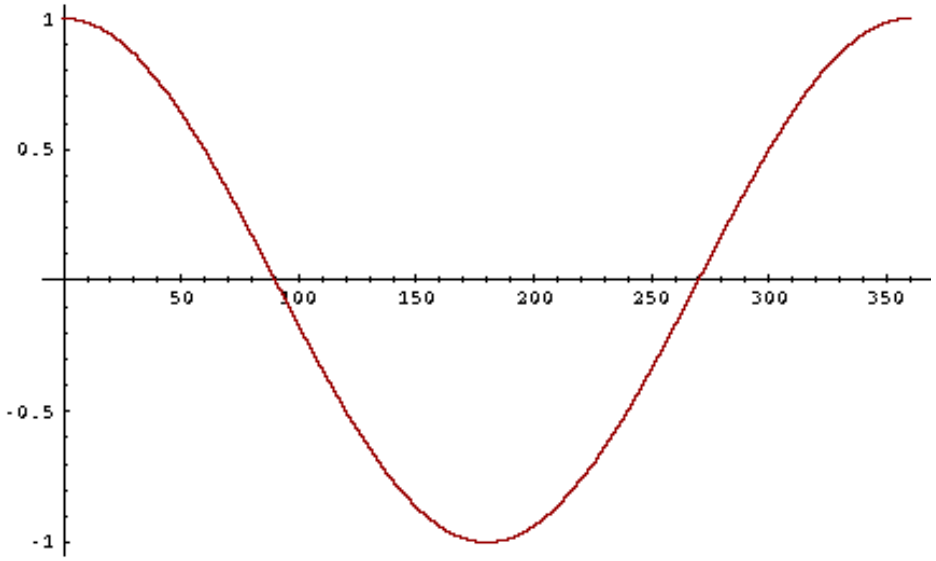
$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$
$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

0: vectors are orthogonal



But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

Cosine examples

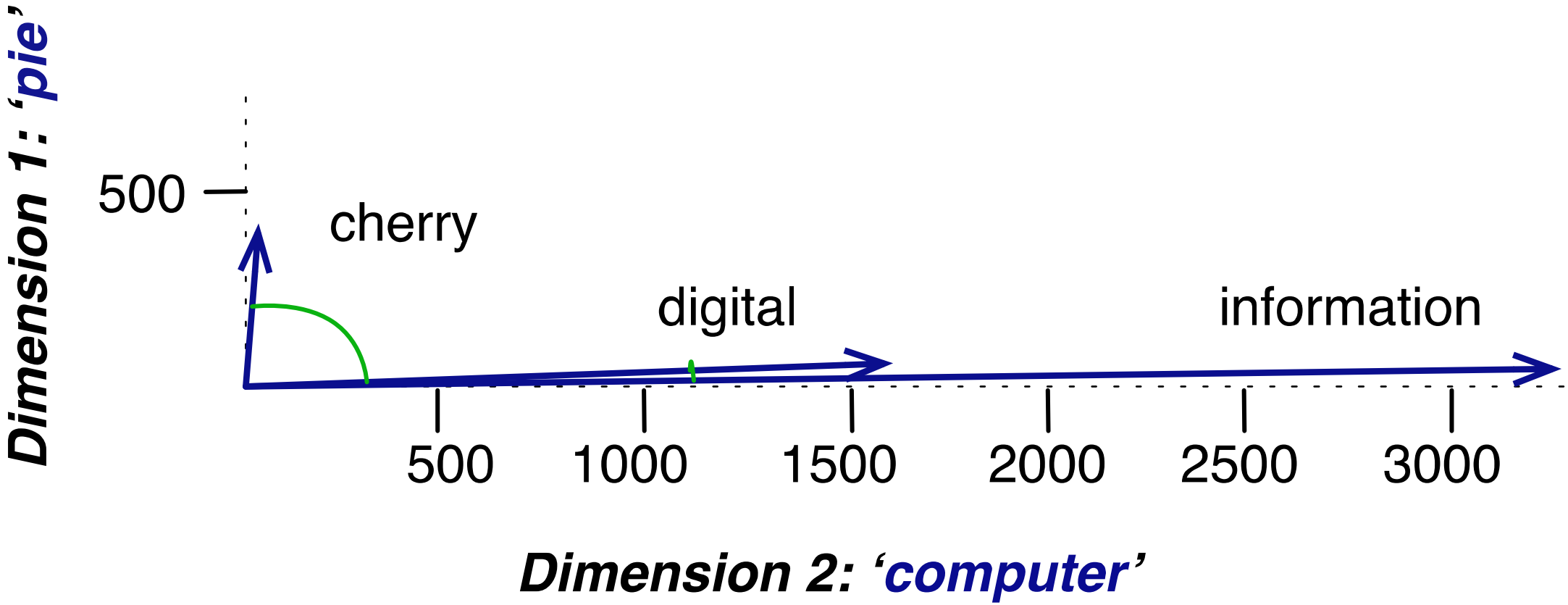
$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\cdot\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry},\text{information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .017$$

$$\cos(\text{digital},\text{information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

Visualizing cosines (well, angles)



Word Embedding

Word Embedding

- Word embedding is a technique in natural language processing (NLP) that represents words as vectors in a continuous vector space. This allows for capturing semantic relationships and similarities between words based on their context in large text corpora.
- Some word embedding models are Word2vec (Google), Glove (Stanford), and fastest (Facebook).

Word Embedding

- Word Embedding is also called a distributed semantic model or distributed represented or semantic vector space or vector space model.
- The similar words can be grouped together. For example, fruits like apples, mango, and banana should be placed close whereas books will be far away from these words.
- In a broader sense, word embedding will create the vector of fruits which will be placed far away from the vector representation of books.

Importance - Word Embedding

- **Semantic Understanding -** semantic meaning of words
- **Contextual Relationships -** Embeddings can capture the context in which words appear.
- **Dimensionality Reduction -** Word embeddings reduce the dimensionality, making computations more efficient and models faster.
- **Transfer Learning**: Pre-trained word embeddings can be used.
- **Improved Performance**: Using word embeddings often leads to better performance in various NLP tasks such as text classification, sentiment analysis, and machine translation. They help models generalize better by capturing nuanced relationships between words.

Applications

- **NLP Tasks**: Used in sentiment analysis, machine translation, information retrieval, and more.
- **Transfer Learning**: Pre-trained models can be fine-tuned for specific tasks, leveraging knowledge from vast corpora.
- **Word Embedding Limitations**
  - **Out-of-Vocabulary Words**: Word2Vec does not handle words that were not present in the training data well.
  - **Lack of Contextual Awareness**: It generates a single vector for each word regardless of its context (e.g., "bank" in "river bank" vs. "financial bank").

Word2Vec

- Word2Vec is a popular technique for creating word embeddings, developed by a team at Google led by Tomas Mikolov. It represents words in a continuous vector space, allowing machines to understand their meanings based on context. Here are the main components of Word2Vec:

Word2Vec - **Architectures**

- **Continuous Bag of Words (CBOW)**:
  - Predicts the target word from the surrounding context words.
  - For example, given the context words "the," "sat," "on," it predicts "cat."
  - Generally faster and more suitable for smaller datasets.
- **Skip-gram Model**:
  - Predicts surrounding context words given a target word.
  - For example, given the target word "cat," it tries to predict words like "sat," "on," "the," etc.
  - Effective for capturing semantic relationships and works well with large datasets.

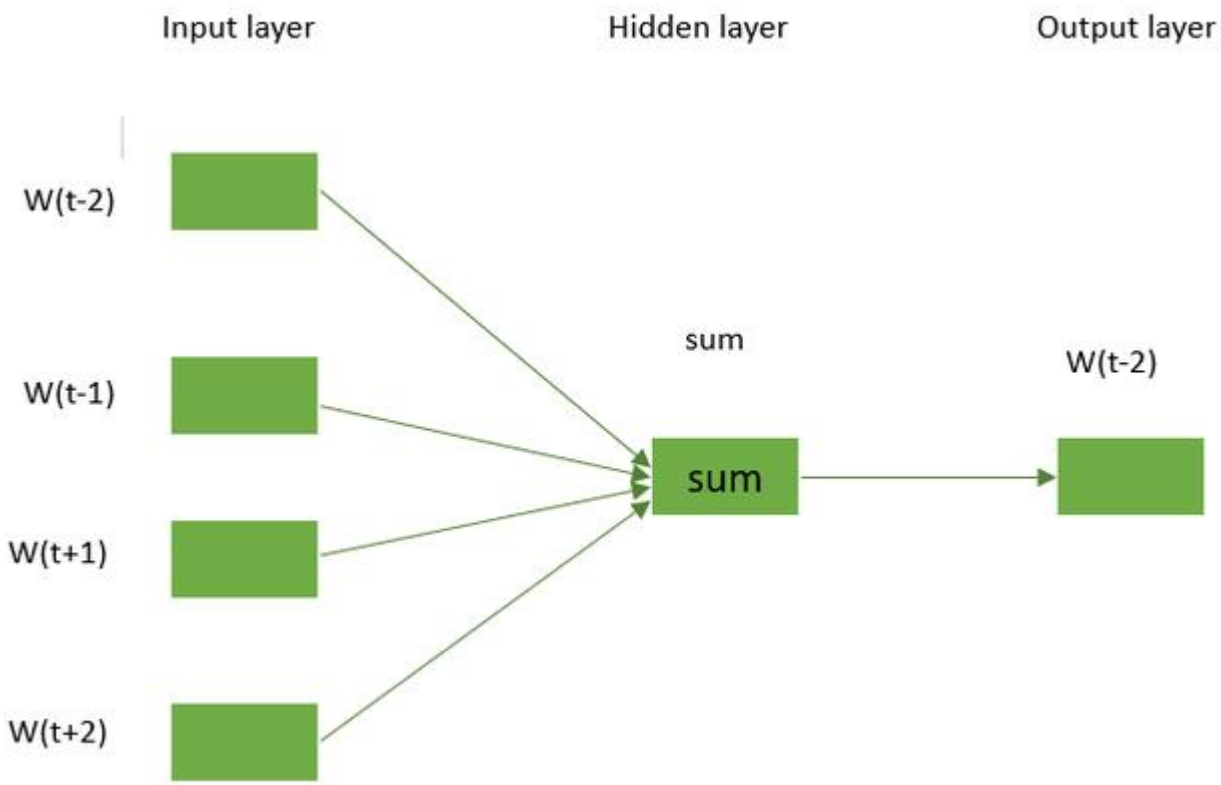
Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don’t actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
    - No need for human labels
    - Bengio et al. (2003); Collobert et al. (2011)

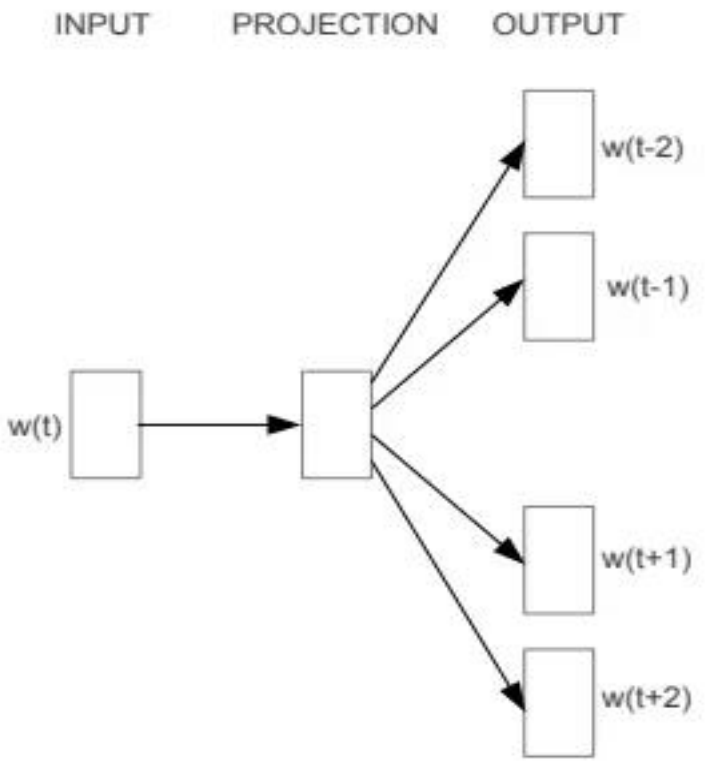
Approach: predict if candidate word *c* is a "neighbor"

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Architecture of the CBOW model



Architecture of Skip-gram Model

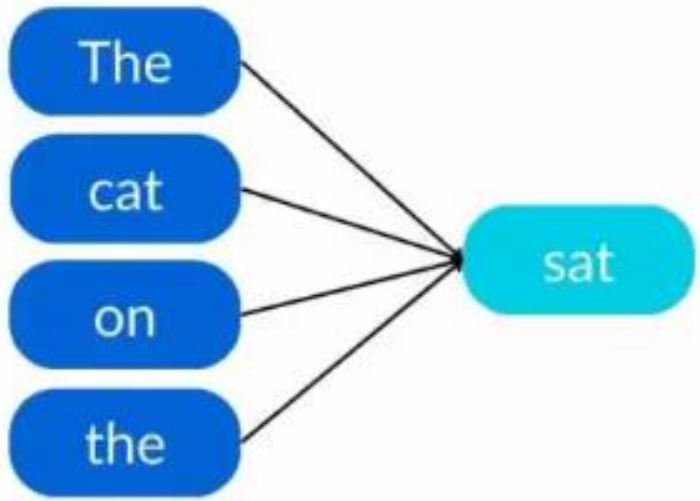


Skip-gram

Example Sentence: The cat sat on the mat.

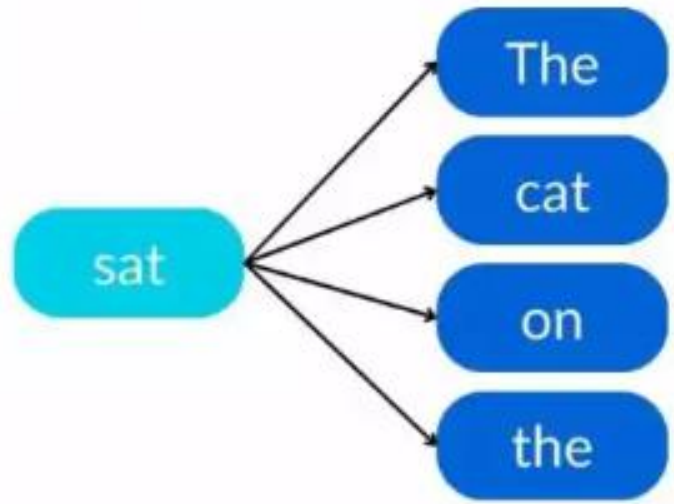
Continuous Bag-of-Words (CBOW)

Goal: Given context words,
predict the target word.



Skip-gram Model

Goal: Given a word,
predict the surrounding context words.



Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- So:
  - Similarity(w,c) $\propto w \cdot c$
- We'll need to normalize to get a probability

CBOW - Example

Step 1: Define the Example

Sentence: "The man and woman are discussing about the king and queen."

Step 2: Identify the Target and Context Words

- **Target Word**: **"king"**
- **Context Words**: **["man", "woman"]**

Step 3: Assign 4-Dimensional Vectors

Let's assign random 4-dimensional vectors to each of the relevant words:

- "king" → [0.6, 0.8, 0.5, 0.9]
- "queen" → [0.5, 0.7, 0.4, 0.8]
- "man" → [0.2, 0.3, 0.6, 0.1]
- "woman" → [0.3, 0.5, 0.7, 0.4]

CBOW – Example Cont…

Step 4: Average the Context Vectors

Now, calculate the average vector of the context words "man" and "woman."

1. **Vectors for Context Words**:
   - "man" → [0.2, 0.3, 0.6, 0.1]
   - "woman" → [0.3, 0.5, 0.7, 0.4]
2. **Calculate the Average**:

$$\text{Average} = \frac{1}{2}\left([0.2, 0.3, 0.6, 0.1] + [0.3, 0.5, 0.7, 0.4]\right)$$

$$= \frac{1}{2}\left([0.5, 0.8, 1.3, 0.5]\right) = [0.25, 0.4, 0.65, 0.25]$$

CBOW – Example Cont…

Step 5: Compute Scores

Now, we'll compute the scores for each word in the vocabulary by calculating the dot product between the average vector and the weight vectors for each word.

Example Weights

Assuming we have learned weight vectors for each word:

- Weights for "king" → [0.7, 0.8, 0.6, 0.9]
- Weights for "queen" → [0.6, 0.7, 0.5, 0.8]
- Weights for "man" → [0.4, 0.3, 0.5, 0.4]
- Weights for "woman" → [0.5, 0.6, 0.7, 0.3]

CBOW – Example Cont…

Calculate the Scores

1. Score for "king":

$$\text{Score}_{king} = (0.25 \times 0.7) + (0.4 \times 0.8) + (0.65 \times 0.6) + (0.25 \times 0.9)$$
$$= 0.175 + 0.32 + 0.39 + 0.225 = 1.110$$

2. Score for "queen":

$$\text{Score}_{queen} = (0.25 \times 0.6) + (0.4 \times 0.7) + (0.65 \times 0.5) + (0.25 \times 0.8)$$
$$= 0.15 + 0.28 + 0.325 + 0.2 = 0.955$$

3. Score for "man":

$$\text{Score}_{man} = (0.25 \times 0.4) + (0.4 \times 0.3) + (0.65 \times 0.5) + (0.25 \times 0.4)$$
$$= 0.1 + 0.12 + 0.325 + 0.1 = 0.645$$

4. Score for "woman":

$$\text{Score}_{woman} = (0.25 \times 0.5) + (0.4 \times 0.6) + (0.65 \times 0.7) + (0.25 \times 0.3)$$
$$= 0.125 + 0.24 + 0.455 + 0.075 = 0.895$$

CBOW – Example Cont…

Step 6: Apply Softmax Function

Next, convert the scores to probabilities using the softmax function.

Calculate Exponentials

1. $e^{1.110} \approx 3.033$
2. $e^{0.955} \approx 2.598$
3. $e^{0.645} \approx 1.905$
4. $e^{0.895} \approx 2.446$

Total:

$$\text{Total} = 3.033 + 2.598 + 1.905 + 2.446 \approx 10.982$$

CBOW – Example Cont…

Softmax Probabilities:

1. Probability for "king":

$$P(\text{king}) = \frac{3.033}{10.982} \approx 0.276$$

2. Probability for "queen":

$$P(\text{queen}) = \frac{2.598}{10.982} \approx 0.237$$

3. Probability for "man":

$$P(\text{man}) = \frac{1.905}{10.982} \approx 0.174$$

4. Probability for "woman":

$$P(\text{woman}) = \frac{2.446}{10.982} \approx 0.223$$

Softmax Calculation:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum e^{x_j}}$$

Why Softmax? - Probabilities can be compared directly, helping to identify which word has the highest likelihood of being the target.

CBOW – Example Cont…

- **Step 7: Identify the Predicted Word**
- The predicted word is the one with the highest probability. In this case, if "king" has the highest probability (approximately 0.276), it will be the predicted target word.
- **Summary of Steps**
  1. **Identify the Target**: Target word "king" and context words "man" and "woman."
  2. **Vector Representation**: Assign 4D vectors to all words.
  3. **Average Context**: Calculate the average of context word vectors.
  4. **Score Calculation**: Compute dot products with weights for each word.
  5. **Softmax**: Convert scores to probabilities.
  6. **Prediction**: Identify the word with the highest probability.

Summary

- In CBOW, context words are chosen based on proximity to the target word, which is determined by the window size.
- Including additional words like "queen" depends on expanding the context window.
- By adjusting the context, you can influence the prediction, capturing more semantic relationships.

Skip-gram model - Ex

Step 1: Define the Example

Sentence: "The man and woman are discussing about the king and queen."

Step 2: Identify the Target and Context Words

- **Target Word**: "king"
- **Context Words**: ["man," "woman," "queen"] (depending on the context window size)

Skip-gram model – Ex Cont...

Step 3: Assign 4-Dimensional Vectors

Assign random 4-dimensional vectors to each of the relevant words:

- "king" → [0.6, 0.8, 0.5, 0.9]
- "queen" → [0.5, 0.7, 0.4, 0.8]
- "man" → [0.2, 0.3, 0.6, 0.1]
- "woman" → [0.3, 0.5, 0.7, 0.4]

Skip-gram model – Ex Cont...

Step 4: Prepare Input for Skip-gram

In Skip-gram, the target word is used to predict the context words. Here, **"king"** will be used to predict its surrounding context words **["man," "woman," "queen"]**.

Step 5: Compute Scores for Context Words

1. **Input Vector**: The vector for the target word "king" is $[0.6, 0.8, 0.5, 0.9]$.
2. **Calculate Scores**: Compute dot products with the weight vectors for the context words.

Skip-gram model – Ex Cont...

Example Weights for Context Words

Assuming the same weight vectors for the context words:

- Weights for "man" → [0.2, 0.3, 0.6, 0.1]
- Weights for "woman" → [0.3, 0.5, 0.7, 0.4]
- Weights for "queen" → [0.5, 0.7, 0.4, 0.8]

Skip-gram model – Ex Cont...

Calculate the Scores

1. Score for "man":

$$\text{Score}_{man} = (0.6 \times 0.2) + (0.8 \times 0.3) + (0.5 \times 0.6) + (0.9 \times 0.1)$$

$$= 0.12 + 0.24 + 0.30 + 0.09 = 0.75$$

2. Score for "woman":

$$\text{Score}_{woman} = (0.6 \times 0.3) + (0.8 \times 0.5) + (0.5 \times 0.7) + (0.9 \times 0.4)$$

$$= 0.18 + 0.40 + 0.35 + 0.36 = 1.29$$

3. Score for "queen":

$$\text{Score}_{queen} = (0.6 \times 0.5) + (0.8 \times 0.7) + (0.5 \times 0.4) + (0.9 \times 0.8)$$

$$= 0.30 + 0.56 + 0.20 + 0.72 = 1.78$$

Skip-gram model – Ex Cont...

Step 6: Apply Softmax Function

Next, convert the scores to probabilities using the softmax function.

Calculate Exponentials

1. $e^{0.75} \approx 2.117$
2. $e^{1.29} \approx 3.615$
3. $e^{1.78} \approx 5.892$

Total:

$$\text{Total} = 2.117 + 3.615 + 5.892 \approx 11.624$$

Skip-gram model – Ex Cont...

Softmax Probabilities:

1. Probability for "man":

$$P(\text{man}) = \frac{2.117}{11.624} \approx 0.182$$

2. Probability for "woman":

$$P(\text{woman}) = \frac{3.615}{11.624} \approx 0.311$$

3. Probability for "queen":

$$P(\text{queen}) = \frac{5.892}{11.624} \approx 0.506$$

Skip-gram model – Ex Cont...

Step 7: Identify the Predicted Context Words

The context words predicted from the target word "king" will be based on the highest probabilities. In this case, "queen" would have the highest probability, followed by "woman" and "man."

Summary of Steps

1. **Identify the Target:** Target word "king" and context words ["man," "woman," "queen"].
2. **Vector Representation:** Assign 4D vectors to all words.
3. **Score Calculation:** Compute dot products between the target word vector and context word vectors.
4. **Softmax:** Convert scores to probabilities.
5. **Prediction:** Identify context words based on the highest probabilities.

Identify the Predicted Context Words:

Threshold: You can set a threshold to consider only those context words whose probabilities exceed a certain value.
Top-N Selection: Alternatively, you can select the top-N context words with the highest probabilities. This is common when you expect multiple context words.

Step 1: Set a Threshold (Optional)

You can define a probability threshold, for example, 0.2. In this case, all three context words exceed the threshold.

Step 2: Select Top-N Context Words

If you want to select the top 2 context words based on probabilities, you would:

1. **Sort the Probabilities:**
   - "queen" → 0.506
   - "woman" → 0.311
   - "man" → 0.182
2. **Choose Top-N:**
   - Top 2 context words: **"queen"** and **"woman"**.

fasText

- fastText introduces a pivotal shift by considering words as composed of character n-grams, enabling it to build representations for words based on these subword units
- This approach allows the model to understand and generate embeddings for words not seen in the training data, offering a substantial advantage in handling morphologically rich languages and rare words.

Difference Between fastText and Word2Vec

- **Handling of Out-of-Vocabulary (OOV) Words**
  - **Word2Vec:** Word2Vec operates at the word level, generating embeddings for individual words. It struggles with out-of-vocabulary words as it cannot represent words it hasn’t seen during training.
  - **fastText:** In contrast, fastText introduces subword embeddings by considering words to be composed of character n-grams. This enables it to handle out-of-vocabulary words effectively by breaking terms into subword units and generating embeddings for these units, even for unseen words. This capability makes fastText more robust in dealing with rare or morphologically complex expressions.

Representation of Words

- **Word2Vec:** Word2Vec generates word embeddings based solely on the words without considering internal structure or morphological information.
- **fastText:** fastText captures subword information, allowing it to understand word meanings based on their constituent character n-grams. This enables fastText to represent words by considering their morphological makeup, providing a richer representation, especially for morphologically rich languages or domains with specialised jargon.

Training Efficiency

- **Word2Vec:** The training process in Word2Vec is relatively faster than older methods but might be slower than fastText due to its word-level approach.
- **fastText:** fastText is known for its exceptional speed and scalability, especially when dealing with large datasets, as it operates efficiently at the subword level.

Use Cases

- **Word2Vec:** Word2Vec's word-level embeddings are well-suited for tasks like finding similar words, understanding relationships between words, and capturing semantic similarities.
- **fastText:** fastText's subword embeddings make it more adaptable in scenarios involving out-of-vocabulary words, sentiment analysis, language identification, and tasks requiring a deeper understanding of morphology.