6CS012 – Artificial Intelligence and Machine Learning.

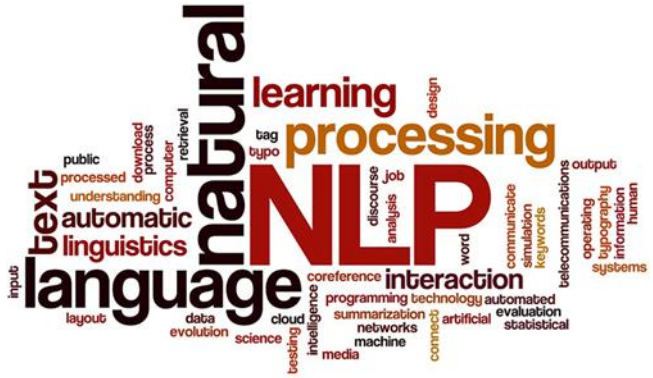
Lecture – 08

Introduction to Natural Language Processing.

Getting Started with

Text Data Representations.

Siman Giri {Module Leader – 6CS012}



What we built?

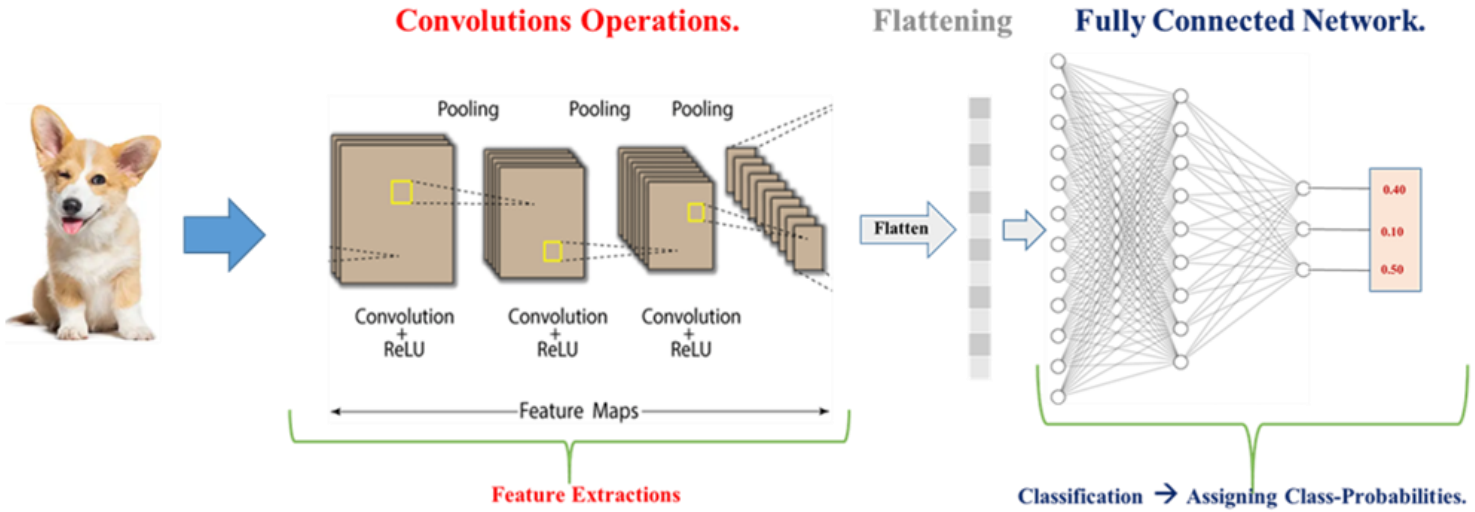
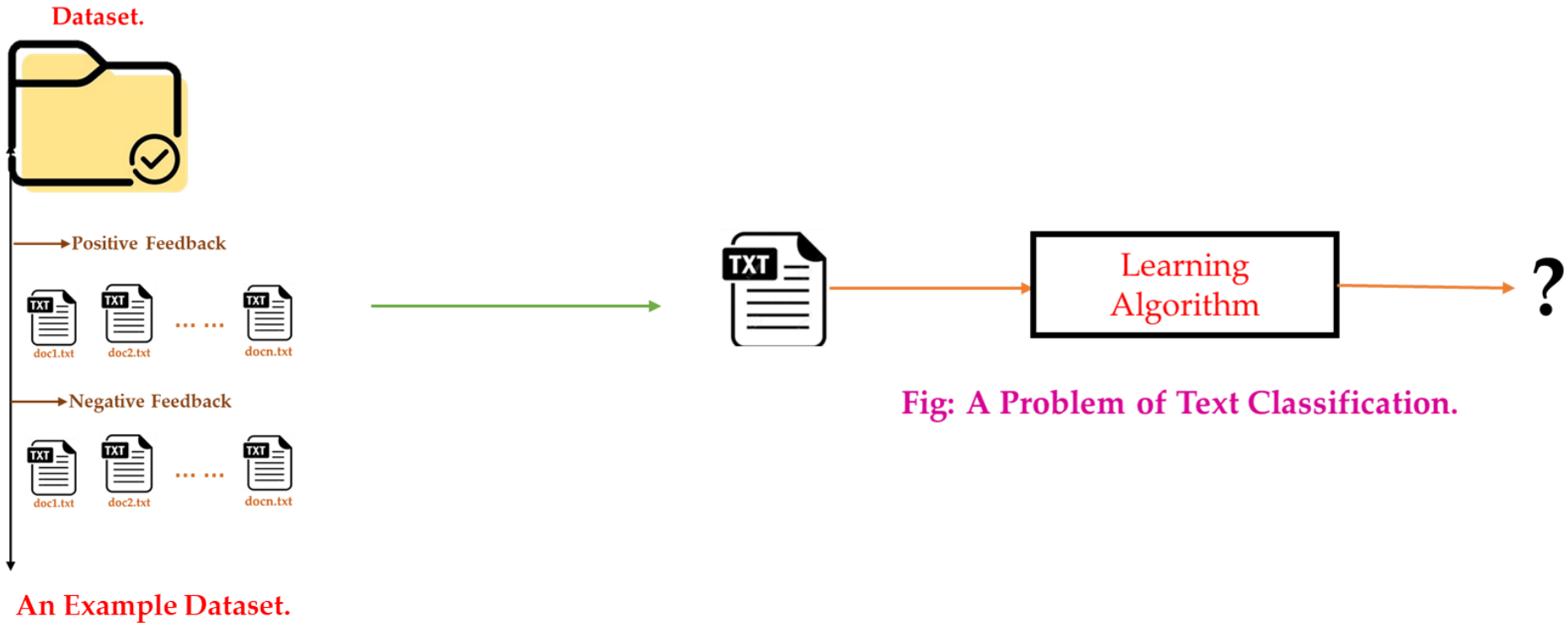


Fig: An End-to-End Model for Image Classification.

- Our Observation:
  - CNNs were designed specifically for image style inputs.
    - It has fixed **dimension – height, width and channel**.
  - By design CNN are not suitable for **inputs of varying sequences** for example **Text Data**.

The New Challenge!!!



An Example Dataset.

Fig: A Problem of Text Classification.

The New Challenge!!!

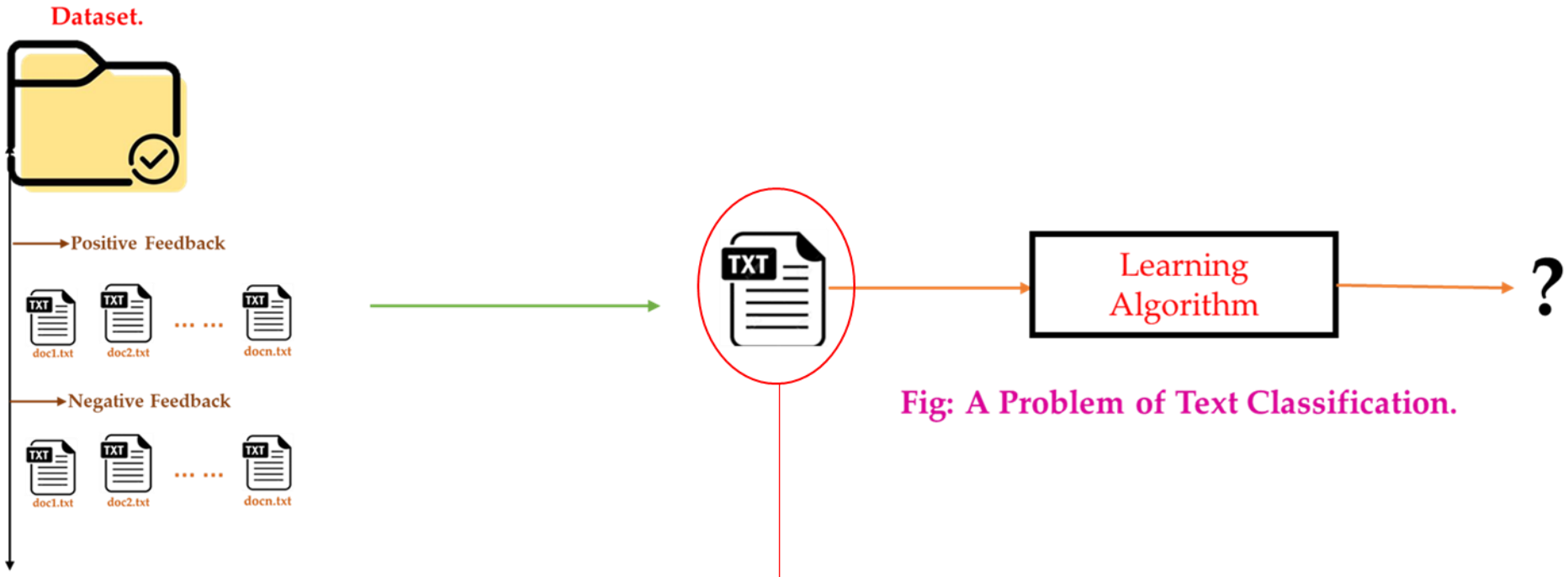
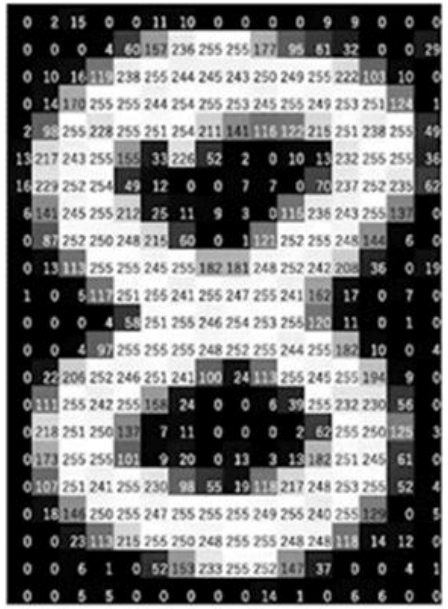


Fig: A Problem of Text Classification.

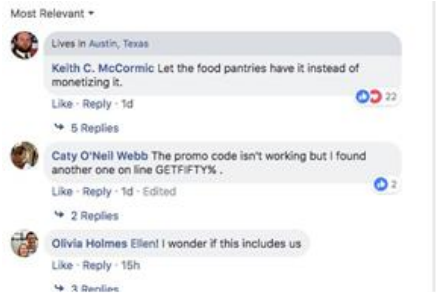
- First Let's Answer:
  - How Does Computer Understand Text?

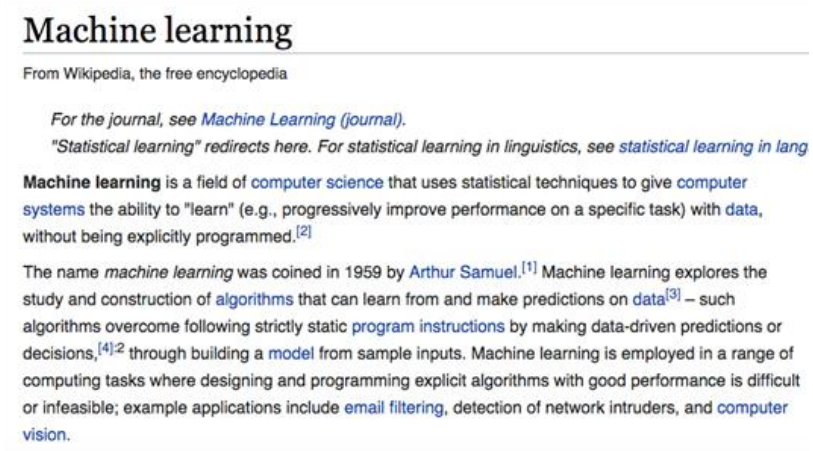
How Does Computer Understand Text?

- For Image:
  - Image are read by computer as a **matrix of numbers**
  - Does not **required any steps** to translate it **to matrix(vector) form**.



- For Text:
  - **How to describe text to a computer?**
    - Transform Text to its Numeric representations – How?





Machine learning

From Wikipedia, the free encyclopedia

For the journal, see Machine Learning (journal).
"Statistical learning" redirects here. For statistical learning in linguistics, see statistical learning in lang

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) with data, without being explicitly programmed.[2]

The name *machine learning* was coined in 1959 by Arthur Samuel.[1] Machine learning explores the study and construction of algorithms that can learn from and make predictions on data[3] – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions,[4]:2 through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders, and computer vision.

Learning Objective,

- Any Task in Natural Language Processing has two part:
  - **Part A – Text representations.**
  - **Part B – Build a Model to Complete the Task.**
- Focus for this week:
  - We will explore various techniques to represent raw text as numerical data suitable for **machine learning models.**
  - Then, **we will use these representations to build a simple Text Classification application**.
  - For modelling, we will use the classic and effective ML method for text-based tasks.
- **Let's get started with Part A.**

1.Text Representation for Deep Learning.

{"How to make Neural Network read and understand textual data?"}

1.1 Natural Language Processing: Introduction.

- **Natural language** is one of the **most complex tools** used by humans for a wide rage of reasons, for instance,
  - to communicate with others, to express thoughts or feelings, and ideas to ask questions, or to give instructions.
- Therefore, it is **essential for computers (intelligence system)** to possess the ability
  - to use the same tool in order to effectively interact with humans.
- The **field** of **Natural Language Processing** is a field of research in computer science including AI and Deep Learning
  - concerned with giving computers the ability to understand text and spoken words.
- Any NLP application consists of two tasks:
  - **Natural Language Understanding:**
    - **NLU** deals with understanding the meaning of human language, usually expressed as a piece of text.
  - **Natural Language Generations:**
    - The goal is for a computer to generate text, or in other words to talk to humans through natural language
      - Either to verbalize and idea or meaning, or to provide a response.
        - **NLG gives answer**.

1.2 NLP: Example.

- For instance: **You ask a question to any chatbots:**
  - **"do penguins fly?"**
    - Before chatbot can answer the question:
      - the very first step for chatbot is **to understand the question**:
        - which in turn depends on **the meaning of penguin and fly**, and their **composition**.
      - **Task of NLU.**
    - To answer the question, it must be able to **generate a response** i.e.
      - **Yes** or **No** – such that we can understand.
      - **Task of NLG.**

1.3 Some Common Application of NLP.

- Majority of the NLP tasks break down human text/audio in ways that help computer understand and make sense out of it.
  - **Language Translation or Machine Translation:**
    - Translating from one language to another.
    - Google translate.
  - **Text Classification or Sentiment analysis:**
    - attempts to extract subjective qualities—attitudes, emotions, sarcasm, confusion, suspicion—from text.
  - **Speech recognition:**
    - Converts voice data into text.
    - Required for any application that follows voice commands or answers spoken question.
  - **Text summarization:**
    - Summarizing the text from any literature.

Plan for the Week

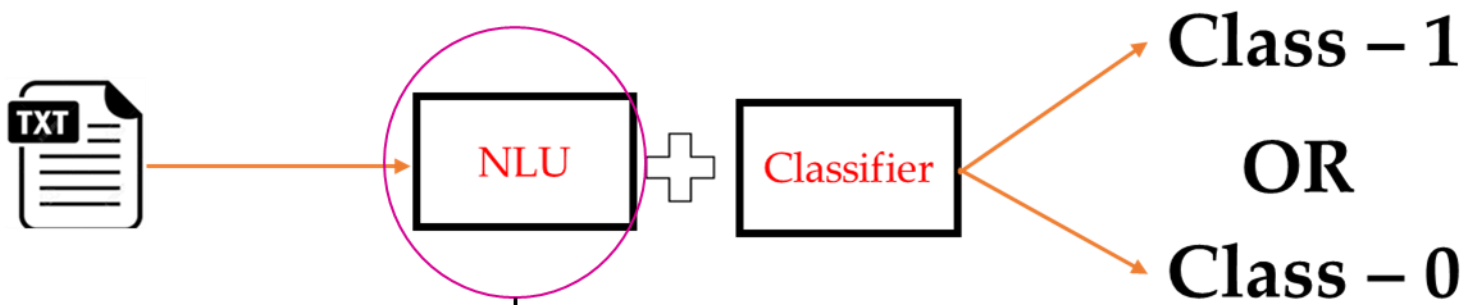


Fig: A Problem of Text Classification.

Text Pre – Processing or Cleaning.
{Focus of Our Tutorial.}

Text Pre – Representations.
{Focus of Our Lecture.}

Combine both to build a Classifier.
{Plan for Workshop.}

1.4. Representing Text as a Symbol.

- **Character Encoding Scheme:**
  - Character encoding scheme like ASCII and Unicode represent input text in the array of 0 and 1.
    - For example:
      - **ASCII encoding for "desk": 01100100 01100101 01110011 01101011.**
      - **ASCII encoding for "table": 01110100 01100001 01100010 01101100 01100101.**
        - {0 and 1 are the bit value}
  - **Why is this not a good idea?**

1.4.1 Represent Text: ASCII or Unicode.

- Why is this not a good idea?
- **Challenge – 1: The representation is character – wise:**
  - Therefore, the **size of the representation** depends on the **length of the words**
    - (**number of characters they have**).

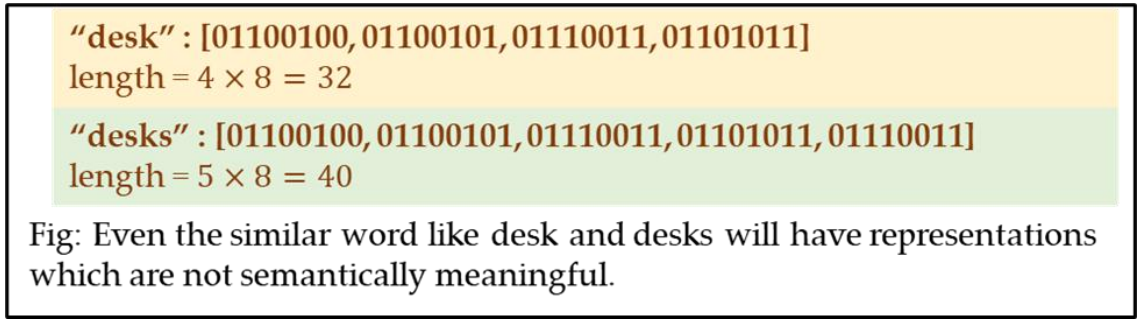


Fig: Even the similar word like desk and desks will have representations which are not semantically meaningful.

  - The **variable size** is an **unwanted property** which **further complicates** the **comparison of representations of different words**.
  - In fact, it is **not straightforward** to integrate these variable-sized representations into machine learning models, which generally "**understand" feature-based representations**.

1.4.2 Represent Text: ASCII or Unicode.

- Why is this not a good idea?
- **Challenge – 2:** The representation cannot incorporate **semantic information** of **words:**
  - **Idea of semantic representation** – words with similar meanings should have similar representations.
  - "desk" : [01100100, 01100101, 01110011, 01101011]
    length = 4 × 8 = 32 → {Modern ASCII uses 8-bit character encoding}
  - "table" : [01100100, 01100101, 01110011, 01101011]
    length = 5 × 8 = 40 → {Modern ASCII uses 8-bit character encoding}
    adding one character increases the length by 8.
  - For example, the **semantically similar words** "**table**" and "**desk**" (or even synonymous words such as "**noon**" and "**midday**") will have totally **different representations.**
  - We are ideally looking for a **representation** that can **encode semantics of words**.
- To summarize:
  - Variable length – inefficient for mathematical operations – we want fixed size representations.
  - Computing Similarity – these fixed representations must also be semantically similar.

1.5 Solution: Vector Representation in Fixed Dimensional Space.

- One way to address the limitations of symbolic ASCII is to represent each word as vector in a fixed dimensional vector space.
- For example: if we define a space of **dimension d = 4**, then:
  - "desk" and "tables" can be represented as a **vector of same dimension**:
    - **"desk"** $\rightarrow \mathbf{v} = [\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_4}] \in \mathbb{R}^4$ & **"tables"** $\rightarrow \mathbf{v} = [\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_4}] \in \mathbb{R}^4$
- Regardless of the original word length (**number of characters**), the word is now represented by:
  - the **fixed size vectors in vector space with dimension** $\mathbb{R}^4$, which means
    - every word has length of 4 and
    - every component can be vector of any real number for $-\infty$ **to** $+\infty$**.**
- Let's called this vector space a **Word Embedding Vector space of dimension d**,
  - Word embedding because we embedded word in its **real valued vector form**.
- Representing in vector space allows:
  - To map words to fixed size vector.
  - We could perform various operations like dot product or cosine similarity.
- This now have challenges:
  - **How do we determine the optimal dimension (d) of the embedding space?**
  - **How do we find the actual values of the components** $[\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v_d}] \in \mathbb{R}^{\mathbf{d}}$**?**

Terminology Alert!!

- **Text data:**
  - Documents.
  - Data that is in the form of text file.

- **Corpus:**
  - Collection of Documents.

- **Vocabulary**
  - Collection of all the **unique terms(words)** in the corpus.
  - Please Note: This Lecture focus on text representations and the process of creating vocabulary will be discuss on your tutorial session.

1.5.1 Example of Vector Space Model – One Hot Encoding.

- One Hot Encoding can be thought as a primitive vector space model, for instance:
  - Let's Consider a language with **vocabulary of 5 words**, there one hot encoding representation will look like:
    - **vocabulary = ["desk", "desks", "table", "plate", "lion"]**
  - We need to consider two things: dimension d and vector element v_1 etc.
    - Dimension **d ⇒ length of vocabulary = 5**
    - **Each values v_1 are binary value True for the term in that row false for rest.**

	desk	desks	table	plate	lion
desk	1	0	0	0	0
desks	0	1	0	0	0
table	0	0	1	0	0
plate	0	0	0	1	0
lion	0	0	0	0	1

- On the contrary to the Character encoding schemes one hot encoding will have the fixed length of the representations i.e. proportional to length of vocabulary.
  - In our example each **word is represented as a 5-dimension vector like representation** in which all the values are zero except at its corresponding column.
- **One – hot representation** addresses the **first limitation** discussed in earlier slide.
  - However, it still suffers from the second limitation each word is assigned a different representation and there is no notion of "similarity" between them.

1.5.2 One Hot Encoding – Limitations.

- **No semantic relationship:**
  - One-hot-encoded representations depends on where you put the word in list.
    - Using this representation, it is not possible to encode the conceptual similarity between "noon" and "midday".
  - Even worse, the two similar looking words such "desk" and "desks" (which would have similar string-based representations) are assigned completely different one-hot vectors.
- **Dimension Vs. Length of Vocabulary (High – dimensional):**
  - the dimensionality of one-hot representations grows with the number of words in the vocabulary.
  - In a typical vocabulary, we should expect hundreds of thousands of words.
- **Sparse Representations:**
  - The vector representations are sparce because it has mostly 0s except for single position that corresponds to the word.
  - It would not be **memory efficient** for large vocabulary sizes.

2. Transition from One – Hot to Classical Vector Space Model.

{ Classical Text Representation Methods – Before Neural Embeddings.}

2.1 Classical Vector Space Model.

- The vector space model, introduced by Gerard Salton in the 1970s, was initially designed for information retrieval problem where:
  - Document and queries were represented as vector and similar document were retrieved by computing the similarity between query and document vector.
  - To represent it was suggested to use classical techniques like co – occurrence matrix and TF – IDF approaches.
- From machine learning perspective, we will borrow the idea of vector representation and try to represent our text data or vocabulary as vector using co – occurrence count matrix and TF – IDF approach.

Gerard Salton (1927 – 1995)

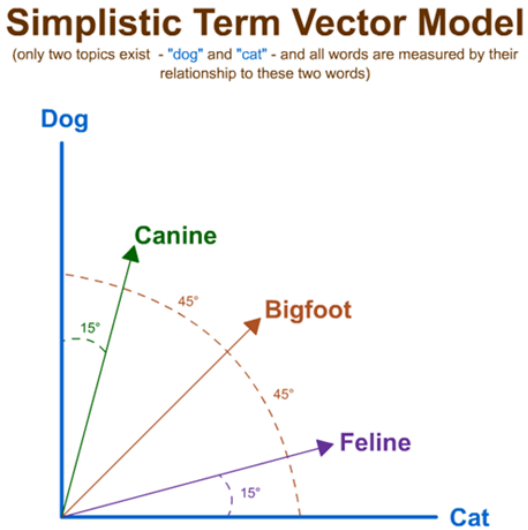


Image from internet: subject to copyright.

2.2 Vector Space Model and Similarity.

- **Notion of Similarity in Vector Space Model:**
  - For instance, assume following is the corpus:
  - Extracting the Vocabulary:
    - **vocab = unique terms in corpus → ["and", "chair", "desk", "lamp", "table"]**
  - We represent each unique word also called term with count matrix as shown below:

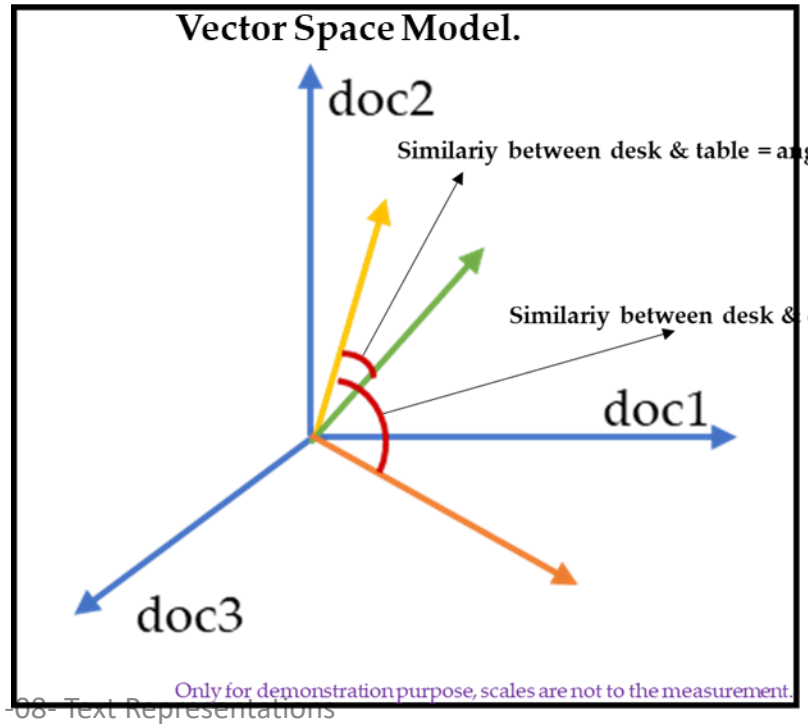
Document ID	Text
Doc 1	"desk and table"
Doc 2	"table and chair"
Doc 3	"chair and lamp"

A Corpus.

Count Matrix.

Term	Doc 1	Doc 2	Doc 3
and	1	1	1
chair	0	1	1
desk	1	0	0
lamp	0	0	1
table	1	1	0

Each cell is the raw count of the term in the document in all corpus.



Large Angle = Less Similar.
Angle could be computed as:

$$\text{Cos}\theta = \frac{< \text{desk} \cdot \text{table} >}{\|\text{desk}\| \cdot \|\text{table}\|}$$

Fig: Notion of Similarity in Vector Space Model.

2.3 Term Frequency

- Idea **a term is more important** if it occurs **more frequently** in a **document.**
  - raw – term frequency: Let $\mathbf{f_{t,d}}$ be the frequency count of term t in doc d.
    - $\mathbf{raw - tf: tf_{td} = f_{t,d}}$ **{frequency of term in document}**
- Limitations of raw – term frequency:
  - Repeated occurrences are less informative than the first occurrence.
  - Relevance does not increase proportionally with number of term occurrence.
- To overcome, **we add a log normalization** thus, **tf – weighting** is actually written as:
  - **tf – weight:**
    - $\mathbf{W_{tf} = \begin{cases} 1 + \log(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{Otherwise} \end{cases}}$

2.3.1 Inverse Document Frequency.

- **Idea:**
  - **Document Frequency**: a term is more discriminative if it occurs only in fewer documents.
- **Actual Implementations:**
  - **IDF – Inverse Document Frequency**: assign a higher weights to the rare terms.
  - Formula:
    - $\mathbf{idf_t = \log\left(\frac{N}{df_t}\right)}$
    - Here:
      - **N** → total number of docs is collection.
      - $\mathbf{df_t}$ → Number of docs containing term t (**doc frequency**).

2.3.2 TF – IDF weights.

- **Combining tf and idf:**
  - **Common in doc → high tf → high weight.**
    - If a term appears frequently in a document (high TF), it suggests that the term is **important** in that specific document.
  - **Rare in Corpus → high idf → high weight.**
    - If a term appears in only a few documents (high IDF), it suggests that the term is **rare** and thus more **informative**.
    - Common terms (like "the", "and", "is") have low IDF, so they don't contribute much to distinguishing documents.
  - **TF-IDF** combines these to reflect both the **local importance** and **global rarity** of a term in a document, making it a useful measure for distinguishing terms that truly capture the **essence** of a document in the corpus.
    - **tf – idf score is given as:**
      - $\mathbf{W_{tf-idf} = tf_{t,d} \times idf_t}$
      - Proposed by G. Salton et. al. 1983 – must probably the most well-known document representation schema.
- **Example Computations:**
  - **For the following corpus, compute the TF – IDF weights for all the unique terms (vocabulary):**
  - **Data – Documents:**

Document ID	Text
Doc 1	"desk and table"
Doc 2	"table and chair"
Doc 3	"chair and lamp"

A Corpus.

Example Computations: TF and IDF weights.

Compute Term Frequency (TF)

- TF for a term t in document d is:
  - $\mathbf{TF_{t,d} = \frac{Number\ of\ times\ t\ appers\ in\ d}{Total\ numbers\ of\ terms\ in\ d}}$

Term	TF – DOC 1	TF – DOC 2	TF – DOC 3
and	0.33	0.33	0.33
chair	0	0.33	0.33
desk	0.33	0	0
lamp	0	0	0.33
table	0.33	0.33	0

- A sample computations:
  - $TF_{and} = \frac{\text{Number of times "and" appears in Doc 1}}{\text{Total number of terms in Doc 1}} = \frac{1}{3} \approx 0.33$

Inverse Document Frequency (IDF)

- IDF for term t is:
  - $\mathbf{IDF_t = \log_{10}\left(\frac{N}{df_t}\right)}$
    - **Here N = 3**

Term	$\mathbf{df_t}$	$\mathbf{idf_t}$
and	3	$\log\left(\frac{3}{3}\right) = \log(1) = 0.00$
chair	2	$\log\left(\frac{3}{2}\right) \approx 0.176$
desk	1	$\log\left(\frac{3}{1}\right) \approx 0.477$
lamp	1	$\log\left(\frac{3}{1}\right) \approx 0.477$
table	2	$\log\left(\frac{3}{2}\right) \approx 0.176$

TF – IDF: Example Computations.

- tf – idf score is given as:
  - $W_{tf-idf} = tf_{t,d} \times idf_t$

Term	DOC 1 – TF × IDF	DOC 2 – TF × IDF	DOC 3 – TF × IDF
and	0.33 × 0.00 = 0.00	0.33 × 0.00 = 0.00	0.33 × 0.00 = 0.00
chair	0 × 0.176 = 0.00	0.33 × 0.176 = 0.058	0.33 × 0.176 = 0.058
desk	0.33 × 0.477 = 0.158	0 × 0.477 = 0.00	0 × 0.477 = 0.00
lamp	0 × 0.477 = 0.00	0 × 0.477 = 0.00	0.33 × 0.477 = 0.158
table	0.33 × 0.176 = 0.058	0.33 × 0.176 = 0.058	0 × 0.176 = 0.00

- Representation for "desk" is:
  - [0.158, 0.00, 0.00]

2.4 Limitations of TF – IDF Representations.

- **No Sematic Understanding:**
  - TF – IDF only counts word occurrences – it treats each word as independent and unique, ignoring word meanings.
  - E.g. "car" and "automobile" have very similar meanings, but TF – IDF vectors for them will be completely different.
- **Sparsity and High Dimensionality:**
  - The vectors are as long as the corpus size → making TF – IDF weight Matrix huge and sparse.
  - Difficult to scale for large corpora.
- **Fixed Vocabulary, No Generalization:**
  - Can not handle out of vocabulary words.
  - Adding new words requires recomputing the whole TF – IDF matrix.
- **Ignores Word Order and Context:**
  - TF – IDF works at the bag of words level:
    - Losses syntax and word position information.
      - "not good" and "good" are treated similarly if "good" dominates the TF – IDF weight.
- **Hard to Compare Across Corpora:**
  - IDF is corpus dependent:
    - TF – IDF values for the same term can vary between corpora.

3. From Symbolic to Semantic: Introduction to Language Model.

{Putting Uncertainties (Probability) in term's Vector Representations.}

3.1 Word Embeddings: Inspiration.

- Q: How to insert sematic understanding in Vector Representations?
  - Inspiration: Distributional Semantics.
    - "The distributional hypothesis says that the **meaning of a word is derived from the context in which it is used**, and **words with similar meaning are used in similar contexts**."
      - Harris in 1954 and Firth in 1957
- **Experiment 1: based on "tezguino" example by Lin in 1998.**

Do you know what the word tezgüino means ?

(We hope you do not)

3.1.1 Word Embeddings: Inspiration.

- **Experiment 1: based on "tezguino" example by Lin in 1998.**

Do you know what the word tezgüino means ?

(We hope you do not)

- Now Let's look into some context:

Now look how this word is used in different contexts:

A bottle of tezgüino is on the table.
Everyone likes tezgüino.
Tezgüino makes you drunk.
We make tezgüino out of corn.

Can you understand what tezgüino means ?

3.1.2 Word Embeddings: Inspiration.

- **Experiment 1: based on "tezguino" example by Lin in 1998.**

Do you know what the word tezgüino means ?

(We hope you do not)

How did you do this?

- **Now Let's look into some context:**

Now look how this word is used in different contexts:

A bottle of tezgüino is on the table.
Everyone likes tezgüino.
Tezgüino makes you drunk.
We make tezgüino out of corn.

Can you understand what tezgüino means ?

3.1.3 Word Embeddings: Inspiration.

- **Experiment 1: based on "tezguino" example by Lin in 1998.**

Do you know what the word tezgüino means ?

(We hope you do not)

How did you do this?

- Now Let's look into some context:

Now look how this word is used in different contexts:

A bottle of tezgüino is on the table.
Everyone likes tezgüino.
Tezgüino makes you drunk.
We make tezgüino out of corn.

Can you understand what tezgüino means ?

Observation: context makes it easier to understand a word's meaning.

3.1.4 Word Embeddings: Inspiration.

- **Experiment 2: What other words could fit into these context?**

(1) A bottle of _____ is on the table.

(2) Everyone likes _____ .

(3) _____ makes you drunk.

(4) We make _____ out of corn.

3.2 Word Embeddings: Inspiration.

- **Experiment 2: What other words could fit into these context?**

(1) A bottle of _____ is on the table.

(2) Everyone likes _____ .

(3) _____ makes you drunk.

(4) We make _____ out of corn.

	(1)	(2)	(3)	(4)
tezgüino	1	1	1	1
loud	0	0	0	0
motor oil	1	0	0	1
tortillas	0	1	0	1
wine	1	1	1	0

Contexts

1 if a word appear in the context;
0 otherwise

3.2.1 Word Embeddings: Inspiration.

- **Experiment 2: What other words could fit into these context?**

(1) A bottle of _____ is on the table.

(2) Everyone likes _____ .

(3) _____ makes you drunk.

(4) We make _____ out of corn.

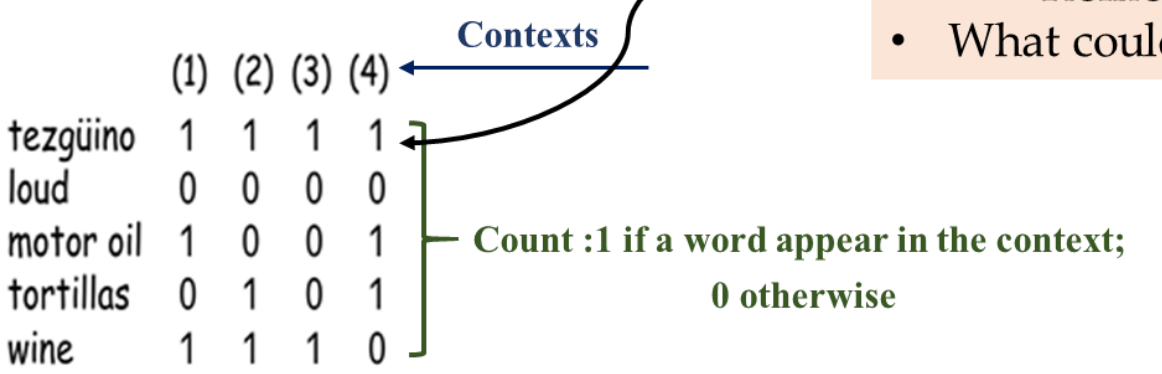
	Contexts → (1)	(2)	(3)	(4)
tezgüino	1	1	1	1
loud	0	0	0	0
motor oil	1	0	0	1
tortillas	0	1	0	1
wine	1	1	1	0

Count :1 if a word appear in the context; 0 otherwise

Observations: Words with similar row {they appear in similar context} they may have similar meanings.

3.2.2 Word Embeddings: Summary.

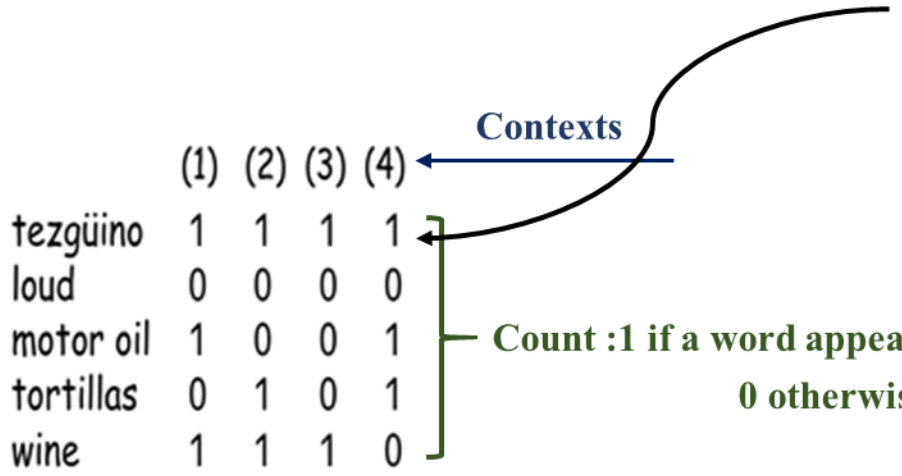
- Conclusion from "tezguino" experiment:
  - Observation 1: **context makes it easier to understand a word's meaning.**
  - Observation 2: **words with similar row {they appear in similar context} they may have similar meanings.**
- Problem to solve:



- How can we fill this matrix?
  - Is Count a good idea?
  - **Remember the limitations of TF – IDF, so no.**
- What could be the solutions?

3.3 Word Embeddings: Summary.

- Conclusion from "tezguino" experiment:
  - Observation 1: **context makes it easier to understand a word's meaning.**
  - Observation 2: **words with similar row {they appear in similar context} they may have similar meanings.**
- Problem to solve:



- How can we fill this matrix?
  - Is Count a good idea?
  - **Remember the limitations of TF – IDF, so no.**
- What could be the solutions?

How can we fill this matrix?
Idea:
Instead of counting the appearance can we estimate the probability of the next word.
Towards Language Model

3.4 Word Embeddings: Language Model.

- Provides a principled way to quantify the (uncertainties) probabilities associated with natural language.
- Allows us to answer questions like:
- Given that we see "John" and "feels", how likely will we see "happy" as opposed to "habit" as the next word?
- Implementations:
  - **N – gram language Models**
    - In general:

$$\mathbf{p(w_1 w_2 \dots w_n) = p(w_1) p(w_2|w_1) \dots p(w_n|w_1 \dots w_{n-1})}$$

      - N-gram: conditioned only on the past N-1 words.

3.5 Estimating Probabilities in N-Gran LM.

- {Approach1: $P_{\Omega} = \frac{\#\text{ Favourable Events}}{\#\text{ Total Events in }\Omega}$}
- **When N = 1 i.e. Unigram Model.**
  - For the unigram language model, we need to estimate
  - P(word) for every word in the text
    - **$P(\text{word}) = \frac{n_i}{N}$**
    - **n_i** is the number of occurrences of word in the collection
    - **N** is the total number of word occurrences (i.e., tokens) in the collection

- **{Approach2: $P(A|B) = \frac{P(A \cap B)}{P(B)}$ (Conditional)}**
- **When N = 2 i.e. Bigram Model**
  - For the bigram language model, we additionally need to estimate
    - P(word | previous word) **for every pair of words** that **appear one after another**
      - **$P(w|w_{i-1}) = \frac{n(w_{i-1}, w_i)}{n(w_{i-1})}$**
        {Approach1}
      - **$n(w_{i-1}, w_i)$** the number of occurrences of bigram **w_{i-1}, w_i** in the collection
      - **$n(w_{i-1})$** the number of occurrences of term **w_{i-1}** in the collection

3.6 Estimating Probabilities in N-Gram LM.

- Examples:
  - We are given a toy collection consisting of three documents
    - **d1 : "Frodo and Sam stabbed orcs"**
    - **d2 : "Sam chased the orc with the sword"**
    - **d3 : "Sam took the sword"**
  - **Estimating word probabilities for the unigram model:**

t_i	Frodo	Sam	orc	chased	sword	...
$P(t_i)$	1/16	3/16	2/16	1/16	2/16	...

  - Estimates : → **How many times** the word appears in the collection of documents?
    - Does not embed the semantics in a vector representations as context are not included in the estimation of probability.

3.6.1 Estimating Probabilities in N-Gram LM.

- Examples:
  - We are given a toy collection consisting of three documents
    - **d1 : "Frodo and Sam stabbed orcs"**
    - **d2 : "Sam chased the orc with the sword"**
    - **d3 : "Sam took the sword"**
- **Estimating the conditional probabilities for the bigram model:**

t_{i-1}, t_i	Frodo, chased	the, sword	the, orc	...
$P(t_i \mid t_{i-1})$	0	2/3	1/3	...

  - This is for N = 2, Estimates:→ How many times word have appeared in pair?
    - This approach tries to embed the context by considering neighboring words, this might get better result if we could consider a greater number of neighboring words i.e. increase N's values.
    - What will be the challenge if we increase N?

3.7 Challenges of N-gram LM.

- **Challenges:**
  - Language models have a major issue
    - The longer the phrase, the harder it is to estimate its true probability in language
      - E.g., **P("bilbo" | "frodo ran around house found ring") = ?**
    - Long phrases have very few appearances even in very large corpora
      - Impossible to compute reliable estimates of their conditional probabilities
      - This is why language models for N ≥ 3 are almost never used
      - They are **sparse**.
- Is there better way to estimate rather than traditional approach?
  - What about using neural network, they also estimate a probability(likelihood) in center?
    - **Towards Neural Embeddings.**

3.8 Neural Embeddings: Inutition.

- Fully Connected Neural Network computes the probability of the features belonging to certain class:

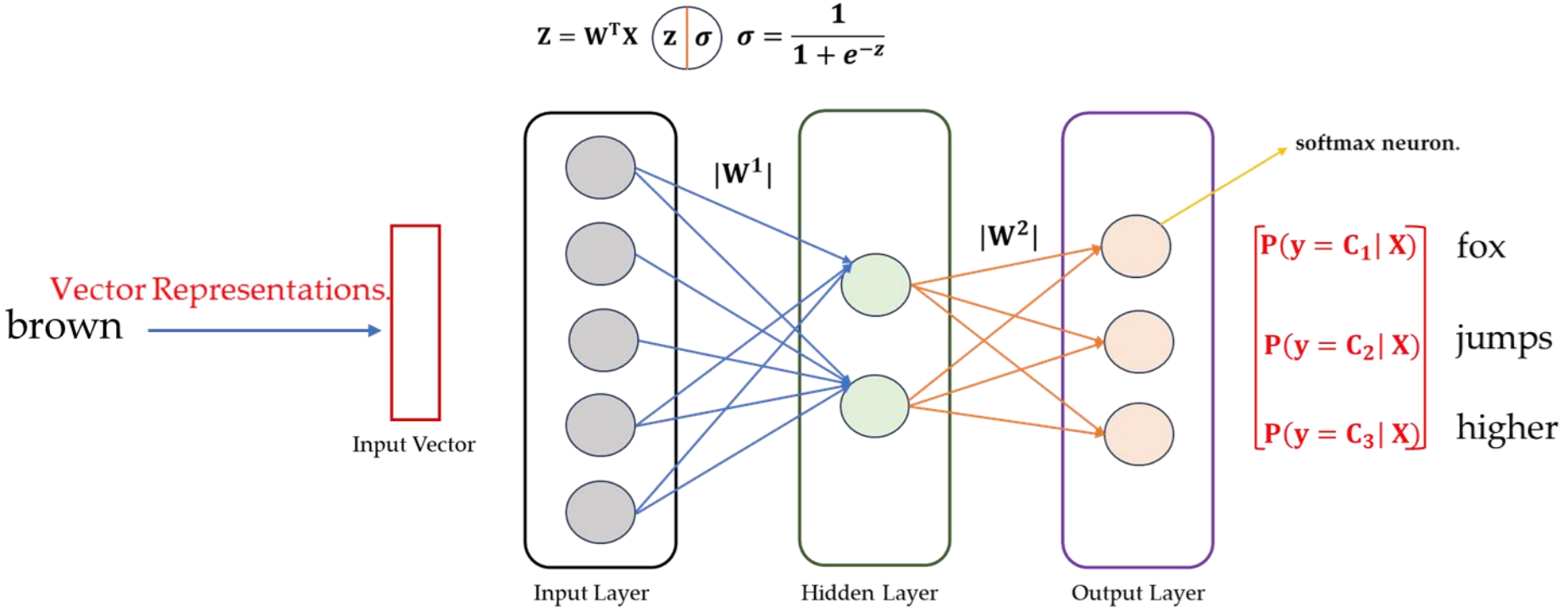


Fig: A Fully Connected Neural Network.

Can we use this to predict the probability of next word ?

4. Neural Word Embeddings.

{"word2vec"}

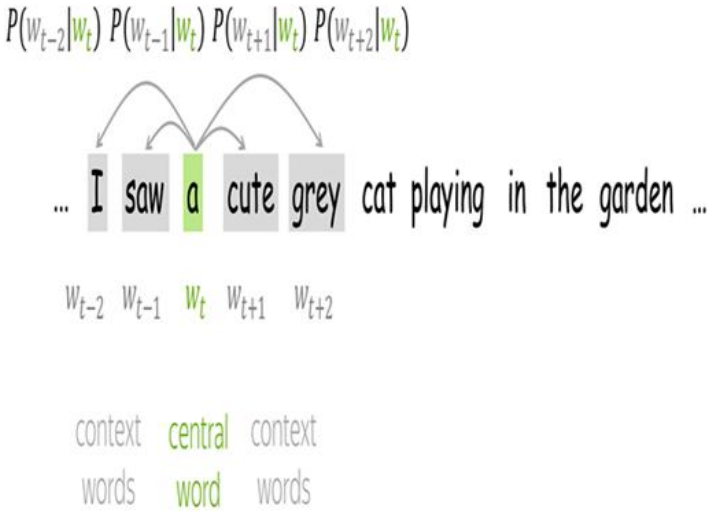
4.1 Neural Word Embeddings: Introduction.

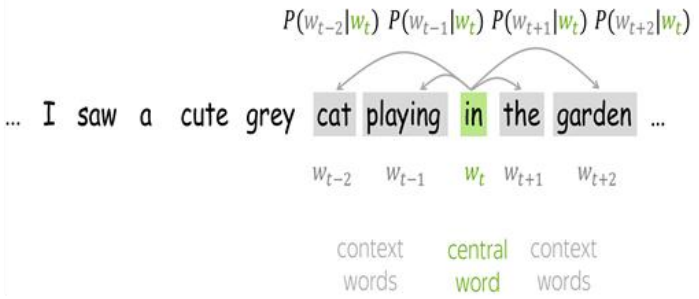
- **Word Embeddings:**
  - The process of **Creating a vector representation** of the **word**. {better if it could represent a semantics of the words}.
  - **How to insert semantics?**
    - It can be achieved if **word embeddings also learns a context of previous them** to **predict next word efficiently** i.e. N-gram language models may not work.
      - **Neural embedding may be the answer.**
- **Neural Word Embeddings:**
  - The idea of using **Neural network** to **learn the relationships** among **the word** is called neural (word) embeddings.
    - !Disclaimer: From this point forward the word embeddings mean Neural word embedding.
  - The idea was first presented by Bengio et. al in there 2003 paper "A Neural Probabilistic Language Model".
  - The idea was further expanded and improved by Mikolov et. al in 2013 in their paper "word2vec".
    - The term "wor2vec" has become so popular now it is used almost synonymously with embeddings or neural embeddings.

Mikolov.

4.2 "word2vec": Intuition.

- "word2vec" is an iterative method. Its main idea is as follows:
  - **take a huge text corpus;**
  - **go over the text with a sliding window, moving one word at a time.**
  - **At each step, there is a central word and context words (other words in this window);**
  - **for the central word, compute probabilities of context words;**
  - **adjust the vectors to increase these probabilities.**



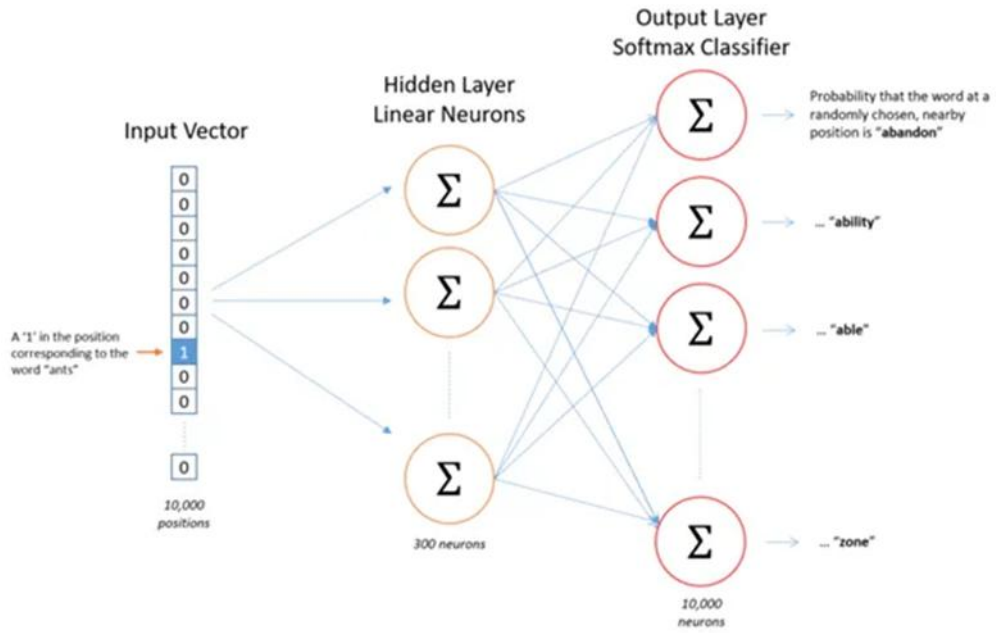


4.3 "word2vec": Introduction.

- Idea: Learn word Embeddings that know what are viable surrounding words.

1. ___ ___ ___ ___ berimbau ___ ___ ___ ___
2. ___ berimbau ___years to learn how to play the _____

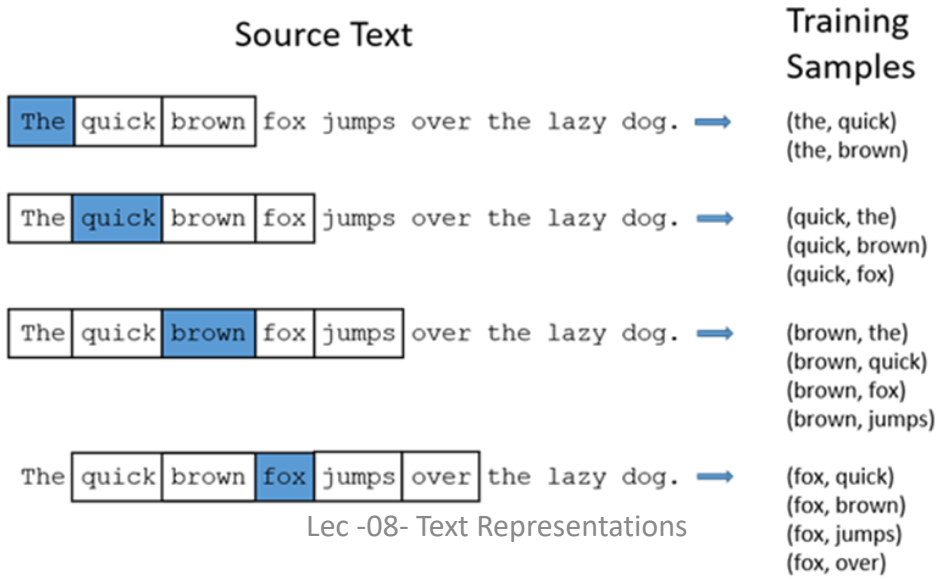
- Task: Given a word, Predict a Nearby word.
  - Approach: Build a Neural Network.



How?

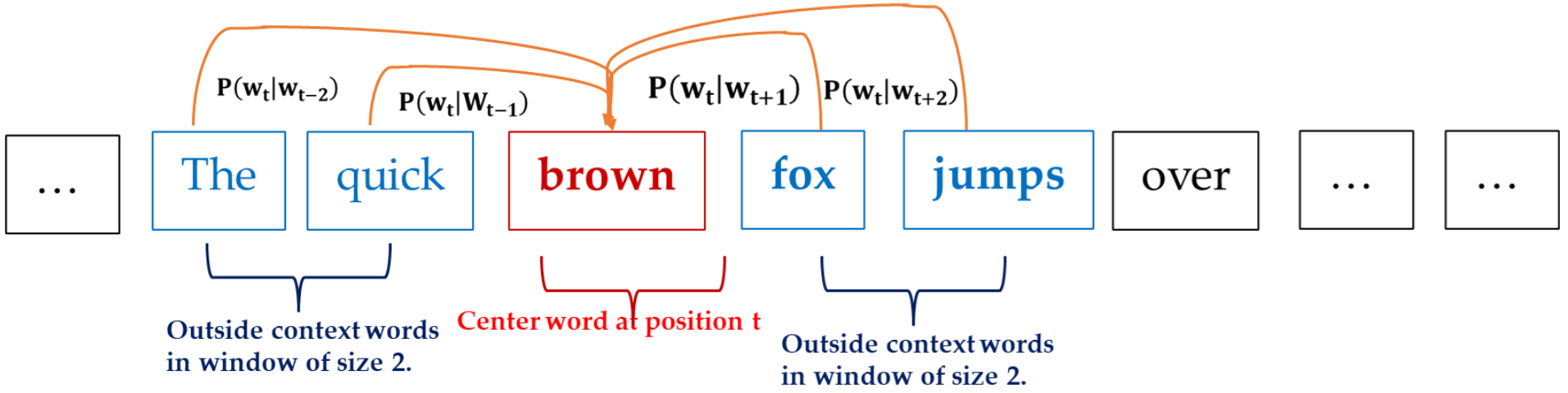
4.4 "word2vec": Local Contexts.

- Instead of entire documents, word2vec uses words k positions away from each center word.
  - These center words are called context words.
- Example: for k = 2
  - "The quick **brown fox jumps** over the lazy dog."
    - **Center word – brown (also called focus word).**
    - **Context words (The, quick) and (fox, jumps) also called target words.**
- **word2vec** considers all words as **center** words, and all their **context** words.



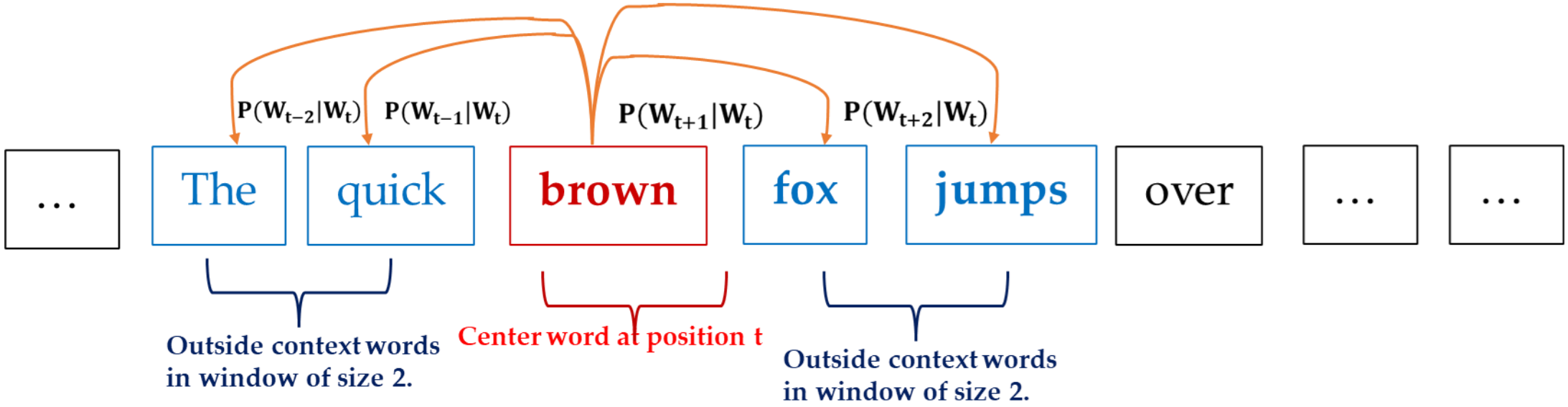
4.5 "word2vec" – variant: CBOW.

- CBOW – Continuous Bag of Words:
  - It is a neural network – based model in word2vec family.
  - Given a sequence of words, CBOW uses the context words within a window of size 2k (i.e. k words before and k words after the target) to predict the center word.

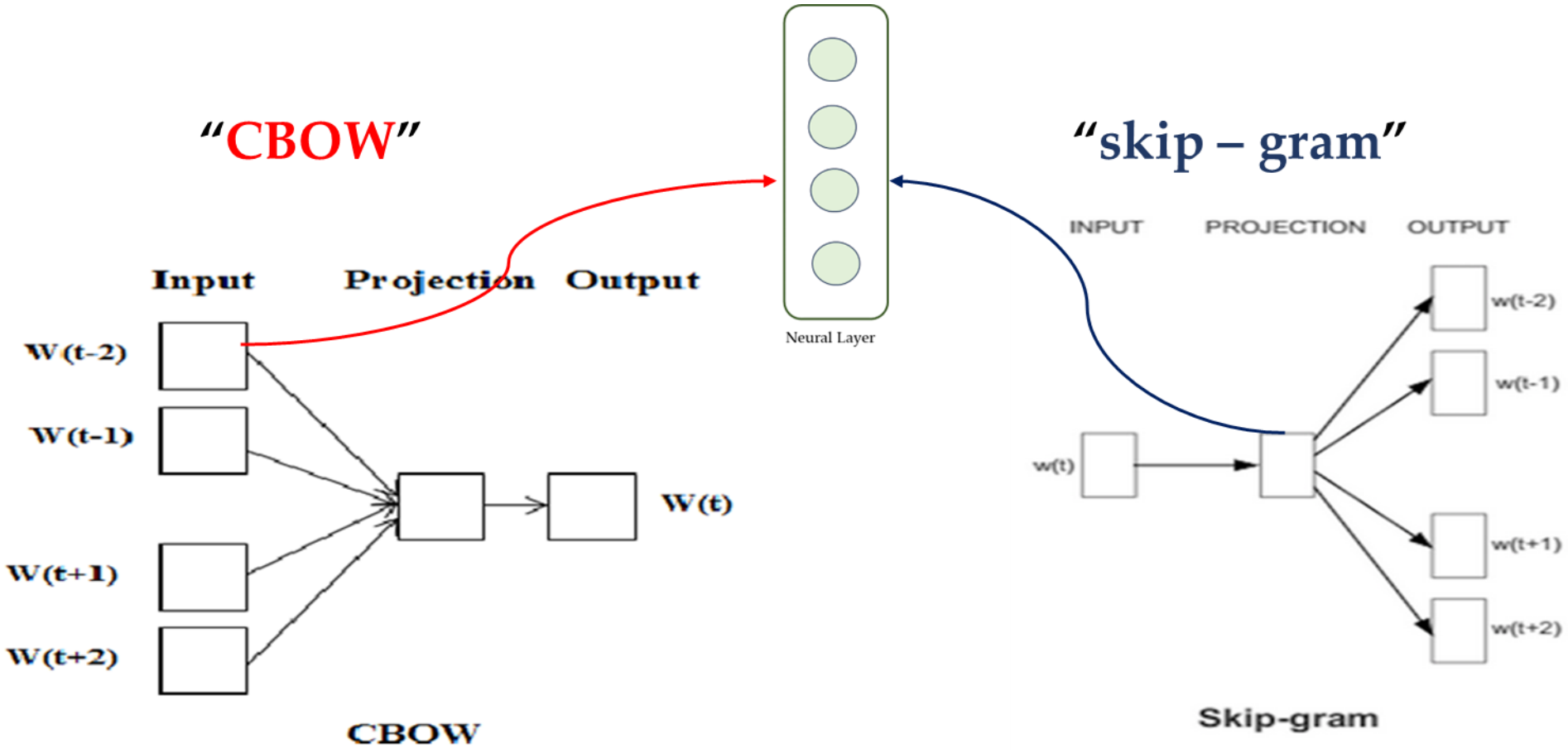


4.5 "word2vec" – variant: skip - gram.

- The idea: we want to use center words to predict their context words.
- Context: a fixed window of size 2k.



4.6 "word2vec": Architecture.



We will concentrate on "CBOW", { Idea is transferrable}

4.7 "CBOW": Dataset Generation.

- **Dataset Generation for "CBOW":**
  - Example Dataset: "the quick brown fox jumps over the lazy dog"
  - Step 1: we create a vocabulary of the unique words using tokenization:
    - **tokens** = ["the", "quick", "brown","fox", "jumps","over", "the", "lazy", "dog"]
    - Sort for consistency:
      - **sorted_tokens** = ["brown","dog","fox","jumps", "lazy", "over","quick","the"]
      - Assign index to each word:
  - Step 2: Create a one hot encoded vector based on tokens.

word	Index - Position	One – Hot Vector
brown	0	[1, 0, 0, 0, 0, 0, 0, 0]
dog	1	[0, 1, 0, 0, 0, 0, 0, 0]
fox	2	[… …]
jumps	3	[… …]
lazy	4	[… …]
over	5	[0, 0, 0, 0, 0, 1, 0, 0]
quick	6	[… …]
the	7	[0, 0, 0, 0, 0, 0, 0, 1]

4.7.1 "CBOW": Dataset Generation.

- **Dataset Generation for "CBOW":**
  - Step 3: set context window size.
    - **window_size, k = 2**
  - Step 4: Generate CBOW Training samples.

Context Words	Index or Position Representations {Context}	Target	Index or Position Representations {Target}
[the, quick, fox, jumps]	[7, 6, 2, 3]	brown	[0]
[quick, brown, jumps, over]	[6, 0, 3, 5]	fox	[2]
[brown, fox, over, the]	[0, 2, 5, 7]	jumps	[3]
[fox, jumps, the, lazy]	[2, 3, 7, 4]	over	[5]
[jumps, over, lazy, dog]	[3, 5, 4, 1]	the	[7]

  - Based on the index created as per sorted vocabulary the whole sentence can be:
  - "**the quick brown fox jumps over the lazy dog**" → **[7, 6, 0, 2, 3, 5, 7, 4, 1]**

4.7.2 "CBOW": Look Up Table.

word	Index - Position	One – Hot Vector
brown	0	[1, 0, 0, 0, 0, 0, 0, 0]
dog	1	[0, 1, 0, 0, 0, 0, 0, 0]
fox	2	[... ...]
jumps	3	[... ...]
lazy	4	[... ...]
over	5	[0, 0, 0, 0, 0, 1, 0, 0]
quick	6	[... ...]
the	7	[0, 0, 0, 0, 0, 0, 0, 1]

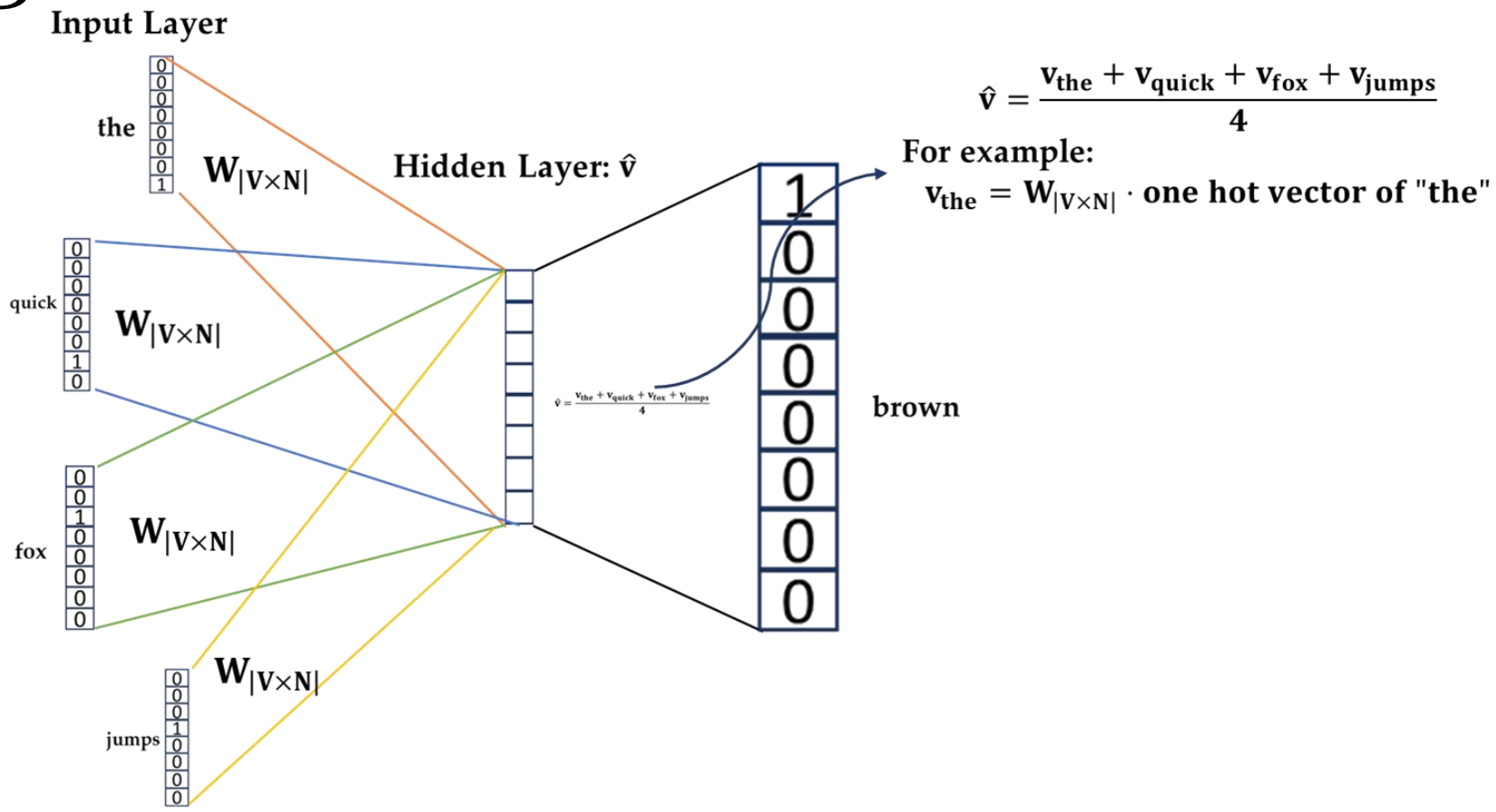
Context Words	Index or Position Representations {Context}	Target	Index or Position Representations {Target}
[the, quick, fox, jumps]	[7, 6, 2, 3]	brown	[0]
[quick, brown, jumps, over]	[6, 0, 3, 5]	fox	[2]
[brown, fox, over, the]	[0, 2, 5, 7]	jumps	[3]
[fox, jumps, the, lazy]	[2, 3, 7, 4]	over	[5]
[jumps, over, lazy, dog]	[3, 5, 4, 1]	the	[7]

Fig: Similar Look Up Table is Created for whole Vocabulary.

Cautions: Every word in doc token representations becomes target word which is not represented in above table.

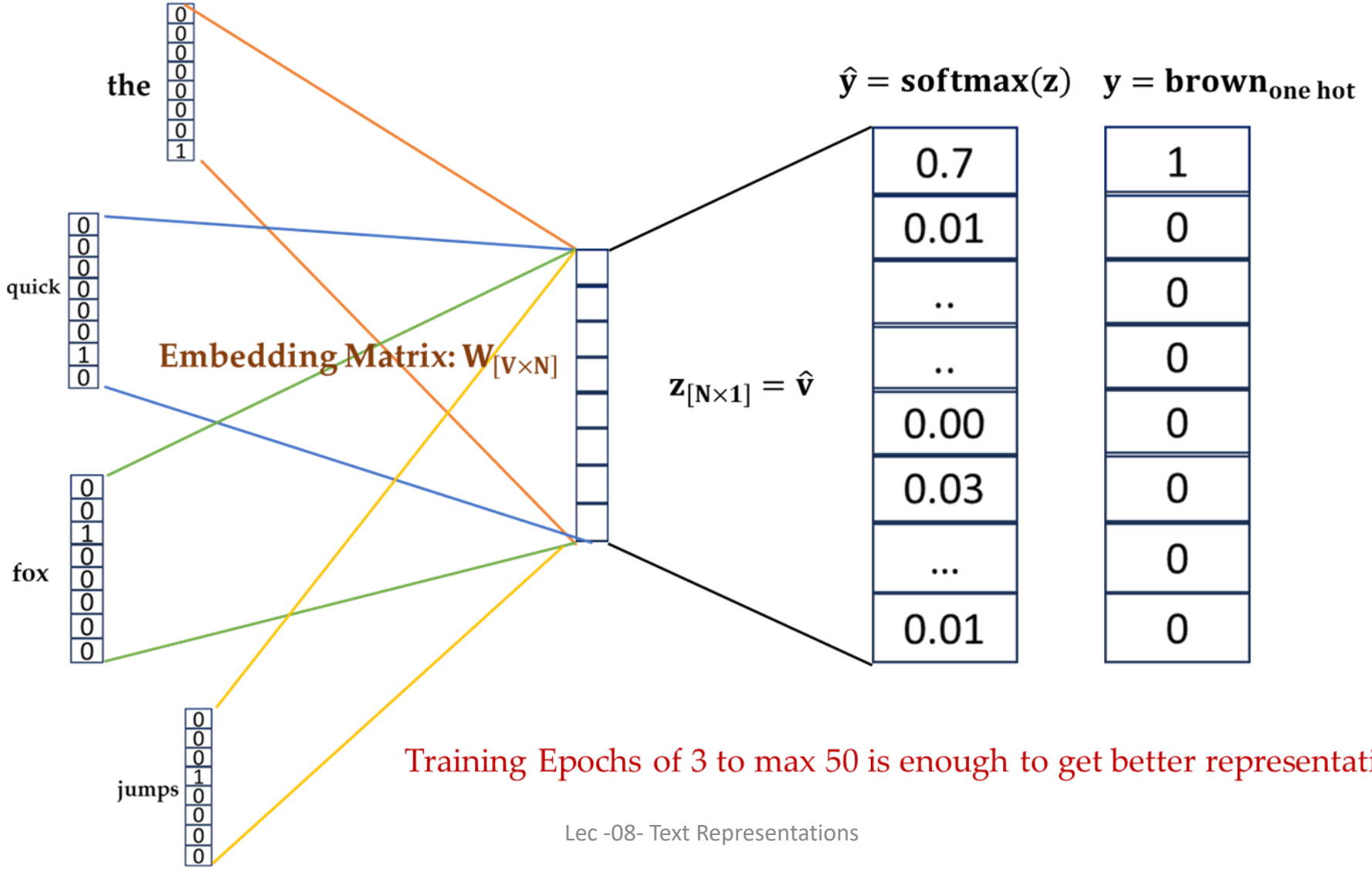
4.8 Training "CBOW".

- Example Input:
  - the quick brown fox …….
  - Window size k = 2
- Position – 2
  - Target = "brown"
  - Context = [the, quick, fox, jumps]



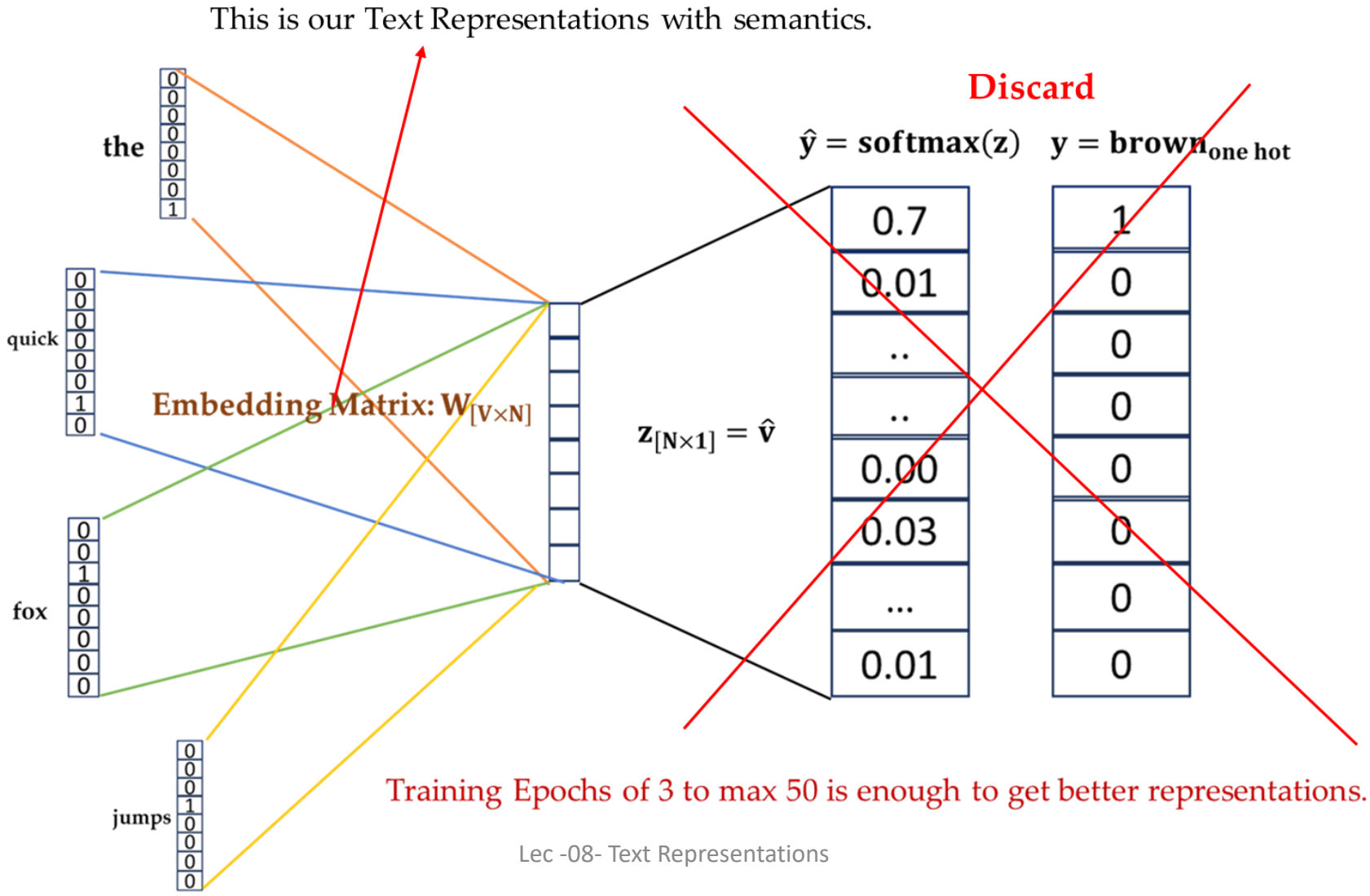
- $W_{|V\times N|}$ → **Embedding Matrix**
  - Here:
    - **V → len of vocabulary**
    - **N → number of neuron in Hidden Layer kept less than V**
    - **Initialized Randomly.**

4.8.1 Train using "Gradient Descent".



Training Epochs of 3 to max 50 is enough to get better representations.

4.8.2 After Training.



4.9 Why "word2vec"?

- Word2Vec embeddings capture **semantic relationships** between words — so well,
  - in fact, that you can do **word analogy arithmetic** like this famous one:
  - king − man + woman ≈ queen i.e. for the question:
    - man is to king as woman is to ...?
      - Probability for queen is the highest in the corpus of English language …

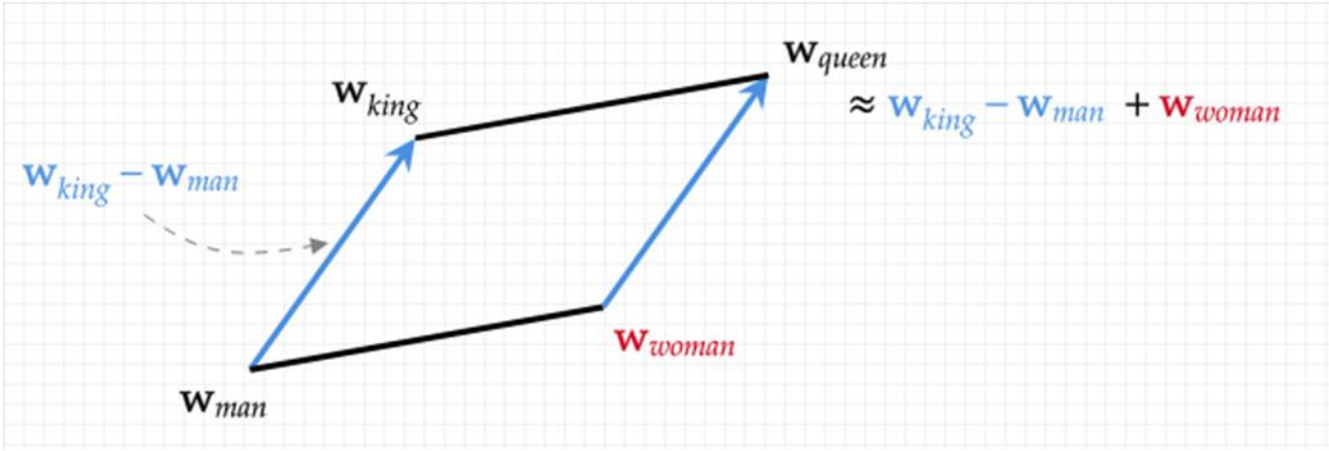


Fig: word2vec Word Embedding Vector Space.

4.10 "word2vec" in Practice.

- In practice, **we usually don't train Word2Vec from scratch,**
  - Instead, we use **pretrained Word2Vec models**, especially the popular one trained by Google.
- **Pre – trained word2vec by Google:**
  - Corpus: Trained on Google News dataset (approx. 10 billion words)
    - Vocab size – approx. 3 million word/phrases.
  - Dimension available: 50D, 100D, 200D, 300D
    - **300D is a most popular and is best suit for capturing word relationships.**
- **Using Genism to Load Pre – trained word2vec:**

```
# Load pretrained Google News Word2Vec model (300 dimensional)
# Download from: https://code.google.com/archive/p/word2vec/
# File: GoogleNews-vectors-negative300.bin.gz (about 1.5GB)
from gensim.models import KeyedVectors

model_path = 'GoogleNews-vectors-negative300.bin.gz'
print("Model loaded!")

# Check vector for a word
vector = word2vec['computer']
print("Vector for 'computer':", vector[:10])  # print first 10 dims

# Check similarity
print("Similarity between 'king' and 'queen':", word2vec.similarity('king', 'queen'))

# Word analogy: king - man + woman ≈ queen
result = word2vec.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print("Result of analogy (king - man + woman):", result)
```

4.11 Alternate to "word2vec".

Model	Description	Dimensions	Trained On
word2vec	Predictive Model (skip –gram / CBOW)	100, 300	Google News
GloVe	Count – based – matrix factorizations	50, 100, 200, 300	Common Crawl, Wikipedia
FastText	Extends word2vec using sub word info	300	Common Crawl, Wikipedia
ELMo, BERT	Contextual word embeddings {Transformer based → LLM are based on this.}	Variable (million +)	Huge Corpora

Thank You