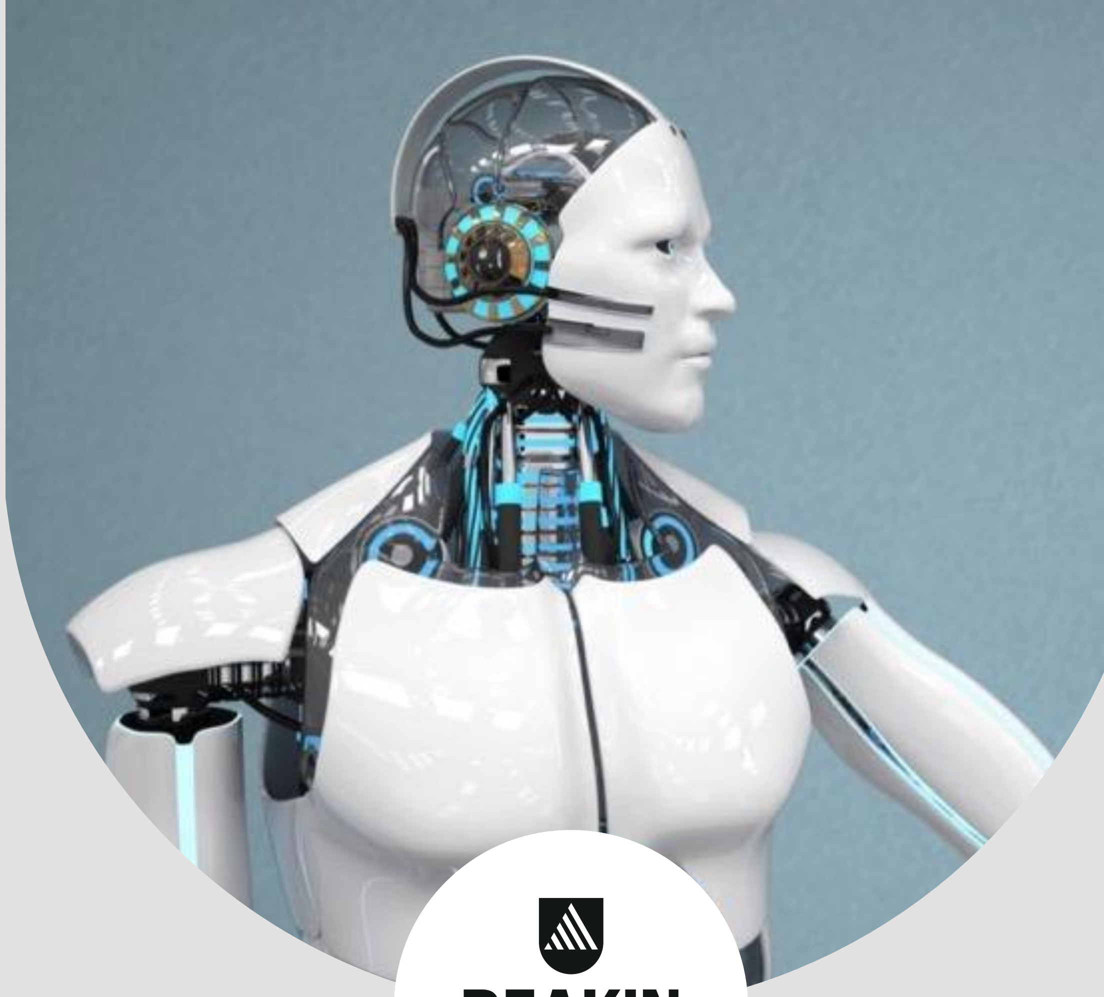
# SIT330-770: Natural Language Processing

## Week 5 - Vector Embeddings and Sequence Labeling

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology, Faculty of Sci Eng & Built Env

reda.bouadjenek@deakin.edu.au

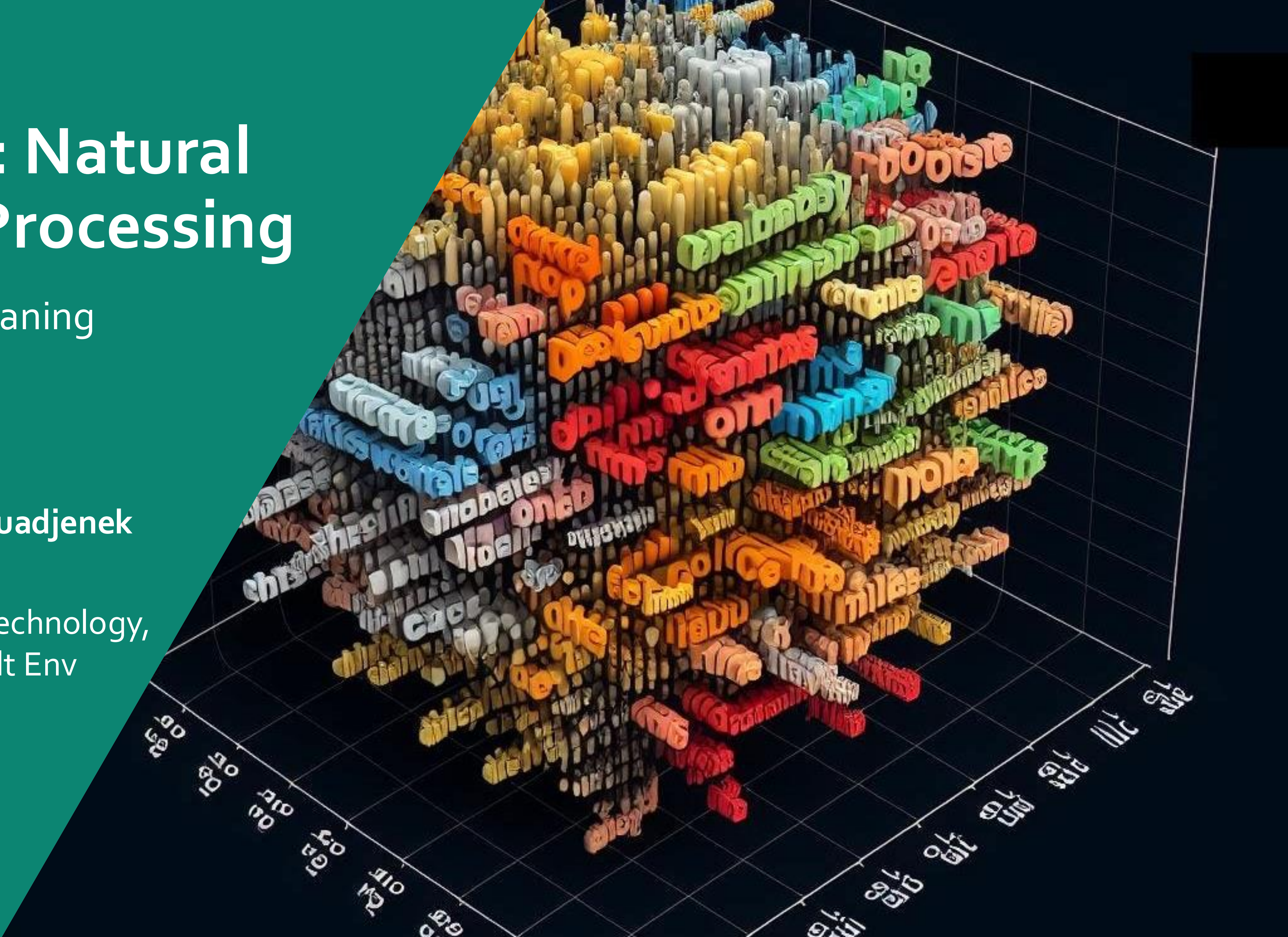


# SIT330-770: Natural Language Processing

Week 5.1 - Word Meaning

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env



## What do words mean?

- N-gram or text classification methods we've seen so far
  - Words are just strings (or indices $w_i$ in a vocabulary list)
  - That's not very satisfactory!
- Introductory logic classes:
  - The meaning of "dog" is DOG; cat is CAT

    $\forall x\ DOG(x) \longrightarrow MAMMAL(x)$
- Old linguistics joke by Barbara Partee in 1967:
  - Q: What's the meaning of life?
  - A: LIFE
- That seems hardly better!

## Desiderata

- What should a theory of word meaning do for us?
- Let's look at some desiderata
- From lexical semantics, the linguistic study of word meaning

## Lemmas and senses

lemma

mouse (N)

sense

1. any of numerous small rodents...
2. a hand-operated device that controls a cursor...

Modified from the online thesaurus WordNet

A sense or “concept” is the meaning component of a word

Lemmas can be polysemous (have multiple senses)

## Relations between senses: Synonymy

- Synonyms have the same meaning in some or all contexts.
  - filbert / hazelnut
  - couch / sofa
  - big / large
  - automobile / car
  - vomit / throw up
  - water / $H_2o$

## Relations between senses: Synonymy

- Note that there are probably no examples of perfect synonymy.
    - Even if many aspects of meaning are identical
    - Still may differ based on politeness, slang, register, genre, etc.

## Relation: Synonymy?

water/$H_2o$

"$H_2o$" in a surfing guide?

big/large

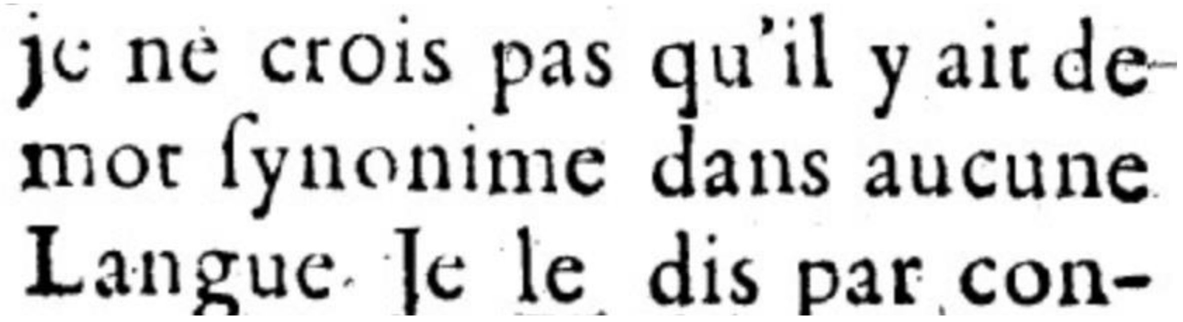
my big sister != my large sister

## The Linguistic Principle of Contrast

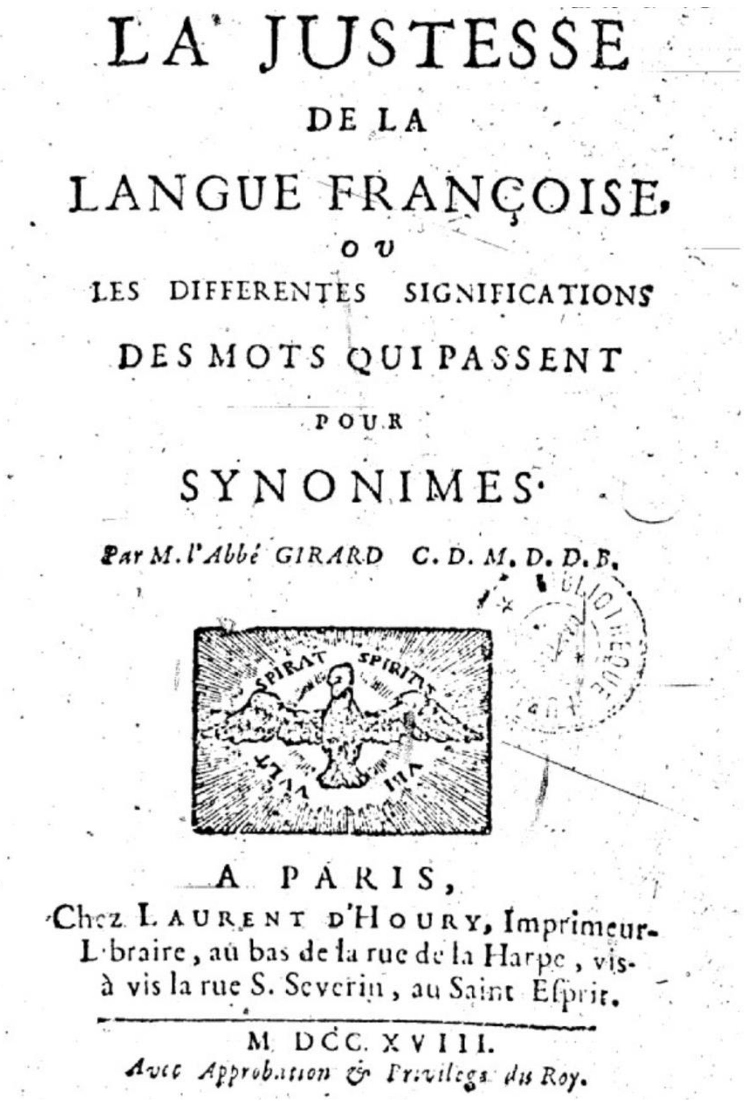
- Difference in form → difference in meaning

# Abbé Gabriel Girard 1718

Re: "exact" synonyms


je ne crois pas qu'il y ait de-
mot ſynonime dans aucune
Langue. Je le dis par con-

[I do not believe that there is a synonymous word in any language]


LA JUSTESSE
DE LA
LANGUE FRANÇOISE,
OU
LES DIFFERENTES SIGNIFICATIONS
DES MOTS QUI PASSENT
POUR
SYNONIMES.
Par M. l'Abbé GIRARD C. D. M. D. D. B.

A PARIS,
Chez LAURENT D'HOURY, Imprimeur-Libraire, au bas de la rue de la Harpe, vis-à vis la rue S. Severin, au Saint Eſprit.
M. DCC. XVIII.
Avec Approbation & Privilege du Roy.

Thanks to Mark Aronoff!

## Relation: Similarity

Words with similar meanings.  Not synonyms, but sharing some element of meaning

```
car, bicycle

cow, horse
```

## Ask humans how similar 2 words are

| word1 | word2 | similarity |
|---|---|---|
| vanish | disappear | 9.8 |
| behave | obey | 7.3 |
| belief | impression | 5.95 |
| muscle | bone | 3.65 |
| modest | flexible | 0.98 |
| hole | agreement | 0.3 |

SimLex-999 dataset (Hill et al., 2015)

## Relation: Word relatedness

- Also called "word association"
- Words can be related in any way, perhaps via a semantic frame or field
  - `coffee, tea:` **similar**
  - `coffee, cup:` **related**, not similar

## Semantic field

- Words that
  - cover a particular semantic domain
  - bear structured relations with each other.

**hospitals**

*surgeon*, *scalpel*, *nurse*, *anaesthetic*, *hospital*

**restaurants**

*waiter*, *menu*, *plate*, *food*, *menu*, *chef*

**houses**

*door*, *roof*, *kitchen*, *family*, *bed*

## Relation: Antonymy

- Senses that are opposites with respect to only one feature of meaning
- Otherwise, they are very similar!

`dark/light` `short/long` `fast/slow` `rise/fall`

`hot/cold` `up/down` `in/out`

- More formally: antonyms can
  - define a binary opposition or be at opposite ends of a scale
    - `long/short, fast/slow`
  - Be *reversives*:
    - `rise/fall, up/down`

## Connotation (sentiment)

- Words have **affective** meanings
  - Positive connotations (*happy*)
  - Negative connotations (*sad*)
- Connotations can be subtle:
  - Positive connotation: *copy, replica, reproduction*
  - Negative connotation: *fake, knockoff, forgery*
- Evaluation (sentiment!)
  - Positive evaluation (*great*, *love*)
  - Negative evaluation (*terrible*, *hate*)

## Connotation

Osgood et al. (1957)

- Words seem to vary along 3 affective dimensions:
  - **valence**: the pleasantness of the stimulus
  - **arousal**: the intensity of emotion provoked by the stimulus
  - **dominance**: the degree of control exerted by the stimulus

| | 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 |

Values from NRC VAD Lexicon (Mohammad 2018)

## So far

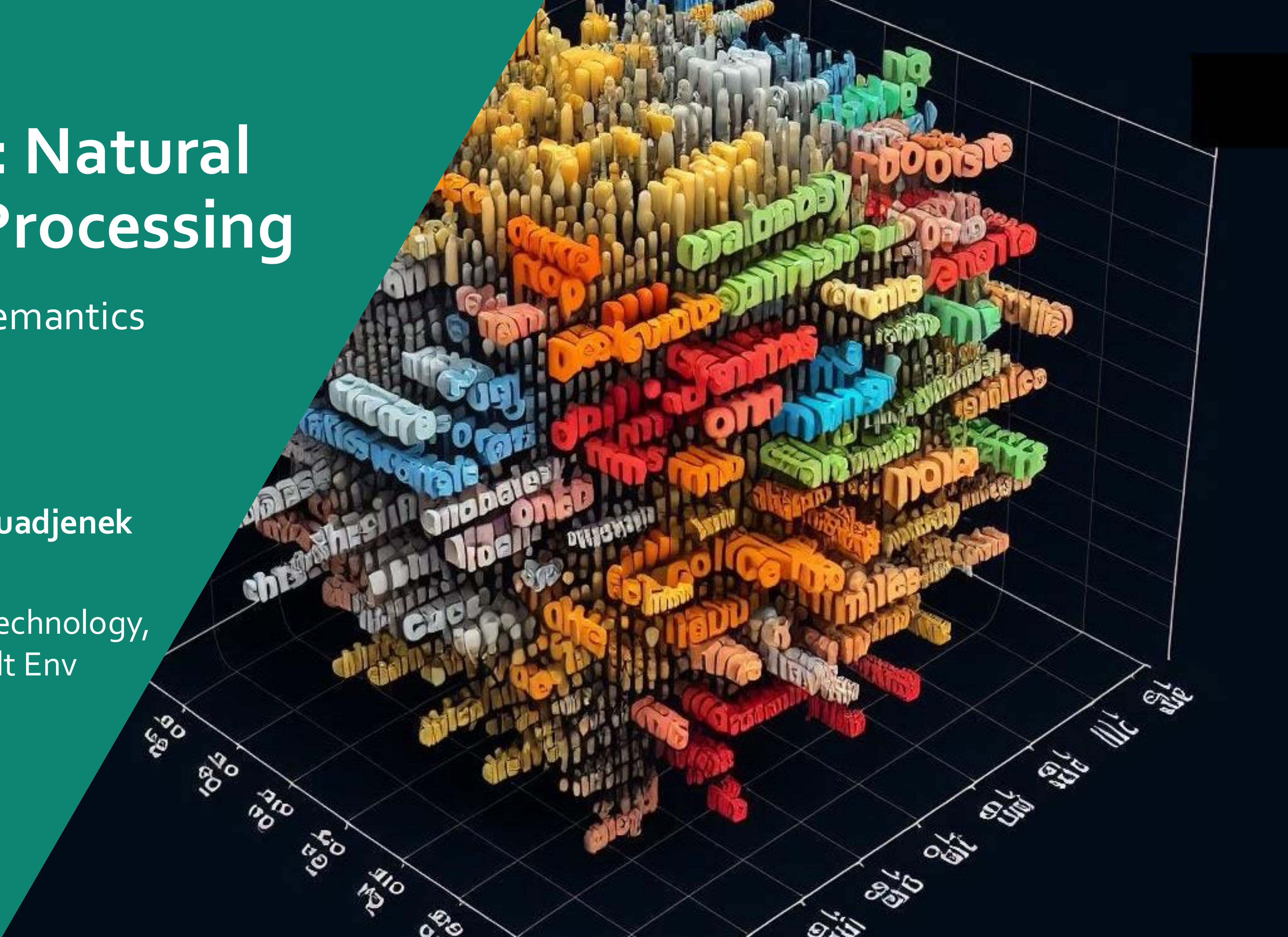
- **Concepts** or word senses
  - Have a complex many-to-many association with **words** (homonymy, multiple senses)
- Have relations with each other
  - Synonymy
  - Antonymy
  - Similarity
  - Relatedness
  - Connotation

# SIT330-770: Natural Language Processing

Week 5.2 - Vector Semantics

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env



## 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"

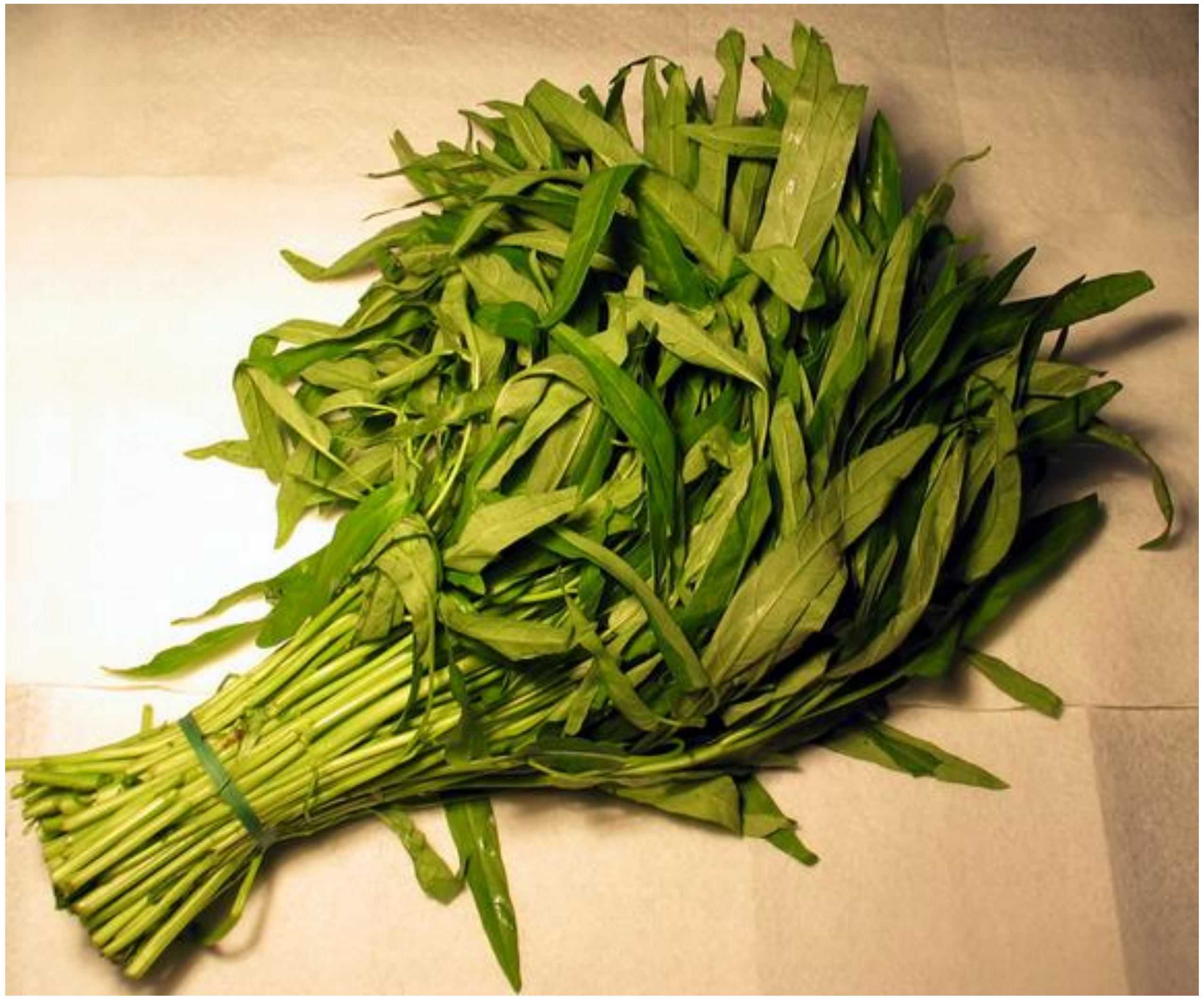
## Ongchoi: *Ipomoea aquatica "Water Spinach"*

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)

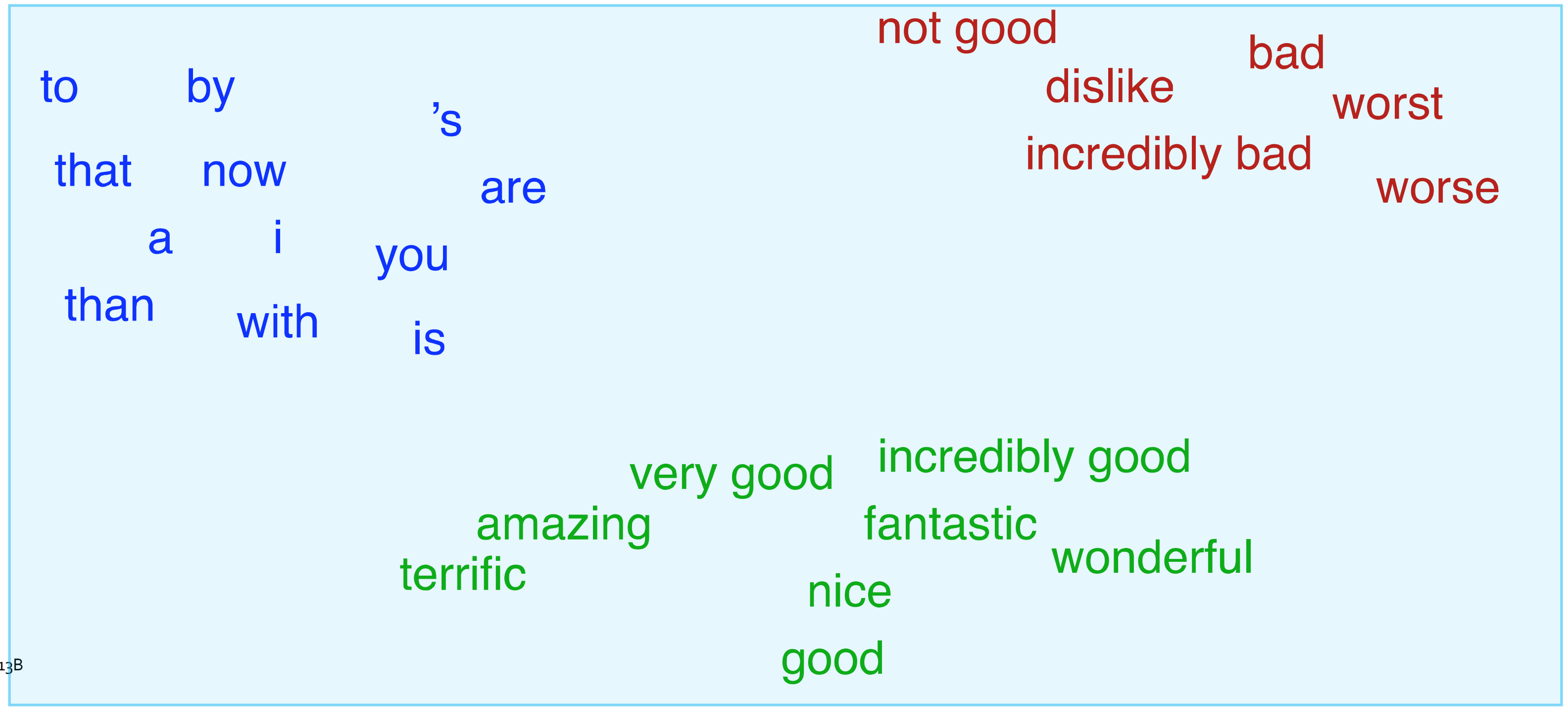
- Hence the connotation of a word is a vector in 3-space

## 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;<br>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;<br>Once you get the meaning, you can forget the words |

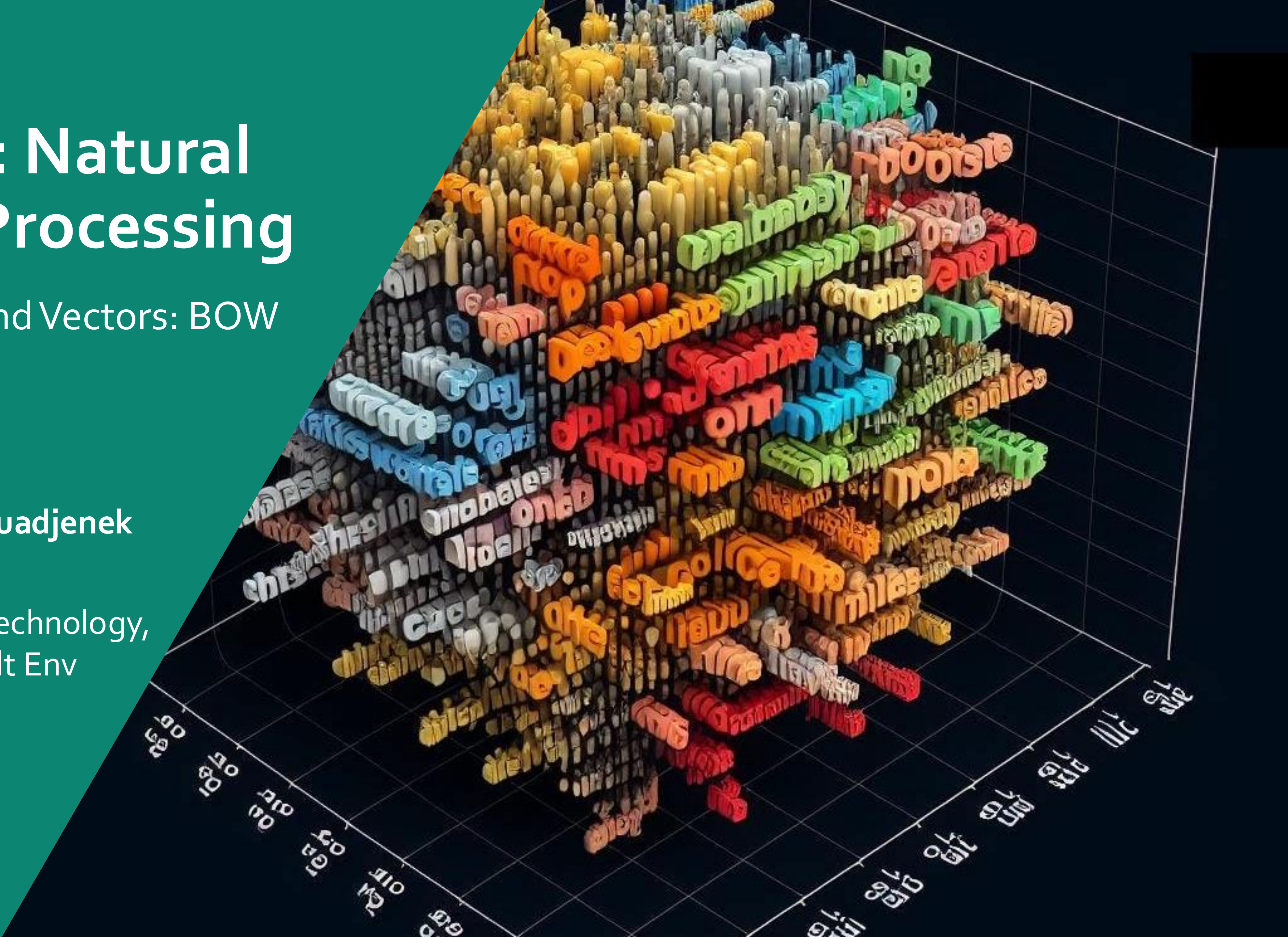
庄子(Zhuangzi), Chapter 26

# SIT330-770: Natural Language Processing

Week 5.3 – Words and Vectors: BOW

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env



## Bag of Words

- A document is represented as vector of words.
  - One dimension per word.
  - Vector size is the vocabulary size, e.g., English may contain 100k words.
  - Different weighting schemas can be used, e.g., tf, log(tf), tf-idf, Boolean, etc.
  - Sparse vector, e.g., almost all values are zeros.

| | 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 |

## Order matters for NLP tasks!

- Assumes independence between words:
  - The sentences "**John likes Mary**" has the same representation as "**Mary likes John**" – even though the semantic is different).
- May work well for Information Retrieval tasks, but not for NLP tasks!
  - Sentiment analysis:

    "Ah **no**, there are good movies on Netflix!" vs. "Ah, there are **no** good movies on Netflix!"

## 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

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

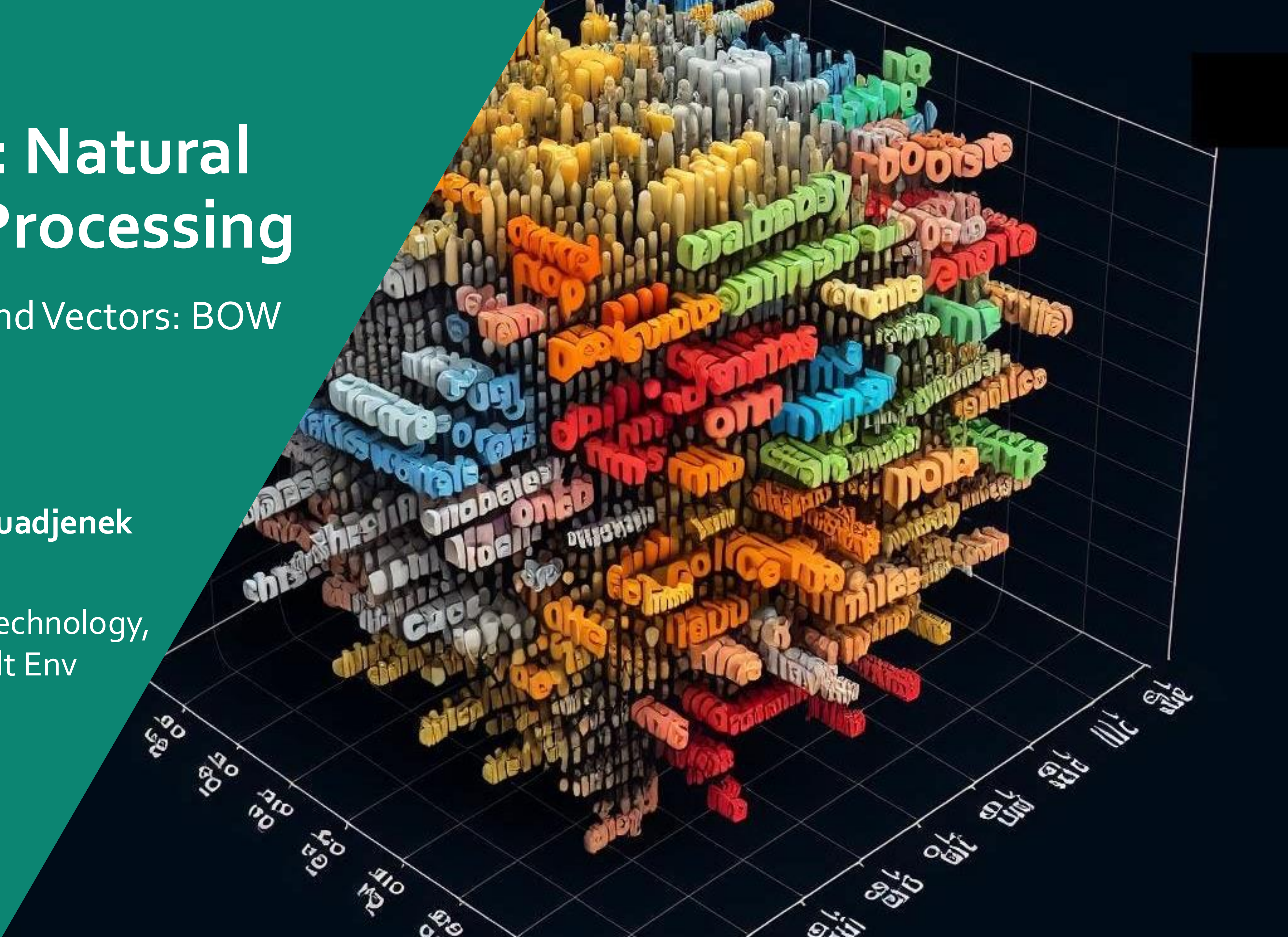
$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

# SIT330-770: Natural Language Processing

Week 5.4 – Words and Vectors: BOW

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env



## Bag of Words

- A document is represented as vector of words.
  - One dimension per word.
  - Vector size is the vocabulary size, e.g., English may contain 100k words.
  - Different weighting schemas can be used, e.g., tf, log(tf), tf-idf, Boolean, etc.
  - Sparse vector, e.g., almost all values are zeros.

| | 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 |

## Order matters for NLP tasks!

- Assumes independence between words:
  - The sentences "**John likes Mary**" has the same representation as "**Mary likes John**" – even though the semantic is different).
- May work well for Information Retrieval tasks, but not for NLP tasks!
  - Sentiment analysis:

    "Ah **no**, there are good movies on Netflix!" vs. "Ah, there are **no** good movies on Netflix!"

## 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

$$\begin{aligned} \mathbf{a}\cdot\mathbf{b} &= |\mathbf{a}||\mathbf{b}|\cos\theta \\ \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} &= \cos\theta \end{aligned}$$

## Sparse versus dense vectors

- tf-idf (or PMI) vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)
- Alternative: learn vectors which are
  - **short** (length 50-1000)
  - **dense** (most elements are non-zero)

## Sparse versus dense vectors

- Why dense vectors?
  - Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
  - Dense vectors may **generalize** better than explicit counts
  - Dense vectors may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are distinct dimensions
      - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
  - **In practice, they work better**

## Common methods for getting short dense vectors

- "Neural Language Model"-inspired models
  - Word2vec (skipgram, CBOW), GloVe
- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- Alternative to these "static embeddings":
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

## Simple static embeddings you can download!

- Word2vec (Mikolov et al)
- https://code.google.com/archive/p/word2vec/

- GloVe (Pennington, Socher, Manning)
- http://nlp.stanford.edu/projects/glove/

## Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** rather than **count**
- Word2vec provides various options. We'll do:
- **skip-gram with negative sampling (SGNS)**

## 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

# Skip-Gram Training Data

- (assuming a +/- 2 word window)

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

- Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

  (apricot, jam)

  (apricot, aardvark)

  …

- And assigns each pair a probability:
  - $P(+|w, c)$
  - $P(-|w, c) = 1 - P(+|w, c)$

## 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
  - (cosine isn't a probability either)

## Turning dot products into probabilities

- Sim(w,c) ≈ w $\cdot$ $c$
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)} \end{aligned}$$

## How Skip-Gram Classifier computes ***P*(+|*w*, *c*)**

$$P(+|w,c) \;=\; \sigma(c\cdot w) = \frac{1}{1+\exp(-c\cdot w)}$$

- This is for one context word, but we have lots of context words.
- We'll assume independence and just multiply them:

$$P(+|w,c_{1:L}) \;=\; \prod_{i=1}^{L}\sigma(c_i\cdot w)$$

$$\log P(+|w,c_{1:L}) \;=\; \sum_{i=1}^{L}\log\sigma(c_i\cdot w)$$

## Skip-gram classifier: summary

- A probabilistic classifier, given
  - a test target word *w*
  - its context window of *L* words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

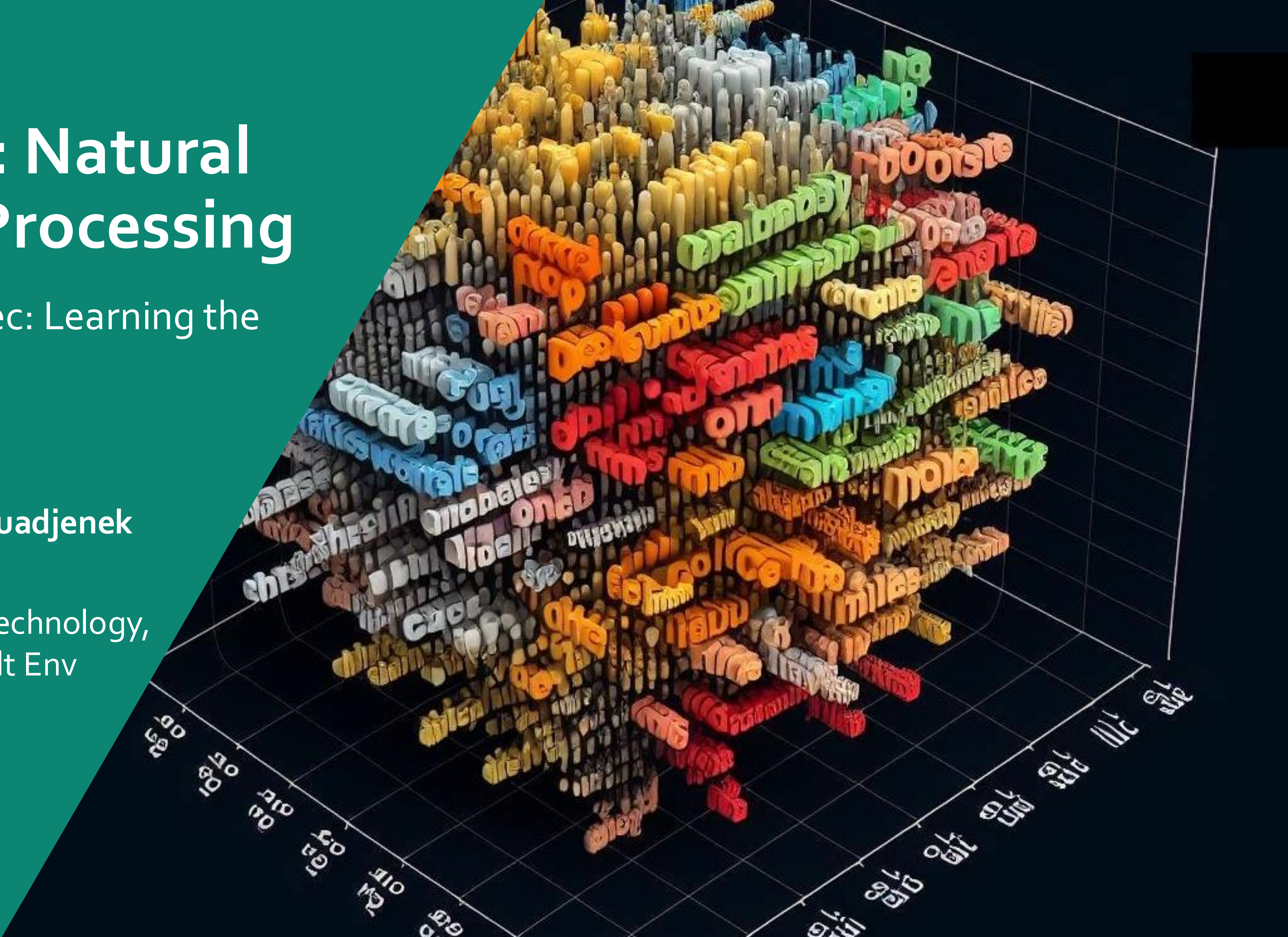
- To compute this, we just need embeddings for all the words.

# SIT330-770: Natural Language Processing

## Week 5.5 – Word2vec: Learning the embeddings

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology, Faculty of Sci Eng & Built Env



## Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

## Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

For each positive example we'll grab k negative examples, sampling by frequency

## Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

**negative examples -**

| t | c | t | c |
|---|---|---|---|
| apricot | aardvark | apricot | seven |
| apricot | my | apricot | forever |
| apricot | where | apricot | dear |
| apricot | coaxial | apricot | if |

## Word2vec: how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
  - **Maximize** the similarity of the target word, context word pairs ($w$ , $c_{pos}$) drawn from the positive data
  - **Minimize** the similarity of the ($w$ , $c_{neg}$) pairs drawn from the negative data.

## Loss function for one *w with* $c_{pos}$ *,* $c_{neg1}$ ...$c_{negk}$

- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the *k* negative sampled non-neighbor words.

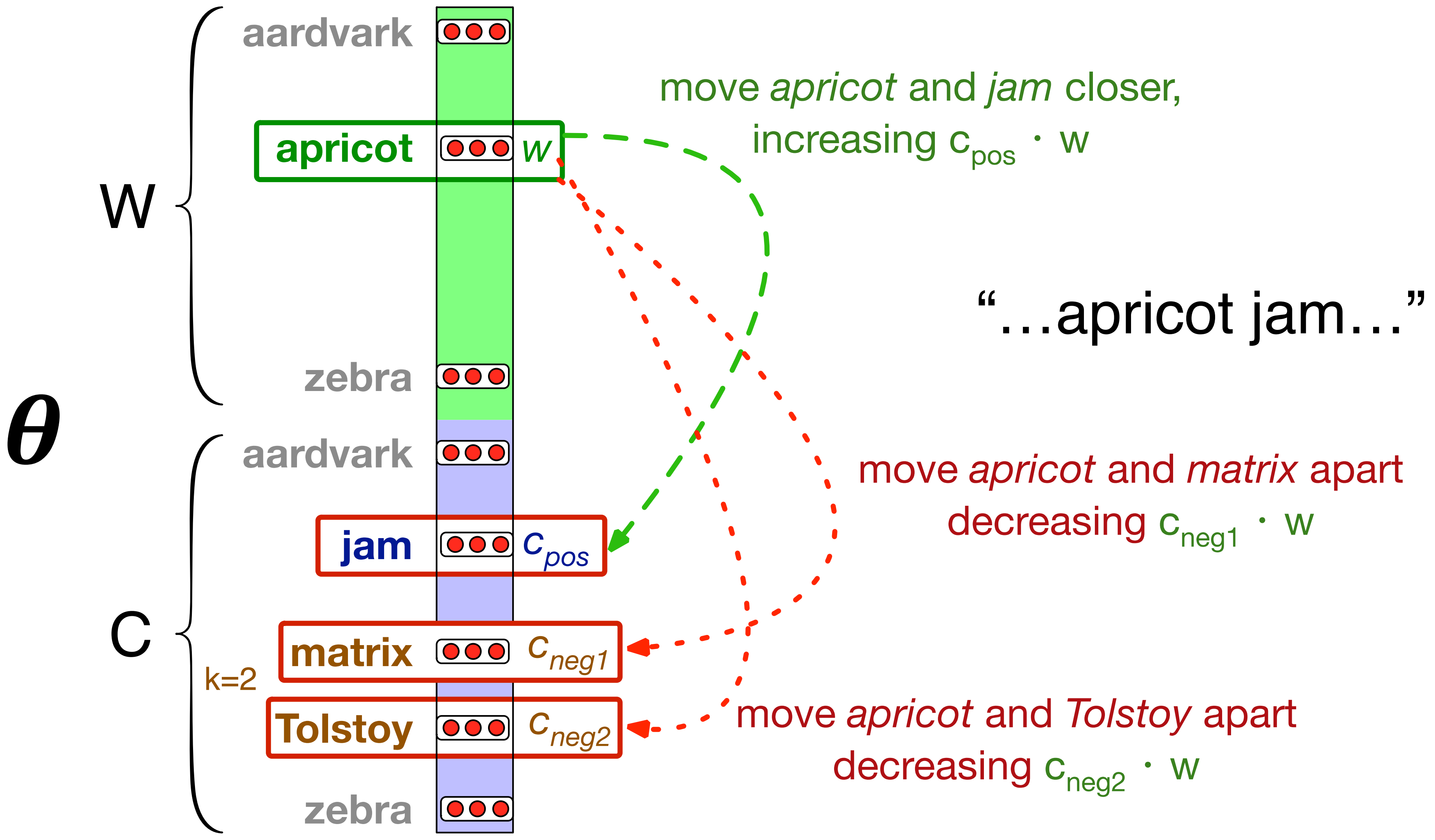
$$\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log\sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}$$

## Learning the classifier

- How to learn?
  - Stochastic gradient descent!

- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.

# Intuition of one step of gradient descent



W
aardvark
apricot
$w$
zebra
$\theta$
C
aardvark
jam
$c_{pos}$
k=2
matrix
$c_{neg1}$
Tolstoy
$c_{neg2}$
zebra
move *apricot* and *jam* closer, increasing $c_{pos} \cdot w$
"…apricot jam…"
move *apricot* and *matrix* apart decreasing $c_{neg1} \cdot w$
move *apricot* and *Tolstoy* apart decreasing $c_{neg2} \cdot w$

## Reminder: gradient descent

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η
  - Higher learning rate means move *w* faster

$$w^{t+1} = w^{t} - h\,\frac{d}{dw}L(f(x;w),y)$$

## The derivatives of the loss function

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w)]c_{neg_i}$$

## Update equation in SGD

- Start with randomly initialized C and W matrices, then incrementally do updates

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i}\right]
\end{aligned}
$$

## Two sets of embeddings

- SGNS learns two sets of embeddings
    - Target embeddings matrix W
    - Context embedding matrix C
- It's common to just add them together, representing word *i* as the vector $w_i + c_i$

## Summary: How to learn word2vec (skip-gram) embeddings

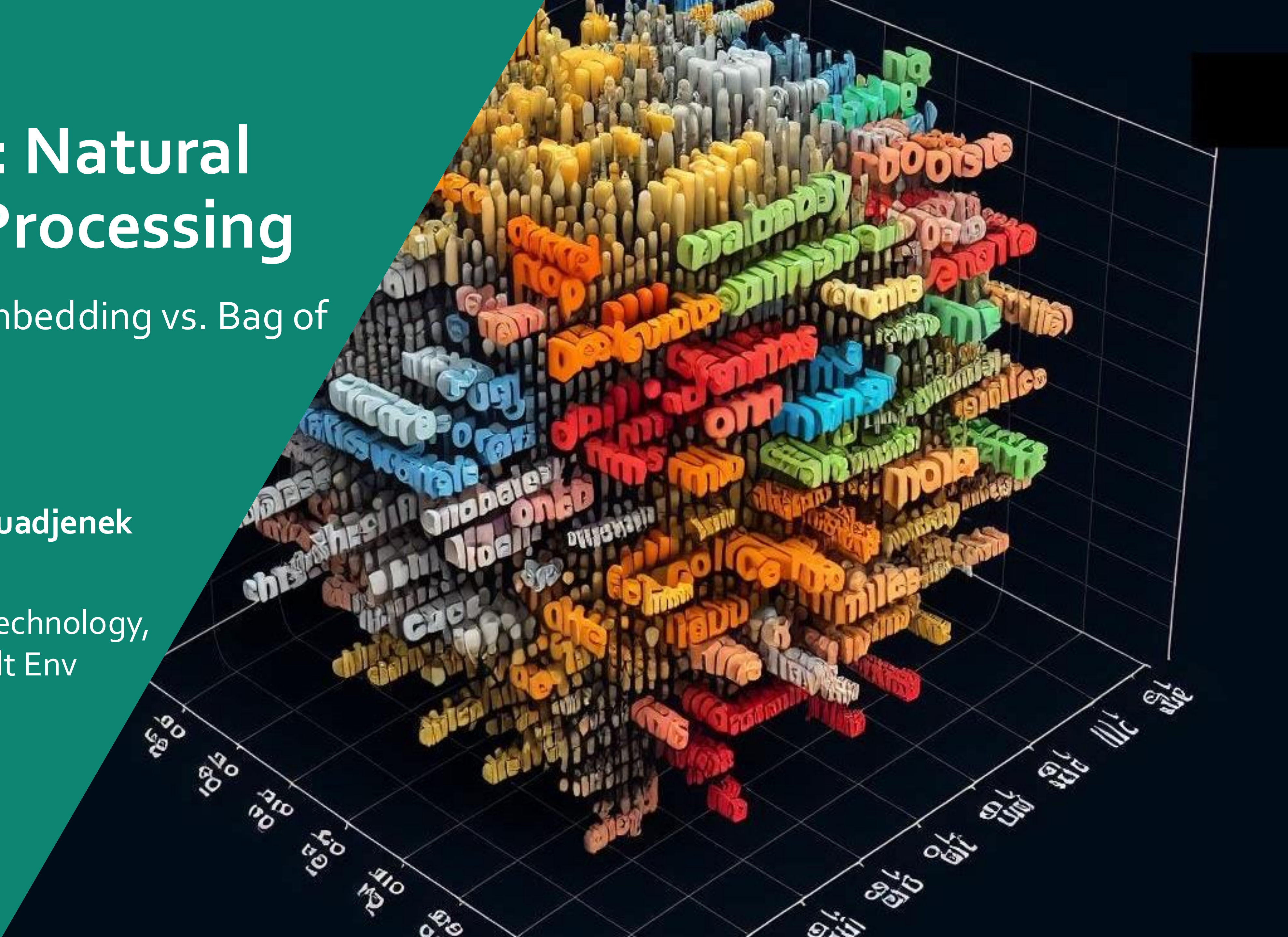
- Start with V random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

# SIT330-770: Natural Language Processing

Week 5.6 – Word Embedding vs. Bag of Words

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology, Faculty of Sci Eng & Built Env



## Word Embedding vs. Bag of Words

### Traditional Method - Bag of Words Model

**Two approaches:**

- Either uses one hot encoding.
  - Each word in the vocabulary is represented by one bit position in a HUGE vector.
  - For example, if we have a vocabulary of 10,000 words, and "aardvark" is the ***4th word in the dictionary***, it would be represented by: [0 0 0 1 0 0 . . . . . . . 0 0 0].
- Or uses document representation.
  - Each word in the vocabulary is represented by its presence in documents.
  - For example, if we have a corpus of 1M documents, and "Hello" is in 1th, 3th and 5th documents ***only***, it would be represented by: [1 0 1 0 1 0 . . . . . . . 0 0 0].
- Assumes independence between words.

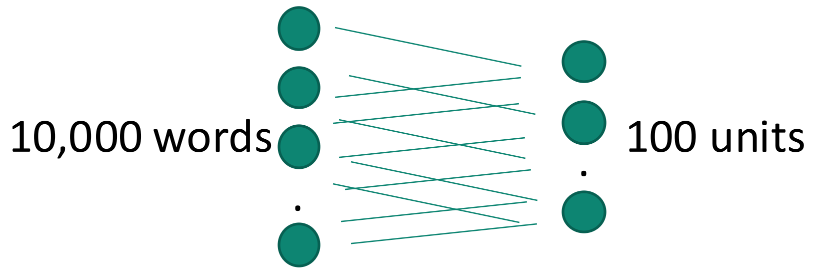
### Word Embeddings

- Stores each word in as a point in space, where it is represented by a dense vector of fixed number of dimensions (generally 300) .
  - For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02].
  - Dimensions are projections along different axes, more of a mathematical concept.
- Unsupervised, built just by reading huge corpus.
- Assumes dependence between words.

## Word Embedding vs. Bag of Words

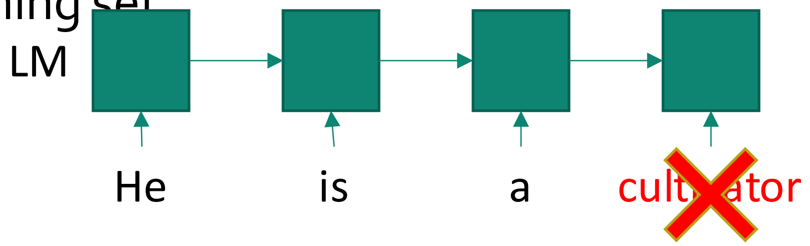
### Traditional Method - Bag of Words Model

- Requires **very** large weight matrix for 1st layers.



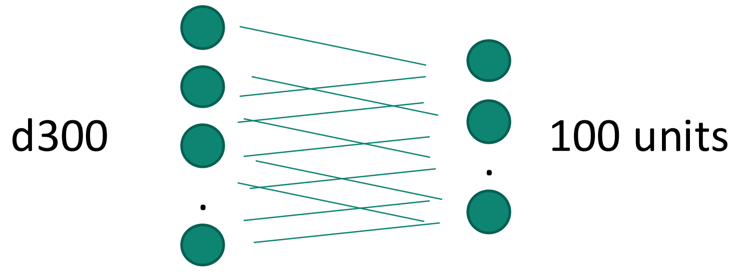
W's size is 10,000x100 = $10^6$

- Models not flexible with unseen words in the training set



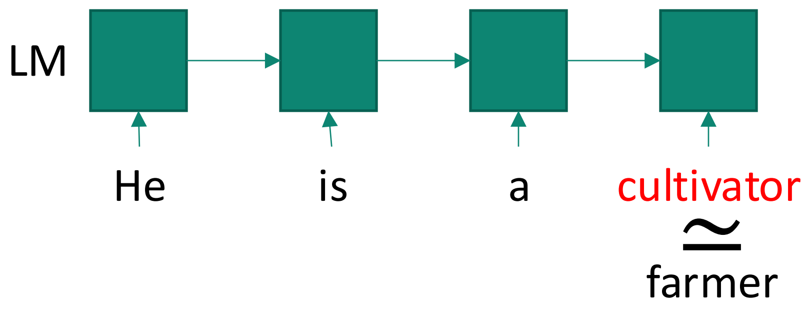
### Word Embeddings

- A **compact** weight matrix for 1st layers.



W's size is 300x100 = $3\text{x}10^4$

- Flexible models with unseen words in the training set.

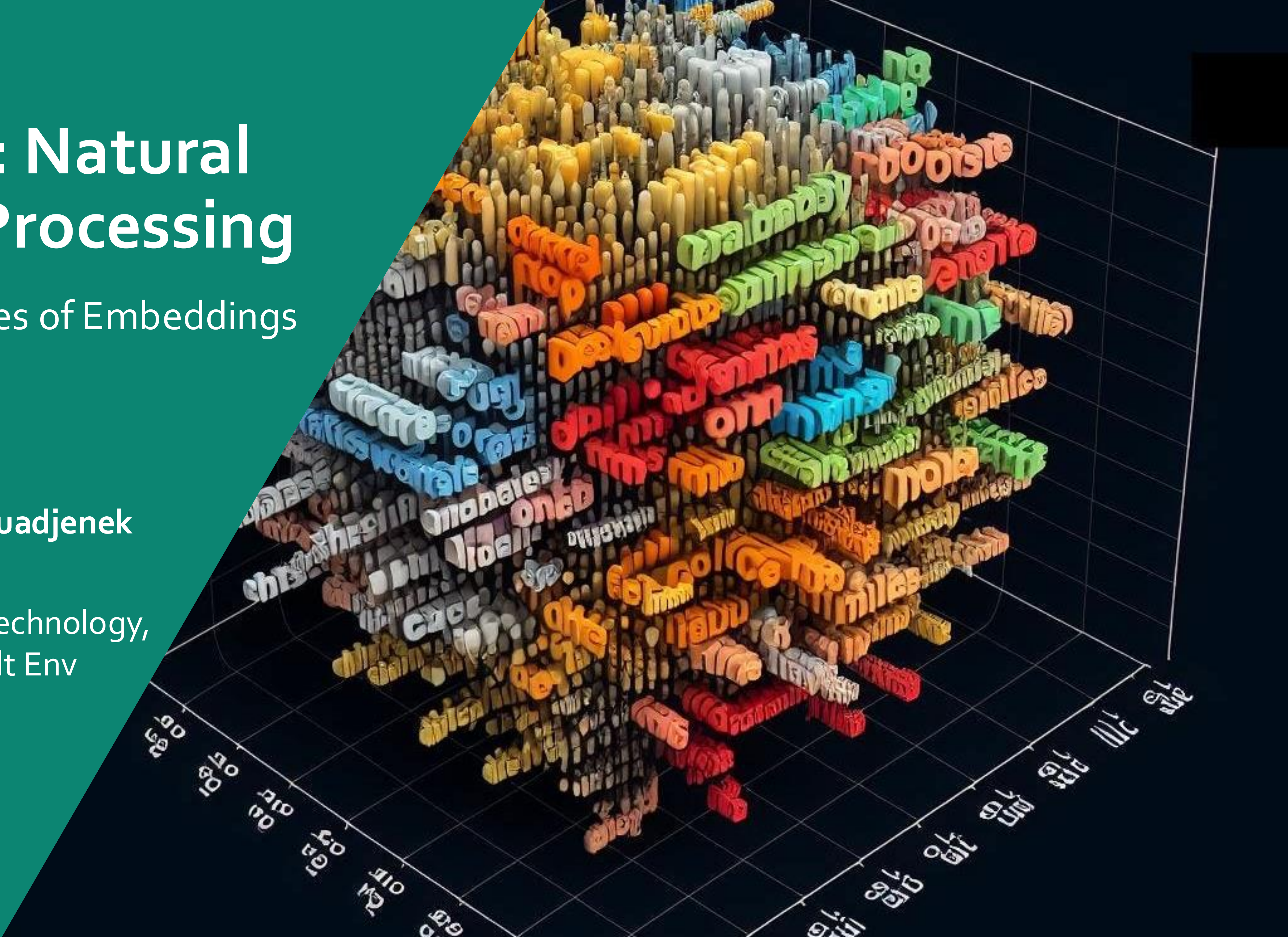


# SIT330-770: Natural Language Processing

Week 5.7 – Properties of Embeddings

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env

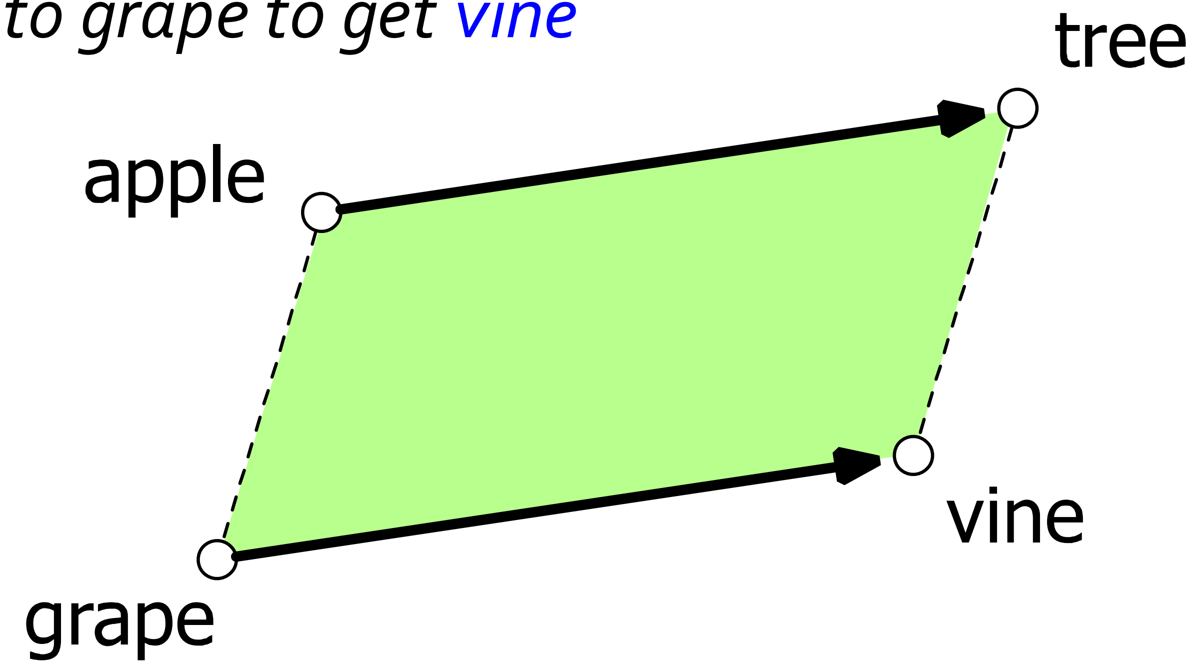


## The kinds of neighbors depend on window size

- **Small windows** (C= +/- 2) : nearest words are syntactically similar words in same taxonomy
  - *Hogwarts* nearest neighbors are other fictional schools
  - *Sunnydale, Evernight, Blandings*
- **Large windows** (C= +/- 5) : nearest words are related words in same semantic field
  - *Hogwarts* nearest neighbors are Harry Potter world:
  - *Dumbledore, half-blood, Malfoy*

## Analogical relations

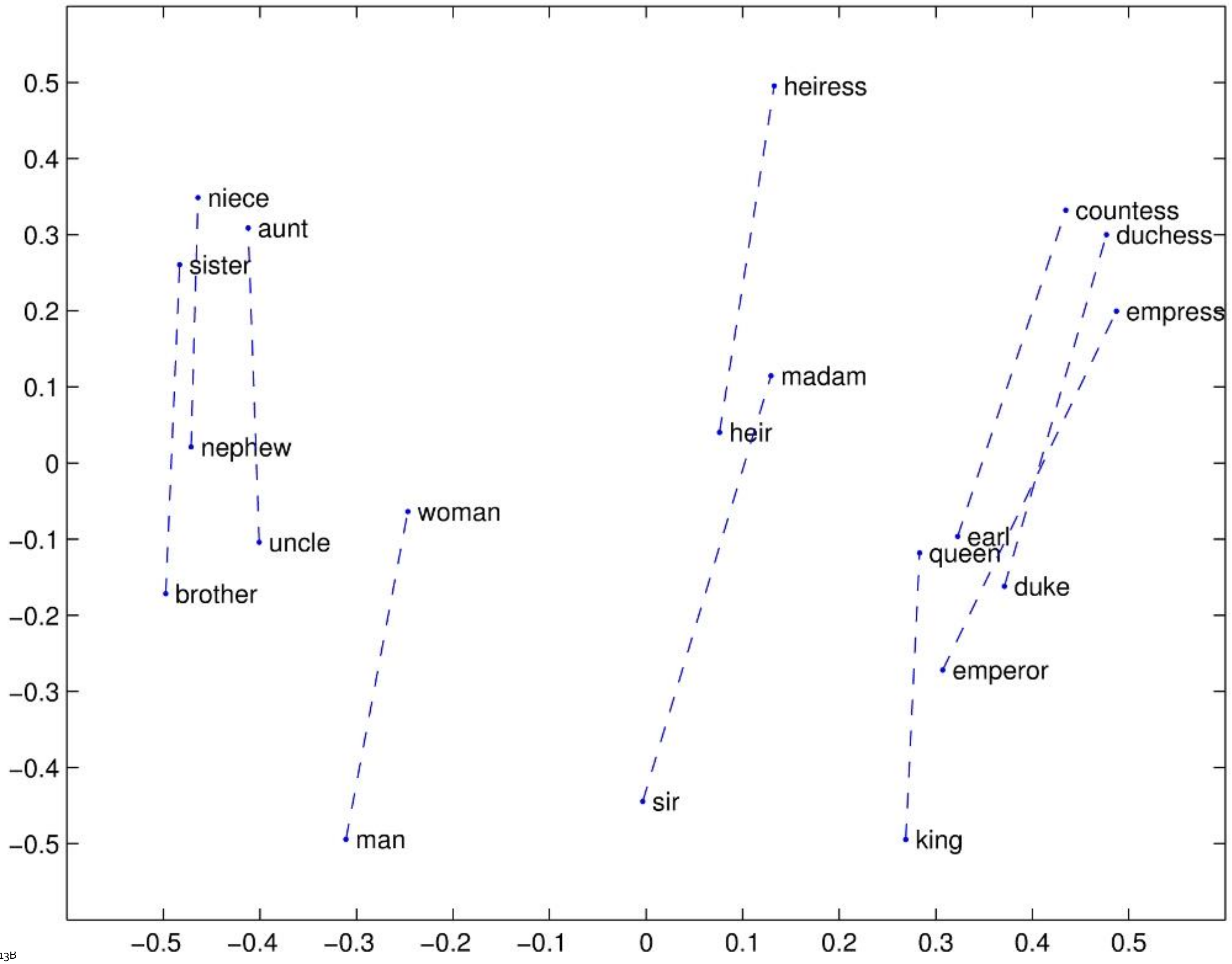
- The classic parallelogram model of analogical reasoning (Rumelhart and Abrahamson 1973)
- To solve: "*apple is to tree as grape is to* _____"
- *Add tree – apple to grape to get vine*



## Analogical relations via parallelogram

- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)
- king – man + woman is close to queen
- Paris – France + Italy is close to Rome
- For a problem a:a*::b:b*, the parallelogram method is:

$$\hat{b}^* = \underset{x}{\operatorname{argmax}}\ \text{distance}(x, a^* - a + b)$$


heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
brother
earl
queen
duke
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5

## Caveats with the parallelogram method

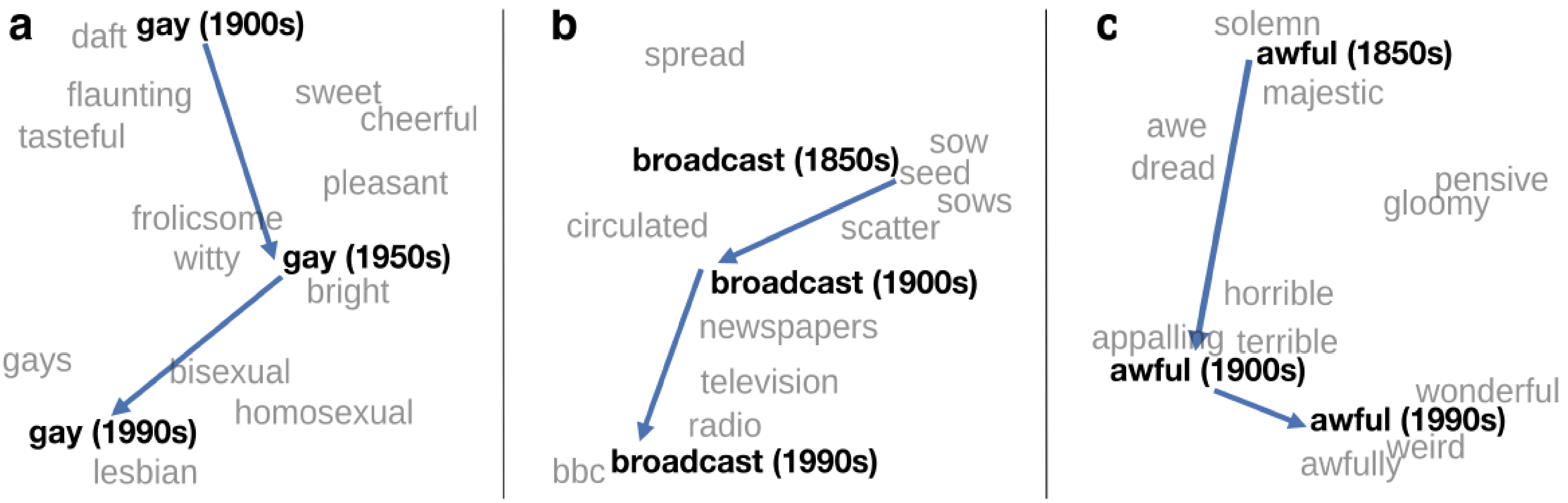
- It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

- Understanding analogy is an open area of research (Peterson et al. 2020)

## Embeddings as a window onto historical semantics

- Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data



William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

## Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

- Ask "Paris : France :: Tokyo : x"
  - x = Japan
- Ask "father : doctor :: mother : x"
  - x = nurse
- Ask "man : computer programmer :: woman : x"
  - x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

## Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical)* are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

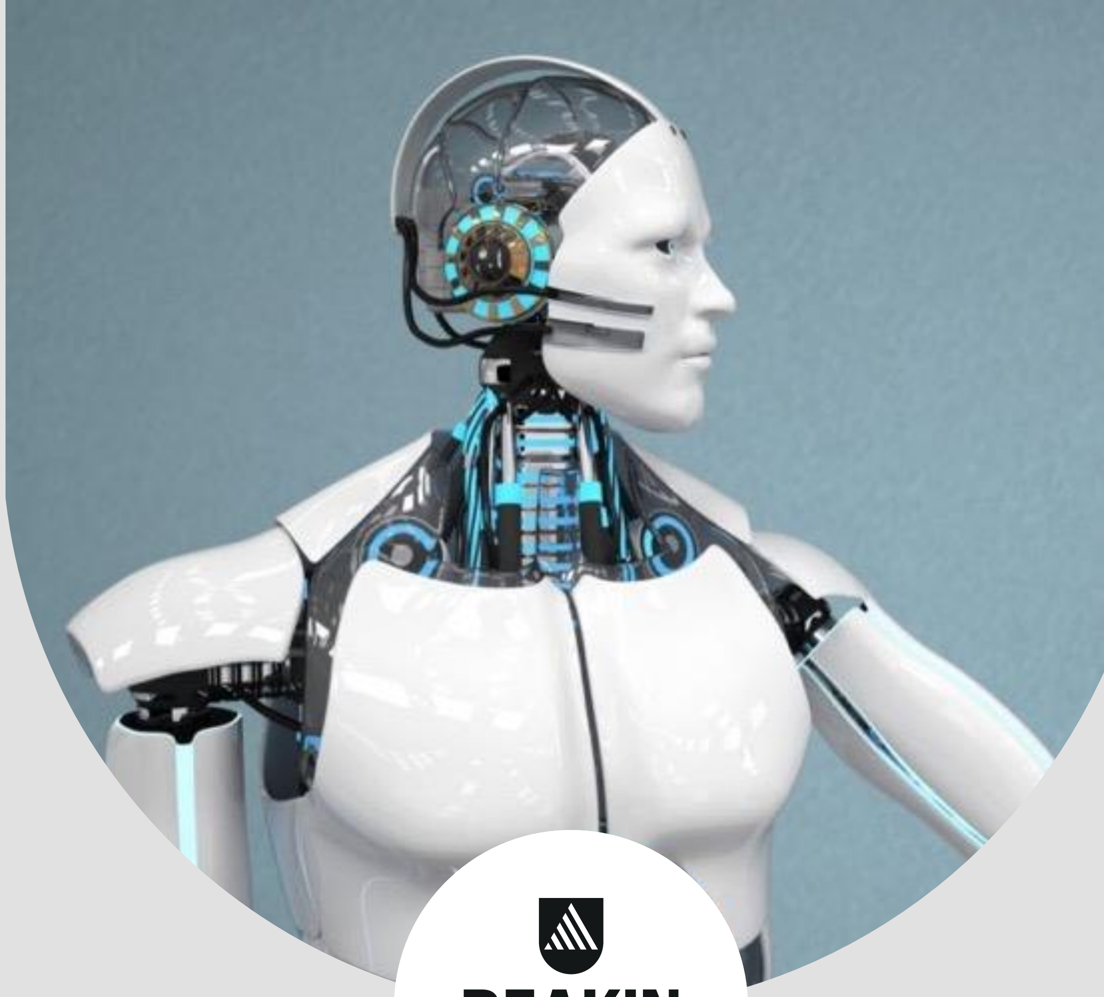
# SIT330-770: Natural Language Processing

## Week 5 - Sequence Labeling

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology, Faculty of Sci Eng & Built Env

reda.bouadjenek@deakin.edu.au



DEAKIN UNIVERSITY

# SIT330-770: Natural Language Processing

## Week 5.8 - English Word Classes

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env

## Parts of Speech

- From the earliest linguistic traditions (Yaska and Panini 5th C. BCE, Aristotle 4th C. BCE), the idea that words can be classified into grammatical categories
  - part of speech, word classes, POS, POS tags
- 8 parts of speech attributed to Dionysius Thrax of Alexandria (c. 1st C. BCE):
  - noun, verb, pronoun, preposition, adverb, conjunction, participle, article
  - These categories are relevant for NLP today.

## Two classes of words: Open vs. Closed

- Closed class words
  - Relatively fixed membership
  - Usually **function** words: short, frequent words with grammatical function
    - determiners: ***a, an, the***
    - pronouns: ***she, he, I***
    - prepositions: ***on, under, over, near, by, …***
- Open class words
  - Usually **content** words: Nouns, Verbs, Adjectives, Adverbs
    - Plus interjections: **oh, ouch, uh-huh, yes, hello**
  - New nouns and verbs like *iPhone* or *to fax*

## **Open class** ("content") words

- Nouns
  - Proper: *Janet*, *Italy*
  - Common: *cat, cats*, *mango*
- Verbs
  - Main: *eat*, *went*
  - Auxiliary: *can*, *had*
- Adjectives *old green tasty*
- Adverbs *slowly yesterday*
- Numbers *122,312* *one*
- Interjections *Ow hello*
- *… more*

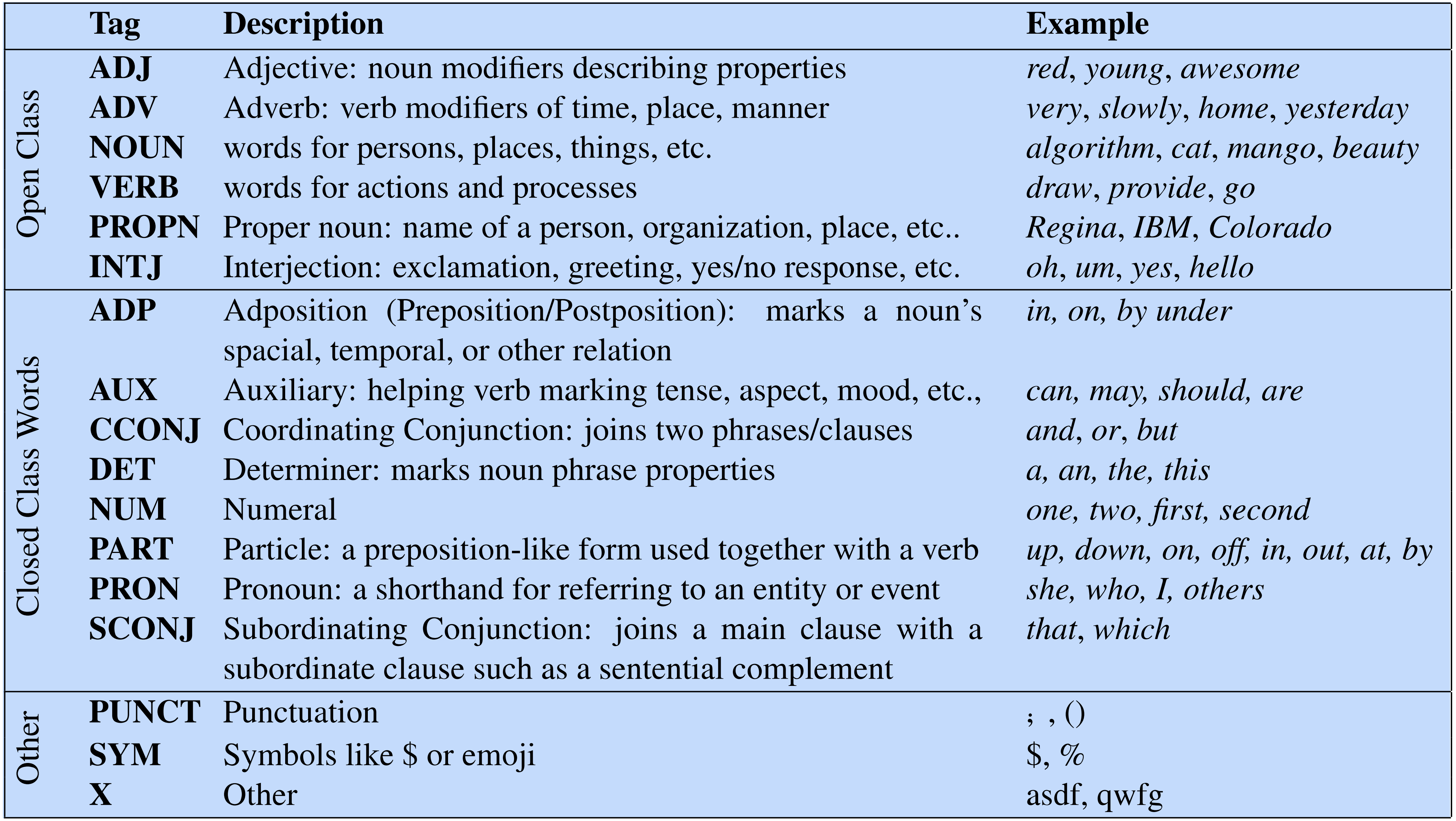
## **Closed class** ("function")

- Determiners *the some*
- Conjunctions *and or*
- Pronouns *they its*
- Prepositions *to with*
- Particles *off up*
- *… more*

## Part-of-Speech Tagging

- Assigning a part-of-speech to each word in a text.
- Words often have more than one POS.
- **book**:
  - VERB: (***Book** that flight*)
  - NOUN: (*Hand me that **book***).

## "Universal Dependencies" Tagset



| | Tag | Description | Example |
|---|---|---|---|
| Open Class | **ADJ** | Adjective: noun modifiers describing properties | *red*, *young*, *awesome* |
| | **ADV** | Adverb: verb modifiers of time, place, manner | *very*, *slowly*, *home*, *yesterday* |
| | **NOUN** | words for persons, places, things, etc. | *algorithm*, *cat*, *mango*, *beauty* |
| | **VERB** | words for actions and processes | *draw*, *provide*, *go* |
| | **PROPN** | Proper noun: name of a person, organization, place, etc.. | *Regina*, *IBM*, *Colorado* |
| | **INTJ** | Interjection: exclamation, greeting, yes/no response, etc. | *oh*, *um*, *yes*, *hello* |
| Closed Class Words | **ADP** | Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation | *in, on, by under* |
| | **AUX** | Auxiliary: helping verb marking tense, aspect, mood, etc., | *can, may, should, are* |
| | **CCONJ** | Coordinating Conjunction: joins two phrases/clauses | *and*, *or*, *but* |
| | **DET** | Determiner: marks noun phrase properties | *a, an, the, this* |
| | **NUM** | Numeral | *one, two, first, second* |
| | **PART** | Particle: a preposition-like form used together with a verb | *up, down, on, off, in, out, at, by* |
| | **PRON** | Pronoun: a shorthand for referring to an entity or event | *she, who, I, others* |
| | **SCONJ** | Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement | *that*, *which* |
| Other | **PUNCT** | Punctuation | ; , () |
| | **SYM** | Symbols like $ or emoji | $, % |
| | **X** | Other | asdf, qwfg |

Nivre et al. 2016

## Sample "Tagged" English sentences

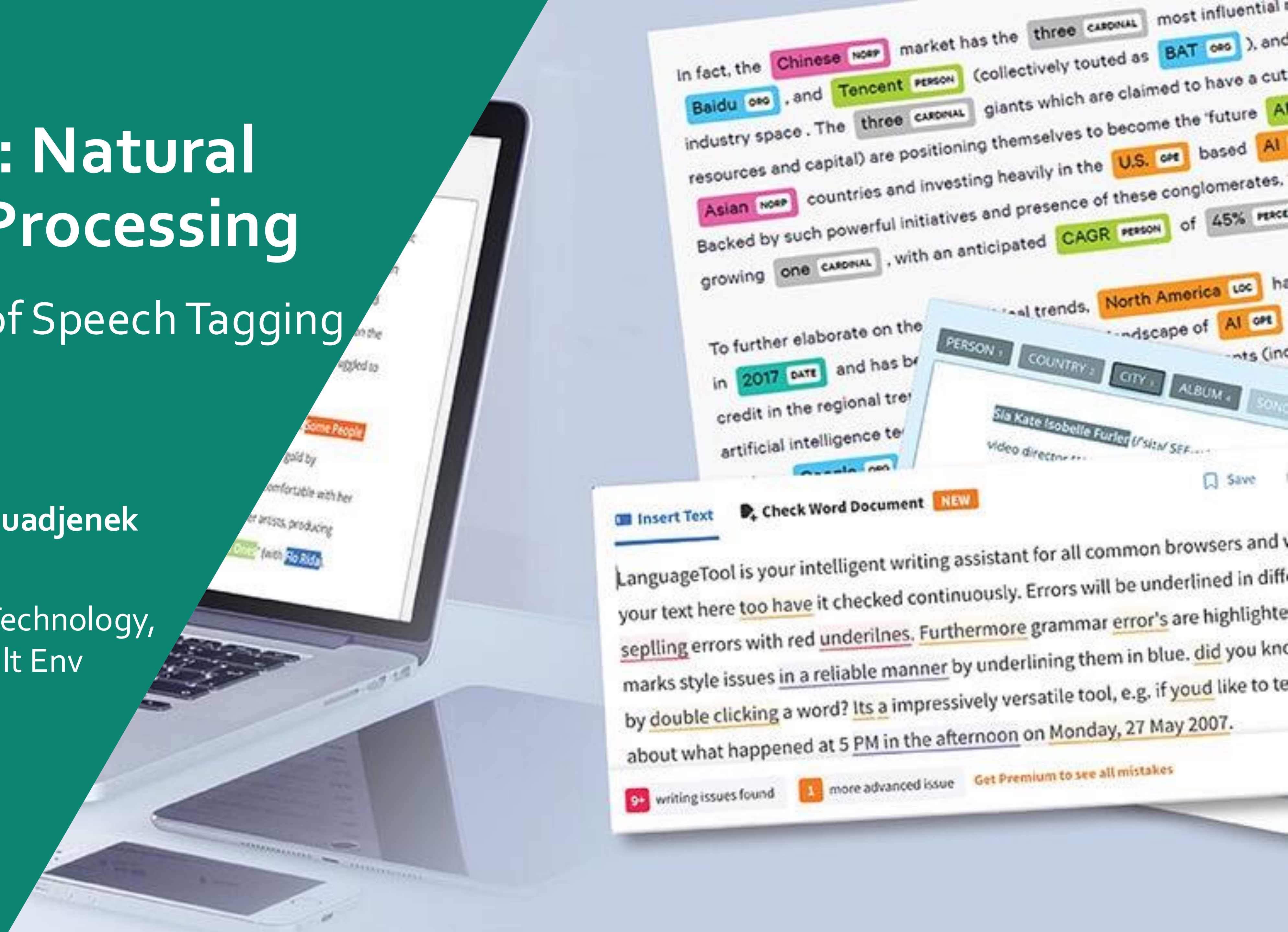
- There/PRO were/VERB 70/NUM children/NOUN there/ADV ./PUNC
- Preliminary/ADJ findings/NOUN were/AUX reported/VERB in/ADP today/NOUN 's/PART New/PROPN England/PROPN Journal/PROPN of/ADP Medicine/PROPN

# SIT330-770: Natural Language Processing

## Week 5.9 - Part of Speech Tagging

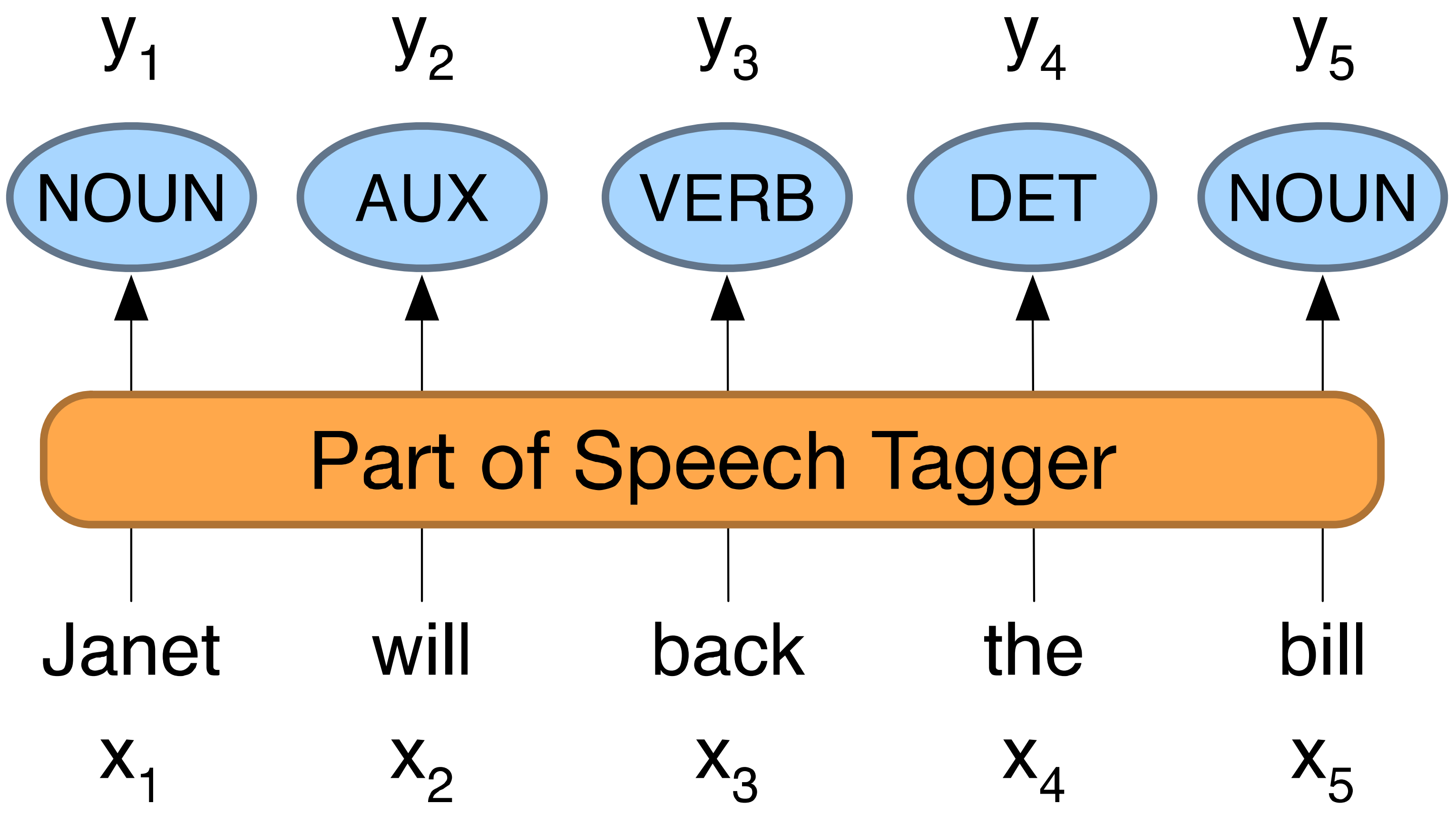
**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env



## Part-of-Speech Tagging

- Map from sequence $x_1,\ldots,x_n$ of words to $y_1,\ldots,y_n$ of POS tags



## Why Part of Speech Tagging?

- Can be useful for other NLP tasks
  - Parsing: POS tagging can improve syntactic parsing
  - MT: reordering of adjectives and nouns (say from Spanish to English)
  - Sentiment or affective tasks: may want to distinguish adjectives or other POS
  - Text-to-speech (how do we pronounce “lead” or "object"?)
- Or linguistic or language-analytic computational tasks
  - Need to control for POS when studying linguistic change like creation of new words, or meaning shift
  - Or control for POS in measuring meaning similarity or difference

## How difficult is POS tagging in English?

- Roughly 15% of word types are ambiguous
  - Hence 85% of word types are unambiguous
  - *Janet* is always PROPN, *hesitantly* is always ADV
- But those 15% tend to be very common.
- So ~60% of word tokens are ambiguous
- E.g., *back*
  - earnings growth took a *back*/ADJ seat
  - a small building in the *back*/NOUN
  - a clear majority of senators *back*/VERB the bill
  - enable the country to buy *back*/PART debt
  - I was twenty-one *back*/ADV then

## POS tagging performance in English

- How many tags are correct? (Tag accuracy)
  - About 97%
    - Hasn't changed in the last 10+ years
    - HMMs, CRFs, BERT perform similarly .
    - Human accuracy about the same
- But baseline is 92%!
  - Baseline is performance of stupidest possible method
    - "Most frequent class baseline" is an important baseline for many tasks
      - Tag every word with its most frequent tag
      - (and tag unknown words as nouns)
  - Partly easy because
    - Many words are unambiguous

## Sources of information for POS tagging

```
Janet will back the bill
```

AUX/NOUN/VERB? NOUN/VERB?

- Prior probabilities of word/tag
  - "will" is usually an AUX
- Identity of neighboring words
  - "the" means the next word is probably not a verb
- Morphology and wordshape:
  - Prefixes unable: un- $\rightarrow$ ADJ
  - Suffixes importantly: -ly $\rightarrow$ ADV
  - Capitalization Janet: CAP $\rightarrow$ PROPN

## Standard algorithms for POS tagging

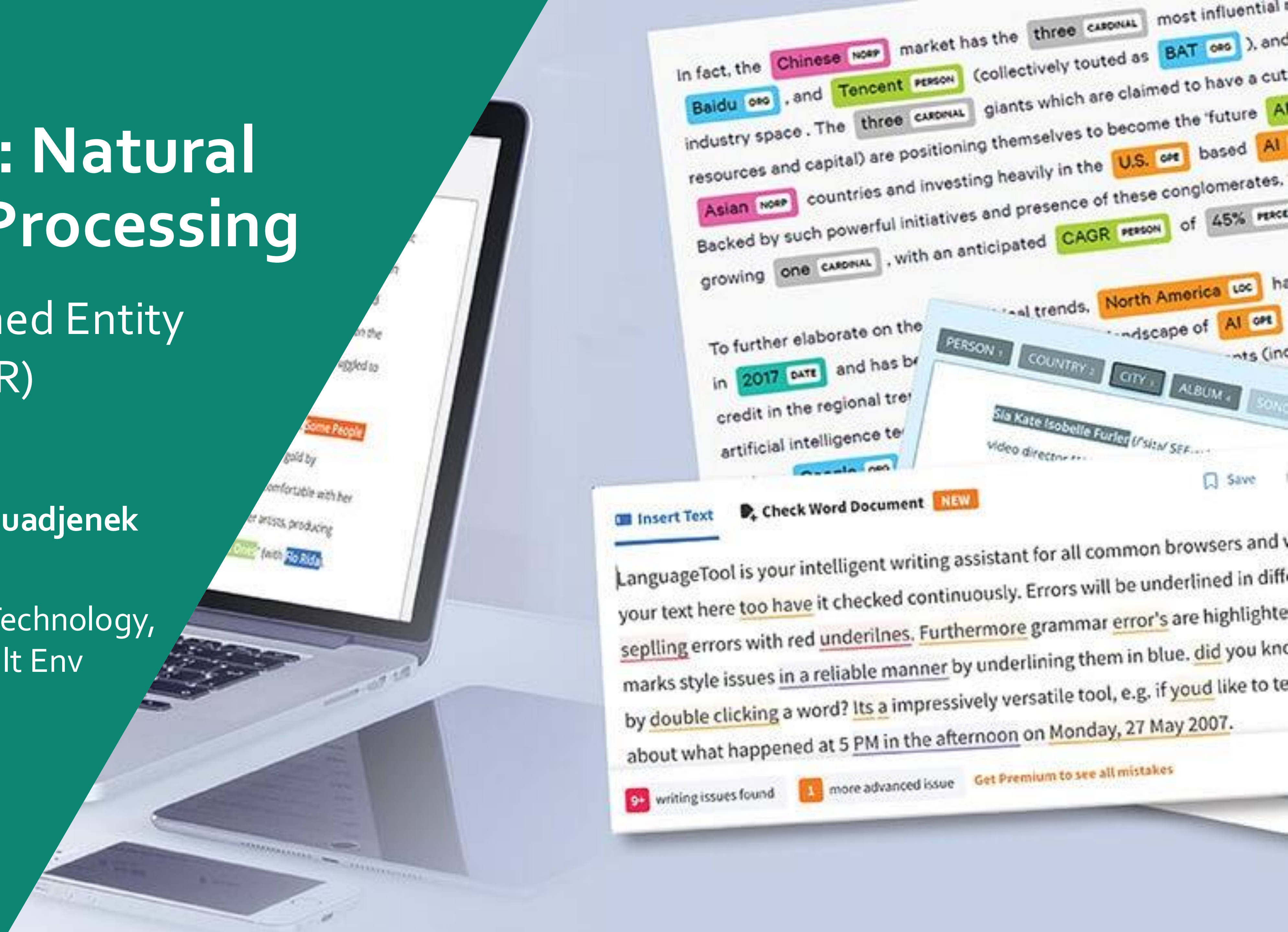
- Supervised Machine Learning Algorithms:
- Hidden Markov Models
- Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Large Language Models (like BERT), finetuned
- All required a hand-labeled training set, all about equal performance (97% on English)
- All make use of information sources we discussed
- Via human created features: HMMs and CRFs
- Via representation learning:  Neural LMs

# SIT330-770: Natural Language Processing

## Week 5.10 - Named Entity Recognition (NER)

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology, Faculty of Sci Eng & Built Env



## Named Entities

- **Named entity**, in its core usage, means anything that can be referred to with a proper name. Most common 4 tags:
  - PER (Person): “Marie Curie”
  - LOC (Location): “New York City”
  - ORG (Organization): “Stanford University”
  - GPE (Geo-Political Entity): "Boulder, Colorado”
- Often multi-word phrases
- But the term is also extended to things that aren't entities:
  - dates, times, prices

## Named Entity tagging

- The task of named entity recognition (NER):
  - find spans of text that constitute proper names
  - tag the type of the entity.

## NER output

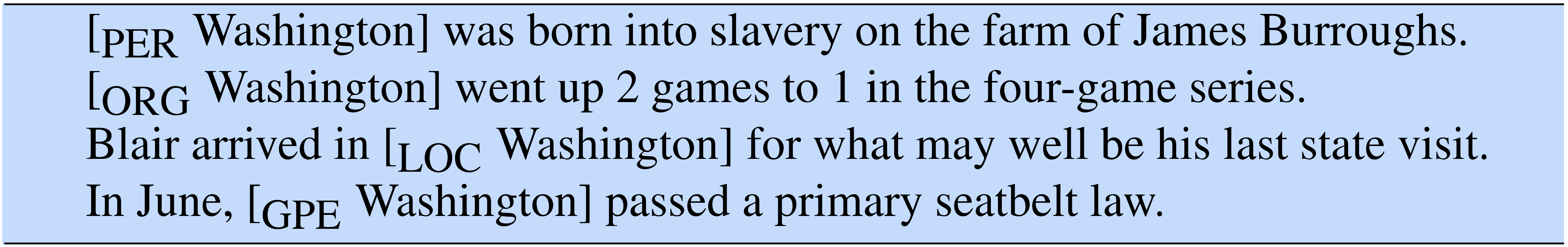
Citing high fuel prices, [$_{\text{ORG}}$ **United Airlines**] said [$_{\text{TIME}}$ **Friday**] it has increased fares by [$_{\text{MONEY}}$ **$6**] per round trip on flights to some cities also served by lower-cost carriers. [$_{\text{ORG}}$ **American Airlines**], a unit of [$_{\text{ORG}}$ **AMR Corp.**], immediately matched the move, spokesman [$_{\text{PER}}$ **Tim Wagner**] said. [$_{\text{ORG}}$ **United**], a unit of [$_{\text{ORG}}$ **UAL Corp.**], said the increase took effect [$_{\text{TIME}}$ **Thursday**] and applies to most routes where it competes against discount carriers, such as [$_{\text{LOC}}$ **Chicago**] to [$_{\text{LOC}}$ **Dallas**] and [$_{\text{LOC}}$ **Denver**] to [$_{\text{LOC}}$ **San Francisco**].

## Why NER?

- Sentiment analysis: consumer’s sentiment toward a particular company or person?
- Question Answering: answer questions about an entity?
- Information Extraction: Extracting facts about entities from text.

## Why NER is hard

1) Segmentation
   - In POS tagging, no segmentation problem since each word gets one tag.
   - In NER we have to find and segment the entities!
2) Type ambiguity



[$_{PER}$ Washington] was born into slavery on the farm of James Burroughs.
[$_{ORG}$ Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [$_{LOC}$ Washington] for what may well be his last state visit.
In June, [$_{GPE}$ Washington] passed a primary seatbelt law.

## BIO Tagging

- How can we turn this structured problem into a sequence problem like POS tagging, with one label per word?

- [PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago ] route.

## BIO Tagging

- [PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago ] route.

| Words | BIO Label |
|---|---|
| Jane | B-PER |
| Villanueva | I-PER |
| of | O |
| United | B-ORG |
| Airlines | I-ORG |
| Holding | I-ORG |
| discussed | O |
| the | O |
| Chicago | B-LOC |
| route | O |
| . | O |

Now we have one tag per token!!!

## BIO Tagging

B: token that *begins* a span

I: tokens *inside* a span

O: tokens outside of any span

# of tags (where n is #entity types):

1 O tag,

*n* B tags,

*n* I tags

total of 2*n*+1

| Words | BIO Label |
|---|---|
| Jane | B-PER |
| Villanueva | I-PER |
| of | O |
| United | B-ORG |
| Airlines | I-ORG |
| Holding | I-ORG |
| discussed | O |
| the | O |
| Chicago | B-LOC |
| route | O |
| . | O |

## BIO Tagging variants: IO and BIOES

- [PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago ] route.

| Words | IO Label | BIO Label | BIOES Label |
|---|---|---|---|
| Jane | I-PER | B-PER | B-PER |
| Villanueva | I-PER | I-PER | E-PER |
| of | O | O | O |
| United | I-ORG | B-ORG | B-ORG |
| Airlines | I-ORG | I-ORG | I-ORG |
| Holding | I-ORG | I-ORG | E-ORG |
| discussed | O | O | O |
| the | O | O | O |
| Chicago | I-LOC | B-LOC | S-LOC |
| route | O | O | O |
| . | O | O | O |

## Standard algorithms for NER

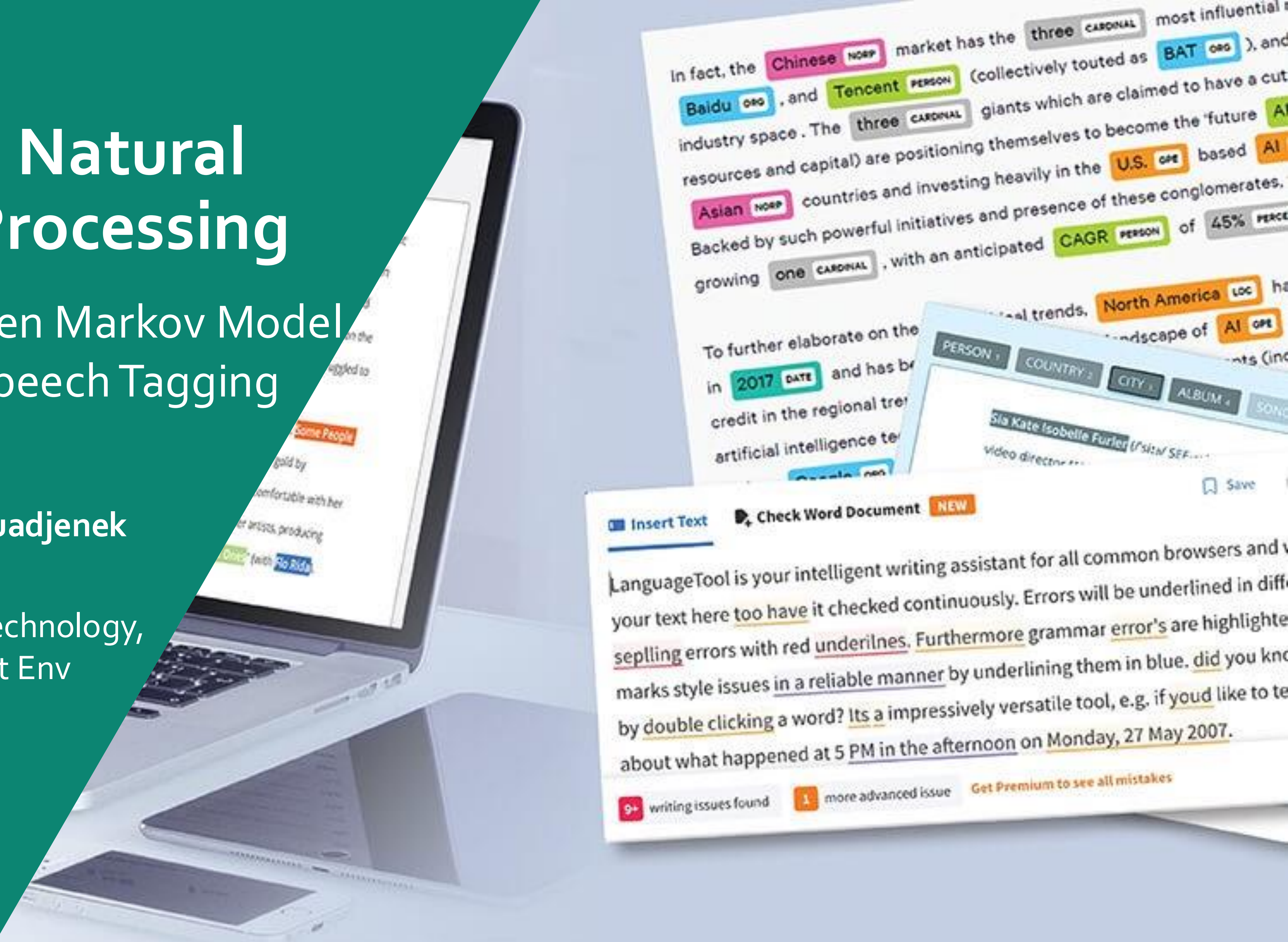
- Supervised Machine Learning given a human-labeled training set of text annotated with tags
    - Hidden Markov Models
    - Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
    - Neural sequence models (RNNs or Transformers)
    - Large Language Models (like BERT), finetuned

# SIT330-770: Natural Language Processing

## Week 5.11 – Hidden Markov Model (HMM) Part-of-Speech Tagging

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology, Faculty of Sci Eng & Built Env



## Introduction to Markov Chains

- A Markov chain models the probabilities of state sequences, each drawn from a specific set.
- It assumes the future state depends only on the current state, not any prior ones.
- Markov chains are used to predict various phenomena
  - E.g., modeling weather patterns or word sequences.

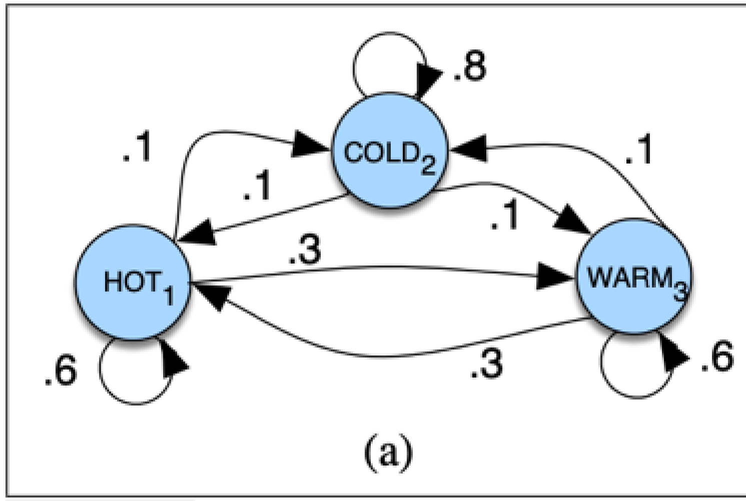
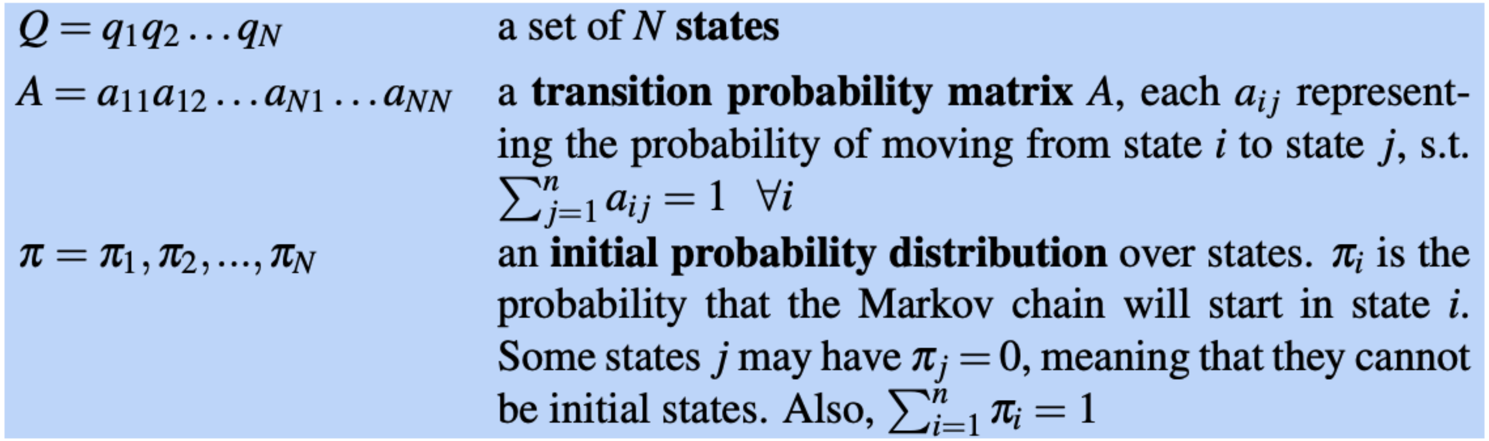


**Figure 8.8** A Markov chain for weather

# Markov Chain Representation



| | |
|---|---|
| $Q = q_1 q_2 \dots q_N$ | a set of $N$ **states** |
| $A = a_{11} a_{12} \dots a_{N1} \dots a_{NN}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$ |
| $\pi = \pi_1, \pi_2, \dots, \pi_N$ | an **initial probability distribution** over states. $\pi_i$ is the probability that the Markov chain will start in state $i$. Some states $j$ may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$ |

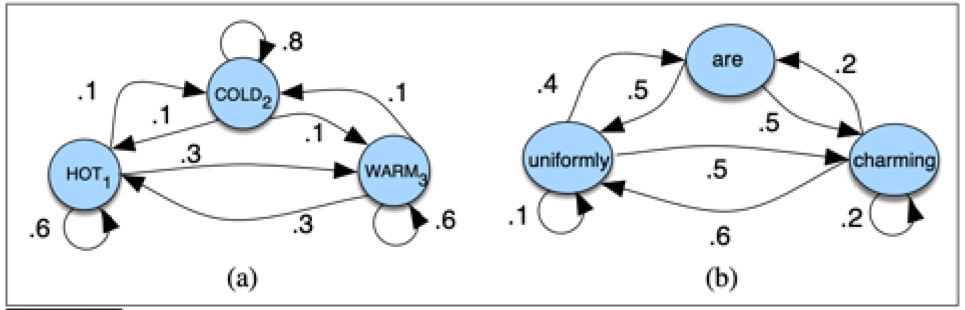


**Figure 8.8** A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution $\pi$ is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

- Markov Assumption:
  - Formally stated as: $P(q_i{=}a|q_1 \dots q_{i-1}) = P(q_i{=}a|q_{i-1})$ implying that when predicting the future, only the present state matters

## The Hidden Markov Model

- A Markov chain computes probabilities for sequences of observable events.
- But often, the events of interest are hidden.
  - **Example:** Part-of-speech tags in text—hidden because we don't observe them directly.
- **Solution:** Hidden Markov Model (HMM) handles both observed and hidden events.
  - HMMs augment Markov chains

## Probabilistic Sequence Modeling with HMMs

- A Hidden Markov Models (HMM) is a probabilistic sequence model that, given a sequence of units (words, letters, morphemes, sentences, etc.), computes a probability distribution over possible sequences of labels.
  - HMMs determine the likelihood of different label sequences and select the most probable sequence based on the observed data.
  - HMM is based on augmenting the **Markov chain**

## Input and Assumptions

- Input (O): Sequence of observations ($o_1$, $o_2$, ..., $o_T$) drawn from vocabulary V.
- Assumptions of first-order HMM:
  - Markov Assumption:
    - Probability of state $q_i$ depends only on the previous state ($q_{i-1}$).
      - $P(q_i|q_1 \ldots q_{i-1}) = P(q_i|q_{i-1})$
  - Output Independence:
    - Probability of observation $o_i$ depends only on the state that produced it $q_i$
      - $P(o_i | q_1, \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i | q_i)$

# SIT330-770: Natural Language Processing

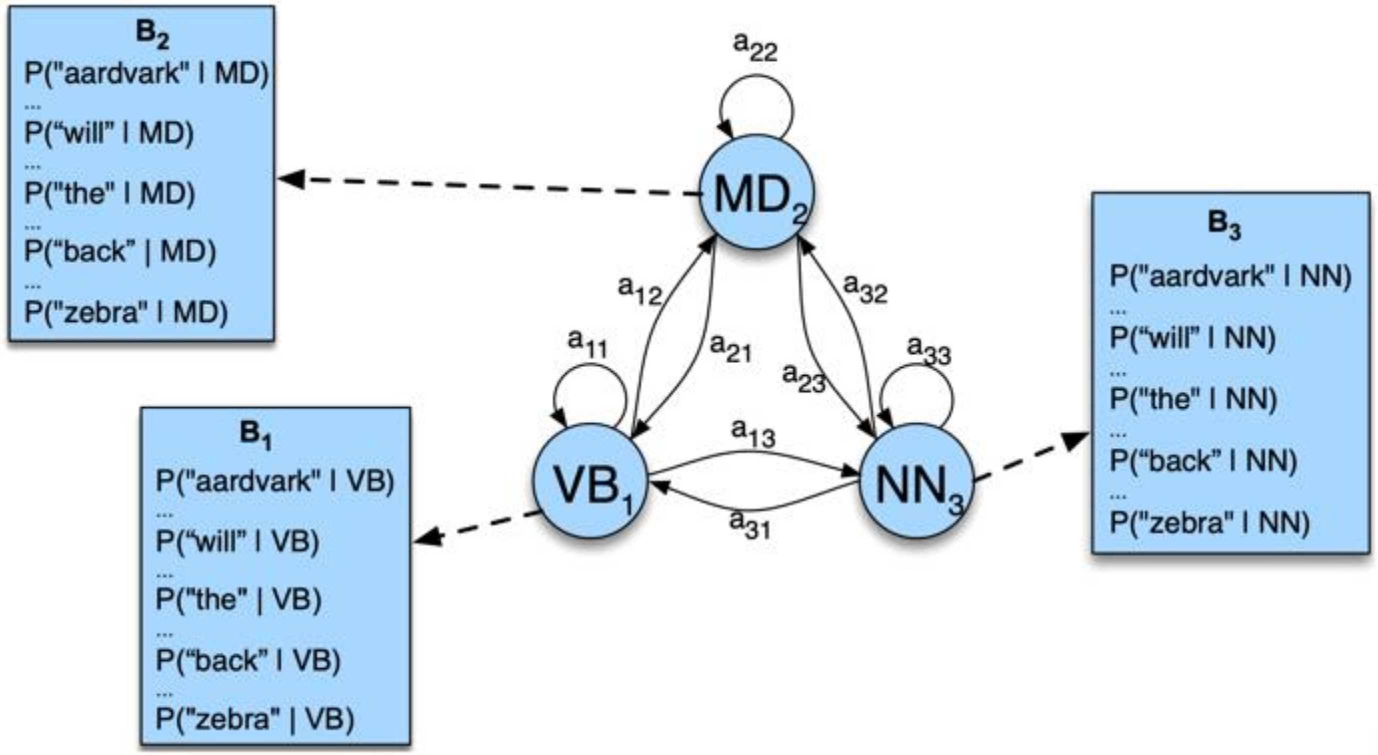
## Week 5.12 – The components of an HMM tagger

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env

## Components of an HMM Tagger

- An HMM tagger consists of two main components:
  - Matrix A which represents the tag transition.
  - Matrix B which represents emission probabilities.



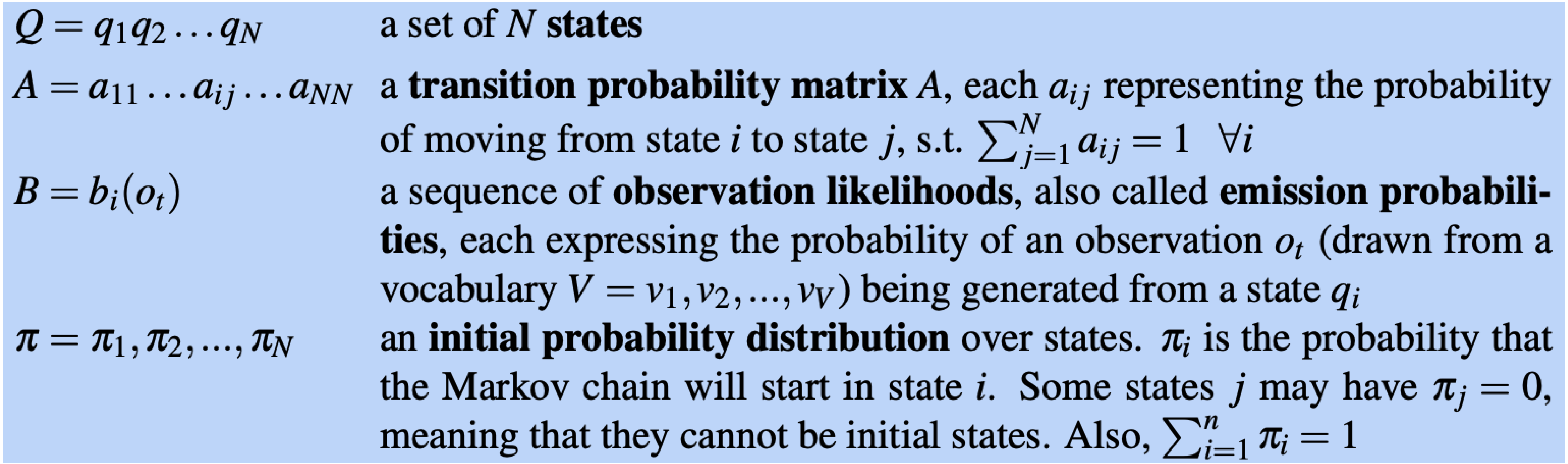
## The A Matrix - Transition Probabilities

- The A matrix encapsulates the tag transition probabilities, $P(t_i|t_{i-1})$, which express how likely a tag follows its predecessor.
    - Example:
        - The modal verb "***will***" commonly precedes the base form of a verb (VB), as in "***will race***", leading to a high transition probability.
    - These probabilities are derived using maximum MLE by counting tag occurrences in a labeled corpus.
- Calculating Transition Probabilities:
    - In the WSJ corpus example, the modal verb tag (MD) is observed 13,124 times.
    - Out of these, MD transitions to a base verb (VB) 10,471 times.
    - Using MLE, we estimate $P(VB|MD) = C(MD, VB) / C(MD) = 10{,}471 / 13{,}124 \approx 0.80$.

## The B Matrix - Emission Probabilities

- The B matrix contains emission probabilities, $P(w_i|t_i)$, which quantify the likelihood of a word being tagged with a specific tag.
- Emission Probability Calculation
  - To calculate emission probabilities, we count how often a word occurs with a particular tag in a corpus.
  - For instance, the MD tag associated with the word 'will' occurs 4,046 times in the WSJ corpus.
  - Hence, P(will|MD) is calculated as C(MD, will) / C(MD) = 4,046 / 13,124 ≈ 0.31.

## Components of HMM



| | |
|---|---|
| $Q = q_1 q_2 \ldots q_N$ | a set of $N$ **states** |
| $A = a_{11} \ldots a_{ij} \ldots a_{NN}$ | a **transition probability matrix** $A$, each $a_{ij}$ representing the probability of moving from state $i$ to state $j$, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$ |
| $B = b_i(o_t)$ | a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation $o_t$ (drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$) being generated from a state $q_i$ |
| $\pi = \pi_1, \pi_2, \ldots, \pi_N$ | an **initial probability distribution** over states. $\pi_i$ is the probability that the Markov chain will start in state $i$. Some states $j$ may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$ |

# SIT330-770: Natural Language Processing

## Week 5.13 – HMM tagging as decoding

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology, Faculty of Sci Eng & Built Env

## Decoding with Hidden Markov Models

- Decoding is the process of determining the most probable sequence of hidden states (tags) based on observed data.
  - Given a sequence of observations $O = o_1, o_2, \ldots, o_T$, decoding aims to find the most probable sequence of states $Q = q_1 q_2 \ldots q_T$.
  - The input is an HMM $\lambda = (A, B)$, with **A** being the transition probabilities and **B** the emission probabilities.

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(t_1 \ldots t_n | w_1 \ldots w_n)$$

## Decoding with Hidden Markov Models (i)

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(t_1 \ldots t_n | w_1 \ldots w_n)$$

MAP is "maximum a posteriori"
= most likely sequence

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} \frac{P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)}{P(w_1 \ldots w_n)}$$

Bayes Rule

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)$$

Dropping the
denominator

## Decoding with Hidden Markov Models (ii)

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} \overbrace{P(w_1 \ldots w_n | t_1 \ldots t_n)}^{\text{"Likelihood"}} \overbrace{P(t_1 \ldots t_n)}^{\text{"Prior"}}$$

- HMM taggers make two further simplifying assumptions.
  - The probability of a word appearing depends only on its own tag and is independent of neighboring words and tags:

$$P(w_1 \ldots w_n | t_1 \ldots t_n) \approx \prod_{i=1}^{n} P(w_i | t_i)$$

  - The second assumption, the bigram assumption, is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence;

$$P(t_1 \ldots t_n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1})$$

## Decoding with Hidden Markov Models (iii)

- Plugging the simplifying assumptions results in the following equation for the most probable tag sequence from a bigram tagger:

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(t_1 \ldots t_n | w_1 \ldots w_n) \approx \underset{t_1 \ldots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}^{\text{emission}} \overbrace{P(t_i|t_{i-1})}^{\text{transition}}$$

- The two parts correspond neatly to the **B** emission probability and **A** transition probability that we defined previously!

# SIT330-770: Natural Language Processing

## Week 5.14 – The Viterbi Algorithm

**Dr. Mohamed Reda Bouadjenek**

School of Information Technology,
Faculty of Sci Eng & Built Env

## Computing the most probable sequence of tags

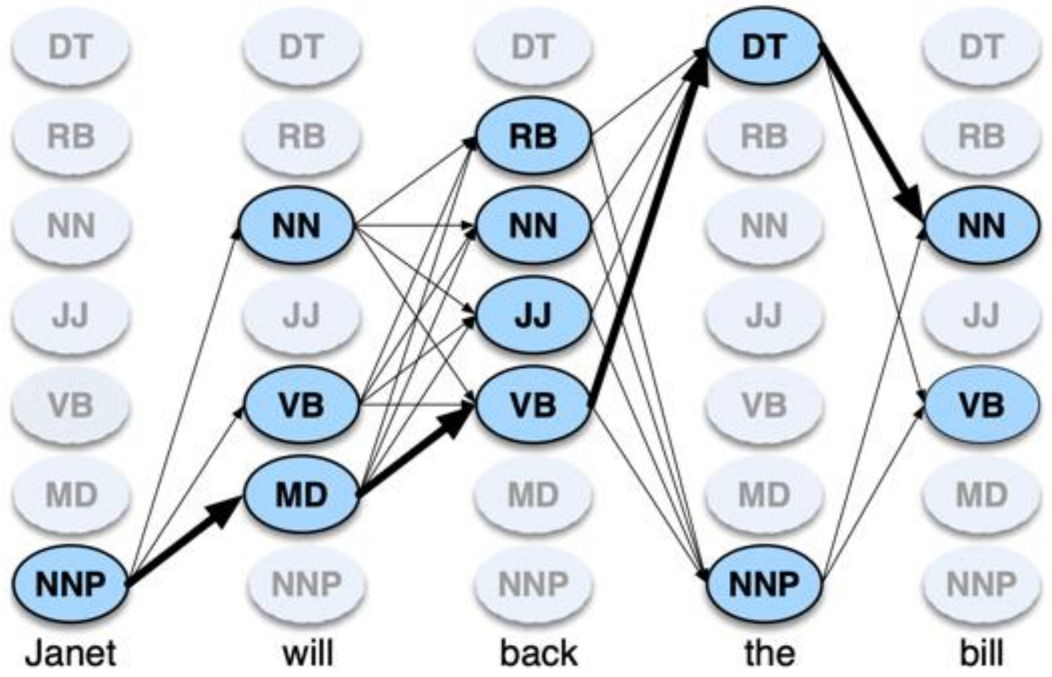
- A brute force approach to identify the most probable sequence of tags faces exponential complexity
  - This method is impractical for large datasets or real-time applications.
- Solution: The Viterbi algorithm **1967**
  - Leverages dynamic programming, streamlining the process by breaking the problem into manageable sub-problems
  - This approach significantly reduces computational demands and enhances processing speed, making it viable for complex tasks in real-world scenarios

**Andrew Viterbi**

## The Viterbi Algorithm (i)

- The decoding algorithm for HMMs is the **Viterbi algorithm**
  - As an instance of **dynamic programming**, Viterbi resembles the dynamic programming minimum edit distance algorithm
- The Viterbi algorithm first sets up a probability matrix or lattice:
  - **Columns as observables** (words of a sentence in the same sequence as in sentence)
  - Rows as hidden states (all possible POS Tags are known)

tag the sentence
Janet will back the bill

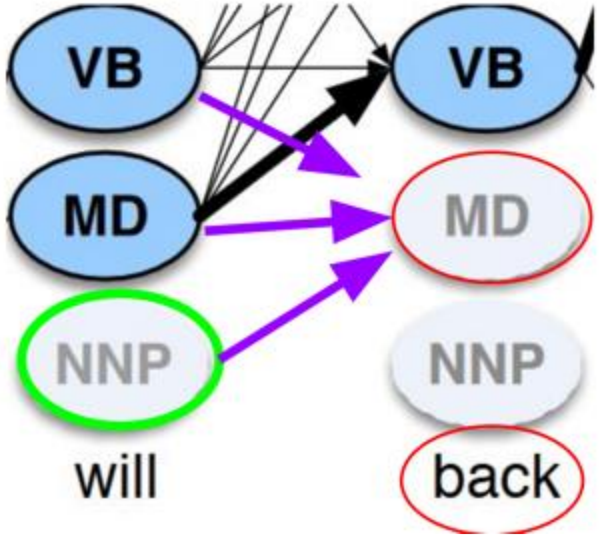


# The Viterbi Algorithm (ii)

- Each cell of the matrix is represented by **$V_t(j)$** (Viterbi value for t: column, j: row) having the probability that the HMM is in **state j** (present POS Tag) after seeing the **first t observations** (past words for which matrix (cell) values has been calculated) and passing through the most **probable state sequence (previous POS Tag)** $q_1.....q_{t-1}$
- Computed by recursively taking the most probable path that could lead us to this cell

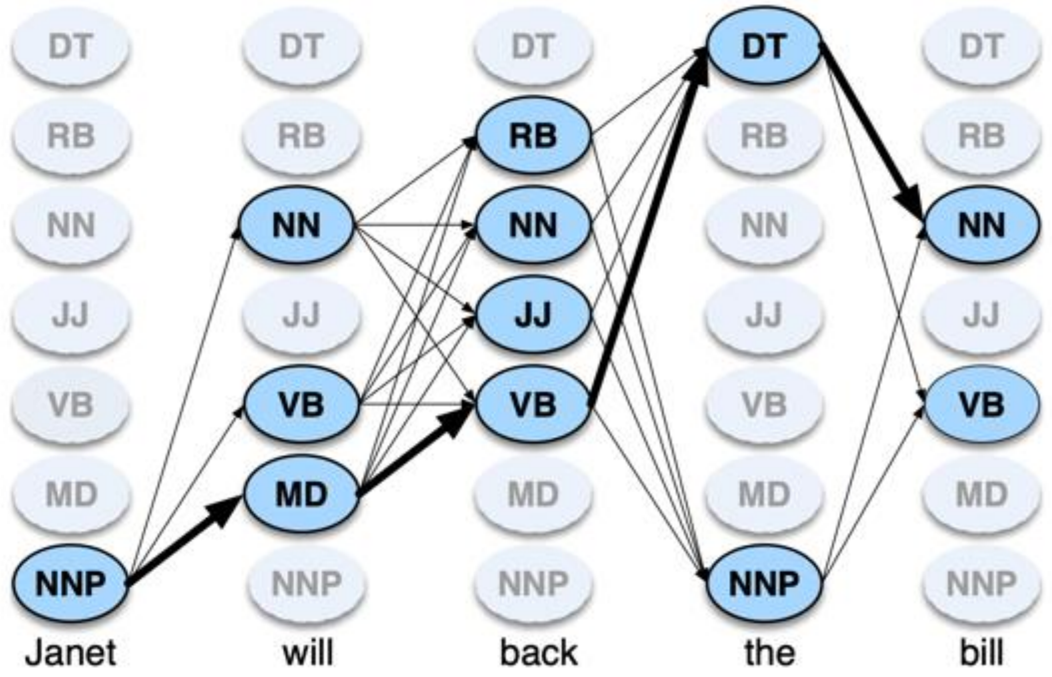
$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

| | |
|---|---|
| $v_{t-1}(i)$ | the **previous Viterbi path probability** from the previous time step |
| $a_{ij}$ | the **transition probability** from previous state $q_i$ to current state $q_j$ |
| $b_j(o_t)$ | the **state observation likelihood** of the observation symbol $o_t$ given the current state $j$ |



# The Viterbi Algorithm (iii)

- Each cell of the matrix is represented by **$V_t(j)$** (Viterbi value for t: column, j: row) having the probability that the HMM is in **state j** (present POS Tag) after seeing the **first t observations** (past words for which matrix (cell) values has been calculated) and passing through the most **probable state sequence (previous POS Tag)** $q_1.....q_{t-1}$



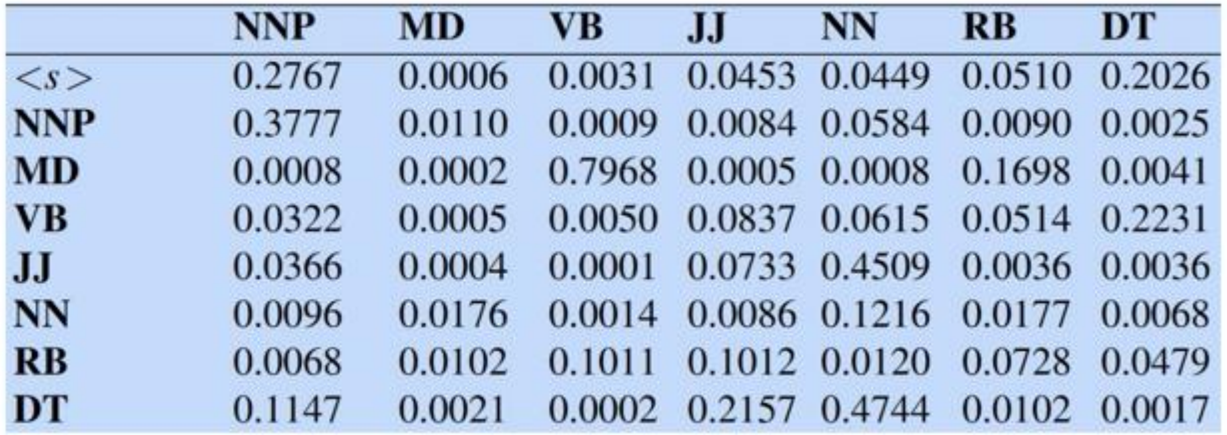
A sketch of the matrix for Janet will back the bill, showing the possible tags ($q_i$) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states

States (parts of speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as Janet) are greyed out

## Working Example (i)

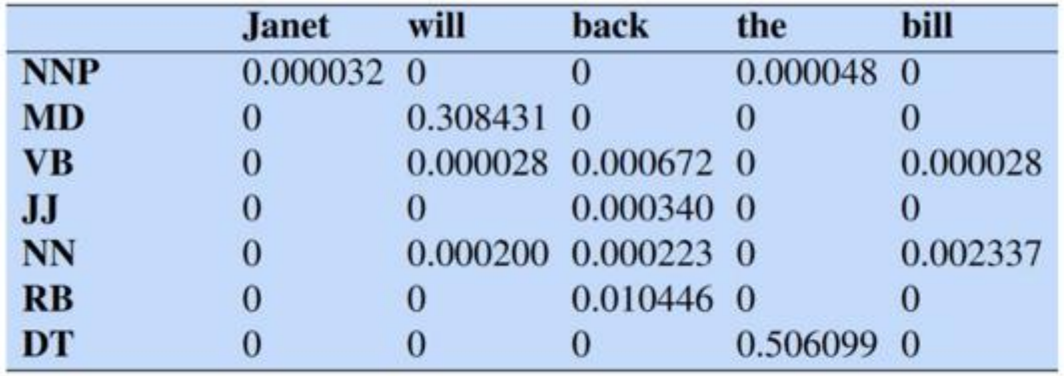
- Janet will back the bill → Janet/NNP will/MD back/VB the/DT bill/NN

The A transition probabilities P($t_i$|$t_{i-1}$) computed from the WSJ corpus without smoothing



| | NNP | MD | VB | JJ | NN | RB | DT |
|---|---|---|---|---|---|---|---|
| <s> | 0.2767 | 0.0006 | 0.0031 | 0.0453 | 0.0449 | 0.0510 | 0.2026 |
| NNP | 0.3777 | 0.0110 | 0.0009 | 0.0084 | 0.0584 | 0.0090 | 0.0025 |
| MD | 0.0008 | 0.0002 | 0.7968 | 0.0005 | 0.0008 | 0.1698 | 0.0041 |
| VB | 0.0322 | 0.0005 | 0.0050 | 0.0837 | 0.0615 | 0.0514 | 0.2231 |
| JJ | 0.0366 | 0.0004 | 0.0001 | 0.0733 | 0.4509 | 0.0036 | 0.0036 |
| NN | 0.0096 | 0.0176 | 0.0014 | 0.0086 | 0.1216 | 0.0177 | 0.0068 |
| RB | 0.0068 | 0.0102 | 0.1011 | 0.1012 | 0.0120 | 0.0728 | 0.0479 |
| DT | 0.1147 | 0.0021 | 0.0002 | 0.2157 | 0.4744 | 0.0102 | 0.0017 |

Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly



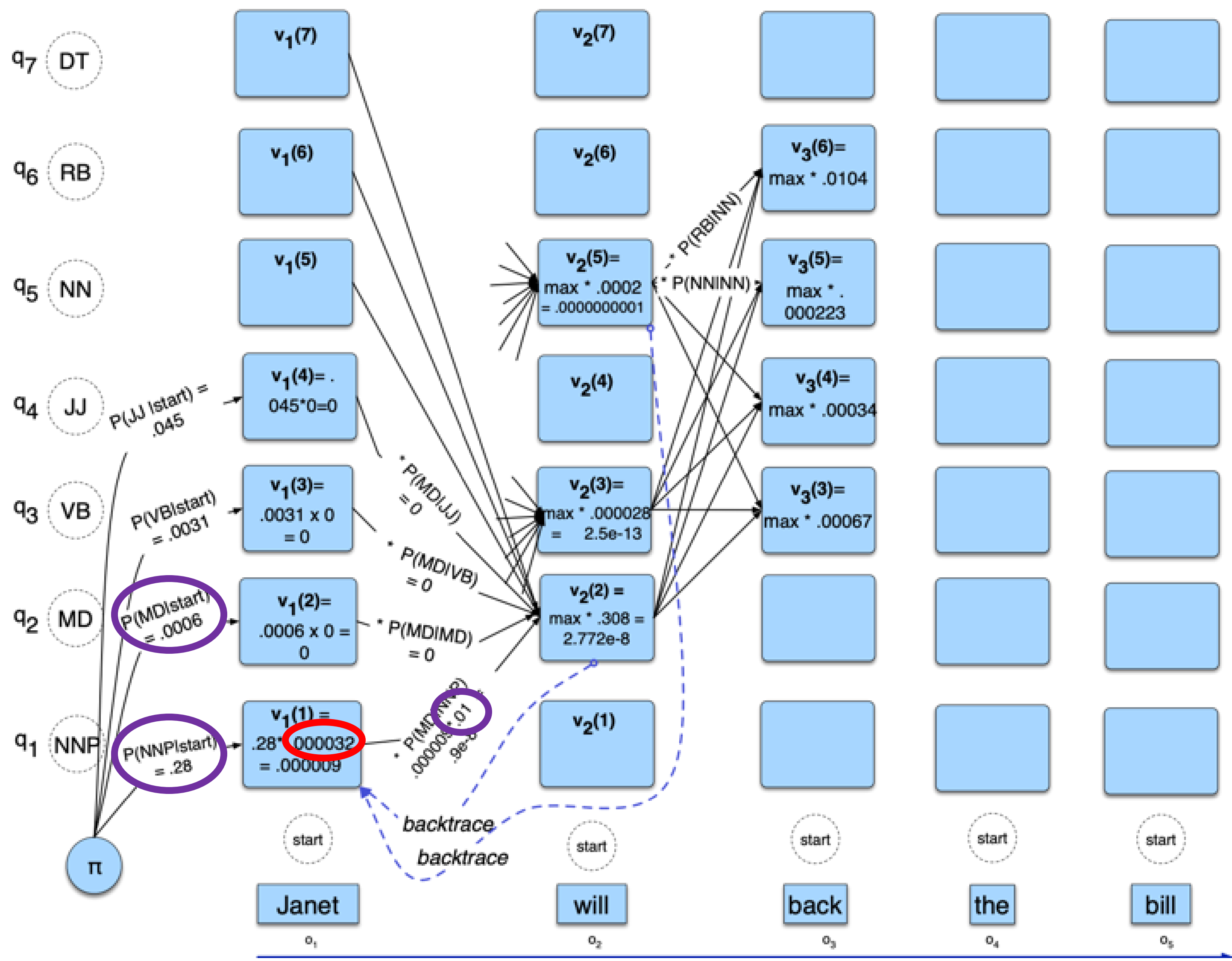
| | Janet | will | back | the | bill |
|---|---|---|---|---|---|
| NNP | 0.000032 | 0 | 0 | 0.000048 | 0 |
| MD | 0 | 0.308431 | 0 | 0 | 0 |
| VB | 0 | 0.000028 | 0.000672 | 0 | 0.000028 |
| JJ | 0 | 0 | 0.000340 | 0 | 0 |
| NN | 0 | 0.000200 | 0.000223 | 0 | 0.002337 |
| RB | 0 | 0 | 0.010446 | 0 | 0 |
| DT | 0 | 0 | 0 | 0.506099 | 0 |

# Working Example (ii)

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

|  | NNP | MD | VB | JJ | NN | RB | DT |
|---|---|---|---|---|---|---|---|
| <s> | 0.2767 | 0.0006 | 0.0031 | 0.0453 | 0.0449 | 0.0510 | 0.2026 |
| NNP | 0.3777 | 0.0110 | 0.0009 | 0.0084 | 0.0584 | 0.0090 | 0.0025 |
| MD | 0.0008 | 0.0002 | 0.7968 | 0.0005 | 0.0008 | 0.1698 | 0.0041 |
| VB | 0.0322 | 0.0005 | 0.0050 | 0.0837 | 0.0615 | 0.0514 | 0.2231 |
| JJ | 0.0366 | 0.0004 | 0.0001 | 0.0733 | 0.4509 | 0.0036 | 0.0036 |
| NN | 0.0096 | 0.0176 | 0.0014 | 0.0086 | 0.1216 | 0.0177 | 0.0068 |
| RB | 0.0068 | 0.0102 | 0.1011 | 0.1012 | 0.0120 | 0.0728 | 0.0479 |
| DT | 0.1147 | 0.0021 | 0.0002 | 0.2157 | 0.4744 | 0.0102 | 0.0017 |

|  | Janet | will | back | the | bill |
|---|---|---|---|---|---|
| NNP | 0.000032 | 0 | 0 | 0.000048 | 0 |
| MD | 0 | 0.308431 | 0 | 0 | 0 |
| VB | 0 | 0.000028 | 0.000672 | 0 | 0.000028 |
| JJ | 0 | 0 | 0.000340 | 0 | 0 |
| NN | 0 | 0.000200 | 0.000223 | 0 | 0.002337 |
| RB | 0 | 0 | 0.010446 | 0 | 0 |
| DT | 0 | 0 | 0 | 0.506099 | 0 |



## Evaluation Metrics for Named Entity Recognition (NER)

- NER Evaluation Basics:
  - Unlike POS tagging, evaluated on accuracy, NER uses recall, precision, and F1 score.
  - Recall measures correctly identified entities against all actual entities.
  - Precision counts correct labels against all labeling attempts.
  - The F1 score provides a balance between precision and recall, serving as a single metric for accuracy.
- Challenges in NER:
  - NER systems treat entities as single units for evaluation, leading to challenges not seen in POS tagging.
  - The system's ability to correctly identify entire entities, such as 'Jane Villanueva', impacts evaluation outcomes.
  - Mismatches in entity recognition across training and test data can skew results.