Welcome Here!!

NATURAL LANGUAGE PROCESSING (NLP)

AN INTRODUCTION TO AI, ML, DEEP LEARNING & NLP

DSN LEKKI-AJAH

BY ABEREJO HABEEBLAH O.

X (TWITTER): @HABEREJO

Day 1

Tuesday, 4th March, 2025

Why are we here

To equip learners with the skills to understand how Artificial Intelligence models are built, explore Machine Learning and Deep Learning, and focus on Natural Language Processing (NLP).

We should build some project right?

Yes!!

If you are in with me,

We should build some project.

Classes Structure

Every Tuesday throughout the month of March.

4 classes in total.

Each class for about 1:45 minutes

Days are NOT likely to change, although, my schedule can be OUCH, I wil reachout earlier before. Understand me bikoooo

Who should be here

- You're curious and ready to learn something new
- You dream of becoming a Data Scientist.
- You're passionate about building the future as a Machine Learning Engineer.
- You have a basic understanding of Python programming.
- You've worked with data and want to take your skills to the next level.
- You're a researcher exploring the exciting world of AI.

which of these are you?

Who should be here

- You're familiar with the fundamentals of Machine Learning and Deep Learning.
- You love problem solving and are excited to tackle real world problems with AI.
- You want to understand the technology that is shaping the future.
- You're driven by a desire to learn and grow in the field of AI.
- You enjoy collaborating and learning from others.

Which of these are you?????
You ticked any of these boxes?
Let's Gooooooo

Where Do We Begin

- Know my audience
- Understand why they join this program
- Discuss previous projects and skills in AI

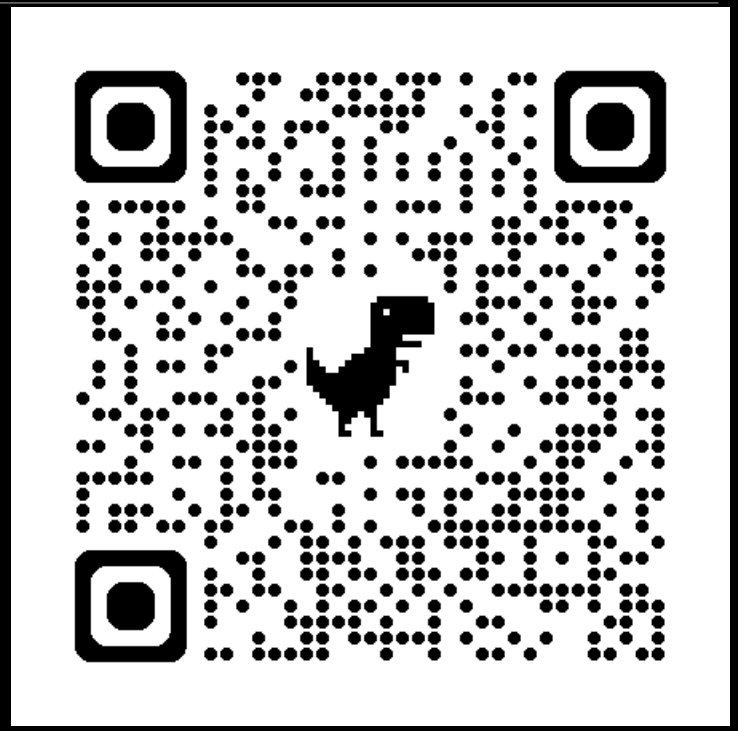
Let's have some discussion here

Before we begin

Who I am?

- A Machine Learning Engineer
- A young MAN eager to master AI

FUN FACT:
I've taught over 100 people Python, but
I'm still learning new things every day.
(Especially how to avoid typos when
coding late at night!)



Scan to visit my portfolio
or
https://bheez.netlify.app

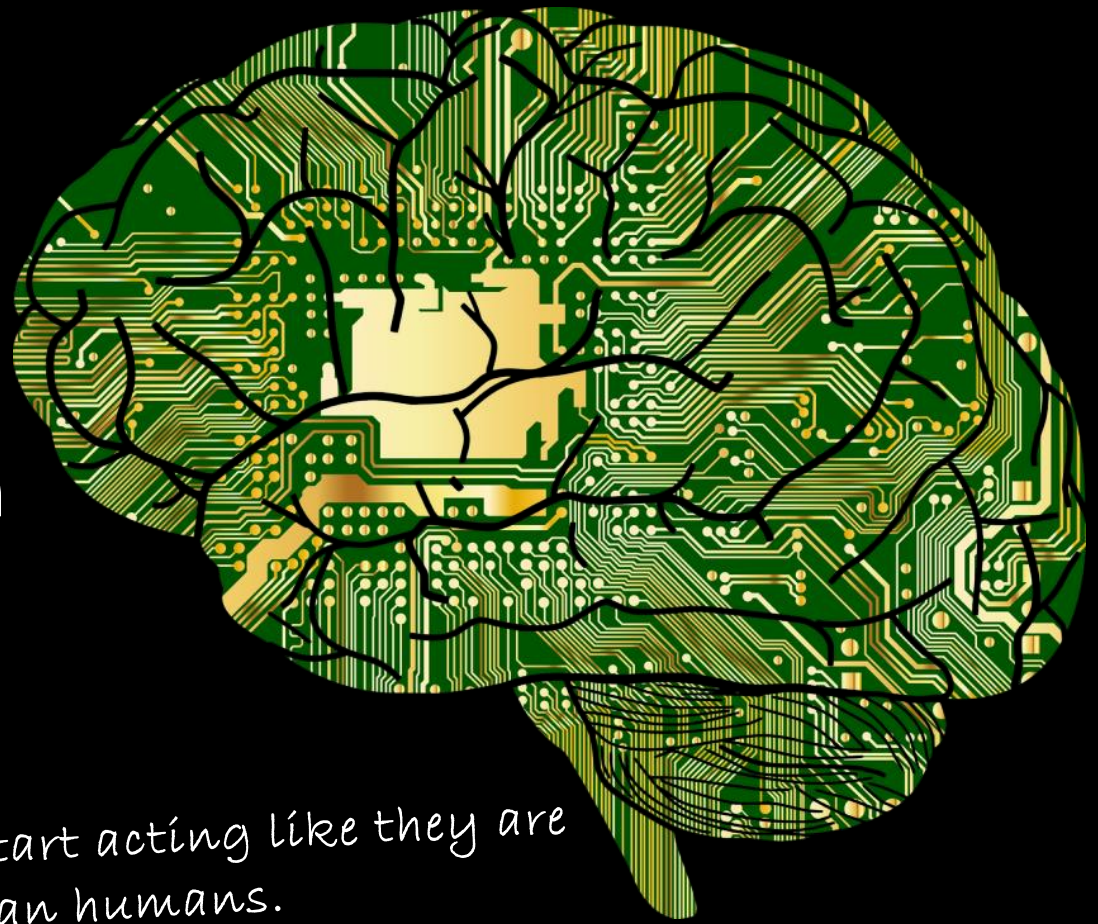
Course Structure

1. Introduction to AI, ML, DL, and NLP
2. Understanding AI vs ML vs DL
3. Introduction to NLP
4. NLP Use Cases
5. Text Preprocessing Techniques
6. Hands-on NLP Tasks
7. Brain Teasers & Discussion

And lots more

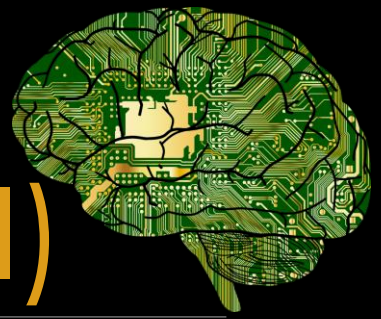
What is Artificial Intelligence (AI)

Artificial Intelligence is the ability of machines to mimic human intelligence. It enables computers and systems to learn from data, solve problem, and make decisions just like humans do.



AI is when machines start acting like they are smarter than humans.

What is Artificial Intelligence (AI)



Let's relate like this again:

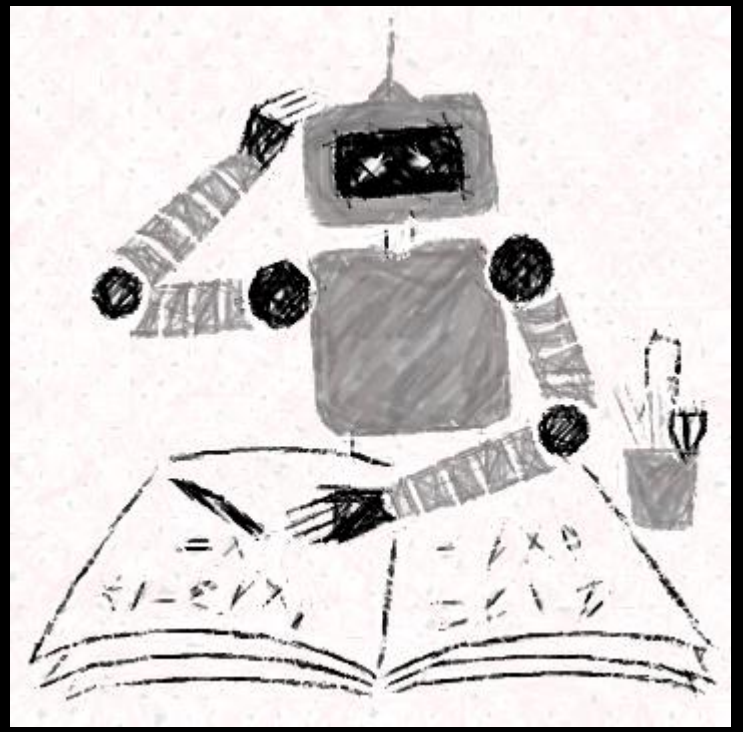
AI is like teaching your computer to do the thinking for you, only that it doesn't demand coffee breaks or complain about Mondays!!

hahaha😀😀😀😀

AI is when machines start acting like they are smarter than humans.

What is Machine Learning (ML)

Machine learning is turning things (data) into numbers and finding patterns in those numbers.



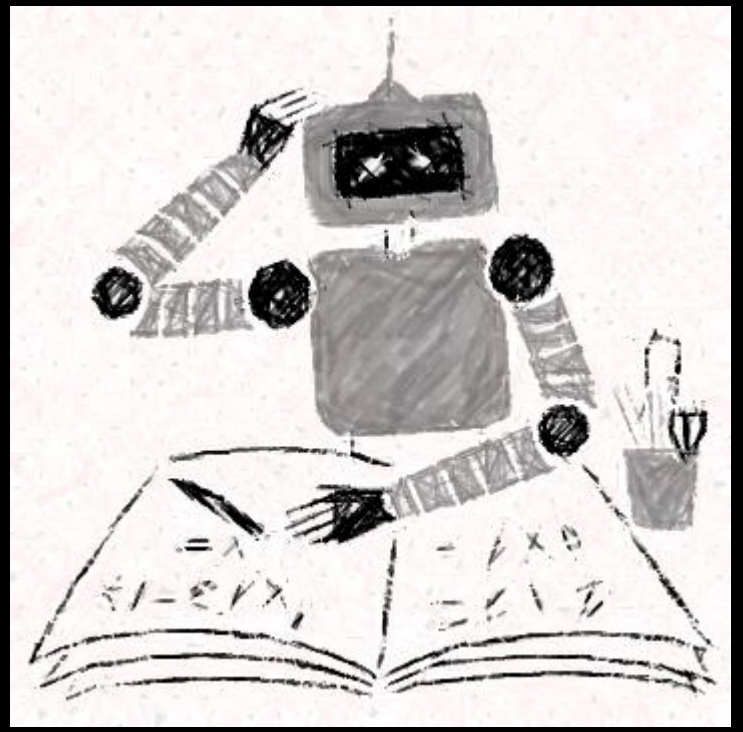
Got It???

Let's Goooooo!!!

What is Machine Learning (ML)

Think of it as teaching a computer how to recognize patterns without explicitly programming it.

It uses some complex algorithms to find patterns.



Got It???

Let's Goooooo!!!

What is Machine Learning (ML)

Find pattern? How?

Imagine teaching a child to identify fruits. You show them a banana, an apple, and an orange, telling them the names. After seeing many examples, they can recognize a new banana without needing you to tell them. That's what ML does—it learns from past examples and generalizes.

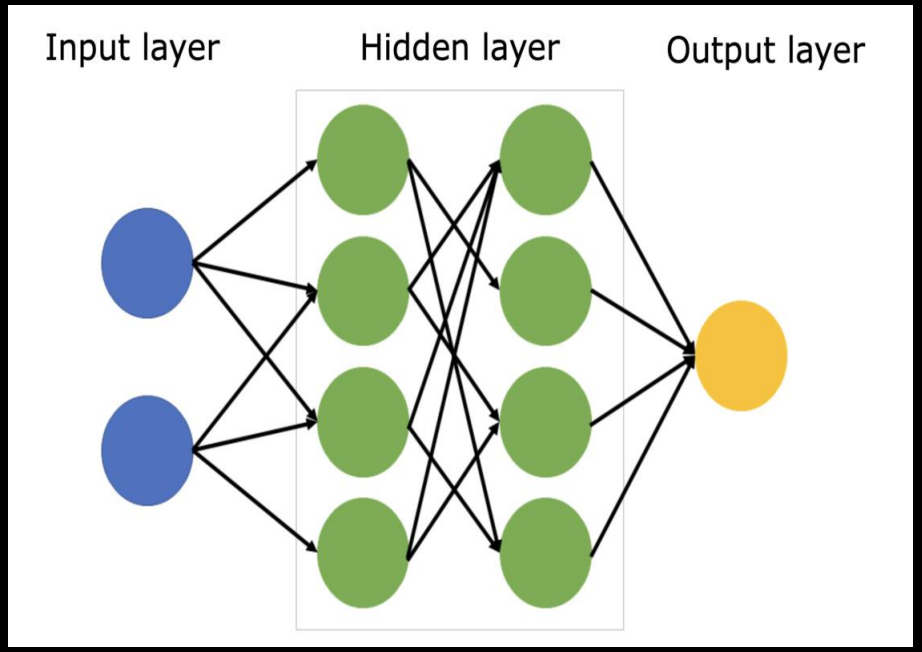
Got It??? Let's Goooooo!!!

What is Deep Learning (DL)

A deeper version of ML.

It works like our brain with many layers of learning.

Instead of just learning basic rules, it finds complex patterns automatically by using lots of examples.



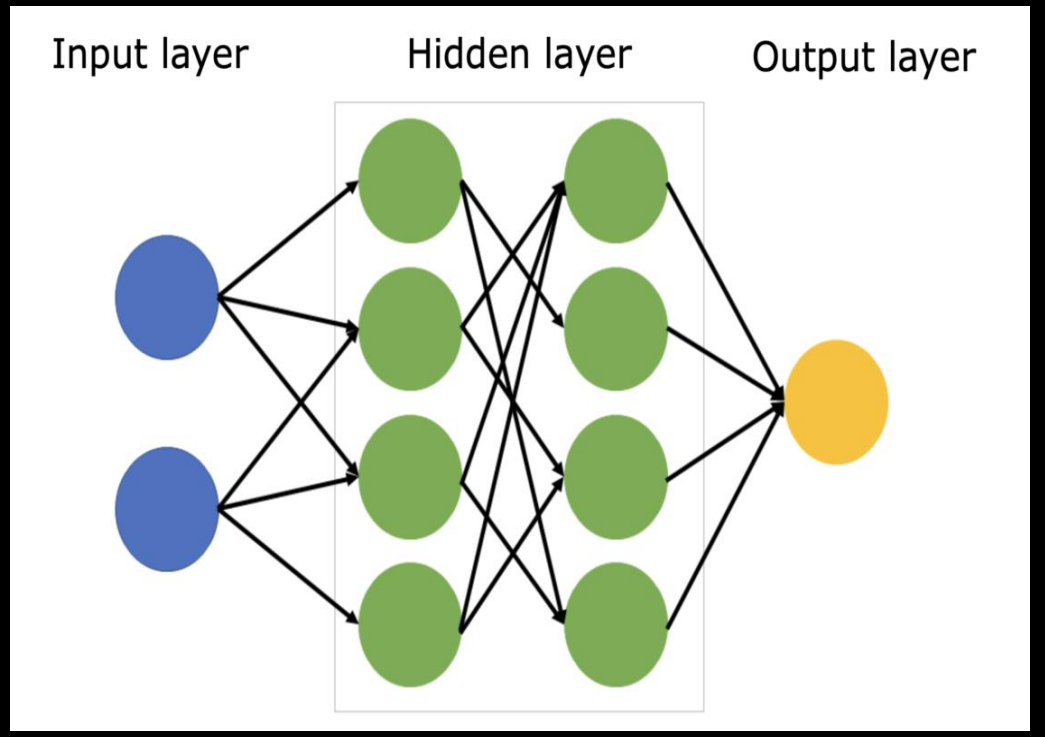
Got It???

Let's Goooooo!!!

What is Deep Learning (DL)

It uses multi-layered neural networks (i.e. Deep Neural Networks) to automatically extract features from data.

It is especially effective for tasks like image recognition, speech processing, and NLP.



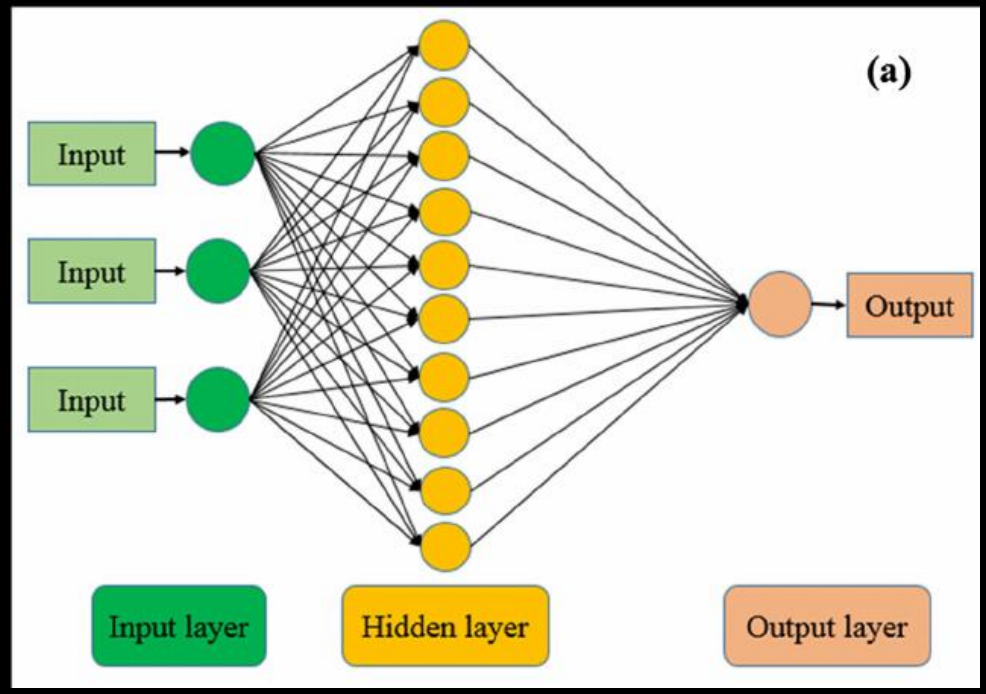
Got It???

Let's Goooooo!!!

What is Deep Learning (DL)

Neural networks? How?

Imagine teaching a robot to recognize handwriting. Traditional ML would require manually programming rules for different letters. But Deep Learning would just look at thousands of handwritten examples and figure out the differences itself—like how humans learn handwriting by seeing and practicing.

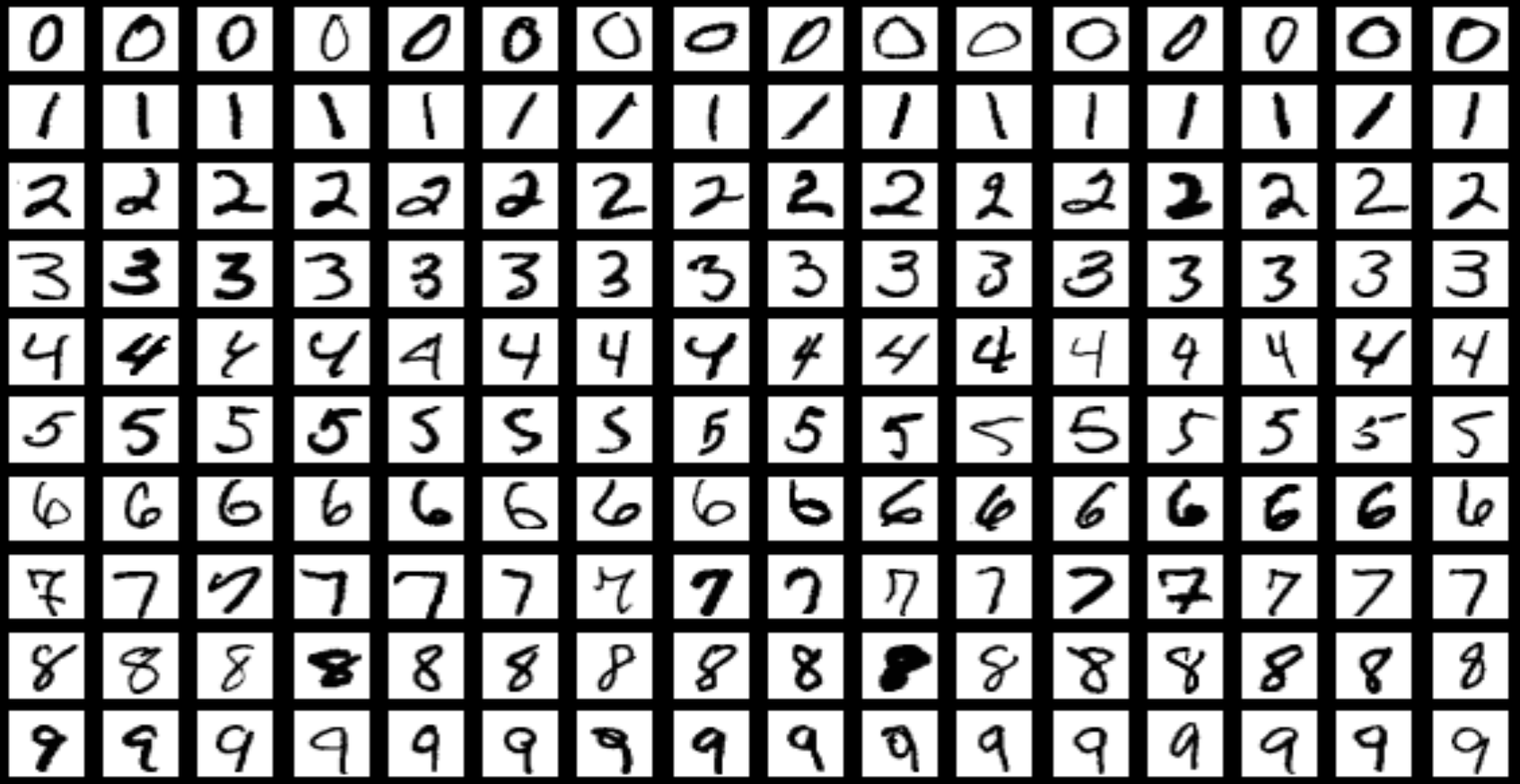


Got It???

Let's Goooooo!!!

What is Deep Learning (DL)

Neural networks? How?

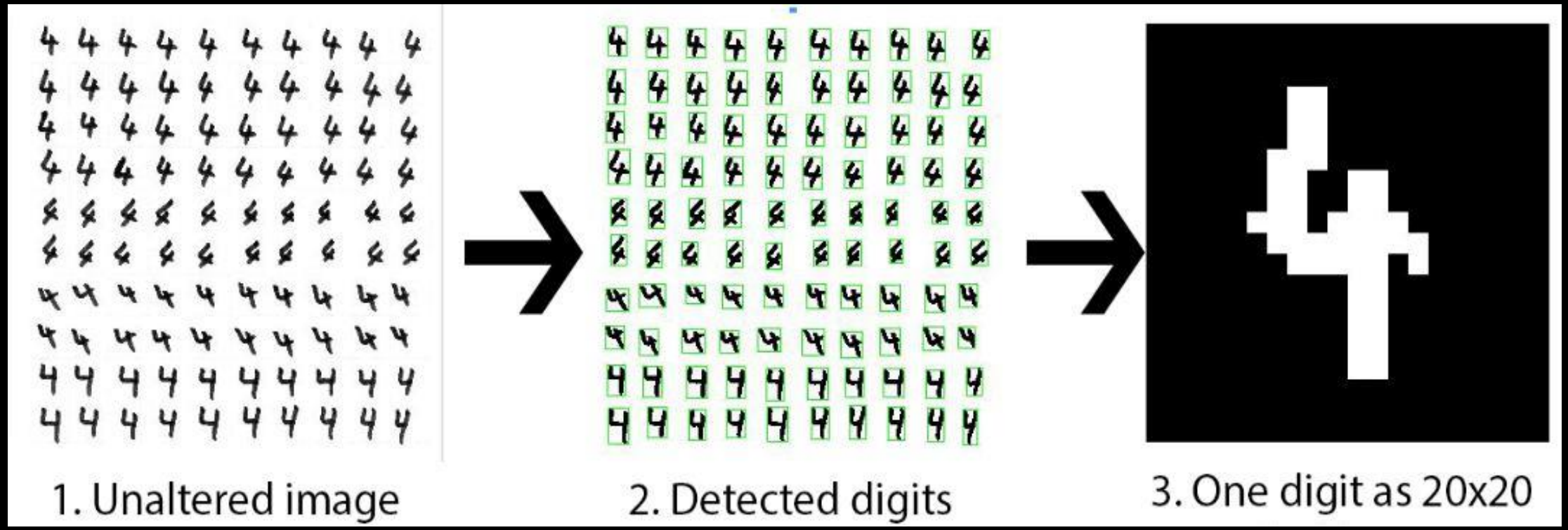


Got It???

Let's Goooooo!!!

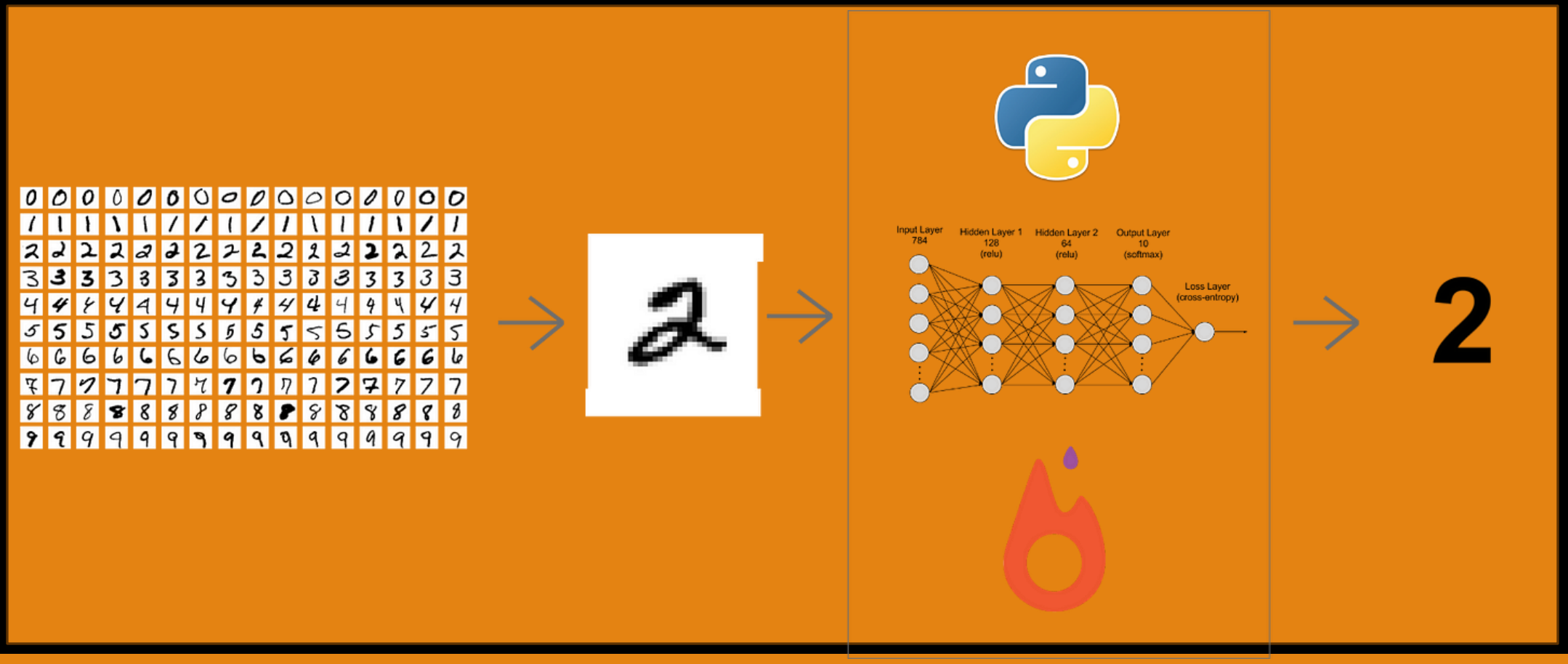
What is Deep Learning (DL)

Neural networks? How?



What is Deep Learning (DL)

Neural networks? How?

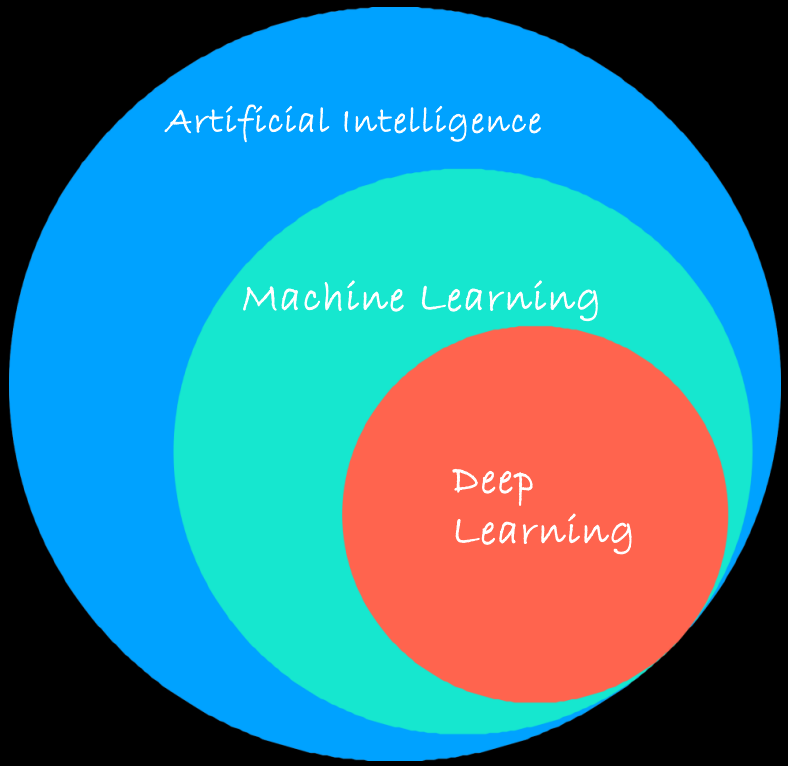


AI vs ML vs Deep Learning (DL)

AI is like a **robotic assistant** that can follow instructions,

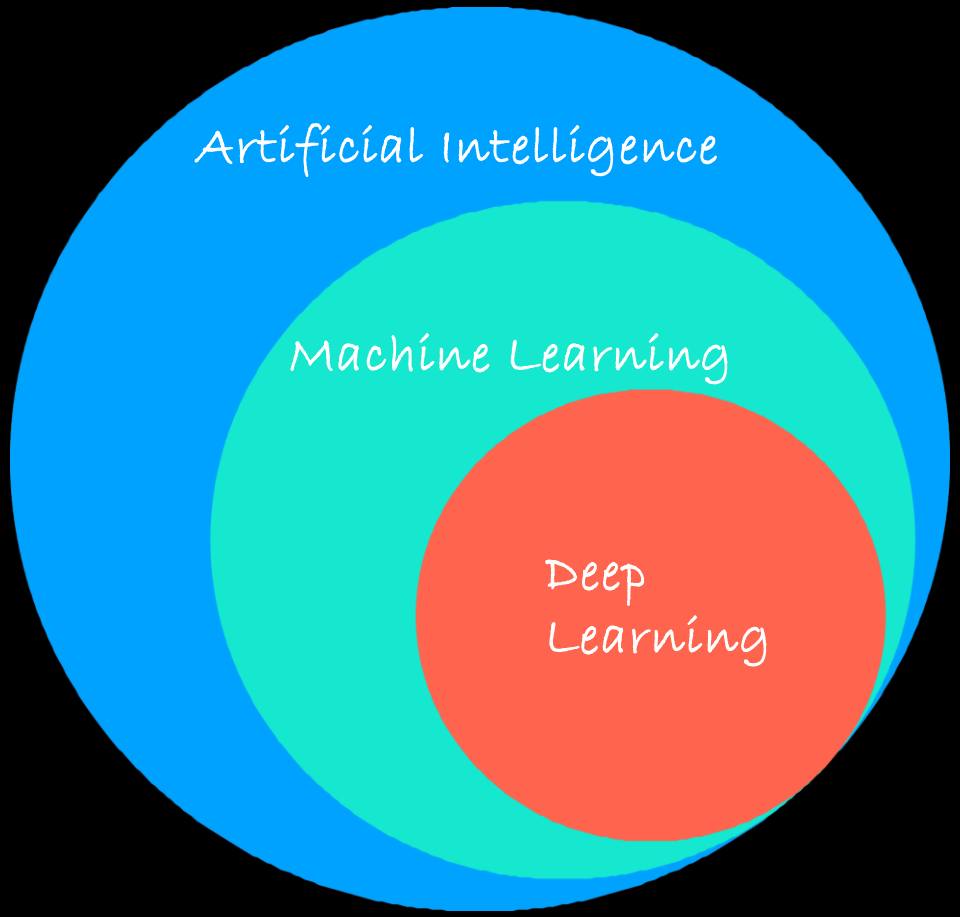
ML is when it learns from experience,

DL is when it learns so deeply that it can recognize faces, voices, and even create art! 🎨🤖



Where do we group NLP??

AI vs ML vs Deep Learning (DL)

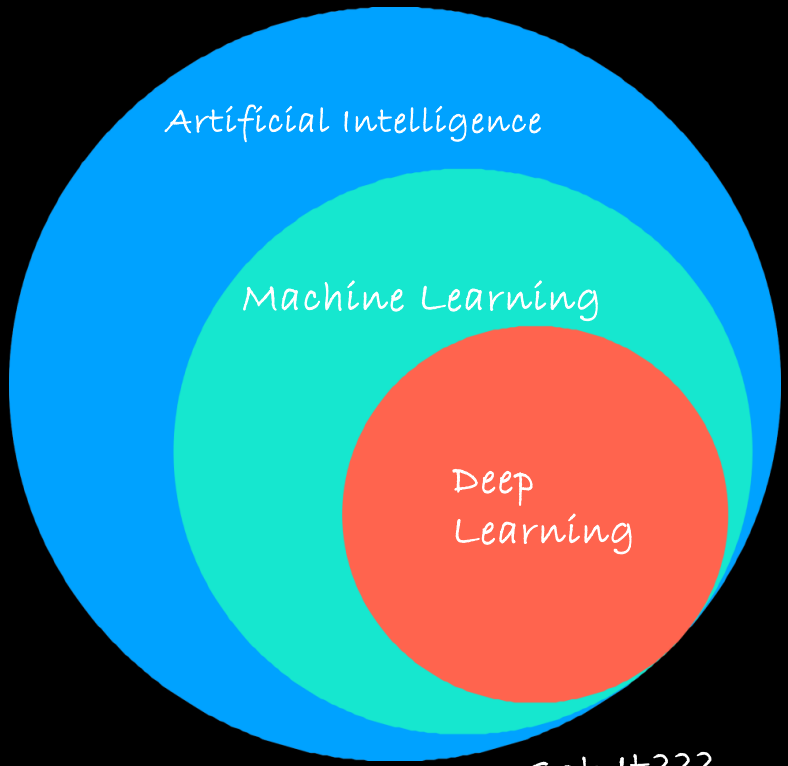


NLP is a sub of Deep Learning..

AI vs ML vs Deep Learning (DL)

Let's relate with this:

AI is like a kitchen assistant following recipes, ML is like a cook who tweaks dishes based on feedback, and DL is like a top chef who invents new recipes just by understanding flavors! 🍳🤖🔥

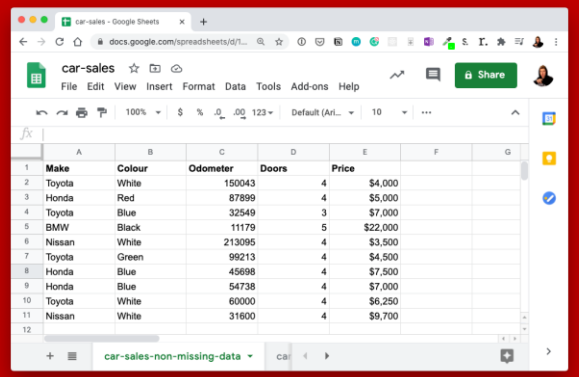


Got It???

Let's Goooooo!!!

ML vs DL

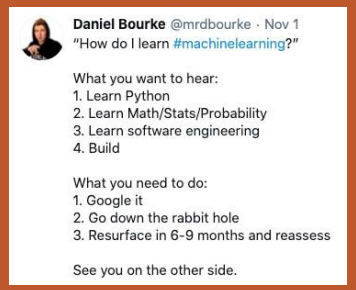
Machine Learning

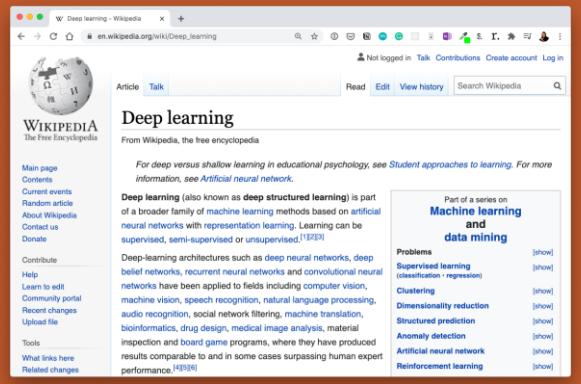


Make	Colour	Odometer	Doors	Price
Toyota	White	150043	4	$4,000
Honda	Red	87899	4	$5,000
Toyota	Blue	32549	3	$7,000
BMW	Black	11179	5	$22,000
Nissan	White	213095	4	$3,500
Toyota	Green	99213	4	$4,500
Honda	Blue	45698	4	$7,500
Honda	Blue	54738	4	$7,000
Toyota	White	60000	4	$6,250
Nissan	White	31600	4	$9,700

Structured data

Deep Leanring

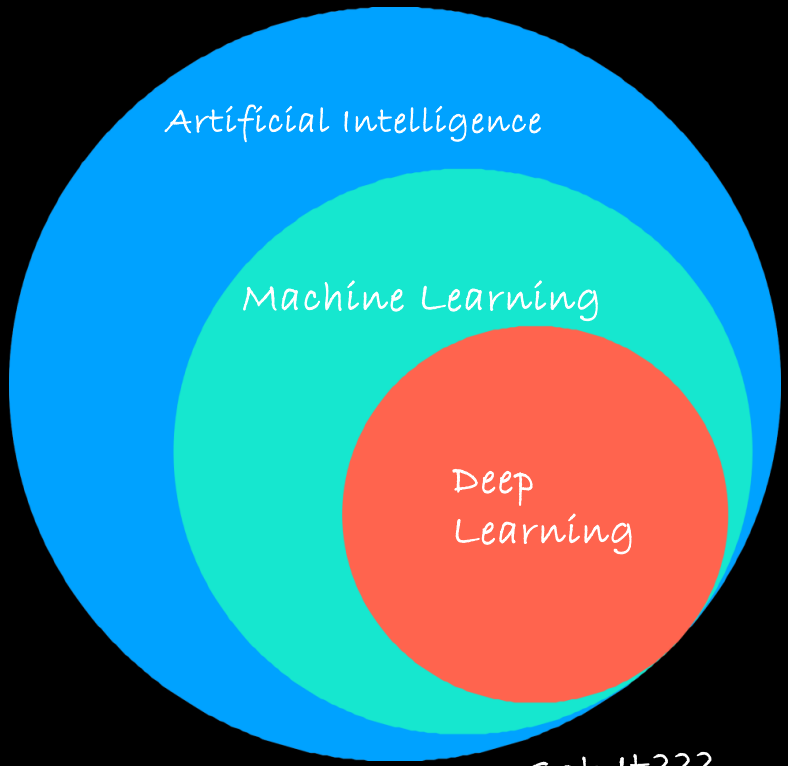




Unstructured Data

AI vs ML vs Deep Learning (DL)

Can AI exist without Machine Learning or Deep Learning???



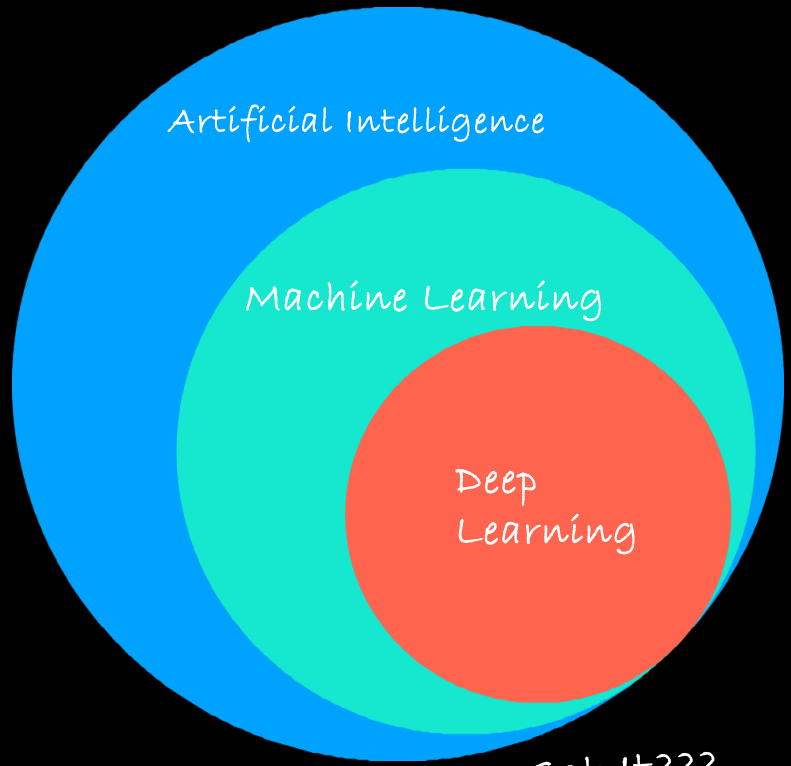
Got It???

Let's Goooooo!!!

AI vs ML vs Deep Learning (DL)

Can AI exist without Machine Learning or Deep Learning???

The answer is yes, AI can exist without ML or DL, although it might not be the kind of AI we often see in movies or science fiction.



Got It???

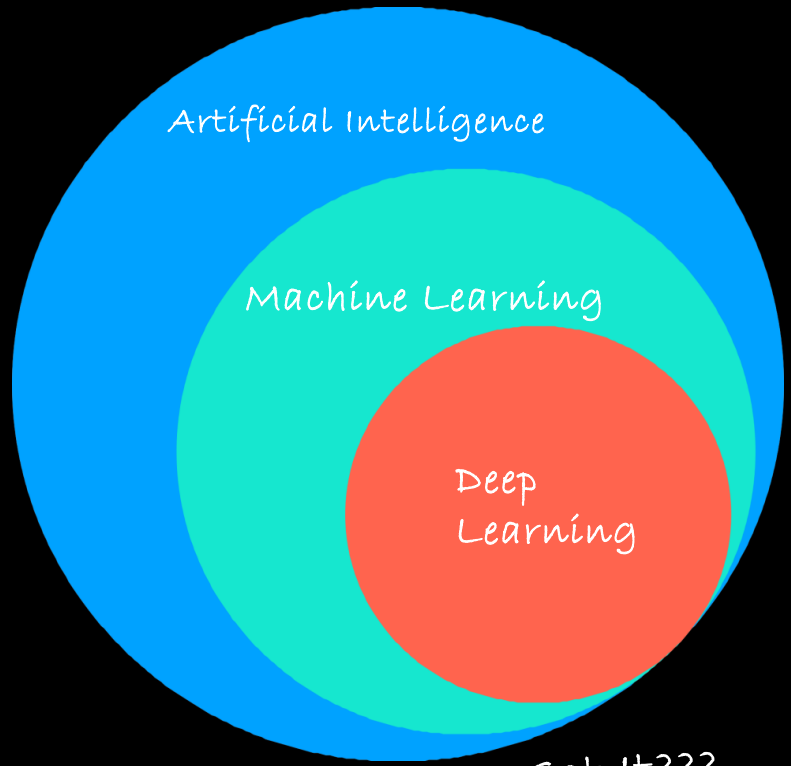
Let's Goooooo!!!

AI vs ML vs Deep Learning (DL)

AI encompasses any technique that enables computers to mimic human intelligence.

Rule-based systems were an early form of AI that relied on predefined rules rather than learning from data.

Machine Learning and Deep Learning, although dominant today, are subsets of AI that use algorithms to learn from data.



Got It???

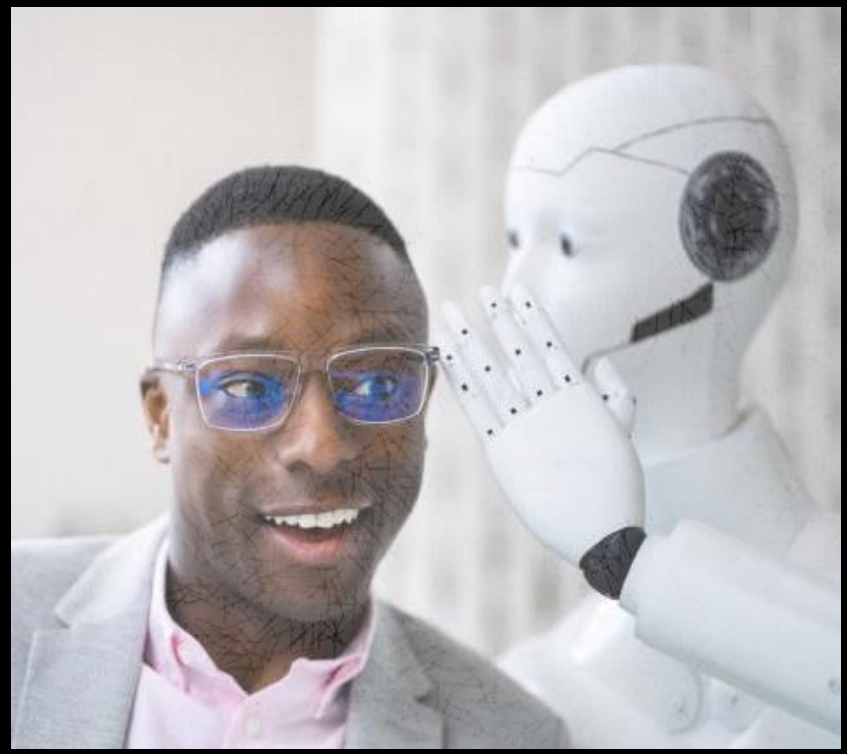
Let's Goooooo!!!

What is Natural Language Processing (NLP)

NATURAL LANGUGAE PROCESSING (NLP) helps computers understand and talk like humans.

It enables computers to read, interpret, and generate human-like text.

It allows your phone to predict your next word, understand voice commands, and translate languages.

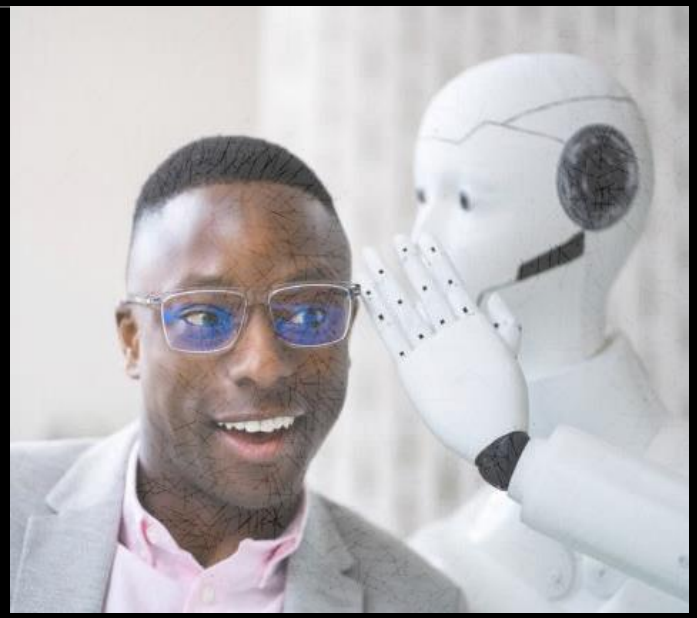


What is Natural Language Processing (NLP)

NLP combines ML and Deep Learning to process and understand human language.

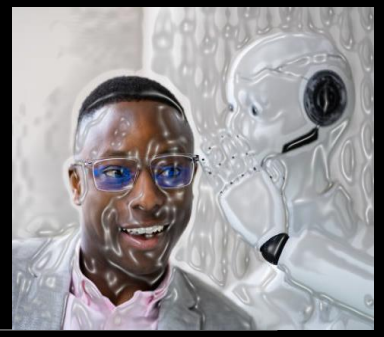
Includes tasks like stemming, tokenization, NER, sentiment analysis.

NER – named entity recognition



Here is a scenario; Imagine chatting with a smart assistant (like ChatGPT or Siri). You say, "What's the weather today?", and it understands your words, fetches present weather data, and responds. That's NLP in action.

Natural Language Processing (NLP) in Industry



Voice Assistants 🎙️ **:** Siri, Alexa, and Google Assistant understand and respond to your voice commands.

Chatbots 💬**:** Websites use NLP-powered bots to answer your questions instantly.

Autocorrect & Predictive Text 📱 **:** Your phone suggests next words while typing to speed things up, Also can correct spellings when you typed wrongly.

All this industry examples uses NLP in operation.

Natural Language Processing (NLP) in Industry

Translation Apps 🌍: Google Translate helps you understand different languages.

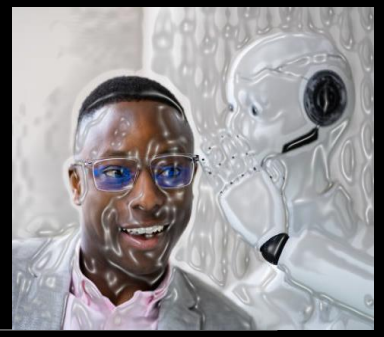
Spam Filters 📧: Your email sorting out spam messages automatically.

Search Engines 🔍: To understands what you mean, even if you type a messy query.

Sentiment Analysis 😀😡: Companies analyze tweets and reviews to see what people feel about their brand.

All this industry examples uses NLP in operation. Can I get from you??

NLP use cases



"What's the topic of this text?" (text classification)

"Does this text contain abuse?" (content filtering)

"Does this text sound positive or negative?" (sentiment analysis)

"What should be the next word in this incomplete sentence?" (language modeling)

"How would you say this in German?" (translation)

"How would you summarize this article in one paragraph?" (summarization)

Got The Point Now????

Brief History of Natural Language Processing (NLP)

2011: George Dahl and Microsoft Research trained a deep neural network to recognize words from speech recordings—a major deep learning breakthrough in NLP.

2012: AlexNet revolutionized deep learning in image recognition, leading to increased interest in machine vision.

2015: Deep learning models started competing in NLP tasks like machine translation, matching traditional ML accuracy with less effort and lower computational cost.

2016-2017: Deep learning models in NLP became more efficient and began outperforming traditional ML approaches in accuracy.

Impact: Enabled real-time translation on mobile devices without requiring an internet connection, making NLP more accessible and practical.

What's Next 😀 😀

Maybe I don't know also

After all of the theoretical explanation, can we begin some hands-on or we stop here today?

What's Next

Maybe I don't know also

If Yes, 

Let's get our Machine setup for hands-on.

Setting up our PC

We need install some tools if we are to work locally (i.e without internet)

Since we are online, we may not delve into this page in details

If you will be working locally, make sure to install the following

1. Python (ensure to add to path when installing)
2. Jupyter Lab (I like to use command - pip install jupyterlab – in my CMD/Terminal)
3. Tensorflow (use the command - pip install tensorflow – into your CMD/Terminal)
4. Some frameworks and libraries we will use as we move

If working online,
visit google colab and let's start

What's Next 😀 😀

Maybe I don't know also

After setup,
Let's begin proper proper

The END

DSN LEKKI-AJAH

BY ABEREJO HABEEBLAH O.

X: @HABEREJO

Day 1

NATURAL LANGUAGE PROCESSING (NLP)

TEXT PREPROCESSING AND TOKENIZATION

DSN LEKKI-AJAH

BY ABEREJO HABEEBLAH O.

X: @HABEREJO

Day 1

How computer sees language

Computers see text as a sequence of character,

Computers don't understand words or meaning like humans do

They see all English words as binary language (i.e. 1s and 0s)

So, now, we need to help computer understand our language to what it can fully understand

What do we do to help computers understand the text? Let's GO

Binary Language == 1 and 0

Understand Human Language

There are arguably 3 steps to help computer understand human texts,

1. Text Preprocessing
2. Tokenization
3. Numerical Representation (Word Embedding)

Now, Lets discuss each further!!

Binary Language == 1 and 0

Step 1: Text Preprocessing

Why Preprocess Text?

Imagine you have a room full of clothes, toys, and papers scattered everywhere. Finding what you need is hard, right?

Raw text data is similar. It's full of "**noise**" – things that make it difficult for computers to understand the meaning.

Got It???

Let's Goooooo!!!

Step 1: Text Preprocessing

At this stage, lets understand preprocessing of text,

Proper Text preprocessing is important to make sure we are only feeding the right processed information into our algorithm and irrelevant text are gone.

e.g: The Quick! bRoWn foX juMPS.

Got It???

Let's Goooooo!!!

Step 1: Text Preprocessing

Text preprocessing cleans and prepares text data for NLP tasks. It addresses issues like inconsistencies (uppercase/lowercase), irrelevant information (stop words), and variations in word forms (running vs. run). This makes it easier for NLP models to learn.

e.g: The Quick! bRoWn foX juMPS.

Got It???

Let's Goooooo!!!

Step 1: Text Preprocessing

We prepare text by:

1. Lowercasing
2. Removing Punctuation Marks
3. Removing Stopwords (i.e words like A, of, in, etc)
4. Stemming / Lemmatization

e.g: The Quick! bRoWn foX juMPS.

NB:
There is no specific way to prepare text, it is much dependent on what type of NLP project we are building!

Step 1: Text Preprocessing

1. Lowercasing: To humans, A capitalized word at the beginning of a sentence (e.g., She) has the same meaning as when it's used later in a sentence (she).

By converting all characters in a corpus to lowercase, we disregard any use of capitalization.

Computers treat "The" and "the" as different words. Lowercasing makes everything consistent.

The Quick! bRoWn foX juMPS.

the quick! brown fox jumps.

NB:
Corpus in this regard means collection of writings.

Step 1: Text Preprocessing

Though, in a larger corpus that has many more examples of individual uses of words, the word, **general (an adjective meaning "widespread")** have different meaning to the word **General (a noun meaning the commander of an army).** One option is not to lowercase everything, or try an option called **Part-Of-Speech (POS) Tagging**

The Quick! bRoWn foX juMPS.

the quick! brown fox jumps.

NB:
Corpus in this regard means collection of writings.

Step 1: Text Preprocessing

2. Removing Punctuation Marks: Punctuation marks generally don't add much value to a natural language model and so are **often** removed.

Removing punctuation would not be an advantage in all cases. Consider, for example, if you were building a question-answering algorithm, which could use question marks to help it identify questions.

The Quick! bRoWn foX juMPS.

the quick brown fox jumps

Depending on the NLP project, some steps are compulsory

Step 1: Text Preprocessing

3. Removing Stopwords: Stopwords are frequently occurring words that tend to contain relatively little distinctive meaning, such as **the**, **at**, **which**, **and**, **of** etc.

There is no universal consensus on the precise list of stop words, but depending on your application it may be sensible to ensure that certain words are (or aren't!) considered to be stop words.

The Quick! bRoWn foX juMPS.

quick brown fox jumps

Depending on the NLP project, some steps are compulsory

Step 1: Text Preprocessing

For example, when building a model to **classify movie reviews** as **positive** or **negative**. Some lists of stop words include negations like **didn't, isn't, and wouldn't** that might be critical for our model to identify the sentiment of a movie review, so these words probably shouldn't be removed.

Negations may be helpful as stop words for some classifiers but probably not for a sentiment classifier

So be careful with this one. In many instances, it will be best to remove only a limited number of stop words.

The Quick! bRoWn foX juMPS.

quick brown fox jumps

Depending on the NLP project, some steps are compulsory

The END Day 1

DSN LEKKI-AJAH
BY ABEREJO HABEEBLAH O.
X: @HABEREJO

Next Class schedule:
Tuesday, 11th March, by 5:00pm.

Welcome Here!!

NATURAL LANGUAGE PROCESSING (NLP)

TEXT PREPROCESSING AND TOKENIZATION

DSN LEKKI-AJAH
BY ABEREJO HABEEBLAH O.
X (TWITTER): @HABEREJO

Day 2

Tuesday, 11th March, 2025

Why are we here

To equip learners with the skills to understand how Artificial Intelligence models are built, explore Machine Learning and Deep Learning, and focus on Natural Language Processing (NLP).

We should build some project right?

Yes!!

If you are in with me,

We should build some project.

Classes Structure

Every Tuesday throughout the month of March.

4 classes in total.

Each class for about 1:45 minutes

Days are NOT likely to change, although, my schedule can be OUCH, I wil reachout earlier before. Understand me bikoooo

Who should be here

- You're curious and ready to learn something new
- You dream of becoming a Data Scientist.
- You're passionate about building the future as a Machine Learning Engineer.
- You have a basic understanding of Python programming.
- You've worked with data and want to take your skills to the next level.
- You're a researcher exploring the exciting world of AI.

which of these are you?

Who should be here

- You're familiar with the fundamentals of Machine Learning and Deep Learning.
- You love problem solving and are excited to tackle real world problems with AI.
- You want to understand the technology that is shaping the future.
- You're driven by a desire to learn and grow in the field of AI.
- You enjoy collaborating and learning from others.

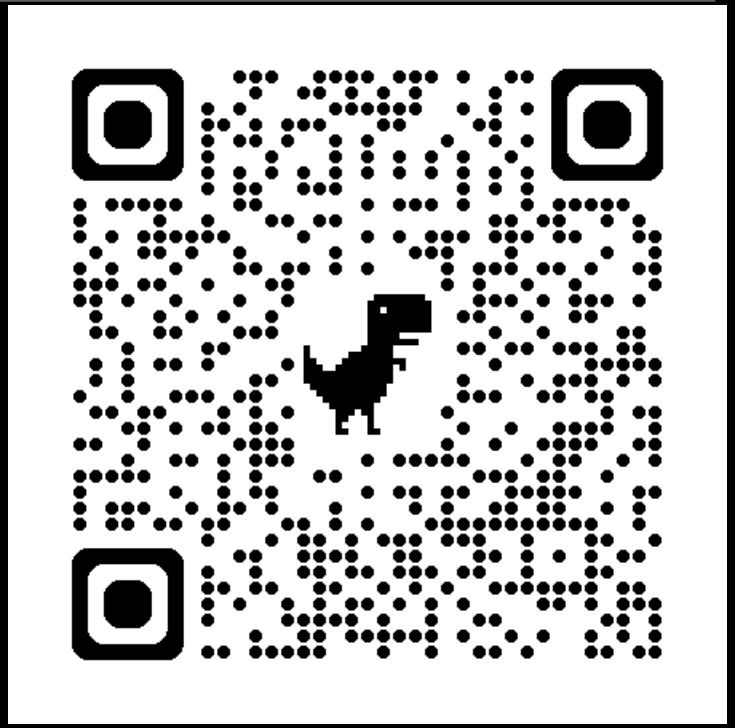
Which of these are you?????
You ticked any of these boxes?
Let's Goooooooo

Before we begin

Who I am?

- A Machine Learning Engineer
- A young MAN eager to master AI

FUN FACT:
I've taught over 100 people Python, but
I'm still learning new things every day.
(Especially how to avoid typos when
coding late at night!)



Scan to visit my portfolio
or
https://bheez.netlify.app

Quick Recap

1. **Understanding AI, ML, DL, and NLP**

2. **Introduction to NLP**

3. **NLP Use Cases:** Translation Apps 🌍, Spam Filters 📧, Search Engines 🔍, Sentiment Analysis 😀😡, Voice Assistants 🎙️, Chatbots 💬, Autocorrect & Predictive Text 📱.

4. **How Computer Understands Text:** 1. Text Preprocessing 2. Tokenization, 3. Numerical Representation (Word Embedding)

5. **Text Preprocessing Techniques:** Lowercasing, Removing Punctuation Marks, Removing Stopwords (i.e. like A, of, in, etc), Stemming / Lemmatization.

Course Structure

1. Text Preprocessing Techniques
2. Stemming and Lemmatization
3. Hands-on NLP Tasks
4. Brain Teasers & Discussion
5. Tokenization: **Sentence Tokenization** (Sentence-Level Tokenization)
   - **Word Tokenization** (Word-Level Tokenization)
   - **Subword Tokenization**
   - **Character-level Tokenization**
6. **Numerical Representation:** Vocabulary and Integer Encoding
7. **Numerical Representation:** One-Hot Encoding

And lots more

Step 1: Text Preprocessing

4. Stemming / Lemmatization:

Stemming: Stemming is the truncation of words down to their stem, **i.e their root form.**

For example, the words **house**, **housed** and **housing** both have the stem **hous**.

With smaller datasets in particular, stemming can be productive because it pools words with similar meanings into a single token.

The Quick! bRoWn foX juMPS.

quick brown fox jumps

Depending on the NLP project, some steps are compulsory

Step 1: Text Preprocessing

There will be more examples of this stemmed token's context, enabling techniques like **word2vec** or **GloVe** to more accurately identify an appropriate location for the token in word-vector space

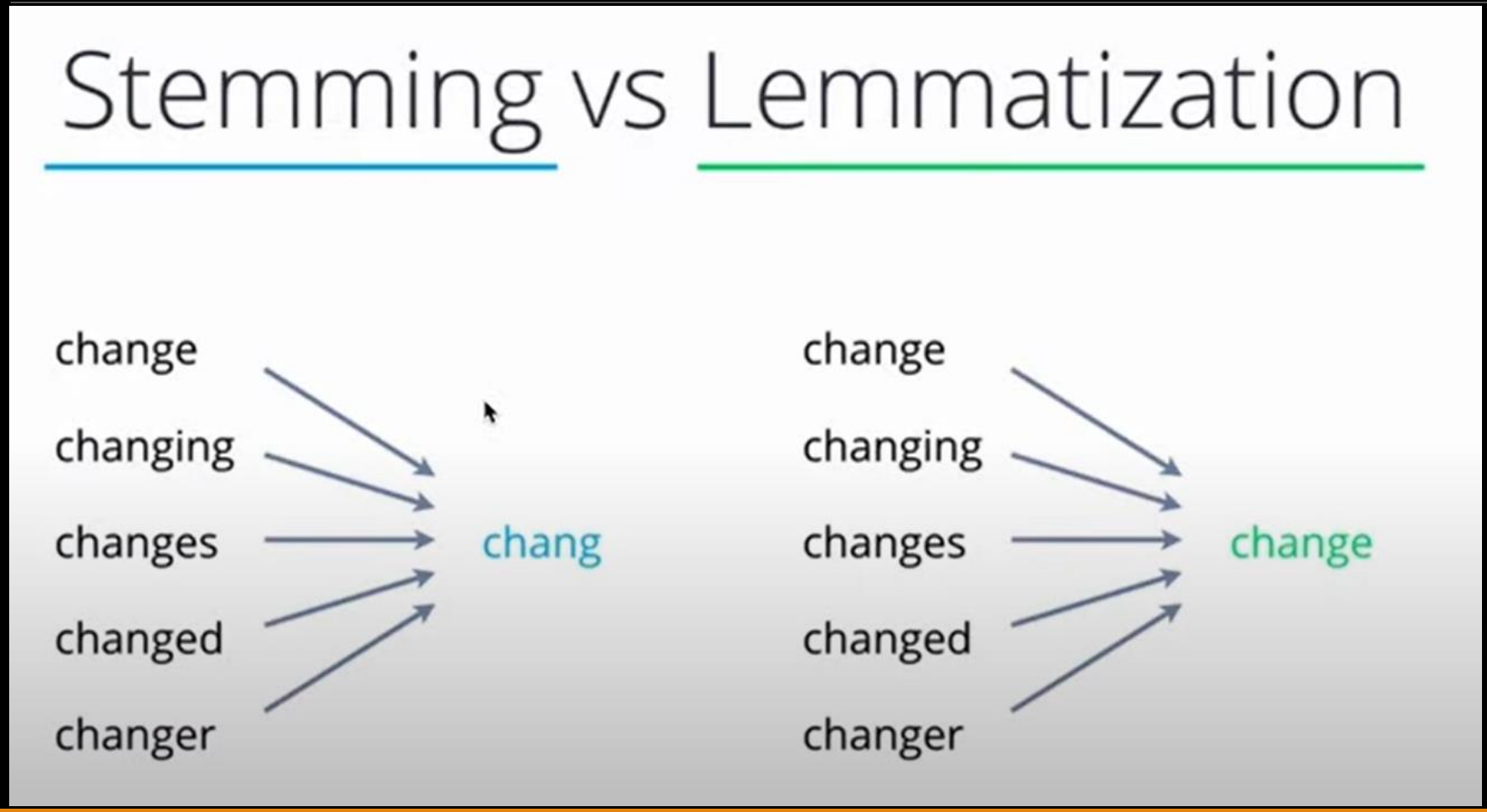
To stem words, you can use the Porter algorithm5 provided by **nltk**. To do this, you create an instance of a **PorterStemmer()** object and then add its stem() method

The Quick! bRoWn foX juMPS.

quick brown fox jumps

NLTK == Python Natural Language Toolkit. It's a package

Step 1: Text Preprocessing



From this comparison, Stemming returns to its root form. Stemming is quite harsh

While

Lemmatization is quite flexible and return it to a common word.

Choose based on the kind of words you have and your project.

Step 1: Text Preprocessing

🧐 Difference Between Stemming and Lemmatization

Both **stemming and lemmatization** reduce words to their **root form**, but they work differently!

Feature	Stemming 🪓	Lemmatization 📖
How It Works	Chops off word endings based on rules (heuristics).	Uses a dictionary to find the actual base form of a word.
Speed	Fast but less accurate.	Slower but more accurate.
Output	May not return a real English word.	Always returns a real word.
Use Case	Search engines, keyword extraction.	NLP tasks requiring correct meaning, like chatbots and sentiment analysis.
Example 1	"running" → "run"	"running" → "run"
Example 2	"happiness" → "happi" ❌	"happiness" → "happiness" ✅
Example 3	"better" → "better" ❌	"better" → "good" ✅

Stemming follows simple rules to cut off suffixes like -ing, -ed, or -ly.

Lemmatization uses linguistic knowledge to find the true root form of a word.

Step 1: Text Preprocessing

0	I absolutely LOVE this product!! ❤️ It's super efficient and really worth the money. Definitely recommend! 👍	i absolutely love this product!! ❤️ it's super efficient and really worth the money. definitely recommend! 👍	i absolutely love this product its super efficient and really worth the money definitely recommend	absolutely love product super efficient really worth money definitely recommend	absolut love product super effici realli worth money definit recommend	absolutely love product super efficient really worth money definitely recommend
1	Worst purchase ever... 😡 Waste of money. DO NOT BUY!! Full of issues.	worst purchase ever... 😡 waste of money. do not buy!! full of issues.	worst purchase ever waste of money do not buy full of issues	worst purchase ever waste money buy full issues	worst purchas ever wast money buy full issu	worst purchase ever waste money buy full issue
2	This product does what it says, but nothing special. 🤷 It's okay for the price, I guess.	this product does what it says, but nothing special. 🤷 it's okay for the price, i guess.	this product does what it says but nothing special its okay for the price i guess	product says nothing special okay price guess	product say noth special okay price guess	product say nothing special okay price guess
3	AMAZING quality and fast shipping!!! 🚀🔥 #satisfied #fastdelivery	amazing quality and fast shipping!!! 🚀🔥 #satisfied #fastdelivery	amazing quality and fast shipping satisfied fastdelivery	amazing quality fast shipping satisfied fastdelivery	amaz qualiti fast ship satisfi fastdeliveri	amazing quality fast shipping satisfied fastdelivery
4	Terrible! Had high expectations, but it broke in a week. Really disappointed. 😞	terrible! had high expectations, but it broke in a week. really disappointed. 😞	terrible had high expectations but it broke in a week really disappointed	terrible high expectations broke week really disappointed	terribl high expect broke week realli disappoint	terrible high expectation broke week really disappointed
5	This phone is great 📱, but the battery drains too fast. 🔋😕	this phone is great 📱, but the battery drains too fast. 🔋😕	this phone is great but the battery drains too fast	phone great battery drains fast	phone great batteri drain fast	phone great battery drain fast
6	I love how easy it is to use! 😍 Definitely a game-changer.	i love how easy it is to use! 😍 definitely a game-changer.	i love how easy it is to use definitely a gamechanger	love easy use definitely gamechanger	love easi use definit gamechang	love easy use definitely gamechanger
7	Do not buy this laptop! 👎 It crashes every 10 minutes. So frustrating! 😡	do not buy this laptop! 👎 it crashes every 10 minutes. so frustrating! 😡	do not buy this laptop it crashes every 10 minutes so frustrating	buy laptop crashes every 10 minutes frustrating	buy laptop crash everi 10 minut frustrat	buy laptop crash every 10 minute frustrating
8	The camera quality is excellent! 📸 Love the night mode. 🌙✨	the camera quality is excellent! 📸 love the night mode. 🌙✨	the camera quality is excellent love the night mode	camera quality excellent love night mode	camera qualiti excel love night mode	camera quality excellent love night mode
9	Meh... the product is just average. 😐 I expected more for this price.	meh... the product is just average. 😐 i expected more for this price.	meh the product is just average i expected more for this price	meh product average expected price	meh product averag expect price	meh product average expected price
10	Great customer service! 🙌 They replaced my faulty item within 24 hours.	great customer service! 🙌 they replaced my faulty item within 24 hours.	great customer service they replaced my faulty item within 24 hours	great customer service replaced faulty item within 24 hours	great custom servic replac faulti item within 24 hour	great customer service replaced faulty item within 24 hour

Love it Right????, I love it too

Some Discussion

Text -> Turn into numbers -> build a model -> train the model to find patterns -> use patterns (make predictions)

That's the flow

📌 Everything in a tip

You've successfully completed text preprocessing on customer feedback data. We've covered:

- ✔ Lowercasing – Ensuring consistency in text.
- ✔ Removing Punctuation & Emojis – Cleaning unnecessary characters.
- ✔ Stopword Removal – Keeping only meaningful words.
- ✔ Stemming & Lemmatization – Reducing words to their root form for better NLP.

🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥

Let's move on

Text -> turn into numbers -> build a model -> train the model to find patterns -> use patterns (make predictions)

If you understand everything up to this point, identify yourself as a Machine Learning Engineer specializing in Natural Language Processing!

😀 😀 😀 😀

QUESTIONS AND ANSWER

🧠 Time for quick Brain Teaser🔍

We've cleaned, transformed, and optimized text like real NLP pros! Now, let's see how well you followed along.🧐

Question: Given this raw text:👉 **"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?“**

If we apply the following preprocessing steps, what will be the output at each stage?

1️⃣ Lowercasing: What does the text look like after converting to lowercase?

🧠 Time for quick Brain Teaser🔍

1️⃣ Lowercasing: What does the text look like after converting to lowercase?

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?“

✅ "i'm loving this nlp session!!! 😍🔥 it's amazing how we can clean text properly. but... is it always the best approach?"

🧠 Time for quick Brain Teaser🔍

Question: Given this raw text:👉

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?“

If we apply the following preprocessing steps, what the output?

2️⃣ Removing Punctuation & Emojis: What’s left after getting rid of #,!, 😍🔥 , and other punctuation marks?

🧠 Time for quick Brain Teaser🔍

2️⃣ Removing Punctuation & Emojis: What's left after getting rid of #,!, 😍🔥, and other punctuation marks?

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

✅ "im loving this nlp session its amazing how we can clean text properly but is it always the best approach"

🧠 Time for quick Brain Teaser🔍

Question: Given this raw text:👉

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?“

If we apply the following preprocessing steps, what the output?

3️⃣ Stopword Removal: Which words will be removed, and what remains?

🧠 Time for quick Brain Teaser🔍

3️⃣ Stopword Removal: Which words will be removed, and what remains?

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?“

✅ "loving nlp session amazing clean text properly always best approach"

🧠 Time for quick Brain Teaser🔍

Question: Given this raw text:👉

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?“

If we apply the following preprocessing steps, whats the output?

4️⃣ Stemming & Lemmatization: If we use:

Porter Stemmer, how would words like "loving" and "amazing" change?

Lemmatization, what would be the base form of "loving" and "clean"?

🧠 Time for quick Brain Teaser🔍

4️⃣ Stemming & Lemmatization: If we use:

Porter Stemmer, how would words like "loving" and "amazing" change?

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

loving" → "love"
"amazing" → "amaz"
"clean" → "clean" (remains the same)
✅ "love nlp session amaz clean text proper always best approach"

🧠 Time for quick Brain Teaser🔍

4️⃣ Stemming & Lemmatization: If we use: Lemmatization (Using WordNet Lemmatizer), what would be the base form of "loving" and "clean"?

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

"loving" → "love"
"amazing" → "amazing" (unchanged)
"clean" → "clean"
✅ "love nlp session amazing clean text properly always best approach"

🧠 🚀 Bonus Question: 🔍

We removed **Stopwords** to clean up our text. But can you think of a situation where removing **Stopwords** might actually hurt the meaning or usefulness of our data?🤔

What can be done in such case???

💡 Hint: Think about cases where small words like "not," "is," or "to" are important in understanding the sentence.

Open-ended answers are welcomed!!!

🧠 🚀 Bonus Question: 🔍

Answer: ✅ Yes! There are several cases where stopwords play a crucial role:

1️⃣ **Sentiment Analysis** – Words like "not" and "very" can change sentiment.
Example: "The movie is not bad" (removing "not" changes the meaning).

2️⃣ **Question Answering –** Stopwords help retain context in queries.
Example: "What is the capital of France?" makes more sense than just "capital France?''

3️⃣ **Machine Translation** – Removing stopwords might lead to incorrect translations.
Example: Translating "I am going to the market" without "am" and "to" could alter the meaning.

So, while removing stopwords often helps clean data, it's important to consider the task before blindly dropping them! 🚀

🧠 Try This

💡 Try This!

Modify the dataset with your own text and observe how preprocessing affects different types of input.

Generate into a CSV File and share before the end of the program.

Let's keep building! 🔥🔥🔥

Let's move on

Text -> turn into numbers -> build a model -> train the model to find patterns -> use patterns (make predictions)

If you get the questions and attempts right to this extent, identify yourself as a Machine Learning Engineer specializing in Natural Language Processing!

😀 😀 😀 😀

Step 2: Tokenization

What is Tokenization?

Tokenization is the process of **splitting** a **text** into individual units, called **tokens**.

These tokens can be words, subwords, characters, or even punctuation marks. It's an important step in most NLP pipelines, as it prepares the text for further processing.

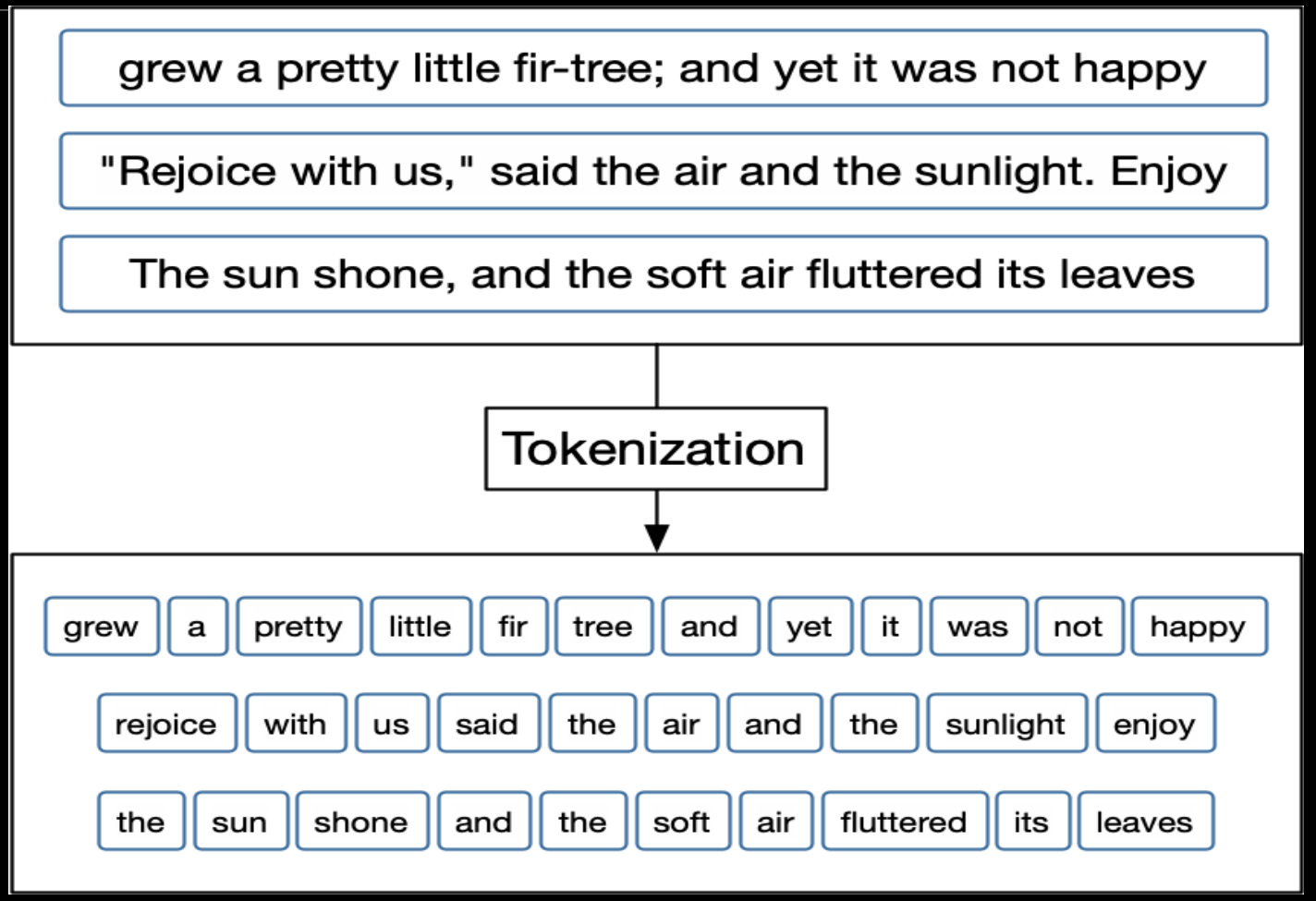
Step 2: Tokenization

What is a Token?

Tokens make it easier for computers to analyze text.

E.G: **grew** is a token.

Pretty is a token



Step 2: Tokenization

There are different types of tokenization:

- **Sentence Tokenization** (Sentence-Level Tokenization)
- **Word Tokenization** (Word-Level Tokenization)
- **Subword Tokenization**
- **Character-level Tokenization**

Step 2: Tokenization

- **Sentence Tokenization** (Sentence-Level Tokenization): Splits text into individual sentences. Important for tasks like text summarization or machine translation.

Example:

"The cat sat on the mat. The dog barked."

becomes

["The cat sat on the mat.", "The dog barked."]

- Uses the `sent_tokenize()` function of the nltk.tokenize library

Step 2: Tokenization

Code sample:

```
import nltk
nltk.download('punkt')  # Download sentence tokenizer data
#nltk.download('punkt_tab') # Download the 'punkt_tab' resource
from nltk.tokenize import sent_tokenize

text = "The cat sat on the mat. The dog barked."

sentences = sent_tokenize(text)

print("Sentences:", sentences)
```

```
Sentences: ['The cat sat on the mat.', 'The dog barked.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Step 2: Tokenization

Word Tokenization (Word-Level Tokenization): Splits text into individual words. This is the most common type.

Example:

"The cat sat on the mat. The dog barked."

becomes

['The', 'cat', 'sat', 'on', 'the', 'mat', '.', 'The', 'dog', 'barked', '.']

-Uses the `word_tokenize()` function of the nltk.tokenize library

Step 2: Tokenization

Code sample:

```
import nltk
nltk.download('punkt')  # Download sentence tokenizer data
#nltk.download('punkt_tab') # Download the 'punkt_tab' resource
from nltk.tokenize import word_tokenize

text = "The cat sat on the mat. The dog barked."

word = word_tokenize(text)

print("Word:", word)
```

```
Word: ['The', 'cat', 'sat', 'on', 'the', 'mat', '.', 'The', 'dog', 'barked', '.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Step 2: Tokenization

Word tokenization and other examples are great, but what happens when our computer encounters a word it hasn't seen before?

Imagine you're learning English and you come across the word 'unbreakable.' You might not know it, but you know 'break' and 'able'.

Think of learning a new language. You might not know every single word, but you can often figure out the meaning of a new word if you recognize parts of it, Here is where Subword Tokenization comes in.

Subword tokenization is a way to handle these **'out-of-vocabulary'** (OOV) words. It breaks words into smaller parts, called **subwords**, so the computer can understand them even if it hasn't seen the whole word before.

Step 2: Tokenization

Subword Tokenization: Splits words into smaller units (subwords). This is crucial for handling out-of-vocabulary words and morphologically rich languages. Common methods include **Byte Pair Encoding (BPE)** and **WordPiece.**

Example:

"unbreakable"

might be tokenized as

["un", "break", "able"].

This allows the model to understand the meaning even if it hasn't seen "unbreakable" before, as it recognizes "break," "un," and "able."

Step 2: Tokenization

Code sample:

```
# Subword Tokenization (Byte Pair Encoding - BPE)
# Example using subword tokenization with Hugging Face tokenizers
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased") # Loads BERT's tokenizer (which uses BPE)
# Apart from BERT, i.e bert-base-uncased, there are other models like
# en_core_web_sm model used with SpaCy,
# SentencePiece by Google
# Even GPT 3 and 4 can be used for tokenization with an API in a way.

text = "I have a new unbreakable toy."
tokens = tokenizer.tokenize(text)
print(tokens) # Output will show how "unbreakable" is broken into subwords
```

```
['i', 'have', 'a', 'new', 'un', '##break', '##able', 'toy', '.']
```

Step 2: Tokenization

Character-level Tokenization: Splits text into individual characters. Useful for languages with complex morphology or when dealing with misspellings.

Example:

"cat"

becomes

["c", "a", "t"]

Character-level Tokenization are useful in Spelling Corrections, New text Generation as learned from patterns of character sequences and also when handling noisy data.

Step 2: Tokenization

Code sample:

```python
# Exaplaining characoer-level tokenization

def character_tokenize(text):
    """Tokenizes text into individual characters."""
    return list(text)

def character_frequency(text):
    """Calculates the frequency of each character in the text."""
    tokens = character_tokenize(text)
    frequency = {}
    for token in tokens:
        if token in frequency:
            frequency[token] += 1
        else:
            frequency[token] = 1
    return frequency

# Sample text
text = "Hello, world! 123"

# Tokenize and calculate frequency
tokens = character_tokenize(text)
freq = character_frequency(text)

# Print results
print("Tokens:", tokens)
print("Character Frequency:", freq)
```

```
Tokens: ['H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!', ' ', '1', '2', '3']
Character Frequency: {'H': 1, 'e': 1, 'l': 3, 'o': 2, ',': 1, ' ': 2, 'w': 1, 'r': 1, 'd': 1, '!': 1, '1': 1, '2': 1, '3': 1}
```

Step 2: Tokenization

While character-level tokenization offers advantages, it also has drawbacks:

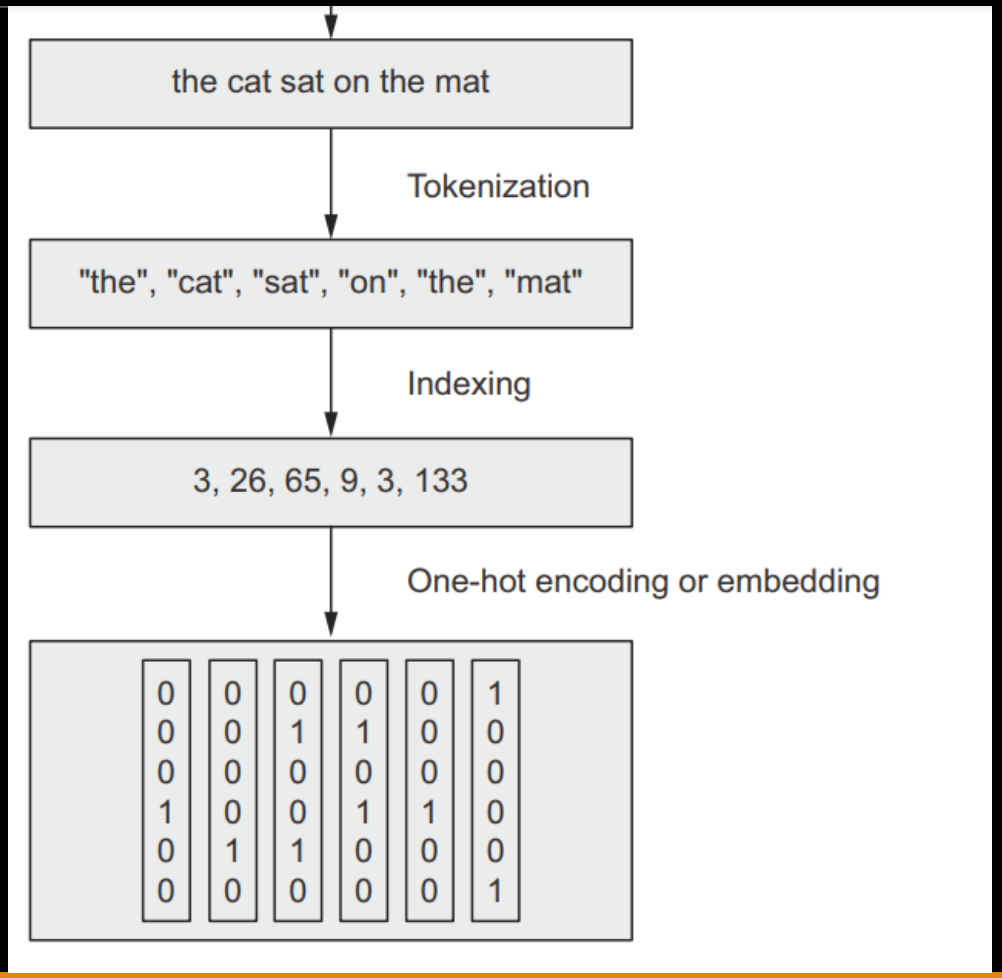
Longer Sequences: Character sequences are much longer than word sequences, which can make training models more computationally expensive.

Loss of Word-Level Meaning: It doesn't capture the semantic meaning of words as effectively as word-level or subword tokenization.

Step 3: Numerical Representation

The next step, which is almost always done immediately after tokenization, is to convert those **tokens** into **numerical representations.**

So, this step converts the **tokens** into **numbers**.



Step 3: Numerical Representation

Why is Numerical Representation necessary:

Machine learning models (and deep learning models) operate on numbers. They cannot directly process text.

The numerical representation is the bridge between human-readable text and machine-understandable data.

Almost every NLP task, requires a numerical representation of text. Without it, we can't apply any machine learning algorithms.

Step 3: Numerical Representation

How does Numerical Representation Works in NLP Projects:

Preprocessing: You'd typically tokenize your text and immediately convert the tokens to numbers (using one of the mentioned in a slide before) before feeding the data to your model.

Model Input: The numerical representation (integer encoded, one-hot encoded, or embeddings) becomes the input to your NLP model.

Model Training: The model learns patterns in these numbers, which correspond to patterns in the original text.

Step 3: Numerical Representation

There are several ways to do this:

a) Word Embeddings (aka Word Vectors) which uses techniques like **Word2Vec**, **GloVe**, **FastText**, **BERT**, **or by creating an embedding layer and training** (i.e creating your model to be used for embedding).

b) One-Hot Encoding

c) Vocabulary and Integer Encoding

Step 3: Numerical Representation

Vocabulary and Integer Encoding

This method of Numerical Representation convert token to numbers in three steps

a) **Build a Vocabulary:** Create a list of all **unique tokens** in your training data. This list is your vocabulary.

"The cat sat on the mat. The dog barked."

becomes

["the", "cat", "sat", "on", "mat", ".", "dog", "barked"]

NB: Notice that the word **THE** appeared just once in the generated list called **Vocabulary**

Step 3: Numerical Representation

b) Integer Encoding: Assign a unique integer to each token in the vocabulary.

i.e: The == 0, cat == 1, sat == 2, on == 3, mat == 4, . == 5, dog == 6, barked == 7

c) Convert Tokens to Integers: Replace each token in your tokenized text with its corresponding integer ID from the vocabulary.

i.e [0, 1, 2, 3, 0, 4, 5, 0, 6, 7]

Step 3: Numerical Representation

Example: Let's say the sentence is "The cat sat on the mat. The dog barked.“.

Stage 1: Vocabulary is: ["the", "cat", "sat", "on", "mat", ".", "dog", "barked"]

Stage 2: The == 0, cat == 1, sat == 2, on == 3, mat == 4, . == 5, dog == 6, barked == 7

Stage 3: encoded token to integer becomes: [0, 1, 2, 3, 0, 4, 5, 0, 6, 7]

NB: The "The" and "the" are normally considered different tokens (unless you do lowercasing first, which was expected to have been done in this regard).

Each word is assigned a unique integer ID.

Step 3: Numerical Representation

The **Vocabulary and Integer Encoding** method is simple to implement and understand, but doesn't capture **semantic relationships**, therefore not so used in the industry like the other two methods.

Semantic Relationship means how words relates, e.g King and Prince, King and Queen.

Step 3: Numerical Representation

One-Hot Encoding

This is a traditional approach to encoding natural language numerically for processing it with a machine.

In this approach, the words of natural language in a sentence **(e.g. The bat sat on the cat:- "the", "bat", "sat", "on", "the" and "cat")** are represented by the columns of a matrix.

Each row in the matrix, meanwhile, represents a unique word.

Step 3: Numerical Representation

Cells within one-hot matrices consist of binary values, that is, they are a 0 or a 1.

Each column contains at most a single 1, but is otherwise made up of 0s.

In this example, the entire corpus has only **six** words, **five** of which are unique.

The bat sat on the cat.

words						
the	1	0	0	0	1	0
bat	0	1	0	0	0	0
on	0	0	0	1	0	0
⋮						
n_{unique_words}						

Step 3: Numerical Representation

In this approach,

If there are 100 unique words across the corpus of documents you're feeding into your natural language algorithm, then your matrix of one-hot-encoded words will have 100 rows.

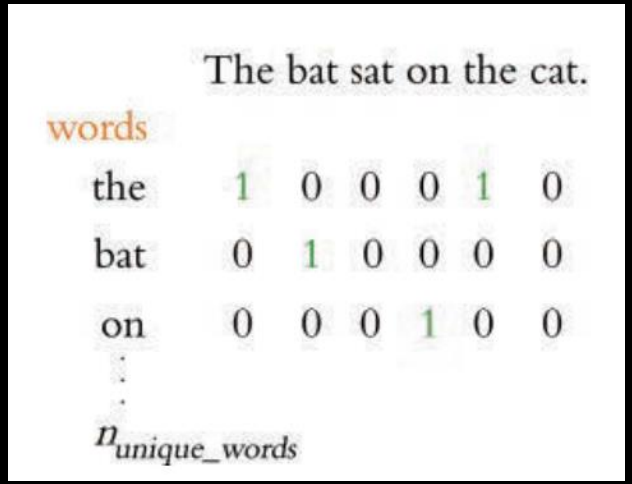
If there are 1,000 unique words across your corpus, then there will be 1,000 rows in your one-hot matrix, and so on.

Step 3: Numerical Representation

One-hot word representations like this are fairly straightforward, and they are an acceptable format for feeding into a deep learning model (or, indeed, other machine learning models).

However, the **simplicity and sparsity of one-hot representations are limiting when incorporated into a natural language application.**

It's can be used for **simpler NLP tasks** or as a basic representation. It's also simple but doesn't capture **semantic relationships and therefore not the best option for major NLP projects.**



QUESTIONS AND ANSWER

The END Day 2

DSN LEKKI-AJAH
BY ABEREJO HABEEBLAH O.
X: @HABEREJO

Next Class schedule:
Tuesday, 18th March, by 5:00pm.

Welcome Here!!

NATURAL LANGUAGE PROCESSING (NLP)

NUMERICAL REPRESENTATION (WORD EMBEDDING) & RECURRENT NEURAL NETWORK

DSN LEKKI-AJAH

BY ABEREJO HABEEBLAH O.

X: @HABEREJO

Day 3

Tuesday, 18th March, 2025

Why are we here

To equip learners with the skills to understand how Artificial Intelligence models are built, explore Machine Learning and Deep Learning, and focus on Natural Language Processing (NLP).

We should build some project right?

Yes!!

If you are in with me,

We should build some project.

Classes Structure

Every Tuesday throughout the month of March.

4 classes in total. This is the third class

Each class for about 1:45 minutes

Days are NOT likely to change, although, my schedule can be OUCH, I wil reachout earlier before. Understand me bikoooo

Who should be here

- You're curious and ready to learn something new
- You dream of becoming a Data Scientist.
- You're passionate about building the future as a Machine Learning Engineer.
- You have a basic understanding of Python programming.
- You've worked with data and want to take your skills to the next level.
- You're a researcher exploring the exciting world of AI.

which of these are you?

Who should be here

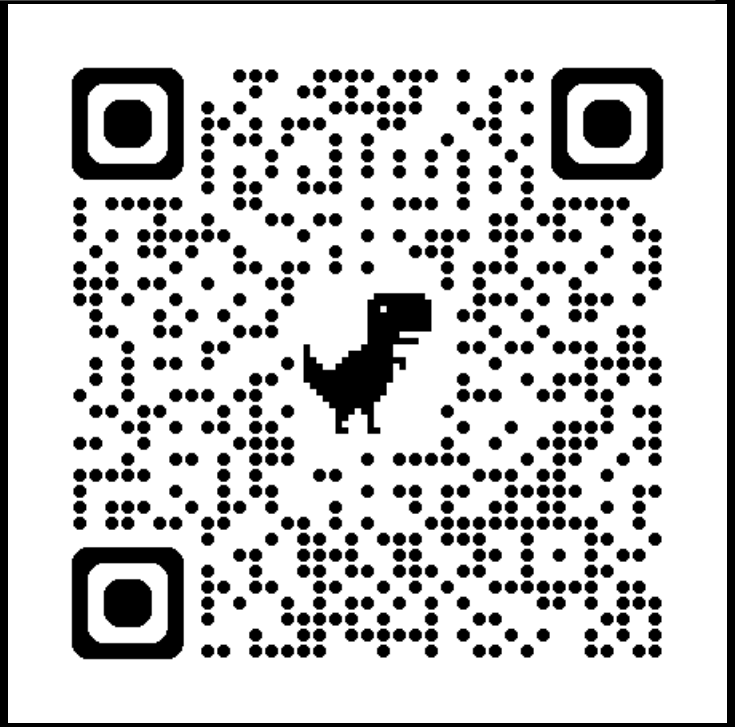
- You're familiar with the fundamentals of Machine Learning and Deep Learning.
- You love problem solving and are excited to tackle real world problems with AI.
- You want to understand the technology that is shaping the future.
- You're driven by a desire to learn and grow in the field of AI.
- You enjoy collaborating and learning from others.

Which of these are you?????
You ticked any of these boxes?
Let's Goooooooo

Who I am?

- A Machine Learning Engineer
- A young MAN eager to master AI

FUN FACT:
I've taught over 100 people Python, but
I'm still learning new things every day.
(Especially how to avoid typos when
coding late at night!)



Scan to visit my portfolio
or
https://bheez.netlify.app

Quick Recap (First Class)

1️⃣ **Understanding AI, ML, DL, and NLP**

2️⃣ **Introduction to NLP**

3️⃣ **NLP Use Cases:** Translation Apps 🌍, Spam Filters 📧, Search Engines 🔍, Sentiment Analysis 😀😡, Voice Assistants 🎙️, Chatbots 💬, Autocorrect & Predictive Text 📱.

4️⃣ **How Computer Understands Text:** 1. Text Preprocessing 2. Tokenization, 3. Numerical Representation (Word Embedding)

5️⃣ **Text Preprocessing Techniques:** Lowercasing, Removing Punctuation Marks, Removing Stopwords (i.e. like A, of, in, etc), Stemming / Lemmatization.

Before we begin

Quick Recap (Second Class)

1. **How Computer Understands Text:** 1. Text Preprocessing
2. **Text Preprocessing Techniques:** Lowercasing, Removing Punctuation Marks, Removing Stopwords (i.e. like A, of, in, etc), Stemming / Lemmatization.
3. **How Computer Understands Text:** 2. Tokenization
4. Tokenization: **Sentence Tokenization** (Sentence-Level Tokenization)

   Word Tokenization (Word-Level Tokenization)

   Subword Tokenization

   Character-level Tokenization
5. **How Computer Understands Text:** 3. Numerical Representation
6. **Numerical Representation:** Vocabulary and Integer Encoding
7. **Numerical Representation:** One Hot Encoding

Course Structure

1. Numerical Representation: Word Embedding
2. Working with **Word2Vec**
3. Working with **Glove**
4. Discussing **t-distributed stochastic neighbor embedding, t-SNE (pronounced tee-snee)**
5. **FastText**
6. **BERT** (bi-directional encoder representations from transformers)

And more

Step 3: Numerical Representation

There are several ways to do this:

a) Word Embeddings (aka Word Vectors) which uses techniques like **Word2Vec**, **GloVe**, **FastText**, **BERT**, **or by creating an embedding layer and training** (i.e creating your model to be used for embedding).

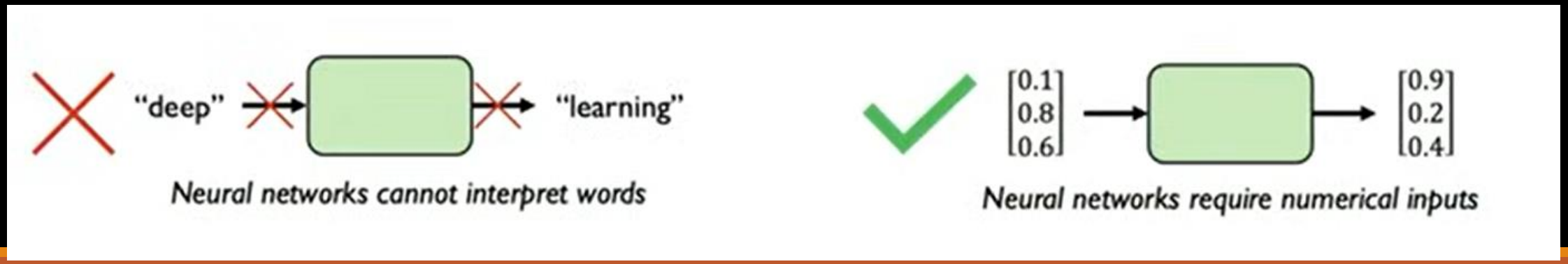
b) One-Hot Encoding

c) Vocabulary and Integer Encoding

Step 3: Numerical Representation: Word Embedding

Word Embeddings (aka Word Vectors): a specific and powerful type of numerical representation designed to capture **semantic meaning.**

Vector representations of words are the information-dense alternative to one-hot encodings of words. Whereas one-hot representations capture information about word location only, word vectors **(also known as word embeddings or vector-space embeddings)** capture information about word meaning as well as location.



Step 3: Numerical Representation: Word Embedding

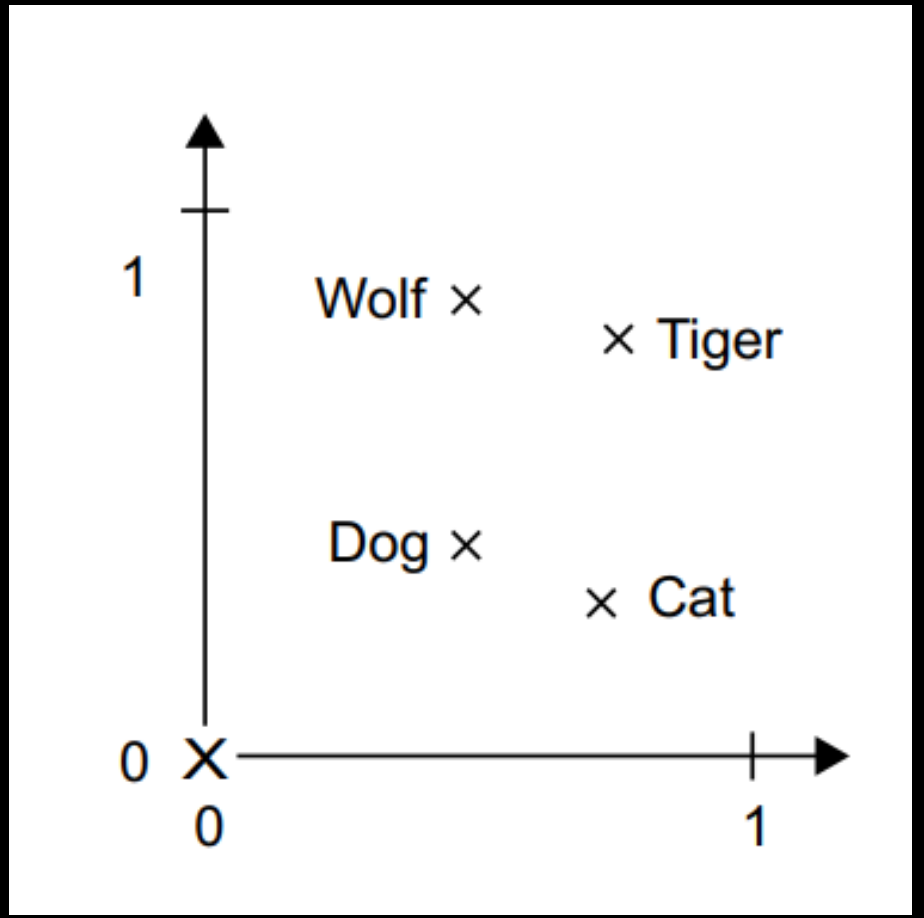
canine== dog like, with pointy tooth btw incisors. FELINE == cat like,

The key advantage, word vectors enable deep learning NLP models to automatically learn linguistic features.

For instance, four words are embedded on a 2D plane: cat, dog, wolf, and tiger.

The same vector allows us to go from **cat** to **tiger** and from **dog** to **wolf.** This vector could be interpreted as the **"from pet to wild animal"** vector.

Similarly, another vector lets us go from **dog** to **cat** and from **wolf** to **tiger**, which could be interpreted as a **"from canine to feline"** vector.



Step 3: Numerical Representation: Word Embedding

In real-world word-embedding spaces, common examples of meaningful geometric transformations are **"gender"** vectors and **"plural"** vectors.

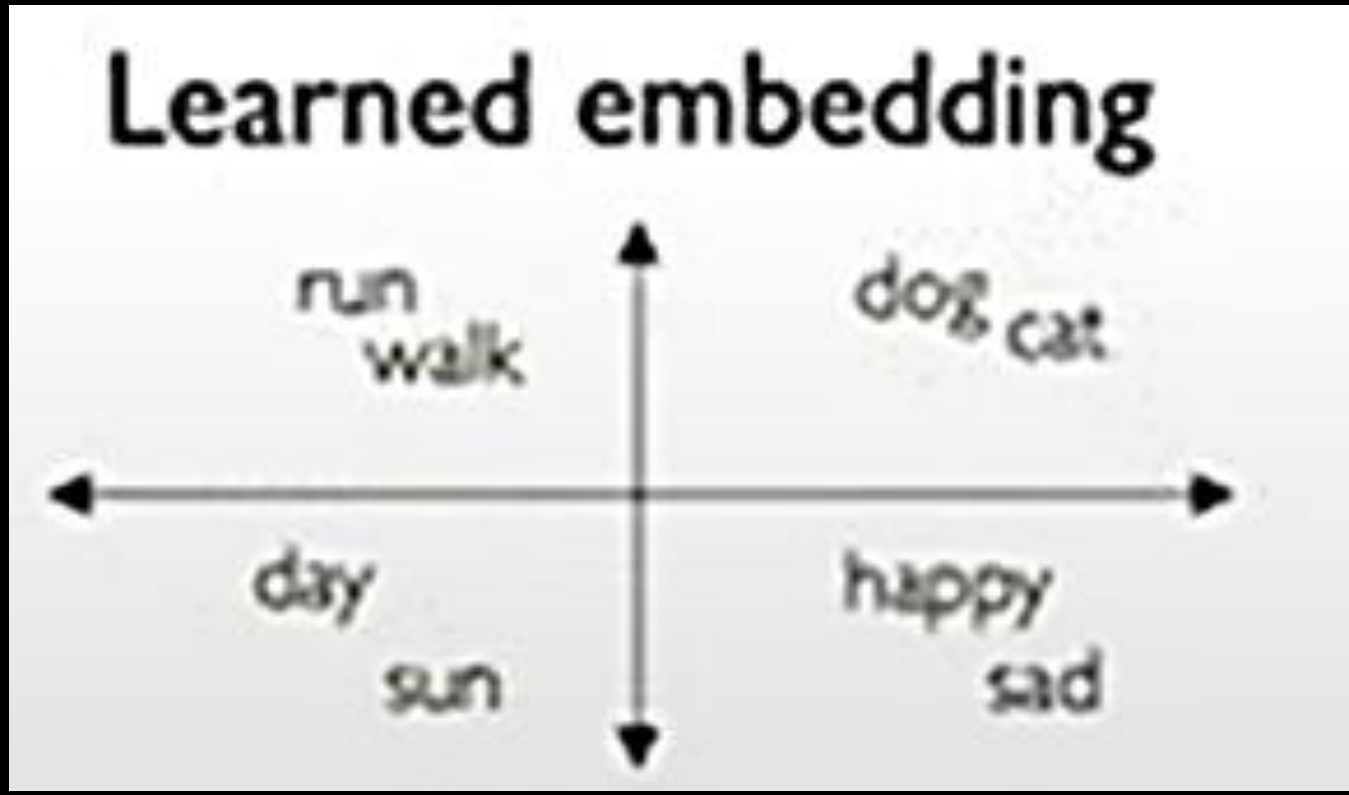
For instance, by adding a **"female"** vector to the vector **"king**," we obtain the vector **"queen."**

By adding a **"plural"** vector, we obtain **"kings."**

Remember one-hot encoding? It tells us where a word is in a list, but it doesn't tell us anything about its meaning. It's like having a list of cities but no map to show how they're related.

Step 3: Numerical Representation: Word Embedding

So basically, Word Embedding converts the words into number such that, things that are related to each other in language, should numerically be similar and close to each other in the space and things that are very dissimilar should numerically dissimilar and far from away in the space.



Step 3: Numerical Representation: Word Embedding

There are two ways to obtain word embeddings:

1. Learn word embeddings jointly with the main task you care about **(such as document classification or sentiment prediction)**. In this setup, **you start with random word vectors** and then learn word vectors in the same way you learn **the weights of a neural network**.

Weights of a neural neural??? Should I explain???

2. Load into your model word embeddings that were precomputed using a different machine learning task than the one you're trying to solve. These are called **pretrained word embeddings**. They include Word2Vec, GloVe, FastText, BERT. **Word2Vec, GloVe, are most popular.**

Step 3: Numerical Representation: Word Embedding

Keras is a framework like Tensorflow but lightweight

Why should we learn (or create) a new embedding space with every new task create out word embedding?

- Because, what makes a good word-embedding space depends heavily on your task
- The perfect word-embedding space for an **English-language movie-review sentiment-analysis model** may look different from the perfect embedding space for an **English-language legal-document classification model**, because the importance of certain semantic relationships varies from task
- It's thus reasonable to learn a new embedding space with every new task.
- Fortunately, backpropagation makes this easy, and **Keras** makes it even easier. It's about learning the weights of a layer.

Step 3: Numerical Representation: Word Embedding

Keras is a framework like Tensorflow but lightweight

Learning (or creating) a new embedding space.

- Instantiating an Embedding layer

```
embedding_layer = layers.Embedding(input_dim=max_tokens,
output_dim=256)
```

```
# Embedding layer: Converts integer sequences to dense vectors (e
embedded = layers.Embedding(input_dim=max_tokens, output_dim=256)
```

The Embedding layer takes at least two arguments, max_token & output_dim.

Parameters: a) input_dim: The number of unique tokens in your vocabulary (the size of your "word list").

b) output_dim: The dimensionality of the embeddings (how many numbers to represent each word).

Step 3: Numerical Representation: Word Embedding

```python
import tensorflow as tf  # Import TensorFlow library
from tensorflow import keras  # Import Keras module from TensorFlow
from tensorflow.keras import layers  # Import layers module from Keras

# --- Data Preparation (Dummy Data for Demonstration) --- # Define example parameters for the data
max_tokens = 10000  # Example vocabulary size (number of unique words/tokens)
input_length = 50   # Example sequence length (maximum length of input sequences)
batch_size = 32     # Batch size for training

# Create dummy training dataset using tf.data.Dataset
int_train_ds = tf.data.Dataset.from_tensor_slices(
    (tf.random.uniform(shape=(1000, input_length), minval=0, maxval=max_tokens, dtype=tf.int64),  # Input sequences (random integers)
     tf.random.uniform(shape=(1000,), minval=0, maxval=2, dtype=tf.int64))  # Target labels (random 0 or 1)
).batch(batch_size)  # Batch the dataset into batches of size batch_size

# Create dummy validation dataset using tf.data.Dataset
int_val_ds = tf.data.Dataset.from_tensor_slices(
    (tf.random.uniform(shape=(200, input_length), minval=0, maxval=max_tokens, dtype=tf.int64),  # Input sequences (random integers)
     tf.random.uniform(shape=(200,), minval=0, maxval=2, dtype=tf.int64))  # Target labels (random 0 or 1)
).batch(batch_size)  # Batch the dataset into batches of size batch_size

# --- Model Definition ---
# Define the input layer, taking integer sequences as input
inputs = keras.Input(shape=(None,), dtype="int64")  # shape=(None,) allows variable sequence lengths

# Embedding layer: Converts integer sequences to dense vectors (embeddings)
embedded = layers.Embedding(input_dim=max_tokens, output_dim=256)(inputs)  # 256-dimensional embeddings

# Bidirectional LSTM layer: Processes the embedded sequences in both forward and backward directions
x = layers.Bidirectional(layers.LSTM(32))(embedded)  # 32 units in each LSTM layer
```

In summary, we created a dummy training dataset and dummy validation dataset, understand that the dummy generated dataset are random integers which serves as tokenized dataset, therefore, I could move directly into the embedding layer for word embedding

Step 3: Numerical Representation: Word Embedding

Why should be use pretrained word embeddings?

- They are trained on massive datasets.
- Capture a lot of semantic information.
- Save training time.

Pretrained word embedding techniques include: Word2Vec, Global Vectors for Word Representation (GloVe), FastText, BERT and others.

Word2Vec, GloVe, are most popular.

NB

Step 3: Numerical Representation: Word Embedding

Word2Vec developed by **Tomas Mikolov at Google in 2013.**

Word2Vec learns by looking at the words surrounding each target word. It tries to predict the context.

Word2Vec dimensions capture specific semantic properties, such as gender.

NB

Step 3: Numerical Representation: Word Embedding

To Use Word2Vec:

Load the model

NB: Loading the data should take about 2gb of data.

```
import gensim.downloader as api  # Import the Gensim downloader for pre-trained models
from sklearn.manifold import TSNE  # Import t-SNE for dimensionality reduction (visualization)
import matplotlib.pyplot as plt  # Import Matplotlib for plotting

# --- 1. Load Pre-trained Word2Vec Model ---

# Load the pre-trained Word2Vec model trained on Google News dataset (300-dimensional vectors)
# 'word2vec-google-news-300' is the model identifier in Gensim's downloader
wv = api.load('word2vec-google-news-300')  # wv stands for word vectors
```

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

Step 3: Numerical Representation: Word Embedding

To Use Word2Vec:

Apply the model to some text:

As seen, similarity btw king and car is LOW

```
# --- 2. Example 1: Semantic Similarity ---

# Calculate and print the cosine similarity between word vectors
# Cosine similarity measures how similar two vectors are (between -1 and 1)
print(wv.similarity('king', 'queen'))  # Similarity between 'king' and 'queen' (should be high)
print(wv.similarity('man', 'woman'))  # Similarity between 'man' and 'woman' (should be high)
print(wv.similarity('king', 'man'))  # Similarity between 'king' and 'man' (should be high)
print(wv.similarity('king', 'car'))  # Similarity between 'king' and 'car' (should be low)
```

```
0.6510957
0.76640123
0.22942673
0.061895393
```

Step 3: Numerical Representation: Word Embedding

As seen, similarity btw king and car is LOW

To Use Word2Vec:

Let's give an analogy, Find words most similar to 'king' + 'woman' - 'man'. (This should be close to 'queen')

```
result = wv.most_similar(positive=['king', 'woman'], negative=['man'])
print(result)
```

RESULT: `[('queen', 0.7118193507194519), ('monarch', 0.6189674139022827), ('princess', 0.5902431011199951), ('crown_prince', 0.5499460697174072), ('prince', 0.5377321839332581), ('kings', 0.5236844420433044), ('Queen_Consort', 0.5235945582389832), ('queens', 0.5181134343147278), ('sultan', 0.5098593831062317), ('monarchy', 0.5087411999702454)]`

This demonstrates the ability of Word2Vec to solve analogy problems

Step 3: Numerical Representation: Word Embedding

To Use Word2Vec:

Let's give another analogy,

Find words most similar to 'king' + girl' + 'young' - 'man' – 'adult'.

What could this be??

```
result = wv.most_similar(positive=['king', 'girl', 'young'], negative=['man', 'adult'])
print(result)
```

Prince? King child? Princess?
Prince_Paras???

Step 3: Numerical Representation: Word Embedding

A Princess!!

Because, A princess relates to a king, is a girl, is young, is not a man, and not an adult.

```
result = wv.most_similar(positive=['king', 'girl', 'young'], negative=['man', 'adult'])
print(result)
```

```
[('princess', 0.4750033020973205 6), ('prince', 0.470218300819397), ('Prince_Paras', 0.4611
```

As seen, princess has the highest.

Step 3: Numerical Representation: Word Embedding

To Use Word2Vec:

To get the vector for a word:

```
# --- 4. Example 3: Getting the Vector for a Word ---

# Retrieve the vector representation of the word 'king'
vector_king = wv['king']
print(vector_king)  # Print the vector (300 numbers)
```

The Vector for the word KING is:

```
# Retrieve the vector representation of the word 'king'
vector_king = wv['king']
print(vector_king)  # Print the vector (300 numbers)
```

```
[ 1.25976562e-01  2.97851562e-02  8.60595703e-03  1.39648438e-01
 -2.56347656e-02 -3.61328125e-02  1.11816406e-01 -1.98242188e-01
  5.12695312e-02  3.63281250e-01 -2.42187500e-01 -3.02734375e-01
 -1.77734375e-01 -2.49023438e-02 -1.67968750e-01 -1.69921875e-01
  3.46679688e-02  5.21850586e-03  4.63867188e-02  1.28906250e-01
  1.36718750e-01  1.12792969e-01  5.95703125e-02  1.36718750e-01
  1.01074219e-01 -1.76757812e-01 -2.51953125e-01  5.98144531e-02
  3.41796875e-01 -3.11279297e-02  1.04492188e-01  6.17675781e-02
  1.24511719e-01  4.00390625e-01 -3.22265625e-01  8.39843750e-02
  3.90625000e-02  5.85937500e-03  7.03125000e-02  1.72851562e-01
  1.38671875e-01 -2.31445312e-01  2.83203125e-01  1.42578125e-01
  3.41796875e-01 -2.39257812e-02 -1.09863281e-01  3.32031250e-02
 -5.46875000e-02  1.53198242e-02 -1.62109375e-01  1.58203125e-01
 -2.59765625e-01  2.01416016e-02 -1.63085938e-01  1.35803223e-03
 -1.44531250e-01 -5.68847656e-02  4.29687500e-02 -2.46582031e-02
  1.85546875e-01  4.47265625e-01  9.58251953e-03  1.31835938e-01
  9.86328125e-02 -1.85546875e-01 -1.00097656e-01 -1.33789062e-01
 -1.25000000e-01  2.83203125e-01  1.23046875e-01  5.32226562e-02
 -1.77734375e-01  8.59375000e-02 -2.18505859e-02  2.05078125e-02
 -1.39648438e-01  2.51464844e-02  1.38671875e-01 -1.05468750e-01
  1.38671875e-01  8.88671875e-02 -7.51953125e-02 -2.13623047e-02
  1.72851562e-01  4.63867188e-02 -2.65625000e-01  8.91113281e-03
  1.49414062e-01  3.78417969e-02  2.38281250e-01 -1.24511719e-01
 -2.17773438e-01 -1.81640625e-01  2.97851562e-02  5.71289062e-02
 -2.89306641e-02  1.24511719e-02  9.66796875e-02 -2.31445312e-01
  5.81054688e-02  6.68945312e-02  7.08007812e-02 -3.08593750e-01
 -2.14843750e-01  1.45507812e-01 -4.27734375e-01 -9.39941406e-03
  1.54296875e-01 -7.66601562e-02  2.89062500e-01  2.77343750e-01
 -4.86373901e-04 -1.36718750e-01  3.24218750e-01 -2.46093750e-01
 -3.03649902e-03 -2.11914062e-01  1.25000000e-01  2.69531250e-01
  2.04101562e-01  8.25195312e-02 -2.01171875e-01 -1.60156250e-01
 -3.78417969e-02 -1.20117188e-01  1.15234375e-01 -4.10156250e-02
 -3.95507812e-02 -8.98437500e-02  6.34765625e-03  2.03125000e-01
  1.86523438e-01  2.73437500e-01  6.29882812e-02  1.41601562e-01
 -9.81445312e-02  1.38671875e-01  1.82617188e-01  1.73828125e-01
  1.73828125e-01 -2.37304688e-01  1.78710938e-01  6.34765625e-02
  2.36328125e-01 -2.08984375e-01  8.74023438e-02 -1.66015625e-01
 -7.91015625e-02  2.43164062e-01 -8.88671875e-02  1.26953125e-01
 -2.16796875e-01 -1.73828125e-01 -3.59375000e-01 -8.25195312e-02
 -6.49414062e-02  5.07812500e-02  1.35742188e-01 -7.47070312e-02
 -1.64062500e-01  1.15356445e-02  4.45312500e-01 -2.15820312e-01
 -1.11328125e-01 -1.92382812e-01  1.70898438e-01 -1.25000000e-01
  2.65502930e-03  1.92382812e-01 -1.74804688e-01  1.39648438e-01
  2.92968750e-01  1.13281250e-01  5.95703125e-02 -6.39648438e-02
  9.96093750e-02 -2.72216797e-02  1.96533203e-02  4.27246094e-02
 -2.46093750e-01  6.39648438e-02 -2.25585938e-01 -1.68945312e-01
  2.89916992e-03  8.20312500e-02  3.41796875e-01  4.32128906e-02
  1.32812500e-01  1.42578125e-01  7.61718750e-02  5.98144531e-02
 -1.19140625e-01  2.74658203e-03 -6.29882812e-02 -2.72216797e-02
 -4.82177734e-03 -8.20312500e-02 -2.49023438e-02 -4.00390625e-01
 -1.06933594e-01  4.24804688e-02  7.76367188e-02 -1.16699219e-01
  7.37304688e-02 -9.22851562e-02  1.07910156e-01  1.58203125e-01
  4.24804688e-02  1.26953125e-01  3.61328125e-02  2.67578125e-01
 -1.01074219e-01 -3.02734375e-01 -5.76171875e-02  5.05371094e-02
  5.26428223e-04 -2.07031250e-01 -1.38671875e-01 -8.97216797e-03
 -2.78320312e-02 -1.41601562e-01  2.07031250e-01 -1.58203125e-01
  1.27929688e-01  1.49414062e-01 -2.24609375e-02 -8.44726562e-02
```

Step 3: Numerical Representation: Word Embedding

The Vector for the word KING is:

[1.25976562e-01 2.97851562e-02 8.60595703e-03 1.39648438e-01 -2.56347656e-02 -3.61328125e-02 1.11816406e-01 -1.98242188e-01 5.12695312e-02 3.63281250e-01 -2.42187500e-01 -3.02734375e-01 -1.77734375e-01 -2.49023438e-02 -1.67968750e-01 -1.69921875e-01 3.46679688e-02 5.21850586e-03 4.63867188e-02 1.28906250e-01 1.36718750e-01 1.12792969e-01 5.95703125e-02 1.36718750e-01 1.01074219e-01 -1.76757812e-01 -2.51953125e-01 5.98144531e-02 3.41796875e-01 -3.11279297e-02 1.04492188e-01 6.17675781e-02 1.24511719e-01 4.00390625e-01 -3.22265625e-01 8.39843750e-02 3.90625000e-02 5.85937500e-03 7.03125000e-02 1.72851562e-01 1.38671875e-01 -2.31445312e-01 2.83203125e-01 1.42578125e-01 3.41796875e-01 -2.39257812e-02 -1.09863281e-01 3.32031250e-02 -5.46875000e-02 1.53198242e-02 -1.62109375e-01 1.58203125e-01 -2.59765625e-01 2.01416016e-02 -1.63085938e-01 1.35803223e-03 -1.44531250e-01 -5.68847656e-02 4.29687500e-02 -2.46582031e-02 1.85546875e-01 4.47265625e-01 9.58251953e-03 1.31835938e-01 9.86328125e-02 -1.85546875e-01 -1.00097656e-01 -1.33789062e-01 -1.25000000e-01 2.83203125e-01 1.23046875e-01 5.32226562e-02 -1.77734375e-01 8.59375000e-02 -2.18505859e-02 2.05078125e-02 -1.39648438e-01 2.51464844e-02 1.38671875e-01 -1.05468750e-01 1.38671875e-01 8.88671875e-02 -7.51953125e-02 -2.13623047e-02 1.72851562e-01 4.63867188e-02 -2.65625000e-01 8.91113281e-03 1.49414062e-01 3.78417969e-02 2.38281250e-01 -1.24511719e-01 -2.17773438e-01 -1.81640625e-01 2.97851562e-02 5.71289062e-02 -2.89306641e-02 1.24511719e-02 9.66796875e-02 -2.31445312e-01 5.81054688e-02 6.68945312e-02 7.08007812e-02 -3.08593750e-01 -2.14843750e-01 1.45507812e-01 -4.27734375e-01 -9.39941406e-03 1.54296875e-01 -7.66601562e-02 2.89062500e-01 2.77343750e-01 -4.86373901e-04 -1.36718750e-01 3.24218750e-01 -2.46093750e-01 -3.03649902e-03 -2.11914062e-01 1.25000000e-01 2.69531250e-01 2.04101562e-01 8.25195312e-02 -2.01171875e-01 -1.60156250e-01 -3.78417969e-02 -1.20117188e-01 1.15234375e-01 -4.10156250e-02 -3.95507812e-02 -8.98437500e-02 6.34765625e-03 2.03125000e-01 1.86523438e-01 2.73437500e-01 6.29882812e-02 1.41601562e-01 -9.81445312e-02 1.38671875e-01 1.82617188e-01 1.73828125e-01 1.73828125e-01 -2.37304688e-01 1.78710938e-01 6.34765625e-02 2.36328125e-01 -2.08984375e-01 8.74023438e-02 -1.66015625e-01 -7.91015625e-02 2.43164062e-01 -8.88671875e-02 1.26953125e-01 -2.16796875e-01 -1.73828125e-01 -3.59375000e-01 -8.25195312e-02 -6.49414062e-02 5.07812500e-02 1.35742188e-01 -7.47070312e-02 -1.64062500e-01 1.15356445e-02 4.45312500e-01 -2.15820312e-01 -1.11328125e-01 -1.92382812e-01 1.70898438e-01 -1.25000000e-01 2.65502930e-03 1.92382812e-01 -1.74804688e-01 1.39648438e-01 2.92968750e-01 1.13281250e-01 5.95703125e-02 -6.39648438e-02 9.96093750e-02 -2.72216797e-02 1.96533203e-02 4.27246094e-02 -2.46093750e-01 6.39648438e-02 -2.25585938e-01 -1.68945312e-01 2.89916992e-03 8.20312500e-02 3.41796875e-01 4.32128906e-02 1.32812500e-01 1.42578125e-01 7.61718750e-02 5.98144531e-02 -1.19140625e-01 2.74658203e-03 -6.29882812e-02 -2.72216797e-02 -4.82177734e-03 -8.20312500e-02 -2.49023438e-02 -4.00390625e-01 -1.06933594e-01 4.24804688e-02 7.76367188e-02 -1.16699219e-01 7.37304688e-02 -9.22851562e-02 1.07910156e-01 1.58203125e-01 4.24804688e-02 1.26953125e-01 3.61328125e-02 2.67578125e-01 -1.01074219e-01 -3.02734375e-01 -5.76171875e-02 5.05371094e-02 5.26428223e-04 -2.07031250e-01 -1.38671875e-01 -8.97216797e-03 -2.78320312e-02 -1.41601562e-01 2.07031250e-01 -1.58203125e-01 1.27929688e-01 1.49414062e-01 -2.24609375e-02 -8.44726562e-02 1.22558594e-01 2.15820312e-01 -2.13867188e-01 -3.12500000e-01 -3.73046875e-01 4.08935547e-03 1.07421875e-01 1.06933594e-01 7.32421875e-02 8.97216797e-03 -3.88183594e-02 -1.29882812e-01 1.49414062e-01 -2.14843750e-01 -1.83868408e-03 9.91210938e-02 1.57226562e-01 -1.14257812e-01 -2.05078125e-01 9.91210938e-02 3.69140625e-01 -1.97265625e-01 3.54003906e-02 1.09375000e-01 1.31835938e-01 1.66992188e-01 2.35351562e-01 1.04980469e-01 -4.96093750e-01 -1.64062500e-01 -1.56250000e-01 -5.22460938e-02 1.03027344e-01 2.43164062e-01 -1.88476562e-01 5.07812500e-02 -9.37500000e-02 -6.68945312e-02 2.27050781e-02 7.61718750e-02 2.89062500e-01 3.10546875e-01 -5.37109375e-02 2.28515625e-01 2.51464844e-02 6.78710938e-02 -1.21093750e-01 -2.15820312e-01 -2.73437500e-01 -3.07617188e-02 -3.37890625e-01 1.53320312e-01 2.33398438e-01 -2.08007812e-01 3.73046875e-01 8.20312500e-02 2.51953125e-01 -7.61718750e-02 -4.66308594e-02 -2.23388672e-02 2.99072266e-02 -5.93261719e-02 -4.66918945e-03 -2.44140625e-01 -2.09960938e-01 -2.87109375e-01 -4.54101562e-02 -1.77734375e-01 -2.79296875e-01 -8.59375000e-02 9.13085938e-02 2.51953125e-01]

Step 3: Numerical Representation: Word Embedding

To Use Word2Vec:

To check if a word exists in the vocabulary:

```
# Check if the word 'cat' exists in the Word2Vec vocabulary
if 'cat' in wv:
    print("Cat is in the vocabulary")
else:
    print("Cat is not in the vocabulary")
```

```
Cat is in the vocabulary
```

Step 3: Numerical Representation: Word Embedding

To Use Word2Vec: To Visualize Embeddings(using t-SNE):

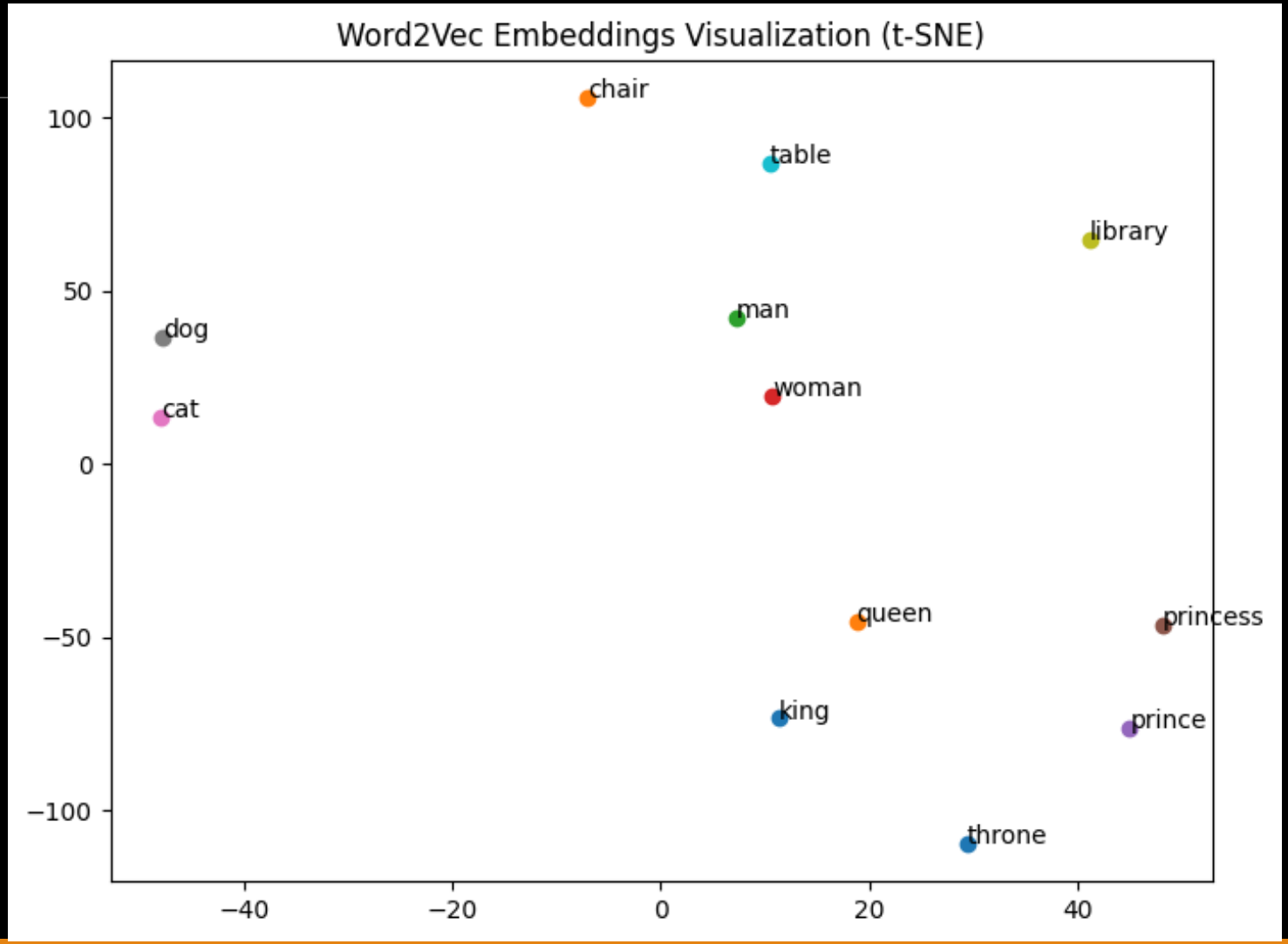
```
# List of words to visualize
words = ['king', 'queen', 'man', 'woman', 'prince', 'princess', 'cat', 'dog', 'library', 'table', 'throne', 'chair']

# Get the vector representations for the words (only if they are in the vocabulary)
embeddings = [wv[word] for word in words if word in wv]

# Convert the list of embeddings to a 2D NumPy array
import numpy as np  # Import NumPy for array manipulation
embeddings = np.array(embeddings) # Convert the list of embeddings to a NumPy array

# Reduce the dimensionality of the vectors to 2D for visualization using t-SNE
# t-SNE (t-Distributed Stochastic Neighbor Embedding) is a technique for visualizing high-dimensional data
# Set perplexity to a value less than the number of samples (8 in this case)
tsne = TSNE(n_components=2, perplexity=5, random_state=42)  # Reduce to 2 dimensions, perplexity=5
embeddings_2d = tsne.fit_transform(embeddings)

# Create a scatter plot of the 2D embeddings
plt.figure(figsize=(8, 6))  # Set the figure size
for i, word in enumerate(words):
    if word in wv:  # Only plot if the word is in the vocabulary
        plt.scatter(embeddings_2d[i, 0], embeddings_2d[i, 1])  # Plot the point
        plt.annotate(word, (embeddings_2d[i, 0], embeddings_2d[i, 1]))  # Add the word label
plt.title('Word2Vec Embeddings Visualization (t-SNE)')  # Set the plot title
plt.show()  # Display the plot
```

Visualizing embedding means how close words relates,

Can we guess??

Word2Vec Visualization

words = ['king', 'queen', 'man', 'woman', 'prince', 'princess', 'cat', 'dog', 'library', 'table', 'throne', 'chair']



Step 3: Numerical Representation: Word Embedding

To Use Word2Vec: To Visualize Embeddings(using t-SNE):

List of words to visualize

['king', 'queen', 'man', 'woman', 'prince', 'princess', 'cat', 'dog', 'library', 'table', 'throne', 'chair']

Let's get this done.....

Can we guess??

Step 3: Numerical Representation: Word Embedding

Waiting!!!!!!!!!

Let's get this done.....

Can we give a trial??

Step 3: Numerical Representation: Word Embedding

List of words to visualize **['king', 'queen', 'man', 'woman', 'prince', 'princess', 'cat', 'dog', 'library', 'table', 'throne', 'chair']**

Say: Royalty: king, queen, prince, princess, throne

Gender: man, woman

Pets/Animals: cat, dog

Books/Reading: library

Furniture: chair, table,

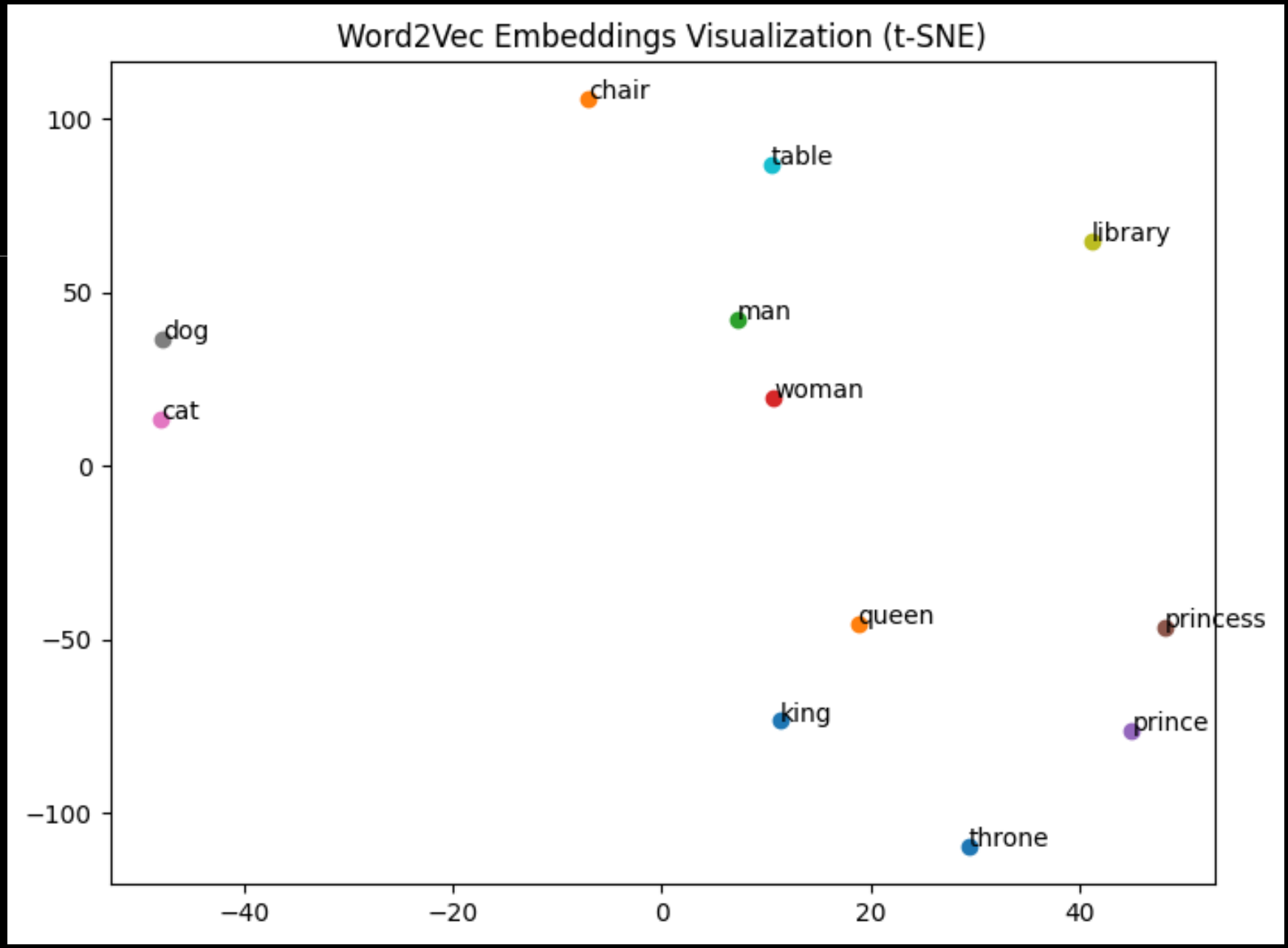
Right????

List of words to visualize

['king', 'queen', 'man', 'woman', 'prince', 'princess', 'cat', 'dog', 'library', 'table', 'throne', 'chair']

So, we have….

Interesting right!!



Step 3: Numerical Representation: Word Embedding

Using Word2Vec, Plot the Embedding od the words below.

```
words = ['king', 'queen', 'prince', 'princess', 'man', 'woman',
'cat', 'dog', 'kitten', 'puppy', 'book', 'library', 'page',
'author', 'chair', 'table', 'furniture', 'sofa']
```

Let's get this done.....

This time more accurate response??

Step 3: Numerical Representation: Word Embedding

Waiting!!!!!!!!!

Let's get this done.....

Can we give a trial??

Step 3: Numerical Representation: Word Embedding

List of words to visualize: `['king', 'queen', 'prince', 'princess', 'man', 'woman', 'cat', 'dog', 'kitten', 'puppy', 'book', 'library', 'page', 'author', 'chair', 'table', 'furniture', 'sofa']`

Say: Royalty: king, queen, prince, princess

Gender: man, woman

Pets/Animals: cat, dog, kitten, puppy

Books/Reading: book, library, page, author

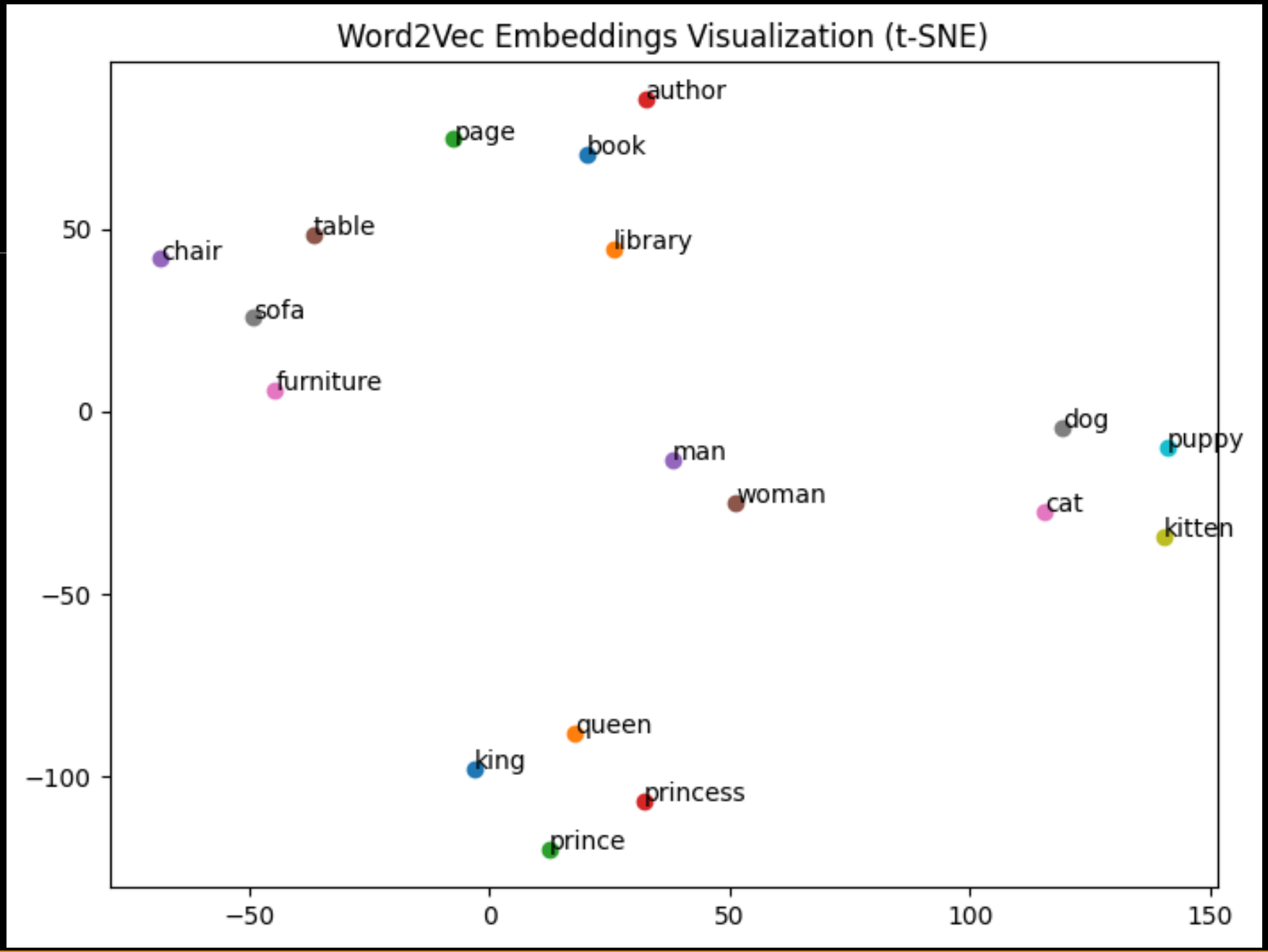
Furniture: chair, table, furniture, sofa

Right????

List of words to visualize:

```
['king', 'queen',
'prince', 'princess',
'man', 'woman', 'cat',
'dog', 'kitten',
'puppy', 'book',
'library', 'page',
'author', 'chair',
'table', 'furniture',
'sofa']
```

So, we have...*interesting right!!*



Step 3: Numerical Representation: Word Embedding

GloVe developed by **Stanford researchers in 2014.**

This embedding technique is based on factorizing a matrix of word co-occurrence statistics.

Its developers have made available precomputed embeddings for millions of English tokens, obtained from Wikipedia data and Common Crawl data.

NB

GloVe == Global Vectors

Step 3: Numerical Representation: Word Embedding

To use the Glove:

First, let's download the GloVe word embeddings precomputed on the 2014 English Wikipedia dataset. It's an 822 MB zip file containing 100-dimensional embedding vectors for 400,000 words (or non-word tokens).

Type the code:

!wget http:/ /nlp.stanford.edu/data/glove.6B.zip

!unzip -q glove.6B.zip

NB

GloVe

To use the Glove:

Downloading GloVe

This about 900mb and another 400mb…..

NOW LETS USE GLOVE TECHNIQUE

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
--2025-02-24 14:25:18--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-02-24 14:25:18--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-02-24 14:25:18--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.00MB/s    in 2m 39s

2025-02-24 14:27:57 (5.19 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

GloVe

Setup some long process which include

Preparing Training and validation sentence

Text Vectorization, embedding, and building the model.

You may need to define some functions you can simply call and pass in it's parameter/arguments to use the built GloVe Model.

NB: This code snippet by the side isn't the full code, we can check that out in a colab

```
# 5. Create the embedding matrix using the consistent vocabulary
embedding_dim = 100
vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))
embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
        embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

# 6. Create the Embedding layer
embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)
# 7. Vectorize your training and validation data
int_train_ds = tf.data.Dataset.from_tensor_slices(
    (text_vectorization(sentences_train), labels_train)  # Use text_vectorizat
).batch(2)

int_val_ds = tf.data.Dataset.from_tensor_slices(
    (text_vectorization(sentences_val), labels_val)  # Use text_vectorization
).batch(2)
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
callbacks = [
    keras.callbacks.ModelCheckpoint("glove_embeddings_sequence_model.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,
          callbacks=callbacks)
model = keras.models.load_model("glove_embeddings_sequence_model.keras")
# Now 'int_train_ds' and 'int_val_ds' have indices matching the embedding layer
```

Step 3: Numerical Representation: Word Embedding

To Use GloVe:

To check for semantic similarities:

You see king and car and very little semantic similarities.

```
# Examples of Semantic Similarity
print(f"Similarity ('king', 'queen'): {glove_similarity('king', 'queen', embeddings_index)}")
print(f"Similarity ('man', 'woman'): {glove_similarity('man', 'woman', embeddings_index)}")
print(f"Similarity ('king', 'man'): {glove_similarity('king', 'man', embeddings_index)}")
print(f"Similarity ('king', 'car'): {glove_similarity('king', 'car', embeddings_index)}")
print(f"Similarity ('cat', 'dog'): {glove_similarity('cat', 'dog', embeddings_index)}")
```

```
Similarity ('king', 'queen'): 0.7507690787315369
Similarity ('man', 'woman'): 0.8323494791984558
Similarity ('king', 'man'): 0.5118681192398071
Similarity ('king', 'car'): 0.28304237127304077
Similarity ('cat', 'dog'): 0.8798074722290039
```

Step 3: Numerical Representation: Word Embedding

To Use GloVe:

To find similar word:

```
# Examples of Finding Similar Words (Continued)
print(f"Most similar to 'cat': {find_most_similar('cat', embeddings_index)}")
print(f"Most similar to 'book': {find_most_similar('book', embeddings_index)}")
```

```
Most similar to 'cat': [('dog', 0.8798075), ('rabbit', 0.74244267), ('cats', 0.7323004), ('monkey', 0.728871), ('pet', 0.71901405)]
Most similar to 'book': [('books', 0.84764856), ('novel', 0.81811666), ('published', 0.8023924), ('story', 0.7941391), ('author', 0.
```

Dog has the most similar word

Step 3: Numerical Representation: Word Embedding

To Use GloVe:

To find similar word:

```
# Examples of Finding Similar Words (Continued)
print(f"Most similar to 'cat': {find_most_similar('cat', embeddings_index)}")
print(f"Most similar to 'book': {find_most_similar('book', embeddings_index)}")
```

```
Most similar to 'cat': [('dog', 0.8798075), ('rabbit', 0.74244267), ('cats', 0.7323004), ('monkey', 0.728871), ('pet', 0.71901405)]
Most similar to 'book': [('books', 0.84764856), ('novel', 0.81811666), ('published', 0.8023924), ('story', 0.7941391), ('author', 0.
```

Dog has the most similar word to Cat

Step 3: Numerical Representation: Word Embedding

To Use GloVe:

To check if specific Word Vector exists:

King exists, randomword doesn't! That's expected.

```
# Examples of Checking Word Vector Existence
check_word_vector("king", embeddings_index)
check_word_vector("randomword", embeddings_index)
```

```
'king' has a GloVe vector.
'randomword' does not have a GloVe vector.
```

Step 3: Numerical Representation: Word Embedding

To Use GloVe:

To check if specific Word Vector exists:

As seen, Vector for "**queen**" exists, but no vector for the word "**anotherword**'

```
# Examples of Getting the Vector for a Word
get_word_vector("queen", embeddings_index)
get_word_vector("anotherword", embeddings_index)
```

```
Vector for 'queen': [-0.50045  -0.70826   0.55388   0.673     0.22486   0.60281  -0.26194
  0.73872  -0.65383  -0.21606  -0.33806   0.24498  -0.51497   0.8568
 -0.37199  -0.58824   0.30637  -0.30668  -0.2187    0.78369  -0.61944
 -0.54925   0.43067  -0.027348  0.97574   0.46169   0.11486  -0.99842
  1.0661   -0.20819   0.53158   0.40922   1.0406    0.24943   0.18709
  0.41528  -0.95408   0.36822  -0.37948  -0.6802   -0.14578  -0.20113
  0.17113  -0.55705   0.7191    0.070014 -0.23637   0.49534   1.1576
 -0.05078   0.25731  -0.091052  1.2663    1.1047   -0.51584  -2.0033
 -0.64821   0.16417   0.32935   0.048484  0.18997   0.66116   0.080882
  0.3364    0.22758   0.1462   -0.51005   0.63777   0.47299  -0.3282
  0.083899 -0.78547   0.099148  0.039176  0.27893   0.11747   0.57862
  0.043639 -0.15965  -0.35304  -0.048965 -0.32461   1.4981    0.58138
 -1.132    -0.60673  -0.37505  -1.1813    0.80117  -0.50014  -0.16574
 -0.70584   0.43012   0.51051  -0.8033   -0.66572  -0.63717  -0.36032
  0.13347  -0.56075 ]
'anotherword' has no vector.
```

Again, the vectors are in numbers just like it was when we used word2vec

Step 3: Numerical Representation: Word Embedding

To Use GloVe:

To visualize Embeddings (using t-SNE):

The word_to_visualize variable stores the list with the words

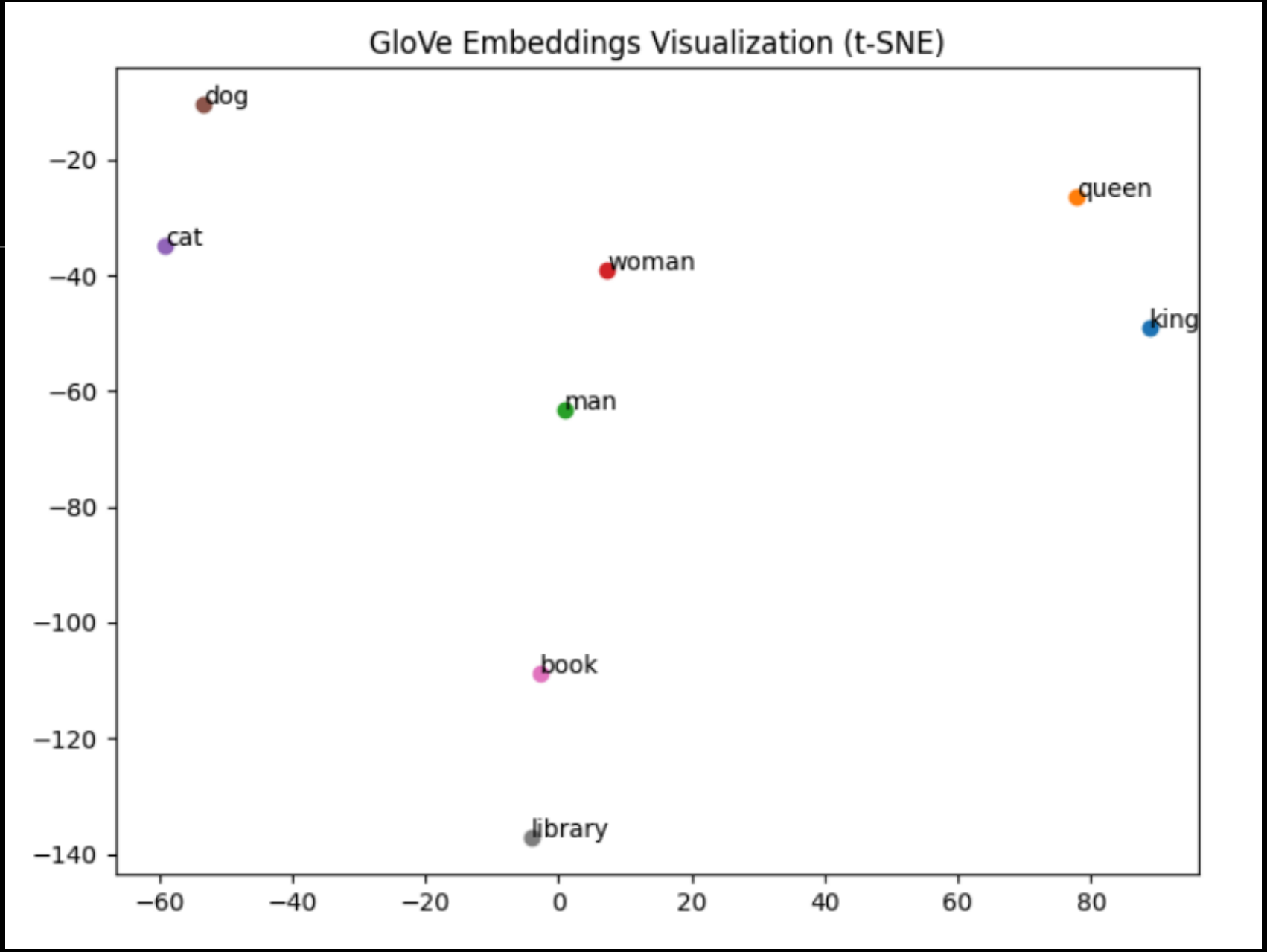
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def visualize_embeddings(words, embeddings_index):
    """Visualizes GloVe embeddings using t-SNE."""
    embeddings = [get_glove_vector(word, embeddings_index) for word in words if get_glove_vector(word, embeddings_index) is not None]
    words_filtered = [word for word in words if get_glove_vector(word, embeddings_index) is not None]

    if not embeddings:
        print("No embeddings to visualize.")
        return

    # Convert the list of embeddings to a NumPy array
    embeddings = np.array(embeddings)

    # Lower the perplexity to be significantly less than the number of samples
    tsne = TSNE(n_components=2, perplexity=3, random_state=42)  # Reduced perplexity to 3
    embeddings_2d = tsne.fit_transform(embeddings)

    plt.figure(figsize=(8, 6))
    for i, word in enumerate(words_filtered):
        plt.scatter(embeddings_2d[i, 0], embeddings_2d[i, 1])
        plt.annotate(word, (embeddings_2d[i, 0], embeddings_2d[i, 1]))
    plt.title('GloVe Embeddings Visualization (t-SNE)')
    plt.show()

words_to_visualize = ['king', 'queen', 'man', 'woman', 'cat', 'dog', 'book', 'library']
visualize_embeddings(words_to_visualize, embeddings_index)
```

GloVe

To Use GloVe:

To visualize Embeddings (using t-SNE):

words_to_visualize = ['king', 'queen', 'man', 'woman', 'cat', 'dog', 'book', 'library']



Step 3: Numerical Representation: Word Embedding

To Use GloVe:

To visualize Embeddings (using t-SNE):

words_to_visualize = ['king', 'queen', 'prince', 'princess', 'man', 'woman', 'cat', 'dog', 'kitten', 'puppy', 'book', 'library', 'page', 'author', 'chair', 'table', 'furniture', 'sofa']

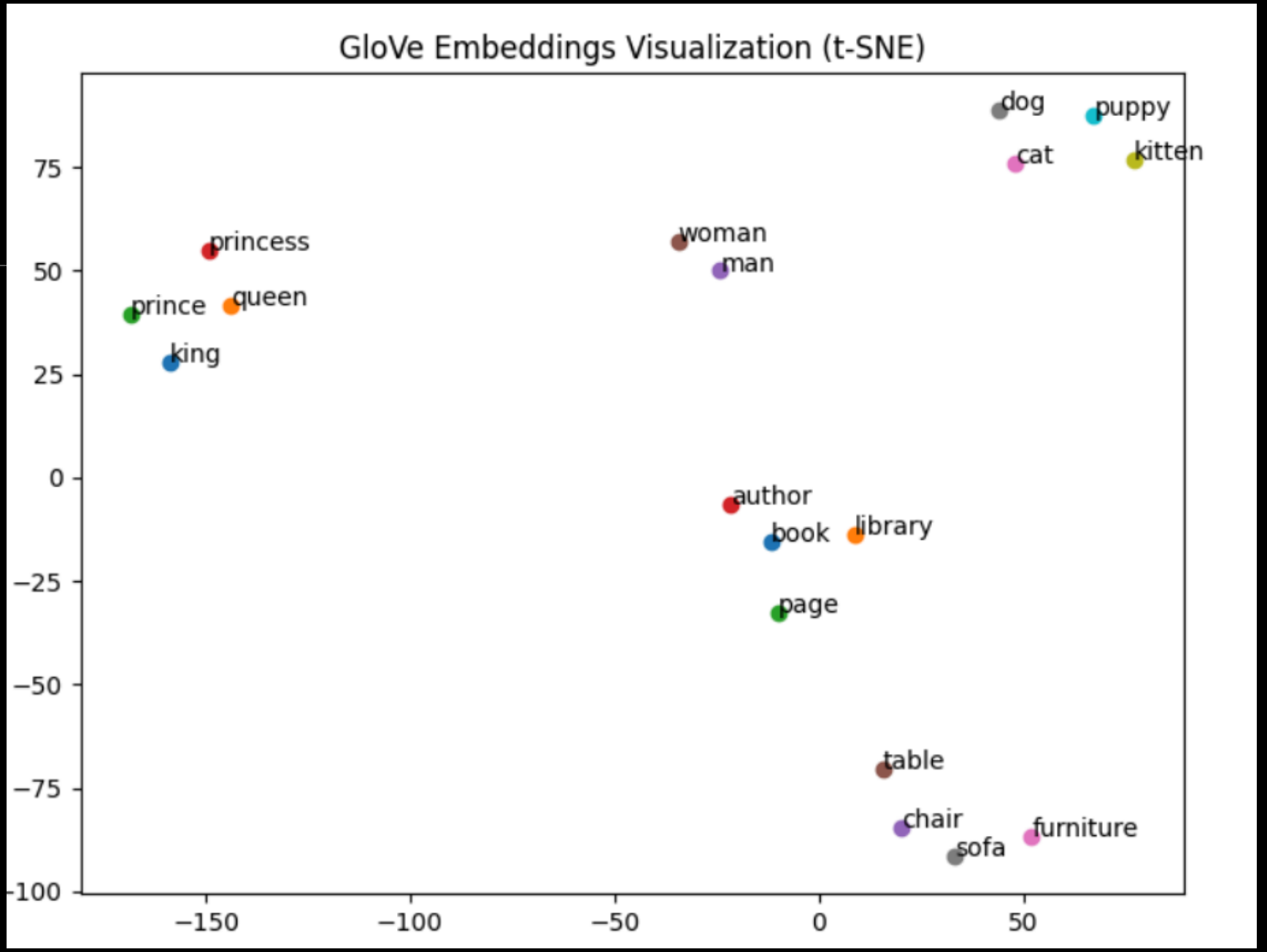
```
words_to_visualize = ['king', 'queen', 'prince', 'princess', 'man', 'woman',
         'cat', 'dog', 'kitten', 'puppy',
         'book', 'library', 'page', 'author',
         'chair', 'table', 'furniture', 'sofa']

visualize_embeddings(words_to_visualize, embeddings_index)
```

GloVe

To Use GloVe:

To visualize Embeddings (using t-SNE):

words_to_visualize = ['king', 'queen', 'prince', 'princess', 'man', 'woman', 'cat', 'dog', 'kitten', 'puppy', 'book', 'library', 'page', 'author', 'chair', 'table', 'furniture', 'sofa']



What is t-SNE (pronounced tee-snee)

t-distributed stochastic neighbor embedding, t-SNE (pronounced tee-snee), simply used for plotting word vector.

It was developed by **Laurens van der Maaten** in collaboration with **Geoff Hinton.**

t-SNE allow us to use the **dimensionality reduction** to approximately map the locations of words from high dimensional word-vector space down to two or three dimensions.

```
# t-SNE (t-Distributed Stochastic Neighbor Embedding) is a technique for visualizing high-dimensional
# Set perplexity to a value less than the number of samples (8 in this case)
tsne = TSNE(n_components=2, perplexity=5, random_state=42)  # Reduce to 2 dimensions, perplexity=5
embeddings_2d = tsne.fit_transform(embeddings)
```

Word2Vec vs GloVe

FEATURE	WORD2VEC	GLOVE
Learning Method	"Guess the Neighbors" (Predictive)	"Count the Co-occurrences" (Statistical)
How It Works	Tries to predict nearby words in a sentence.	Analyzes how often words appear together in the whole text.
Focus	Local context (words close together)	Global statistics (overall word relationships)
Analogy	"Gossip Network" (learns from nearby "friends")	"Census Taker" (counts co-occurrences like neighbors)
Strength	Captures subtle semantic relationships.	Captures general word similarity and relatedness.
Example Use	Understanding "run" in different contexts (exercise vs. program).	Knowing "cat" and "dog" are generally similar animals.
Key Idea	Learns by predicting what words go together in a sentence.	Learns by counting how often words appear together in a large text.
Think of It As	Learning from the immediate surrounding words.	Learning from the entire collection of words.

Numerical Representation: Word Embedding

FastText: Developed by facebook's AI Research lab (meta) in 2015

- It's the contemporary leading alternative to both word2vec and GloVe, Designed to address their limitations, particularly in handling rare words and morphologically rich languages (as in word2vec).
- FastText can understand that 'unbreakable' is related to 'break' even if it hasn't seen 'unbreakable' before.
- It considers subword information, making it effective for morphologically rich languages and handling out-of-vocabulary words.
- This enables fastText to work around some of the issues related to rare words and out-of-vocabulary words addressed in the preprocessing section at the outset of this chapter.

Step 3: Numerical Representation: Word Embedding

BERT (bi-directional encoder representations from transformers):

- BERT was developed by Google and introduced in 2018.
- It revolutionized NLP with its ability to understand context deeply.
- It's based on the **Transformer** architecture, which uses **attention mechanisms.**

What's Attention Mechanism??? What are transformers???

- Unlike Word2Vec and GloVe, which are unidirectional or consider context in a limited way, BERT is trained bidirectionally.
- This means it considers the context of a word from both the left and right sides, leading to a much richer understanding.

Step 3: Numerical Representation: Word Embedding

Because BERT is Transformer Based, we need to understand a new topic called **Attention Mechanism** and **Transformers** first.

For now, understand that, considering Architecture, Word2Vec and GloVe are simpler models. BERT is a deep **Transformer-based model.**

Also, Unlike Word2Vec and GloVe, which are unidirectional or consider context in a limited way, **BERT** is trained **bidirectionally.**

This means it considers the context of a word from both the left and right sides, leading to a much richer understanding.

Yaa, Let's wait till then,

Meanwhile, before then, DYOR

Pre-Trained Model for Word Embedding history

Recap:

Word2Vec = 2013

Glove = 2014

FastText = 2015

BERT = 2018

What's Next

Maybe I don't know also!!!

If Yes, 

Let's move back a bit!!

Understand Human Language

There are arguably 3 steps to help computer understand human texts,

1. Text Preprocessing
2. Tokenization
3. Numerical Representation (Word Embedding)

Now, Lets discuss each further!!

Remember this? We've explained the 3 steps..... Whats next in NLP? Can we now build our ChatGPT?

What's Next 😃 😃

I remembered something

How about
Transformers ??
and
Attention ??

Transformers: Attention is All You Need

A transformer is a type of artificial intelligence model that learns to understand and generate human-like text by analyzing patterns in large amounts of text data

Transformers are a revolutionary type of **neural network** that rely entirely on something called **attention**.

They process all the words at the same time, making them much faster than **RNNs (Recurrent Neural Networks).**

We Move!!!!

What do you notice 😃 😃

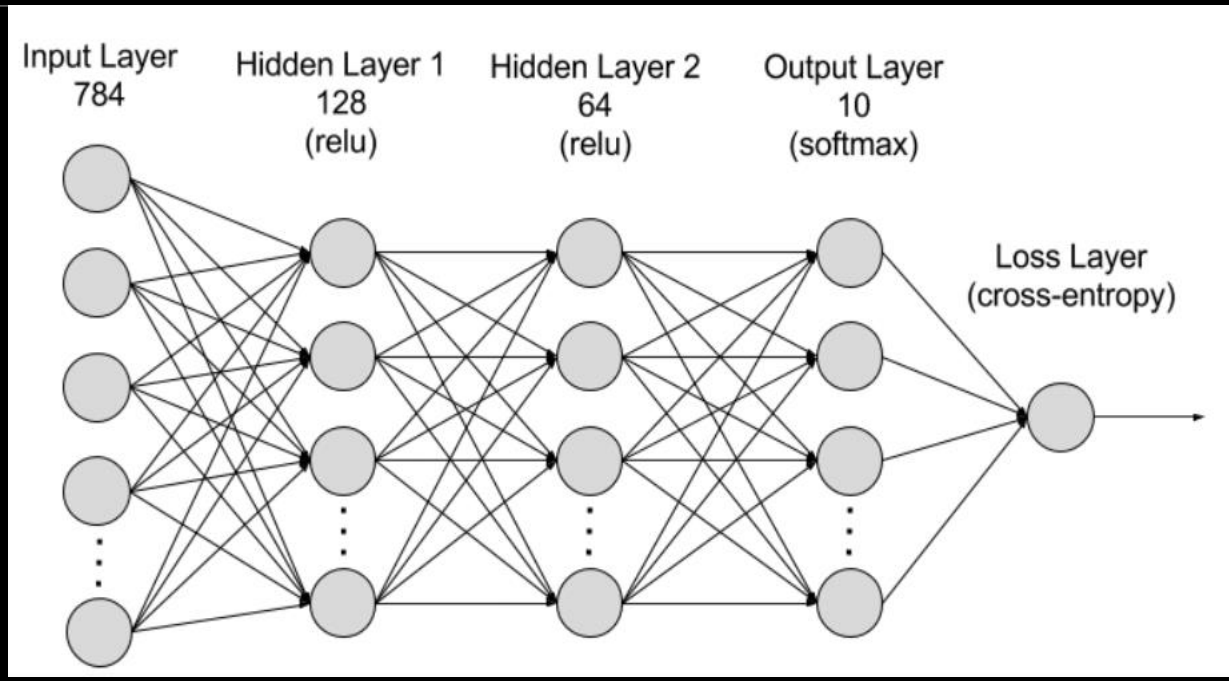
We have mentioned RNNs (Recurrent Neural Network) severally with minimal explanation

RNN, let's discuss briefly

Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) is a family of deep learning models that can handle long sequences of data.

The family include specialized layer types like **long short-term memory units (LSTMs)** and **gated recurrent units (GRUs).**

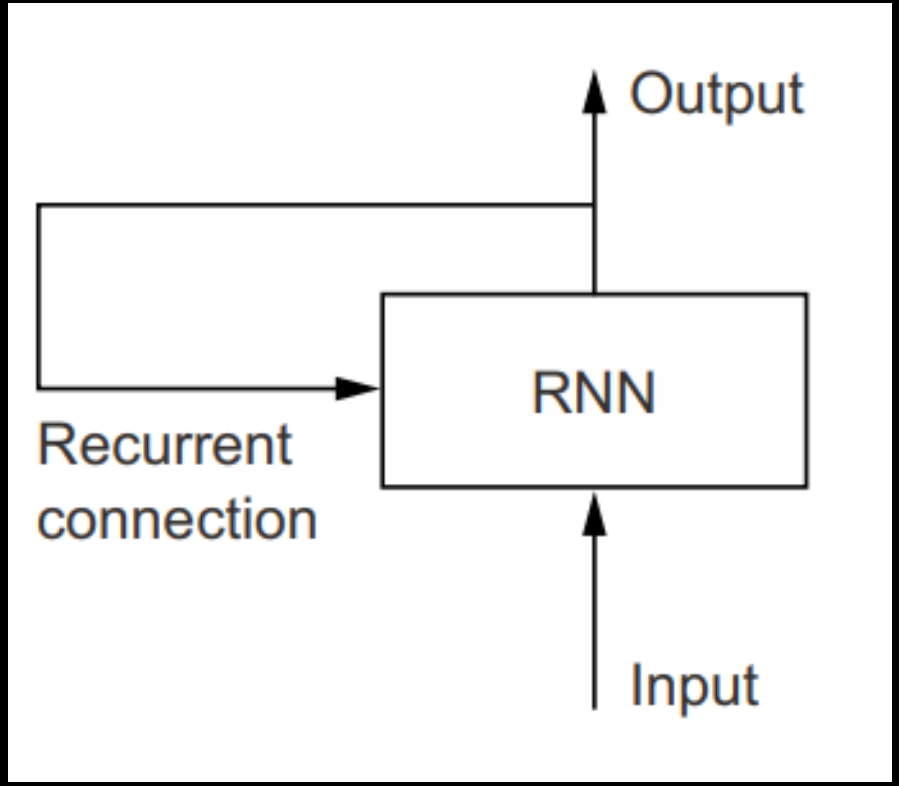


Sequential data is data, such as words, sentences, or time-series data, where sequential components interrelate based on complex semantics and syntax rules.

Recurrent Neural Networks (RNNs)

In effect, an RNN is a type of neural network that has an internal loop, hence, **Recurrent Neural Network**.

In simple terms; an RNN is a **for loop** that reuses quantities computed during the **previous iteration of the loop**.



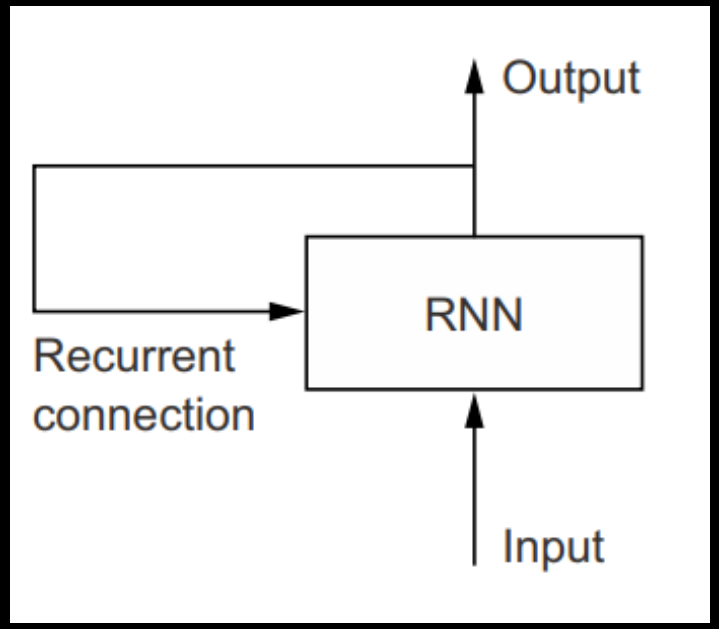
A recurrent network:
a network with a loop

Recurrent Neural Networks (RNNs)

RNNs maintain a **hidden state** that acts as a form of memory.

At each time step, the hidden state is updated based on the current input and the previous hidden state.

This allows the network to capture dependencies and patterns across the sequence, though forgetting captured information from the very beginning of a long Sentence.



Types of RNN Architectures:

Simple RNNs (Vanilla RNNs): The basic form of RNNs. Prone to vanishing and exploding gradient problems.

Long Short-Term Memory (LSTM) Networks: Designed to address the vanishing gradient problem. Use **"gates"** to control the flow of information in the hidden state. Better at capturing long-range dependencies.

Gated Recurrent Unit (GRU) Networks: A simplified version of LSTMs. Also uses gates to control information flow. Often performs comparably to LSTMs with fewer parameters.

Bidirectional RNNs: Process sequences in both forward and backward directions. Useful for tasks where context from both past and future is important.

Deep RNNs: Stack multiple RNN layers on top of each other. Allow the network to learn more complex representations.

Bottleneck of RNNs

RNNs have trouble dealing with very long sequences, as they tend to progressively forget about the past.

By the time you've reached the 100th token in either sequence, little information remains about the start of the sequence.

That means RNN-based models can't hold onto long-term context, which can be essential for translating long documents.

QUESTIONS AND ANSWERS

The END Day 3

DSN LEKKI-AJAH
BY ABEREJO HABEEBLAH O.
X: @HABEREJO

Next Class schedule:
Tuesday, 25th March, by 5:00pm.

Welcome Here!!

NATURAL LANGUAGE PROCESSING (NLP)

TRANSFORMERS, ATTENTION AND PRETRAINED NLP MODELS (HUGGING FACE, GPT)

DSN LEKKI-AJAH

BY ABEREJO HABEEBLAH O.

X (TWITTER): @HABEREJO

Day 4

Tuesday, 25th March, 2025

Why are we here

To equip learners with the skills to understand how Artificial Intelligence models are built, explore Machine Learning and Deep Learning, and focus on Natural Language Processing (NLP).

Project Class

Classes Structure

Every Tuesday throughout the month of March.

4 classes in total. This is the fourth class

Each class for about 1:45 minutes

Yuppppp, Last class

Who should be here

- You're curious and ready to learn something new
- You dream of becoming a Data Scientist.
- You're passionate about building the future as a Machine Learning Engineer.
- You have a basic understanding of Python programming.
- You've worked with data and want to take your skills to the next level.
- You're a researcher exploring the exciting world of AI.

which of these are you?

Who should be here 

- You're familiar with the fundamentals of Machine Learning and Deep Learning.
- You love problem solving and are excited to tackle real world problems with AI.
- You want to understand the technology that is shaping the future.
- You're driven by a desire to learn and grow in the field of AI.
- You enjoy collaborating and learning from others.

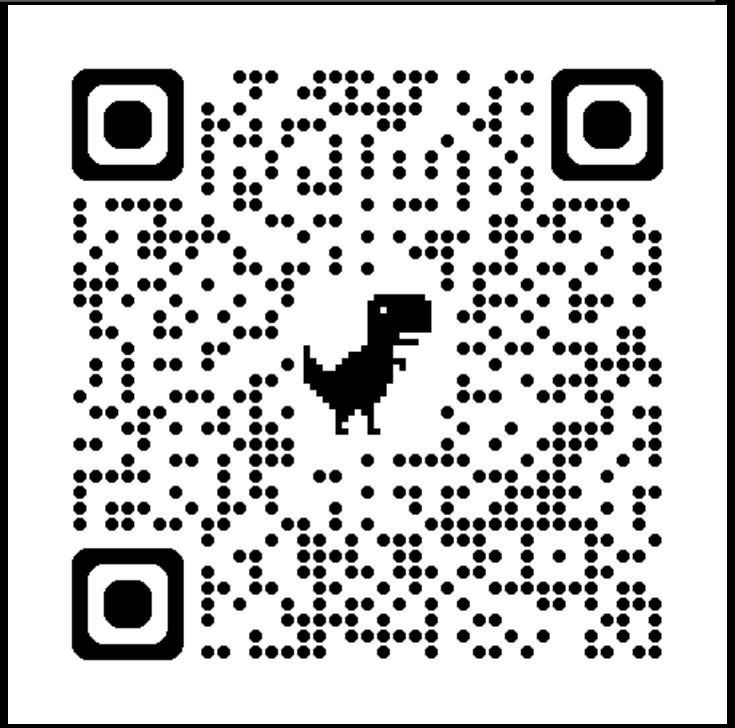
Which of these are you?????
You ticked any of these boxes?
Let's Goooooooo

Before we begin

Who I am?

- A Machine Learning Engineer
- A young MAN eager to master AI

FUN FACT:
I've taught over 100 people Python, but
I'm still learning new things every day.
(Especially how to avoid typos when
coding late at night!)



Scan to visit my portfolio
or
https://bheez.netlify.app

Quick Recap (First Class)

1️⃣ **Understanding AI, ML, DL, and NLP**

2️⃣ **Introduction to NLP**

3️⃣ **NLP Use Cases:** Translation Apps 🌍, Spam Filters 📧, Search Engines 🔍, Sentiment Analysis 😀😡, Voice Assistants 🎙️, Chatbots 💬, Autocorrect & Predictive Text 📱.

4️⃣ **How Computer Understands Text:** 1. Text Preprocessing 2. Tokenization, 3. Numerical Representation (Word Embedding)

5️⃣ **Text Preprocessing Techniques:** Lowercasing, Removing Punctuation Marks, Removing Stopwords (i.e. like A, of, in, etc), Stemming / Lemmatization.

Before we begin

Quick Recap (Second Class)

1. **How Computer Understands Text:** 1. Text Preprocessing
2. **Text Preprocessing Techniques:** Lowercasing, Removing Punctuation Marks, Removing Stopwords (i.e. like A, of, in, etc), Stemming / Lemmatization.
3. **How Computer Understands Text:** 2. Tokenization
4. Tokenization: **Sentence Tokenization** (Sentence-Level Tokenization)

   Word Tokenization (Word-Level Tokenization)

   Subword Tokenization

   Character-level Tokenization
5. **How Computer Understands Text:** 3. Numerical Representation
6. **Numerical Representation:** Vocabulary and Integer Encoding
7. **Numerical Representation:** One Hot Encoding

Quick Recap (Third Class)

1. Numerical Representation: Word Embedding
2. **Word2Vec**
3. **Glove**
4. **t-distributed stochastic neighbor embedding, t-SNE (pronounced tee-snee)**
5. **FastText**
6. **BERT** (bi-directional encoder representations from transformers)
7. Recurrent Neural Networks (RNNs), Transformers

Class Structure

1. Address RNNs Limitations
2. **Transformers**: Attention is all you Need
3. **Attentions**
4. **Attention mechanism:** Queries, Keys and Value
5. Encode and Decoder
6. Evaluating models with metrics
7. Questions and Answer

Bottleneck of RNNs

RNNs have trouble dealing with very long sequences, as they tend to progressively forget about the past.

By the time you've reached the 100th token in either sequence, little information remains about the start of the sequence.

That means RNN-based models can't hold onto long-term context, which can be essential for translating long documents.

Addressing Limitations:

1. **Attention Mechanisms** which allow the model to focus on relevant parts of the input sequence.
2. **Transformers** which replaces **recurrence** with **attention mechanisms**, enabling parallel processing.

Don't get confused, and Transformers are a model, Attention is a component that can be used in any Neural Network like CNN.

3. **Using LSTMs and GRUs** which uses gating mechanisms to control information flow.

This limitation is what has led the ML community to embrace the Transformer architecture for sequence-to-sequence problems

The Transformer

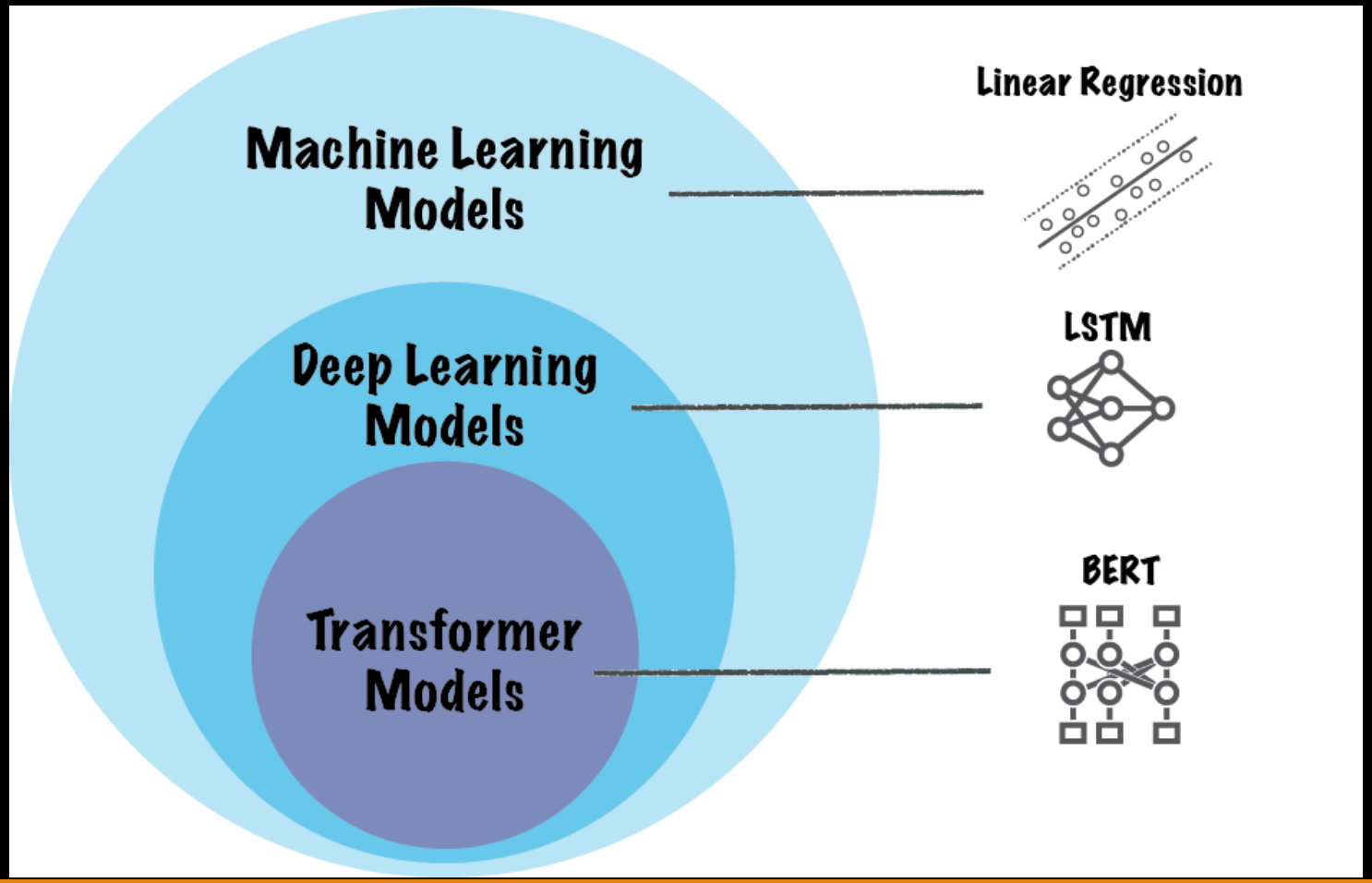
1. Starting in 2017, a new model architecture started overtaking recurrent neural networks across most natural language processing tasks: **the Transformer.**
2. Transformers were introduced in the seminal paper **"Attention is all you need"** by Vaswani et al.
3. The gist of the paper is right there in the title: as it turned out, a simple mechanism called **"neural attention"** could be used to build powerful sequence models that didn't feature any recurrent layers or convolution layers.

Read the paper via this link:
https://arxiv.org/abs/1706.03762. or a Google search

Transformers: Attention is All You Need

So now, let's redefine Transformers,

Transformers is a type of **Neural Network** architecture that is built on **ATTENTION** as its foundational mechanism. **e.g BERT, GPT.**



Transformers: Attention is All You Need

Transformers are a revolutionary type of **neural network** that rely entirely on something called **attention**.

Transformers are based entirely on **attention mechanisms**, eliminating **recurrence** like in RNN.

Unlike RNN that analyze words one by one, **Transformers** read all the words at the same time.

Time to now explain Attention better

Attention

Attention is a technique that allows a neural network to focus on specific parts of an input sequence when processing it.

They enable the model to weigh the importance of different input elements, giving more "**attention**" to the most relevant ones.

Think of it as a way for the model to selectively focus on the most important information.

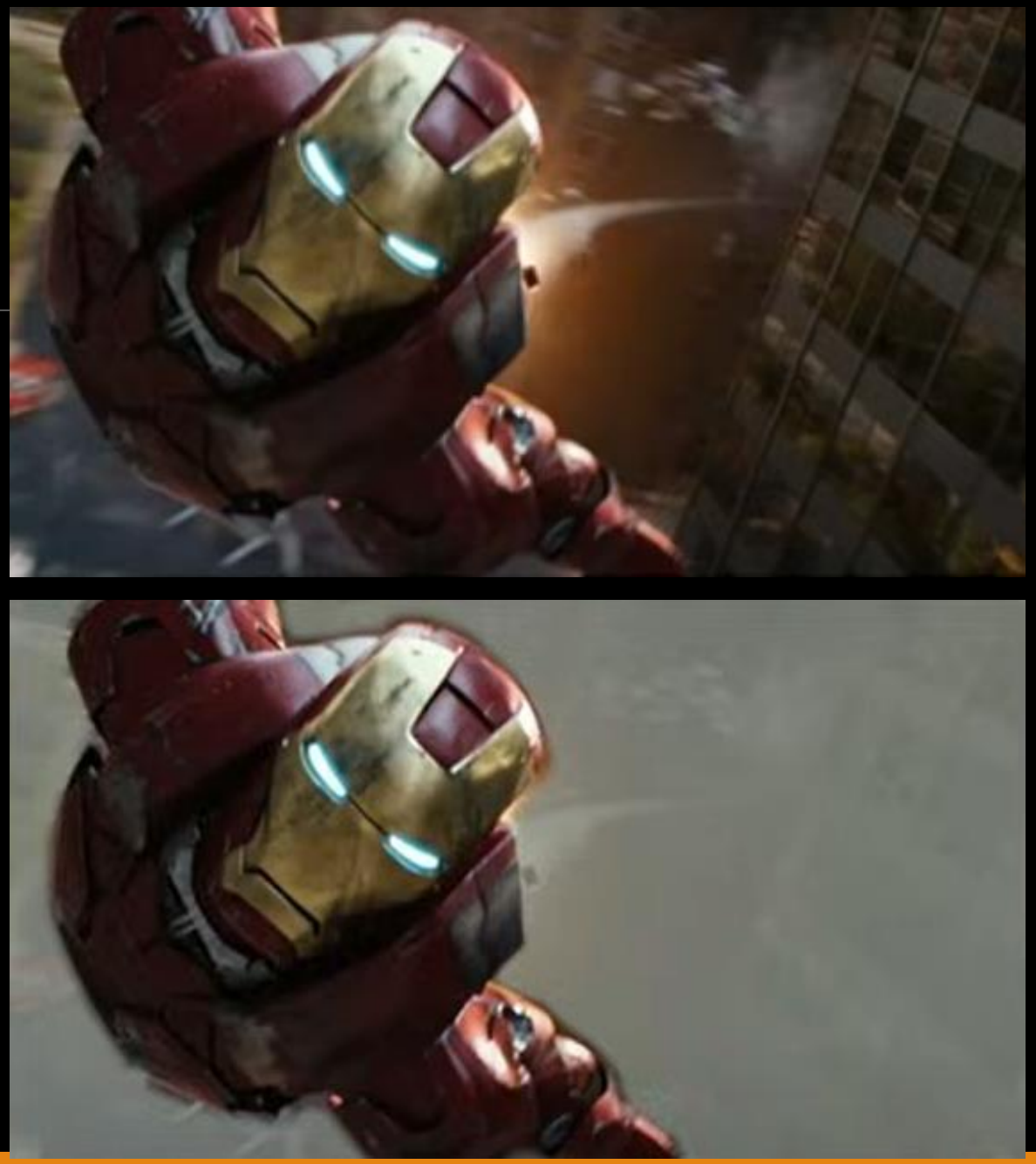
Attention

Attention (or Neural Attention) is a simple yet powerful idea, Because not all input information seen by a model is equally important to the task at hand, so models should **"pay more attention"** to some features and **"pay less attention"** to other features.

Attention

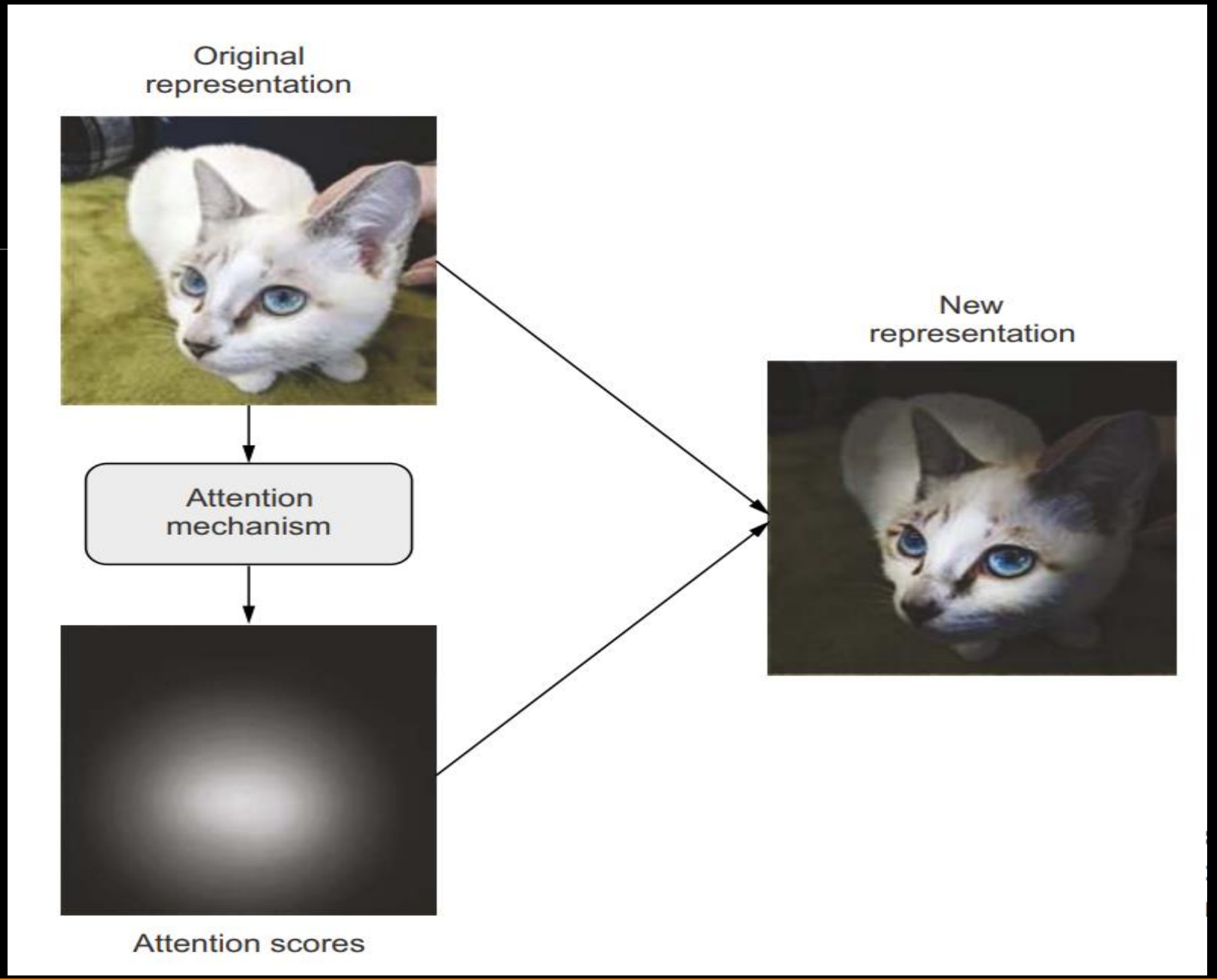
Because not all input information seen by a model is equally important to the task at hand, so models should **"pay more attention"** to some features and **"pay less attention"** to other features.

For a task, What/who is moving?



Attention

This idea looks similar to that of Max pooling in CNN, Max pooling in convnets looks at a pool of features in a spatial region and selects just one feature to keep. That's an "all or nothing" form of attention: keep the most important feature and discard the rest



Attention

Let's consider an example sentence:

"the train left the station on time."

Now, consider one word in the sentence: **station**.

What kind of station are we talking about? Could it be a radio station? Maybe the International Space Station? Let's figure it out algorithmically via self-attention

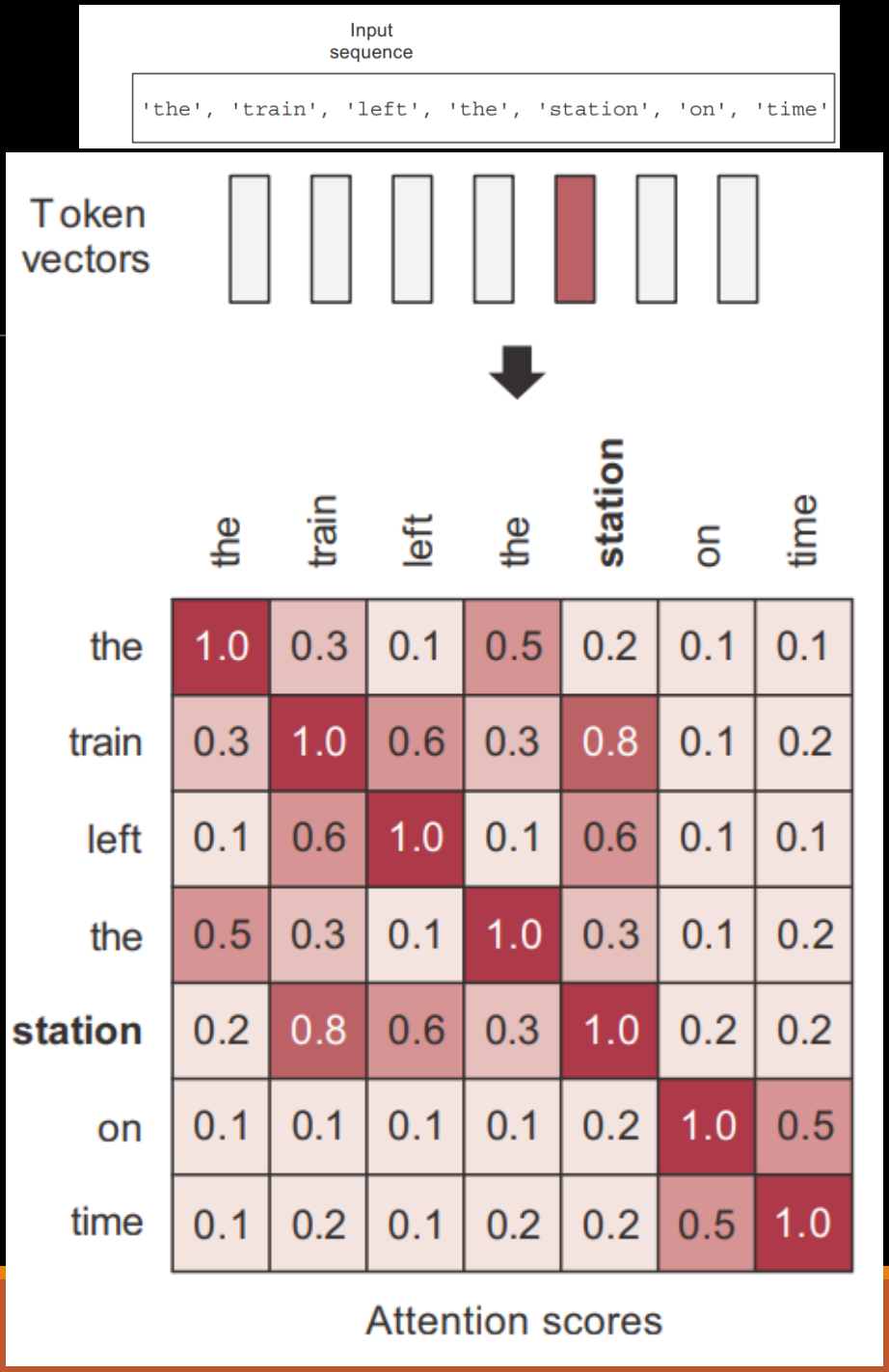
```
'the', 'train', 'left', 'the', 'station', 'on', 'time'
```

Attention

Step 1 is to compute relevancy scores between the vector for "**station**" and every other word in the sentence.

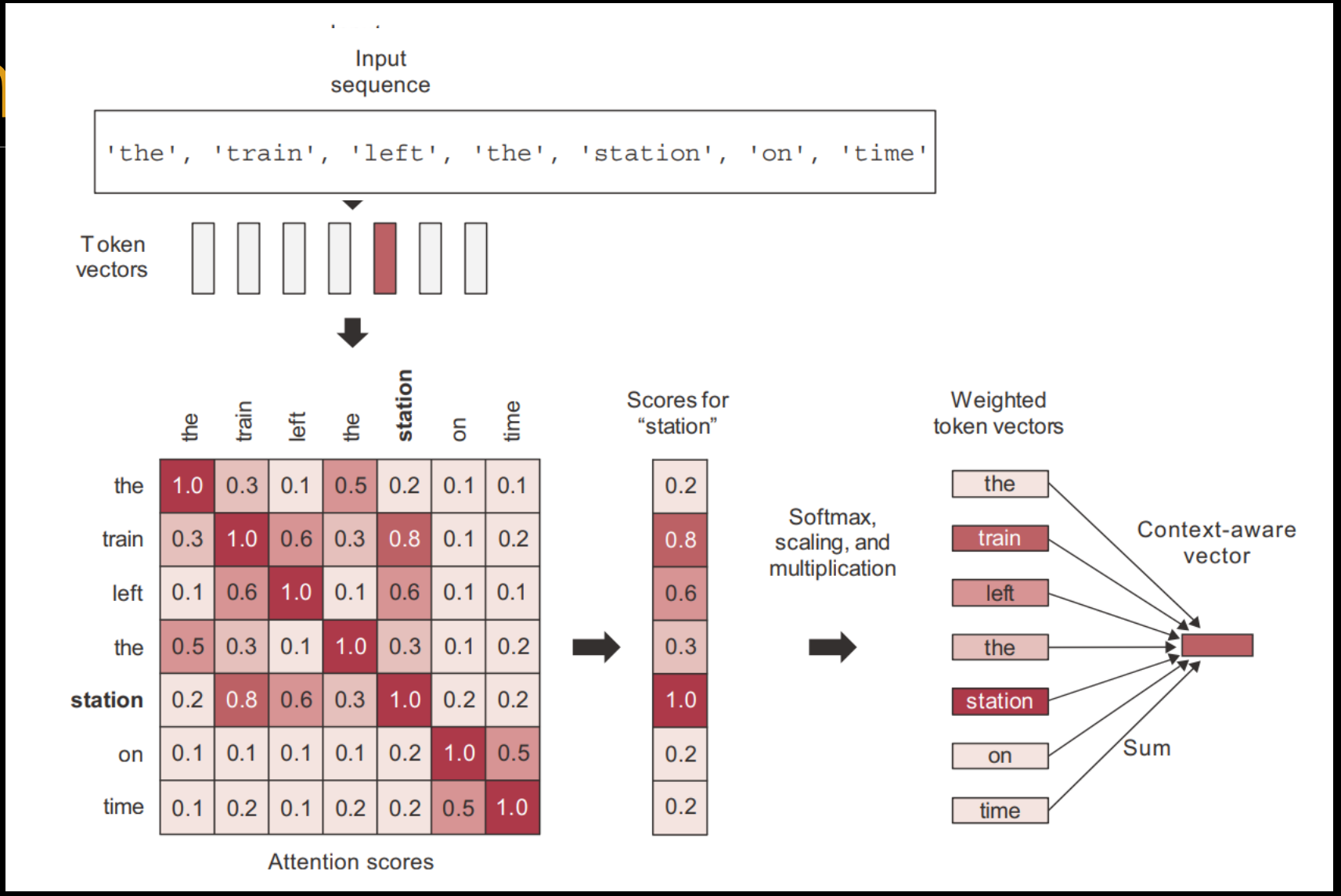
These are our **"attention scores."** We're simply going to use the dot product between two-word vectors as a measure of the strength of their relationship.

It's a very computationally efficient distance function, and it was already the standard way to relate two-word embeddings to each other long before **Transformers**.



Attention

Right?



Attention

Helps us attend to the most important parts of an input by

1. Identify which parts to attend to

2. Extract the features with high attention

Attention Mechanism (Queries, Keys, Values)

Attention mechanisms helps to fine the right information, using **QUERIES**, **KEYS**, **VALUES**.

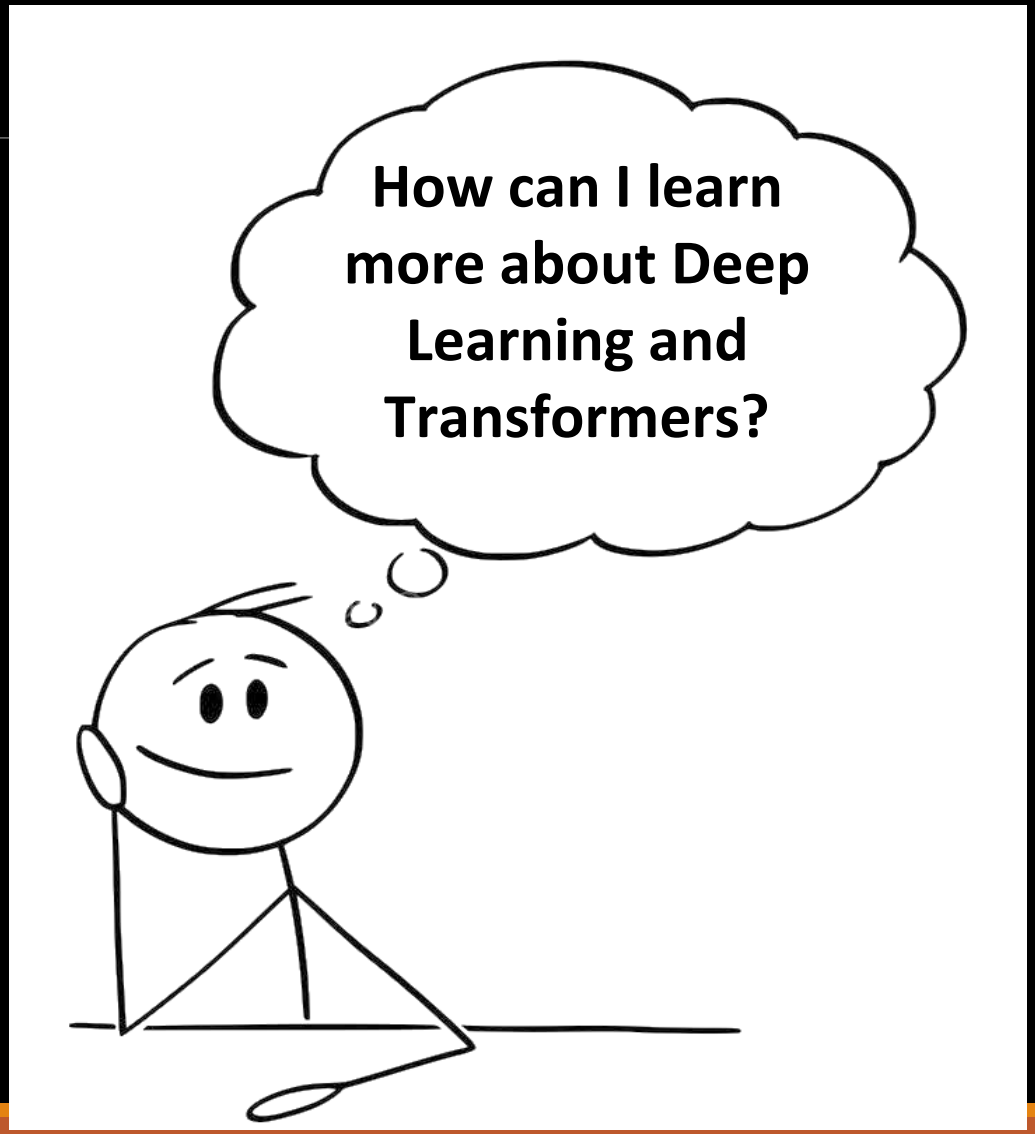
Let me help you understand **Attention Mechanism** with **SEARCH.** (i.e. using a SEARCH example)

Attention Mechanism (Queries, Keys, Values)

Attention mechanisms helps to fine the right information, using **Queries**, **Keys**, **Values**

Let's explain like this……

Because you are here to **Learn and Understand NLP and Deep Learning**, you therefore have the question:



Attention Mechanism (Queries, Keys, Values)

In addition to joining this class, what you may do again is

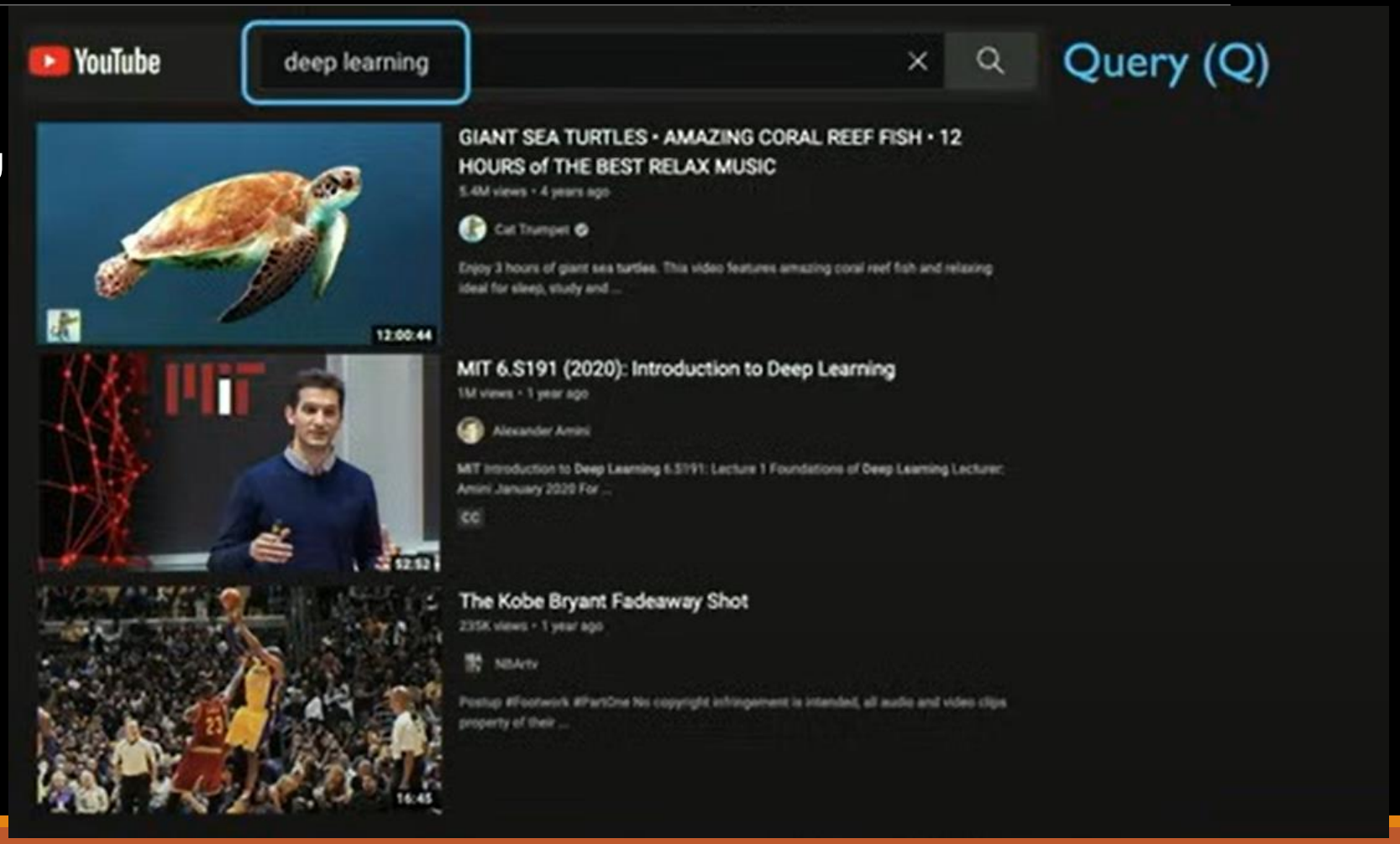
1. Go search online,

YouTube Precisely

Attention Mechanism (Queries, Keys, Values)

As you get to YouTube, how do you find the needed video from the giant database?

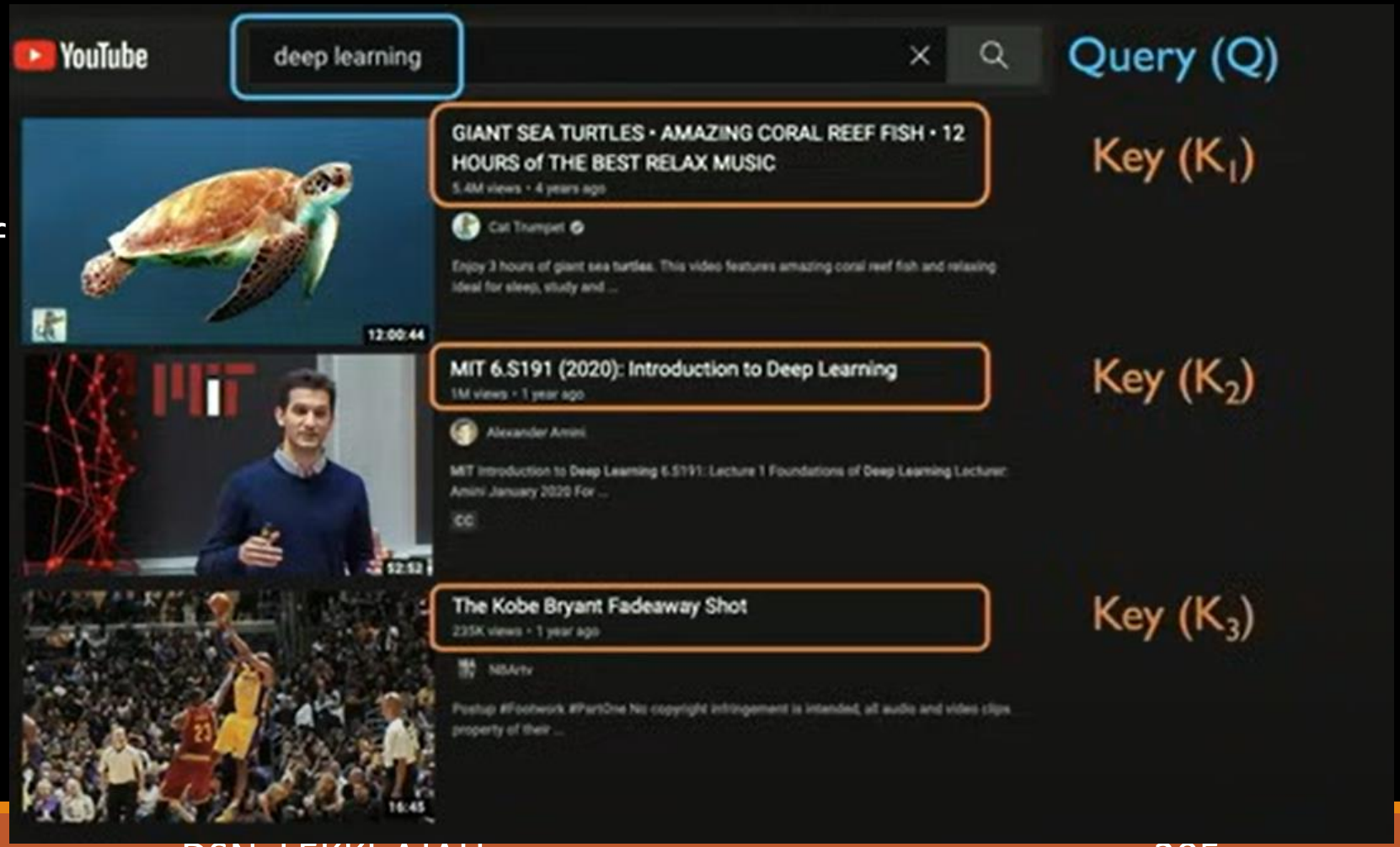
- With a search, by supplying the **Query**, **Deep Learning.**



Attention Mechanism (Queries, Keys, Values)

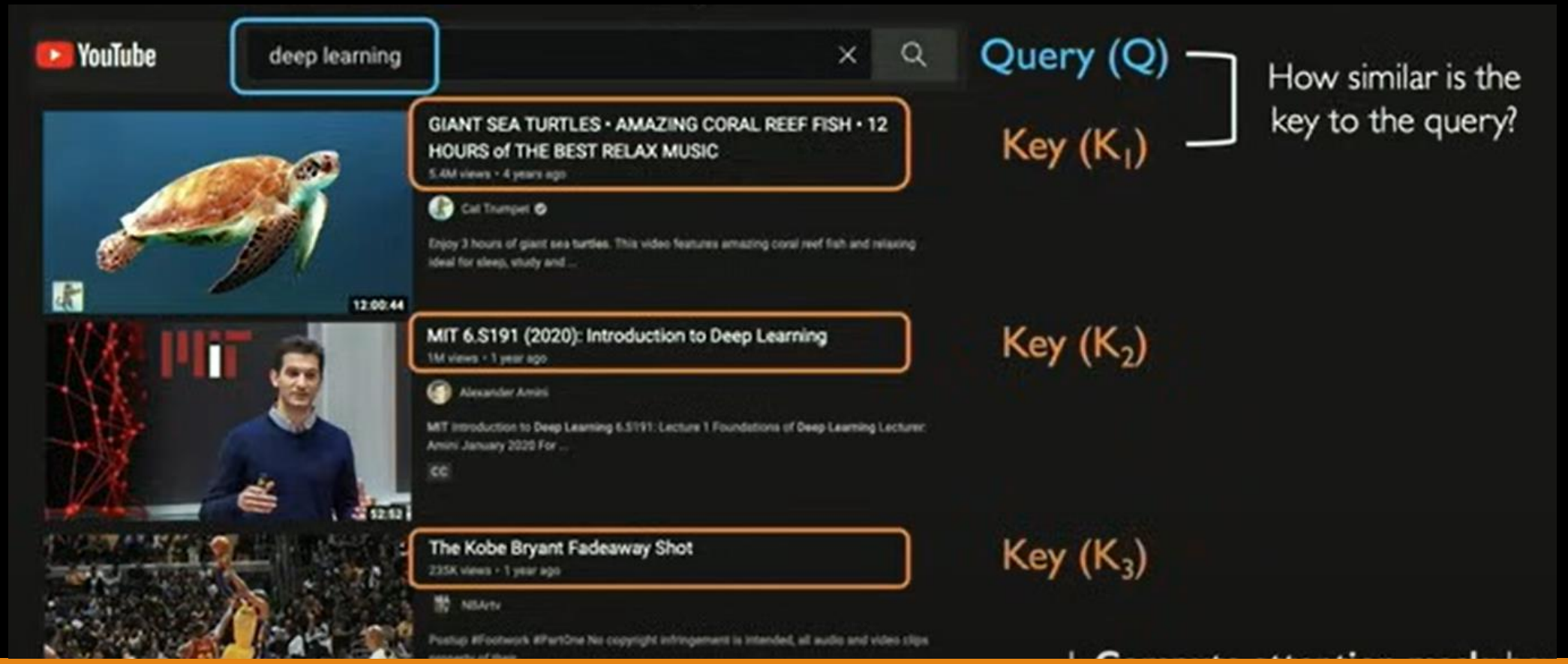
Now, the submitted query needs to be compared with **title** of videos on YouTube.

All the titles are considered **Keys**



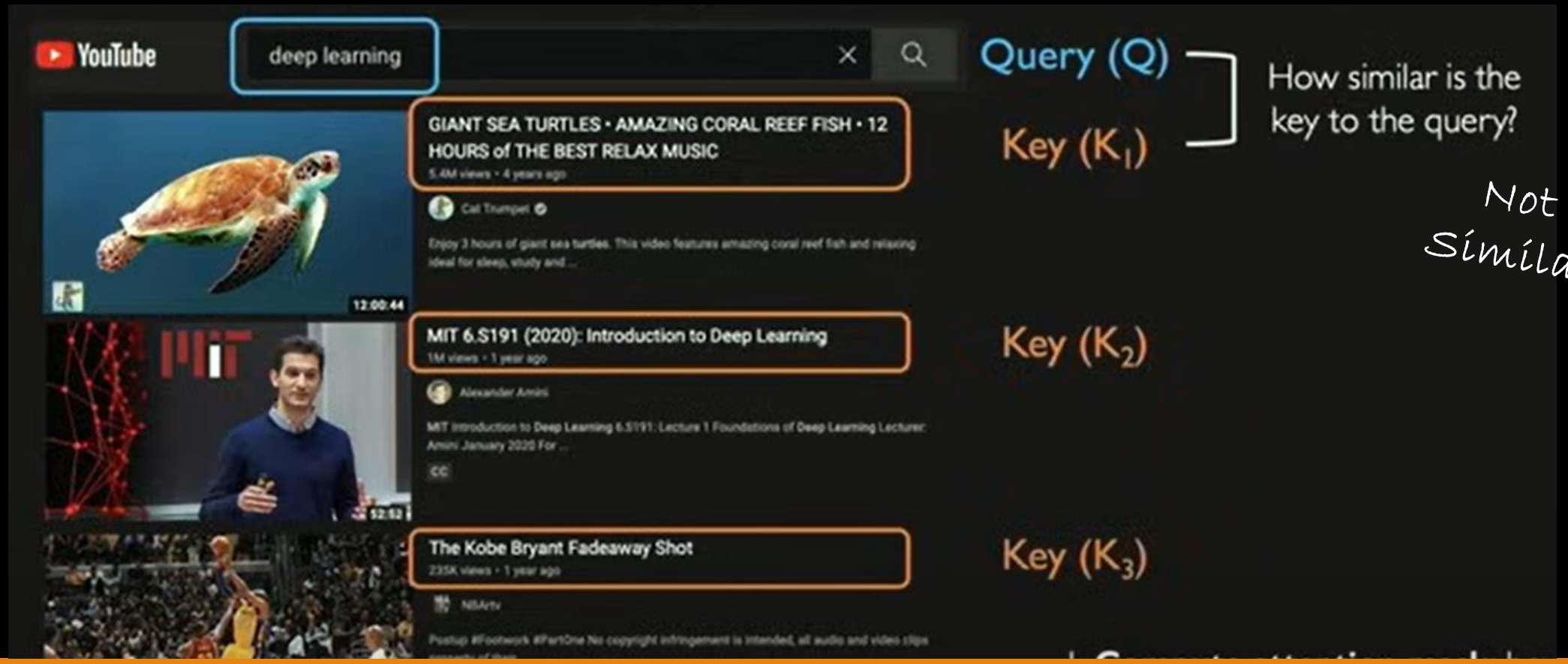
Attention Mechanism (Queries, Keys, Values)

Next is to Take my query and check for Keys with closest meaning,



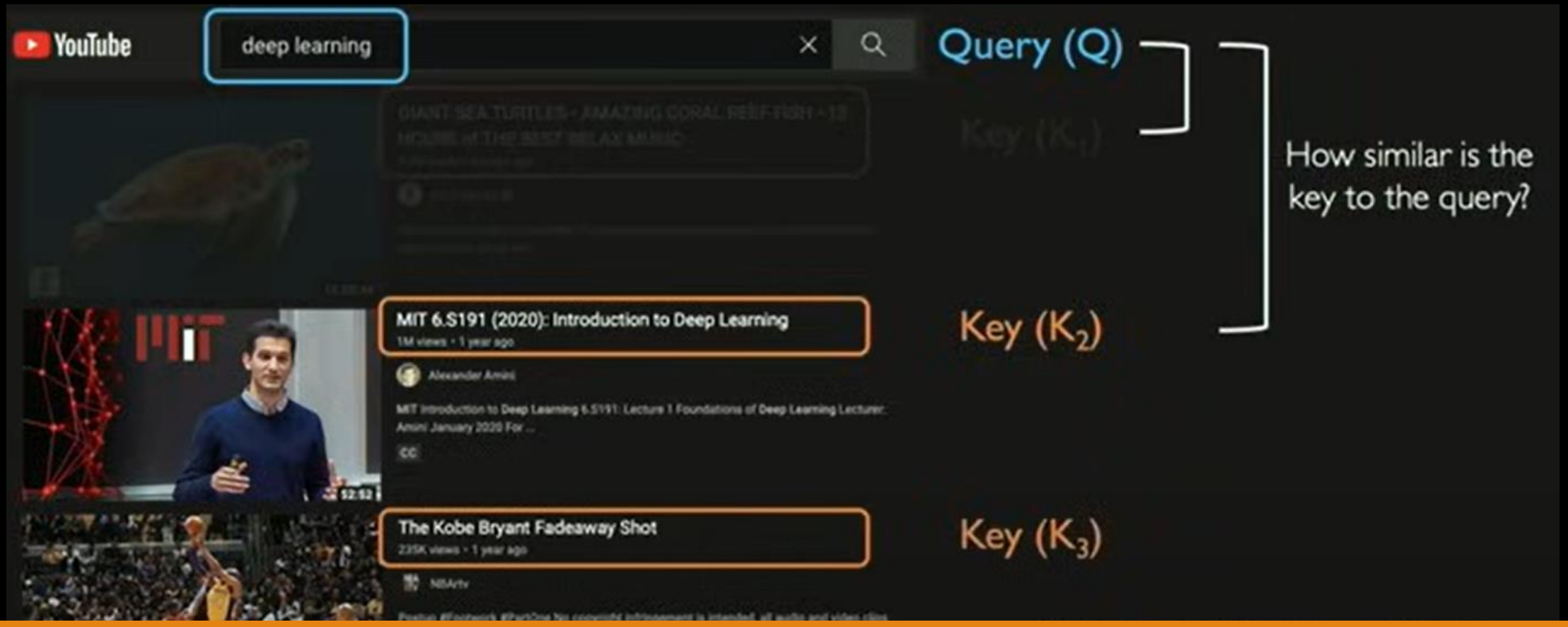
Attention Mechanism (Queries, Keys, Values)

Next is to Take my query and check for Keys with closest meaning,



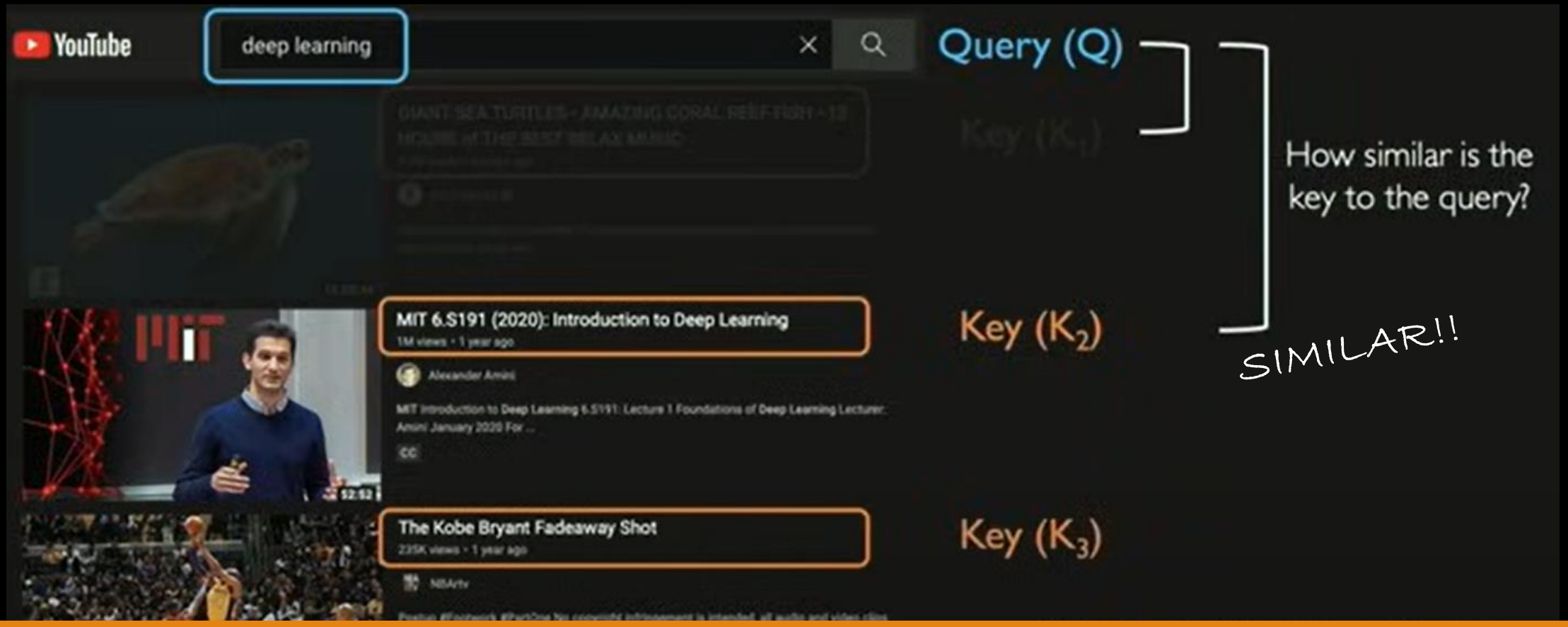
Attention Mechanism (Queries, Keys, Values)

Next is to Take my query and check for Keys with closest meaning,



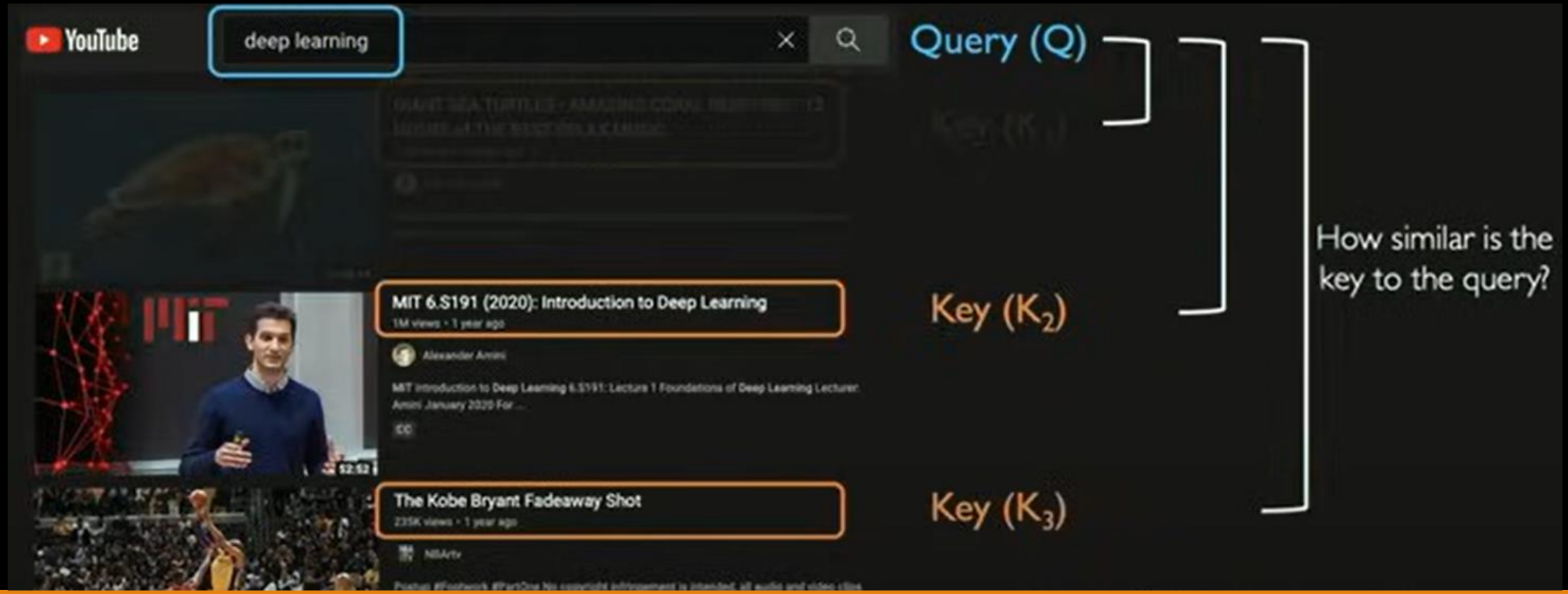
Attention Mechanism (Queries, Keys, Values)

Next is to Take my query and check for Keys with closest meaning,



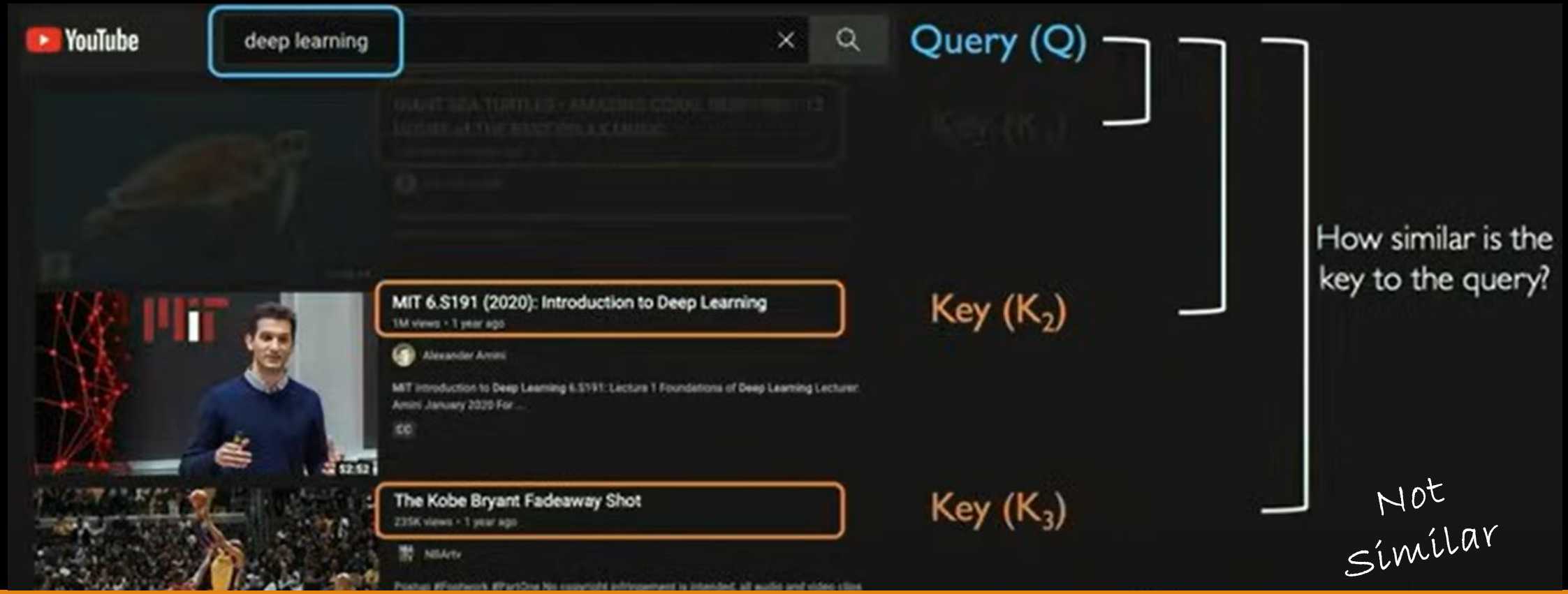
Attention Mechanism (Queries, Keys, Values)

Next is to Take my query and check for Keys with closest meaning,



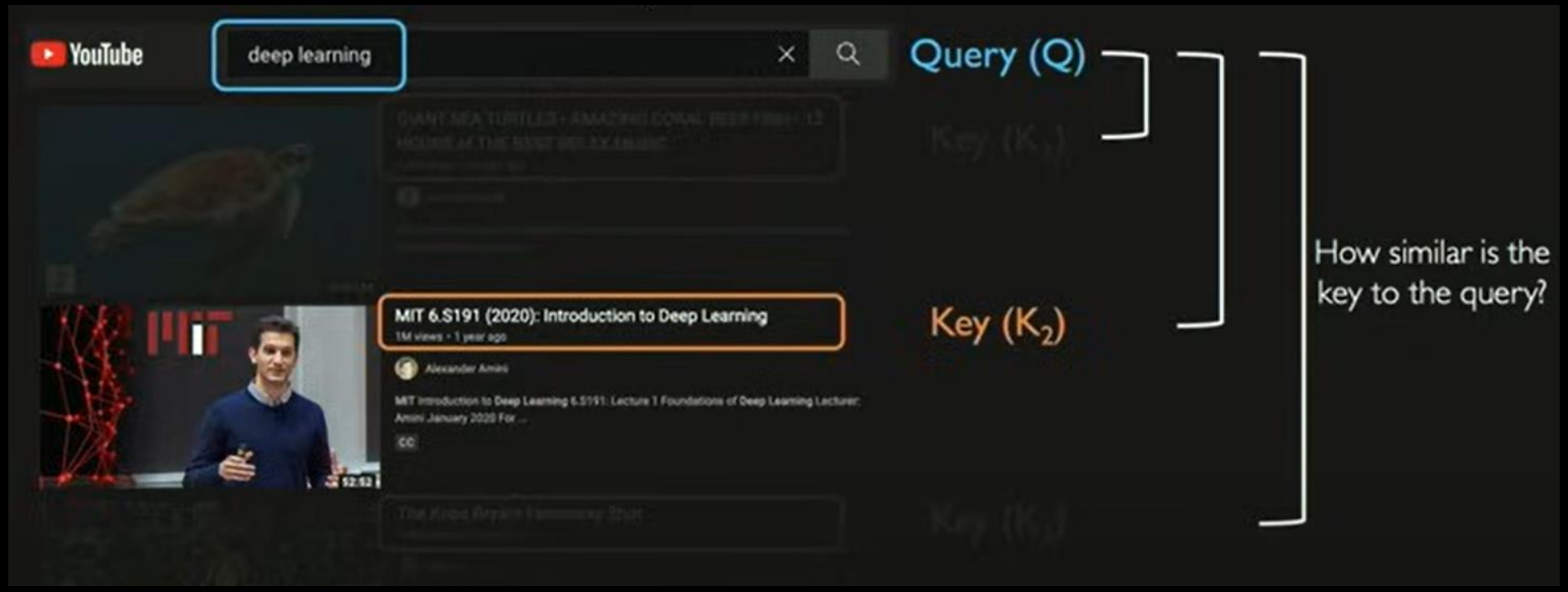
Attention Mechanism (Queries, Keys, Values)

Next is to Take my query and check for Keys with closest meaning,



Attention Mechanism (Queries, Keys, Values)

The check of how similar is each key to the desired Query is a process called "**Computing Attention Mask**"



Attention Mechanism (Queries, Keys, Values)

After Explaining **Queries and Keys**, how about Value?? Let's move on.....

Value, basically the key with the **highest Attention Number**

Attention Mechanism (Queries, Keys, Values)

Attention mechanisms helps to find the right information, using Queries, Keys, Values

Let's use another analogy: **Searching for a book in a library**

Query (Question): What you're looking for **(e.g., 'books about animals').**

Keys (Labels): The labels on the books (e.g., titles, keywords like 'science', 'animals', 'physics' etc).

Values (Information): The actual books (for this example, books about animals).

The computer compares your 'Query' with the 'Keys' to find the most relevant 'Values'.

What's Next after Attention Mechanism 😀😀

Let's briefly discuss another important layer and component of a Model Architecture

Encoder
and
Decoder

What's Encoder-Decoder

The Encoder-Decoder architecture is a neural network design pattern that's particularly effective for sequence-to-sequence tasks. It's used in various applications, including machine translation, text summarization, and question answering.

Models that used **Encoder** alone are tagged **Encode-Only Model, eg BERT**

Models that used **Decoder** alone are tagged **Decode-Only Model, eg GPT**

Models that use both an **Encoder** and a **Decoder** are tagged **Encoder-Decoder Models, e.g: T5 by Google, BART by Facebook AI**

Encoder-Decoder

The Encoder-Decoder architecture though, isn't exclusive to Transformers. While Transformers popularized it and significantly improved its performance, the general concept has been used with RNNs and, to a lesser extent, with CNNs as well.

A typical **Seq2Seq model** consists of an RNN encoder and an RNN decoder.

But: Transformers have significantly improved the performance of Encoder-Decoder models, particularly for NLP tasks.

Encoder

The Encoder: Helps to understand the Input. The encoder's job is to take an input sequence of work embeddings (e.g., the token of an English sentence) processes these embeddings, using **self-attention** and **feed-forward networks**, to create contextualized representations of the input sequence.

i.e The encoder then takes those numerical representations and builds a deeper understanding of the relationships between the tokens in the context of the entire input sequence.

Encoder

E.g.:

Example:

Let's say you have the sentence: "The cat sat on the mat."

1. **Tokenization:**
   - The sentence might be tokenized as: `["the", "cat", "sat", "on", "the", "mat"]`
2. **Word Embeddings:**
   - Each of these tokens would be converted into a vector. For example, "cat" might become `[0.2, -0.5, 0.8, ...]`.
3. **Encoder:**
   - The sequence of these vectors `[[0.2, -0.5, 0.8, ...], [...], [...], ...]` is then fed into the Transformer encoder.
   - The encoder's self-attention mechanism would allow each word to attend to all the other words in the sentence.
   - This would allow the model to understand that "the" refers to "cat" and "mat" in this context.

Encoder-Only Models

- **BERT (Bidirectional Encoder Representations from Transformers):** Its bidirectional training and **masked language modeling** focus on deep contextual understanding.
- **RoBERTa (Robustly Optimized BERT Pretraining Approach):** An optimized variant of BERT, with improved training procedures and larger datasets.
- **ALBERT (A Lite BERT for Self-supervised Learning of Language Representations):** A parameter-efficient version of BERT, using parameter sharing and sentence-order prediction.
- **DistilBERT:** A distilled version of BERT, that is much smaller and faster, but retains much of the performance.

Decoder

The Decoder: The decoder's primary function is to generate output sequences, such as translated sentences, summaries, or chatbot responses. It does this step-by-step, taking into account the encoded representation of the input and the previously generated tokens.

The **decoder** receives the encoded representations from the encoder (in encoder-decoder architectures) or the context from previous decoder layers (in decoder-only architectures like GPT)

Decoder-Only Models

- **GPT (Generative Pre-trained Transformer) Family (GPT, GPT-2, GPT-3, GPT-4):** These models are the prime examples of decoder-only architectures. Their autoregressive nature and causal language modeling are geared towards text generation.
- **CTRL (Conditional Transformer Language Model):** Allows for fine-grained control over text generation through control codes.
- **Transformer-XL (Transformer Extra Long):** An enhanced version of the transformer decoder that is designed to handle much longer context windows.

Comparison

Encoder-Only (e.g BERT) vs. Decoder-Only (e.g GPT)

Feature	BERT (Encoder-Only)	GPT (Decoder-Only)
Input Processing	Bidirectional (sees entire sequence at once)	Unidirectional (sees previous tokens only)
Attention Mechanism	Self-attention (all tokens attend to all others)	Masked self-attention (tokens attend to previous tokens)
Primary Task	Understanding input context	Generating output text
Example Task	Question answering, sentiment analysis	Text generation, chatbots
How it Works	Processes the entire input sequence to create contextualized representations.	Generates text token by token, conditioned on previous tokens.
Use of Embeddings	Embeddings are used to represent the input for understanding by the encoder.	Embeddings are used to represent the input prompt and the generated tokens for generation by the decoder.
Output	Contextualized representations of the input.	Generated sequence of tokens (text).
Typical use case	To classify or understand the input.	To create new text.

Can a model? 😃 😃

Let's attend to this

Can a model have both Encode and Decoder Architecture?

Can a model? 😃 😃

Let's attend to this

They are called Encode-Decoder Models

Encoder-Decoder Transfer Learning Models:

- **T5 (Text-to-Text Transfer Transformer):** Trained to treat all NLP tasks as text-to-text problems. Available on Hugging Face: t5-small, t5-base, t5-large, etc. It's Highly versatile for various NLP tasks.

  T5 Model links:

  - t5-small: https://huggingface.co/t5-small
  - t5-base: https://huggingface.co/t5-base
  - t5-large: https://huggingface.co/t5-large

Encoder-Decoder Transfer Learning Models:

- **BART (Bidirectional and Auto-Regressive Transformers):** Excellent for sequence-to-sequence tasks like summarization and translation. Available on Hugging Face: facebook/bart-base, facebook/bart-large, etc.

BART Models link:

- facebook/bart-base: https://huggingface.co/facebook/bart-base
- facebook/bart-large: https://huggingface.co/facebook/bart-large

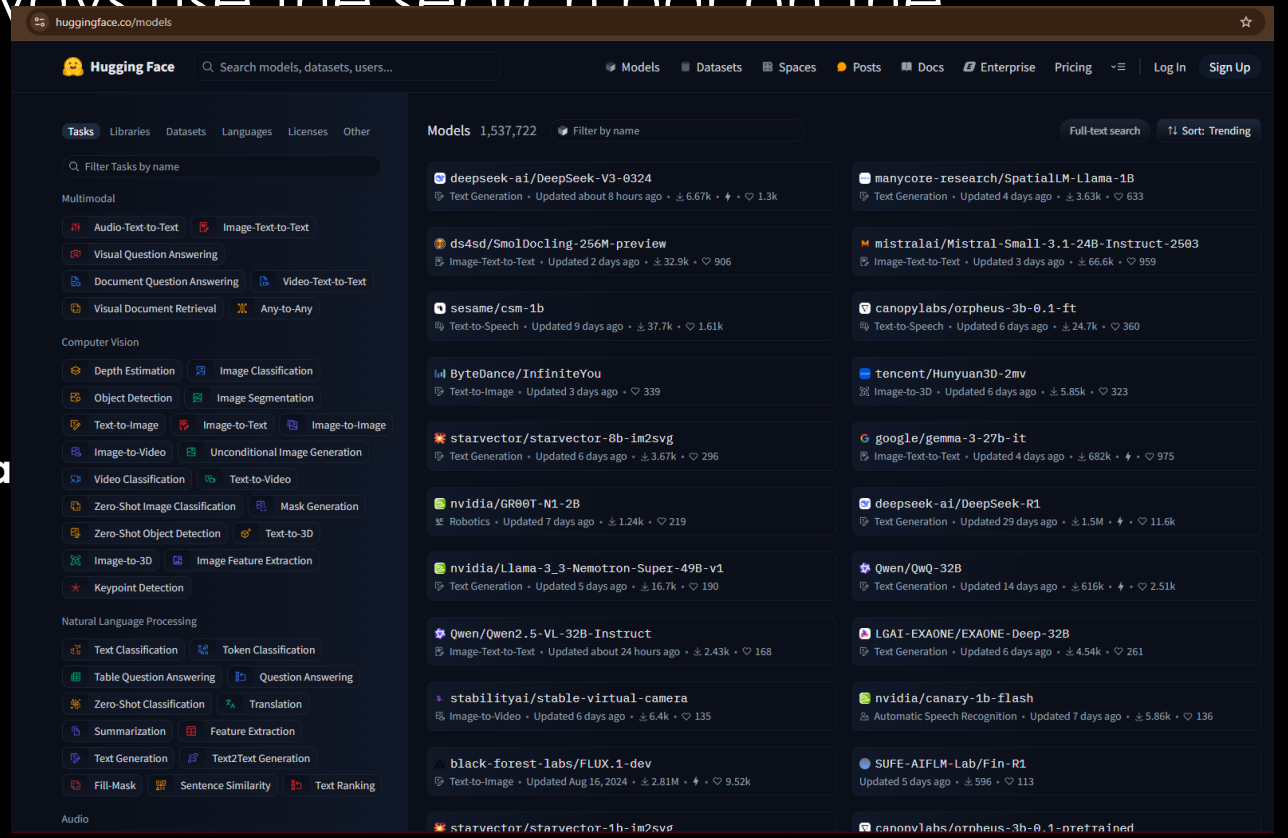
Encoder-Decoder Transfer Learning Models:

- **MarianMT:** Specifically designed for machine translation. Available on Hugging Face: Helsinki-NLP/opus-mt-*. A large number of language pairs are available. I can find all MarianMT models by searching "Helsinki-NLP/opus-mt" on Hugging Face. https://huggingface.co/Helsinki-NLP
- **Pegasus:** Designed for abstractive summarization. Available on hugging face: google/pegasus-*. https://huggingface.co/google/pegasus-large
- **Nb: the '*' means that there are many models that follow the pattern: e.g Helsinki-NLP/opus-mt-en-fr, Helsinki-NLP/opus-mt-fr-en, etc.**

Hugging Face Hub Search:

To find other models, you can always use the search bar on the Hugging Face Hub:

https://huggingface.co/models.



NB: When using the Hugging Face hub, it is always a good idea to read the model card that is attached to each model, as it contains important information about the model, such as intended usage, limitations, and ethical considerations.

Evaluating models with metrics

It is impossible to compare one transformer model to another transformer model (or any other NLP model) without a universal measurement system that uses metrics.

1. Accuracy score
2. F1-score
3. Matthews Correlation Coefficient (MCC)

Accuracy score

The basic function is:

$$Accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

The accuracy score, in whatever variant you use, is a practical evaluation. The score function calculates a straightforward true or false value for each result. Either the model's outputs, yyy, match the correct predictions, yy, for a given subset, samples$_i$, of a set of samples or not.

We will obtain 1 if the result for the subset is correct and 0 if it is false.

Let's now examine the more flexible F1-score.

F1-score

The F1-score introduces a more flexible approach that can help when faced with datasets containing uneven class distributions.

The F1-score uses the weighted values of precision and recall. It is a weighted average of precision and recall values:

F1score= 2* (precision * recall)/(precision + recall)

The equation is:

$$F1\ score = 2 \times \frac{P \times R}{P + R}$$

$$P = \frac{T_p}{T_p + F_p}$$

$$R = \frac{T_p}{T_p + F_n}$$

In its equation, true *(T)* positives *(p)*, false *(F)* positives *(p)*, and false *(F)* negatives *(n)* are plugged into the precision *(P)* and recall *(R)* equations

Matthews Correlation Coefficient (MCC)

The Matthews Correlation Coefficient (MCC) was initially designed to measure the quality of binary classifications and can be modified to be a multi-class correlation coefficient.

A two-class classification can be made with four probabilities at each prediction:

• TP = True Positive, • TN = True Negative, • FP = False Positive, • FN = False Negative

Brian W. Matthews, a biochemist, designed it in 1975, inspired by his predecessors' phi function. Since then, it has evolved into various formats such as the following one:

$$MCC = \frac{TP\ x\ TN - FP\ x\ FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The value produced by MCC is between -1 and +1. +1 is the maximum positive value of a prediction. -1 is an inverse prediction. 0 is an average random prediction.

Matthews Correlation Coefficient (MCC)

$$MCC = \frac{TP\ x\ TN - FP\ x\ FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The value produced by MCC is between -1 and +1. +1 is the maximum positive value of a prediction. -1 is an inverse prediction. 0 is an average random prediction.

To Use, MCC is imported from sklearn.metrics:

```
#@title Evaluating Using Matthew's Correlation Coefficient
# Import and evaluate each test batch using Matthew's correlation
coefficient
from sklearn.metrics import matthews_corrcoef
```

Matthews Correlation Coefficient (MCC)

MCC computes a measurement with true positives

(TP), true negatives (TN), false positives (FP), and false negatives (FN).

The MCC can be summarized by the followin

The equation is:

$$F1\ score = 2 \times \frac{P \times R}{P + R}$$

$$P = \frac{T_p}{T_p + F_p}$$

$$R = \frac{T_p}{T_p + F_n}$$

Links to materials:

https://jalammar.github.io/illustrated-transformer/

Hands-On Large Language Models: Language Understanding and Generation: https://www.amazon.com/Hands-Large-Language-Models-Understanding/dp/1098150961

https://www.deeplearning.ai/short-courses/how-transformer-llms-work/?utm_campaign=handsonllm-launch&utm_medium=partner

QUESTIONS AND ANSWERS

what's next??

PRACTICE AND ATTEND OTHER SESSIONS ON

Model Development and Deployment

Attribution & Acknowledgment

While diligent effort has been made to attribute sources, some content and images within this presentation may be the property of others. This presentation is for educational use and does not claim ownership of all materials.

i.e.:

A few things in here might be someone else's. I am ONLY using them to teach, and I respect the original creators

The END Day 4

DSN LEKKI-AJAH
BY ABEREJO HABEEBLAH O.
X: @HABEREJO

Next Class should be on LLM and it's advanced techniques and Model Deployment

Takeaway:

Next page!

A Glossary of Essential AI/ML/DL/NLP Terminologies

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

General AI Terms are:

1. **Artificial Intelligence (AI):** The broad concept of creating machines that can perform tasks that usually require human intelligence. **i.e.**: Making computers smart like us. **Scenario**: Imagine a robot that can cook dinner, do laundry, and even hold a conversation.

2. **Agent:** An entity (software or hardware) that perceives its environment and acts to achieve a goal **i.e.**: Something that can see what's around it and make decisions to do something. **Scenario**: Think of a video game character. It sees the game world (environment) and you control it to jump, shoot, etc. (actions) to win the game (goal).

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

3. **Machine Learning (ML):** A subset of AI where machines learn from data without being explicitly programmed. **i.e.**: Teaching a computer to learn by showing it examples, like training a dog with treats. **Scenario**: A music app recommending songs you might like. It learns your taste based on what you've listened to before.

4. **Deep Learning (DL):** A subfield of ML using artificial neural networks with many layers to learn complex patterns. **i.e.**: A really complex way of teaching a computer, like how our brains learn with lots of interconnected parts. **Scenario**: Image recognition software that can tell the difference between a cat and a dog, even if they look similar.

5. **Natural Language Processing (NLP):** A branch of AI focused on enabling computers to understand, interpret, and generate human language. **i.e.**: Teaching computers to understand and talk like us. **Scenario**: Google Translate, which can translate text from one language to another.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

Machine Learning Terms:

1. **Supervised Learning:** Learning from labeled data (input-output pairs). **i.e.**: Teaching a computer with answer keys. **Scenario**: Training a program to predict house prices. You give it examples of houses with their sizes, locations, and prices (the "answer key").

2. **Unsupervised Learning:** Learning from unlabeled data, finding patterns. **i.e.**: Giving a computer a bunch of stuff and asking it to sort it into groups. **Scenario**: A website suggesting groups you might like to join based on your interests.

3. **Reinforcement Learning:** An agent learns by interacting with an environment and getting rewards or penalties. **i.e.**: Training a pet with positive and negative reinforcement. **Scenario**: An AI playing a video game. It gets points for winning and loses points for losing, learning to play better over time.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

4. **Training Data:** The data used to train a model. **i.e.**: The examples you show the computer to learn from. **Scenario**: All the photos of cats and dogs you use to train the image recognition software.

5. **Test Data:** Data used to check how well the trained model performs on new data. **i.e.**: Showing the computer pictures, it hasn't seen before to see if it learned correctly. **Scenario**: Using a different set of cat and dog photos to test the image recognition software.

6. **Feature:** A measurable characteristic. **i.e.**: A detail about something. **Scenario**: In predicting house prices, features are size, location, number of rooms, etc.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

7. **Model:** A mathematical representation of learned patterns. **i.e.**: The rules the computer learns. **Scenario**: The formula the computer uses to predict house prices.

8. **Algorithm:** The specific set of rules or instructions used for learning. Simple: The method the computer uses to learn. Scenario: Different ways of training a computer, like different study techniques.

9. **Overfitting:** A model that learns the training data too well (including noise) and performs poorly on test data. **i.e.**: Memorizing the answers to practice questions but failing the real test. **Scenario**: A model that can perfectly predict prices of houses it has seen, but it can't predict prices of houses it has not seen.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

10. **Underfitting:** A model that is too simple to capture the patterns and performs poorly on both training and test data. **i.e.**: Not studying enough and failing the test. Scenario: A model that is too simple to predict house prices.

11. **Accuracy:** How often the model makes correct predictions. **i.e.**: How often the computer gets the answer right.

12. **Precision:** Out of all positive predictions, how many were actually positive? **i.e.**: Of all the things the computer said were cats, how many were actually cats?

13. **Recall:** Out of all actually positive cases, how many did the model correctly identify? **i.e.**: Of all the actual cats, how many did the computer correctly identify?

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

14. **Bias**: Systematic error in the model due to flawed assumptions. **i.e.**: A prejudice in the computer's thinking. Scenario: A model trained only on pictures of sunny days will be biased and might not recognize cloudy days.

15. **Variance**: Sensitivity of the model to fluctuations in the training data. **i.e.**: The computer getting confused by small changes in the examples. Scenario: A model that works perfectly on one set of data but fails on another very similar set of data.

16. **Hyperparameters**: Parameters set before training that control the learning process. **i.e.**: Settings you adjust before you start training. **Scenario**: Like adjusting the focus and brightness on a camera before taking a picture.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

Deep Learning Terms:

1. **Neural Network:** Interconnected layers of neurons that process information. **i.e.**: A network of connected "brains" working together.
2. **Neuron (Node):** A basic processing unit in a neural network. **i.e.**: A single "brain" in the network.
3. **Layer:** A group of neurons (input, hidden, output layers). **i.e.**: A level in the network.
4. **Activation Function:** Introduces non-linearity to neurons. **i.e.**: A way for the "brains" to make more complex decisions.
5. **Backpropagation:** Adjusting connection weights to minimize prediction errors. **i.e.**: A way for the network to learn from its mistakes.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

6. **Epoch:** One complete pass through the entire training dataset. **i.e.**: Showing the computer all the examples once.
7. **Batch Size:** Number of training examples processed at once. **i.e.**: How many examples the computer looks at a time.
8. **Loss Function:** Measures the model's prediction error. **i.e.**: How the computer knows it made a mistake.
9. **Weight**: The strength of the connection between two neurons. **i.e.**: How strongly two "brains" influence each other.
10. **Learning Rate:** How quickly the model updates its weights. **i.e.**: How quickly the computer learns.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

Natural Language Processing (NLP) Terms:

1. **Tokenization:** Breaking down text into tokens (words, phrases). **i.e.**: Splitting a sentence into words.
2. **Stop Words:** Common words (e.g., "the," "a," "is") often removed. **i.e.**: Words that don't carry much meaning.
3. **Stemming:** Reducing words to their root form. **i.e.**: Chopping off the endings of words.
4. **Lemmatization:** Reducing words to their dictionary form (lemma). **i.e.**: Finding the base word.
6. **Corpus:** A large collection of text. **i.e.**: A big library of text.

A Glossary of Essential AI/ML/DL/NLP Terminologies

This is a PLUS +

6. **Word Embeddings:** Representing words as vectors capturing semantic relationships. **i.e.**: Giving each word a set of coordinates in "meaning-space".
7. **Sentiment Analysis:** Determining the emotional tone of text. **i.e.**: Figuring out if someone is happy or sad based on what they wrote.
8. **Machine Translation:** Translating text from one language to another.
9. **Text Summarization:** Generating a concise summary of text.
10. **Named Entity Recognition (NER):** Identifying and classifying named entities. **i.e.**: Finding and labeling names of people, places, and organizations.

The END Day 4

DSN LEKKI-AJAH
BY ABEREJO HABEEBLAH O.
X: @HABEREJO

Next Class should be on LLM and it's advanced techniques and Model Deployment